5. Kapitel

Computer-Simulation

Hans Ueckert

1. Einleitung

In der Geschichte der Psychologie gab es - in einer etwas groben Rasterung - drei chronologische Einschnitte, die die methodologische Entwicklung der Psychologie zu einer selbständigen Einzelwissenschaft kennzeichnen. Der erste war die Einführung des Experiments in die psychologische Forschung durch Wilhelm Wundt vor etwa 100 Jahren, der zweite die Entwicklung der quantitativen Methodik für die Auswertung psychologischer Daten in der ersten Hälfte unseres Jahrhunderts und der dritte die - wenn man so will - ,,Erfindung" der Computer-Simulation zur Nachbildung psychischer Vorgänge auf einer elektronischen Rechenanlage vor rund 20 Jahren. Mit jeder dieser methodologischen Entwicklungslinien sind wissenschaftshistorische Begleiterscheinungen verbunden, die als spezifische Herausforderungen des Selbstverständnisses der Psychologie verstanden werden können. Die Experimentalmethodik wurde aus den Naturwissenschaften übernommen und mußte erst dem psychologischen Gegenstandsbereich angepaßt werden; die quantitative Methodik führte in ihrer konsequentesten Weiterentwicklung zu einer mathematischen Psychologie, die letztlich eine wie auch immer zu bewertende ,,Mathematisierung" des Menschen - als dem Hauptgegenstand der Psychologie - bedingt; die Computer-Simulation schließlich ist von der Entwicklung einer ,,künstlichen Intelligenz" begleitet, die die Maschine nicht nur - wie bisher - auf dem Gebiet materieller Tätigkeiten in Konkurrenz zum Menschen stellt, sondern - zuvor noch ganz unvorstellbar - gerade auch auf dem Gebiet informationeller (oder ,,geistiger") Tätigkeiten.

Trotz dieser zunehmenden, durch die empirische Methodenentwicklung bedingten ,,Entzauberung" psychischen Geschehens scheint die ,,technischste" aller Methoden, die Computer-Simulation, dem Untersuchungsgegenstand der Psychologie näher kommen zu können als die vorangehenden Entwicklungen. Das Experiment ist zwar der primäre Datenlieferant über psychologische Sachverhalte, bleibt aber aufgrund der methodologisch begründeten Künst-

lichkeit seiner Untersuchungsbedingungen oft von jeglicher Alltagswirklichkeit weit entfernt; die Quantifizierbarkeit psychischer Gegebenheiten ist zwar legitimes Interesse wissenschaftlicher Forschung, verfehlt aber ihren Gegenstand so lange, als die ,,Meßbarkeiten“ des Psychischen ungeklärt oder nicht nachweisbar sind. Die Computer-Simulation dagegen ist eine Methode, die den Untersuchungsgegenstand nicht nur in beliebig fein abstufbarer Weise beschreibbar macht, sondern die die untersuchten Phänomene mit dem Instrument des Rechners vollständig nachzubilden und damit tatsächlich sogar zu reproduzieren gestattet. Dabei spielt es im Grunde keine Rolle, ob die untersuchten psychischen Gegebenheiten quantifizierbar sind oder nicht - sowohl numerisch als auch nicht-numerisch gegebene Daten sind in ihren Wirkungsbedingungen auf dem Rechner reproduzierbar. Es spielt im Prinzip auch keine Rolle, ob die auf einem Rechner erstellte Nachbildung - oder am Beispiel der ,,künstlichen Intelligenz“ auch Erzeugung - psychischer Vorgänge auf experimentalpsychologisch erhobenen Daten beruht oder aber auf der Intuition oder der Introspektion des Modellkonstrukteurs (wie beispielsweise für die Entwicklung von Systemen der ,,künstlichen Intelligenz“ kennzeichnend); beide Arten der Modellbildung sind, wenn auch mit unterschiedlicher ,,Abbildungstreue“ hinsichtlich ihres Gegenstandes, gleichermaßen durchführbar.

Allgemein betrachtet kann man die Computer-Simulation als eine psychologische Methode definieren, die die Modellierung psychischer Gegebenheiten und/oder Vorgänge auf einem Rechner derart zu realisieren gestattet, daß nicht nur eine beliebig feine Abbildung des Untersuchungsgegenstandes ermöglicht wird, sondern daß dieser darüber hinaus auch in all seinen interessierenden Merkmalen, Eigenschaften und Funktionszusammenhängen nachgebildet werden kann: Das resultierende Modell beschreibt und erklärt nicht nur den gewählten Wirklichkeitsausschnitt, sondern es reproduziert ihn auch in der von der zugrunde gelegten psychologischen Theorie vorhersagbaren Art und Weise.

Mit der Entwicklung der Methode der Computer-Simulation von Anfang an eng verknüpft war eine grundlegend neue psychologische Theorienbildung, deren sichtbare Ausprägung heute die kognitive Psychologie ist. In ihrem Gefolge wurde der Begriff der Information als Grundkategorie psychischen Geschens eingeführt: Psychisches Geschehen ist nicht das Wahrnehmungsgefüge substanzloser ,,seelischer“ oder ,,geistiger“ Vorgänge (wie es die geisteswissenschaftliche Psychologie sehen möchte), ist nicht am Verhalten ablesbares materielles Tun (wie es beispielsweise der Behaviorismus beschreibt), sondern ist ein trotz aller Komplexität aufschlüsselbarer Prozeß der Informationsverarbeitung, deren substanzielle Grundlage anatomisch und physiologisch nachweisbare Strukturen und Funktionen des Nerven- und Sinnessystems sind. Jedoch nicht Neuroanatomie und -physiologie sind der Gegenstandsbereich der kognitiven Psychologie, sondern die ,,höheren“ Prozesse der Auf-

nahme, Verarbeitung (einschließlich Erzeugung) und Ausgabe von Information über diesem materiellen Fundament. Daß einerseits eine mit nicht-physikalischen Eigengesetzlichkeiten operierende Informationsverarbeitung möglich ist, andererseits diese jedoch ohne geeignete materielle Trägersysteme nicht existieren kann, hätte der Mensch in seiner mehrtausendjährigen Beschäftigung mit sich selbst längst erkennen können; offensichtlich bedurfte es dazu erst der Entwicklung des Computers, dessen Fähigkeit zur Informationsverarbeitung geradezu als Existenzbeweis dafür angesehen werden kann, daß nicht-materielle, ,,geistige“ oder - wie präziser zu sagen ist - ,,informationelle“ Prozesse mit ihren Eigengesetzlichkeiten in materiellen Systemen realisierbar sind.

Daß dabei der Computer eigentlich gar kein Rechner ist, der mit irgendwelchen numerischen Werten (,,Zahlen“) operiert, kein System also, das im wörtlichen Sinne ,,rechnet“, sondern ein System, das Information beliebiger Art verarbeitet (sei diese Information als Zahlen, Buchstaben, Wörter oder was auch immer interpretierbar), diese Erkenntnis bedurfte erst einer intellektuellen Anstrengung zur Beseitigung eines durch die Computerentwicklung und der anfänglichen Rechnerverwendung bedingten Vorurteils. Der Computer ist nichts anderes als eine Maschine, die aufgrund ihrer jeweiligen Programmierung Zeichen in für sie ,,lesbarer“ Form verarbeitet, gleichgültig, was immer diese Zeichen ,,bezeichnen“ mögen.

Wenn die Denkpsychologie beispielsweise den Prozeß des menschlichen Problemlösens als einen Prozeß der sukzessiven Verarbeitung von Information beschreiben und erklären kann, dann ist es ein naheliegender Schritt, diesen Prozeß auch auf einer Maschine nachzubilden, die zur Informationsverarbeitung aufgrund ihrer Konstruktionsmerkmale und ihrer diesbezüglichen Programmierbarkeit fähig ist. Genau dieser Schritt stand auch am Anfang der Entwicklung der Computer-Simulation als psychologische Methode, wie die ersten Arbeiten von Simon, dem eigentlichen Begründer der modernen kognitiven Psychologie, und seinen Mitarbeitern zeigen (vgl. beispielsweise Newell, Shaw & Simon, 1958).

In ihrer weiteren Entwicklung hat die Computer-Simulation - und in ihrem Gefolge auch die ,,künstliche Intelligenz“ - eine Vielzahl von Modellen psychischer Vorgänge geliefert, die in ihrer Gesamtheit einerseits eine Herausforderung an den Menschen bezüglich seiner ,,intellektuellen Einzigartigkeit“ darstellen, andererseits aber ohne eine genauere Kenntnis der Grundlagen, Techniken und Anwendungsmöglichkeiten der Computer-Simulation als wissenschaftliche Methode nicht angemessen beurteilt werden können.

2. *Das Paradigma der Computer-Simulation in der Psychologie*

Der vielfaltige Gebrauch der Computer-Simulation in den verschiedensten Wissenschafts- und Anwendungsbereichen hat zur Folge, daß man nicht von dem Paradigma der Computer-Simulation sprechen kann, sondern nur von unterschiedlichen Paradigmen entsprechend ihrer Verwendungsweisen in den einzelnen Gebieten.

2.1 Zur Klassifikation von Simulationsmodellen

Eine systematische Klassifikation von Simulationsmodellen sollte sowohl von formalen als auch von inhaltlichen Kriterien ausgehen können. Bisher vorgelegte Klassifikationen betonen entweder den einen oder den anderen Ausgangspunkt und variieren daher von Autor zu Autor.

Nach *formalen* methodologischen Kriterien unterscheidet beispielsweise Harbordt (1974, S. 22-29)

(1) statische und dynamische,
(2) deterministische und indeterministische,
(3) quantitative und qualitative,
(4) analytische und synthetische
(5) Erkundungs- und Entscheidungsmodelle.

(1) *Statische* Modelle sind in ihren Abbildungseigenschaften auf einen Zeitpunkt bezogen, während der Zeitablauf über mehreren Zeitpunkten *dynamische* Modelle kennzeichnet. Beispielsweise ist die funktionale Verknüpfung f der Variablen X_1, X_2 und Y zum Zeitpunkt t mit der Gleichung

$$Y(t) = f[X_1(t),X_2(t)]$$

ein statisches Modell, bei dem die Zeitvariable auch weggelassen werden kann, während die Gleichung

$$Y(t) = f[X_1(t),X_2(t-1)]$$

ein dynamisches Modell bezeichnet, bei dem der Wert der Variable Y zum Zeitpunkt t von der funktionalen Beziehung zwischen den Werten der Variable X_1 zum gleichen Zeitpunkt t und der Variable X_2 zum vorangehenden Zeitpunkt t-1 abhängt. In aller Regel sind Simulationsmodelle als derartige dynamische Modelle konzipiert.

(2) *Deterministische* Modelle sind in ihrem Eingabe-Ausgabe-Verhalten eindeutig bestimmt, d.h. bei bestimmten Eingabewerten einer Variable X sind die Ausgabewerte der Variable Y exakt vorhersagbar. Bei *indeterministischen* Mo-

dellen ist diese Vorhersagbarkeit nicht mehr eindeutig gegeben. Nach Harbordt (1974, S. 23-25) kann man hier noch zwischen stochastischen und probabilistischen Modellen unterscheiden. *Stochastische* Modelle enthalten eine oder mehrere Zufallsvariable, deren Werte nur mit bestimmten Auftritts-Wahrscheinlichkeiten vorhersagbar sind, wie z.B. in der Gleichung

$$Y = f(X,Z),$$

in der das Ausgabeverhalten der Variable Y von der funktionalen Beziehung zwischen der (deterministischen) Variable X und der (stochastischen) Zufallsvariable Z abhängt.

Probabilistische Modelle sind demgegenüber dadurch gekennzeichnet, daß die Ausgabevariable Y selbst eine Zufallsvariable ist, deren Wahrscheinlichkeitsverteilung P(Y) aufgrund ihrer funktionalen Beziehung zu deterministischen Variablen X_i bekannt ist. Solche Modelle werden auch als ,,Monte-Carlo-Simulationen" bezeichnet, da die tatsächliche Modellausgabe Y aufgrund eines gleichverteilten Zufallsgenerators unter Berücksichtigung des funktionalen Zusammenhangs der Eingabevariablen X_i erzeugt wird. - In der Mehrzahl der Fälle sind Simulationsmodelle in der Psychologie deterministische Modelle, ggf. um gewisse stochastische oder probabilistische Komponenten ergänzt.

(3) *Quantitative* Modelle verwenden in ihren Abbildungseigenschaften, insbesondere für das Ausgabeverhalten, numerische Variablen (meist mit mindestens Intervallskalenniveau), während *qualitative* Modelle die funktionalen Zusammenhänge zwischen nicht-numerischen Variablen nachbilden (numerisch als Nominal- und/oder Ordinalskalenniveau beschreibbar). Wie im weiteren gezeigt werden soll, bedient sich die Simulationsmethode in der Psychologie in bevorzugter Weise der qualitativen, nicht-numerischen Modellierung ihrer Untersuchungsgegenstände.

(4) Die Unterscheidung zwischen analytischen und synthetischen Modellen der Computer-Simulation ist - wie auch die folgende zwischen Erkundungs- und Entscheidungsmodellen - in gewisser Weise willkürlich. Der Unterschied liegt eher in den Vorgehensweisen des Modellkonstrukteurs als in den Modelleigenschaften selbst.

Analytische Modelle gehen von dem beobachtbaren Gesamtverhalten des zu modellierenden Systems aus und versuchen dieses auf das Zusammenwirken seiner mehr oder minder gut beobachtbaren Komponenten zurückzuführen. *Synthetische* Modelle können dagegen von einer ausreichenden Kenntnis solcher Komponenten ausgehen und das aus diesen zusammensetzbare Gesamtverhalten zu reproduzieren versuchen. - Charakteristisch für die psychologische Simulationsforschung dürfte aufgrund des noch bescheidenen Wissensstandes der Psychologie die analytische Vorgehensweise sein, auch wenn die

synthetische Modellierbarkeit als erstrebenswertes wissenschaftliches Ziel anzusehen sein mag.

(5) Praktisch gesehen fällt die Unterscheidung zwischen Erkundungs- und Entscheidungsmodellen mit der zwischen analytischen und synthetischen Simulationsmodellen zusammen.

Erkundungsmodelle wollen einen noch nicht hinreichend untersuchten Gegenstandsbereich mit Hilfe der Simulationsmethode zugänglich machen, in der Regel also ausgehend vom beobachtbaren Gesamtverhalten und dessen Zurückführbarkeit auf seine Komponenten (analytische Modellierung). *Entscheidungsmodelle* sollen dagegen die Möglichkeit eröffnen, zwischen Modellvarianten zu entscheiden (und ggf. zu wählen), deren Verhalten sich in eindeutiger Weise aus bestimmten Komponenten erzeugen läßt (synthetische Modellierung), meist mit der Maßgabe, eine optimale Modellvariante herauszufinden (wie beispielsweise in der Operationsforschung). - Aus den gleichen Gründen wie zuvor herrschen in der Psychologie Erkundungsmodelle vor, obgleich es gerade für die Anwendbarkeit der Simulationsmethode in der psychologischen Praxis - beispielsweise in pädagogischen und in therapeutischen Bereichen - wünschenswert sein dürfte, vermehrt auch mit Entscheidungsmodellen arbeiten zu können.

Eine andere, nicht nach formalen, sondern nach *inhaltlichen* Kriterien ausgerichtete Klassifikation von Simulationsmodellen haben Dutton & Starbuck (1971) entwickelt, die für ihre Verwendbarkeit besonders in den Sozialwissenschaften spricht. Dutton & Starbuck unterscheiden nach den Gegenständen - oder ,,Merkmalsträgern“ - und deren Zusammenhängen, die Bezugs- und Anwendungsbereich zugleich für die Entwicklung von Simulationsmodellen sind. Danach sind Gegenstands- und Merkmalsbereich der ,,Computer-Simulation menschlichen Verhaltens“ (so der Titel des Sammelbandes von Dutton & Starbuck):

(1) Individuen (im Sinne von Einzelfalluntersuchungen),
(2) interagierende Individuen,
(3) aggregierte Individuen (im Sinne von in Gruppen zusammengefaßten Individuen),
(4) aggregierte und interagierende Individuen.

Aus dieser Übersicht ist zu entnehmen, daß mit den vier Kategorien der gesamte Bereich der Sozialwissenschaften - von der Psychologie über Soziologie und Politologie bis hin zu den Wirtschaftswissenschaften - abgedeckt werden kann. Die Psychologie ist verständlicherweise mehr in den Kategorien (1) und (2), Individuen und interagierende Individuen vertreten, wie die Beiträge in dem Sammelband von Dutton & Starbuck zeigen, aber auch, wie sich aus der von den Autoren ausgearbeiteten Bibliographie von über 2000 Titeln

der bis zum Ende der 60er Jahre erschienenen Arbeiten zur Computer-Simulation ablesen läßt.

Zusammenfassend kann man sagen, daß die Computer-Simulation in der Psychologie den Typ der dynamischen, deterministischen, qualitativen und analytischen Erkundungsmodelle zur Nachbildung menschlichen Verhaltens am Beispiel von einzelnen und interagierenden Individuen bevorzugt. An dem Programmbeispiel des nächsten Abschnitts soll diese Typologie, aus der sich das Paradigma der Computer-Simulation in seiner in der Psychologie vorherrschenden Form ableiten läßt, illustrativ konkretisiert werden.

2.2 Programmbeispiel: „Simple Concept Attainment“

Ein bevorzugtes Gebiet mathematischer Modellbildungen in der Psychologie ist die experimentelle Begriffsbildungsforschung. In einer illustrativen Arbeit haben Gregg & Simon (1967) am Beispiel der einfachen Begriffsbildung („Simple Concept Attainment“) eine Gegenüberstellung von Prozeßmodellen und stochastischen Theorien vorgenommen, die sich für eine Einführung in das Paradigma der Computer-Simulation in besonderer Weise eignet, da sich hieraus sowohl die Methodik als auch die Problematik des Verfahrens veranschaulichen lassen.

Im experimentellen Design zur einfachen Begriffsbildung (genauer: „Begriffsfindung“) werden der Versuchsperson n-dimensionale Reize mit je zwei möglichen Ausprägungen vorgegeben (z.B. „großer roter Kreis“ mit den drei Dimensionen „Größe“, „Farbe“ und „Form“ und deren zweifachen Ausprägungsmöglichkeiten „klein, groß“, „rot, blau“ und „Kreis, Quadrat“). Relevant für die Begriffsbildung ist genau eine Ausprägung einer bestimmten Dimension (z.B. „blau“), die die Versuchsperson im Verlauf des Experiments herauszufinden hat, indem sie mit „ja“ oder „nein“ antwortet, je nachdem, ob das von ihr vermutete Konzept in dem vom Versuchsleiter vorgelegten Beispiel enthalten ist oder nicht. Auf jede Antwort der Versuchsperson gibt der Versuchsleiter eine Rückmeldung darüber, ob die Antwort „richtig“ oder „falsch“ ist. Der Versuch wird so lange fortgesetzt, bis die Anzahl richtiger Antworten der Versuchsperson hintereinander einen bestimmten, vom Versuchsleiter festgelegten Kriteriumswert erreicht hat.

2.2.1 Flußdiagrammdarstellung

Ein nützlicher vorbereitender Schritt in der Entwicklung eines Simulationsmodells ist - noch vor der eigentlichen Modelldarstellung in einer bestimmten Programmiersprache - die Ausarbeitung einer Verlaufsskizze des Modellver-

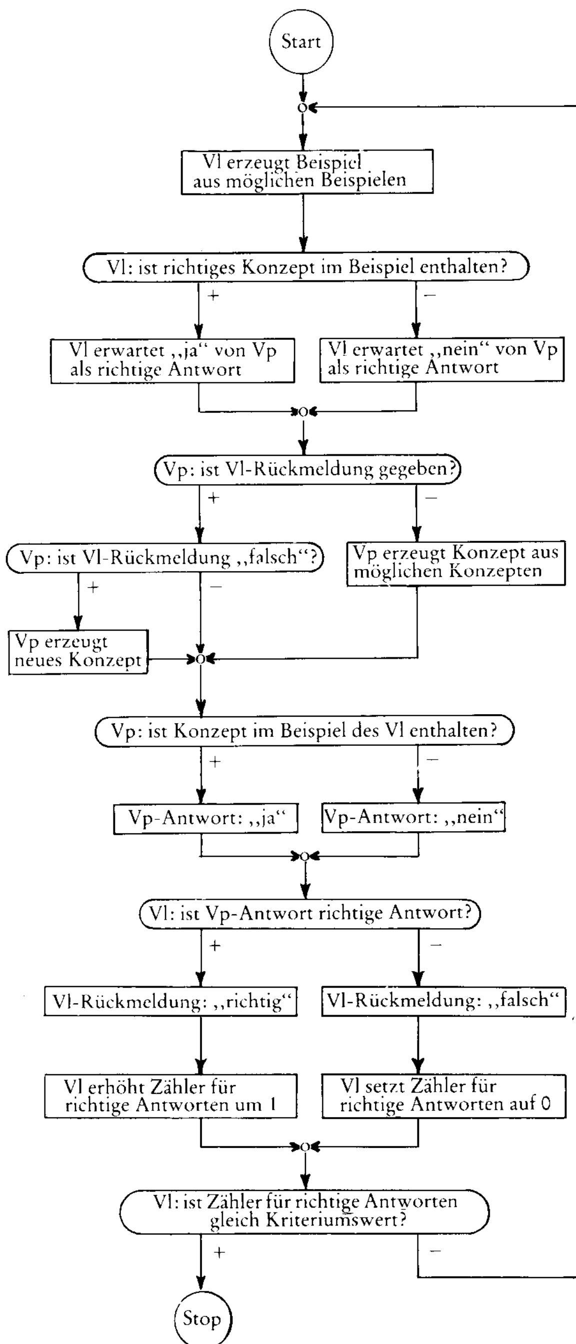

Abb. 1: Flußdiagramm zur einfachen Begriffsbildung.

haltens in Form eines Flußdiagramms. Ein *Flußdiagramm* ist die graphische Darstellung des Verhaltensablaufs aufgrund der im Modell auszuführenden Operationen und Entscheidungen. Operationen werden in einem rechteckigen Kästchen mit je einem Eingang und Ausgang, *Entscheidungen* über alternative Wege in einem ovalen Kästchen mit einem Eingang und zwei Ausgängen dargestellt; Richtung und Reihenfolge der Operationen und Entscheidungen wird durch Pfeile zwischen den Kästchen angedeutet.

In Abb. 1 ist nun in Form eines Flußdiagramms der Versuchsablauf zur einfachen Begriffsbildung schon recht detailliert dargestellt. Klar erkennbar sind drei Hauptphasen des Experiments: (1) die Aktivitäten des Versuchsleiters (Vl) zur Vorgabe eines Reizbeispiels, (2) die Handlungen der Versuchsperson (Vp) für ihre Antwort auf die Beispielvorgabe und (3) die Operationen und Entscheidungen des Versuchsleiters für eine Rückmeldung an die Versuchsperson und zur eventuellen Beendigung des Experiments.

Die Entwicklung eines angemessenen Simulationsmodells für den interessierenden Gegenstandsbereich kann jedoch nicht bei einer Flußdiagrammdarstellung stehen bleiben, da sie - trotz aller Ausführlichkeit - noch zu ungenau ist. Was beispielsweise ,,Vl erzeugt Beispiel aus möglichen Beispielen“ oder ,,Vp erzeugt Konzept aus möglichen Konzepten“ genau heißen soll, d.h. welche konkreten Operationen im einzelnen hier wirklich auszuführen sind, ist noch viel zu unbestimmt, um schon von einem Simulationsmodell der einfachen Begriffsbildung sprechen zu können, das das individuelle Verhalten von Versuchsleiter und Versuchsperson in allen interessierenden Aspekten nachzubilden gestattet.

Der wichtigste Schritt in der Entwicklung eines Simulationsmodells ist daher die Programmierung des Modells in einer geeigneten Programmiersprache, um den Verhaltensablauf tatsächlich auf einem Rechner - und zwar Schritt für Schritt - beobachten zu können. In den folgenden Unterabschnitten wird eine derartige Programmierung exemplarisch für das ,,Simple Concept Attainment“ - in Anlehnung an die Arbeit von Gregg & Simon (1967) - vorgeführt, wobei die Programmiersprache LOGO ihrer Einfachheit und leichten Verständlichkeit wegen als Illustrationssprache verwendet werden soll.

2.2.2 Das Hauptprogramm (Versuchsablaufprogramm)

In Abb. 2 ist das Hauptprogramm des ,,Simple Concept Attainment“ wiedergegeben, wie es sich in der Programmiersprache LOGO formulieren läßt; es beschreibt den Versuchsablauf in seinen drei Hauptphasen, wie sie im vorigen Unterabschnitt herausgestellt wurden.

```
to  SIMPLE.CONCEPT.ATTAINMENT
    :TRUE.CONCEPT:
    :POSSIBLE.CONCEPTS:
    :CRITERION.VALUE:
    :MODEL.VARIANT:
10  make ,,FEEDBACK“ :empty:
20  make ,,RIGHT.ANSWER.COUNT“ 0
30  print :empty:
40  print sentence ,,INSTANCE IS A“ EXPERIMENTER’S.INSTANCE
50  print sentence ,,*** I SAY“ SUBJECT’S.ANSWER
60  print sentence ,,YOUR ANSWER IS“ EXPERIMENTER’S.FEEDBACK
70  if :RIGHT.ANSWER.COUNT: = :CRITERION.VALUE:
    then stop
    else go to line 30
end
```

Abb. 2: Hauptprogramm (Versuchsablaufprogramm) zur einfachen Begriffsbildung.

Bevor auf inhaltliche Einzelheiten dieses Programms eingegangen werden kann, seien einige Erläuterungen zur LOGO-Programmierung vorangestellt:

(1) Alle hier *klein* geschriebenen Ausdrücke und die Zahlen, die Rechenzeichen (+, -, *, /), das Gleichheitszeichen (=) sowie die numerischen Prädikate (<, >) gehören zum Grundvokabular von LOGO. Die meisten dieser Ausdrücke sind schon aufgrund ihrer Wortwahl selbsterklärend, so daß auf ihre genaue Beschreibung verzichtet werden kann. (Anmerkung: In LOGO selbst gibt es die Unterscheidung zwischen Groß- und Kleinschreibung nicht in dieser, sondern in der umgekehrten Form, d.h. die zum Grundvokabular von LOGO gehörenden Ausdrücke sind stets groß zu schreiben, während die benutzerdefinierten Ausdrücke wahlweise groß oder klein geschrieben werden können. Die hier gewählte Schreibweise dient lediglich zu Darstellungszwecken für die vorliegende Arbeit.)

(2) Die *groß* geschriebenen Ausdrücke sind vom Benutzer für seine Programmierung frei wählbare Bezeichnungen, für die es folgende syntaktische Vereinbarungen gibt:

(a) Ein in Anführungszeichen eingeschlossener Ausdruck wird von LOGO als eine wörtliche Zeichenfolge - als ein ,,Literale“ - gelesen und nicht weiter ausgewertet. Literale sind - als ,,LOGO-Wörter“ oder als ,,LOGO-*Sätze“*, die aus LOGO-Wörtern bestehen - die Grunddaten bzw. die Informationen, die von LOGO verarbeitet werden können; dabei werden die Zahlen als LOGO-Wörter dargestellt, ohne daß sie in Anführungszeichen zu setzen sind. Der Ausdruck ,,FEEDBACK“ in Programmzeile 10 von Abb. 2 ist ein Beispiel für ein LOGO-Wort, der Ausdruck ,,INSTANCE IS

A“ in Programmzeile 40 ein Beispiel für einen LOGO-Satz, während die Zahl 0 in Programmzeile 20 ein Beispiel für eine ohne Anführungsstriche geschriebene LOGO-Zahl ist (wie übrigens auch alle Zeilennummern).

(b) Ein in Doppelpunkte eingeschlossener Ausdruck wird von LOGO als der Name einer Variable interpretiert, deren *Werte* irgendwelche LOGO-Wörter oder LOGO-Sätze sein können (einschließlich des leeren Ausdrucks „“, für den auch der LOGO-Name :empty: stehen kann). Beispielsweise ist der Ausdruck :RIGHT.ANSWER.COUNT: in Programmzeile 70 der Name einer Variable, deren Wert in Programmzeile 20 mit der LOGO-Anweisung ‚make‘ und dem entsprechenden LOGO-Wort ,,RIGHT.ANSWER. COUNT“ zunächst einmal auf 0 gesetzt wird.

(c) Alle übrigen Ausdrücke werden von LOGO als auszuführende *Funktionen* (,,Prozeduren“) verstanden, die entweder eine Anweisung darstellen (Beispiel: ‚print‘ in den Programmzeilen 30-60) oder eine Operation mit LOGO-Wörtern oder LOGO-Sätzen bilden (Beispiel: ‚sentence‘ in den Programmzeilen 40-60). Der Unterschied zwischen Anweisungen und Operationen wird aus den weiteren Beispielen noch ersichtlich werden. (Anmerkung: Die LOGO-Ausdrücke ‚of‘ und ‚and‘ - sie kommen in den weiteren Beispielen wiederholt vor - sind keine LOGO-Funktionen, sondern Füllwörter [,,noise words“], die von LOGO überlesen werden, zur besseren Lesbarkeit von LOGO-Programmen jedoch beitragen können.)

(3) Die vom Benutzer für sein Programm zu definierenden Funktionen werden mit der LOGO-Anweisung ‚to‘ eröffnet und mit der LOGO-Anweisung ‚end‘ abgeschlossen. Dazwischen stehen die mit einer fortlaufenden, frei wählbaren Zeilennumerierung versehenen Anweisungen und Operationen, die innerhalb der Funktion zum Zeitpunkt ihres Aufrufs ausgeführt werden sollen. Der vom Benutzer für seine Funktionsdefinition zu vereinbarende Name wird anschließend an die LOGO-Anweisung ‚to‘ geschrieben, gefolgt von einer Angabe der in Doppelpunkte eingeschlossenen Variablennamen, die für den späteren Funktionsaufruf von Interesse sind. (Anmerkung: Will man - wie im Beispiel mit dem Wort SIMPLE.CONCEPT.ATTAINMENT - längere Funktionsnamen bilden, so sind die einzelnen Wörter am besten mit einem Punkt zu einem LOGO-Wort aneinanderzuhängen, da der sonst übliche Bindestrich in LOGO schon als Minuszeichen vergeben ist; gleiches gilt auch für die Bildung längerer Variablennamen.)

Nach diesen Erläuterungen dürfte das LOGO-Programm von Abb. 2 schon leichter zu verstehen sein: Das Programm hat den Funktionsnamen SIMPLE. CONCEPT.ATTAINMENT mit den Variablen

:TRUE.CONCEPT: (das vom Versuchsleiter gewählte und von der Versuchsperson zu findende Konzept),

:POSSIBLE.CONCEPTS: (die vorzugebenden 2n Werte für die n-dimensionalen Reizbeispiele),
:CRITERION.VALUE: (der für die Beendigung des Versuchs vom Versuchsleiter gewählte Kriteriumswert),
:MODEL.VARIANT: (die für das Versuchspersonenverhalten formulierbaren Modellvarianten; Einzelheiten dazu weiter unten).

In den Programmzeilen 10 und 20 werden die Voreinstellungen für den Versuchsbeginn vorgenommen, d.h. die Rückmeldung seitens des Versuchsleiters ist zunächst leer und die Anzahl richtiger Antworten der Versuchsperson 0. Kernstück des Programms sind die Druckanweisungen der Programmzeilen 30-60 (sie entsprechen den in der Flußdiagrammdarstellung genannten drei Hauptphasen des Experiments): Zunächst wird das Versuchsleiterbeispiel in Form des LOGO-Satzes ,,INSTANCE IS A . . .“ ausgegeben, dann folgt die

```
SIMPLE.CONCEPT.ATTAINMENT
  ,,BLUE“
  ,,SMALL BIG RED BLUE CIRCLE SQUARE“
  5
  ,,GLOBAL.CONSISTENCY.MODEL“

INSTANCE IS A BIG BLUE CIRCLE
**** I SAY NO
YOUR ANSWER IS WRONG

INSTANCE IS A SMALL RED SQUARE
**** I SAY NO
YOUR ANSWER IS RIGHT

INSTANCE IS A SMALL RED CIRCLE
**** I SAY NO
YOUR ANSWER IS RIGHT

INSTANCE IS A BIG BLUE SQUARE
**** I SAY YES
YOUR ANSWER IS RIGHT

INSTANCE IS A BIG RED SQUARE
**** I SAY NO
YOUR ANSWER IS RIGHT

INSTANCE IS A SMALL BLUE SQUARE
**** I SAY YES
YOUR ANSWER IS RIGHT
```

Abb. 3: Probelauf des SIMPLE.CONCEPT.ATTAINMENT-Programms.

Antwort der Versuchsperson in Form von ,,**** I SAY . . .“ und anschließend gibt der Versuchsleiter seine Rückmeldung in Form von ,,YOUR ANSWER IS...“. Programmzeile 70 dient als Test dafür, ob der Versuch durch Kriteriumserreichung abgeschlossen werden kann (‚stop‘) oder noch fortzusetzen ist (‚go to line 30‘); dies entspricht der letzten Zeile in dem Flußdiagramm von Abb. 1.

Ein Versuchsablauf könnte beispielsweise das in Abb. 3 gezeigte Aussehen haben.

Dieses Beispiel liest sich durchaus wie ein tatsächliches Versuchsablaufprotokoll einer Experimentalsitzung und hat zumindest aus dieser - noch oberflächlichen - Sicht Anspruch auf eine realitätsgerechte Beschreibung beobachtbaren Verhaltens.

2.2.3 Zur ,,Binnenstruktur“ der Informationsverarbeitung

Interessanter für die Computer-Simulation kognitiver Vorgänge ist stets die ,,Binnenstruktur“ der innerhalb der Versuchsperson ablaufenden Informationsverarbeitung. Tatsächlich würde ja auch das in Abb. 2 formulierte Programm noch nicht laufen, wenn man es so wie im Kopf von Abb. 3 angegeben in LOGO aufriefe, denn in den Programmzeilen 40-60 des Hauptprogramms sind die drei Funktionen EXPERIMENTER’S.INSTANCE, SUBJECT’S. ANSWER und EXPERIMENTER’S.FEEDBACK noch vom Benutzer zu definieren, bevor er das Programm starten kann.

In den nachfolgenden Abb. 4-6 sind diese ,,Unterprogramme“ des SIMPLE. CONCEPT.ATTAINMENT wiedergegeben. Sie sind die eigentlichen ,,Simulationsprogramme“ für das Versuchsleiterverhalten einerseits und das Versuchspersonenverhalten andererseits.

```
to   EXPERIMENTER’S.INSTANCE
10   make ,,INSTANCE“
           GENERATE.INSTANCE from :POSSIBLE.CONCEPTS:
20   if CONTAINS :INSTANCE: :TRUE.CONCEPT:
     then make ,,RIGHT.ANSWER“ ,,YES“
     else make ,,RIGHT.ANSWER“ ,,NO“
30   output :INSTANCE:
end
```

Abb. 4: Unterprogramm für das Erzeugen eines Beispiels durch den Versuchsleiter.

Wie ein Vergleich zeigt, entspricht EXPERIMENTER'S.INSTANCE (Abb. 4) dem oberen, dreizeiligen Teil des Flußdiagramms in Abb. 1. In Programmzeile 10 von EXPERIMENTER'S.INSTANCE wird das der Versuchsperson vorzulegende Beispiel mit dem - vom Benutzer noch zu definierenden - Teilprogramm GENERATE.INSTANCE erzeugt. In Programmzeile 20 wird geprüft, ob das zu lernende „wirkliche Konzept" in diesem Beispiel enthalten ist oder nicht, worauf der Versuchsleiter als „richtige Antwort" „ja" oder „nein" von der Versuchsperson erwarten wird, wenn diese das Konzept gefunden hat. In Programmzeile 30 schließlich wird das Beispiel ausgegeben, um vom Hauptprogramm in Programmzeile 40 (Abb. 2) in Form des LOGO-Satzes „INSTANCE IS A :INSTANCE:" ausgedruckt zu werden (vgl. auch Abb. 3).

```
to SUBJECT'S.ANSWER
10 if not :FEEDBACK: = :empty:
    then go to line 40
20 make ,,MY.POSSIBLE.CONCEPTS" :POSSIBLE.CONCEPTS:
30 make ,,MY.CONCEPT"
        GENERATE.CONCEPT from :MY.POSSIBLE.CONCEPTS:
40 if :FEEDBACK: = ,,WRONG"
    then do :MODEL.VARIANT:
50 if CONTAINS :INSTANCE: :MY.CONCEPT:
    then make ,,ANSWER" ,,YES"
    else make ,,ANSWER" ,,NO"
60 make ,,OLD.INSTANCE" :INSTANCE:
70 output :ANSWER:
end
```

Abb. 5: Unterprogramm für die Antwort der Versuchsperson.

SUBJECT'S.ANSWER (Abb. 5), das dem mittleren, fünfzeiligen Teil des Flußdiagramms in Abb. 1 entspricht, ist dasjenige Unterprogramm, das von zentralem Interesse für die in einer Versuchsperson bei der einfachen Begriffsbildung ablaufenden Prozesse der Informationsverarbeitung ist. Zunächst wird die Versuchsperson, da sie im ersten Versuchsdurchgang noch kein :FEEDBACK: vom Versuchsleiter bekommen hat (vgl. Programmzeile 10 von SUBJECT'S.ANSWER), ihr mögliches Konzeptrepertoire aus den vom Versuchsleiter vorgegebenen möglichen Konzepten bilden (Programmzeile 20). Dann wählt sie sich ein eigenes Konzept mit dem - noch zu definierenden - Teilprogramm GENERATE.CONCEPT aus ihren möglichen Konzepten aus (Programmzeile 30). Da im ersten Versuchsdurchgang aufgrund des noch fehlenden :FEEDBACK: die Programmzeile 40 nicht zur Ausführung kommt, wird die Versuchsperson laut Programmzeile 50 ihre Antwort generieren, und

zwar derart, daß sie „ja“ sagen wird, wenn ihr Konzept in dem vorgelegten Beispiel enthalten ist, und „nein“, wenn dies nicht der Fall ist. In Programmzeile 70 gibt sie diese Antwort dann aus, und zwar in Form des LOGO-Satzes „**** I SAY :ANSWER:“ aufgrund der Programmzeile 50 des Hauptprogramms (vgl. Abb. 2 und für den Probelauf Abb. 3). Programmzeile 60 von SUBJECT'S.ANSWER dient lediglich dem späteren Erinnern des gerade vorgelegten Beispiels und ist eigentlich nur für eine bestimmte Modellvariante des Versuchspersonenverhaltens relevant.

Hat die Versuchsperson ihre Antwort gegeben, dann verlangt die Experimentalanordnung ein :FEEDBACK: auf diese Antwort durch den Versuchsleiter (vgl. Programmzeile 60 in SIMPLE.CONCEPT.ATTAINMENT, Abb. 2). Das Unterprogramm EXPERIMENTER'S.FEEDBACK (Abb. 6), das dem unteren, dreizeiligen Teil des Flußdiagramms in Abb. 1 entspricht (außer dessen letzter Zeile), erfüllt diesen Zweck.

```
to EXPERIMENTER'S.FEEDBACK
10 if :ANSWER: = :RIGHT.ANSWER:
     then make „FEEDBACK“ „RIGHT“
     else make „FEEDBACK“ „WRONG“
20 if :FEEDBACK: = „RIGHT“
     then make „RIGHT.ANSWER.COUNT“:RIGHT.ANSWER.COUNT: + 1
     else make „RIGHT.ANSWER.COUNT“ 0
30 output :FEEDBACK:
end
```

Abb. 6: Unterprogramm für die Rückmeldung über die Versuchspersonenantwort.

In Programmzeile 10 wird geprüft, ob die Antwort der Versuchsperson der zu erwartenden richtigen Antwort entspricht, wenn die Versuchsperson das zu suchende Konzept gefunden haben sollte; im positiven Fall wird die Rückmeldung auf „richtig“ gesetzt, im negativen Fall auf „falsch“, und in Programmzeile 30 wird dann der entsprechende Wert ausgegeben. Zuvor jedoch (Programmzeile 20) wird noch der Zähler für die richtigen Antworten der Versuchsperson um 1 erhöht, wenn das :FEEDBACK: „richtig“ ergab, bzw. auf 0 gesetzt, wenn es „falsch“ ergab. Dieser Zähler wird für die Beendigung des Experiments benötigt (vgl. Programmzeile 70 des Hauptprogramms, Abb. 2).

2.2.4 Die Modellvarianten

Die Modellvarianten, die in den einzelnen Versuchsdurchgängen immer dann zur Ausführung kommen, wenn das :FEEDBACK: durch den Versuchsleiter

„falsch“ ist (Programmzeile 40 von SUBJECT’S.ANSWER), sind das Kernstück der Theorie, die man zur psychologischen Erklärung der einfachen Begriffsbildung heranziehen kann (sie beinhalten einen Aspekt des Simulationsmodells, der in der Flußdiagrammdarstellung von Abb. 1 nur mit einem kleinen, unscheinbaren Operationskästchen namens „Vp erzeugt neues Konzept“ ausgewiesen ist). Gregg & Simon (1967) formulierten in ihrer Arbeit vier solcher Modellvarianten (und das sind bei weitem nicht alle, die man sich für die einfache Begriffsbildung ausdenken kann), deren Programmierung in LOGO in den folgenden Abb. 7-10 wiedergegeben ist.

```
to GLOBAL.CONSISTENCY.MODEL
10 make ,,MY.POSSIBLE.CONCEPTS“
        REMOVE :MY.CONCEPT: from :MY POSSIBLE.CONCEPTS:
20 make ,,NEW.CONCEPT“
        GENERATE.CONCEPT from :MY.POSSIBLE.CONCEPTS:
30 make ,,MY.CONCEPT“ :NEW.CONCEPT:
end
```

Abb. 7: Erste Modellvariante der einfachen Begriffsbildung.

Das Modell für ein möglichst optimales Versuchspersonenverhalten ist das GLOBAL.CONSISTENCY.MODEL (Abb. 7): Die Versuchsperson entfernt ihr augenblickliches Konzept aus ihrem möglichen Konzeptrepertoire (Programmzeile 10), wenn das :FEEDBACK: durch den Versuchsleiter „falsch“ war (vgl. Programmzeile 40 von SUBJECT’S.ANSWER, Abb. 5, wo die jeweilige Modellvariante durch die LOGO-Anweisung ‚do‘ aufgerufen wird). Dann bildet die Versuchsperson ein neues Konzept aus den verbleibenden möglichen Konzepten (Programmzeile 20 von Abb. 7) und macht dieses zu ihrem Konzept (Programmzeile 30), mit dem sie weiterarbeiten wird (vgl. Programmzeile 50 von SUBJECT’S.ANSWER, Abb. 5). Es ist klar, daß mit dieser Strategie das gesuchte Konzept in jedem Fall von der Versuchsperson gefunden werden kann, da mit der Zeit alle falschen Konzepte aus der Menge der möglichen Konzepte ausgeschlossen werden.

```
to LOCAL.CONSISTENCY.MODEL
20 make ,,NEW.CONCEPT“
        GENERATE.CONCEPT from :MY.POSSIBLE.CONCEPTS:
30 if CONTAINS :OLD.INSTANCE: :NEW.CONCEPT:
   then go to line 20
   else make ,,MY.CONCEPT“ :NEW.CONCEPT:
end
```

Abb. 8: Zweite Modellvariante der einfachen Begriffsbildung.

Etwas weniger optimal ist die Strategie des LOCAL.CONSISTENCY. MODEL (Abb. 8). Hier wird lediglich geprüft, nachdem die Versuchsperson ein neues Konzept generiert hat (Programmzeile 20), ob dieses in dem vorangehenden, aber ,,falsch“ beantworteten Beispiel (:OLD.INSTANCE:) enthalten ist oder nicht (Programmzeile 30); wenn ja, wird nochmals ein neues Konzept generiert, wenn nein, wird das neue Konzept als das weiter zu verwendende Konzept beibehalten. Diese Modellvariante ist die einzige, die von der Konstruktion ‚make ,,OLD.INSTANCE“ :INSTANCE:‘ in SUBJECT’S.ANSWER (Abb. 5, Programmzeile 60) Gebrauch macht. Denkbar wäre jedoch auch eine Kombination dieses zusätzlichen Verarbeitungsschrittes mit dem GLOBAL.CONSISTENCY.MODEL (in dessen Programmzeile 30 einzubauen).

```
to LOCAL.NON.REPLACEMENT.MODEL
20 make ,,NEW.CONCEPT“
        GENERATE.CONCEPT from :MY POSSIBLE.CONCEPTS:
30 if :NEW.CONCEPT: = :MY.CONCEPT:
    then go to line 20
    else make ,,MY.CONCEPT“ :NEW.CONCEPT:
end
```

Abb. 9: Dritte Modellvariante der einfachen Begriffsbildung.

Das LOCAL.NON.REPLACEMENT.MODEL (Abb. 9) vergleicht nur noch, ob das neu generierte Konzept dem vorher verwendeten, aber falschen Konzept entspricht (Programmzeile 20); wenn ja, wird erneut ein Konzept generiert, wenn nein, wird es beibehalten.

```
to REPLACEMENT.MODEL
20 make ,,NEW.CONCEPT“
        GENERATE.CONCEPT from :MY.POSSIBLE.CONCEPTS:
30 make ,,MY.CONCEPT“ :NEW.CONCEPT:
end
```

Abb. 10: Vierte Modellvariante der einfachen Begriffsbildung.

Die simpelste Modellvariante ist das REPLACEMENT.MODEL (Abb. 10). In diesem wird lediglich ein neues Konzept generiert und dann beibehalten. Dabei kann es natürlich leicht vorkommen, daß das neu generierte Konzept

genau das gleiche wie das zuvor verwendete, aber bereits als falsch erwiesene Konzept ist, also eine wenig „intelligente“ Strategie einer Versuchsperson.

2.2.5 Abschließende Funktionsdefinitionen

Die in den bisher vorgestellten Unterprogrammen vorkommenden Teil- und Hilfsprogramme sind in den nachfolgenden Abb. 11-16 wiedergegeben.

```
to GENERATE.INSTANCE :CONCEPTS:
10 make „INSTANCE“ TAKE.ONE.OF.TWO from :CONCEPTS:
20  make „CONCEPTS“ butfirst of butfirst of :CONCEPTS:
30 if :CONCEPTS: = :empty:
    then output :INSTANCE:
40 make „INSTANCE“
         sentence of :INSTANCE:
                   and TAKE.ONE.OF.TWO from :CONCEPTS:
50  go to line 20
end
```

Abb. 11: Teilprogramm zum Erzeugen eines Beispiels durch den Versuchsleiter.

```
to GENERATE.CONCEPT :CONCEPTS:
10 make „NUMBER“ random
20  if not :NUMBER: < count of :CONCEPTS:
    then go to line 10
30 if :NUMBER: = 0
    then output first of :CONCEPTS:
40 make „NUMBER“ :NUMBER: - 1
50 make „CONCEPTS“ butfirst of :CONCEPTS:
60  go to line 30
end
```

Abb. 12: Teilprogramm zum Erzeugen eines Konzepts durch die Versuchsperson.

```
to TAKE.ONE.OF.TWO :SENTENCE:
10  if (remainder of random and 2) = 0
    then output first of :SENTENCE:
    else output first of butfirst of :SENTENCE:
end
```

Abb. 13: Hilfsprogramm zur Ausgabe eines Elementes aus einem zweifach gegliederten Satz.

```
to CONTAINS :SENTENCE: :WORD:
10 if :SENTENCE: = :empty:
     then output „false“
20   if :WORD: = first of :SENTENCE:
     then output ,,true“
     else output CONTAINS butfirst of :SENTENCE: :WORD:
end
```

Abb. 14: Hilfsprogramm zum Prüfen auf Enthaltensein eines Wortes in einem Satz.

```
to REMOVE :WORD: :SENTENCE:
10 if :SENTENCE: = :empty:
     then output „ “
20   if :WORD: = first of :SENTENCE:
     then output butfirst of :SENTENCE:
     else output
          sentence of first of :SENTENCE:
                    and REMOVE :WORD: from butfirst of :SENTENCE:
end
```

Abb. 15: Hilfsprogramm zum Entfernen eines Wortes aus einem Satz.

```
to   from :ANYTHING:
10 output :ANYTHING:
end
```

Abb. 16: Definition des Füllwortes ‚from‘.

Von psychologischer Bedeutung für die Simulation der einfachen Begriffsbildung sind nur noch die beiden Teilprogramme GENERATE.INSTANCE und GENERATE.CONCEPT. Mit GENERATE.INSTANCE (Abb. 11) wird ein Beispiel mit einer zufälligen Verteilung der beiden Werte des n-dimensionalen Reizes erzeugt, so daß jedes vom Versuchsleiter vorgelegte Beispiel eine Zufallsauswahl aus dem Reizmaterial darstellt (um beispielsweise Reihungseffekte zu vermeiden). Mit GENERATE.CONCEPT (Abb. 12) wird von der Versuchsperson eine Zufallsauswahl aus ihren möglichen Konzepten gemacht, wobei die psychologische Annahme zugrunde liegt, eine Bevorzugung bestimmter Reizdimensionen (wie z.B. Farbe oder Form) werde von der Versuchsperson nicht vorgenommen (eine empirisch durchaus widerlegbare Annahme).

Zum besseren Verständnis dieser Teilprogramme und der in Abb. 13-16 wiedergegebenen Hilfsprogramme (die ihrerseits nur noch von technischem

Interesse sind), seien noch einige Erläuterungen zur LOGO-Programmierung angefügt:

(1) Die LOGO-Operationen ‚first' und ‚butfirst' dienen zum *Zerlegen* von LOGO-Wörtern und LOGO-Sätzen; ‚first' liefert den ersten Buchstaben eines LOGO-Wortes bzw. das erste Wort eines LOGO-Satzes, ‚butfirst' liefert den Rest eines LOGO-Wortes ohne dessen ersten Buchstaben bzw. den Rest eines LOGO-Satzes ohne dessen erstes Wort.

(2) Die LOGO-Operation ‚random' liefert eine zufällige Zahl von 0-9, die LOGO-Operation ‚remainder' gibt den ganzzahligen Rest der Division zweier Zahlen aus, und die LOGO-Operation ‚count' zählt die Anzahl der Elemente eines LOGO-Wortes bzw. eines LOGO-Satzes.

(3) Im Unterschied zu allen anderen Programmen sind die beiden Hilfsprogramme CONTAINS (Abb. 14) und REMOVE (Abb. 15) *rekursiv* definierte Funktionen, d.h. sie rufen sich selbst innerhalb ihrer Definition wieder auf (in Programmzeile 20), bis die zugehörige Endbedingung erreicht ist (in Programmzeile 10).

(4) In Abb. 16 schließlich ist gezeigt, auf welch einfache Weise es möglich ist, in LOGO sog. Füllwörter („noise words") zu schreiben, die zwar für den Programmablauf überflüssig sind (das Beispiel ‚from' ist nichts anderes als eine Identitätsoperation), für die Verständlichkeit einer Programmzeile jedoch von Nutzen sein können. (Anmerkung: Im Unterschied zu den bisherigen Konventionen ist ‚from' als benutzerdefinierte Operation nicht groß, sondern klein geschrieben, um das Schriftbild nicht mit solchen „noise words" zu belasten.)

2.3 Diskussion des Programmbeispiels

2.3.1 Modellcharakteristika

Das im vorigen Abschnitt vorgestellte Programmbeispiel des „Simple Concept Attainment" kann als Prototyp eines Simulationsmodells in der Psychologie angesehen werden:

- Es ist ein *dynamisches* Modell, denn es stellt den Zeitverlauf einer Experimentalsitzung in allen wesentlichen Aspekten dar - denen des Versuchsleiterverhaltens und insbesondere denen des Versuchspersonenverhaltens (des „Lernverhaltens" der Versuchsperson).

- Es ist ein *deterministisches* Modell (mit probabilistischen Komponenten), da das Eingabe-Ausgabe-Verhalten des Modells eindeutig bestimmt und daher

vorhersagbar ist - bis auf die beiden Zufallsprozesse der Beispielgenerierung durch den Versuchsleiter (GENERATE.INSTANCE) und der Konzeptwahl durch die Versuchsperson (GENERATE.CONCEPT).

- Es ist ein qualitatives Modell insofern, als alle relevanten Modellvariablen eine nicht-numerische Spezifikation aufweisen, ohne jedoch eine weitergehende Quantifizierbarkeit des beobachtbaren Modellverhaltens auszuschließen.

- Es ist ein *analytisches* Modell, da von dem im realen Experiment beobachtbaren Gesamtverhalten ausgegangen wird, um das psychologisch zugrundeliegende Komponentenverhalten zu erschließen.

- Es ist ein *Erkundungsmodell* dahingehend, daß der Gegenstandsbereich der einfachen Begriffsbildung mit dem Hilfsmittel der Computer-Simulation in einer Weise untersucht werden kann, wie dies mit den herkömmlichen Mitteln der quantitativen Experimentalauswertung und der mathematischen Modellbildung nicht möglich ist. Gleichzeitig ist mit der Formulierung der verschiedenen Modellvarianten (GLOBAL.CONSISTENCY.MODEL, LOCAL.CONSISTENCY.MODEL, LOCAL.NON.REPLACEMENT. MODEL, REPLACEMENT.MODEL) auch die Möglichkeit gegeben, das Simulationsmodell im Sinne eines *Entscheidungsmodells* zu nutzen - das GLOBAL.CONSISTENCY.MODEL ist eine vergleichsweise optimale, das REPLACEMENT.MODEL die simpelste Variante der einfachen Begriffsbildung.

- Bezugspunkt des Simulationsmodells sind *interagierende Individuen,* nämlich Versuchsleiter und Versuchsperson. Will man sich allein auf das an der Versuchsperson beobachtbare Lernverhalten beschränken, da nur dieses einer psychologischen Erklärung bedarf, dann ist das isoliert betrachtete *Individuum* Bezugspunkt der Modellbildung.

2.3.2 Nicht-numerisches Programmieren

Hauptkennzeichen von Simulationsmodellen in der Psychologie ist, daß es sich um *nicht-numerische Modelle* handelt, für die eine ganz bestimmte Art des Programmierens charakteristisch ist, nämlich die des „nicht-numerischen Programmierens“. Nach Harbordt (1974, S. 41-42) ist ein nicht-numerisches Modell im einzelnen durch folgende Merkmale gekennzeichnet:

(1) Der Gegenstandsbereich wird auf nicht-numerische Weise dargestellt, d.h. die Modellvariablen beschreiben den realen Prozeß oder das reale System in qualitativen Kategorien. (Das angeführte Programmbeispiel ist eine direkte Übersetzung der experimentellen Versuchsdurchführung in die nicht-numerische Programmiersprache LOGO.)

(2) Die qualitativen Variablen werden in einer Hierarchie von „Listen“ angeordnet, und ihre Verarbeitung besteht in der Veränderung solcher Listen durch elementare Prozesse zum Sortieren, Ordnen, Speichern, Wiederaufsuchen, Vergleichen und Auswählen der jeweiligen Variablenwerte. (Diese Elementarprozesse werden in LOGO durch dessen Grundvokabular bereitgestellt.)

(3) Der Modellablauf besteht in einer verschachtelten Abfolge von Programmen und Programmteilen (Haupt-, Unter-, Teil- und Hilfsprogramme des Programmbeispiels).

(4) Die Modellausgabe sind in der Regel nicht-numerische, qualitative Daten, nämlich die Inhalte bestimmter Listen von Variablenwerten, die aufgrund der Programmdefinitionen zu eindeutig bestimmten Merkmalsklassen gehören. (Jede der Funktionsdefinitionen des Programmbeispiels beinhaltet eine genaue Spezifikation der jeweiligen Programmausgabe.)

Grundlage der Erstellung eines so charakterisierten Simulationsmodells ist jedoch eine andere Form des Programmierens, als man es durch die gängigen Programmiersprachen - wie z.B. ALGOL oder FORTRAN - gewohnt ist. Diese andere Form, das *nicht-numerische* Programmieren, ist nur mit speziell zu diesem Zweck entworfenen, sog. listenverarbeitenden Programmiersprachen möglich. Ein Beispiel dieser „nicht-numerischen“ Programmiersprachen ist die - hier als Illustrationssprache verwendete - Sprache LOGO (vgl. Feurzeig et al., 1971), eine andere die in den weitaus meisten Fällen der Erstellung von Simulationsmodellen und von Programmen der „künstlichen Intelligenz“ verwendete Programmiersprache LISP („LISt Programming language“, vgl. McCarthy et al., 1962).

Gemeinsames Merkmal aller nicht-numerischen oder listenverarbeitenden Programmiersprachen ist die Tatsache, daß der Computer in diesen Sprachen nicht eigentlich „rechnet“, sondern Daten beliebiger Struktur verarbeitet, sofern sie sich in digitalisierter Form (als aus zwei Grundwerten - z.B. 0/1 oder ON/OFF - bestehende „Bit-Folgen“) im Rechner darstellen lassen (wovon die Zahlen lediglich eine bestimmte Teilmenge bilden). Ihre konkrete Bedeutung erhalten diese Zeichen und Zeichenstrukturen erst durch die vom Benutzer eingeführten Funktionsdefinitionen, die über dem Grundvokabular einer Programmiersprache in deren „Grammatik“ realisierbar sind.

2.3.3 „Listenverarbeitung”

Grundlegendes Konzept des nicht-numerischen Programmierens ist das der *Listenverarbeitung:* Alle Zeichen und Zeichenstrukturen werden in Form von Listen dargestellt und verarbeitet, wobei es keinen prinzipiellen Unterschied

zwischen Programm und Daten gibt - Programm und Daten haben die gleiche Struktur, wenn auch im Programmablauf unterschiedliche Funktion. Das Konzept einer Listenstruktur läßt sich am klarsten am Beispiel der Programmiersprache LISP veranschaulichen. Eine *Liste* ist in LISP definiert als ein Klammerausdruck, dessen Elemente Wörter („Atome“ in LISP) sein können - oder aber weitere Listen („Unterlisten“). Mit dieser Definition eröffnet sich die Möglichkeit, Programme wie auch Daten als beliebig komplexe, verschachtelte Listen aufzubauen. Ein Beispiel ist das in Abb. 17 wiedergegebene, nunmehr in LISP formulierte Hauptprogramm des „Simple Concept Attainment“, das in Abb. 2 in der Programmiersprache LOGO vorgestellt wurde. (Zur besseren Verständlichkeit sind auch hier die vom Benutzer frei wählbaren Bezeichnungen groß und die zum LISP-Vokabular gehörigen Bezeichnungen klein geschrieben.)

```
(de  SIMPLE-CONCEPT-ATTAINMENT
     (TRUE-CONCEPT POSSIBLE-CONCEPTS CRITERION-VALUE
     MODEL-VARIANT)
     (prog (FEEDBACK RIGHT-ANSWER-COUNT)
            (setq FEEDBACK nil)
            (setq RIGHT-ANSWER-COUNT 0)
     LBL (print nil)
            (print (append (quote (INSTANCE IS A))
                           (EXPERIMENTER'S-INSTANCE)))
            (print (append (quote (**** I SAY)) (SUBJECT'S-ANSWER)))
            (print (append (quote (YOUR ANSWER IS))
                           (EXPERIMENTER'S-FEEDBACK)))
            (cond ((eq RIGHT-ANSWER-COUNT CRITERION-VALUE) (return))
                  (t (go LBL))) ))
```

Abb. 17: Hauptprogramm zur einfachen Begriffsbildung in LISP.

Listenverarbeitung heißt nun, daß innerhalb einer Programmiersprache - hier LISP - jede Liste allein aufgrund ihrer Struktur und den in ihr vorkommenden Bezeichnungen eine bestimmte Bedeutung erhält. Am Beispiel von Abb. 17 ist die Listenstruktur anhand der Klammerung - und zur zusätzlichen optischen Verdeutlichung auch an den Einrückungen - klar erkennbar. Eine mit der LISP-Funktion ‚de‘ beginnende Liste erhält die Bedeutung einer Definitionsstruktur. Das zweite Element einer derartigen Struktur ist der vom Benutzer frei wählbare Name der zu definierenden Funktion. Als nächstes erwartet LISP eine Liste der Variablen oder Argumente, die später, beim Aufruf der Funktion, die einzugebenden Daten bezeichnen. Sodann folgt eine Liste, hier mit der LISP-Funktion ‚prog‘ eröffnet, die den eigentlichen Kern

der Definition enthält. Als erstes steht nach dem ‚prog' eine Liste von lokalen Variablen (die auch leer sein kann), die innerhalb des Definitionskerns lediglich eine lokale Bedeutung haben (im Unterschied zu den globalen, für alle möglichen Unterprogramme geltenden Variablen der Argumentliste nach dem Funktionsnamen). Die nachfolgende Sequenz von unterschiedlich tief geschachtelten Listen beschreibt den gleichen Programmablauf, hier in LISP-Terminologie, wie den in Abb. 2 in LOGO programmierten Versuchsablauf der einfachen Begriffsbildung. Wenn auch im Vergleich zwischen Abb. 2 und Abb. 17 der Begriff der Listenstruktur in dem LISP-Programm deutlicher zum Ausdruck kommt als in dem LOGO-Programm, so ist dieses doch - schon aufgrund seiner Wortwahl - wesentlich leichter zu verstehen. Zudem besteht auch in LOGO die Möglichkeit, eine der LISP-Notation ähnliche Klammerschreibweise zu verwenden, um zusammengehörige Programmteile - insbesondere bei mehrfachen Funktionsaufrufen innerhalb einer Programmzeile - übersichtlicher zu gestalten; in manchen Fällen müssen in LOGO sogar Klammern geschrieben werden, wie beispielsweise in der LOGO-Operation TAKE.ONE.OF.TWO (Abb. 13) am Anfang von Programmzeile 10 (weitere Beispiele finden sich in den Abb. 20, 21, 25-28).

Auch die Datenstruktur, mit der das Programmbeispiel von Abb. 17 in LISP aufzurufen wäre, ist eine Listenstruktur, wie aus Abb. 18 zu ersehen ist. (Voraussetzung für einen Programmaufruf wäre natürlich noch eine den LOGO-Programmen von Abb. 4-16 entsprechende LISP-Programmierung.)

```
(SIMPLE-CONCEPT-ATTAINMENT
    (quote BLUE)
    (quote (SMALL BIG RED BLUE CIRCLE SQUARE))
    5
    (quote GLOBAL-CONSISTENCY-MODEL))
```

Abb. 18: Aufrufbeispiel für das LISP-Programm der einfachen Begriffsbildung.

Aus dem rein syntaktischen Vergleich zwischen Abb. 17 und 18 ist ersichtlich, daß zwischen „Programm" und „Daten" kein grundsätzlicher Unterschied besteht. Der Unterschied ergibt sich erst durch den „Gebrauch" der verschieden strukturierten und inhaltlich gefüllten Listen innerhalb der Programmiersprache LISP.

Nicht-numerisch ist das Programmieren in listenverarbeitenden Sprachen insofern, als numerische Werte - wie die ‚5' in Abb. 18 - nur einen Spezialfall unter den Daten eines so konzipierten Programms darstellen; in der Regel wird also nicht „gerechnet", sondern es werden diskrete („digitalisierte") Zeichen und Zeichenstrukturen in Listenform verarbeitet.

2.3.4 Modulares Programmieren

Aus dem Programmbeispiel des „Simple Concept Attainment" wird ein wichtiges Konstruktionsprinzip von Simulationsmodellen erkennbar: das „modulare" Programmieren (Modellkonstruktion nach dem „Baukastenprinzip"). Jedes der in den Abb. 2-16 wiedergegebenen LOGO-Programme („Haupt-, Unter-, Teil- und Hilfsprogramme" in der dort eingeführten Terminologie) ist eine in sich selbständige Einheit - ein „Modul" -, und erst aus dem verschachtelten Zusammenwirken dieser „Bausteine" ergibt sich die Ablaufcharakteristik des Gesamtprogramms. Konkret heißt das für die Programmierarbeit an einem Simulationsmodell, daß man die einzelnen Programmeinheiten entweder „von oben nach unten" („top-down") oder „von unten nach oben" („bottom-up") erstellt.

Im *„top-down programming"* schreibt man zuerst das Hauptprogramm (im Programmbeispiel: SIMPLE.CONCEPT.ATTAINMENT) und setzt zunächst nur die Namen von Unterprogrammen ein (Beispiele: EXPERIMENTER'S.INSTANCE, SUBJECT'S.ANSWER, EXPERIMENTER'S.FEEDBACK), über deren genauere Definitionsstruktur man sich hier noch keine Gedanken machen muß - es reicht eine vorläufige Vorstellung über die jeweiligen Ein- und Ausgaben dieser Unterprogramme. Bei der späteren Erstellung der Unterprogramme verfährt man in analoger Weise, bis am Ende alle für das Gesamtprogramm zu schreibenden Funktionen definiert sind. (Beispiel: Beim Schreiben des Unterprogramms SUBJECT'S.ANSWER setzt man die Namen der noch undefinierten Teilprogramme GENERATE.CONCEPT und CONTAINS so ein, daß man ein ganz bestimmtes Verhalten der Modellkomponente SUBJECT'S.ANSWER nach der Definition ihrer Teilprogramme erwarten kann. Im konkreten Fall dieses Beispiels ist insbesondere zu beachten, daß die Werte der globalen Variable :MODEL.VARIANT: - die ihrerseits im Hauptprogramm SIMPLE.CONCEPT.ATTAINMENT eingeführt wurde - später von LOGO nicht als Literale, sondern als auszuführende LOGO-Funktionen - durch die LOGO-Anweisung ‚do' veranlaßt - gelesen werden sollen.) Vorteil dieser Vorgehensweise ist die Möglichkeit, von zunächst relativ allgemeinen Vorstellungen eines Simulationsmodells zu immer konkreteren und spezifischeren Details überzugehen, ohne den Gesamtzusammenhang des Modellverhaltens aus dem Auge zu verlieren; in dieser Charakterisierung entspricht das „top-down programming" der von Harbordt beschriebenen *analytischen* Modellentwicklung (vgl. Abschnitt 2.1, Punkt 4).

Im *„bottom-up programming"* geht man den umgekehrten Weg: Von der Programmierung relativ spezifischer, meist unabhängig voneinander definierbarer Modellkomponenten gelangt man durch deren Einbau in allgemeinere Programme zu der Erstellung eines Gesamtprogramms, von dessen Verhalten man anfangs nur recht globale Vorstellungen haben muß. Erst durch die zunehmende Zusammenfassung von Modellkomponenten erfahren diese Vorstellun-

gen ihre konkrete Ausgestaltung, bis am Ende das erwünschte Gesamtverhalten des Simulationsmodells erreicht wird. Der Vorteil dieser Vorgehensweise liegt in der nahezu beliebig verfeinerbaren Herausbildung eines bestimmten Modellverhaltens; in der Terminologie von Harbordt entspricht diese Programmiertechnik der *synthetischen* Erstellung von Simulationsmodellen.

In der Programmierpraxis werden allerdings diese „reinen“ Formen der Modellerstellung eher die Ausnahme als die Regel sein; eine gemischte Strategie aus „top-down“ und „bottom-up“ Programmieren ist für das modulare Programmieren - insbesondere bei der Konstruktion komplexerer und umfangreicherer Simulationsmodelle - wohl mehr kennzeichnend. Seinen größten Vorteil hat das modulare Programmieren - im Unterschied zu einem alle Modellkomponenten in einem einzigen Programm enthaltenden Hauptprogramm - in der leichteren Fehlersuche („debugging“): Ist das Modellverhalten an einer bestimmten Stelle fehlerhaft - oder entspricht es nicht den intendierten Absichten -, so hat man nur die den Fehler bewirkende Modellkomponente (das entsprechende Unterprogramm) herauszufinden und zu verbessern (oder zu ersetzen), ohne die übergeordneten Programme oder gar das Hauptprogramm verändern zu müssen. Das erleichtert das Programmieren ganz wesentlich, eingedenk der alten Programmiererfahrung, daß jedes größere Programm so seine „Macken“ („bugs“) hat.

3. Simulationsmodelle und psychologische Theorienbildung

Im Unterschied zu dem empiristischen, oft nur mit ad-hoc-Hypothesen begründeten Vorgehen der quantitativ orientierten Methodik war die Entwicklung der Computer-Simulation als wissenschaftliche Methode von Anfang an mit einer expliziten psychologischen Theorienbildung verbunden, die die Verwendung des Rechners als Darstellungsmittel erst zu rechtfertigen gestattet. Die allgemeine Gestalt, die diese Theorienbildung angenommen hat, haben Newell & Simon (1972) in ihrem umfangreichen Buch über menschliches Problemlösen ausführlich beschrieben: Der Mensch als Gegenstand der Psychologie wird als ein *informationsverarbeitendes System* betrachtet, als ein mit seiner Umgebung in einem primär informationellen, nicht-materiellen Austausch befindliches System, vermittelt durch komplexe, intern gesteuerte Vorgänge des Wahrnehmens, Denkens, Fühlens und Verstehens, deren externe Beobachtbarkeit nur in eingeschränkter Weise gegeben ist - ein sehr ernst zu nehmendes empirisches Problem der kognitiven Psychologie. Nach Newell & Simon (1972, S. 9-13) ist die Theorie vom Menschen als einem informationsverarbeitenden System

(1) eine *Prozeßtheorie,* die von der Annahme einer begrenzten Anzahl intern wirksamer, das extern (oder an sich selbst) beobachtbare Verhalten produzierender Prozesse ausgeht;

(2) eine Theorie des *Individuums,* in der individuelles Verhalten in spezifischen Einzelsituationen modelliert wird;
(3) eine *inhaltsorientierte* Theorie, die die in ihren Modellen konkretisierten Aufgabenstellungen (wie z.B. Problemlösen) nicht nur zu beschreiben und zu erklären, sondern auch selbst auszuführen gestattet;
(4) eine *dynamische* Theorie, in der ein Verhaltensablauf über der Zeit für jeden Handlungsschritt als eine Funktion des unmittelbar vorangehenden Zustands des Systems und seiner Umgebung dargestellt werden kann;
(5) eine *empirische, nicht-experimentelle* Theorie dahingehend, daß einerseits so viele Daten wie nur möglich über die individuell verfügbare und tatsächlich verarbeitete Information benötigt werden, andererseits diese Daten jedoch nicht in dem reduzierten Bedingungsgefüge der herkömmlichen Experimentalpraxis zu gewinnen sind;
(6) eine *nicht-statistische* Theorie, die - bis heute jedenfalls -von dem Apparat der Inferenzstatistik wenig Gebrauch machen kann, da die Daten und „Parameter" der Modelle primär nicht-numerisch, qualitativ sind;
(7) aber eine *hinreichende* Theorie in ihrer Fähigkeit, die kognitiven Phänomene, die sie untersucht, nicht nur am Menschen entdecken und beschreiben, sondern sie sogar in einem künstlichen System reproduzieren zu können.

Ob die psychologische Rahmentheorie, die mit der Computer-Simulation einhergeht, diese Merkmale aufweisen soll oder nicht, oder ob sie nicht eine ganz andere Charakterisierung erfahren sollte, kann und wird noch weiter diskutiert werden. Unabweisbar ist jedoch die Forderung, daß eine sinnvolle Verwendung der Computer-Simulation ohne eine begleitende psychologische Theorienbildung nicht wünschenswert ist.

3.1 Empirische Grundlagen psychologischer Simulationsmodelle

3.1.1 Methoden der Datengewinnung

Ausgangspunkt einer „inhaltsorientierten dynamischen Prozeßtheorie des Individuums" - so die Charakterisierung in der Terminologie von Newell & Simon - ist eine empirische Datengewinnung, die die Eigenart und Vielfalt menschlicher Informationsverarbeitung widerzuspiegeln zumindest annäherungsweise gestattet. Wenig geeignet hierzu ist eine experimentalpsychologische Methodik, die vornehmlich an der Beobachtung isolierbarer Reaktionsweisen, an dem extern beobachtbaren Ergebnis von - möglicherweise sehr komplexen - internen Vorgängen orientiert ist, ohne sich ernsthaft die Frage zu stellen, ob nicht diese internen Prozesse ebenso untersuchenswert sind wie deren externe Resultate, auch wenn sie dem Experiment weniger zugänglich erscheinen. Nicht eine nur ergebnisorientierte, sondern eine mehr prozeßgeleitete empirische Datengewinnung kann den Zugang zu den psychologischen Phänomenen der menschlichen Informationsverarbeitung eröffnen.

Als Methode der Wahl hat die kognitive Psychologie auf die schon von Wundt eingeführte, heute jedoch in liberalisierter Form gehandhabte „Introspektion“ zurückgegriffen, jene Form selbstexplorativen Verhaltens, die dem Behaviorismus stets wissenschaftlich verdächtig gewesen ist. Diese liberalisierte Variante der Introspektion ist die sog. *Methode der „lauten Denkens“,* des spontanen, sich frei entwickelnden Verbalisierens von Inhalten und Vorgängen des Bewußtseins beim Denken, Lernen, Wahrnehmen, Verstehen, Handeln usw. Typischerweise sieht eine derartige Datengewinnung so aus, daß einem Probanden eine Aufgabenstellung, sei es in einer Versuchssituation oder in einer alltagsnahen Umgebung, vorgegeben ist, deren Lösung er nicht still in seinem Kopf, sondern durch ein begleitendes „lautes Denken“ Schritt für Schritt entwickelt, wobei sein externalisiertes Verhalten einschließlich seines Verbalisierungsverhaltens auf Videoband oder Tonband aufgezeichnet wird. Ergebnis ist ein „Protokoll“, das - ganz im Sinne von Punkt (4) der Charakterisierung von Newell & Simon - den „Verhaltensablauf über der Zeit für jeden Handlungsschritt als eine Funktion des unmittelbar vorangehenden Zustands des Systems und seiner Umgebung“ beinhaltet, wenn auch noch ganz im Sinne von nichtanalysierten „Rohdaten“.

Als Beispiel für ein derartiges „Rohdatenprotokoll“ ist in Tabelle 1 das Verbalisierungsprotokoll eines Probanden wiedergegeben, der Intelligenztestaufgaben vom Typ des „Unpassenden Streichens“ mit der Methode des „lauten Denkens“ zu lösen hatte: Gegeben seien die Buchstabengruppen

AABC ACAD ACSH AACG;

welches ist die unpassende Buchstabengruppe, die nicht zu den anderen paßt?

Tabelle 1: Verbalisierungsprotokoll eines Probanden beim Lösen einer Aufgabe des „Unpassenden Streichens“.

1 „Also, wir haben vier Buchstabengruppen hier:
2 AABC, ACAD, ACSH und AACG.
3 Woll'n zuerst mal seh'n, welche von denen gleiche Buchstaben drin haben:
4 AABC hat zwei A's,
5 ACAD auch,
6 ACSH nicht;
7 aha!
8 Aber AACG wieder.
9 Also ist ACSH die unpassende Gruppe, paßt nicht zu den andern.“

Das Verbalisierungsprotokoll in Tabelle 1 wurde der Übersichtlichkeit halber schon in zusammengehörige Segmente gegliedert und zur leichteren Orientierung mit einer Zeilennumerierung versehen. Im Grunde genommen ist diese

Segmentierung - und sogar die Übertragung des Tonbandprotokolls in die schriftliche Form mit entsprechender Interpunktion - schon ein erster Schritt der Datenauswertung (auf deren Aspekte im nächsten Abschnitt eingegangen werden wird). Die 9 Protokollzeilen geben zwar sicher kein vollständiges Bild der in dem Probanden insgesamt ablaufenden Prozesse des Problemlösens, sind aber doch informativ genug, um sich eine Vorstellung von dem Lösungsprozeß derartiger Aufgaben zu machen - eine bessere Vorstellung jedenfalls, als wenn man von dem Probanden nur die Angabe „ACSH“ in Protokollzeile 9 bekäme, wie dies bei der üblichen Durchführung von Intelligenztests der Fall ist.

Neben der speziellen Verbalisierungsmethode des „lauten Denkens“ sind natürlich auch alle anderen Methoden der Datengewinnung verwendbar, die eine Aufzeichnung von Sprachverhalten - als unmittelbarer Ausdruck von Vorgängen der Informationsverarbeitung - ermöglichen. Dazu zählen beispielsweise Befragungstechniken wie das freie oder das standardisierte Interview, Gruppendiskussionen in Problemlöse- und Entscheidungssituationen, Interaktions- und Gesprächsverläufe von therapeutischen Sitzungen, ja selbst das Geschehen in Encounter-Gruppen könnte die empirische Grundlage für ein Simulationsmodell von „selbstexplorativen Gruppenvorgängen“ - so das hier zugrunde zu legende theoretische Konzept - liefern.

Darüber hinaus können durchaus auch Datenquellen der herkömmlichen Experimentalforschung herangezogen werden, sofern sie nicht nur ergebnisorientiert sind, sondern auch die Protokollierbarkeit von Prozeßabläufen beinhalten. Ein Beispiel ist die *Blickbewegungsregistrierung* einer Versuchsperson in Experimenten, die ein großflächig projizierbares Bildmaterial - wie z.B. Matrizenaufgaben aus Intelligenztests oder Schachpositionen - verwenden. Die Aufzeichnung der Blickbewegungen erfolgt mit einer speziell dazu konstruierten „Eye Marker“-Kamera, meist noch verbunden mit einer Tonbandaufzeichnung des Verbalisierungsverhaltens der Versuchsperson (vgl. Newell & Simon, 1972).

3.1.2 Möglichkeiten der Datenauswertung

In den meisten Fällen dient die empirische Datenerhebung der vorbereitenden Phase der Erstellung eines Simulationsmodells; aber auch in der abschließenden Phase der Modellprüfung ist auf die empirische Basis zurückzugreifen, um über die Validität des Modells etwas aussagen zu können (vgl. Abschnitt 4, Validierung und Anwendbarkeit von Simulationsmodellen). Die Phase der Modellerstellung auf der Grundlage empirischer Daten ist ein komplizierter - und bei den Datenmengen von Verbalisierungsprotokollen ein aufwendiger - Vorgang der Datenauswertung, der ohne ein theoriegeleitetes Arbeiten am

Material kaum durchführbar ist. Eine rein empiristische, allenfalls durch ad-hoc-Hypothesen angereicherte Datenauswertung ist bei der Vielfalt und dem Reichtum von Prozeßdaten schnell zum Scheitern verurteilt, da selbst das umfangreichste Datenmaterial hinsichtlich der Vorgänge, deren Abbild es ist, so lückenhaft sein kann, daß ohne eine theoriegestützte Ergänzung ein vollständiges Bild des Prozeßgeschehens nicht erreichbar ist.

Die Auswertung von Verbalisierungsprotokollen, für die es auch schon computergestützte Verfahren gibt (vgl. Simon, 1979), steht im Vordergrund der psychologischen Datenanalyse bei der Erstellung eines Simulationsmodells. Ziel der *sog. Protokollanalyse* ist die detaillierte Aufschlüsselung des den Verbalisierungsdaten zugrundeliegenden Prozesses der Informationsverarbeitung, zu dessen Rekonstruktion das Simulationsmodell erstellt werden soll. Die *psychologische Rahmentheorie,* in deren Kontext die Protokollanalyse ihre Grundlage hat, läßt sich - in Verallgemeinerung der am Beispiel des Problemlösens entwickelten Theorie von Newell & Simon (1972, Kap. 14) - auf zwei Annahmen über die allgemeine Natur der menschlichen Informationsverarbeitung aufbauen:

(1) Zu jedem Zeitpunkt befindet sich das informationsverarbeitende System (Mensch, Computer) in einem bestimmten *Kenntniszustand,* dessen interne Darstellung die Form von Zeichen und Zeichenstrukturen („Symbols", „symbol structures") hat, die externe Gegebenheiten in der Umgebung des Systems (Dinge, Ereignisse, Vorgänge) oder interne, das System selbst betreffende Sachverhalte (Wahrnehmungen, Gedanken, Erinnerungen, Empfindungen, Stimmungen - wenn auch in dieser Terminologie nicht so sehr auf einen Computer zutreffend!) abbilden.

(2) Jeder Kenntniszustand wird durch die geeignete Anwendung eines bestimmten *kognitiven Operators* in einen anderen Kenntniszustand überführt, so daß das gesamte Geschehen in einem informationsverarbeitenden System aus der Abfolge der einzelnen Operationen vollständig beschrieben werden kann; jeder kognitive Operator ist durch die Angabe seiner als Eingabe dienenden und seiner als Ausgabe resultierenden Kenntniszustände definierbar. Die *Elementaroperationen* kognitiver Operatoren sind im einzelnen (vgl. auch Newell & Simon, 1972, S. 29-30; von den Autoren als „elementare Informationsprozesse" bezeichnet):

(a) Aufnehmen von Information aus der Systemumgebung („Sinneswahrnehmung") bzw. aus dem Systeminnern („Befindlichkeiten") und deren interne Repräsentation als Zeichen und/oder Zeichenstrukturen;
(b) Abgeben von Information an geeignete Effektororgane des Systems (Sprechen, Schreiben, manuelle Tätigkeiten, Körperfunktionen im Bereich der menschlichen Informationsverarbeitung);
(c) Speichern von Information in verschiedenen Speichermedien (Kurz- und

Langzeitgedächtnis, ggf. auch ,,sensorische Speicher“ der Sinnessysteme des Menschen);
(d) Erkennen von Information als im Arbeitsspeicher (,,Kurzzeitgedächtnis“) repräsentierte Zeichen und Zeichenstrukturen;
(e) Vergleichen von Information hinsichtlich Gleichheit/Ähnlichkeit/Verschiedenheit von Zeichen und Zeichenstrukturen;
(f) Erzeugen von Information durch Zusammensetzen von Zeichenstrukturen aus einzelnen Zeichen oder Teilstrukturen bzw. durch Zerlegen von Zeichenstrukturen in ihre Bestandteile (Zeichen, Teilstrukturen);
(g) Löschen von für die weitere Verarbeitung nicht mehr benötigter Information in den verschiedenen Speichermedien.

Wie diese rahmentheoretischen Vorstellungen über die Grundlagen eines informationsverarbeitenden Systems die Protokollanalyse von Verbalisierungsdaten anzuleiten gestatten, sei am Beispiel des in Tabelle 1 wiedergegebenen Probandenprotokolls illustriert. Eine Analyse dieses Protokolls hat für jede verbalisierte Äußerung zu zeigen, von welchem Kenntniszustand ausgehend ein nachfolgender Kenntniszustand durch die Anwendung ganz bestimmter kognitiver Operatoren erzeugt worden sein kann. Das Ergebnis einer derartigen Analyse ist zunächst noch rein hypothetisch und erst die nachfolgende Erstellung eines entsprechenden Simulationsmodells kann die Angemessenheit der Datenauswertung sichtbar machen. Bei genauerer Durchsicht des Verbalisierungsprotokolls von Tabelle 1 kann man annehmen, daß in den einzelnen Kenntniszuständen des Probanden Konzepte wie ,,Buchstabengruppe“, ,,Unpassende Gruppe“ und das Lösungskonzept ,,Gleiche Buchstaben“ eine Rolle gespielt haben. Als Operatoren mögen dem Probanden kognitive Operationen wie ,,Lesen“ (von Buchstabengruppen), „Suchen“ (nach einem Lösungskonzept), ,,Verwenden“ (des gefundenen Lösungskonzeptes), ,,Merken“ (der unpassenden Gruppe) und ,,Beantworten“ (der Aufgabenstellung) zur konkreten Gestaltung seines Problemslöseprozesses zur Verfügung gestanden haben. Operationen also, die sich teils als Elementaroperationen und teils aus solchen zusammengesetzt interpretieren lassen. Die Rekonstruktion des Gesamtprozesses der Informationsverarbeitung am Beispiel dieser Intelligenztestaufgabe ist in Tabelle 2 wiedergegeben, aus der nicht nur die genaue Bedeutung der oben angeführten Konzepte und Operationen ersichtlich wird, sondern auch deren jeweilige Zuordnung zu den entsprechenden Ausschnitten aus dem Verbalisierungsprotokoll des Probanden. Bemerkenswert ist hierbei, daß selbst ein so klares und schlüssiges wie das hier mitgeteilte Protokoll ,,Verbalisierungslücken“ aufweist (vgl. die Leerstellen im Protokollteil von Tabelle 2), die sowohl ,,entdeckt“ als auch ,,geschlossen“ werden können nur durch die begleitenden rahmentheoretischen Vorstellungen über die Stringenz eines zielführenden Prozesses der Informationsverarbeitung. Deren Begründbarkeit kann dann allerdings erst die nachfolgende Erstellung eines Simulationsmodells liefern.

Die Notation in Tabelle 2 wurde bereits so gewählt, daß eine eventuelle Programmierung in LOGO erleichtert wird: Die Operatoren wären als Funktionen zu definieren und die in die Kenntniszustände eingehenden Konzepte als Variablen, deren Werte Literale sind (in diesem Fall LOGO-Wörter und -Sätze). Zu beachten ist, daß alle Ein- und Ausgaben der Operatoren sich auf Inhalte des sog. Arbeitsspeichers („Kurzzeitgedächtnis") beziehen, so daß in manchen Fällen die Eingabe (bei dem LESE-Operator) und gegebenenfalls auch die Ausgabe leer sein kann. Die ,,Programmlogik" der Informationsverarbeitung an diesem Beispiel einer Intelligenztestaufgabe ist aus der Darstellung von Tabelle 2 klar erkennbar: Nach dem Einlesen der Buchstabengruppen wird zunächst nach einem Lösungskonzept (,,GLEICHE BUCHSTABEN") gesucht. Danach wird dieses (mittels des Operators VERWENDE :LÖSUNGSKONZEPT:) auf die einzelnen Buchstabengruppen angewendet, wobei jede Buchstabengruppe erneut eingelesen wird (vgl. LESE :BUCHSTABENGRUPPE:, wofür es in dem Verbalisierungsprotokoll keine Hinweise gibt; hier könnte eine Blickbewegungsregistrierung die Verbalisierungslücken überbrücken helfen). Wie ersichtlich wird, ist nur auf die Buchstabengruppe ,,ACSH" das Lösungskonzept nicht zutreffend; also ist diese die :UNPASSENDE GRUPPE:, die am Ende als Beantwortung der Aufgabenstellung auch ausgegeben wird.

Die tatsächliche Erstellung eines Simulationsmodells für Aufgaben des ,,Unpassenden Streichens" müßte die Stringenz des oben gezeigten Programmablaufs deutlich zum Ausdruck bringen; insbesondere sollte das Modell auch eine hinreichende Begründung für das Ausfüllen von Verbalisierungslücken liefern, wie dies für das obige Beispiel vorgenommen wurde.

Andere Methoden der Analyse von Verbaldaten - wie z.B. die in Soziologie und Politologie verwendete Methode der ,,Inhaltsanalyse" zur Auswertung von Textmaterial hinsichtlich quantitativ-statistischer Zusammenhänge (Themen- und Worthäufigkeiten und Korrelationen darüber) - spielen für die Computer-Simulation in der Psychologie nur eine untergeordnete Rolle.

Von zunehmender Bedeutung ist dagegen die Kombination von Auswertungsmethoden wie beispielsweise die mit der Protokollanalyse von Verbalisierungsdaten verknüpfbare Auswertung von Blickbewegungsdaten. Nicht nur, daß hierbei die Wahrnehmungskomponente stärker in die Modellierung der menschlichen Informationsverarbeitung einbezogen werden kann, ist der Vorteil dieser Methodenkombination, sondern auch, daß damit gezeigt werden kann, inwieweit die ,,verbalisierte" Information mit der ,,visualisierten" kongruent geht, oder ob - was zu vermuten wäre - letztere nicht vielmehr ersteren vorauseilt. Das in Tabelle 2 wiedergegebene Beispiel einer Protokollanalyse wäre - wie schon angedeutet - empirisch mit einer Analyse von Blickbewegungsdaten sicher leichter abzusichern. - Eine noch ,,ganzheitlichere" Darstellung von Prozessen der Informationsverarbeitung ließe sich

Tabelle 2: Protokollanalyse der Verbalisierungsdaten eines Probanden beim Lösen einer Aufgabe des „Unpassenden Streichens“.

Operatoren (OP) und deren Eingaben (E) und Ausgaben (A)	Protokollausschnitt
OP (LESE :BUCHSTABENGRUPPEN:) E A (:BUCHSTABENGRUPPEN: „AABC ACAD ACSH AACG“)	„Also, wir haben vier Buchstaben-gruppen hier: AABC, ACAD, ACSH und AACG.“
OP (SUCHE :LÖSUNGSKONZEPT:) E (:BUCHSTABENGRUPPEN: „AABC ACAD ACSH AACG“) A (:LÖSUNGSKONZEPT: „GLEICHE BUCHSTABEN“)	„Woll'n zuerst mal seh'n, welche von denen gleiche Buch-staben drin haben:”
OP (LESE :BUCHSTABENGRUPPE:) E A (:BUCHSTABENGRUPPE: „AABC“)	
OP (VERWENDE :LÖSUNGSKONZEPT:) E (:BUCHSTABENGRUPPE: „AABC“) A (:GLEICHE BUCHSTABEN: „JA, ZWEI A“)	„AABC hat zwei A's,”
OP (LESE :BUCHSTABENGRUPPE:) E A (:BUCHSTABENGRUPPE: „ACAD“)	
OP (VERWENDE :LÖSUNGSKONZEPT:) E (:BUCHSTABENGRUPPE: „ACAD“) A (:GLEICHE BUCHSTABEN: „JA, ZWEI A“)	„ACAD auch,“
OP (LESE :BUCHSTABENGRUPPE:) E A (:BUCHSTABENGRUPPE: „ACSH“)	
OP (VERWENDE :LÖSUNGSKONZEPT:) E (:BUCHSTABENGRUPPE: „ACSH“) A (:GLEICHE BUCHSTABEN: „NEIN, KEINE“)	„ACSH nicht;“
OP (MERKE :BUCHSTABENGRUPPE:) E (:BUCHSTABENGRUPPE: „ACSH“) (:GLEICHE BUCHSTABEN: „NEIN, KEINE“) A (:UNPASSENDE GRUPPE: „ACSH“)	„aha!“

Tabelle 2: Fortsetzung

Operatoren (OP) und deren Eingaben (E) und Ausgaben (A)	Protokollausschnitt
OP (LESE :BUCHSTABENGRUPPE:) E A (:BUCHSTABENGRUPPE: ,,AACG“)	
OP (VERWENDE :LÖSUNGSKONZEPT:) E (:BUCHSTABENGRUPPE: ,,AACG“) A (:GLEICHE BUCHSTABEN: ,,JA, ZWEI A“)	,,Aber AACG wieder.“
OP (BEANTWORTE :AUFGABENSTELLUNG:) E (:UNPASSENDE GRUPPE: ,,ACSH“) A (:UNPASSENDE GRUPPE: ,,ACSH“)	,,Also ist ACSH die unpassende Gruppe, paßt nicht zu den andern. “

erzielen, könnte man das gesamte, beispielsweise auf Videoband aufgezeichnete Verhalten einer Person in die Datenauswertung einbeziehen: Verbalverhalten wäre sinnvoll durch nichtverbales Verhalten ergänzt. Allerdings - die Kategorien und Verfahren für eine derartige ,,ganzheitliche“ Datenauswertung sind erst noch zu entwickeln.

3.2 Informationelle Produktionssysteme

Wesentliches Merkmal der in Verbindung mit der Computer-Simulation entwickelten psychologischen Theorienbildung ist die Darstellungsbreite, mit der der jeweilige Gegenstandsbereich abgebildet wird. Vor allem auffallend ist der Versuch, die Prozeßtheorie der menschlichen Informationsverarbeitung in ihren unterschiedlichen Anwendungsbereichen mit Strukturtheorien des Gedächtnisses zu verknüpfen (vgl. Wender, Colonius & Schulze, 1980), wie es beispielsweise die neueren Theorien zum Sprachverstehen von Anderson & Bower (1973), Norman, Rumelhart & LNR (1975), Anderson (1976) und Schank & Abelson (I977), aber auch die ,,Vorläufer-Theorie“ zum menschlichen Problemlösen von Newell & Simon (1972) zeigen.

Die in ihrer Bedeutung wohl umfassendste Systemarchitektur des menschlichen Gedächtnisses und seiner Informationsverarbeitung baut auf der von Newell & Simon (1972) entwickelten Konzeption *,,informationeller* (oder kognitiver) *Produktionssysteme“* auf (vgl. Hunt & Poltrock, 1974: Ueckert, 1980a). Von ihrer Verwendungsweise her betrachtet sind informationelle Produktionssysteme die ,,Assembler-Sprache“ der menschlichen Informationsver-

arbeitung, in der - sollte die Theorie sich als empirisch zutreffend durchsetzen - der „informationelle Kode“ kognitiver Aktivität geschrieben ist, der dann in Form von konkreten Simulationsmodellen auf dem Rechner nachgebildet werden kann. Mit diesem Anspruch hat die Produktionssystem-Konzeption nicht nur in die Computer-Simulation kognitiver Prozesse Eingang gefunden, sondern auch in die „künstliche Intelligenz“-Forschung (vgl. beispielsweise den Sammelband von Waterman & Hayes-Roth, 1978).

3.2.1 Die Modellarchitektur von Produktionssystemen

Grundlegendes Konzept der Modellarchitektur von Produktionssystemen ist der Begriff der *Produktionsregel* (oder kurz: Produktion). Eine Produktionsregel beschreibt den Sachverhalt, unter welchen *Konditionen K* (d.h. bei welchem gegebenem Kenntniszustand des informationsverarbeitenden Systems) welche *Aktionen A* (d.h. welche kognitiven Operationen) ausgeführt werden sollen, um einen neuen Kenntniszustand des Systems zu erreichen.

Ein einfaches Beispiel sind die beiden Produktionsregeln:

FA_1 „Ampel ist rot“ $\rightarrow$ Warten, Ampel beobachten
FA_2 „Ampel ist grün“ $\rightarrow$ Gehen

Formal hat eine Produktionsregel stets die Struktur

$$N\ K \rightarrow A,$$

wobei K den Konditionalteil, A den Aktionsteil und der „Übergangspfeil“ $\rightarrow$ die Kopplung von A an K bezeichnet; N ist der „Name“ der Produktionsregel.

Ein *Produktionssystem* ist dann eine Menge (Liste) von Produktionsregeln, die in bestimmter Weise abgearbeitet werden. Obiges Beispiel der beiden Produktionen FA_1 und FA_2 kann man als Produktionssystem für das Verhalten an einem Fußgängerüberweg mit Ampelregelung ansehen, in dem aufgrund bestimmter Gegebenheiten („Ampel ist rot“ oder „grün“) bestimmte Handlungen (Warten und Ampel beobachten oder Gehen) ausgeführt werden (wenn auch in diesem Falle keine „kognitiven Operationen“ vorliegen, sondern motorische Aktivitäten, deren „Programmierung“ man sich jedoch als entsprechende Produktionssysteme vorstellen kann).

Zur Modellarchitektur von informationellen Produktionssystemen gehören - sowohl in ihrer Realisierung im Menschen als auch für deren Simulation auf einem Rechner - die folgenden drei *Systemkomponenten:*

(1) Ein oder mehrere *Arbeitsspeicher,* in denen alle augenblicklich verfügbare Information - aus welchen Quellen auch immer - kurzzeitig gespeichert

wird; psychologisch betrachtet sind der oder die Arbeitsspeicher das menschliche Kurzzeitgedächtnis und die unterschiedlichen „sensorischen Speicher".

(2) Ein *Produktionsspeicher,* in dem die zu Produktionssystemen zusammengefaßten Produktionsregeln langfristig verfügbar sind und nach den jeweiligen Anforderungen der Informationsverarbeitung aktiviert, aber auch modifiziert und gelöscht werden können; die psychologische Instanz für den Produktionsspeicher ist das menschliche Langzeitgedächtnis.

(3) Ein *Interpreter,* der in der Lage ist, sowohl die Inhalte der Arbeitsspeicher (d.h. deren „Daten") als auch die des Produktionsspeichers (d.h. dessen „Regeln") zu „lesen" und entsprechend dem jeweils aktivierten Produktionssystem zu „handeln". Psychologisch gesehen ist der Interpreter von Produktionssystemen der „zentrale Prozessor" oder die „kognitive Exekutive" und kann somit durchaus als Konstrukt für das menschliche Bewußtsein verstanden werden (vgl. Ueckert, 1980b): Hauptmerkmal unseres Bewußtseins ist die Fähigkeit, die Aufmerksamkeitsverteilung über die Bewußtseinsinhalte so zu regeln, daß ein zielgerichtetes Verhalten des Gesamtsystems sowohl intern (in der Informationsverarbeitung selbst) als auch extern (in dem von außen beobachtbaren individuellen Handeln) resultiert.

3.2.2 Beispiel eines Produktionssystems als Simulationsmodell

Als Einführung in Konzeption und Arbeitsweise von informationellen Produktionssystemen sei im folgenden ein Produktionssystem vorgestellt, das als ein Simulationsmodell für die Aufgabe des „Unpassenden Streichens" („Single Letter Exclusion") angesehen werden kann, wobei auf die Verbalisierungsdaten zu dieser Aufgabe (vgl. Tabelle 1) sowie auf deren Protokollanalyse (vgl. Tabelle 2) für eine Diskussion des Modells zurückgegriffen werden kann. Wünschenswert ist eine Modellentwicklung, in der der konkrete Ablauf der Informationsverarbeitung so detailliert verfolgt werden kann, daß ein direkter Vergleich mit den Verbalisierungsdaten und der Protokollanalyse möglich ist.

Das Modellbeispiel ist - wie die bisherigen Programmbeispiele - in der Notation der Programmiersprache LOGO formuliert, so daß eine Übertragung in ein lauffähiges Computer-Programm unmittelbar gegeben ist. Zur Realisierung der Modellarchitektur wird als Arbeitsspeicher auf den Variablenspeicher („Namensspeicher") von LOGO zurückgegriffen, während als Produktionsspeicher das von LOGO mit der ‚get'-Anweisung aktivierbare Datei-System (langfristiges Speichersystem) verwendet wird. Der Interpreter ist als ein sequentielles LOGO-Programm konzipiert. Die Darstellung der Produktionsregeln wird einheitlich in der Form

N
K
→ A

durchgeführt (wobei N den Namen, K den Konditionalteil und A den Aktionsteil der Produktionsregel bezeichnet).

In Abb. 20 sind die für das Produktionssystem ,,Single Letter Exclusion“ benötigten Produktionsregeln wiedergegeben. Bevor auf sie inhaltlich eingegangen werden kann, sollen die neu vorkommenden LOGO-Ausdrücke kurz erläutert werden: Die LOGO-Operation ‚request‘ ist eine Anfrage an den Benutzer (am Terminal durch das Ausdrucken eines Sterns * angezeigt), dem Programm eine Eingabe einzutippen. Der Ausdruck ‚thing‘ ist eine LOGO-Operation, die den Wert (,,das Ding“) einer Variable liefert (normalerweise wird diese Operation nicht benötigt, da man durch Aufruf der Variable deren Wert bekommt; ist jedoch dieser Wert selbst wieder eine Variable, so kann man zu deren Wert mit der Operation ‚thing‘ zugreifen). Der Ausdruck ‚both‘ ist die logische Und-Funktion (Konjunktion), d.h. ‚both‘ liefert den Wert ,,true“, wenn die beiden in der Konjunktion stehenden Prädikate den Wert ,,true“ haben. Die in LOGO vorgegebene Variable :bell: hat das am Terminal vorhandene Klingelsignal als ihren Wert.

Die Programmierung der Produktionsregeln erfolgt in LOGO in Form von Funktionsdefinitionen: Man schreibt vor den Namen einer Produktionsregel die LOGO-Anweisung ‚to‘, beginnt den Konditionalteil mit dem Ausdruck ‚10 test‘ (‚test‘ ist eine alternative LOGO-Operation zu der ‚if-then-else‘-Konstruktion), ersetzt den Übergangspfeil → durch den Ausdruck ‚20 iftrue’ (‚iftrue‘ entspricht dem ‚then‘ in ‚if-then-else‘) und beendet die Funktionsdefinition mit der LOGO-Anweisung ‚end‘.

Die konkrete Erstellung von Produktionsregeln folgt ganz dem Prinzip des ,,bottom-up programming“ und ist damit ein Beispiel für das modulare Programmieren bei der Entwicklung eines Simulationsmodells (vgl. Abschnitt 2.3.4): Jede Produktionsregel ist eine selbständige Einheit, die von allen anderen Produktionsregeln unabhängig ist (d.h. Produktionen können sich wechselseitig nicht aufrufen). Das ,,bottom-up programming“ wird wesentlich erleichtert, wenn empirische Daten (wie im vorliegenden Fall beispielsweise ein Verbalisierungsprotokoll und dessen Analyse) gegeben sind, die die Formulierung einzelner Produktionsregeln anzuleiten gestatten. Die in Abb. 20 wiedergegebenen Produktionsregeln zum ,,Single Letter Exclusion“ sind so auch leichter zu verstehen, wenn sie in direktem Vergleich zu der in Tabelle 2 dargestellen Protokollanalyse gelesen werden. Die ersten vier Produktionen entsprechen ziemlich genau den ersten vier Abschnitten der Protokollanalyse; sie beschreiben das Einlesen von Buchstabengruppen (SILEX1 bzw. SILEX3), die Suche nach einem Lösungskonzept (SILEX2) und dessen Anwendung auf

```
SILEX1
:GIVEN.ITEMS: = :empty:
→ make ,,GIVEN.ITEMS“ request ,
   get LETTER CONCEPTS

SILEX2
:CONCEPT: = :empty:
→ make ,,CONCEPT“ first of :LETTER.CONCEPTS:

SILEX3
:ITEM: = :empty:
→ make ,,ITEM“ request

SILEX4
ACTIVE :ITEM:
→ do :CONCEPT:

SILEX5
(first of thing of :CONCEPT:) = ,,YES“
→ make thing of ,,CONCEPT“ :empty: ,
   make „ITEM“ :empty:

SILEX6
(first of thing of :CONCEPT:) = ,,NO“
→ make ,,UNSUITABLE.ITEM“
         sentence of :ITEM: and :UNSUITABLE.ITEM: ,
   make thing of ,,CONCEPT“ :empty: ,
   make ,,ITEM“ :empty:

SILEX7
both (count of :UNSUITABLE.ITEM:) = 1 and :ITEM: = :bell:
→ print :UNSUITABLE.ITEM: ,
   make ,,HALT“ ,,PROBLEM IS SOLVED“

SILEX8
both not (count of :UNSUITABLE.ITEM:) = 1 and :ITEM: = :bell:
→ make ,,UNSUITABLE.ITEM“ :empty: ,
   make ,,LETTER.CONCEPTS“
         REMOVE :CONCEPT: from :LETTER.CONCEPTS: ,
   make ,,ITEM“ :empty:
```

Abb. 20: Produktionsregeln für die Aufgabe des ,,Unpassenden Streichens“ (,,Single Letter Exclusion“).

die jeweils zu bearbeitende Buchstabengruppe (SILEX4). Für die übrigen Produktionen läßt sich in der Protokollanalyse nicht immer ein direktes Analogon finden, doch ist ihre psychologische Plausibilität einsichtig: Produktionen SILEX5 und SILEX6 beschreiben das Problemlöseverhalten nach Anwendung

des Lösungskonzeptes und Produktionen SILEX7 und SILEX8 das entsprechende Verhalten nach Bearbeitung aller vorgegebenen Buchstabengruppen (durch das Klingelsignal :bell: angezeigt); mit SILEX7 wird das Problemlösen erfolgreich abgeschlossen, mit SILEX8 jedoch fortgesetzt, nachdem sich das gewählte Lösungskonzept als unbrauchbar erwiesen hat (d.h. der ganze Prozeß wiederholt sich mit der Suche nach einem neuen Lösungskonzept).

Zur Arbeitsweise einzelner Produktionsregeln sind einige Erläuterungen angebracht:

(1) Die LOGO-Anweisung ‚get LETTER CONCEPTS' im Aktionsteil von SILEX1 setzt das Vorhandensein einer Datei unter dem Namen LETTER CONCEPTS voraus, in der die für Buchstabenaufgaben verwendbaren Lösungskonzepte und deren Funktionsdefinitionen langfristig gespeichert sind. Im konkreten Beispiel der vorliegenden Aufgabe habe die Datei LETTER CONCEPTS den in Abb. 21 wiedergegebenen Inhalt (wobei aus Einfachheitsgründen von den beiden Lösungskonzepten nur IDENTICAL.LETTERS definiert ist; die hier in Programmzeile 10 verwendete LOGO-Anweisung ‚local' dient zum Einrichten einer lokalen Variable im Arbeitsspeicher, die nur für die Laufzeit der Funktion Gültigkeit hat).

```
:LETTER.CONCEPTS: is
    ,,IDENTICAL.LETTERS ALPHABETICAL.SEQUENCE“

to IDENTICAL.LETTERS
10 local ,,WORD“
20 make ,,WORD“ :ITEM:
30  if CONTAINS butfirst of :WORD: first of :WORD:
    then make ,,IDENTICAL.LETTERS“
              sentence of ,,YES“ and first of :WORD: , stop
    else make ,,WORD“ butfirst of :WORD:
40  if (count of :WORD:) = 1
    then make ,,IDENTICAL.LETTERS“ ,,NO ONE“ , stop
    else go to line 30
end
```

Abb. 21: Inhalt der Datei LETTER CONCEPTS.

(2) Die LOGO-Operation ‚first of :LETTER.CONCEPTS:' im Aktionsteil von SILEX2 wird ermöglicht, nachdem mit SILEX1 die Datei LETTER CONCEPTS aktiviert worden ist.

(3) Die LOGO-Anweisung ‚do :CONCEPT:' im Aktionsteil von SILEX4 beinhaltet die Ausführung des Lösungskonzeptes als eine Programmfunktion, im vorliegenden Fall also die Ausführung von IDENTICAL.LETTERS (vgl. Abb. 21).

(4) Die Abfrage ,first of thing of :CONCEPT:‘ im Konditionalteil von SILEX5 und SILEX6 bezieht sich auf das Ergebnis der Funktionsausführung des Lösungskonzeptes, im Beispiel also auf das Ergebnis der Programmfunktion IDENTICAL.LETTERS (vgl. deren Programmzeilen 30 bzw. 40 in Abb. 21).

(5) Das Abarbeiten der einzelnen Buchstabengruppen wird vom Benutzer durch die Eingabe des Klingelsignals *BELL (in SILEX3) abgeschlossen und im Konditionalteil von SILEX7 bzw. SILEX8 von dem Programm mit der Abfrage ,:ITEM: = :bell:‘ erkannt, worauf je nach dem Ergebnis von ,(count of :UNSUITABLE.ITEM:) = 1‘ entweder SILEX7 oder SILEX8 ,,feuert“ (so der Ausdruck in der Terminologie von Produktionssystemen).

Damit die in Abb. 20 wiedergegebenen Produktionsregeln in der erwünschten Weise arbeiten können, müssen sie zu einem Produktionssystem zusammengefaßt werden, das vom Interpreter gelesen und ausgeführt werden kann. Die Definition eines Produktionssystems ist eine relativ einfache Aufgabe, wie aus der Darstellung in Abb. 22 zu ersehen ist: Die einzelnen Produktionsregeln werden lediglich auf einer Liste ihrem Namen nach in eine bestimmte Reihenfolge gebracht, die für die Abarbeitung durch den Interpreter von Bedeutung ist.

```
to SINGLE.LETTER.EXCLUSION
 10 make ,,PRODUCTION.LIST“
           ,,SILEX5 SILEX6 SILEX7 SILEX8 SILEX1 SILEX2 SILEX3 SILEX4“
end
```

Abb. 22: Definition des Produktionssystems für die Aufgabe des ,,Unpassenden Streichens“.

Der Interpreter selbst ist ein einfaches sequentielles LOGO-Programm mit einigen Unterprogrammen, dargestellt in den Abb. 23 und 24.

Die Arbeitsweise des Interpreters läßt sich wie folgt beschreiben:

(1) Mit RUN :PRODUCTION.SYSTEM: wird der Programmlauf eines Produktionssystems gestartet, im vorliegenden Fall beispielsweise mit RUN ,,SINGLE.LETTER.EXCLUSION“.

(2) In Programmzeile 10 von RUN wird durch die ,do‘-Anweisung die Funktionsdefinition des Produktionssystems ausgeführt, was nichts anderes besagt, als daß im Arbeitsspeicher die entsprechende Produktionsliste aktiviert wird (vgl. die Funktionsdefinition von SINGLE.LETTER.EXCLUSION in Abb. 22).

```
to RUN :PRODUCTION.SYSTEM:
10 do :PRODUCTION.SYSTEM:
20 PROCESS :PRODUCTION.LIST:
30 if ACTIVE :HALT:
     then stop
     else go to line 20
end
```

Abb. 23: Hauptprogramm des Interpreters für Produktionssysteme.

```
to PROCESS :PRODUCTIONS:
10 if :PRODUCTIONS: = :empty:
     then make ,,HALT" ,,NO PRODUCTIONS READY" , stop
20   if READY first of :PRODUCTIONS:
     then FIRE first of :PRODUCTIONS:
     else PROCESS butfirst of :PRODUCTIONS:
end

to READY :PRODUCTION:
10   do butfirst of text of :PRODUCTION: 10
20 iftrue output ,,true"
30 iffalse output ,,false"
end

to FIRE :PRODUCTION:
10   do butfirst of text of :PRODUCTION: 20
end

to ACTIVE :NAME:
10 if :NAME: = :empty:
     then output ,,false"
     else output ,,true"
end

to   ,
end
```

Abb. 24: Unter- und Hilfsprogramme des Interpreters.

(3) Programmzeile 20 setzt den Prozeß des Abarbeitens der Produktionsliste in Gang; dieser Prozeß ist - wie aus dessen Funktionsdefinition in Abb. 24 hervorgeht - ein rekursiver Vorgang des Suchens nach der *ersten* ausführbaren Produktion (per READY getestet und per FIRE ausgeführt, wobei mittels

der LOGO-Operation ‚text‘ auf die jeweilige Programmzeile der entsprechenden Produktionsregel zugegriffen wird). Der Interpreter folgt dabei dem Dominanzprinzip der Regelabarbeitung, weshalb die Reihenfolge der Produktionsregeln in der Produktionsliste von Bedeutung ist, um Konflikte bei der Ausführung von Produktionen zu vermeiden, wenn zu einem bestimmten Zeitpunkt der Konditionalteil mehrerer Produktionsregeln gleichzeitig erfüllbar ist.

(4) In Programmzeile 30 wird schließlich geprüft, ob im Arbeitsspeicher das Haltsignal - mit welchem Wert auch immer - gesetzt ist, worauf im positiven Fall die Interpretation des Produktionssystems abgeschlossen ist (‚stop‘), im negativen Fall jedoch so lange fortgesetzt wird, bis das Haltsignal erscheint (entweder über den Aktionsteil einer Produktionsregel wie in SILEX7 oder über Programmzeile 10 von PROCESS, wenn keine der Produktionen ausführbar ist).

Eine anschauliche Vorstellung von dem konkreten Ablauf der Informationsverarbeitung beim Lösen einer Aufgabe des ,,Unpassenden Streichens“ gewinnt man, wenn man sich einen Programmlauf per RUN ,,SINGLE.LETTER.EXCLUSION“ ansieht und mit den empirischen Daten vergleicht, wie dies in Tabelle 3 dargestellt ist. (Anmerkung: Am Terminal ist mit dem Programmaufruf RUN ,,SINGLE.LETTER.EXCLUSION“ tatsächlich nur das zu sehen, was in Tabelle 3 als [Eingabe] bzw. [Ausgabe] spezifiziert ist. Um einen vollständigen überblick über die zum jeweiligen Zeitpunkt ,,feuernden“ Produktionen und die damit resultierenden Arbeitsspeicherinhalte zu bekommen, ist der Interpreter um entsprechende Druckanweisungen zu erweitern: In Programmzeile 20 von RUN ist an ‚PROCESS :PRODUCTION.LIST:‘ die LOGO-Anweisung ‚list all names‘ anzuhängen, die den gesamten Arbeitsspeicherinhalt am Terminal ausdruckt, und in Programmzeile 20 von PROCESS ist vor ‚FIRE first of :PRODUCTIONS:‘ die Druckanweisung ‚print first of :PRODUCTIONS:‘ zu schreiben, um sehen zu können, welche Produktion zu diesem Zeitpunkt feuert.)

3.2.3 Transparenz und Abbildtreue von Produktionssystemen

Der Vergleich des in Tabelle 3 wiedergegebenen Programmlaufs des Produktionssystems SINGLE.LETTER.EXCLUSION mit den beigefügten Verbalisierungsdaten, aber auch deren in Tabelle 2 mitgeteilter Protokollanalyse zeigt einen Auflösungsgrad der abgebildeten Vorgänge des Problemlösens und einen Annäherungsgrad an die Realität menschlicher Informationsverarbeitung, die mit den Mitteln des herkömmlichen Programmierens, insbesondere des ,,top-down programming“, in dieser Transparenz und Abbildtreue kaum zu erreichen ist. Das läßt sich noch stärker verdeutlichen, wenn man statt des Produk-

Tabelle 3: Programmlauf des Produktionssystems SINGLE.LETTER. EXCLUSION.

PR	Arbeitsspeicher-Veränderungen durch Aktionen der Produktionsregeln (PR)		Protokollausschnitt
SILEX1	+AABC ACAD ACSH AACG :GIVEN.ITEMS: is ,,AABC ACAD ACSH AACG" :LETTER.CONCEPTS: is ,,IDENTICAL.LETTERS ALPHABETICAL.SEQUENCE"	[Eingabe]	„Also, wir haben vier Buchstabengruppen hier: AABC, ACAD, ACSH und AACG."
SILEX2	:CONCEPT: is ,,IDENTICAL.LETTERS"		„Woll'n zuerst mal seh'n, welche von denen gleiche Buchstaben drin haben:"
SILEX3	*AABC :ITEM: is ,,AABC"	[Eingabe]	,,AABC"
SILEX4	:IDENTICAL.LETTERS: is ,,YES A"		„hat zwei A's,"
SILEX5	:IDENTICAL.LETTERS: is ,," :ITEM: is ,,"		
SILEX3	*ACAD :ITEM: is ,,ACAD"	[Eingabe]	,,ACAD"
SILEX4	:IDENTICAL.LETTERS: is ,,YES A"		„auch,"
SILEX5	:IDENTICAL.LETTERS: is ,," :ITEM: is ,,"		
SILEX3	*ACSH :ITEM: is ,,ACSH"	[Eingabe]	,,ACSH"
SILEX4	:IDENTICAL.LETTERS: is ,,NO ONE"		,,nicht;"
SILEX6	:UNSUITABLE.ITEM: is ,,ACSH" :IDENTICAL.LETTERS: is ,," :ITEM: is ,,"		„aha!"
SILEX3	*AACG :ITEM: is ,,AACG"	[Eingabe]	,,Aber AACG"

Tabelle 3 : Fortsetzung

SILEX4	:IDENTICAL.LETTERS: is ,,YES A“	,,wieder.“
SILEX5	:IDENTICAL.LETTERS: is ,,“ :ITEM: is ,,“	
SILEX3	*BELL [Eingabe] :ITEM: is :bell:	
SILEX7	ACSH [Ausgabe] :HALT: is ,,PROBLEM IS SOLVED“	,,Also ist ACSH die unpassende Gruppe, paßt nicht zu den andern.“

tionssystems ein sequentielles Programm für das ,,Single Letter Exclusion“ schreibt, wie es in Abb. 25 wiedergegeben ist.

In seinem Verhalten ist das sequentielle Programm zwar identisch mit dem Produktionssystem, die Transparenz der Informationsverarbeitung, insbeson-

```
to SINGLE.LETTER.EXCLUSION
10 make ,,GIVEN.ITEMS“ request
20 get LETTER CONCEPTS
30  make ,,CONCEPT“ first of :LETTER.CONCEPTS:
40 make ,,ITEM“ request
50  if both (count of :UNSUITABLE.ITEM:) = 1 and :ITEM: = :bell:
    then print :UNSUITABLE.ITEM: , stop
60  if both not (count of :UNSUITABLE.ITEM:) = 1 and :ITEM: = :bell:
    then make ,,LETTER.CONCEPTS“
             REMOVE :CONCEPT: from :LETTER.CONCEPTS: ,
         go to line 30
70 do :CONCEPT:
80  if (first of thing of :CONCEPT:) = ,,NO“
    then make ,,UNSUITABLE.ITEM“
             sentence of :ITEM: and :UNSUITABLE.ITEM:
90 go to line 40
end
```

Abb. 25: Sequentielles LOGO-Programm für die Aufgabe des ,,Unpassenden Streichens“.

dere hinsichtlich der konkreten Arbeitsspeicher-Veränderungen, geht jedoch weitgehend verloren. Die im Programmkode noch sichtbaren Entsprechungen zwischen den Programmzeilen von Abb. 25 und den Produktionen von Abb. 20 sind im einzelnen:

- Programmzeile 10 und 20: SILEX1
- Programmzeile 30: SILEX2
- Programmzeile 40: SILEX3
- Programmzeile 50: SILEX7
- Programmzeile 60: SILEX8
- Programmzeile 70: SILEX4
- Programmzeile 80: SILEX6

Für Programmzeile 90 gibt es keine Entsprechung, ferner fallt Produktion SILEX5 ganz aus dem sequentiellen Programm heraus. Vor allem aber ist in dem Programm nicht mehr immer klar, unter welchen „Datenbedingungen“ - den im Konditionalteil von Produktionsregeln angesprochenen Arbeitsspeicherinhalten - einzelne Programmoperationen ausgeführt werden (vgl. Programmzeilen 10, 20, 30, 40 und 70), so daß die mit einem Programmlauf einhergehenden Arbeitsspeicher-Veränderungen nicht mehr so leicht nachvollziehbar sind wie bei der Abarbeitung des Produktionssystems.

Noch entscheidender sind die Mängel einer sequentiellen Programmierung hinsichtlich der Abbildtreue eines so erstellten Simulationsmodells. Die Modellarchitektur von Produktionssystemen - Arbeitsspeicher als Kurzzeitgedächtnis, Produktionsspeicher als Langzeitgedächtnis und Interpreter als „kognitive Exekutive“ - ist konstitutiv für die psychologische Relevanz eines Simulationsmodells, im Falle von sequentiellen Programmen wird aber beispielsweise über Arbeitsspeicher und Interpreter überhaupt nichts ausgesagt, obwohl auch hier für den konkreten Programmlauf beide Komponenten gegeben sind (nämlich in Form des von der verwendeten Programmiersprache zur Verfügung stehenden Arbeitsspeichers und Interpreters). Erst die Einbeziehung einer umfassenden Modellarchitektur wie die mit der Produktionssystem-Konzeption verbundenen macht es möglich, über Transparenz und Abbildtreue von Simulationsmodellen sinnvoll diskutieren zu können.

Als bemerkenswertestes Beispiel einer mit der Produktionssystem-Konzeption verknüpften psychologischen Theorienbildung sei nur auf die ACT-Theorie von Anderson (1976) verwiesen, in der der bis heute weitgehendste Versuch unternommen wurde, Strukturmodelle des menschlichen Gedächtnisses - mittels propositionaler semantischer Netze („deklaratives Wissen“ oder „Wissen Was“) - mit Prozeßmodellen der Informationsverarbeitung - mittels informationeller Produktionssysteme („prozedurales Wissen“ oder „Wissen Wie“) - zu einer einheitlichen Theorie kognitiver Aktivität im Bereich des Sprachverstehens und der Sprachproduktion zu verbinden. Trotz aller Kritik

(vgl. z.B. Wexler, 1978) verdient diese Theorie nicht nur ihres psychologischen Anspruchs, sondern auch ihres empirischen Gehaltes wegen Aufmerksamkeit, denn die Kriterien der psychologischen Relevanz und der empirischen Testbarkeit sind gerade im Zusammenhang mit der Computer-Simulation besonders schwer zu beurteilen (vgl. Abschnitt 4, Validierung und Anwendbarkeit von Simulationsmodellen).

3.3 Das Interpreterproblem von Produktionssystemen

Obwohl die Produktionssystem-Konzeption als die im Zusammenhang mit der Simulationsmethodik bisher am weitesten entwickelte psychologische Theorienbildung gelten kann, ist das Interpreterproblem, die Frage nach der Bedeutsamkeit des Interpreters als der ,,zentrale Prozessor“ oder die ,,kognitive Exekutive“ der menschlichen Informationsverarbeitung, erst in Ansätzen diskutiert worden (vgl. Ueckert, 1980b). Welchen Anforderungen ein effizienter Interpreter unter anderem genügen sollte, haben beispielsweise McDermott & Forgy (1978) ausgeführt:

- Der Interpreter sollte das informationsverarbeitende System, während es an einer bestimmten Aufgabe arbeitet, in allen Versuchen unterstützen, sensitiv für die Vielfältigkeit seiner externen Informationsquellen zu bleiben.

- Ebensosehr sollte der Interpreter das System in die Lage versetzen, sensitiv für seine eigenen, internen Tätigkeiten zu sein.

- Der Interpreter sollte fähig sein, mit ,,widersprüchlichen Daten“ umgehen zu können, insbesondere zwischen zu einem Zeitpunkt relevanter und nicht mehr relevanter (oder noch nicht relevanter) Information unterscheiden können.

- Hierbei sollte der Interpreter vor allem erkennen können, ob sich aufgrund eines gegebenen Arbeitsspeicherinhalts gleichzeitig mehrere Produktionen aktivieren lassen, und gegebenenfalls geeignete ,,Konfliktlösungsmöglichkeiten“ anbieten können.

Konkret formuliert kann das Interpreterproblem unter folgenden Fragestellungen behandelt werden:

(1) Welche Lesarten von Produktionsregeln sind von dem Interpreter realisierbar?
(2) Welche Möglichkeiten der *Konfliktlösung* gibt es für den Interpreter bei gleichzeitiger Erfüllbarkeit mehrerer Konditionalteile von Produktionsregeln?

(3) Welche Formen von *Lernfähigkeit* muß der Interpreter für eine realitätsgerechte Informationsverarbeitung aufweisen?

(4) Welche *„Bewußtseinsfunktionen“* - wenn überhaupt - sollte der Interpreter als „kognitive Exekutive“ eines informationsverarbeitenden Systems ausführen können?

Eine einführende Diskussion dieser Fragestellungen soll in den folgenden Unterabschnitten gegeben werden.

3.3.1 Lesarten von Produktionsregeln

In seiner bisher beschriebenen Funktionsweise arbeitet der Interpreter alle Produktionsregeln von links nach rechts ab (vgl. die Definition von PROCESS in Abb. 24): Er liest zuerst den linken Teil einer Produktion, den Konditionalteil, und prüft, ob dieser im Arbeitsspeicher erfüllt ist (mittels ‚do butfirst of text of :PRODUCTION: 10‘ in READY); ist dies der Fall, dann führt er den rechten Teil der Produktion aus, den Aktionsteil (mittels ‚do butfirst of text of :PRODUCTION: 20‘ in FIRE). Diese Arbeitsweise von links nach rechts wird als *datengesteuert* („data-driven“, „condition-driven“) bezeichnet, da die Daten des Arbeitsspeichers - die augenblicklichen Arbeitsspeicherinhalte - die aktuelle Interpretation des Produktionssystems bestimmen.

Es ist jedoch auch die umgekehrte Lesart realisierbar, ein Abarbeiten der Produktionsregeln von rechts nach links: Der Interpreter liest zuerst den rechten Aktionsteil einer Produktion und prüft - um die in diesem angegebenen Operationen ausführen zu können -, ob die im linken Konditionalteil dieser Produktion spezifizierten Daten im Arbeitsspeicher gegeben sind; ist dies der Fall, kann der Aktionsteil ausgeführt werden, ist dies jedoch nicht der Fall, sucht sich der Interpreter diejenige Produktionsregel, in deren Aktionsteil die Daten des ersteren, noch nicht erfüllbaren Konditionalteils erzeugt werden. Dieser Prozeß wird so lange fortgesetzt, bis ein im Arbeitsspeicher erfüllbarer Konditionalteil gefunden ist und dessen zugehöriger Aktionsteil ausgeführt werden kann. Diese Arbeitsweise des Interpreters wird als *handlungsgesteuert* („action-driven“, „goal-driven“) bezeichnet, da die angestrebten Handlungen - die Aktionen von Produktionen - die Interpretation des Produktionssystems bestimmen.

Auf eine Kurzformel gebracht, lassen sich die beiden Lesarten von Produktionsregeln so beschreiben:

(1) Datengesteuerte Interpretation:
für alle N: wenn K erfüllt ist, tue A.
(2) Handlungsgesteuerte Interpretation:
für alle N: um A tun zu können, erfülle K.

(Wobei N den Namen, K den Konditionalteil und A den Aktionsteil einer Produktion bezeichnet.)

Die psychologische Bedeutsamkeit der beiden unterschiedlichen Interpretationsweisen liegt auf der Hand. Datengesteuerte Interpretation ist überall da angebracht, wo es auf schnelle, angepaßte, situationsgerechte Informationsverarbeitung ankommt, insbesondere mit gut ausgearbeiteten oder „überlernten“ Programmen. Handlungsgesteuerte Interpretation ist dagegen „offener“; man kann sie als eine Art von „Probehandeln“ auffassen, als Denken im engeren Sinne, das beim Planen, Beweisen, Schlußfolgern, aber konkret auch beim Ausarbeiten und Verbessern von kognitiven Produktionssystemen eine Rolle spielt. Als Beispiel hierzu könnte man sich einen handlungsgesteuerten Lauf des Produktionssystems SINGLE.LETTER.EXCLUSION vorstellen, bei dem im Arbeitsspeicher lediglich der Ausdruck ‚:UNSUITABLE.ITEM: is „ACSH“‘ gegeben ist, von dem zu beweisen sei, daß er die richtige Lösung einer Aufgabe des „Unpassenden Streichens“ am Beispiel von „AABC ACAD ACSH AACG“ darstellt. Der Leser möge sich selbst vergegenwärtigen, welche Produktionen der Interpreter hier in welcher Reihenfolge zu betrachten (und gegebenenfalls auch zu feuern) hätte, um einen folgerichtigen Beweis der Richtigkeit von ‚:UNSUITABLE.ITEM: is „ACSH“ ‘ für ‚:GIVEN.ITEMS: is „AABC ACAD ACSH AACG“ ‘ vorzulegen.

Offensichtlich müßte für die handlungsgesteuerte Arbeitsweise das Hauptprogramm des Interpreters in Abb. 23 um eine entsprechend zu definierende Prozeßfunktion - etwa als TRY.PROCESS zu bezeichnen - verändert oder ergänzt werden (unter Berücksichtigung von Bedingungen, wann PROCESS und wann TRY.PROCESS aktiviert werden sollen).

3.3.2 Konfliktlösungsstrategien („conflict resolution“)

In Anbetracht der Tatsache, daß jede Produktionsregel eine in sich geschlossene Einheit darstellt („Modularität“), ist es insbesondere bei umfangreicheren Produktionssystemen möglich, daß aufgrund des Arbeitsspeicherinhalts mehrere Produktionen gleichzeitig - und das zu unterschiedlichen Zeiten immer wieder - feuern könnten. Die Theorie der Produktionssysteme verlangt jedoch, daß zu einem Zeitpunkt stets nur genau *eine* Produktion ausgeführt werden kann, wie klein oder groß auch immer der Zeittakt sein mag. Das Problem ist also, welche der Produktionen der Interpreter dann aktivieren soll.

In der Literatur sind bisher recht unterschiedliche Konfliktlösungsmöglichkeiten für diesen Fall vorgeschlagen worden. McDermott & Forgy (1978) beispielsweise diskutieren die folgenden Möglichkeiten:

(1) Reihungs-Dominanz der Produktionsregeln: Die erste Produktion innerhalb des Produktionssystems, deren Konditionalteil erfüllt ist, wird ausgeführt. (In diesem Sinne arbeitet der in Abb. 23 und 24 wiedergegebene Interpreter.)
(2) Spezialfall-Dominanz: Produktionen mit einem spezifischen Konditionalteil werden allgemeineren Produktionen vorgezogen. (Von den Produktionsregeln in Abb. 20 ist beispielsweise SILEX3 ein Spezialfall von SILEX4.)
(3) Neuheits-Dominanz („recency“): Zuletzt gefeuerte Produktionen oder solche, die zuletzt erfüllte Datenelemente in ihrem Konditionalteil aufweisen, werden bevorzugt.
(4) Unterschiedlichkeits-Dominanz („distinctiveness“): Möglichst in ihrem Konditionalteil und/oder Aktionsteil von vorangehenden Produktionen verschiedene Regeln werden vorgezogen.
(5) Zufallsauswahl in Ermangelung anderer Kriterien.

Abgesehen von der Reihungs-Dominanz und der Zufallsauswahl sind diese - und weitere, von anderen Autoren (z.B. Davis & King, 1977; Rychener & Newell, 1978) zitierte - Konfliktlösungsmöglichkeiten nicht immer eindeutig, d.h. sie führen in vielen Fällen nicht zu genau einer ausführbaren Produktion. Sinnvoll ist daher die geeignete Kombination dieser - in sich auch noch weiter unterteilbarer - Möglichkeiten zu ausgefeilten „Konfliktlösungsstrategien“, wie sie von McDermott & Forgy diskutiert werden, insbesondere auch unter dem Aspekt, in welcher Weise sie die eingangs erwähnten Anforderungen an einen effizienten Interpreter zu stützen erlauben.

3.3.3 Adaptivität (Lernfähigkeit) von Produktionssystemen

Eine der wichtigsten Aufgaben des Interpreters ist die Fähigkeit, Produktionssysteme für sich ändernde Anforderungen an eine situationsgerechte Informationsverarbeitung adaptiv zu halten. Theoretisch ist das Problem der Adaptivität oder Lernfähigkeit von Produktionssystemen leicht zu lösen: Produktionsregeln können jederzeit in einem Produktionssystem

- hinzugefügt,
- entfernt,
- generalisiert,
- spezialisiert

werden. Die Frage der praktischen Realisierbarkeit derartiger Modifikationsmöglichkeiten ist ein empirisches - oder auch nur technisches - Problem:

- Lernen durch Belehrung,
- Lernen durch Beispiele,

- Erfolgskontrolle durch Rückmeldung,
- Generalisieren durch Meta-Regeln

sind einige der in der Literatur bisher behandelten Formen der Adaptivität von Produktionssystemen (vgl. beispielsweise das Kapitel „Learning" in Waterman & Hayes-Roth, 1978).

Ein konkretes Beispiel zur Diskussion des Adaptivitätsproblems wird in Abschnitt 3.4 vorgestellt.

3.3.4 „Bewußtseinsfunktionen" des Interpreters

Die Frage, inwieweit der Interpreter von Produktionssystemen „Bewußtseinsfunktionen" ausüben sollte, wie sie für den Bereich menschlicher Kognition charakteristisch sind, ist ein bisher noch kaum diskutiertes Problem der Simulationsmethodik. Unter dem Aspekt des „zentralen Prozessors" oder der „kognitiven Exekutive" von informationsverarbeitenden Systemen kann diese Frage jedoch nicht ausgeklammert werden, und die Produktionssystem-Konzeption bietet mit ihrer Modellarchitektur den bisher brauchbarsten Ansatz zu deren Diskussion.

Nach einem Vorschlag von Ueckert (1980b) kann man zwei Grundfunktionen des menschlichen Bewußtseins unterscheiden, deren Realisierbarkeit durch den Interpreter als „kognitive Exekutive" von Produktionssystemen gegeben erscheint:

(1) Eine *Zeigerfunktion* derart, daß der Interpreter jederzeit in der Lage ist, auf den Inhalt eines beliebigen Arbeitsspeichers zu „zeigen", was im Bereich des menschlichen Bewußtseins der Fähigkeit der Aufmerksamkeitslenkung auf beliebige Bewußtseinsinhalte entspricht.

(2) Eine *Übersetzerfunktion* dahingehend, daß der Interpreter fähig ist, beliebige Arbeitsspeicherinhalte mit Hilfe geeigneter Operationen nach außen zu „übersetzen", d.h. im Sinne der menschlichen Fähigkeit zur sprachlichen und/oder manuellen Umsetzung von Bewußtseinsinhalten alles das zu externalisieren, was durch die Zeigerfunktion zu einem Zeitpunkt im Fokus der Aufmerksamkeit gehalten werden kann.

Es ist naheliegend, daß ein effizienter Interpreter, der in gleicher Weise eine datenorientierte wie handlungsorientierte Informationsverarbeitung realisieren, Konfliktlösungsmöglichkeiten für konkurrierende Produktionen bereitstellen und Formen einer situationsgerechten Adaptivität aufweisen soll, über beide „Bewußtseinsfunktionen" verfügen muß: Die Zeigerfunktion ermöglicht die unterschiedlichen Lesarten von Produktionsregeln und die Entwicklung

von Strategien der Konfliktlösung, die Übersetzerfunktion ist Voraussetzung für die Anpassungsleistungen des informationsverarbeitenden Systems in seiner Auseinandersetzung mit der Umwelt. Inwiefern diese Funktionen jedoch heute schon in einem einzigen Interpreter implementiert - d.h. in Form eines Computer-Modells programmiert - werden können, ist noch eine offene Frage; würde ihre Beantwortung doch bedeuten, auch von einem „Bewußtsein der Maschinen“ sprechen zu können.

3.4 „Künstliche Intelligenz“ oder: Wie man dem Rechner das Rechnen beibringen kann

Im Grunde genommen ist jedes Simulationsmodell, das Prozesse der Informationsverarbeitung auf dem Rechner nachbildet, eine Form von „künstlicher Intelligenz“ (KI), da die Realisierung des Modells auf einem artifiziellen System - dem Computer - erfolgt und die zugrundeliegenden Prozesse informationell - und damit, wenn man so will, „geistig“ oder „intelligent“ - sind. Von diesem weiteren Begriff der „künstlichen Intelligenz“ abzuheben ist ein engerer Begriff, der für die Entwicklung künstlicher Systeme im Bereich der Informatik charakteristisch ist: „Künstliche Intelligenz“ weist jedes Computer-Programm auf, das *Leistungen* produziert, die, wenn beim Menschen beobachtet, als „intelligent“ zu bezeichnen wären, wobei es keine Rolle spielt, ob die den Leistungen zugrundeliegenden *Vorgänge* „menschenähnlich“ sind oder nicht. Beispiele für diese Form „künstlicher Intelligenz“ gibt es zu den unterschiedlichsten Leistungsbereichen, wie jedes Buch zu diesem Forschungsgebiet belegt (als lesenswerte allgemeinverständliche Einführung vgl. das Buch von Boden, 1977).

Inzwischen hat sich auch auf diesem Gebiet die Produktionssystem-Konzeption so weit durchgesetzt, daß heute kaum noch ein neues KI-System geschrieben wird, ohne auf diesen Ansatz zu rekurrieren (vgl. Waterman & Hayes-Roth, 1978). Insbesondere die mit Produktionssystemen verbundene Flexibilität und Adaptivität hat sich als entscheidender Vorteil nicht nur in der Entwicklung von Simulationsmodellen, sondern auch in der Konstruktion von KI-Systemen herausgestellt. Dies mag an einem simplen Beispiel demonstriert werden: einem „lernfähigen“ Produktionssystem zum Addieren zweier ganzer positiver Zahlen.

Das Modell, das zunächst nicht als ein Simulationsmodell konzipiert ist - und von daher als ein KI-System im engeren Sinne zu verstehen ist -, soll die Addition auf die elementaren Operationen des Zählens (in der Programmiersprache LOGO mit der ‚count‘-Operation gegeben) und der Konkatenation (Zusammenfügen zweier Einheiten zu einer neuen Einheit; in LOGO mit der ‚word‘- bzw. der ‚sentence‘-Operation ausführbar) zurückführen. Die Adap-

tivität oder ,,Lernfähigkeit“ des Modells soll so realisiert werden, daß das Ergebnis einer Addition mit zwei vorgegebenen Zahlenwerten in einer ,,Additionstabelle“ langfristig gespeichert wird, so daß bei einer erneuten Vorgabe der beiden Zahlenwerte das Ergebnis nur noch aufgesucht und ausgegeben werden braucht. Die Flexibilität, die sich in der Produktionssystem-Konzeption dadurch ergibt, daß sich Produktionssysteme wechselseitig aufrufen können, wird für dieses KI-System auf den einfachsten Fall reduziert, den wechselseitigen Aufruf zweier Produktionssysteme: (1) ADD, in dem das eigentliche ,,Rechnen“ (durch Zählen und Konkatenation) stattfindet, und (2) ADD. TABLE, in dem die ,,Rechenergebnisse“ nach und nach gespeichert werden. Wie die Interaktion dieser beiden Produktionssysteme bewerkstelligt wird, ist aus der - wiederum im LOGO-Formalismus gehaltenen - Darstellung in den Abb. 26 und 27 zu ersehen.

Der wechselseitige Aufruf dieser beiden Produktionssysteme erfolgt in den Produktionsregeln ADD1, TAB1 und ADD5, und zwar, wie ersichtlich, auf unterschiedliche Weise: In ADD1 und TAB1 wird das jeweils andere Produktionssystem durch die ‚do‘-Anweisung lediglich über eine Veränderung der

```
to ADD
10 make ,,PRODUCTION.LIST“ ,,ADD1 ADD2 ADD3 ADD4 ADD5“
end

ADD1
both :NUMBER1: = :empty: and :NUMBER2: = :empty:
→ make ,,NUMBER1“ request ,
  make ,,NUMBER2“ request ,
  do ,,ADD.TABLE“

ADD2
not (count of :TALLY1 :) = :NUMBER1:
→ make ,,TALLY1“ word of ,,X“ and :TALLY1:

ADD3
not (count of :TALLY2) = :NUMBER2:
→ make ,,TALLY2“ word of ,,X“ and :TALLY2:

ADD4
:SUM: = :empty:
→ make ,,SUM“ count of word of :TALLY1: and :TALLY2:

ADD5
ACTIVE :SUM:
→ make ,,PRODUCTION.SYSTEM“ ,,ADD.TABLE“,
  do :PRODUCTION.SYSTEM:
```

Abb. 26: Produktionssystem für das Addieren zweier ganzer positiver Zahlen.

```
to ADD.TABLE
10 make „PRODUCTION.LIST“ „TAB1 TAB2“
end

TAB1
:SUM: = :empty:
→ do „ADD“

TAB2
ACTIVE :SUM:
→ make „NEW.PRODUCTION“
          (words „TAB.“ :NUMBER1: „.“ :NUMBER2:) ,
   make „NEW.CONDITION“
          (sentences „both :NUMBER1 : = “ :NUMBER1 :
                   „and :NUMBER2: =“ :NUMBER2:) ,
   make „NEW.ACTION“
          (sentences „print“ :SUM: „,“
                   „make“ :quote: „HALT“ :quote:
                   :quote: „THE SUM HAS BEEN FOUND“ :quote:) ,
   make „HALT“ „THE SUM HAS BEEN COMPUTED AND
                 ENTERED INTO THE TABLE“ ,
   print :SUM:
```

Abb. 27: Produktionssystem zum Erzeugen einer Additionstabelle.

Liste der Produktionen, nicht jedoch auch des *Namens* des Produktionssystems aktiviert (was einer hierarchischen - oder übergeordneten - Abhängigkeit der Produktionssysteme entspricht); in ADD5 dagegen wird auch der Name des Produktionssystems verändert und der Sprung per ‚do :PRODUCTION.SYSTEM:‘ ausgeführt (dies entspricht eher einer heterarchischen - oder nebengeordneten - Abhängigkeit). Der unterschiedliche Gebrauch ergibt sich aus der Zielsetzung für die Interaktion der beiden Produktionssysteme. In ADD wird mit der Produktion ADD1 zunächst einmal die Eingabe der beiden Zahlenwerte abgefragt, worauf mit einem Sprung nach ADD.TABLE festzustellen ist, ob das Ergebnis dort schon gespeichert ist; anfangs ist das natürlich noch nicht der Fall, weshalb über TAB1 der Rücksprung nach ADD erfolgt. Sodann wird über die Produktionen ADD2, ADD3 und ADD4 das Berechnen des Ergebnisses durch zwei „X-Strichlisten“ (in ADD2 bzw. ADD3) und das Auszählen der Länge der daraus gebildeten Gesamtliste (in ADD4) durchgeführt. Damit ist die Summe berechnet und es erfolgt über ADD5 ein Sprung in das Produktionssystem ADD.TABLE, um in diesem das Ergebnis abzuspeichern, was mittels Produktion TAB2 vorbereitet wird: Im Arbeitsspeicher werden der Name für eine neue Produktionsregel sowie ein neuer Konditionalteil und ein neuer Aktionsteil eingerichtet (dabei sind die LOGO-Operatio-

nen ‚words' und ‚sentences' Erweiterungen der einfachen Operationen ‚word' und ‚sentence' - zum Zusammensetzen eines LOGO-Wortes bzw. eines LOGO-Satzes - mit beliebig vielen Argumenten; die LOGO-Variable :quote: hat das Anführungszeichen ,, als ihren Wert). Der Sinn der Aktionen in TAB2 ist, das berechnete Additionsergebnis als eine neue Produktionsregel in die Produktionsliste von ADD.TABLE aufzunehmen.

Um diese primitive Form von ,,Lernfähigkeit" realisieren zu können, muß der Interpreter um diese Möglichkeit der Adaptivität erweitert werden: Er muß, noch bevor er das Haltsignal im Arbeitsspeicher liest, auf :NEW.PRODUCTION: mit dem Schreiben einer neuen Produktionsregel (in LOGO also dem Schreiben einer neuen Funktionsdefinition) reagieren können. Der bereits in Abb. 23 (Abschnitt 3.2.2) vorgestellte Produktionssystem-Interpreter ist nunmehr um die Programmzeile

```
25 if ACTIVE :NEW.PRODUCTION:
   then EXPAND :PRODUCTION.SYSTEM:
```

zu ergänzen. Das hierin verwendete Unterprogramm EXPAND ist in Abb. 28 definiert.

```
to EXPAND :PRODUCTION.SYSTEM:
10 do sentence ,,to" :NEW.PRODUCTION:
15 do sentence ,,10 test" :NEW.CONDITION:
20 do sentence ,,20 iftrue" :NEW.ACTION:
25 do ,,end"
30 do sentence ,,erase" :PRODUCTION.SYSTEM:
35 do sentence ,,to" :PRODUCTION.SYSTEM:
40 do (sentences
       ,,10 make" :quote: ,,PRODUCTION.LIST" :quote:
       :quote: :NEW.PRODUCTION: :PRODUCTION.LIST: :quote:)
45 do ,,end"
50 make ,,NEW.PRODUCTION" :empty:
55 make ,,NEW.CONDITION" :empty:
60 make ,,NEW.ACTION" :empty:
65 do :PRODUCTION.SYSTEM:
end
```

Abb. 28: Unterprogramm zum Erweitern eines Produktionssystems.

Dieses Unterprogramm ist im Grunde ein ,,Programm zum Schreiben von Programmen": In den Programmzeilen 10-25 wird die Funktionsdefinition einer neuen Produktionsregel durchgeführt und in den Programmzeilen 35-45 die Funktionsdefinition des modifizierten, um die neue Produktionsregel erweiterten Produktionssystems, nachdem dessen alte Version mit der LOGO-Anweisung ‚erase' in Programmzeile 30 erst einmal gelöscht wurde.

Der Rest der EXPAND-Anweisung dient nur noch dem Löschen der nicht mehr benötigten Arbeitsspeicherinhalte (Programmzeilen 50-60) und dem Aktivieren des nunmehr veränderten Produktionssystems (Programmzeile 65). Aus der Funktionsdefinition von EXPAND wird nun verständlich, weshalb in Produktion ADD5 auch der Name des Produktionssystems (und nicht nur die Produktionsliste wie in ADD1 bzw. TAB1) geändert werden mußte: EXPAND erwartet den Wert der Variable :PRODUCTION.SYSTEM: als Argument (und das ist der Name eines Produktionssystems), um eben dieses Produktionssystem ,,expandieren“ zu können (im vorliegenden Fall also ,,ADD.TABLE“ und nicht ,,ADD“, von wo aus - in ADD5 - der Sprung erfolgt).

Der in Tabelle 4 dargestellte Programmlauf gibt einen überblick über das Interaktionsgeschehen zwischen den beiden Produktionssystemen ADD und ADD.TABLE. Am Ende des ersten Programmlaufs hat der Interpreter die neue Produktionsregel

```
to TAB.3.2
10  test both :NUMBER1: = 3 and :NUMBER2: = 2
20 iftrue print 5,
          make ,,HALT“ ,,THE SUM HAS BEEN FOUND“
end
```

geschrieben und an den Anfang der Produktionsliste von ADD.TABLE gesetzt, so daß diese Produktion bei einem erneuten Programmlauf mit den gleichen Zahlenwerten das Ergebnis sofort ausgeben kann, ohne es nochmals berechnen zu müssen (vgl. den zweiten Programmlauf im zweiten Teil von Tabelle 4).

Inwieweit dieses KI-System auch als ein Simulationsmodell für das Addieren angesehen werden kann, bliebe zu diskutieren. Zumindest für die anfänglichen Fähigkeiten eines Kindes im Zahlenrechnen bietet es eine brauchbare Beschreibung: Das Aufstellen der beiden ,,X-Strichlisten “ ist recht ähnlich dem kindlichen ,,Fingerrechnen“, insbesondere bei einstelligen Zahlen, und auch das Behalten derartiger Rechenergebnisse -wenn auch vielleicht erst nach längerer Übung und nicht beim erstenmal wie in ADD.TABLE - erscheint kindgemäß. Von weit größerer Bedeutung für die ,,künstliche Intelligenz“-Forschung im engeren Sinne ist jedoch die Frage nach der Effizienz dieser ,,Addiermaschine“. Man kann sich leicht vorstellen, daß die Arbeitsweise mit ADD und ADD.TABLE schnell unökonomisch wird, wenn ADD mit großen Zahlen rechnen soll (was lange ,,X-Strichlisten“ ergäbe) oder wenn in ADD. TABLE umfangreiche Mengen von Rechenergebnissen zu speichern sind (was ein u.U. langwieriges Suchen nach einem bestimmten Ergebnis bedeutete). Beide Produktionssysteme müßten um geeignete Produktionen ergänzt werden, um die Arbeitsweise zu optimieren. Beispielsweise könnte auf die Zerleg-

Tabelle 4: Zwei Programmläufe der Produktionssysteme ADD und ADD.TABLE.

1. Lauf mit RUN ,,ADD"

PR	Arbeitsspeicher-Veränderungen
Start	:PRODUCTION.SYSTEM: is ,,ADD" :PRODUCTION.LIST: is ,,ADD1 ADD2 ADD3 ADD4 ADD5"
ADD1	*3 [Eingabe] *2 [Eingabe] :NUMBER1: is ,,3" :NUMBER2: is ,,2" :PRODUCTION.LIST: is ,,TAB1 TAB2,,
TAB1	:PRODUCTION.LIST: is ,,ADD1 ADD2 ADD3 ADD4 ADD5"
ADD2	:TALLY1: is ,,X"
ADD2	:TALLY1: is ,,XX"
ADD2	:TALLY1: is ,,XXX"
ADD3	:TALLY2: is ,,X"
ADD3	:TALLY2: is ,,XX"
ADD4	:SUM: is ,,5"
ADD5	:PRODUCTION.SYSTEM: is ,,ADD.TABLE" :PRODUCTION.LIST: is ,,TAB1 TAB2"
TAB2	:NEW.PRODUCTION: is ,,TAB.3.2" :NEW.CONDITION: is ,,both :NUMBERl: = 3 and :NUMBER2: = 2" :NEW.ACTION: is ,,print 5 , make ,,HALT" ,,THE SUM HAS BEEN FOUND" " :HALT: is ,,THE SUM HAS BEEN COMPUTED AND ENTERED INTO THE TABLE" 5 [Ausgabe]
Ende (nach EXPAND)	:PRODUCTION.LIST: is ,,TAB.3.2 TAB1 TAB2" :NEW.PROSUCTION: is ,, " :NEW.CONDITION: is ,, " :NEW.ACTION: is ,, "

Tabelle 4: Fortsetzung

2. Lauf mit RUN „ADD“ und der gleichen Eingabe

PR	Arbeitsspeicher-Veränderungen	
Start	:PRODUCTION.SYSTEM: is „ADD“ :PRODUCTION.LIST: is „ADD1 ADD2 ADD3 ADD4 ADD5“	
ADD1	+3 +2 :NUMBER1: is „3“ :NUMBER2: is „2“ :PRODUCTION.LIST: is „TAB.3.2 TAB1 TAB2“	[Eingabe] [Eingabe]
TAB.3.2	:HALT: is „THE SUM HAS BEEN FOUND“ 5	 [Ausgabe]

barkeit von Zahlen im dekadischen System zurückgegriffen werden, um die Zähloperationen in ADD zu vereinfachen, oder es könnte die Kommutativität der Addition ausgenutzt werden, um den Umfang von ADD.TABLE zu reduzieren. Die Frage ist nur, *wer* diese weitergehende Form von Adaptivität - nämlich „Lernen aus der Unzulänglichkeit des Systemverhaltens“ - bewerkstelligen soll, der Modellkonstrukteur oder aber der Produktionssystem-Interpreter selbst, dem weitergehende Möglichkeiten des „Lernens“ als die simple EXPAND-Anweisung eingebaut sein sollten. Insbesondere wären hier - nach einiger Laufzeit von ADD (z.B. mit großen Zahlen) und ADD.TABLE (z.B. nach einer umfangreichen Tabellierung) - Phasen einer handlungsgesteuerten Interpretation (vgl. Abschnitt 3.3.1) sinnvoll, um zu einer Effektivitätsbeurteilung beider Produktionssysteme durch den Interpreter selbst gelangen zu können.

Eines der bemerkenswertesten KI-Systeme, das in dieser Richtung auf der Grundlage der Produktionssystem-Konzeption entwickelt wurde, ist das „AM-System“ von Lenat (1978, 1979). Das System „entdeckt“ mit einem Repertoire elementarer mathematischer Begriffe aus der Mengenlehre neue mathematische Konzepte und Relationen (wie z.B. Zahlbegriff, arithmetische Operationen, Primzahlpaare, Goldbachsche Vermutung, Diophantische Gleichungen, aber auch neuartige, in der Mathematik bisher *unbekannte* Begriffe wie z.B. „Zahlen mit maximal vielen Teilern“). Im Zahlenspiegel seiner Statistiken ist das AM-System schon ein recht interessantes „mathematisches Spielzeug“ : In etwa 1 Stunde Rechenzeit rekonstruiert es, wenn man so will, 100 Jahre Mathematikgeschichte auf der Basis von 115 Grundbegriffen und

250 Produktionsregeln und entwickelt 185 sehr differenzierte neue mathematische Konzepte (davon vom Autor 25 als „Gewinner“ und 60 als „Verlierer“ - neben 100 akzeptablen Begriffen - klassifiziert).

Dabei ist die Leistungsfähigkeit dieses KI-Systems nicht nur durch die Effektivität der 250 Produktionsregeln, die die „heuristische Suche“ in einem so großen Problemraum wie dem der Mathematik als einen „regelgeleiteten Explorationsprozeß“ gestalten helfen, sondern ebensosehr durch die Arbeitsweise des Produktionssystem-Interpreters bestimmt. Der Interpreter realisiert einen „Zwei-Paß-Prozeß“: In einem ersten Schritt wird durch eine zielgerichtete „Aufmerksamkeits-Fokussierung“ der augenblicklich „interessanteste Job“ ausgewählt (wobei AM einen konkreten Begriff von „Interessantheit“ hat); in einem zweiten Schritt werden die für diesen Job relevanten Produktionsregeln zusammengestellt und in der Reihenfolge ihrer Spezifität abgearbeitet, bis die maximal für einen Job zugestandene Rechenzeit (durchschnittlich 30 Sekunden Kernspeicherzeit) verbraucht ist. Sodann wird mit Schritt 1 der „Zwei-Paß-Prozeß“ für einen neuen Job gestartet, bis der Benutzer seine „mathematische Spielsitzung“ beendet. - Zumindest rudimentär sind in diesem KI-System schon so etwas wie „Bewußtseinsfunktionen“ (vgl. Abschnitt 3.3.4) realisiert: Die erwähnte „Aufmerksamkeits-Fokussierung“ wird durch eine entsprechende „Zeigerfunktion“ des Interpreters ermöglicht, während eine „übersetzerfunktion“ das intern erzeugte Verhalten so externalisiert, daß dessen Ergebnisse dem AM-System dialoggesteuert wieder rückgemeldet werden können.

4. *Validierung und Anwendbarkeit von Simulationsmodellen*

Die Frage der Gültigkeit oder Validität von Simulationsmodellen und deren Anwendbarkeit ist im Grunde nichts anderes als die Frage nach dem Verhältnis von *Theorie* und *Empirie,* wie es sich generell in den Wissenschaften als methodologisches Problem stellt, hier nurmehr konkretisiert auf das Verhältnis der Theorie der Informationsverarbeitung zur Empirie psychischer Phänomene wie Wahrnehmen, Denken, Lernen, Handeln usw. Das heißt dann aber auch, daß es für die Simulationsmethodik keine grundsätzlich anderen, von den übrigen wissenschaftlichen Methoden verschiedenen Probleme der Gültigkeits- und Anwendbarkeitsprüfung gibt. Es gilt lediglich zu bedenken, daß das Paradigma der Computer-Simulation in der Psychologie - wie in Abschnitt 2.1 ausgeführt - „den Typ der dynamischen, deterministischen, qualitativen und analytischen Erkundungsmodelle zur Nachbildung menschlichen Verhaltens am Beispiel von einzelnen und interagierenden Individuen bevorzugt“, wir es also im wesentlichen mit Einzelfalluntersuchungen zu tun haben, so daß viele der gängigen Überprüfungsmethoden - wie z.B. inferenzstatistische Verfahren für statische, stochastische, quantitative Aggregatmodelle - in der Simulationsmethodik wenig anwendbar sind.

Ausgehend von dem in dem Theorie-Empirie-Verhältnis vermittelnden Modellbegriff lassen sich Kriterien entwickeln, nach denen einerseits der Wirklichkeitsbezug, andererseits aber auch der theoretische Status von Simulationsmodellen und KI-Systemen beurteilt werden kann.

4.1 Wirklichkeitsbezug und Modellrelationen

Unter erkenntnistheoretisch-methodologischen Aspekten ist das Paradigma der Computer-Simulation in der Psychologie mit den Grundproblemen der psychologischen Meßtheorie vergleichbar: Wirklichkeitsbezug und Modellrelationen bestimmen sich aus dem dreifachen Verhältnis von *Abbildbarkeit, Eindeutigkeit* und *Bedeutsamkeit* wissenschaftlicher Beobachtung. Eine Aufschlüsselung dieses Dreiecksverhältnisses wird die Grundlage für eine Diskussion der Validierungs- und Anwendbarkeitsproblematik der Simulationsmethodik abgeben.

4.1.1 Modellbildung als *homomorphe Abbildung*

Voraussetzung für die Modellierbarkeit psychischer Phänomene (in der Meßtheorie also die Meßbarkeit psychischer Merkmale wie z.B. „Intelligenz“ oder „Persönlichkeit“) ist (1) die Existenz eines *empirischen Relativs,* formal

$$\text{EmpiR} = \langle M; P_1, \ldots, P_k \rangle,$$

in dem bestimmte, empirisch gegebene „Merkmalsträger“ M (z.B. Personen) und „Beziehungen“ P_i zwischen diesen Merkmalsträgern (z.B. Intelligenz oder Persönlichkeit) definierbar sein müssen, und (2) die Konstruierbarkeit eines *theoretischen Relativs,* formal

$$\text{TheoR} = \langle N; R_1, \ldots, R_k \rangle,$$

in dem geeignete „theoretische Objekte“ N (z.B. Zahlen) und Relationen R_i zwischen diesen Objekten (z.B. numerische Prädikate und Operationen) herstellbar sind. Die *Modellbildung* besteht dann aus der homomorphen („strukturerhaltenden“) Abbildung φ des empirischen Relativs in das theoretische Relativ, formal

$$\varphi: \text{EmpiR} \rightarrow \text{TheoR},$$

so daß für beliebige m, m' $\in$ M und i = 1,. . . , k gilt:

$$\varphi[P_i(m,m')] = R_i[\varphi(m),\varphi(m')] \text{ mit } \varphi(m),\varphi(m') \in N.$$

Mit anderen Worten: Durch den Homomorphismus φ einer Modellbildung bleibt die in den empirischen Relationen P_i gegebene reale Struktur $\langle M; P_1,$

$\ldots, P_k\rangle$ in der durch die theoretischen Relationen R_i definierten formalen Struktur $\langle N; R_1, \ldots, R_i\rangle$ erhalten. Am Beispiel des Messens besteht die homomorphe Abbildung aus dem Zuordnen von (ganzen oder reellen) Zahlen zu den (empirischen) Merkmalsträgern in bezug auf die jeweilige Merkmalsausprägung.

Auf das Paradigma der Computer-Simulation angewandt, wird der Homomorphismus einer Modellbildung durch die Programmierung in einer geeigneten Programmiersprache erstellt, wie dies in den Beispielen der Abschnitte 2.2 und 3.2 illustriert wurde. Empirisches Relativ für das in Abschnitt 2.2 eingeführte Beispiel des „Simple Concept Attainment" war die reale Struktur $\langle M; P_1, P_2, P_3, P_4\rangle$ mit

M = {Vl, Vp}		
P_1 = Vl-Beispielvorgabe	}	vgl.
P_2 = VP-Antwort	}	Flußdiagramm
P_3 = Vl-Rückmeldung	}	von Abb. 1
P_4 = Vl-Kriteriumswert,		

das durch die Programmierung in LOGO zugeordnete theoretische Relativ die Programmstruktur $\langle N; R_1, R_2, R_3, R_4\rangle$ mit

N = {EXPERIMENTER, SUBJECT}		
R_1 = EXPERIMENTER'S.INSTANCE	}	vgl.
R_2 = SUBJECT'S.ANSWER	}	Programme
R_3 = EXPERIMENTER'S.FEEDBACK	}	in
R_4 = :CRITERION.VALUE:	}	Abb.2, 4-6 u.a.

Für das in Abschnitt 3.2 eingeführte Beispiel des „Single Letter Exclusion" war empirisches Relativ die reale Struktur (M; {Pi}) mit

M = {Vp, Aufgabe des „Unpassenden Streichens"}
$\{P_i\}$ = die fünf in der Protokollanalyse von Tabelle 2 verwendeten Operatoren,

das in Form eines Produktionssystems programmierte theoretische Relativ die Programmstruktur $\langle N; \{R_i\}\rangle$ mit

N = {SINGLE.LETTER.EXCLUSION, :GIVEN.ITEMS:}
$\{R_i\}$ = {SILEX1, . . ., SILEX8}.

4.1.2 Grundprobleme der Modellrelationen

Kennzeichnend für das Verhältnis von Wirklichkeitsbezug und Modellrelationen sind drei Grundprobleme, wie sie in der Theorie des Messens herausgear-

beitet wurden und die sinngemäß auch auf die Theorie der Informationsverarbeitung übertragen werden können:

(1) Das Abbildbarkeits- oder Repräsentationsproblem.
(2) Das Eindeutigkeits- oder Transformierbarkeitsproblem.
(3) Das Bedeutsamkeits- oder Testbarkeitsproblem.

(1) Das *Abbildbarkeitsproblem* beinhaltet die Frage nach der Darstellbarkeit eines Homomorphismus zwischen einem empirischen und einem theoretischen Relativ, wie sie in der obigen Diskussion eingeführt worden ist; Antwort in der Meßtheorie ist die Formulierung eines Abbildbarkeitstheorems (meist aufgrund einer geeigneten Axiomatisierung), in der Theorie der Informationsverarbeitung die Programmierung eines entsprechenden Simulationsmodells. Die oben angeführten Programmbeispiele illustrieren diesen Sachverhalt.

(2) Das *Eindeutigkeitsproblem* stellt die Frage nach der Zulässigkeit von Transformationen eines theoretischen Relativs in ein anderes, um aus den Transformationsbedingungen die Invarianzeigenschaften des gewählten Homomorphismus ablesen zu können. Antwort in der Meßtheorie ist die Angabe der zulässigen numerischen Transformationen, wodurch der Skalentyp des dem Meßvorgang zugrundeliegenden Homomorphismus festgelegt wird. In der Simulationsmethodik würde dem die Neuprogrammierung des ursprünglichen Simulationsmodells - entweder in einer anderen Programmiersprache oder mit einem anderen Modellansatz - entsprechen, ohne daß dadurch das Modellverhalten, der ,,Trace" des Simulationsprogramms, verändert wird. Beispiele sind zum einen die ursprüngliche LOGO-Programmierung von SIMPLE.CONCEPT.ATTAINMENT und dessen in Abschnitt 2.3 diskutierte Programmierbarkeit in LISP (vgl. Abb. 17), zum anderen das ursprünglich als Produktionssystem programmierte SINGLE.LETTER.EXCLUSION und dessen in Abschnitt 3.2.3 behandelte sequentielle Programmversion (vgl. Abb. 25); jede der beiden Programmvarianten zeigt ein völlig identisches Modellverhalten, so daß ihre unterschiedliche Programmierung zulässige Transformationen für den jeweils zugrundeliegenden Homomorphismus darstellen.

(3) Das *Bedeutsamkeitsproblem* bezieht sich auf die für die Validitätsprüfung wichtigste Frage der Relevanz theoretischer Aussagen aufgrund einer bestimmten Repräsentation und deren zulässigen Transformationen. Antwort in der Meßtheorie ist die Formulierung konkreter Skalierungsvorschriften und die Angabe zulässiger Rechenoperationen für die Skalenwerte (z.B. zur Berechnung statistischer Kennwerte und zur Anwendung statistischer Tests). Für die Simulationsmethodik wäre eine Antwort in der Verfügbarkeit geeigneter empirischer Tests (wie z.B. der Turing-Test oder der Protokoll-Trace-Vergleich, vgl. Abschnitt 4.3) und in der Ableitbarkeit empirisch prüfbarer Hypo-

thesen aus dem Simulationsmodell (oder aus der Rahmentheorie, die dem Modell zugrunde liegt) zu suchen.

4.1.3 Kommutatives Diagramm

Der Zusammenhang der drei Problembereiche läßt sich in Form eines kommutativen Diagramms darstellen, wie es in Abb. 29 - in Anlehnung an das kommutative Diagramm der Meßtheorie (vgl. Ueckert, 1980c, S. 193) - wiedergegeben ist.

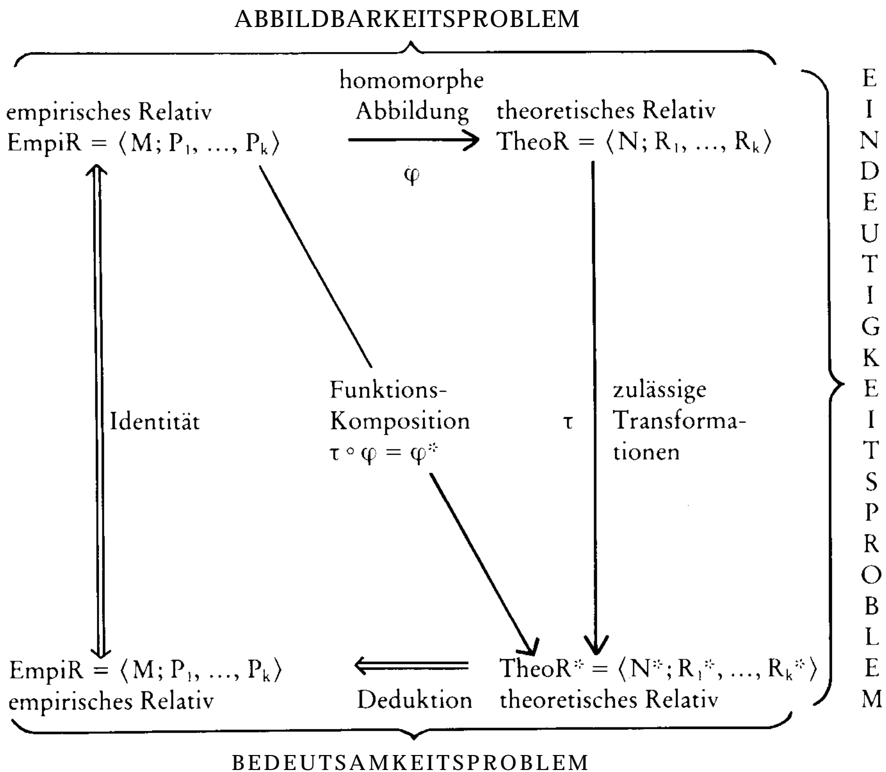

Abb. 29: Kommutatives Diagramm der Modellrelationen für die Computer-Simulation.

Die formal-mathematische Behandlung des Abbildbarkeitsproblems wurde in Abschnitt 4.1.1 schon gegeben. Das Eindeutigkeitsproblem und das Bedeut-

samkeitsproblem beinhaltet den Sachverhalt, daß es zu dem theoretischen Relativ

$$\text{TheoR} = \langle N; R_1, \ldots, R_k \rangle$$

stets ein weiteres theoretisches Relativ

$$\text{TheoR}^* = \langle N^*; R_1^*, \ldots, R_k^* \rangle$$

mit anderen „theoretischen Objekten" N^* und Relationen R_i^* zwischen diesen gibt, das die zulässigen Transformationen in Form einer homomorphen Abbildung

$$\tau: \text{TheoR} \rightarrow \text{TheoR}^*$$

beschreibt, so daß für beliebige $n, n' \in N$ und $i = 1, \ldots, k$ gilt:

$$\tau[R_i(n,n')] = R_i^*[\tau(n),\tau(n')] \text{ mit } \tau(n),\tau(n') \in N^*.$$

Das aber ist kommutativ äquivalent mit dem Sachverhalt, daß die Funktionskomposition $\tau \circ \varphi$, d.h. die Hintereinanderausführung der beiden Abbildungen φ und τ, einen neuen Homomorphismus

$$\varphi^*: \text{EmpiR} \rightarrow \text{TheoR}^*$$

erzeugt, der das ursprüngliche empirische Relativ $\langle M; P_1, \ldots, P_k \rangle$ in das neue theoretische Relativ $\langle N^*; R_1^*, \ldots, R_k^* \rangle$ abbildet, so daß gilt:

$$\varphi^* = \{n^* \mid \bigwedge (m \in M) \bigvee (n \in N)\ \varphi(m) = n \text{ und } \tau(n) = n^*\},$$

d.h. für beliebige $m \in M$ gilt stets

$$\tau[\varphi(m)] = \varphi^*(m) = n^*.$$

Mit anderen Worten: In einer Modellbildung sind alle die aus ihr deduzierbaren theoretischen Aussagen (Hypothesen) bedeutsam, die die Identität des empirischen Relativs aufgrund der zulässigen Transformationen des theoretischen Relativs erhalten bzw. nicht verletzen, was empirisch in geeigneter Weise - durch entsprechende Tests oder Experimente - nachgewiesen werden kann.

An den bisher eingeführten Programmbeispielen läßt sich dieser Sachverhalt anschaulich illustrieren. Zu dem empirischen Relativ des „Simple Concept Attainment" wurden in Abschnitt 2.2 und 2.3 zwei verschiedene theoretische Relative diskutiert, einmal die in LOGO programmierte ursprüngliche Version von SIMPLE.CONCEPT.ATTAINMENT (Abb. 2) und zum anderen die entsprechende Version in LISP als SIMPLE-CONCEPT-ATTAINMENT (Abb. 17). Die LISP-Version ist offensichtlich trivial, da sie eine identische Transformation der LOGO-Version darstellt; sie würde zwar ein - für die

Eindeutigkeitsbestimmung gefordertes - gleiches Modellverhalten zeigen, für die Bedeutsamkeitsfrage jedoch wenig hergeben, da sie über das empirische Relativ des „Simple Concept Attainment" nichts aussagt, was nicht schon in der LOGO-Version - nach einer entsprechenden Gültigkeitsprüfung hierzu - gesagt werden könnte.

Anders verhält es sich dagegen mit dem Beispiel der Aufgabe des „Unpassenden Streichens" aus Abschnitt 3. Zu dem empirischen Relativ des „Single Letter Exclusion" wurde das theoretische Relativ des SINGLE.LETTER.EXCLUSION nach der Konzeption von informationellen Produktionssystemen konstruiert, dem in Abb. 25 ein zweites, als sequentielles LOGO-Programm geschriebenes theoretisches Relativ gegenübergestellt wurde; beide Versionen zeigen wiederum ein identisches Modellverhalten und bestimmen damit die zulässigen Transformationen des Simulationsmodells. Unter dem Bedeutsamkeitsaspekt läßt sich hier jedoch ableiten, daß die Modellarchitektur von Produktionssystemen - Arbeitsspeicher, Produktionsspeicher und Interpreter (vgl. Abschnitt 3.2.1) - *nicht* abbildungsrelevant für das Lösen von Aufgaben des „Unpassenden Streichens" ist (was ja auch nicht das Ziel einer Modellbildung zu dieser Aufgabenstellung ist), denn das sequentielle LOGO-Programm leistet das gleiche wie die Produktionssystemversion, ohne explizit auf Speicherverwaltung und Programminterpretation einzugehen (wenn dies auch implizit dennoch geschieht, vgl. die Diskussion dazu in Abschnitt 3.2.3). Ist die Produktionssystem-Konzeption jedoch selbst - als generelle Modellarchitektur für die menschliche Informationsverarbeitung - Gegenstand der Modellbildung, und zwar im Rahmen einer allgemeinen Theorie der Informationsverarbeitung, dann sind die Fragen der Abbildbarkeit, Eindeutigkeit und Bedeutsamkeit natürlich erneut zu stellen und zu beantworten; eine informelle Behandlung wurde in den Abschnitten 3.2, 3.3 und 3.4 schon gegeben.

4.2 Das Eindeutigkeitstheorem von Anderson

Simulationsmodelle in der Psychologie sind nicht nur *statische,* homomorphe Abbildungen psychischer Phänomene, sondern immer auch *dynamische,* prozedurale Nachbildungen der untersuchten Vorgänge, die auf einem Rechner reproduziert werden können. Oder allgemein: Die Theorie der Informationsverarbeitung hat es in ihrer Modellbildung stets mit einem „Repräsentations-Prozeß-Paar" (Anderson, 1976) zu tun, d.h. jedes in programmierter Form realisierte theoretische Relativ besteht aus einer *Repräsentation* von Information und *Prozessen,* die diese Repräsentation für die Verarbeitung von Information verwenden.

Das Problem ist jedoch, daß die interne Struktur der Repräsentation und die interne Funktionsweise der Prozesse des *empirischen* Relativs, dessen Nachbil-

dung in dem theoretischen Relativ eines Simulationsmodells angestrebt wird, der direkten Beobachtung nicht zugänglich sind, sondern aus dem beobachtbaren Verhalten erschlossen werden müssen. Diese zunächst das Abbildbarkeitsproblem betreffende Situation charakterisiert Anderson (1976, S. 10-11) folgendermaßen.

Zu jedem Zeitpunkt t erfolgt eine Eingabe i(t) in das informationsverarbeitende System, das eine interne Struktur s(t) mit Hilfe einer *Enkodierfunktion* E aus dieser Eingabe erzeugt, formal

$$s(t) = E[i(t)].$$

Die Enkodierfunktion

$$E: I \rightarrow S$$

bildet unser Modell der internen Repräsentation von Information in einem informationsverarbeitenden System.

Zum Zeitpunkt t' $\neq$ t bestimmt die interne Struktur s(t') mit Hilfe einer *Dekodierfunktion* D eine Ausgabe o(t'), die Systemantwort, formal

$$o(t') = D[s(t')].$$

Die Dekodierfunktion

$$D: S \rightarrow O$$

stellt unser Prozeßmodell der Informationsverarbeitung dar.

Ein anschauliches Beispiel ist das Paraphrasieren (Nacherzählen, freie Wiedergabe) von Sätzen: i(t) wäre der zu paraphrasierende Satz, E der Vorgang des Verstehens dieses Satzes, s(t) das damit erzielte (sprachlich-inhaltliche) Verständnis, D der Prozeß des Erzeugens einer paraphrasierenden Umschreibung des Verstandenen, und o(t') die (mündliche oder schriftliche) Darstellung der Paraphrase.

Die empirischen Daten, die wir über das beobachtbare Verhalten eines informationsverarbeitenden Systems haben, sind Folgen von $\langle i(t), o(t') \rangle$-Paaren (oder in der behavioristischen Terminologie: von Reiz-Reaktions-Paaren) - und das sind in der Tat oft nur „paraphrasierende" Daten wie beispielsweise die Verbalisierungsprotokolle des „lauten Denkens" (vgl. Abschnitt 3.1.1). Die Frage ist nun, ob diese Daten hinreichend sind, um die Kodierungsfunktionen E und D erschließen zu können. Die Antwort, die Anderson darauf gibt, ist eindeutig „Nein", und zwar aus folgenden Gründen.

Angenommen, es existiere ein Simulationsmodell M mit den Kodierungsfunktionen E und D (in unserer Terminologie: es gäbe einen Homomorphismus von einem empirischen Relativ in ein theoretisches Relativ). Mit der Definierbarkeit von Äquivalenzklassen $[i]_E$ unter E, formal

$$i', i'' \in [i]_E \text{ genau dann, wenn } E(i') = E(i''),$$

lassen sich die zulässigen Transformationen des Simulationsmodells M in ein anderes Modell M* angeben, wenn für die neue Enkodierfunktion E*: gezeigt werden kann, daß für alle Eingaben $i \in I$

$$[i]_{E^*} \subset [i]_E$$

gilt. Da eine Dekodierfunktion D*, gegeben die Enkodierfunktion E*, immer so gewählt werden kann, daß das resultierende Modellverhalten im Vergleich zwischen M und M* unverändert bleibt, ist es stets möglich, das Originalmodell M durch ein Zweitmodell M* nachzubilden. Anderson beschreibt dies in dem folgenden - hier als *Eindeutigkeitstheorem* bezeichneten - Satz:

> Ein Modell M mit einer Enkodierfunktion E kann durch ein anderes Modell M* mit einer Enkodierfunktion E* vollständig nachgebildet werden, wenn für alle Eingaben $i \in I$ gilt, daß $[i]_{E^*} \subset [i]_E$.

Mit anderen Worten: Die Nachbildung von M ist immer möglich, wenn es dem nachbildenden Modell M* in seiner internen Repräsentation gelingt, jede von M unterscheidbare Eingabe ebenfalls zu unterscheiden. Der triviale Fall wäre, wenn M* jeder Eingabe eine eigene - wenn auch u.U. redundante - interne Repräsentation zuordnete; da dies immer erreicht werden kann, gibt es für jedes beliebige M ein nachbildendes M*. (Für einen Beweis des Theorems vgl. Anderson, 1976, S. 11-12.)

Allgemein kann man sagen, daß jedes psychologische Simulationsmodell ein Repräsentations-Prozeß-Paar ist und daß man völlig verschiedene Repräsentationsmodelle (dargestellt durch unterschiedliche Enkodierfunktionen) wählen und dennoch zu äquivalenten Aussagen mit Hilfe der damit konstruierbaren Modelle gelangen kann, da die Repräsentationsunterschiede durch die geeignete Wahl von Prozeßmodellen (dargestellt durch entsprechende Dekodierfunktionen) kompensiert werden können.

4.3 Empirische Tests von Simulationsmodellen

Inwieweit die Modellbildung in Form eines Homomorphismus φ bzw. φ^* als gelungen angesehen werden kann (was eine Antwort auf das Abbildbarkeitsproblem der Computer-Simulation darstellen würde) und inwiefern bestimmte Aussagen aus der Realisierung eines Simulationsmodells abgeleitet werden

können (was eine Beantwortung des Bedeutsamkeitsproblems beinhaltete), sind empirische Fragestellungen. Zu ihrer Lösung haben sich in der Simulationsmethodik der Psychologie zwei Überprüfungsverfahren herausgebildet, die insbesondere der Einzelfallcharakteristik psychologischer Simulationsmodelle Rechnung tragen: der sog. Turing-Test des Modellverhaltens und der Protokoll-Trace-Vergleich zwischen Verbalisierungsdaten und Modellausgabe.

4.3.1 Turing-Test

Der Turing-Test ist nach der Intention seines Erfinders, des englischen Mathematikers Turing, eher ein geistreiches Frage-Antwort-Spiel mit dem Rechner als ein ernsthafter Test für Simulationsmodelle (vgl. Turing, 1950). In seiner ursprünglichen Form ist der Turing-Test ein „Imitationsspiel" derart, daß der Rechner so programmiert ist, einen Menschen in seinem Verbalverhalten perfekt „nachzuahmen", daß es von dem eines richtigen Menschen nicht mehr zu unterscheiden ist. Konkret sieht der Test - zumindest von der Konzeption her - so aus, daß ein Fragesteller an einem Terminal sitzt und - am besten über zwei getrennte Fernschreiber - sowohl mit dem Rechner als auch mit einem menschlichen Kommunikationspartner verbunden ist; in einem beliebig strukturierbaren Dialog kann nun der Fragesteller versuchen herauszufinden, welcher seiner beiden Gesprächspartner der Rechner bzw. die Person ist. Spielt man dieses „Imitationsspiel" mit einer Reihe von Fragestellern durch, dann sollten sich bei der „Maschinenfrage", d.h. wer der Rechner und wer die Person ist, richtige und falsche Zuordnungen nur nach dem Zufallsprinzip verteilen (d.h. einer Gleichverteilung folgen), wenn das Simulationsprogramm perfekt das nachbildet, was es nachbilden soll.

Es ist klar, daß in dieser unstandardisierten Form der Test wenig brauchbar ist, zumal die ursprüngliche Version von Turing noch etwas komplizierter ist als die oben beschriebene. Danach hat es der Fragesteller entweder mit einer Frau und einem *Mann,* der eine Frau imitiert, oder mit einer Frau und einem *Rechner,* der eine Frau imitiert, zu tun; in beiden Fällen wird am Ende dieses „Imitationsspiels" die „Frauenfrage" gestellt: Wer ist jeweils die Frau und wer der eine Frau imitierende Mann bzw. Rechner? In dieser konfundierenden Weise ist der Test natürlich noch weniger brauchbar als in der vereinfachten, auf die „Maschinenfrage" reduzierten Form. Abelson (1968) hat daher einen „erweiterten Turing-Test" vorgeschlagen, in dem vor der eigentlichen „Frauenfrage" die Basisrate für die „Rollenqualität" des Mannes bestimmt wird, eine Frau in einer bestimmten Dimension (wie z.B. Intelligenz oder Persönlichkeit) zu imitieren; diese Basisrate sollte dann für das Rechnerprogramm ebenfalls erreicht werden (wobei die „Maschinenfrage" dann ganz entfällt): Liegt das Programm signifikant *über* der Basisrate, ist es „zu männlich", liegt es *darunter,* ist es „zu weiblich" in der untersuchten Verhaltensdimension.

Wie alle Rating-Verfahren ist auch dieses wenig zuverlässig, abgesehen von seiner theoretisch recht schwachen Fundierung. Colby und seine Mitarbeiter (vgl. Colby, 1975) entwickelten daher zu ihrem Simulationsmodell des paranoiden Prozesses einen differenzierteren ,,experimentellen Ununterscheidbarkeitstest", in dem 10stufige Ratings für die interessierenden Dimensionen (hier: Paranoia) zu transkribierten Rechnerläufen bzw. Verbalprotokollen (hier von paranoiden Patienten, in beiden Fällen als Arzt-Patient-Dialoge) möglich sind. Die Autoren berichten über eine Reihe von statistisch abgesicherten Analysen auf der Grundlage dieses Verfahrens (mit ganz passablen Rater-Übereinstimmungen), so daß dieser ,,Ununterscheidbarkeitstest" durchaus als ein praktikabler Ansatz der Validitätsprüfung von Simulationsmodellen angesehen werden kann.

4.3.2 Protokoll-Trace-Vergleich

Grundsätzlich gibt es für jede Modellbildung zwei Arten von Abbildungsfehlern: Das Modell kann zu *wenig* über die abgebildete Realität aussagen, d.h. es bildet nicht all die Aspekte ab, die man in die Modellbildung einbeziehen wollte *(Abbildungsfehler 1. Art),* und das Modell kann *zu viel* über die abgebildete Realität aussagen, d.h. es überzeichnet Aspekte, die in dem abgebildeten Bereich so gar nicht vorkommen *(Abbildungsfehler 2. Art).* Die beste Möglichkeit, solchen Abbildungsfehlern in der Simulationsmethodik auf die Spur zu kommen, ist der Protokoll-Trace-Vergleich zwischen den Verbalisierungsdaten eines Probanden und dem in geeigneter Ausgabeform realisierten Programmlauf des Modells, dem ,,Trace" des Rechners (Beispiele für einen solchen ,,Trace" sind die Programmläufe in Abb. 3 für das ,,Simple Concept Attainment", in Tabelle 3 für das ,,Single Letter Exclusion" - hier auch mit entsprechenden Protokolldaten - und in Tabelle 4 für die Produktionssysteme ADD und ADD.TABLE). Im Idealfall sollten Protokoll und ,,Trace" in allen die Modellbildung betreffenden Aspekten so übereinstimmen, daß ,,Vorbild" und ,,Nachbild" nicht mehr unterscheidbar sind (etwa im Sinne des ,,erweiterten Turing-Tests" nach Abelson oder des ,,experimentellen Ununterscheidbarkeitstests" von Colby).

Im Grunde ist der Protokoll-Trace-Vergleich, wie am Beispiel des ,,Single Letter Exclusion" in Abschnitt 3.2 schon gezeigt wurde, konstruktiver Bestandteil der Modellentwicklung. An diesem Beispiel ist auch die besondere Problematik des Verfahrens erkennbar: In jedem Fall handelt es sich um einen *qualitativen* Vergleich zwischen *natürlichsprachlichen* Aussagen und *programmsprachlichen* Ausdrücken, deren inhaltliche Äquivalenz in vielen Fällen durchaus strittig sein kann.

Dennoch eröffnet der Protokoll-Trace-Vergleich einen konkreten Ansatz, Abbildungsfehler in der Entwicklung und Überprüfung von Simulationsmodellen

erkennen und beseitigen zu können. Abweichungen zwischen Protokoll und „Trace“ dahingehend, daß für die Modellbildung relevante Teile von Verbalisierungsdaten im Modell noch nicht reproduziert werden (wie beispielsweise für einen Probanden charakteristische Um- und Irrwege der Informationsverarbeitung), was einem Abbildungsfehler 1. Art entspricht, können für eine entscheidende Modellverbesserung herangezogen werden (weshalb der Protokoll-Trace-Vergleich auch schon in der Phase der Modellentwicklung verwendet wird). Andererseits können Abweichungen zwischen „Trace“ und Protokoll, die sich auf Teile des Programmlaufs beziehen, für die es in den Verbalisierungsdaten keine Entsprechungen gibt (Abbildungsfehler 2. Art), Anlaß für eine Modellanpassung sein, die ein Simulationsmodell „realitätsgerechter“ - und das heißt in den meisten Fällen „weniger künstlich“ oder einfach „menschlicher“ -werden lassen. Dabei ist allerdings zu unterscheiden, ob das Modell lediglich „Verbalisierungslücken“ ausfüllt, die notwendigerweise in einem funktionsfähigen Simulationsmodell überbrückt werden müssen (vgl. das Beispiel des „Single Letter Exclusion“ in Tabelle 3), oder ob es tatsächlich „zu viel“ an Kapazität und Effizienz hinsichtlich der abgebildeten Vorgänge der Informationsverarbeitung bringt, was einem echten Abbildungsfehler 2. Art entspräche.

Reichhaltiges Anschauungsmaterial für die Vorgehensweise des Protokoll-Trace-Vergleichs liefern Newell & Simon (1972) in ihrem Buch über menschliches Problemlösen, in dem eine Fülle von Beispielen (Kryptarithmetik, Logikaufgaben, Schachprobleme) in Protokoll-Trace-Ausschnitten vorgeführt wird. Ein Studium dieser Materialsammlung ist für eine Diskussion der empirischen Testbarkeit von Simulationsmodellen von außerordentlichem Wert.

4.4 Nicht-Falsifizierbarkeit von KI-Systemen

Die gezielte experimentalpsychologische Überprüfung von Simulationsmodellen hat in der Computer-Simulation bisher nur eine untergeordnete Rolle gespielt. Daß hier inzwischen ein Wandel eingetreten ist, beweisen viele neuere, mit der Simulationsmethodik arbeitende Ansätze in der Psychologie (beispielsweise im Bereich der mit semantischen Netzen operierenden Gedächtnistheorien, vgl. die einführende Darstellung von Wender, Colonius & Schulze, 1980).

Besonderer Vorteil der Verwendung von Simulationsmodellen ist die Möglichkeit der unbeschränkten Durchführung von Modellexperimenten, insbesondere mit verschiedenen Modellvarianten: Jede dieser Modellvarianten kann mit den empirischen Daten experimentalpsychologischer Untersuchungen verglichen werden, bis am Ende eine Variante resultiert, die als optimales Modell des nachgebildeten Realitätsbereichs angesehen werden kann. Ein Beispiel ist das

in Abschnitt 2.2 mit vier Modellvarianten vorgestellte Programm des „Simple Concept Attainment". In ihrer diesbezüglichen Arbeit haben Gregg & Simon (1967) eine ausführliche Diskussion der Modellvarianten im Vergleich mit empirischen Daten aus einer Reihe von Experimenten geliefert, deren Ergebnis nicht zuletzt darin zu sehen ist, daß die Simulationsmethodik sowohl der verbalsprachlichen als auch der mathematischen Modellbildung in Präzision (qua modelltheoretischer Annahmen), Explizitheit (qua Programmerstellung) und Validität (qua Experiment-Modell-Vergleich) überlegen ist.

Dennoch, trotz aller empirischer Testbarkeit mit Turing-Test oder Protokoll-Trace-Vergleich und experimentalpsychologischer Überprüfbarkeit durch Experiment-Modell-Vergleich kann man sich die Frage stellen, ab wann ein Simulationsmodell oder gar die dahinterstehende Theorie der Informationsverarbeitung als verifiziert - oder nach wissenschaftstheoretischen Überlegungen richtiger als falsifiziert - angesehen werden kann. Aus der Logik von Maschinen, die per Konstruktionsprinzip zur Informationsverarbeitung in der Lage *sind,* läßt diese Frage nur eine Antwort zu: Jedes lauffähige Computer-Programm ist - gleichgültig, ob es einen realen Prozeß der Informationsverarbeitung nachbildet oder nicht - eine Anwendung, eine konkrete Realisation der *Theorie* der Informationsverarbeitung und kann von daher grundsätzlich nicht als Falsifikationsinstanz für eben diese Theorie angesehen werden. Mit anderen Worten: Systeme der „künstlichen Intelligenz" - im weiten wie im engeren Sinne - sind prinzipiell *nicht falsifizierbar,* sondern allenfalls in unterschiedlichem Umfang auf die Modellierung realer Prozesse *anwendbar.*

Die einen derartigen Anspruch rechtfertigende Theoriekonzeption, der sog. strukturalistische Theoriebegriff oder „non-Statement view" von Theorien (vgl. Stegmüller, 1973), soll im folgenden auf die mit der Computer-Simulation verbundene Theorie der Informationsverarbeitung, insbesondere im Zusammenhang mit der Modellarchitektur von informationellen Produktionssystemen, direkt bezogen und diskutiert werden.

4.4.1 Der strukturalistische Theoriebegriff

Nach der herkömmlichen wissenschaftstheoretischen Auffassung besteht eine Theorie aus einem System von Sätzen (oder „Aussagen über die Realität"), deren Zusammenhang untereinander durch logische Ableitungsbeziehungen (Widerspruchsfreiheit, Kohärenz u. a.) und deren Gültigkeitsanspruch durch Wahrheitskriterien (Bestätigung, Falsifikation, Korrespondenz mit der Realität usw.) geregelt ist. Dieser als „Aussagenkonzeption" bezeichneten Auffassung von Theorien stellte Sneed einen anderen, ursprünglich im Bereich der theoretischen Physik entwickelten Ansatz gegenüber, den sog. „non-statement view" von Theorien, den Stegmüller (1973, 1980) auf die aktuelle wissen-

schaftstheoretische Diskussion übertragen hat (insbesondere vor dem Hintergrund der Kuhnschen Thesen über „normalwissenschaftlichen Fortschritt" und „revolutionären Wandel der Wissenschaft"). Nach diesem - im folgenden als „strukturalistische Theoriekonzeption" bezeichneten - Ansatz bestehen Theorien nicht aus Satzsystemen mit logischen Ableitungsbeziehungen und wahrheitsdefiniten Gültigkeitskriterien, sondern aus logisch-mathematischen Konstruktionen, deren instrumenteller Gebrauch durch modelltheoretische Formulierungen geregelt wird.

Ohne auf Einzelheiten allzusehr einzugehen (man vergleiche dazu die ausführliche Darstellung von Stegmüller, 1973), wird im folgenden die Übertragbarkeit dieses Ansatzes auf die Theorie der Informationsverarbeitung versucht, um damit eine Argumentationsbasis zu gewinnen, auf der die Anwendbarkeit und der instrumentelle Gebrauch der Computer-Simulation und der „künstlichen Intelligenz"-Forschung in plausibler Weise gerechtfertigt werden kann.

Nach der strukturalistischen Theoriekonzeption besteht eine Theorie T untrennbar aus zwei Komponenten:

(1) einer *logischen* Komponente K, die den „Kern" der diese Theorie kennzeichnenden Struktur in logisch-mathematischen Kategorien beschreibt, und
(2) einer *empirischen* Komponente A, die die „intendierten Anwendungen" der Theorie auf konkrete Gegenstandsbereiche der Realität beinhaltet.

Formal ist jede Theorie durch das geordnete Paar

$$T = (K, A)$$

darstellbar, so daß es grundsätzlich nicht zulässig ist, in einer Theorie nur von der einen Komponente zu reden, ohne die andere zu erwähnen, und umgekehrt. Oder allgemein gesagt: Es gibt keine Theorie ohne Anwendungen und es gibt keine Anwendungen ohne entsprechende Theorie.

In erster Annäherung kann man die in Abschnitt 4.1 eingeführten Begriffe des empirischen und des theoretischen Relativs auf diesen Theoriebegriff beziehen: Theoretische Relative gehören zum Strukturkern K, empirische Relative zu den Anwendungen A einer bestimmten Theorie T.

Kennzeichnend für den strukturalistischen Theoriebegriff ist der Sachverhalt, daß eine Theorie einerseits bei *gleichem* Strukturkern *verschiedene* intendierte Anwendungen haben kann (beispielsweise wenn die Individuenbereiche dieser Anwendungen verschieden sind), die durch *allgemeine,* in allen diesen Anwendungen geltenden Gesetzmäßigkeiten oder „Nebenbedingungen" miteinander verbunden sind, und andererseits der Strukturkern *so erweitert* werden kann, daß spezielle Gesetze nur in bestimmten, nicht jedoch in anderen Anwendun-

gen gelten, was durch die Angabe von *speziellen* Nebenbedingungen geregelt wird. Auf die Theorie der Informationsverarbeitung übertragen heißt das, daß mit der Formulierung eines informationsverarbeitenden Systems (IVS) zunächst der Strukturkern K(IVS) für die Theorie festgelegt wird, als deren intendierte Anwendungen A(IVS) primär der Rechner, sekundär aber auch - durch die Entwicklung der Computer-Simulation bedingt - der Mensch angesehen wird (also zwei verschiedene, mit unterschiedlichen Individuenbereichen arbeitende Anwendungen des gleichen Strukturkerns). Darüber hinaus gibt es jedoch auch spezielle Anwendungen der Theorie der Informationsverarbeitung, die im Falle des Menschen zu Eigengesetzlichkeiten führt, die von denen eines Rechners verschieden sind.

4.4.2 Die logische Komponente der Theorie der Informationsverarbeitung

Jedes Modell der Theorie der Informationsverarbeitung, d.h. jedes konkrete informationsverarbeitende System IVS, setzt sich aus folgenden Systemteilen zusammen:

(1) Eingabe-/Ausgabe-Einheiten,
(2) Datenspeicher,
(3) Programmspeicher,
(4) Prozessor für Daten-/Programmspeicher.

In der Modellarchitektur von informationellen Produktionssystemen (vgl. Abschnitt 3.2.1) entsprechen den Systemteilen (2) bis (4) Arbeitsspeicher, Produktionsspeicher und Interpreter.

Über diesen vier Systemteilen läßt sich die logische Komponente der Theorie der Informationsverarbeitung, der Strukturkern K(IVS), systematisch aufbauen.

Auf den Eingabe-/Ausgabe-Einheiten operieren zwei Mengen

I := nicht-leere („empirische“) Menge von *Eingaben* („Inputs“) für ein IVS (einschließlich der leeren Eingabe),
O := nicht-leere („empirische“) Menge von Ausgaben („Outputs“) für ein IVS (einschließlich der leeren Ausgabe),

deren Verknüpfung durch eine Funktion geregelt wird:

$R: I \rightarrow O$: = nicht-theoretische („empirische“) Funktion *(„Response-Funktion“)* des IVS, die die Menge der Eingaben in die Menge der Ausgaben abbildet, so daß für beliebige benachbarte Zeitpunkte $t < t'$ gilt:

$$R[i(t)] = o(t').$$

Die Eingabe-/Ausgabemengen bezeichnen die für das IVS empirisch verfügbare Information („Reiz-Reaktions-Muster“ der behavioristischen Psychologie). Die Response-Funktion R ist eine dynamische Funktion dahingehend, daß die Systemausgabe o(t') stets mit einer gewissen Latenzzeit t'-t auf die Systemeingabe i(t) folgt. Diese Funktion ist insofern „nicht-theoretisch“, als sie sich empirisch stets aus der Auswertung von $\langle i(t), o(t') \rangle$-Paaren, also aus dem Eingabe-Ausgabe-Verhalten eines IVS, bestimmen läßt. Die Einbeziehung der leeren Eingabe bzw. Ausgabe ist notwendig, um in der Zeitvariable t keine „Lücken“ offen zu lassen. (D.h. es wird nicht ausgeschlossen, daß auf eine Eingabe unmittelbar keine Ausgabe folgt oder für eine Ausgabe unmittelbar keine Eingabe gegeben ist oder aber daß das IVS zeitweise auch nach außen hin inaktiv ist, also ein leeres Eingabe-Ausgabe-Verhalten zeigt.)

Mit diesen Definitionen ist eine Menge von Systemen gegeben, die ausschließlich empirisch bestimmt sind und die als Menge der *„partiellen potentiellen Modelle”*

$$M_{pp} = \{x \mid x = \langle R; I, O \rangle\}$$

die *erste* Komponente des Strukturkerns K(IVS) darstellen; „partiell“ heißen diese Modelle deshalb, weil sie noch keine theoretischen Systemgrößen enthalten, und „potentiell“, weil sie nur „mögliche“, nicht jedoch auch schon „wirkliche“ Modelle eines IVS sind. Die Modelle $x \in M_{pp}$ sind dynamische Modelle, da das Modellverhalten durch die empirische Response-Funktion zeitabhängig ist.

Auf dem Datenspeicher - oder dem Arbeitsspeicher von informationellen Produktionssystemen - sind weitere Entitäten definierbar, die in bezug auf die Theorie der Informationsverarbeitung theoretischen Status haben, weil sie der unmittelbaren Beobachtung nicht zugänglich sind. Im einzelnen sind dies die Menge

S := nicht-leere („theoretische“) Menge von *internen Repräsentationen* („Symbolstrukturen“) eines IVS zur systemeigenen Darstellung von Information,

und die Systemfunktionen

$E: I \rightarrow S$:= theoretische („nicht-empirische“) Funktion *(„Enkodierfunktion“)* des IVS, die die Menge der Eingaben in die Menge interner Repräsentationen abbildet, so daß für jeden beliebigen Zeitpunkt t gilt:

$$E[i(t)] = s(t);$$

$D: S \rightarrow O$:= theoretische („nicht-empirische“) Funktion *(„Dekodierfunktion“)* des IVS, die die Menge der internen Repräsentationen in die Menge der Ausgaben abbildet, so daß für jeden beliebigen Zeitpunkt t gilt:

$$D[s(t)] = o(t);$$

U: S $\rightarrow$ S := theoretische („nicht-empirische“) Funktion *(„Umstrukturierungsfunktion“)* des IVS, die die Menge der internen Repräsentationen in sich selbst abbildet, so daß für beliebige benachbarte Zeitpunkte t < t’ gilt:

$$U[s(t)] = s'(t').$$

Die Repräsentationsmenge S und die Kodierungsfunktionen E und D sind aus der Darstellung des Eindeutigkeitstheorems von Anderson (vgl. Abschnitt 4.2) schon bekannt und entsprechen der dort eingeführten Bedeutung. Die beiden Kodierungsfunktionen sind statische Funktionen, da sie ihre Werte jeweils zeitgleich aus den Argumenten erzeugen. Die Umstrukturierungsfunktion U dagegen ist eine dynamische Funktion, da jede Umstrukturierung mit einem gewissen Zeitaufwand t’-t verbunden ist. Diese Funktion dient sowohl der Darstellbarkeit von sequentiellen Vorgängen der Informationsverarbeitung mittels

$$U[s(t)] = s'(t')$$

als auch der Erklärbarkeit von Eingaben, aus denen unmittelbar keine Ausgabe erfolgt, mittels

$$U[E[i(t)]] = s(t'),$$

und von Ausgaben, für die es keine direkte Eingabe gibt, mittels

$$D[U[s(t)]] = o(t').$$

Auf dem Programmspeicher - oder dem Produktionsspeicher von Produktionssystemen - sind damit drei Mengen von *„Programmstrukturen”* definierbar:

E(1) := Menge der Verknüpfungen der Enkodierfunktion mit ihrer Eingabemenge;
D(S) := Menge der Verknüpfungen der Dekodierfunktion mit der zu Systemausgaben geeigneten Repräsentationsmenge;
U(S) := Menge der Verknüpfungen der Umstrukturierungsfunktion mit der zu internen Umstrukturierungen verfügbaren Repräsentationsmenge.

Die Programme eines IVS bestehen ausschließlich aus der geeigneten Zusammenstellung von Programmstrukturen dieser drei Verknüpfungsmengen.

Schließlich operiert auf dem Prozessor für Daten- und Programmspeicher - dem Interpreter von Produktionssystemen - eine Meta-Funktion

P: {„E(I)“, „D(S)“, „U(S)“} $\rightarrow$ {E(I), D(S), U(S)} := theoretische („nicht-empirische“) Funktion höherer Ordnung *(„Prozessorfunktion”)* des IVS, die die Programmausdrücke der Form „F(X)“ derart in Programmausführungen F(X) übersetzt, daß für beliebige F(X) = E(I)/D(S)/U(S) gilt:

$$P[„F(X)“] = F(X).$$

Die Prozessorfunktion P ist insofern eine Metafunktion, als sie die konkrete Interpretation der drei Systemfunktionen E, D, U auf ihren jeweiligen Argumentmengen nach Maßgabe der im Programmspeicher gegebenen Programmstrukturen regelt.

Mit diesen zusätzlichen Definitionen ist nunmehr eine spezifischere Menge von Systemen angebbar: Neben empirischen Systemgrößen enthalten sie auch solche, die für die interne Struktur eines IVS und deren Interpretation kennzeichnend sind. Diese Systeme bilden als die Menge der *„potentiellen Modelle“*

$$M_p = \{y \mid y = \langle R, E, D, U, P; I, O, S\rangle\}$$

die *zweite* Komponente des Strukturkerns K(IVS); auch sie sind nur „mögliche“ und noch nicht „wirkliche“ Modelle eines IVS, da der Zusammenhang zwischen empirischen und theoretischen Systemgrößen noch nicht vollständig spezifiziert ist. Auch die Modelle $y \in M_p$ sind dynamische Modelle, da sie neben der empirischen Zeitfunktion R auch noch die theoretische Zeitfunktion U enthalten.

Schließlich wird über eine Menge von Systemen der Zusammenhang zwischen der empirischen Response-Funktion R und den theoretischen Systemfunktionen E, D, U so konkretisiert, daß jede Ausgabe des IVS eindeutig aus den Kodierungs- und/oder Umstrukturierungsoperationen von Eingaben bestimmbar ist; es ist dies die Menge der eigentlichen *Modelle*

$$M = \{z \mid z \in M_p \text{ und } R[i(t)] = D[U[E[i(t)]]]\}.$$

Diese Modelle bilden die *dritte* Komponente des Strukturkerns K(IVS); sie sind die „wirklichen“ Modelle eines IVS dahingehend, daß in ihnen alle Systemgrößen vollständig spezifiziert und in Zusammenhang gebracht sind, die für ein eindeutiges Systemverhalten relevant sind. Die Modelle $z \in M$ erklären die über die empirische Response-Funktion R beobachtbaren Latenzzeiten t'-t aus der Zeitverschiebung von Umstrukturierungsoperationen durch die theoretische Zeitfunktion U; damit sind auch diese Modelle als dynamische Modelle anzusehen.

Als letztes sind noch zwei weitere Komponenten des Strukturkerns der Theorie der Informationsverarbeitung zu definieren:

(1) Eine *„Restriktionsfunktion“*

$$\varrho: M_p \rightarrow M_{pp},$$

die die Menge der potentiellen Modelle in die Menge der partiellen potentiellen Modelle derart abbildet, daß gilt:

$$\varrho(\langle R, E, D, U, P; I, O, S\rangle) = \langle R; I, O\rangle,$$

d.h. die Restriktionsfunktion ermöglicht den Nachweis, ob ein Modell der Theorie der Informationsverarbeitung tatsächlich theoriespezifische Funktionen enthält (d.h. ob $M_p \neq M_{pp}$ gilt) oder nicht (d.h. ob $M_p = M_{pp}$, so daß die als „theoretisch" angesehenen Systemfunktionen nichts anderes als identische Transformationen sind).

(2) Eine Angabe von *allgemeinen Nebenbedingungen,* die den theoretischen Funktionen (und nur diesen) der potentiellen Modelle M_p auferlegt werden; für die Theorie der Informationsverarbeitung ist dies die äquivalenzklassenbildende Nebenbedingung

$$N(M_p) = \langle \approx, = \rangle$$

für die Funktionen E, D, U und P derart, daß gilt:

$i', i'' \in [i]_E \Leftrightarrow E(i') = E(i'')$ (Enkodierungs-Äquivalenz)
$s', s'' \in [s]_D \Leftrightarrow D(s') = D(s'')$ (Dekodierungs-Äquivalenz)
$s', s'' \in [s]_U \Leftrightarrow U(s') = U(s'')$ (Umstrukturierungs-Äquivalenz)
$„F'(A)“, „F''(B)“ \in [„F(X)“]_P \Leftrightarrow F'(A) = F''(B)$
(Interpretations-Äquivalenz)

Mit anderen Worten: Jedes IVS muß in der Lage sein, äquivalenten Eingaben eine identische interne Repräsentation zu geben bzw. aufgrund äquivalenter Repräsentationsstrukturen zu identischen Ausgaben und/oder identischen Umstrukturierungen zu kommen. Und umgekehrt: Ein IVS kann nur aufgrund identischer interner Repräsentationen Eingaben als äquivalent und nur aufgrund identischer Ausgaben bzw. Umstrukturierungen Repräsentationsstrukturen als äquivalent erkennen. Ferner müssen äquivalente Programmausdrücke zu einer identischen Interpretation durch das IVS führen (und umgekehrt).

Mit diesen fünf Komponenten ist nunmehr der Strukturkern der Theorie der Informationsverarbeitung formal definiert als

$$K(IVS) = \langle M_{pp}, M_p, M; \varrho, N(M_p) \rangle.$$

Die bisher diskutierten Systemgrößen eines informationsverarbeitenden Systems lassen sich anhand der Programmiersprache LOGO und den in ihr eingeführten Programmbeispielen leicht veranschaulichen.

Eingabe-Einheit ist die Tastatur des Fernschreibers oder des Sichtgerätes, Ausgabe-Einheit Papier oder Bildschirm. Eingaben erfolgen in laufenden LOGO-Programmen mit der LOGO-Operation ‚request', Ausgaben mit der LOGO-Anweisung ‚print' (oder anderer, in LOGO verfügbarer Ausgabefunktionen). Der empirischen Systemfunktion $R[i(t)]$ entspricht also im einfachsten Falle die LOGO-Programmzeile ‚print request' (was die identische Ausgabe einer

Eingabe bedeutet). Für das in Abschnitt 3.4 eingeführte ADD-Produktionssystem entspräche der empirischen Systemfunktion beispielsweise die Folge

‚RUN „ADD“ *3, *2‘ mit der Ausgabe ‚5‘.

In LOGO dient ein Teil des Arbeitsspeichers, der sog. Variablenspeicher, als Datenspeicher, während der Rest des Arbeitsspeichers den Programmspeicher darstellt (wobei Daten und Programme auch langfristig in sog. Dateien gespeichert werden können). Interne Repräsentationsstrukturen sind in LOGO als variable Namen+Wert-Zuweisungen (in Form von LOGO-Wörtern und -Sätzen) darstellbar. Beispiele für die Grundform theoretischer Systemfunktionen in LOGO sind:

Enkodierung E[i(t)]: make „X“ request
Dekodierung D[s(t)]: print :X:
Umstrukturierung U[s(t)]: make „Y“ :X:

Operationssequenzen anhand dieser Beispiele wären dann:

U[E[i(t)]]: make „X“ request , make „Y“ :X:
D[U[s(t)]]: make „Y“ :X: , print :Y:
D[U[E[i(t)]]]: make „X“ request , make „Y“ :X: , print :Y:

Die weitaus größte Zahl von Operationen und Anweisungen in einem LOGO-Programm sind in der Regel Umstrukturierungsoperationen an internen Repräsentationsstrukturen, in „lernfähigen“ Programmen aber auch an Programmstrukturen selbst, wie die EXPAND-Funktion für das ADD.TABLE-Produktionssystem in Abschnitt 3.4 zeigt.

Die Bedeutung des Prozessors für Daten- und Programmspeicher läßt sich am Beispiel des in Abschnitt 3.2 eingeführten Interpreters von informationellen Produktionssystemen gut demonstrieren. Im Grunde ist die ‚do‘-Anweisung die Prozessorfunktion von LOGO (der top-level von LOGO ist nichts anderes als eine endlose ‚do request‘-Schleife), und wie aus der Programmierung des Interpreters in Abb. 23 und 24 zu ersehen ist, spielt hier-vor allem in der READY- und in der FIRE-Funktion von PROCESS - die ‚do‘-Anweisung die zentrale Rolle für die Interpretation von Produktionssystemen und von Produktionsregeln. (Dabei ist es sogar möglich, daß die Prozessorfunktion ‚do‘ selbst im Aktionsteil von Produktionsregeln wieder vorkommen kann, wie die Produktionen SILEX4 von SINGLE.LETTER.EXCLUSION und ADD1, ADD5 und TAB1 von ADD bzw. ADD.TABLE zeigen.) Anhand der oben eingeführten LOGO-Beispiele für die Systemfunktionen E, D, U wäre für die Prozessorfunktion P zu schreiben:

P[„E[i(t)]“]: do „make „X“ request“
P[„D[s(t)]“]: do „print :X:“
P[„U[s(t)]“]: do „make „Y“ :X:“

Und für die Operationssequenzen:

P[„U[E[i(t)]]“]: do „make „X“ request , make „Y“ :X:“
P[„D[U[s(t)]]“]: do „make „Y“ :X: , print :Y:“
P[„D[U[E[i(t)]]]“]: do ,,make ,,X“ request , make ,,Y“ :X: , print :Y:“

wobei die Anführungszeichen zu Beginn und am Ende einer ‚do‘-Anweisung von solchen innerhalb der ‚do‘-Anweisung selbstverständlich zu unterscheiden sind (und von LOGO tatsächlich auch unterschieden werden).

4.4.3 Die empirische Komponente der Theorie der Informationsverarbeitung

Die empirische Komponente A(IVS) der Theorie der Informationsverarbeitung beinhaltet die Menge der ,,intendierten Anwendungen“ dieser Theorie auf reale informationsverarbeitende Systeme, zu denen vor allem - wie bereits erwähnt - einerseits der Rechner, andererseits aber auch der Mensch zählt.

Unter Bezugnahme auf die im vorigen Abschnitt vorgestellte logische Komponente K(IVS) besteht die Menge A(IVS) aus einer konkret spezifizierbaren, *empirisch benennbaren* Teilmenge der partiellen potentiellen Modelle M_{PP}, formal

$$A(IVS) \subset M_{pp},$$

d.h. derjenigen Komponente des Strukturkerns K(IVS), die in ihren Individuenbereichen (der Eingabemenge 1 und der Ausgabemenge 0) und deren funktionaler Verknüpfung (mittels der Response-Funktion R) ausschließlich empirisch bestimmt ist. Die spezifizierende Bedingung der ,,empirischen Benennbarkeit“ von intendierten Anwendungen A(IVS) besagt, daß konkrete informationsverarbeitende Systeme angebbar sein müssen, ohne in deren Beschreibung auf die Theorie der Informationsverarbeitung selbst einzugehen. Im Falle des Rechners heißt das, seine „hardware“ zu beschreiben (zentrale Recheneinheit, periphere Speicher, Eingabe-Ausgabe-Geräte usw.) und die Implementierbarkeit bestimmter Programmiersprachen anzugeben (Alphabet, Grundvokabular, Syntax und Grammatik). Für den Menschen heißt das, ihn in seiner Gegenständlichkeit als psychologisches Subjekt zu beschreiben: als Lebewesen in einem biologischen und sozialen Lebenszusammenhang mit beobachtbaren Reaktionsformen (,,Verhalten“), Aktionsmöglichkeiten (,,Handeln“) und Erlebnisweisen („Motivationen und Kognitionen“).

Im Gegensatz zu der expliziten Bestimmbarkeit (durch Aufzählung) oder der impliziten Definierbarkeit (durch Angabe notwendiger und hinreichender Merkmale) von partiellen potentiellen Modellen (vgl. die definitorische Einführung von M_{pp} in Abschnitt 4.4.2) sind die intendierten Anwendungen A(IVS) in der Regel nur als ,,paradigmatische Beispiele“ B, als ,,typische

Exemplare“ informationsverarbeitender Systeme angebbar (vgl. Stegmüller, 1973, S. 195-207). Insbesondere muß es - für den Erfinder einer Theorie T bzw. für all diejenigen, die diese Theorie akzeptieren - eine *paradigmatische Beispielmenge* $B_o \subset A$ (mit $B_o \subset B$) geben, die *unverzichtbarer* Bestandteil für die Anwendbarkeit der Theorie T ist, so daß letztlich der nicht-reduzierbare Umfang einer Theorie mit

$$T = \langle K, B_o \rangle$$

formulierbar ist.

Die Frage ist nun: Was ist die paradigmatische Beispielmenge B_o(IVS) für die Theorie der Informationsverarbeitung? Eine umfassende Antwort auf diese Frage läßt sich sicher nicht geben, doch dürften sich - für den Bereich der Computer-Simulation und der KI-Forschung - jene frühen Modelle dazu zählen lassen, die beispielsweise bereits in dem wegweisenden Buch von Feigenbaum & Feldman (1963) versammelt sind: Maschinen, die

Schach (Newell, Shaw & Simon) und Dame (Samuel) spielen,
Theoreme aus Logik (Newell, Shaw & Simon) und Geometrie (Gelernter u. Mitarb.) beweisen,
natürlichsprachliche Fragen beantworten (Green u. Mitarb., Lindsay),
visuelle Muster erkennen (Selfridge & Neisser, Uhr & Vossler),
Probleme lösen (Newell & Simon),
sinnlose Silben (Feigenbaum) und sinnvolle Begriffe (Hunt & Hovland) lernen,
Entscheidungen unter Unsicherheit treffen (Feldman, Clarkson),
ja, sogar interpersonelles soziales Verhalten nachbilden (Gullahorn & Gullahorn).

Für den Bereich menschlicher Systeme der Informationsverarbeitung läßt sich die paradigmatische Beispielmenge B_o(IVS) weniger eindeutig angeben. Mit Sicherheit dazu zählen kann man all jene empirischen Versuche, die schon am Anfang der Computer-Simulation die Grundlage für die Modellentwicklung (wie beispielsweise für den „Logic Theorist“ und den „General Problem Solver“ von Newell, Shaw & Simon, 1957, 1959) bildeten. Man kann aber, auch wenn der Begriff der Information bzw. eine Theorie der Informationsverarbeitung noch unbekannt waren, all die früheren experimentalpsychologischen Untersuchungen, die im Gefolge des Behaviorismus entstanden (bis hin, wenn man will, zu den ersten psychophysischen Versuchen im 19. Jahrhundert), als paradigmatische Beispiele ansehen, sofern nur für das beobachtbare Eingabe-Ausgabe-Verhalten immer eine empirische Response-Funktion bestimmbar bleibt. Gerechterweise sollte man aber den Menschen nur unter jenen Aspekten als Paradigma der Informationsverarbeitung verstehen, die explizit auf eine entwickelte Theorie in dem hier dargestellten Sinne Bezug nehmen.

4.4.4 Der instrumentelle Gebrauch der Theorie der Informationsverarbeitung

Nach der strukturalistischen Theoriekonzeption konkretisiert sich der Umgang mit einer Theorie T auf die Frage, wie aufgrund eines gegebenen Strukturkerns K aus den intendierten Anwendungen A genau jene spezifiziert werden können, die für die augenblickliche Betrachtung interessieren. In der Regel gelten in bestimmten Anwendungen ganz spezifische Gesetzmäßigkeiten, die für andere Anwendungen nicht zutreffen, so daß deren Besonderheit durch die Angabe spezieller Nebenbedingungen herausgearbeitet werden muß. Dabei dient der vorgegebene Strukturkern, in dem die *allgemeinen,* in allen Anwendungen geltenden Nebenbedingungen formuliert sind, als Instrument für die Herausarbeitung der *speziellen,* nur in einer bestimmten Anwendung gültigen Nebenbedingungen.

Für den Bereich der Computer-Simulation (und auch der KI-Forschung) gilt beispielsweise als spezielle Nebenbedingung, daß sowohl die Eingabemenge I als auch die Ausgabemenge O eines IVS *Teilmenge* der internen Repräsentationsmenge S ist, deren Alphabet und Grundvokabular zudem noch von der verwendeten Programmiersprache abhängig ist. Für die Gültigkeit von Simulationsmodellen heißt das dann, daß diese sich notwendigerweise auf die Nachbildung menschlichen Sprachverhaltens bzw. auf die Darstellung sprachlich beschreibbaren nonverbalen Verhaltens beschränken müssen.

Für den Menschen als Gegenstand der Theorie der Informationsverarbeitung stehen dem andere spezielle Nebenbedingungen entgegen, die die Besonderheit der menschlichen Informationsverarbeitung bestimmen. Vor allem die Mehrkanaligkeit (Parallelität) der Enkodierung und Dekodierung von Information, möglicherweise aber auch eine Multiplizität der Repräsentation von Information im Gehirn bedingen Eigengesetzlichkeiten, die sich in einer Vielfalt psychischer Phänomene niederschlagen und sich von daher von den Eigengesetzlichkeiten eines Rechners unterscheiden werden. Zweifellos gibt es auch eine Reihe von Bewußtseinsfunktionen, die im Bereich der menschlichen Informationsverarbeitung über die einfache Prozessorfunktion der kognitiven Exekutive eines IVS hinausgehen (vgl. dazu Ueckert, 1980b).

Die konkrete Herausarbeitung solcher spezieller Nebenbedingungen - methodisch in Form von geeigneten Erweiterungen des Strukturkerns der Theorie (vgl. Stegmüller, 1973, S. 122-139) - ist Aufgabe einzelwissenschaftlicher Untersuchungen. Die Theorie der Informationsverarbeitung ist nach der strukturalistischen Theoriekonzeption nur die *formale* Rahmentheorie (vergleichbar etwa der Meßtheorie für die „Meßbarkeiten des Psychischen"), deren instrumenteller Gebrauch die Entwicklung inhaltlich-psychologischer Theorien für die einzelnen Gegenstandsbereiche menschlicher Informationsverarbeitung erleichtert.

Das Problem der Falsifizierbarkeit von Simulationsmodellen reduziert sich hierbei auf die Frage, ob ein mit der Modellentwicklung unternommener Anwendungsversuch der Theorie der Informationsverarbeitung als *erfolgreich* anzusehen ist oder nicht, d.h. ob ein konkret vorgegebenes Simulationsmodell noch zu der Menge der intendierten Anwendungen A(IVS) gehört. Im negativen Fall ist nicht die Theorie der Informationsverarbeitung - weder für den Menschen noch für den Rechner - falsifiziert, sondern nur der Versuch ihrer Anwendung an dem Modell gescheitert, was aber nicht ausschließt, daß ein erneuter, mit einem verbesserten Modell arbeitender Anwendungsversuch nicht doch noch erfolgreich sein wird.

In ihrer Übertragung auf den Rechner ist die Theorie der Informationsverarbeitung bisher stets erfolgreich angewendet worden, sofern die Programme das intendierte Modellverhalten zeigen konnten. In diesem Zusammenhang ist die Theorie der Informationsverarbeitung sogar prinzipiell nicht falsifizierbar: Die theoretischen Systemfunktionen eines potentiellen Modells der Informationsverarbeitung können in einem Rechner jederzeit in dem Sinne „empirisch“ gemacht werden, daß ihre Implementierung und Arbeitsweise durch geeignete Modellausgaben (z. B. über entsprechende „Trace“-Anweisungen) detailliert beobachtet und beurteilt werden kann. Diese Systemfunktionen bleiben zwar theoretisch im Rahmen der Theorie der Informationsverarbeitung, da sie nur *innerhalb* dieser Theorie die Erklärbarkeit der Informationsverarbeitung ermöglichen, für andere Theorien jedoch - beispielsweise die Automatentheorie oder die Systemtheorie - können sie als nicht-theoretische Größen behandelt werden.

Inwieweit die Theorie der Informationsverarbeitung auch im Bereich geistiger Tätigkeit des Menschen, für dessen kognitive Aktivität, als grundsätzlich nicht falsifizierbar betrachtet werden soll, ist eine offene Frage. Die Methode der Computer-Simulation wird jedoch - trotz aller möglicher und tatsächlicher Verschiedenheit zwischen maschineller und menschlicher Informationsverarbeitung - bevorzugtes Instrument der kognitiven Psychologie bleiben.

5. Kommentiertes Literaturverzeichnis

In das Literaturverzeichnis wurde neben den im Text erwähnten Titeln eine knappe Auswahl der wichtigsten Arbeiten zu den Forschungsgebieten der Computer-Simulation und der „künstlichen Intelligenz“ (KI) aufgenommen, um einen möglichst repräsentativen Querschnitt aus der Vielfalt der bisher erschienenen Literatur zu geben. Jedem Titel ist ein kurzer kommentierender Verweis auf Inhalt und Bezug der jeweiligen Arbeit beigegeben.

Literatur

Abelson, R. P. 1968. Simulation of social behavior. In G. Lindzey & E. Aronson (Eds): Handbook of social psychology. Vol. 2. Reading: Addison-Wesley. - Darstellung der Simulationsmethodik in ihrer Anwendbarkeit auf sozialpsychologische Fragestellungen unter besonderer Berücksichtigung der Validierbarkeit von Simulationsmodellen.

Anderson, J. R. 1976. Langtrage, memory, and thought. Hillsdale: Erlbaum. - Entwicklung des ACT-Simulationsmodells der menschlichen Wissensrepräsentation als kognitives Produktionssystem über einem propositionalen Netzwerk. Diskussion der Anwendbarkeit des Modells auf Inferenzprozesse, Lernen und Behalten, Verstehen und Erzeugen von Sprache und Induktion von Prozeduren.

Anderson, J. R. & Bower, G. H. 1973. Human associative memory. Washington: Winston. - Entwurf einer Theorie der Gedächtnisrepräsentation als propositionales semantisches Netzwerk (Vorläufer der ACT-Theorie von Anderson, 1976).

Apter, M. J. 1970. The Computer Simulation of behaviour. London: Hutchinson. - Allgemeinverständliche, inzwischen etwas veraltete Darstellung der Simulationsmethodik mit Diskussion von Anwendungen aus Bereichen des Lernens, des Problemlösens, des Mustererkennens, der Sprache und der Persönlichkeitstheorie bis hin zum Problem des Bewußtseins.

Bauer, W. 1973. Methodische Probleme der Computer-Simulation. In G. Reinert (Hg.): Bericht über den 27. Kongreß der DGfPs in Kiel 1970. Göttingen: Hogrefe. - Einführender Artikel in die Simulationsmethodik unter dem Aspekt der Kommunizierbarkeit, Validierbarkeit und Generalisierbarkeit von Simulationsmodellen.

Boden, M. A. 1977. Artificial intelligence and natural man. Hassocks: Harvester Press. - Ausführliche, nicht-technische Einführung in das Forschungsgebiet der KI mit einer umfassenden Darstellung der wichtigsten neueren Arbeiten aus den unterschiedlichsten Bereichen der KI-Forschung und einer eingehenden Relevanzdiskussion hinsichtlich psychologischer, philosophischer und sozialer Implikationen der KI-Forschung.

Cohen G. 1977. The psychology of cognition. London: Academic Press. - Eine systematische, breit gefächerte Einführung in die kognitive Psychologie mit einem methodologischen Kapitel zur Computer-Simulation.

Colby, K. M. 1975. Artificial Paranoia. A Computer Simulation of paranoid processes. New York: Pergamon Press. - Darstellung des PARRY-Programms zur Simulation paranoider Prozesse einschließlich einer Diskussion von Validierungsstudien zu dem Simulationsmodell.

Davis, R. & King, J. 1977. An overview of production systems. In E. W. Elcock & D. Michie (Eds): Machine intelligence 8. Chichester: Horwood. - Kurzgefaßte Darstellung der Konzeption von Produktionssystemen von einem mehr technisch-methodischen Standpunkt aus.

Dutton, J. M. & Starbuck, W. H. (Eds). 1971. Computer Simulation of human behavior. New York: Wiley. - Sammelband ausgewählter Arbeiten aus allen Be-

reichen der Computer-Simulation bis etwa zum Jahre 1970 und vollständige Bibliographie aller bis 1969 erschienenen einschlägigen Publikationen.

Ernst, G. W. & Newell, A. 1969. GPS. A case study in generality and problem solving. New York: Academic Press. - Vollständigste Darstellung des „General Problem Solver" als einem Simulationsmodell des menschlichen Problemlösens.

Feigenbaum, E. A. & Feldman, J. (Eds). 1963. Computers and thought. New York: McGraw-Hill. - Erster Sammelband zu den Bereichen der Computer-Simulation und der KI-Forschung (mittlerweile von historischem Wert).

Feurzeig, W., Lukas, G. & Grant, R. 1971. LOGO reference manual. The LOGO project NSF-C615. Cambridge, Mass.: Bolt, Beranek & Newman. - Benutzerhandbuch für die Programmiersprache LOGO in ihrer ersten Version.

Gregg, L. W. & Simon, H. A. 1967. Process models and stochastic theories of simple concept formation. Journal of Mathematical Psychology 4. - Darstellung von Prozeßmodellen zur Simulation der einfachen Begriffsbildung im Vergleich zu stochastischen Theorien hierzu aus dem Bereich der mathematischen Psychologie.

Harbordt, S. 1974. Computersimulation in den Sozialwissenschaften. 1. Einführung und Anleitung. 2. Beurteilung und Modellbeispiele. Reinbek: Rowohlt. - Allgemeinverständliche Einführung in die Simulationsmethodik von einem mehr soziologischen und weniger psychologischen Standpunkt aus.

Heinrich, H. C. 1978. Möglichkeiten der Computersimulation in der Psychologie. Psychologische Rundschau 29. - überblicksartikel über die theoretischen und praktischen Möglichkeiten der Simulationsmethodik mit Diskussion von Anwendungsbeispielen.

Hunt, E. B. 1969. Computer Simulation. Artificial intelligence studies and their relevance to psychology. Annual Review of Psychology 19. - Erster bibliographischer überblicksartikel über die Simulationsmethodik, ihre Grundlagen für die psychologische Modellbildung und ihre Beziehungen zur KI-Forschung.

Hunt, E. B. & Poltrock, S. E. 1974. The mechanics of thought. In B. H. Kantowitz (Ed.): Human information processing. Tutorials in performance and cognition. Hillsdale: Erlbaum. - Kurzgefaßte, aber systematische Einführung in die Simulationsmethodik unter Verwendung des Produktionssystem-Ansatzes (mit Beispielen von kognitiven Produktionssystemen).

Lenat, D. B. 1978. The ubiquity of discovery. Artificial Intelligence 9. - Diskussion der KI-Forschung am Beispiel des AM-Systems des Autors zum Entdecken von mathematischen Konzepten und Relationen der elementaren Zahlentheorie.

Lenat, D. B. 1979. On automated scientific theory formation. A case study using the AM program. In J. E. Hayes, D. Michie & L. I. Mikulich (Eds): Machine intelligence 9. Chichester: Horwood. - Ausführliche Darstellung des AM-System des Autors mit zahlreichen Beispielen.

McCarthy et al. 1962. LISP 1.5 programmer's manual. Cambridge, Mass.: M.I.T. Press. - Benutzerhandbuch für die Programmiersprache LISP in ihrer erstveröffentlichten Version.

McDermott, J. & Forgy, C. 1978. Production system conflict resolution strategies. In D. A. Waterman & F. Hayes-Roth (Eds): Pattern-directed inference systems.

New York: Academic Press. - Exemplarische Diskussion und Evaluation von Konfliktlösungsregeln in Produktionssystemen.

Miller, G. A., Galanter, E. & Pribram, K. H. 1960. Plans and the structure of behavior. New York: Holt, Winston & Rinehart. (Deutsch: Strategien des Handelns. Pläne und Strukturen des Verhaltens. Stuttgart: Klett, 1973.) - Umfassender Entwurf einer neuen Psychologie auf kybernetischer Grundlage, der geradezu eine programmatische Vorwegnahme der Entwicklung der neueren kognitiven Psychologie darstellt.

Newell, A., Shaw, J. C. & Simon, H. A. 1957. Empirical explorations with the Logic Theory Machine. A case study in heuristics. In E. A. Feigenbaum & J. Feldman (Eds): Computers and thought. New York: McGraw-Hill, 1963. - Darstellung und Diskussion eines der ersten Programme, des „Logic Theorist", zur Simulation von Prozessen des Problemlösens, hier im Bereich der Aussagenlogik.

Newell, A., Shaw, J. C. & Simon, H. A. 1958. Elements of a theory of human problem solving. The Psychological Review 65. - Erste, programmatische Darstellung einer Theorie des menschlichen Problemlösens auf der Grundlage der Modellierbarkeit von Denkprozessen mit Hilfe der Computer-Simulation.

Newell, A., Shaw, J. C. & Simon, H. A. 1959. Report on a general problem solving program. In Proceedings of the International Conference on Information Processing. Paris: UNESCO House. - Bericht über das erste, als reines Simulationsmodell des menschlichen Problemlösens entwickelte Computer-Programm, den ,,General Problem Solver" (GPS).

Newell, A. & Simon, H. A. 1963. Computers in psychology. In R. D. Luce, R. R. Bush & E. Galanter (Eds): Handbook of mathematical psychology. Vol. 1. New York: Wiley. - Ausführlicher Handbuchbeitrag über den Rechnergebrauch in der Psychologie, insbesondere in Form der Computer-Simulation (mit Grundlagendiskussion und Anwendungsbeispielen).

Newell, A. & Simon, H. A. 1972. Human problem solving. Englewood Cliffs: Prentice-Hall. - Umfassendste theoretische, methodische und empirische Darstellung des Simulationsansatzes und dessen Anwendung auf das menschliche Problemlösen in so unterschiedlichen Bereichen wie Kryptarithmetik, Logik und Schach.

Norman, D. A., Rumelhart, D. E. & LNR Research Group. 1975. Explorations in cognition. San Francisco: Freeman. (Deutsch: Strukturen des Wissens. Stuttgart: Klett, 1978.) - Entwurf einer Repräsentationstheorie menschlichen Wissens mit Hilfe der Konzeption von semantischen Netzen und Implementierung als Computer-Modell mit Anwendungen auf Sprache, Wahrnehmung und Problemlösen.

Ringle, M. (Ed.). 1979. Philosophical perspectives in artificial intelligence. Atlantic Highlands: Humanities Press. - Sammelband mit grundlegenden Themenstellungen und kritischen Stellungnahmen zur KI-Forschung.

Rychener, M. D. & Newell, A. 1978. An instructable production system. Basic design issues. In D. A. Waterman & F. Hayes-Roth (Eds): Pattern-directed inference systems. New York: Academic Press. - Entwurf eines lernenden, selbstmodifizierenden Produktionssystems.

Schank, R. C. & Abelson, R. P. 1977. Scripts, plans, goals, and understanding. An inquiry into human knowledge structures. Hillsdale: Erlbaum. - Darstellung

und Diskussion der sog. Skripttheorie menschlichen Sprachverstehens auf der Grundlage eines Computer-Modells über alltagsnahes Sprachhandeln.

Schank, R. C. & Colby, K. M. (Eds). 1973. Computer models of thought and language. San Francisco: Freeman. - Sammelband über den Bereich der KI-Forschung in ihrer Anwendbarkeit auf Sprache und Denken.

Simon, H. A. 1969. The sciences of the artificial. Cambridge, Mass.: M.I.T. Press. - Eine grundlegende Untersuchung der mit der Computer-Entwicklung verbundenen ,,Künstlichkeit“ von Wissenschaft und Welt, dargestellt an vier Themenbereichen: (1) Verstehen von natürlichen und künstlichen Welten; (2) Psychologie des Denkens: Einbettung des Künstlichen in Natur; (3) Die Wissenschaft vom Entwerfen: Erzeugung des Künstlichen; (4) Die Architektur von Komplexität.

Simon, H. A. 1979. Information processing models of cognition. Annual Review of Psychology 30. - Bibliographische Darstellung des Ansatzes der Informationsverarbeitung unter besonderer Berücksichtigung der Simulationsmethodik.

Simon, H. A. 1979a. Models of thought. New Haven: Yale University Press. - Sammelband über eine repräsentative Auswahl von Arbeiten des Autors und seiner Mitarbeiter zur kognitiven Psychologie von 1955 bis 1977.

Sloman, A. 1978. The Computer revolution in philosophy. Philosophy, science, and models of mind. Hassocks: Harvester Press. - Eine in philosophische und wissenschaftstheoretische Fragestellungen des Rechnergebrauchs in der KI-Forschung und in der kognitiven Psychologie eingehende Untersuchung der Computer-Metapher.

Stegmüller, W. 1973. Probleme und Resultate der Wissenschaftstheorie und Analytischen Philosophie. Band 2: Theorie und Erfahrung. Studienausgabe Teil D: Logische Analyse der Struktur ausgereifter physikalischer Theorien. ,,Non-statement view“ von Theorien. Studienausgabe Teil E: Theoriendynamik. Normale Wissenschaft und wissenschaftliche Revolutionen. Methodologie der Forschungsprogramme oder epistemologische Anarchie? Berlin: Springer. - Detaillierte Darstellung der strukturalistischen Theoriekonzeption (,,non-Statement view“) am Beispiel physikalischer Theorienbildungen und wissenschaftstheoretischer Paradigmenvorstellungen.

Stegmüller, W. 1980. Neue Wege der Wissenschaftsphilosophie. Berlin: Springer. - Weiterführende Arbeiten zur strukturalistischen Theoriekonzeption (,,non-statement view“).

Tomkins, S. S. & Messick, S. (Eds). 1963. Computer Simulation of personality. New York: Wiley. - Sammelband über erste Arbeiten zur Anwendung der Simulationsmethodik auf Bereiche der Persönlichkeitsforschung.

Turing, A. M. 1950. Computing machinery and intelligence. Mind 59. (Reprinted in E. A. Feigenbaum & J. Feldman (Eds): Computers and thought. New York: McGraw-Hill, 1963. Deutsch: Kann eine Maschine denken? Kursbuch 8, 1967.) - Berühmte Arbeit zu der Frage, ob Computer „denken“ können, mit einer Diskussion einer Reihe grundsätzlicher Gegenargumente.

Ueckert, H. 1980a. Cognitive production systems. Toward a comprehensive theory of mental functioning. In F. Klix & J. Hoffmann (Eds): Cognition and memory. Knowledge and meaning comprehension as functions of memory. Amsterdam:

North-Holland Publishing Company. - Diskussion der Produktionssystem-Konzeption als einer umfassenden Theorie kognitiver Aktivität.

Ueckert, H. 1980b. The cognitive executive. From artificial intelligence toward a psychological theory of consciousness. In Proceedings of the XXIInd International Congress of Psychology in Leipzig. - Anwendung der Produktionssystem-Konzeption auf die Bewußtseinsproblematik der menschlichen Informationsverarbeitung.

Ueckert, H. 1980c. Das Lösen von Intelligenztestaufgaben. Eine test- und meßkritische Untersuchung. Göttingen: Hogrefe. - Entwicklung eines prozeßorientierten Ansatzes zur Intelligenzforschung auf der Grundlage der Produktionssystem-Konzeption und Diskussion der Axiomatisierbarkeit der Intelligenztestung als additiv-verbundener Messung.

Ueckert, H. & Rhenius, D. (Hg.). 1979. Komplexe menschliche Informationsverarbeitung. Beiträge zur Tagung „Kognitive Psychologie" in Hamburg 1978. Bern: Huber. - Bericht über die erste Fachtagung zur kognitiven Psychologie in der BRD mit sechs Themenschwerpunkten: (1) Kognitive Psychologie als integrativer Bestandteil psychologischer Grundlagenforschung; (2) Kognitive Organisation der menschlichen Informationsverarbeitung; (3) Informationelle Produktionssysteme und Computer-Simulation; (4) Sprachliche Kognition und semantisches Gedächtnis; (5) Entscheidungstheoretische Ansätze zur kognitiven Psychologie; (6) Anwendungsfragen der kognitiven Psychologie.

Uhr, L. 1973. Pattern recognition, learning, and thought. Computer-programmed models of higher mental processes. Englewood Cliffs: Prentice-Hall. - Eine ganz auf die methodischen Fertigkeiten des Programmierens (in der hierzu vom Autor konzipierten Programmiersprache EASEY-1) angelegte Einführung in den Bereich des Computer-gestützten Modelliereris von Prozessen des Wahrnehmens, Lernens und Denkens.

Vukovich, A. F. 1967. Die Simulierung des Problemslösens auf logischen Automaten. In F. Merz (Hg.): Bericht über den 25. Kongreß der DGfPs in Münster 1966. Göttingen: Hogrefe. - Erster deutschsprachiger überblicksartikel über die Computer-Simulation in der psychologischen Forschung unter theoretischen und methodologischen Aspekten.

Waterman, D. A. & Hayes-Roth, F. (Eds). 1978. Pattern-directed inference systems. New York: Academic Press. - Sammelband mit einer systematischen Auswahl von Arbeiten auf der Grundlage der Produktionssystem-Konzeption mit Anwendungen im Bereich der Computer-Simulation und in der KI-Forchung.

Wegener, H. & Dörner, D. 1973. Simulation als Forschungstechnik. Bericht über ein Symposion. In G. Reinert (Hg.): Bericht über den 27. Kongreß der DGfPs in Kiel 1970. Göttingen: Hogrefe. - Zusammenfassende Darstellung einer Diskussion über die Computer-Simulation als Forchungsinstrument in der Psychologie.

Weizenbaum, J. 1976. Computer power and human reason. From judgment to calculation. San Francisco: Freeman. (Deutsch: Die Macht der Computer und die Ohnmacht der Vernunft. Frankfurt am Main: Suhrkamp, 1977.) - Eine sehr kritische, kompetente Diskussion der Computer-Metapher und ihrer Verwendung in Forschung und Praxis.

Wender, K. F., Colonius, H. & Schulze, H.-H. 1980. Modelle des menschlichen Gedächtnisses. Stuttgart: Kohlhammer. - Kurzgefaßte Darstellung der wichtigsten Ansätze zur Gedächtnisrepräsentation auf der Grundlage von semantischen Netzen.

Wexler, K. 1978. A review of John R. Anderson's Language, Memory, and Thought. Cognition 6. - Ausführliche und kritische Besprechung der ACT-Theorie von Anderson (1976).

Winograd, T. 1972. Understanding natural language. Cognitive Psychology, Whole Number 3. (Also published by Academic Press, New York.) - Darstellung eines der ersten KI-Systeme zum natürlichsprachlichen Verstehen auf der Grundlage einer Computer-simulierten „Mini-Welt".

Winston, P. H. 1977. Artificial intelligence. Reading: Addison-Wesley. - Eine sehr instruktiv geschriebene und illustrative Einführung in den Forschungsbereich der „künstlichen Intelligenz" mit besonderer Berücksichtigung des Programmierens (in der Programmiersprache LISP).