Kleine Computersimulationen: Computerunterstützte Theoriebildung mit Beispielen aus der Psychologie.

Hansruedi Kaiser & Beat Keller
1991, Bern: Huber

0 Wozu Computersimulation?

Dieses Buch ist ein Versuch, unsere Erfahrungen mit dem Einsatz von Computersimulationen in der psychologischen Forschung für andere zugänglich zu machen. Mit Computersimulationen meinen wir das Schreiben von Computerprogrammen mit dem Ziel, dass das Verhalten des entsprechend programmierten Computers ausschnittweise (in einem noch zu definierenden Sinn) dem Verhalten von Menschen gleicht.

Warum sollte ein Psychologe oder eine Psychologin daran interessiert sein, eine Computersimulation zu machen? Es gibt viele Gründe dafür, und wir wollen sie hier nicht erschöpfend auflisten. Der Leser findet sie explizit und implizit verstreut über den ganzen folgenden Text. Explizit erwähnt sind sie dann, wenn wir in irgendeiner Form darauf hinweisen, welche spezifischen Vorteile eine Computersimulation gegenüber einer "Papier-und-Bleistift-Theorie" hat (z.B. im Kapitel 8). Implizit stösst der Leser immer wieder dann darauf, wenn er sich überlegt, was er persönlich aus einer Computersimulation nach unseren Vorschlägen gewinnen würde.

Als Motivationshilfe möchten wir aber hier doch die wichtigsten Gründe vorstellen, die für Computersimulationen in der psychologischen Forschung sprechen.

Computersimulationen kann man als Beschreibung psychischer Prozesse mit Hilfe eines bestimmten Mediums (Programmiersprache und Computer) betrachten. In diesem Sinn unterscheiden sie sich nicht von anderen derartigen Beschreibungen, wie sie konventioneller mit Papier und Bleistift festgehalten werden. Und sie können genauso wie diese von der bescheidenen Beschreibung eines Einzelfalls bis hin zur grossen Theorie reichen. Um ein Gefühl dafür zu bekommen, welche Hilfe eine Beschreibung in Form einer Computersimulation anstelle einer "Papier-und-Bleistift-Theorie" sein könnte, lohnt es sich deshalb, sich einmal vorzustellen, man müsste auf die üblichen Hilfsmittel - eben Papier und Bleistift - verzichten.

Selbstverständlich ist da einmal die kommunikative Funktion jeder schriftlichen Darstellung, die wegfallen würde. Hinzu kommen aber viele Funktionen, die das Geschriebene für die Autorin/den Autor selbst hat. Diese Funktionen stehen meist im Zusammenhang mit Gedächtnisproblemen. Zum Beispiel erlaubt es die Niederschrift einer umfangreichen Beschreibung (eines Modells oder einer Theorie, je nach formalem Anspruch) früher Geschriebenes mit später Geschriebenem zu vergleichen, ohne dass man alles wörtlich und präzise im Gedächtnis präsent haben muss. So wird es im Prinzip möglich, relativ grosse Gedankengebäude auf logische Stimmigkeit und Vollständigkeit hin zu überprüfen. Durch hin- und zurückblättern kann man im Prinzip feststellen, ob man sich nirgends widerspricht. Und indem man das Geschriebene sorgfältig immer wieder durchgeht, kann man klären, ob man nichts vergessen hat.

Wir sagen aber ausdrücklich "im Prinzip", denn "Papier ist geduldig" und ob das, was man auf Seite vier schreibt mit dem, was auf Seite sechs steht, irgendwie übereinstimmt, ist ihm egal. Es schreit nicht auf oder kräuselt sich vor Entsetzen, wenn Widersprüche auftreten, sondern es bleibt dem Schreibenden überlassen, die Zusammenhänge zu überprüfen.

Und so ist jede "Papier-und-Bleistift-Theorie" nur insoweit logisch stimmig und vollständig, wie der Autor oder die Autorin fähig war, die notwendige Kontrolle vorzunehmen - was mit zunehmendem Umfang und zunehmender Komplexität der Beschreibung immer aufwendiger wird und bald einmal an Grenzen stösst, da der kognitive Apparat Normalsterblicher seine Leis-

tungsgrenzen hat und Zusammenhänge ab einer gewissen Komplexität nicht mehr vollständig durchdenken kann. Zudem ist es aber auch einfach schwierig, gegenüber den eigenen Geisteskindern beliebig kritisch zu sein. Und so übersieht man halt (oft?) logische Widersprüche und Lücken im Text, die in den eigenen Gedanken mit stillschweigenden Vorannahmen überbrückt werden.

Wertvolle Hilfe leistet da natürlich der kritische Kollege. Aber auch ihm sind Grenzen gesetzt. Und so gibt es eine - recht niedrige - Obergrenze an Umfang und Komplexität, oberhalb der sich "Papier-und-Bleistift-Theorien" nicht mehr mit Sicherheit auf logische Vollständigkeit und Stimmigkeit überprüfen lassen. Dass wir glauben, dass diese Obergrenze recht niedrig ist, entspringt einerseits den Erfahrungen mit unserem eigenen Denkapparat (und der Leser ist aufgerufen, sich in dieser Hinsicht einmal selbst kritisch zu betrachten). Andererseits fällt aber auch auf, dass sämtliche einigermassen verbreiteten psychologischen Theorien von seltener Einfachheit sind.

Diese Beschränkung erweist sich unseres Erachtens in der psychologischen Forschung als echtes Problem. Denn man kann sich zwar darüber streiten, wie gross der Erkenntnisgewinn der ersten hundert Jahre psychologische Forschung wirklich ist. Eins scheint aber in dieser Zeit klar geworden zu sein: Psychologische Prozesse lassen sich kaum je durch ein paar wenige "wenn.. dann.. " Beziehungen (wie etwa "wenn Frustration dann Aggression") beschreiben. Wenn auch unklar sein mag, wie eine zufriedenstellende psychologische Beschreibung (Theorie, Modell, etc.) auszusehen hat, so ist doch sicher, dass sie auf jeden Fall eine Vielzahl interagierender Einflüsse und komplexe, dynamische Abläufe berücksichtigen muss. Es wäre deshalb schön, wenn ein Instrument zur Verfügung stehen würde, mit dessen Hilfe die Komplexitätsschranke von "Papier-und-Bleistift-Theorien" durchbrochen werden könnte. Und genau solch ein Instrument stellt der Computer dar, wenn man ihn als Medium benutzt, um darin (Prozess–) Beschreibungen darzustellen.

Oberflächlich gesehen besteht der Unterschied zwischen einer Computersimulation und einer "Papier-und-Bleistift-Theorie" darin, dass man die Beschreibung nicht in einer natürlichen Sprache mit Hilfe von Papier und Bleistift fixiert, sondern in einer Programmiersprache in einem Computer festhält. Ist die Programmiersprache geeignet gewählt, dann ist dieser Unterschied gar nicht so gross. Wesentlicher ist aber nun, dass diese Beschreibung in Form eines Programms auf dem Computer "laufen" kann. D.h. der Computer fasst - im Gegensatz zum geduldigen Papier - den (Programm-)Text als Aufforderung auf, etwas zu tun. Er zieht aus dem Geschriebenen logische Konsequenzen und bringt verschiedene Stellen des "Textes" zueinander in Beziehung.

Das mag eine überraschende Darstellung dessen sein, was in einem Computer vorgeht, denn üblicherweise stellt man sich wohl vor, dass der Text aus Befehlen besteht, die der Computer Schritt um Schritt abarbeitet. Dies ist aber eine zu enge und einseitige Sichtweise. Angemessener ist das Bild, dass die Aufgabe des Computers darin besteht, aus dem "Text" (der ganz unterschiedliche Formen annehmen kann) gewisse logische Konsequenzen zu ziehen.

Durch diesen Einsatz des Computers erhält man Kontrollinformationen zur logischen Stimmigkeit des "Textes", die man sonst selbst erarbeiten müsste - oder eben gar nicht erarbeiten könnte. Im Wesentlichen ergeben sich zwei Arten von Fehlerinformationen: Entweder läuft das Programm überhaupt nicht oder es produziert ein unerwartetes Resultat. Dies kann natürlich in beiden Fällen das Resultat trivialer Programmierfehler sein, vergleichbar mit Rechtschreibefehlern im "Papier-und-Bleistift" Fall. Sind diese aber einmal behoben, dann erhält man so wichtige Hinweise.

Läuft das Programm gar nicht, dann ist dies oft ein Hinweis darauf, dass man einen entscheidenden Teil der "Beschreibung" vergessen hat. Es kann sein, dass ein ganzer Teilprozess fehlt, es kann aber auch sein, dass dem Programm "Wissen" fehlt, das es zur Bearbeitung seiner Aufgabe braucht. Derartige Fehler treten meist dann auf, wenn man sich über die eigenen impliziten Vorannahmen noch zu wenig klar ist. Der Computer macht dann sanft darauf aufmerksam. Produziert das Programm hingegen falsche Resultate, dann lässt sich daran erkennen, dass sich irgendwo bei der Formulierung der "Beschreibung" Fehler eingeschlichen

haben, entweder aus Flüchtigkeit oder weil man nicht in der Lage war, die Konsequenzen korrekt zu Ende zu denken.

Neben dieser Kontrolle hat die Verwendung des Computers als Darstellungsmedium aber auch noch andere Vorteile. Der Einsatz als Kontrollinstrument setzt voraus, dass man eine Vorstellung davon hat, welches die logischen Folgen des Geschriebenen sind, so dass man dann diese Vorstellungen mit dem vergleichen kann, was der Computer tatsächlich produziert. Es kann aber auch sein, dass man gar nicht in der Lage ist, das Resultat in allen Aspekten vorherzusagen, und dass man deshalb den Computer einmal dazu benutzt, festzustellen, welche Konsequenzen z.B. ein bestimmtes Prozessmodell überhaupt hat.

Um es also nochmals kurz zusammenzufassen: Computersimulation ist der Versuch, Gedanken, Modelle, Theorien, etc. in einer Form aufzuschreiben, die es dem Computer ermöglicht, auf logische Konsequenzen des Geschriebenen aufmerksam zu machen. Da er dies besser und gründlicher kann, als normalsterbliche Forscher und Forscherinnen, stellt die Computersimulation ein sehr wertvolles Instrument im Rahmen der Theoriebildung dar. Die folgenden Seiten handeln nun davon, wie man am besten vorgeht, um in den Genuss dieser Hilfe zu gelangen.

Wir möchten dabei auch versuchen, die Angst davor zu nehmen, dass es sich bei einer Computersimulation um etwas ungeheuer Aufwendiges und Komplexes handelt. Computersimulationen können zwar sehr aufwendig werden; sowohl was die notwendige Computerleistung (Hardware) betrifft wie auch im Hinblick auf die Arbeit, die dahinter steckt. Und wenn man sich ein bisschen in der Literatur umschaut, entsteht schnell der Eindruck, dass solch aufwendige Projekte die Regel und damit wohl unumgänglich sind. Dieser Eindruck ist aber nur bedingt richtig, denn wie so oft hängt auch hier vieles davon ab, mit welchem Ziel man an die Arbeit geht. Will man im Sinne einer "künstlichen Intelligenz"-Leistung ein Programm schreiben, das schnell und effizient eine bestimmte Leistung unter realen Bedingungen erbringen kann, dann wird man allerdings weder bei der Hardware noch beim Arbeitsaufwand sparen dürfen.

Setzt man aber Computersimulation ein, um im Rahmen psychologischer Theoriebildung seine eigenen Überlegungen auf logische Stimmigkeit und Vollständigkeit zu überprüfen - so wie wir das hier vorschlagen - dann kann man auch mit recht bescheidenen Mitteln Interessantes erreichen. Ja uns erscheint es sogar von Vorteil, wenn man sich auf eine derartige Simulation im kleinen Rahmen - eine Mini-Computersimulation oder MINCS, wie wir sie im Folgenden der Einfachheit halber nennen wollen - beschränkt. Denn bei einer MINCS besteht weniger die Gefahr, dass man ob all der Faszination, die ein Computer zweifellos ausüben kann, die Psychologie vergisst.

Eine Beschränkung ist auch deshalb leicht möglich, weil meist nicht erst die vollständig elaborierte Simulation, sondern bereits die Etappen auf dem Weg dorthin, zu wesentlichen Einsichten führen. Oder wie Johnson-Laird es formuliert hat: "For a cognitive scientist, the single most important virtue of programming should come not from a finished program itself, or what it does, but rather from the business of developing it. Indeed the aim should be.... to force the theorist to think again." Und: "The development of small-scale programs to explore part of a general theory can be a genuinely dialectical process leading to new ideas both about the theory and even about how to test it experimentally." (Johnson-Laird, 1981, S. 186)

1 Unterschiedliche Arten von Computersimulationen

1.1 Geschichtliches

Genau genommen ist die Idee der Computersimulation älter als der Computer selbst. Denn schon der Bau der allerersten Computer hatte zum Ziel, Leistungen, die man bisher nur als Produkte menschlichen Denkens kannte, durch einen Mechanismus zu "simulieren". Ja schon etliche Vorläufer des Computers, wie etwa die Rechnungsmaschine, waren das Resultat solcher Versuche. Einerseits ging es dabei selbstverständlich darum, die entsprechende Leistung überhaupt zu erbringen. Zum zweiten wurde die Mechanisierung der Vorgänge aber auch genutzt, um besser zu verstehen, was z.B. beim "Rechnen" geschieht.

Diese zweite Möglichkeit gelang bereits sehr früh zur Anwendung. A. Turing konstruierte 1937 (also immer noch einige Jahre bevor es funktionstüchtige Computer gab) in Gedanken einen besonders einfachen, idealisierten Computer (die Turing-Maschine), den er benutzte, um diverse mathematische Fragen zu bearbeiten. Unter anderem konnte er zeigen, dass jedes Verfahren zur Manipulation von Symbolen, das sich mit endlichen Mitteln exakt beschreiben lässt, auf einer solchen Maschine dargestellt werden kann. Dieses Resultat bedeutet, dass jedes Modell, das die Zustände des Originales durch Symbole beschreibt und in dem die Veränderungen von Zustand zu Zustand explizit als Regeln und Gesetze angegeben sind, auf einer Turing-Maschine (und damit auf jedem modernen Computer) "laufen" kann.

Computersimulationen wurden und werden dann auch für ein breites Spektrum von Aufgaben eingesetzt. Das reicht von der Simulation des Verhaltens eines neuen Flugzeugtyps beim Landeanflug über Untersuchungen darüber, welchen Effekt die Einführung eines neuen Postverteilzentrums auf den vorweihnachtlichen Paketverkehr hat, bis zur Darstellung menschlichen Verhaltens.

Letzteres wurde historisch gesehen sogar sehr früh in Angriff genommen. Das braucht nicht zu überraschen, denn von der Feststellung, dass sich der Computer gut als äusserst schnelle Rechnungsmaschine missbrauchen lässt, bis zur Frage, ob der Computer neben rechnen auch noch andere Dinge tun kann, die bisher als spezifisch menschliche Leistungen betrachtet wurden, war es kein grosser Schritt. Und bereits 1950 widmete wieder A. Turing einen Artikel der Frage, ob Computer intelligent sind. Er schlug zur Beantwortung dieser Frage ein pragmatisches Vorgehen vor (den Turing-Test): Ein neutraler Tester sitzt allein in einem Raum und ist über zwei Fernschreiber mit zwei anderen Räumen verbunden. Am anderen Ende der einen Leitung sitzt ein Mensch, die zweite ist an einen Computer angeschlossen. Der Tester darf beiden Gesprächspartnern beliebige Fragen stellen. Der Computer soll als intelligent gelten, wenn der Tester nicht in der Lage ist zu entscheiden, welche Leitung zum Computer und welche zum Menschen führt. (Bisher wurde noch für kein Computerprogramm der Anspruch erhoben, dass es diesen Test in seiner allgemeinen Form bestehen würde. Es ist auch kein solches in Sicht und die Anzahl derer, die daran glauben, dass irgendwann ein entsprechendes Programm auftauchen wird, ist heute sicher kleiner als zu Turings Zeiten.)

In den Fünfzigerjahren entstand dann ein Forschungsbereich, der heute als Künstliche Intelligenz bekannt ist (KI oder auch AI: Artificial Intelligence). Zu Beginn wurde an drei unterschiedlichen Fronten versucht, Computer durch geeignete Programmierung zu menschlichen Leistungen zu bringen: Automatische Sprachübersetzung, Problemlösen (inklusive Schachspielen) und Mustererkennung.

1.1.1 Sprache

Die Vorläufer der Sprachübersetzungsversuche gehen auf den zweiten Weltkrieg zurück, wo Computer erfolgreich dazu eingesetzt wurden, die Verschlüsselungscodes gegnerischer Geheimdienste zu brechen (vgl. Barr & Feigenbaum, 1981). Es lag nahe, darin eine Art Übersetzung zu sehen und in Analogie dazu zu erwarten, dass die Übersetzung aus einer natürlichen Sprache in eine andere durch einen Computer ebenfalls möglich sein sollte. 1947 wurde das

erste Programm geschrieben, das in der Lage war, Wörter in einem gespeicherten Wörter-"Buch" nachzuschlagen und sieben Jahre später schrieb Oettinger das erste Programm, das in der Lage war, Wort-für-Wort Übersetzungen von Russisch ins Englische durchzuführen, indem es jedes Wort in einem derartigen elektronischen Wörterbuch "nachschlug". Da allerdings die meisten Wörter keine eindeutige Übersetzung hatten, wurden jeweils sämtliche möglichen Bedeutungen als Liste ausgedruckt. Es blieb dem Leser überlassen, aus dieser Aneinanderreihung einen verständlichen Text zu konstruieren (Oettinger, 1955). Die Leistung dieses Programmes war äusserst bescheiden. Dies zeigte sich unter anderem, wenn man einen englischen Text zuerst ins Russische und dann wieder ins Englische rückübersetzen liess. Meist bestand keine grosse Ähnlichkeit zwischen dem ursprünglichen Text und der Rückübersetzung. (Z.B. soll das Programm einmal den Satz "Der Geist ist willig aber das Fleisch ist schwach." in "Der Vodka ist ok aber das Steak ist miserabel." rückübersetzt haben.)

Dabei blieb es dann für viele Jahre, denn es zeigte sich - sehr zur Überraschung der Beteiligten - dass auch ein Computer-Programm einen Text zuerst "verstehen" muss, bevor es ihn sinnvoll übersetzen kann. Als Konsequenz daraus verschob sich das Interesse vom allgemeinen Übersetzungsprogramm zu Programmen, die Wissen über einen sehr beschränkten Ausschnitt der Welt verfügten (microworld), diesen also "verstanden" und somit dann auch in der Lage waren, darüber ein vernünftiges Gespräch zu führen. Das berühmteste einer ganzen Reihe solcher Programme ist SHRDLU von Winograd (1972). SHRDLU simuliert einen Roboter, der mit seiner Hand Klötze, die auf einem Tisch stehen, manipulieren kann. Und diese "blocks-world" ist auch der Gesprächsgegenstand, über den man sich mit SHRDLU unterhalten kann. Es ist in der Lage, auf Wunsch Klötze zu verschieben, aufeinander zu stapeln, etc. Es kann aber auch Fragen über die Klötze und ihre Stellungen zueinander, sowie über die eigenen Handlungen beantworten.

Zu Beginn der siebziger Jahre kam es dann zu neuen Versuchen, doch noch ein Übersetzungsprogramm zu schreiben. Wilks (1973) und Schank (1975) realisierten je Systeme, die eine Idee von Weaver aus dem Jahr 1949 (Barr & Feigenbaum, 1981, p. 234) aufgreifen. Hier wird ausgehend vom Ursprungstext zuerst eine sprachfreie Repräsentation der Bedeutung dessen, was im Text gesagt wird, erstellt (Weaver nannte diese Zwischenrepräsentation "Interlingua"). Und diese Bedeutungsrepräsentation dient dann als Grundlage um den Zieltext zu generieren.

Etwa zur selben Zeit wagte man sich auch an das Problem des "Verstehens" gesprochener Sprache ("speech" im Gegensatz zu "language") heran. Mit Verstehen ist hier gemeint, dass das Programm z.B. eine Frage, die sich auf seinen Wissensbereich bezieht, beantworten kann. Als besonders interessant, da dort erstmals viele heute geläufige AI-Techniken erprobt wurden, seien HEARSAY-I und HEARSAY-II hervorgehoben. HEARSAY-I war mit einem Schachprogramm verbunden und konnte gesprochene Anweisungen, welcher Zug als nächstes auszuführen war, verstehen (Reddy et al., 1976). HEARSAY-II war in der Lage, Fragen bezüglich einer Literatur-Datenbasis zu beantworten (Erman & Lesser, 1980). HEARSAY-II verfügte über ein Vokabular von rund tausend Wörtern.

Die verschiedenen, hier nur kurz angedeuteten Ansätze wurden in der Zwischenzeit natürlich weiterentwickelt, so dass heute Programme geschrieben werden können, die relativ gut Sprache "verstehen", solange mit eingeschränktem Vokabular über einen sehr begrenzten Ausschnitt der Welt gesprochen (oder geschrieben) wird. Daran wird sich aller Voraussicht nach auch in nächster Zeit nichts ändern.

1.1.2 Problemlösen und Spielen

Einer der frühesten Versuche, Computer Probleme "lösen" zu lassen, waren die Überlegungen von Shannon (1950) zu einem Schachprogramm. 1958 publizierten dann Newell, Shaw und Simon (Newell, Shaw & Simon, 1963a) ein Programm, das tatsächlich, wenn auch eher schlecht als recht, korrekt Schach spielen konnte. Seither haben Schachprogramme stetig Fortschritte gemacht und sind heute in der Lage, relativ gutes Schach zu spielen, auch wenn sie von menschlichen Spitzenspielern noch immer geschlagen werden. Wesentlich schneller

erfolgreich waren die von Samuel geschriebenen Dame-Programme, die bereits 1959 ähnlich gut spielten, wie die besten menschlichen Spieler (Samuel, 1963).

Vor allem Newell, Shaw und Simon waren aber auch an ernsthafteren Problemen interessiert. 1956 stellten sie ihren "Logic Theorist" vor, ein Programm, dass in der Lage war, etwa zwei Drittel der Theoreme in Whitehead und Russells "Principia Mathematica" aus den Axiomen abzuleiten (Newell, Shaw & Simon, 1963b). Interessant an diesem Programm war unter anderem, dass seine Autoren sogenannte "Heuristiken" einsetzten. Sie gaben dem Programm eine Art Faustregeln mit, wie man zu einem Beweis gelangen könnte. Diese Faustregeln sind in manchen Situationen wirksam, in manchen versagen sie aber auch. Es handelte sich somit um den ersten Versuch, ungenaues und unvollständiges Wissen in einem Computerprogramm abzubilden. Ein Jahr später begannen die gleichen Autoren ihre Arbeit am sogenannten "General Problem Solver" (GPS), einem Programm, das von der Zielsetzung her hätte in der Lage sein sollen, beliebige Probleme zu lösen (Newell & Simon, 1963). Ihre Absicht war aber nicht nur, ein funktionstüchtiges Programm zu schreiben, sondern sie versuchten, explizit das Vorgehen menschlicher Problemlöser zu modellieren. Die Leistungen, die der GPS erbrachte, waren eher bescheiden, er diente aber als Grundlage für das berühmte Buch "Human Problem Solving" von Newell und Simon (1972).

Da der Begriff "Problemlösen" zu allgemein ist (auch das "Verstehen" eines Textes kann man als "Problemlösen" betrachten), sind die Programme, die die Tradition des GPS fortsetzen, später v.a. unter der Bezeichnung "Planen" bekannt geworden. Eines der wichtigsten Programme dieser Kategorie ist NOAH von Sacerdoti (1974), das den Zusammenbau von Geräten (z.B. Waschmaschinen) planen kann. Eine der zentralen Schwierigkeiten, mit denen sich solche Planungsprogramme auseinandersetzen müssen, ist die, dass sehr oft scheinbar unabhängige Teilaktionen sich gegenseitig beeinflussen. Beim Zusammenbau einer Waschmaschine könnte es z.B. sein, dass die Montage des Heizsystems vor der Montage der Pumpe erfolgen muss, weil sonst der Ort, an dem das Heizsystem befestigt werden muss, nicht mehr zugänglich ist. Dies bedingt, dass das Programm nicht nur über generelle Planungsstrategien verfügt (wie etwa der GPS), sondern auch über einiges aufgabenspezifisches Wissen, insbesondere über die Abhängigkeit von Teilzielen.

Die Erkenntnis, dass es kaum Aufgaben gibt, die sich nur mit generellen Strategien bewältigen lassen, sondern dass immer eine grosse Portion bereichsspezifisches Wissen über die Randbedingungen etc. der Aufgabe notwendig ist, führte ab Ende der sechziger Jahre zur Entwicklung sogenannter Expertensysteme. Etwas überspitzt formuliert bestehen Expertensysteme nur noch aus bereichsspezifischem Wissen zu einem eng umgrenzten Wissensbereich (korrekter wäre deshalb wohl die Bezeichnung "Fachidioten-Systeme"). Das wohl bekannteste dieser Systeme ist MYCIN (Shortliffe, 1976). MYCIN ist spezialisiert auf die Diagnose von Infektionskrankheiten und seine Leistungen scheinen vergleichbar mit denen menschlicher Spezialisten zu sein (Yu et al., 1979).

Expertensysteme sind die bisher einzigen Produkte der KI, die in grossem Umfang praktisch eingesetzt werden. Zurzeit werden jährlich hunderte von neuen Expertensystemen entwickelt, die ihren Einsatz in der Industrie, in Banken, etc. finden. Der Stand der Dinge ist hier ähnlich wie oben für die "Sprachverarbeitung" beschrieben. Es ist zurzeit möglich, Programme zu schreiben, die in einem eng begrenzten Gebiet gute Leistungen erbringen. Programme, die ein breiteres Spektrum von Aufgaben bewältigen können, sind aber auch hier nicht in Sicht.

1.1.3 Mustererkennung

Die dritte historische Wurzel der KI ist das Gebiet der Mustererkennung. Menschen sind in der Lage, Muster - wie z.B. handgeschriebene Buchstaben - wiederzuerkennen, auch wenn diese beträchtlich verzerrt wurden. Das Lesen von Handschrift ist ein eindrückliches Beispiel dafür.

1959 stellte Rosenblatt sein "Perceptron" vor (Minsky & Papert, 1969). Das interessante am Perceptron war weniger, welche Muster es erkennen bzw. klassifizieren konnte, sondern vielmehr die Art, wie das Programm organisiert war. Seine Grundeinheit war eine Art abstraktes Neuron, eine stark vereinfachte Simulation einer Nervenzelle. Diese Neuronen wurden zu

mehr oder weniger grossen Netzen verbunden und die Leistung des Programms ergab sich aus ihrem Zusammenwirken. Diese Idee der Simulation nicht nur auf der "Verhaltensebene" (das Programm "verhält" sich intelligent), sondern auch auf der "Architekturebene" (ein Netz abstrakter Neuronen als direkte Simulation des Gehirns) geht auf einen wichtigen Artikel von McCulloch und Pitts (1943) zurück. Diese konnten demonstrieren, dass und wie abstrakte Neuronennetze sich als "Computer" verwenden lassen.

Dieser Ansatz (heute wird er "Konnektionismus" genannt) wurde von verschiedenen Forschern weiterverfolgt. Unter anderem hat ihn Kohonen (Kohonen, Mäkisara & Saramäki, 1984) zur Wahrnehmung gesprochener Sprache eingesetzt. Daneben gab und gibt es aber auch weiterhin Versuche, Mustererkennung mit Methoden zu bearbeiten, wie sie auch die Problemlöseprogramme etc. anwenden ("klassische KI" als Abgrenzung zum "Konnektionismus"). Die oben bereits angesprochenen Spracherkennungsprogramme sind Beispiele dafür.

Aber auch hier zeigte sich, dass das "Verstehen" von Mustern, also z.B. das Erkennen von Gegenständen bei der visuellen Wahrnehmung nicht ohne umfangreiches Vorwissen über die spezielle Situation, die es wahrzunehmen gilt, auskommt. Einen frühen Versuch in diese Richtung machte Guzman (1968). Er konnte zeigen, dass in einem Bild, das eine Anordnung von Bauklötzen zeigt, die Linien (oder Kanten der Klötze) nur auf relativ wenige Arten aufeinandertreffen können. Sein Programm nutzte dies und die Tatsache, dass jede dieser wenigen Möglichkeiten einem bestimmten Arrangement der Klötze entspricht, aus, um Strichzeichnungen von Bauklötzen zu verstehen. Waltz (1975) erweiterte diese Analyse, so dass sein Programm schliesslich auch Informationen wie Schattenwurf etc. verarbeiten konnte. Neuere Entwicklungen verwenden noch wesentlich mehr Vorwissen, so z.B. ACRONYM (Brooks, 1981). Ausgerüstet mit der geeigneten Wissensbasis war ACRONYM z.B. in der Lage, Luftaufnahmen von Flughäfen zu analysieren, also etwa zu erkennen, wo Flugzeuge von welchem Typ zu sehen sind.

Auch hier ist der aktuelle Stand der Forschung vergleichbar mit dem der schon besprochenen Gebieten: Für eng begrenze Aufgaben existieren befriedigende Lösungen; generelle Programme sind dagegen nicht in Sicht.

1.1.4 Wissensrepräsentation

All den bisher besprochen Programmen ist ein Problem gemeinsam, nämlich die Frage, wie das Wissen, über das sie verfügen müssen, am besten dargestellt wird. Die Lösung dieses Problems wurde besonders dringlich, als allmählich klar wurde, dass ein Programm ohne aufgabenspezifisches Wissen kaum interessante Leistungen erbringen kann.

Die historisch gesehen älteste Form der in der KI verwendeten Wissensrepräsentationen sind logische Notationen wie etwa:

(A oder (B und C))

Wobei A, B, C etc. Aussagen sind, die wahr oder falsch sein können (z.B. "Es regnet"). Ein frühes Beispiel für die Verwendung einer solchen Notation ist QA3 von Green (1969), ein Programm, das Fragen zu einem bestimmten Problem beantworten konnte. Später wurde eine ganze Programmiersprache, nämlich PROLOG, entwickelt, die auf dieser Repräsentation basiert (z.B. Clocksin & Mellish, 1981).

Ein Problem mit logischen Notationen ist, dass in einer derartigen Wissensrepräsentation nirgends festgehalten ist, was man mit diesem Wissen anfangen kann. Sie ist rein deklarativ. Als Gegenstück dazu wurden auch rein prozedurale Repräsentationen entwickelt, in denen das Wissen die Form von Prozeduren annimmt, die Aussagen bei Bedarf generieren. Ein frühes Beispiel für diese Klasse von Repräsentationen ist ein Programm von Woods (1968), das Fragen über Flugpläne beantworten konnte. Dieses Programm übersetzte Fragen in eigentliche kleine Programme, die dann auf die Suche nach der gewünschten Information gingen. Heute werden v.a. Mischungen deklarativer und prozeduraler Repräsentationen verwendet.

Eine häufig eingesetzte Methode, deklaratives Wissen zu strukturieren, sind die von Quillian (1968) eingeführten semantischen Netze. In einem semantischen Netz sind die einzelnen Be-

griffe wie "Amsel" oder "Vogel" durch Beziehungen wie "ist ein" verknüpft. So können Informationen, die für alle Vögel typisch sind, bei "Vogel" abgespeichert und dann bei Bedarf z.B. für "Amsel" über die entsprechende Beziehung abgerufen werden. Eine Frage, die sich bei semantischen Netzwerken stellt, ist, aus welchen Grundelementen, sogenannten "semantischen Primitiva", das gesamte Netz aufgebaut werden soll. Hier haben v.a. die im Zusammenhang mit der Sprachübersetzung erwähnten Autoren Schank (1975) und Wilks (1973) Pionierarbeit geleistet.

Gerade im Zusammenhang mit Sprachverstehen wurde aber klar, dass Wissen nicht nur aus derartigen vernetzten Bruchstücken bestehen kann. Erfahren wir z.B., dass jemand in ein Restaurant ging und dort viel Geld ausgegeben hat, dann wissen wir, dass diese Person höchstwahrscheinlich im Restaurant etwas gegessen hat, dabei an einem Tisch gesessen ist, etc. Dies lässt sich nicht in einem semantischen Netz repräsentieren, sondern bedarf einer übergeordneten organisatorischen Einheit. Minsky (1975) prägte für solche Einheiten den Begriff des "frames". Ein frame ist eine Wissensstruktur mit "Löchern" (slots), die durch geeignete Informationen gefüllt werden. Ein frame für Restaurant-Szenen (Schank & Abelson, (1977) nennen einen derartigen frame für stereotype Situationen "script") würde also z.B. slots für die Art des Essens, den Preis, etc. enthalten. Der frame organisiert, wie diese Informationen zueinander in Beziehung stehen.

Später zeigte sich dann aber wiederum, dass derartige Strukturen wie etwa ein Restaurant script zu wenig flexibel sind. Denn es gibt unzählige Dinge, die in einem Restaurant passieren können und dadurch eine bestimmte Bedeutung erhalten, die unmöglich alle im script vorgesehen sein können (ich kann Kopfweh bekommen, es kann ein Feuer ausbrechen, ein Bettler kommt ins Restaurant, etc. Schank, 1982). Neuere Versuche gehen deshalb wieder mehr in Richtung fundamentaler Einheiten, die sich bei Bedarf spontan zu frame-artigen Strukturen vereinen. Dabei gelangen vermehrt wieder neuronale Netzwerke, wie wir sie bei Rosenblatts Perceptron gesehen haben (Minsky & Papert, 1969), zum Einsatz, die jetzt als "konnektionistische" Modelle bezeichnet werden (Rumelhart, McClelland et al., 1986). Bisher sind aber hier keine grossen Durchbrüche erfolgt, und dies dürfte der Hauptgrund dafür sein, dass alle oben besprochenen Ansätze in Spracherkennung, etc. immer nur in sehr beschränkten Gebieten erfolgreich sind. Es ist bisher einfach noch nicht gelungen, wirklich grosse Mengen von Wissen flexibel zu repräsentieren.

1.1.5 Lernen

Weniger ein Grundproblem als vielmehr ein Fernziel der KI wäre es, dass all das benötigte Wissen nicht mehr von Hand programmiert werden muss, sondern dass die Programme dies selbst lernen. Auch hier sind die Bemühungen so alt wie die KI selbst. Samuels oben erwähntes Dameprogramm (Samuel, 1963) war z.B. unter anderem darum so erfolgreich, weil es aus seinen Erfolgen und Misserfolgen lernen konnte. Es lernte einerseits erfolgreiche Zugkombinationen auswendig und veränderte andererseits seine Bewertungsfunktionen, so dass es besser in der Lage war, erfolgsversprechende von schlechten Brettsituationen zu unterscheiden. Auch Rosenblatts Perceptron war nicht nur wegen seiner Architektur von besonderem Interesse, sondern auch, weil es die richtige Klassifikation von Mustern anhand von Beispielen lernen konnte (Minsky & Papert, 1969).

Das Lern-Problem erwies sich aber als äusserst schwierig und längere Zeit wurde auf diesem Gebiet nicht viel publiziert. Winston präsentierte 1970 (Winston, 1975) ein Programm, das aufgrund von Beispielen in der Lage war, einfache Begriffe zu lernen. 1975 stellte Lenat AM vor, ein äusserst interessantes Beispiel für ein Art "entdeckendes Lernen" (Davis & Lenat,1980). AM verfügt über grundlegendes Wissen zur Mengenlehre und eine Menge von Suchheuristiken wie etwa "Eine mathematisch interessante Menge ist eine Menge, deren Elemente sich einfach beschreiben lassen". Ausgehend von diesem Vorwissen ist AM in der Lage, z.B. die Primzahlen als interessante Menge von Zahlen zu "entdecken" und auch einige ihrer Eigenschaften zu erforschen.

Seit Beginn der achtziger Jahre hat sich aber die Situation radikal gewandelt und Lernen ist heute eines der am stärksten bearbeiteten Gebiete. Dabei stehen sich zwei "Schulen" gegen-

über. Auf der einen Seite findet man die "machine learning" Ansätze, die Lernprogramme im Stil der "klassischen KI" schreiben, sowie dies auch Winston und Lenat getan haben (Michalski, Carbonell & Mitchell, 1983). Die andere Seite bilden die oben schon erwähnten "konnektionistischen" Programme, die Netze abstrakter Neuronen simulieren. Beide Ansätze haben ihre Vor- und Nachteile, die sich dann auch in den Aufgaben spiegeln, die diese Programme lernen können. Auf der "machine learning" Seite ist ein typisches Beispiel das Programm von Langley (Langley, 1985), das aus Erfahrung lernen kann, wie man beim "Turm von Hanoi" die Scheiben bewegen muss, so dass man schnell ans Ziel kommt. Dieses Programm lernt und verbessert explizite Verhaltensregeln. Auf der "konnektionistischen" Seite findet man Programme wie das von Rumelhart und McClelland (Rumelhart & McClelland, 1986), das in der Lage ist, aufgrund von Beispielen die Vergangenheitsformen englischer Verben zu lernen. Dieses Programm bildet implizite Assoziationen zwischen Input- und Output-Mustern.

Beide Ansätze sind bisher nur in der Lage, ganz beschränkte Aufgaben zu bewältigen. Eine Verbindung beider, wie dies neuerdings versucht wird, dürfte einen weiteren Schritt nach vorne bedeuten, aber wirklich leistungsfähige Programme sind auch hier noch nicht in Sicht.

In den letzten dreissig bis vierzig Jahren haben also gewaltige Bemühungen, menschliche Leistungen auf dem Computer zu simulieren, stattgefunden. Wer sich heute neu damit zu befassen beginnt, muss also nicht bei null beginnen, sondern kann sich auf einige Vorarbeiten stützen. Wir hoffen, dass wir mit unserer knappen Zusammenfassung einige Anregungen geben konnten, wo was zu finden ist. Ausführlichere Darstellungen enthalten Boden (1977) und das mehrfach zitierte "Handbook of Artificial Intelligence" (Barr & Feigenbaum, 1981, Cohen & Feigenbaum, 1982). Einschlägige Zeitschriften sind vor allem "Artificial Intelligence", "Cognitive Science", "Cognition" und "Machine Learning".

1.2 Künstliche Intelligenz und kognitive Simulation

Die theoretischen Ausrichtungen, die hinter all diesen verschiedenen Arbeiten stehen, sind recht unterschiedlich. Es lassen sich zwei Hauptpole ausmachen, die gut anhand von zwei frühen Ansätzen (Schachprogramme und General Problem Solver), illustriert werden können.

Ziel der Hersteller von Schachprogrammen ist es, eine Maschine zu bauen, die möglichst gut Schach spielt. Durch welche Mechanismen das erreicht wird, ist prinzipiell gleichgültig, solange die eingesetzten Mittel zum Ziel führen. Unter anderem ist es z.B. gleichgültig, ob sich das Programm das gleiche "denkt" wie ein menschlicher Schachspieler in derselben Situation, ob es dieselben Fehler macht, wie sie ein Mensch machen würde, etc. Was zählt, ist nur die Nettoleistung. Eine Analogie zu und damit eine Simulation von menschlichem Verhalten besteht nur für das Endprodukt: Ein solches Programm kann eben auch Schach spielen. Die Art, wie es dies tut, ist nicht Teil der Analogie. Wir bezeichnen im Folgenden nur diesen Ansatz als "Künstliche Intelligenz" (KI).

Demgegenüber sind die Erbauer des GPS (Newell & Simon, 1963) explizit von der Analyse menschlicher Problemlöseprozesse ausgegangen und haben versucht, ihr Programm so zu schreiben, dass es die vorgelegten Probleme nicht nur löst, sondern sie so löst, wie das ein Mensch tun würde. Das schliesst unter anderem auch mit ein, dass das Programm dieselben Fehler machen sollte, wie sie ein Mensch macht, dass für das Programm dieselben Aufgaben "schwierig" bzw. "leicht" sein sollten, wie für Menschen, etc. Bezeichnenderweise gipfelte dieses Vorhaben 1972 im berühmten Buch "Human Problem Solving" von Newell und Simon. Ihr Interesse lag also weniger darin, das Programm zu bestimmten Leistungen zu bringen, sondern sie versuchten vielmehr, etwas über menschliches Denken zu erfahren, indem sie menschliches Problemlöseverhalten in möglichst vielen Details auf dem Computer abbildeten. Diesen Ansatz bezeichnen wir als "Kognitive Simulation" (KS).

Selbstverständlich sind beide Ansätze legitim, je nach dem Ziel, das damit verfolgt wird. Und es ist durchaus auch möglich, Resultate, die aus der KI stammen in der KS zu verwenden, bzw. umgekehrt. Entscheidend ist dabei nur, dass man sich beim Studium der umfangreichen Literatur jeweils bewusst ist, welches Ziel der jeweilige Autor verfolgt und welche Ansprüche er folglich mit seinen Aussagen erhebt.

1.3 Über die "Machbarkeit" von Computersimulationen

Man kann sich nun natürlich zu Recht fragen, ob die Ziele, die hier angestrebt werden - sei es im Rahmen "Künstliche Intelligenz" oder "Kognitive Simulation" - überhaupt erreichbar sind. Ist es überhaupt möglich, einen Computer so zu programmieren, dass er sich "menschlich" verhält? Einer der vehementesten Kritiker und Zweifler in diesem Zusammenhang ist H.L. Dreyfus, der seit der Mitte der Sechzigerjahre immer wieder mit den Anhängern der Cognitive Science scharf ins Gericht geht (z.B. Dreyfus, 1985, aber auch Winograd & Flores, 1986). Zu Recht wirft er ihnen vor, dass sie verschiedentlich in Ausbrüchen von Euphorie überrissene Versprechungen gemacht haben, die dann nicht eingelöst werden konnten. Z.B. prophezeite Simon 1958 (Simon & Newell, 1958, p. 6), dass in spätestens zehn Jahren ein Computer Schachweltmeister sein würde. Ein solcher Computer ist aber auch heute noch nirgends auszumachen.

Interessanter als die scharfe Polemik von Dreyfus sind seine Überlegungen dazu, welche Überzeugungen man teilen muss, damit man an die Möglichkeit von Computersimulationen menschlichen Denkens glauben kann. Er führt der Reihe nach vier Annahmen auf, von denen er jede folgende jeweils als schwächer und damit als weniger leicht widerlegbar betrachtet.

1. Die biologische Annahme
2. Die psychologische Annahme
3. Die erkenntnistheoretische Annahme
4. Die ontologische Annahme

Die biologische Annahme lässt sich bündig auf die Formel bringen "Das Gehirn ist ein Computer". Trifft dies zu, dann sollte es natürlich möglich sein, einen anderen Computer so zu programmieren, dass er sich genau gleich wie der erste verhält. (Dies ist ebenfalls eine Konsequenz aus Turings Untersuchungen.) Etwas vorsichtiger ist die psychologische Annahme, die sich nicht darauf festlegen will, ob das Gehirn tatsächlich wie ein Computer arbeitet. Sie lässt sich etwa wie folgt zusammenfassen: "Sowohl im Gehirn wie in einem Computer laufen Informationsverarbeitungs-Prozesse ab, die - auf dem richtigen Abstraktionsniveau beschrieben - von gleicher Art sind". Trifft dies zu, dann sollte es möglich sein, auf einen Computer dieselben Prozesse zu installieren, die auch im Gehirn ablaufen, ungeachtet dessen, dass gewisse Details, die unter dem "richtigen" Abstraktionsniveau liegen, nicht übereinstimmen. Noch vorsichtiger ist die erkenntnistheoretische Annahme. Sie geht nicht mehr von der Frage aus, wie das Gehirn bzw. die darin ablaufenden Prozesse "wirklich" funktionieren, sondern sie setzt auf der Ebene der Beschreibung dieser Prozesse an. Man kann sie so zusammenfassen: "Psychische Prozesse lassen sich durch formale Regeln beschreiben". Trifft dies zu, dann lässt sich auch ein Computerprogramm schreiben, das genau diesen formalen Regeln folgt und damit durch die gleiche Beschreibung beschrieben werden kann, wie der entsprechende psychische Prozess. Dreyfus illustriert sehr schön anhand eines Beispiels aus der Physik, was damit gemeint ist: Die Bahnen, die die Planeten um die Sonne einhalten, lassen sich als Differentialgleichungen beschreiben und will man voraussagen, wohin sich ein bestimmter Planet in nächster Zeit bewegt, erhält man die Antwort, wenn man die entsprechende Differentialgleichung löst. Das bedeutet nun aber nicht unbedingt, dass die Planeten ständig dabei sind, Differentialgleichungen zu lösen, damit sie entscheiden können, wie ihre Bahn weitergeht. Wahrscheinlich tun sie es nicht. Es ist nur so, dass sich Ihr Verhalten formal so beschreiben lässt. Und entsprechend ist es auch möglich, ein Computerprogramm zu schreiben, das genau diesen Regeln - den Differentialgleichungen - folgt, so dass sich die simulierten Planeten gleich wie die Originale verhalten. Die erkenntnistheoretische Annahme geht nun davon aus, dass gleiches in der Psychologie auch möglich ist.

Die vierte Annahme, die ontologische, hebt einen Teilaspekt der dritten explizit hervor: "Die Beschreibung psychischer Prozesse lässt sich aus einer Menge festgelegter, elementarer Einheiten aufbauen". D.h. es muss eine Art Baukasten geben, mit welchem sich Beschreibungen psychischer Prozesse aufbauen lassen, genauso wie es Baukästen gibt (die Programmiersprachen), mit welchen die Programme aufgebaut sind.

Welche dieser Annahmen haltbar sind und welche nicht, ist im Prinzip eine empirische Frage, die nur die zukünftige Forschung klären kann. Wir glauben jedenfalls nicht, dass die eine oder die andere schon definitiv widerlegt oder gesichert ist.

1.4 Unsere Position

Unsere eigene Haltung zur "Machbarkeit" von Computersimulationen ist relativ pragmatisch. Betrachtet man den Computer nicht als mystisches Wesen, sondern als Werkzeug, dann stellt er eine Maschine dar, die in der Lage ist, das, was man ihr eingibt, nach anerkannten logischen Regeln umzuformen. Er eignet sich also (wie im einleitenden Kapitel besprochen) vorzüglich dafür, eine formale Aussage oder Theorie auf ihre logischen Konsequenzen hin zu überprüfen. Wir betrachten also Computersimulation als Hilfsmittel zum Aufbau logisch konsistenter und vollständiger Theorien.

Für unsere Position ist folglich die Entscheidung über die ersten beiden Annahmen von Dreyfus nicht von Bedeutung. Für unsere Auffassung von Computersimulation spielt es keine Rolle, ob das Gehirn wirklich ein Computer ist, oder ob psychische und computerisierte Informationsverarbeitungsprozesse wirklich im Prinzip identisch sind. Das bedeutet für uns aber gleichzeitig auch, dass eine laufende Computersimulation dadurch, dass sie läuft, keinesfalls bereits etwas über die empirische Validität der damit verbundenen Theorie aussagt. Die als Computersimulation implementierte Theorie muss - gleich wie jede andere Theorie - durch gezielte empirische Untersuchungen überprüft werden (Herrmann, 1990).

Mit unserem Einsatz von Computersimulation zur Theorieprüfung auf logische Stimmigkeit verpflichten wir uns aber der dritten und vierten Annahme: Nur wenn eine Theorie als formales System von Regeln formuliert ist, lässt sie sich als Computerprogramm darstellen; und sie muss dazu notwendigerweise aus eindeutig definierten Bausteinen aufgebaut sein. Ob dies für psychologische Theorien möglich ist, ist eine offene Frage. Als Kronzeugen gegen die Formalisierbarkeit psychologischer Beschreibungen werden oft Wittgensteins Sprachuntersuchungen angeführt, die ihn zur Überzeugung führten, dass es keine Möglichkeit gibt, den Gebrauch von Sprache durch klar definierte Regeln zu beschreiben (Wittgenstein, 1984, Winograd & Flores, 1986). Sollte sich dies bewahrheiten, dann würde dies nicht nur die Unmöglichkeit von Computersimulation in der Psychologie bedeuten, sondern hätte auch das Ende einer ganzen Tradition "wissenschaftlicher" Theoriebildung in der Psychologie zur Folge. Dieses Buch baut also auf dem Glauben daran auf, dass die Todesanzeige für diese Art Psychologie noch nicht erschienen ist.

2 Die wichtigsten Komponenten einer MINCS

2.1 Allgemeines

Nachdem wir uns mit den Zielen und Möglichkeiten einer MINCS allgemein auseinandergesetzt haben, wollen wir in diesem Kapitel in groben Zügen darstellen, wie der Aufbau einer solchen Simulation vor sich geht. Jeder einzelne der dabei erwähnten Schritte wird in einem der folgenden Kapitel ausführlicher behandelt.

2.1.1 Die Spielwelt

Psychische Prozesse - um deren Darstellung es ja geht - laufen nicht in einem Vakuum ab, sondern immer auf dem Hintergrund einer Umgebung. In dieser Umgebung wird die Aufgabe formuliert, die zu bewältigen ist, und sie setzt die Randbedingungen, unter denen das geschieht. Bei psychischen Prozessen wird diese Umgebung im Allgemeinen die äussere Umwelt sein, mit der sich ein Mensch auseinandersetzt. Da es kaum je möglich sein wird, das Simulationsprogramm mit dieser realen Umwelt in ihrer vollen Komplexität interagieren zu lassen, ist ein grundlegender Bestandteil jeder Simulation die Simulation auch dieser Umgebung. Wir nennen sie hier Spielwelt, da es sich dabei selbstverständlich nur um die Realisierung eines reduzierten Ausschnitts der gesamten natürlichen Umwelt handeln kann. Von der Wahl einer geeigneten Spielwelt hängt das Gelingen der Simulation wesentlich ab, denn die simulierten Prozesse werden sich erst einmal in der Bewältigung der Spielwelt bewähren müssen. In ihr sollte sich die Aufgabe idealerweise so formulieren lassen, dass einerseits zu ihrer Bewältigung nur gerade der zu untersuchende Prozess notwendig ist, dass andererseits dieser Prozess aber genau gleich gefordert wird, wie wenn eine reale Situation zu bewältigen wäre.

2.1.2 Neutrale Teilprozesse

Diese Maximalforderung an die Spielwelt lässt sich kaum je verwirklichen. Meist wird es notwendig sein, über den zu untersuchenden Prozess hinaus einige weitere Prozesse zu programmieren, damit die Simulation überhaupt läuft. Wie diese programmtechnisch genau realisiert werden, ist im Prinzip gleichgültig. Als weitere Vorbereitung für die eigentliche Simulation ist es deshalb sinnvoll, diese möglichst frühzeitig so zu implementieren, dass sie zwar vorhanden sind, die eigentliche Simulation aber nicht gross belasten. Zu diesem Zweck muss man sich u.a. klar werden, welchen Prozess man tatsächlich untersuchen will und welche anderen, bei der Lösung der Aufgabe beteiligten Prozesse nur Nebenschauplätze sind.

2.1.3 Das Simulationsprogramm

Nach diesen Vorbereitungen folgt als drittes die eigentliche Hauptarbeit, nämlich der Aufbau des Simulationsprogrammes. Diese Arbeit wird im Allgemeinen zyklisch verlaufen. Ausgangspunkt sind mehr oder weniger konkrete Vorstellung darüber, wie der Prozess und die damit verbundenen Strukturen aussehen sollten. Diese Vorstellungen werden in ein Programm umgesetzt, wobei sich normalerweise bereits hier zeigt, dass sie präzisiert werden müssen. Meist wird schon ein beträchtlicher Erkenntnisgewinn realisiert, bis das erste Programm läuft. Ist es dann soweit, stellt sich mit Sicherheit die ernüchternde Beobachtung ein, dass das Programm keineswegs das leistet, was man von ihm erwartet hat. Das führt zu einer Reformulierung des Prozesses, zu einem neuen Programm, etc. Diese Schlaufe - im günstigen Fall eine Spirale - wird solange durchlaufen, bis ein befriedigendes Produkt vorliegt.

2.1.4 Heuristische Empirie

Auf diesem Weg gibt es zwei Möglichkeiten, in die Irre zu gehen. Die erste liegt darin, dass mit der Zeit das Teilziel, endlich ein laufendes Programm zu haben, das Hauptziel, nämlich eine psychologisch valide Darstellung des Prozesses zu erhalten, vollständig verdrängt: Die Kognitive Simulation wird zur Künstlichen Intelligenz. Dem gilt es vorzubeugen, indem während die-

ser Phase immer wieder der Kontakt zur psychologischen Empirie gesucht wird. Die unmittelbarste Quelle solcher Empirie ist die eigene Introspektion, die sicher einige Zeit gute Dienste leistet. Früher oder später wird sich dann aber unvermeidlich die Wahrnehmung der eigenen Prozesse dem anpassen, was sich auf dem Computer am leichtesten verwirklichen lässt. Als weitere Informationsquelle sind hier deshalb kleine, informelle Beobachtungen und Experimente mit gut motivierten Versuchspersonen ideal.

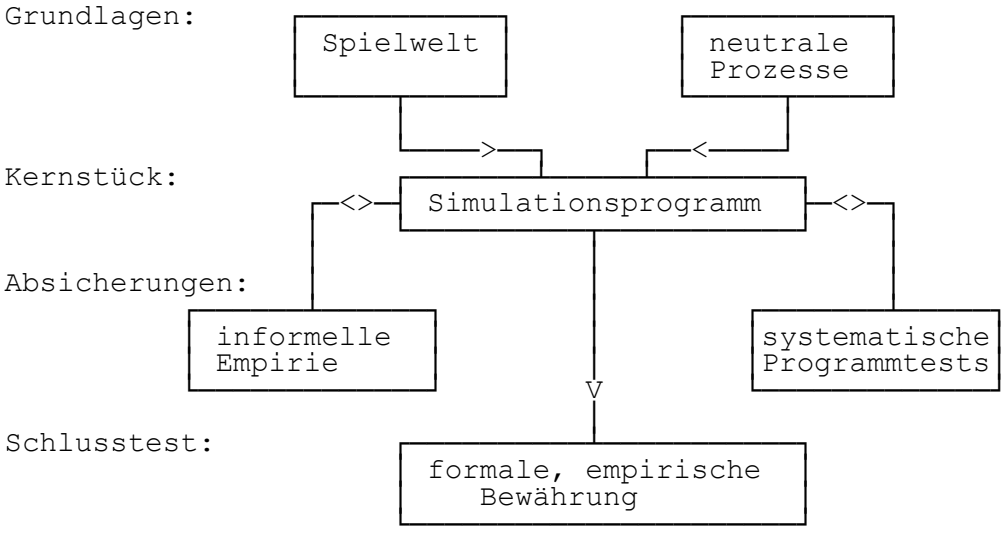
2.1.5 Lauftests

Als zweite Gefahr droht, dass man das Programm allzu sehr auf eine ganz bestimmte Ausgestaltung der Spielwelt ausrichtet. Zu Beginn wird man ja einmal diese in allen Details festlegen und dann damit arbeiten. Auch wird man es bereits als Erfolg werten, wenn das Programm genau mit diesen Randbedingungen fertig wird. Meistens soll der Prozess aber nicht nur eine ganz bestimmte Konstellation von Umweltbedingungen verarbeiten können, sondern eine ganze Klasse. Um dies sicherzustellen, ist es notwendig, dass man die entscheidenden Grössen der Spielwelt systematisch variiert. Dabei wird man meistens entdecken, dass der Prozess nicht so allgemein ist, wie man angenommen hat, und das Programm deshalb eventuell nochmals verändern. Oder man wird hier abbrechen, damit nicht aus einer MINCS eine MACS (Mords-Aufwendige Computer Simulation) wird.

2.1.6 Empirische Tests

Zu guter Letzt verfügt man dann endlich über eine Beschreibung eines psychischen Prozesses, die insofern vollständig und logisch stimmig ist, als das aufgrund dieser Beschreibung angefertigte Computerprogramm läuft und das von ihm erwartete Verhalten zeigt. Trotz aller Absicherung der Programmentwicklung über ständige Kontakte mit der Empirie ist aber nicht garantiert, dass die entsprechende Theorie die psychologische Empirie korrekt beschreibt. Dies muss abschliessend mit den üblichen Methoden empirischer Überprüfung demonstriert werden.

Alles in allem unterscheiden wir also sechs verschiedene Komponenten, die uns bei der Darstellung einer MINCS speziell erwähnenswert scheinen. Figur 1 zeigt schematisch, wie diese ineinandergreifen.



Figur 1: Komponenten einer MINCS

2.2 Beispiele

Im Folgenden werden wir nun jede dieser sechs Komponenten in einem separaten Kapitel ausführlicher behandeln. Was wir dabei an Allgemeinem zu sagen haben, wird jeweils schnell gesagt sein, denn es sollen wirklich nur die wichtigsten Punkte dargestellt werden. Daran an-

schliessend werden wir aber versuchen, das Gesagte anhand ausführlicher Beispiele zu illustrieren. Einerseits sollen diese Beispiele dokumentieren, aufgrund welcher Erfahrungen wir dazu gekommen sind, dieses Buch zu schreiben, und andererseits glauben wir, dass sie informativer sind als lange, abstrakte Abhandlungen. Sie stammen zum Teil aus der Literatur, die meisten haben ihren Ursprung aber in zwei von uns durchgeführten Forschungsprojekten, die auch an anderen Orten ausführlich dokumentiert sind:

Beispiel 1: Problemlösen zu zweit (Kaiser, 1985, 1987)

Ausgangspunkt dieses Projekts war der Eindruck, dass Gespräche zwischen zwei Partnern, die versuchen, gemeinsam ein Problem zu lösen, oft nicht zu einem Ziel gelangen, obwohl es weder an der Motivation der Beteiligten noch am zur Problemlösung notwendigen Wissen fehlt. Eine typische Situation dieser Art wäre etwa, wenn zwei Psychologiestudenten gemeinsam an einer Statistikaufgabe scheitern, obwohl ihr kombiniertes Wissen für eine Lösung ausreichen würde. Irgendwie gelingt es ihnen nicht, in ihrem Dialog aus den beiden Wissensstücken ein Ganzes zu machen.

Mit Hilfe einer MINCS versuchten wir, die eigentliche Steuerung eines solchen Dialogs (Welche Teilprobleme werden wann bearbeitet? Wie werden Resultate integriert? etc.) darzustellen. Dabei hatten wir ein doppeltes Ziel. Einmal versuchten wir auf diesem Weg zu erfahren, wie diese Steuerung ablaufen muss, damit das Gespräch überhaupt gelingen kann. Und zum zweiten versuchten wir dann auf diesem Hintergrund das tatsächliche Verhalten von Versuchspersonen und ihre Abweichungen vom optimalen Vorgehen darzustellen. Auf einzelne Punkte werden wir im Verlauf der folgenden Kapitel eingehen.

Der Einsatz der MINCS erwies sich als äusserst fruchtbar, denn es gelang uns, gewisse Gesprächsregeln zu identifizieren, die in spontanen Dialogen praktisch durchgehend verletzt werden. Werden diese Regeln mit Versuchspersonen trainiert, dann verbessert sich ihr Problemlöseverhalten deutlich. Besonders interessant war die Verwendung eines Simulationsprogramms in diesem Fall zusätzlich, weil es so möglich war, "Gespräche" zwischen einem simulierten und einem menschlichen Partner durchzuführen.

Beispiel 2: Elementare Lernprozesse (Kaiser & Keller, 1985, 1986, 1987a, 1987b, 1989)

Dieses zweite Projekt hatte zum Ziel zu ergründen, welche Lernprozesse und -kompetenzen mindestens angenommen werden müssen, damit ein Lerner ohne grosses Vorwissen nur aus den Erfahrungen, die er mit einer Umwelt macht, diese in den Griff bekommt. Wir simulierten also unterschiedlichste Lernprozesse und prüften, was sich damit überhaupt lernen lässt.

Zwar handelte es sich in diesem Fall nicht um eine vollständige MINCS im obigen Sinn, da wir nie bis zur abschliessenden formalen Prüfung gelangten. Das Beispiel zeigt aber einen weiteren interessanten Aspekt von Computersimulationen generell: Da ein unprogrammierter Computer eine echte "tabula rasa" darstellt, lässt er sich gut einsetzen um zu ergründen, welche Prozesse zur Lösung einer Aufgabe minimal notwendig sind.

Beispiele 3 und 4:

Die übrigen zwei Beispiele aus der Literatur werden wir jeweils in den Kapiteln, in denen wir auf sie zurückgreifen, näher beschreiben. Wir werden sie einerseits als Beleg dafür verwenden, dass andere Autoren ähnlich vorgehen, wie wir das hier beschreiben. Andererseits dienen sie aber auch dazu, Punkte zu veranschaulichen, zu denen sich aus unseren Projekten keine direkten Beispiele ableiten lassen.

2.3 Zum Nachdenken

Für diejenigen Leser und Leserinnen, die das Gelesene jeweils gleich an einem Beispiel anwenden möchten, haben wir uns ein kleines Projekt ausgedacht, das parallel zur Lektüre des Buches durchgeführt werden kann. Thema dieses Projekts ist das Morsen, bzw. die Wahrnehmungsprozesse, die beim Decodieren von Morsezeichen ablaufen. (Die Anregung zu diesem Projekt stammt aus Pfisterer, 1988.) Das Morsen wurde zwar unterdessen beinahe vollständig durch andere Telekommunikationsmittel ersetzt. Man kann sich aber trotzdem noch

die Frage stellen, wie ein geübter Morser den beinahe kontinuierlichen Strom von kurzen und langen Pieptönen in eine Sequenz von Zeichen dekodiert.

Ziel dieses Übungsprojektes soll ein Simulationsprogramm sein, das diese Frage zu beantworten versucht. Dabei können je nach Interesse unterschiedliche Aspekte in den Vordergrund treten. Man könnte sich z.B. auf die Frage konzentrieren, wie der Dekodierungsprozess organisiert sein muss, dass er mit einer derartigen Geschwindigkeit ablaufen kann. Offensichtlich müssen dabei verschiedene Vorgänge parallel stattfinden. Beispielsweise ist die Person, die eine längere Morsesequenz abnimmt, ja immer noch mit dem Niederschreiben des letzten Zeichens beschäftigt, wenn sie bereits das nächste wahrnimmt. Fragen, die sich daraus ergeben könnten, sind etwa: Wie muss man sich die Interaktion der Teilprozesse vorstellen? Lassen sich daraus bestimmte Fehlermuster erklären? etc.

Der Leser oder die Leserin kann sich aber auch der Frage widmen, wie ein solcher Wahrnehmungsprozess gelernt wird. Die Trainingszeit, die bis zur Perfektion benötigt wird, ist recht lang (zwischen 300 und 600 Stunden; Pfisterer, 1988) und verschiedenste Versuche, sie durch bessere Lehrmethoden zu reduzieren, brachten kaum Erfolge. Mögliche Fragen wären hier: Wie muss der Wahrnehmungsprozess beschaffen sein, dass er überhaupt lernbar ist? Wie kann man sich den Lernprozess vorstellen? Lassen sich so Unterschiede in den Fehlermustern zwischen Anfängern und Experten erklären?

Welchen Teilfragen sich der Leser oder die Leserin auch zuwenden wird, die Welt des Morsens ist in sich genügend abgeschlossen, so dass daraus ein klar abgegrenztes Übungsprojekt entstehen kann. Wieweit dabei in eigener Verantwortung in die Details gegangen wird, ist jedem selbst überlassen. Wir werden einfach jeweils am Ende eines Kapitels einige Fragen aufwerfen, deren Beantwortung das "Morse-Projekt" einen Schritt weitertreiben sollte. Soweit wir Antworten auf unsere Fragen kennen, haben wir sie im Anhang gesammelt.

3 Die Spielwelt

3.1 Allgemeines

Das Simulationsprogramm benötigt eine Umgebung und eine Aufgabe in dieser Umgebung, damit es seine Fähigkeiten zeigen kann. Selbstverständlich könnte man zu diesem Zweck versuchen, das Programm direkt der "Realität" auszusetzen. Dies ist aber im Rahmen einer MINCS kaum je sinnvoll. Denn nur schon die Entwicklung jenes Programmteils, der aus der Umwelt die für die Aufgabe relevanten Informationen herauszufiltern hätte, wäre in den meisten Fällen ein eigenständiges KI-Projekt grossen Umfangs. Dem Programm muss also die "Realität" in reduzierter Form gegenübergestellt werden. Als erstes geht es deshalb bei einer MINCS darum, eine simulierte Umwelt, eine Spielwelt, zu schaffen, in der sich das eigentliche Simulationsprogramm bewähren kann.

Damit keine Verwirrung entsteht, sei ausdrücklich betont, dass wir es also mit zwei Simulationen zu tun haben (vgl. auch Opwis, Spada & Schwiersch, 1985). Einerseits simulieren wir die Umwelt, mit der sich das eigentliche Simulationsprogramm auseinanderzusetzen hat, in Form einer Spielwelt ("Simulationsmodell 1" bei Opwis et al. 1985). Mit dieser Spielwelt könnte auch eine Versuchsperson interagieren. Und zum zweiten simulieren wir dann die psychischen Prozesse, die z.B. in einer derartigen Versuchsperson ablaufen, wenn sie in der Spielwelt eine bestimmte Aufgabe löst ("Simulationsmodell 2" bei Opwis et al. 1985). Die eigentliche Kognitive Simulation ist diese zweite Simulation. Die Spielwelt ist nur Mittel zum Zweck.

Optimalerweise umfasst die Spielwelt genau jene Aspekte der "Realität", die für den zu untersuchenden Prozess von Bedeutung sind. Denn dadurch würde erreicht, dass das Simulationsprogramm genauso gefordert wird, wie das in der Realität der Fall wäre, ohne dass es aber unnötig aufgebläht werden muss. Zu Beginn der Untersuchung wird man sich allerdings mit einer bescheidenen Annäherung an dieses Ideal begnügen müssen. Denn damit man die Spielwelt optimal auf den zu untersuchenden Prozess abstimmen könnte, müsste dieser bereits sehr gut bekannt sein - was kaum der Fall sein dürfte. Es kann also in einem ersten Schritt nur darum gehen, die Spielwelt aufgrund von Vermutungen über die Form des Prozesses so gut wie möglich auf diesen zuzuschneiden. Im Verlauf der weiteren Arbeit wird sie dann wahrscheinlich mehrfach angepasst werden müssen, so dass sie einer Idealform immer näher kommt. Das Wissen über den zu untersuchenden Prozess und über die dazu optimale Spielwelt sollten sich also im Verlauf der Erfahrungen, die man mit einer MINCS macht, gegenseitig (dialektisch) weiterentwickeln.

Dies ist keineswegs so ungewöhnlich, wie es auf den ersten Blick aussehen mag. Ganz ähnlich dialektisch haben sich in der Forschung die Theorien und die sie prüfenden experimentellen Methoden schon immer entwickelt. Ein Beispiel dazu aus neuerer Zeit ist die Forschung zum "Komplexen Problemlösen" im deutschsprachigen Raum und die damit verbundene Frage, durch welche Merkmale sich "komplexe" Aufgaben auszeichnen. Dörners erste Versuche waren die eines Pioniers und entsprechend war "Lohausen" die erste Annäherung an eine brauchbare experimentelle Aufgabe/Spielwelt. "Komplexität" wurde dabei v.a. durch eine grosse Anzahl variierbarer Grössen und ihre Vernetztheit verwirklicht (Dörner, Kreuzig, Reither & Stäudel, 1983). Die vertiefte Auseinandersetzung mit dem Gegenstand führte dann zur Kritik an der Angemessenheit der gewählten Spielwelt (z.B. Funke, 1984) und es folgten auch Arbeiten, in denen "Komplexität" unter anderem mit wesentlich weniger Variablen verwirklicht wurde (z.B. Kluwe, Misiak & Schmidle, 1985). Bis nun Dörner selbst eine Arbeit publiziert hat, bei der eine Aufgabe mit nur einer einzigen Variablen Verwendung findet (Reichert & Dörner, 1988).

Ähnliche dialektische Abhängigkeiten werden wir übrigens im Folgenden auch noch für weitere Aspekte der MINCS finden. Die gesamte MINCS besteht aus mehreren, gegenseitig voneinander abhängigen Komponenten, die alle zu Beginn einmal in irgendeiner Form verwirklicht werden müssen, damit der Forschungsprozess überhaupt in Gang kommt. Ihre endgültige Form erhalten sie dann aber erst im Verlauf der Arbeit.

Es stellt sich allerdings die Frage, ob diese endgültige Idealform im Falle der Spielwelt auch nur annäherungsweise erreicht werden kann. Sowohl die im Rahmen der Cognitive Science verwendeten "Mikrowelten" wie auch die experimentelle "Laborsituation" sind ja z.T. scharf als unbrauchbare Instrumente kritisiert worden. Dreyfus etwa fasst seine Kritik an der Verwendung von Mikrowelten am Beispiel von SHRDLU (Winograd, 1972) wie folgt zusammen: "(Es ist irreführend,) eine bestimmte Anzahl von Fakten und Verfahren im Hinblick auf Bauklötze eine Mikrowelt zu nennen, wo es eigentlich darum geht zu verstehen, was eine Welt ist. Eine Menge von zusammenhängenden Fakten kann ein Universum, einen Bereich oder eine Gruppe bilden, aber es bildet keine Welt, denn eine Welt ist ein organisiertes Ganzes von Objekten, Zwecken, Fertigkeiten und Gebräuchen, innerhalb dessen menschliches Handeln eine Bedeutung hat oder einen Sinn ergibt." (Dreyfus, 1985, S. 275). Diese Kritik ist mehr oder weniger identisch mit der Ablehnung der im ersten Kapitel erwähnten "ontologischen Annahme", nämlich der Annahme, dass "sich intelligentes ... Verhalten zerlegen lässt" (Dreyfus, 1985, S.154). Denn wenn sich die "Gesamtwelt" psychologisch sinnvoll in Teil- oder Mikrowelten zerlegen lässt, dann muss sich auch das Verhalten in Teile zerlegen lassen, von denen jeder mehr oder weniger unabhängig mit einem Teilaspekt der "Gesamtwelt" interagiert. Dreyfus bestreitet also rundweg, dass es möglich ist, bedeutungsvolle Teilwelten (Mikrowelten, Spielwelten, etc.) zu bilden, wobei er sich z.T. auf Husserls phänomenologische Untersuchungen beruft (Husserl, 1950). Ein Standpunkt, der unterdessen auch von Winograd, dem Vater von SHRDLU, geteilt wird (Winograd & Flores, 1986). Dieser Einwand stellt für jede MINCS ein echtes Problem dar und sollte immer als mögliche Erklärung beim Scheitern von Simulationsversuchen in Betracht gezogen werden. Wie wir schon im ersten Kapitel betonten, können und wollen wir ihn jedenfalls nicht abschliessend widerlegen. Dass wir trotzdem an die Möglichkeit erfolgreicher MINCS glauben, beruht - wie schon gesagt - einzig auf der Überzeugung, dass die Gegenposition bisher ebenfalls nicht widerlegt wurde.

Die grundsätzlichste Kritik am "Laborexperiment" aus den psychologischen Reihen bemängelt im Allgemeinen die fehlende "ökologische Validität" der Experimente (Holzkamp, 1973). Damit wird meist weniger bezweifelt, dass sich überhaupt Teilwelten zu Untersuchungszwecken isolieren lassen, sondern es geht vielmehr darum, welche Eigenschaften eine solche experimentelle Teilwelt aufweisen muss, damit sie dieselben Anforderungen wie die "Realität", die "Gesamtwelt", stellt und somit ökologisch valide ist. Es ist keine Frage, dass damit ein schwieriges Problem angesprochen ist. Es ist aber auch keine Frage, dass man es nicht lösen kann, indem man Experimente entwirft, die ökologisch valide aussehen. Denn - und hier deckt sich die Situation genau mit der Situation beim Entwurf einer Spielwelt - um zu wissen, was ökologisch valide im Hinblick auf einen zu untersuchenden psychischen Prozess ist, muss man diesen bereits sehr gut kennen. Und da dies nicht der Ausgangspunkt, sondern allenfalls das Endziel der Forschungsbemühungen sein kann, besteht die einzige Chance darin, dass das Wissen über den zu untersuchenden Prozess und der dazu passenden ökologisch validen Untersuchungssituation gemeinsam wachsen - wie oben vorgeschlagen.

Wir stellen also folgende beiden Anforderungen an eine Spielwelt:

1. Die Spielwelt sollte die gesamte Komplexität der "Realität" abbilden, soweit die Bewältigung eben dieser Komplexität Aufgabe des zu untersuchenden psychischen Prozesses ist.
2. Die Spielwelt sollte alle anderen Aspekte der "Realität" soweit wie möglich vereinfachen.

Dazu möchten wir noch ein drittes Kriterium aufnehmen, dessen Bedeutung wir später erläutern werden:

3. Die Spielwelt sollte sich als Umgebung verwenden lassen, in der Versuchspersonen spielerisch Aufgaben lösen können.

3.1.1 Realitätsnahe Spielwelt

Neben Vorkenntnissen über den zu untersuchenden Prozess sind zur Gestaltung der Spielwelt selbstverständlich auch Kenntnisse in Bezug auf den entsprechenden Realitätsauschnitt notwendig, damit man diesen überhaupt abbilden kann.

Liegen bereits formalisierte Beschreibungsmodelle des entsprechenden Realitätsauschnittes vor - aus Physik, Biologie, Linguistik, etc. - dann stellen diese natürlich eine wertvolle Quelle für die Gestaltung der Spielwelt dar (Opwis, 1985). Will man z.B. einen Prozess simulieren, der in der Lage ist, die Flugbahn eines Balles abzuschätzen, dann enthält die klassische Physik bereits alles, was man zur Darstellung einer solchen Flugbahn benötigt. Derartig formalisierte Modelle lassen sich meist recht direkt in ein lauffähiges Programm umsetzen, so dass der Aufwand klein gehalten werden kann. Allerdings ist dabei zu berücksichtigen, dass selbstverständlich auch solche Modelle bereits Vereinfachungen darstellen. Und da diese Vereinfachungen auf Grund anderer Kriterien vorgenommen wurden, als sie für eine Spielwelt gelten (z.B. mathematische Eleganz der resultierenden Theorie, Übereinstimmung mit dem geltenden Paradigma, etc. (Kuhn, 1967), ist nicht garantiert, dass sie noch alle Aspekte der "Realität" enthalten, die für die MINCS benötigt werden. Z.B. ist die klassische Physik nur in der Lage, Flugbahnen im Vakuum exakt zu beschreiben. Für reale Bedingungen, d.h. mit Luftwiderstand, liefert sie nur Annäherungen (Brancazio, 1984). Es empfiehlt sich deshalb auf jeden Fall, das, was man abbilden möchte, möglichst naiv zu betrachten und zu prüfen, wieweit die formalen Modelle diesem Eindruck gerecht werden (siehe unten).

Die Literatur zur Künstlichen Intelligenz enthält ebenfalls eine Fülle von Anregungen, die helfen, unzulässige Vereinfachungen zu vermeiden. Denn durch die Geschichte der KI zieht sich wie ein roter Faden das Muster, dass fast alle bearbeiteten Probleme (wie Sprachübersetzung, Problemlösen, natürlichsprachlicher Dialog, etc.) anfänglich unterschätzt wurden, da eine zu einfache Vorstellung der "Umwelt" und der Aufgabe bestand (Dreyfus, 1985). Diese musste mit der Zeit jeweils korrigiert werden und so lässt sich die Geschichte der KI als eine fortlaufende Auffächerung neuer Facetten der entsprechenden "Umwelten" lesen. Wir haben versucht, etwas davon in unserer kurzen Zusammenfassung im ersten Kapitel festzuhalten. Eine interessante Sequenz, die diesen Trend illustriert, bilden etwa die Arbeiten von Schank zum Thema Sprachverarbeitung (Schank, 1975; Schank & Abelson, 1977; Schank, 1982, 1984). Wurde also der Prozess, den man untersuchen möchte, bereits auch aus der Sicht der KI behandelt, lohnt es sich, diesen Versuchen nicht nur im Hinblick auf das eigene Simulationsprogramm im engeren Sinn (Kapitel 5), sondern auch beim Aufbau der Spielwelt nachzugehen.

Im Allgemeinen wird man sich aber selber eine Phänomenologie der Umwelt, in der man sein Simulationsprogramm "leben" lässt, erarbeiten müssen. Es hat sich bewährt , zu diesem Zweck zusammenzustellen, welchen Situationen der zu untersuchende Prozess in der Umwelt begegnen kann, und dann im Sinne eines brain storming alles zu notieren, was sich an Beobachtungen, Überlegungen und Assoziationen einstellt. Meist resultiert daraus eine interessante und überraschend reichhaltige Liste von Phänomenen. Die Spielwelt sollte dann so entworfen werden, dass sie möglichst vielen dieser Phänomene gerecht wird. Wir werden darauf anhand der Beispiele zurückkommen.

3.1.2 Einfache Spielwelt

Die Spielwelt ist aber eben nur dann Spielwelt, wenn alle unwesentlichen Aspekte der Umwelt radikal vereinfacht werden. Wir werden auf die damit verbundenen Fragen ausführlicher im nächsten Kapitel unter dem Titel "neutrale Teilprozesse" zurückkommen. Dort werden wir vor allem die Frage behandeln, wie sich "uninteressante" Teile des eigentlichen Simulationsprogramms möglichst kostengünstig realisieren lassen. Oft besteht die einfachste Lösung darin, dass man die Spielwelt so gestaltet, dass diese Programmteile überflüssig werden.

Vor allem "Wahrnehmungs-Komponenten" können durch eine geschickte Wahl der Spielwelt reduziert werden. Nehmen wir an, wir wollten Begriffsbildungsprozesse untersuchen und hätten uns bereits festgelegt, dass unser "Begriffsbildner" die einzelnen Beispiele und Gegenbeispiele zuerst in Form von Eigenschaftslisten repräsentiert, bevor die eigentliche Begriffsbildung zum Zuge kommt. Unter diesen Umständen ist es müssig, sich gross über die Umsetzung von Beispielen in Eigenschaftslisten Gedanken zu machen, da dies offensichtlich nicht zum Kern des Prozesses gehört. Und wir benötigen auch keinen derartigen Übertragungsprozess, wenn in der Spielwelt die Beispiele direkt in der Form von Eigenschaftslisten gegeben sind.

3.1.3 Versuchspersonengerechte Spielwelt

Wie im zweiten Kapitel angedeutet, ist für den erfolgreichen Aufbau des Simulationsprogramms entscheidend, dass man den Kontakt zur psychologischen Empirie nicht verliert. Man ist also daran interessiert, laufend das "Verhalten" des Programms mit dem von Versuchspersonen zu vergleichen. Dies gelingt am einfachsten, wenn die Versuchspersonen auf dieselbe Art mit der Spielwelt interagieren können, wie es das Programm tut. Damit dies möglich ist, muss die Spielwelt jedoch einige weitere Anforderungen erfüllen, die sich nicht bereits aus der Logik der reinen Simulation ergeben.

Einmal ist es dem Simulationsprogramm gleichgültig, ob man es immer wieder auf die haargenau gleiche Aufgabe ansetzt oder nicht. Es kennt keine Motivationsprobleme und es vergisst auch von einem Simulationslauf zum nächsten alles, was es gelernt hat - anders als die meisten Versuchspersonen. Will man die gleichen Versuchspersonen mehrmals einsetzen (was sehr sinnvoll sein kann, vgl. Kapitel 6), dann muss die Spielwelt so variierbar sein, dass sich darin eine ganze Menge gleichartiger Aufgaben formulieren lassen. Optimalerweise haben all diese Aufgaben in den Punkten, die für den zu untersuchenden Prozess relevant sind, genau die gleichen Eigenschaften, unterscheiden sich aber beliebig in irrelevanten Punkten. Im erwähnten Begriffsbildungsbeispiel könnte das bedeuten, dass die Versuchsperson immer wieder Eigenschaftslisten präsentiert erhält, dass es sich dabei aber einmal um die Beschreibung von Bäumen, einmal von Haustieren und einmal von Marsmenschen handelt. (Auch hier gilt natürlich, dass erst die Erfahrung zeigen wird, ob es gelungen ist, die richtigen Aspekte konstant zu halten.)

Zum zweiten ist es dem Simulationsprogramm ebenfalls gleichgültig, wie abstrakt die Aufgabe gehalten ist. Versuchspersonen brauchen dagegen einen gewissen Bezug zur Aufgabe, damit keine Motivationsprobleme auftreten. Im Begriffsbildungsbeispiel könnten die Eigenschaftslisten, die das Programm erhält, durchaus einfach Listen von Zahlen sein. Versuchsperson werden aber mit einem derartigen Input erhebliche Schwierigkeiten haben (vgl. etwa Wason & Johnson-Laird, 1972). Eine minimale semantische Einbettung, wie etwa Beschreibungen von Marsmenschen, ist deshalb notwendig. Wieweit aber durch diese Einbettung der Versuchsperson Informationen zur Verfügung gestellt werden, die das Simulationsprogramm nicht nutzt und nicht nutzen kann, ist sorgfältig zu prüfen (die Einbettung könnte z.B. einen unerwünschten Anschluss an Vorwissen ermöglichen).

Und zum dritten ist es dem Simulationsprogramm gleichgültig, wie schwierig die gestellte Aufgabe ist. Frustriert wird höchstens der Programmierer, wenn sie sich hartnäckig einer Lösung widersetzt. Hingegen sollte sie für Versuchspersonen einen mittleren Schwierigkeitsgrad aufweisen, damit diese motiviert daran arbeiten können. Da sich aber die Schwierigkeit der Aufgabe apriori kaum abschätzen lässt, ist es von Vorteil, wenn sie problemlos auf dieser Dimension variiert und somit den Möglichkeiten der Versuchspersonen angepasst werden kann. Lässt man zudem die gleichen Versuchspersonen mehrmals ähnliche Aufgaben lösen (z.B. um Experten für diesen Typ Aufgabe zu gewinnen, vgl. Kapitel 6), ist diese Variierbarkeit ebenfalls von Vorteil, weil man so den Schwierigkeitsgrad allmählich steigern kann.

3.2 Beispiele

Wie in allen folgenden Kapiteln werden wir nun versuchen, die bewusst knapp gehaltenen Bemerkungen unter "Allgemeines" anhand von Beispielen weiter auszuführen. Als Beispiele dienen die in Kapitel 2 erwähnten eigenen Projekte sowie von Fall zu Fall geeignete Projekte aus der Literatur.

Bei jedem unserer beiden eigenen Projekte stand am Anfang eine längere Suche nach einer geeigneten Spielwelt. Wir möchten diese Erfahrungen herbeiziehen um folgende Punkte zu illustrieren: (1) Lernprojekt: Entwicklung einer "Phänomenologie" und Umsetzung in eine Spielwelt, sowie Weiterentwicklung der Spielwelt im Verlauf der Arbeit. (2) Problemlöseprojekt: Aufbau einer Spielwelt, die gleichzeitig den Bedürfnissen der Simulation und den Bedürfnissen möglicher Versuchspersonen gerecht wird.

3.2.1 Elementare Lernprozesse

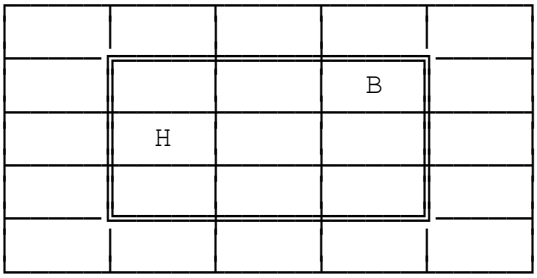
Unser Lernprojekt ging von der Frage aus, welche Lernprozesse minimal notwendig sind, damit Lernen in der Art, wie wir es beim Menschen beobachten, überhaupt möglich ist. Uns war dabei klar, dass wir mit dieser Fragestellung noch mehr als gewöhnlich nur dann eine Chance hatten, zu einer Antwort zu gelangen, wenn die Spielwelt nicht in entscheidenden Punkten übermässig stark vereinfacht war. Wir verwendeten deshalb einige Anstrengungen darauf, eine möglichst "komplexe" Spielwelt zu konstruieren.

Zu diesem Zweck sammelten wir als erstes eine völlig unstrukturierte Liste von Phänomenen, mit denen sich nach unseren Vorstellungen ein menschlicher Lerner auseinandersetzen muss. Es ergaben sich folgende Punkte (vgl. Kaiser & Keller, 1985 für eine etwas ausführlichere Darstellung):

1. Wahrnehmungssystematisierung: Ein Teil des Lernens besteht daraus, Regelmässigkeiten im sensorischen Input bzw. in den Veränderungen dieses Inputs zu entdecken.
2. Inter-modale Koordination: Der Input erfolgt über mehrere, unterscheidbare Kanäle gleichzeitig.
3. Exploration: Jeder menschliche Lerner wirkt ständig auf seine Umwelt ein.
4. Kontrolle: Die Umwelt ist im allgemeinen nicht vollständig kontrollierbar.
5. Vorstrukturierte Umwelt: Oft ist die Umwelt von einem "Tutor" zu didaktischen Zwecken speziell arrangiert.
6. Symbolisch kodierte Information: Über die gleichen Wahrnehmungskanäle erhält der Lerner symbolisch kodierte und nicht symbolische Information.
7. Belohnung und Bestrafung: Gewisse Inputs haben eine klare symbolische Bedeutung als "Belohnung" bzw. "Bestrafung".
8. Hineinwachsen in vorgegebene Konzepte: Die in unserer Kultur von Erwachsenen benutzten Konzepte sind viel zu komplex, als dass jedes Kind sie selbst entdecken könnte.
9. Koordination von Eigenaktivität und Aussensteuerung: Eigenaktivitäten (Wahrnehmungssystematisierung und Exploration) und "didaktische" Interventionen laufen nicht automatisch synchron.
10. Sprachspiele: Sprachliche Begriffe werden vom "Tutor" nie einheitlich und konsequent gebraucht.
11. Voraussetzungen: Lernprozesse beginnen nicht bei Null, sondern bauen auf vorhandenem Vorwissen auf.
12. Fehlende Voraussetzungen: Das Vorwissen analysiert die Umwelt nicht notwendigerweise in einer Form, die für die aktuelle Aufgabe brauchbar ist.

Dies ist eine recht bunte Liste. Sie ist auch kaum vollständig, zeigt aber deutlich, welches Spektrum von Phänomenen eine allgemeine Spielwelt für Untersuchungen zum Thema Lernen umfassen müsste. Da uns das für den Anfang etwas zu anspruchsvoll erschien, konzentrierten wir uns in einem ersten Schritt auf Eigenaktivitäten des Lerners und nicht-symbolisch vermittelte Eingriffe eines Tutors (alle Punkte ausser 6,7 und 10).

Wir glaubten all diese Phänomene mit folgender Spielwelt verwirklichen zu können: Sie besteht aus fünf mal fünf Positionen (Figur 2) oder Orten, an denen sich zu einem bestimmten Zeitpunkt ein oder mehrere Gegenstände befinden können.



Figur 2: Eine Lernwelt

Als "Gegenstände" sind vorhanden ein Ball (B) und die Hand (H) des Lerners. Der Ball bewegt sich von Zeit zu Zeit zufällig (Punkt 4 der Phänomenliste). Der Lerner nimmt die Umwelt über ein Auge und über taktile Information der Hand wahr (2). Das Auge erfasst einen drei mal drei Ausschnitt der Welt, die taktile Wahrnehmung registriert, ob sich Ball und Hand im gleichen Feld befinden (4). Für die visuelle Wahrnehmung sind die neun Felder des Gesichtsfeldes durchnummeriert und der Lerner erfährt einfach, in welchem Feld sich die Hand bzw. der Ball gerade befindet, ohne dass die visuelle Information direkt irgendwelche räumlichen Beziehungen enkodieren würde (11,12). Der Lerner kann das Blickfeld und die Hand in jedem Zeittakt jeweils je um ein Feld in eine der vier Richtungen bewegen. Auch die Null-Bewegung, die nichts bewirkt, ist für ihn eine Aktion (3). Ein Tutor kann eingreifen, indem er den Ball von sich aus verschiebt oder an einer Stelle festhält (5). Die Aufgabe des Lerners ist es, die Hand möglichst ständig in Kontakt mit dem Ball zu halten, der Tutor kann ihn dabei unterstützen, indem er den Ball geeignet positioniert (9). Der Lerner muss zu diesem Zweck im Wesentlichen einerseits die Korrelation zwischen visuellem und taktilem Input erkennen und andererseits lernen, wie seine Aktionen die visuelle Konstellation verändern (1).

Diese Spielwelt ist einerseits natürlich recht "vereinfacht", wie es sich für eine Spielwelt gehört. Andererseits hat sie sich aber als äusserst komplex erwiesen, indem sie den Lerner vor eine Fülle von Problemen stellt. Für uns war sie also eine "gute", da äusserst anregende, Spielwelt. Ob und an welchen Stellen sie die "Realität" für unsere Zwecke unzulässig vereinfacht, können wir beim jetzigen Stand des Projektes nicht sagen. Ausnahmen sind gewisse Punkte, an denen wir bereits gezwungen waren, sie zu modifizieren.

Einmal haben sich gewisse Aspekte der Spielwelt, von denen wir zu Beginn annahmen, sie seien für die eigentliche Aufgabe irrelevant, später als wichtig erwiesen. Anfangs gingen wir davon aus, dass das Verhältnis zwischen der Grösse des Gesichtsfeldes und der Grösse der Gesamtumgebung willkürlich ist. Wir stellten aber bald fest, dass eine frei bewegliche Hand sich bei einem eher kleinen Gesichtsfeld so häufig ausserhalb desselben befindet, dass der Lerner erheblich Mühe hat, überhaupt etwas zu lernen. Um die Lernchancen zu erhöhen, darf also entweder das Gesichtsfeld nicht zu klein sein, oder der Aktionsradius der Hand muss mit dem Gesichtsfeld gekoppelt werden. Beim Kleinkind dürfte vor allem das zweite der Fall sein, zumal Reflexe wie der tonische Nackenreflex (Ayers, 1984, S.22-23) ebenfalls in diese Richtung wirken. Wir haben in unserer Spielwelt die erste Variante verwirklicht, indem das Gesichtsfeld über einen Drittel der gesamten Welt umfasst.

Später stellten wir fest, dass auch die Aufgabe modifiziert werden musste. Wir hatten sie in der Erwartung, dass der Lerner zur Lösung der Aufgabe eine räumliche Repräsentation der Umwelt würde aufbauen müssen, festgelegt, denn wir waren an entsprechend leistungsfähigen Lernprozessen interessiert. Bei genauerer Analyse der Aufgabe zeigte sich aber, dass er nur lernen muss, wie er Bewegungen des Balles durch Handbewegungen kompensieren kann. Denn ist die Hand einmal beim Ball, was früher oder später rein zufällig eintritt, dann reduziert sich die Aufgabe effektiv darauf, diesen nicht mehr zu verlieren. Um den Lerner doch noch zum Aufbau räumlicher Repräsentationen zu zwingen, führten wir deshalb in der Spielwelt noch einen Korb ein, in den der Lerner den Ball befördern musste. Waren nun Ball und Korb weit genug auseinander, so dass der Lerner nicht beide gleichzeitig sehen konnte, musste er sich in irgendeiner Form die Position des Korbes merken, wozu er nun wirklich eine Art räumliche Repräsentation benötigte.

3.2.2 Problemlösen zu zweit

Untersuchungsgegenstand dieses Projektes waren Problemlösegespräche zwischen zwei Partnern. Festzulegen war also in Form einer Spielwelt der Gegenstand, über den gesprochen wurde, das Problem, dass dieser Gegenstand bieten sollte, und die Sprache, in der über den Gegenstand gesprochen werden konnte.

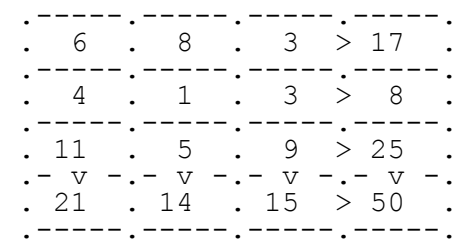
Da, wie unser kurzer Überblick über die Geschichte der Cognitive Science im ersten Kapitel gezeigt hat, Sprachverarbeitung im Allgemeinen nur schwer auf einem Computer realisiert werden kann, und da die eigentliche Sprachverarbeitung nicht im Zentrum des Interesses stand, setzten die Überlegungen zum Aufbau der Spielwelt bei der Wahl der Sprache ein. Ei-

nerseits bestand also aus simulationstechnischen Gründen die Forderung, dass die Sprache möglichst "einfach" sein sollte, wobei hier "einfach" einen kleinen Wortschatz und eine einfache Grammatik bedeutet. Dadurch kann die Sprachverarbeitungskomponente im Simulationsprogramm entsprechend einfach gehalten werden. Auf der anderen Seite wollten wird denselben Gesprächsgegenstand aber auch in Experimenten mit menschlichen Gesprächsteilnehmern verwenden. Die Sprache sollte deshalb für Versuchspersonen möglichst natürlich wirken, so dass sich einigermassen ungezwungene Gespräche ergeben würden. Beides gleichzeitig liesse sich durch einen Gesprächsgegenstand verwirklichen, über den Versuchspersonen spontan in einer im obigen Sinn einfachen Sprache sprechen würden.

Der Gesprächsgegenstand selbst durfte aber nicht zu einfach sein. Denn einmal gingen wir von der Vermutung aus, dass eine Ursache von Schwierigkeiten bei Problemlösedialogen darin liegt, dass die Partner den Gegenstand unterschiedlich repräsentiert haben. Der Gegenstand musste also komplex genug sein, so dass er tatsächlich unterschiedlich repräsentiert werden konnte. Auch hier sollten aber solche Unterschiede im Simulationsprogramm ohne grossen Aufwand darstellbar sein.

Und zum dritten musste der Gesprächsgegenstand in Punkten, die vermutlich für die untersuchten Kommunikationsprobleme nicht von Bedeutung waren, variabel sein, so dass sich eine grosse Menge unterschiedlicher Aufgaben konstruieren liess. Er musste aber auch in Aspekten, die den Dialog beeinflussen würden, eine gewisse Variabilität aufweisen, damit sich Aufgaben unterschiedlicher Schwierigkeit stellen liessen.

Wie wir, ausgerüstet mit diesem Anforderungskatalog, schliesslich eine geeignete Spielwelt fanden, lässt sich nicht mehr rekonstruieren. Der Prozess benötigte jedenfalls einige Zeit. Als Gesprächsgegenstand verwendeten wir schliesslich Zahlenquadrate, wie Figur 3 eines darstellt.



Figur 3: Beispiel eines Gesprächsgegenstandes

Jedes Feld enthielt eine ganze Zahl, wobei den Gesprächspartnern zu Beginn des Gesprächs jeweils nur einige davon bekannt waren. Zudem bestanden zwischen diesen Zahlen unterschiedliche Zusammenhänge. Im Beispiel etwa ergeben in jeder Zeile bzw. Kolonne die ersten drei Zahlen summiert die vierte.

Über derartige Zahlenquadrate lässt sich wirklich in einer sehr einfachen Sprache sprechen. Auch ist diese Art von "Gegenständen" zwar jedermann bekannt, aber trotzdem relativ weit von der Alltagserfahrung entfernt, so dass bei den meisten Personen eine sehr einfache, hochnormierte Sprache existiert, in der sie darüber sprechen. Die meisten werden es als ganz natürlich empfinden, über "Kolonnen", "Zeilen" und "Summen", etc. zu sprechen.

Welche Zahlen genau in welchem Feld stehen, dürfte das Gespräch kaum beeinflussen, so dass eine grosse Zahl ähnlicher Gegenstände zur Verfügung stand. Unterschiede in der Repräsentation liessen sich einmal dadurch erreichen, dass den beiden Partnern unterschiedliche Zusammenhänge bekannt waren (im Beispiel etwa dem einen nur die Summen innerhalb der Zeilen, dem anderen nur die innerhalb der Kolonnen). Zum zweiten konnte man aber auch ganz einfach Zeilen oder Kolonnen vertauschen, etc.

Die eigentliche Aufgabe bestand darin, dass jeder der Partner zu Beginn des Gesprächs ein unvollständiges Zahlenquadrat erhielt und dass sie gemeinsam versuchen mussten, gewisse leere Felder zu füllen. Durch die Variation der zu Beginn gefüllten Felder und der bestehenden Zusammenhänge liessen sich Aufgaben ganz unterschiedlicher Schwierigkeit erstellen.

Diese Spielwelt hat sich sehr bewährt. Einmal erwies sie sich als optimale "Versuchspersonen-Spielwelt", da die daraus resultierenden Kommunikationsspiele von den meisten Teilnehmern begeistert stundenlang bearbeitet wurden. Zum zweiten erlaubte sie auf der Simulationsseite mit ausserordentlich geringem Programmieraufwand einige interessante Entdeckungen (Kaiser, 1985, 1987).

3.2.3 LOGO Debugger

Eine interessante MINCS, die ohne eigentliche Spielwelt auskommt, stellt der von Klahr und Carver verwendete "LOGO Debugger" dar (Carver, 1986; Klahr & Carver, 1988). LOGO ist eine speziell einfache Programmiersprache, die bereits Grundschülern einen Einstieg in die Informatik ermöglicht (Papert, 1980). Dabei müssen diese u.a. auch lernen, wie man in einem fehlerhaften Programm die Fehlerstelle (bug) findet und den Fehler anschliessend behebt (debugging). Das Ziel von Klahr und Carver war es, den dazu notwendigen Denkprozess zu simulieren.

Die "Welt", in der dieses Simulationsprogramm lebt, sind also LOGO Programme. Diese können ihm problemlos ohne weitere Vereinfachungen präsentiert werden, denn LOGO-Programme, wie sie sich für Kinder eignen, sind bereits sehr einfach. (Wir werden im nächsten Kapitel allerdings sehen, dass dies doch nicht ganz so einfach ist.)

3.2.4 Syllogismen

Johnson-Laird und Steedman benutzten in einer MINCS für eine von ihnen durchgeführte Untersuchung des syllogistischen Schliessens zwei verschiedene Spielwelten parallel: Eine für das Simulationsprogramm und eine für die Versuchspersonen (Johnson-Laird & Steedman, 1978). Ihr Ziel war es, einen Schlussfolgerungsprozess zu modellieren, der dieselben Fehler macht, wie man sie bei Menschen beobachten kann. Die Aufgaben, die sie dem Prozess bzw. ihren Versuchspersonen stellten, sind klassische Syllogismen der Art:

```
                Alle Komponisten sind Musiker.
                Einige Komponisten sind Vegetarier.
                ----------------------------------
daraus folgt:   Einige Musiker sind Vegetarier.
```

Prinzipiell standen ihnen dabei als Spielwelt zwei Möglichkeiten offen. Sie konnten entweder direkt derartige konkrete Syllogismen verwenden, oder sie konnten Syllogismen in einer mehr abstrakten Form einsetzen, wie etwa:

```
                Alle A sind B
                Alle B sind C.
                --------------
daraus folgt:   Alle A sind C.
```

Programmtechnisch ist die zweite, die abstraktere Variante die einfachere und entsprechend setzten sie sie auch als Spielwelt für das Simulationsprogramm ein. Da Johnson-Laird und Steedman aber die Erfahrung gemacht hatten, dass "a psychologist who studies reasoning with abstract materials is not so much studying a pure deduction, unsullied by his subjects' knowledge or attitudes, as a very special sort of reasoning designed to compensate for the absence of everyday content." (Johnson-Laird & Steedman, 1978, p. 66; vgl. auch Wason & Johnson-Laird, 1972), wählten sie für ihre Versuchspersonen die konkrete Form. Dies konnten sie tun, weil sie annahmen, dass die beiden Spielwelten in den für die Aufgabe relevanten Aspekten identisch sind. D.h. sie nahmen an, dass die Verwendung konkreter Aufgaben bei den Versuchspersonen zwar ihr "normales" Verhalten auslöst (im Gegensatz zum "abnormalen" Verhalten, dass sie gegenüber abstrakten Aufgaben zeigen), dass dieses normale Verhalten aber auch nicht mehr Information nutzt, als sie in der abstrakten Form der Syllogismen enthalten ist.

Johnson-Laird und Steedman machen im Übrigen in ihrem Artikel auf ein Beispiel dafür aufmerksam, dass bereits formalisierte Beschreibungen der Spielwelt mit Vorsicht zu geniessen sind. Ihre Spielwelt - die Welt der möglichen Syllogismen - ist seit Aristoteles von Logikern systematisch erforscht und beschrieben worden. In dieser Tradition werden 256 verschiedene Arten von Syllogismen formal unterschieden. Aus einem psychologischen Blickwinkel müsste man aber mindestens doppelt so viele, nämlich 512 unterscheiden. Die Differenz entsteht dadurch, dass von einem logischen Standpunkt aus die Reihenfolge der Prämissen keine Rolle spielt. Die beiden Syllogismen

```
Alle Komponisten sind Musiker.
Einige Komponisten sind Vegetarier.
-----------------------------------
Einige Musiker sind Vegetarier.
```

und

```
Einige Komponisten sind Vegetarier.
Alle Komponisten sind Musiker.
-----------------------------------
Einige Musiker sind Vegetarier.
```

sind logisch gesehen äquivalent. Es ist aber durchaus denkbar, dass die unterschiedliche Reihenfolge psychologisch bedeutsam ist und z.B. zu anderen Fehlermustern führt. Ja Johnson-Laird und Steedman behaupten: "One immediate consequence of the slavish adherence to scholastic logic is a general neglect of half of the possible syllogisms, and this omission has had serious consequences for understanding the psychology of syllogisms." (Johnson-Laird & Steedman, 1978, p. 65).

3.3 Zum Nachdenken

1. Benötigen wir für unser Morseprojekt überhaupt eine Spielwelt, oder spielt sich das Abnehmen von Morsezeichen bereits in einem derart reduzierten Realitätsbereich ab, dass wir diesen nicht mehr weiter vereinfachen müssen?
2. Nehmen wir an, wir hätten die zu decodierenden Morsezeichen irgendwo (z.B. in einem file) als eine Sequenz von Punkten, Strichen und Pausen gespeichert und das Simulationsprogramm würde diese Glied um Glied der Reihe nach abrufen. Welchen "Phänomenen" wäre es dann nicht ausgesetzt, die unter Umständen den Erfolg eines menschlichen Morseempfängers beeinflussen?

4 Neutrale Teilprozesse

4.1 Allgemeines

Damit eine MINCS auch wirklich vom Umfang her beschränkt - also "mini" - bleibt, ist es entscheidend, dass im Programm nur die wesentlichen Aspekte des darzustellenden psychischen Prozesses ausformuliert werden. Am besten verschafft man sich deshalb möglichst schnell Klarheit darüber, welches diese Aspekte sind. Denn sonst besteht die Gefahr, dass man sich die Zähne an nebensächlichen Details ausbeisst und nie zur eigentlichen Hauptsache gelangt.

Um genauer umschreiben zu können, was mit "wesentlich" gemeint ist, möchten wir einige Begriffe aus der Modelltheorie einführen. Ein Simulationsprogramm kann als ein Modell betrachtet werden - und zwar gleich in zweifacher Hinsicht. Einerseits kann man es als Modell des Prozesses sehen, der untersucht werden soll (im gleichen Sinn, wie eine Modelleisenbahn ein Modell einer richtigen Bahn ist); andererseits aber auch als Modell einer Theorie über den zu untersuchenden Prozess (im gleichen Sinn, wie eine Regressionsgerade ein Modell-(Fall) eines "je mehr x, desto mehr y" Zusammenhangs ist). Wie schon im ersten Kapitel festgelegt, betrachten wir hier Simulationsprogramme als Modelle im zweiten Sinn, also als "theoretische Modelle", die dazu dienen "Konsistenz, Unabhängigkeit und Vollständigkeit von Theorien" zu überprüfen (Fertig, 1977, S.32).

Intuitiv ist ein Modell ein "Ding", das einem anderen "Ding" in irgendeiner Art ähnlich ist, wobei sich diese Ähnlichkeit bewusst nur auf bestimmte Aspekte beschränkt. Eine Modelleisenbahn ist einer richtigen Bahn ähnlich, indem es Züge, Lokomotiven, Schienen, Bahnhöfe, etc. gibt. Auch gewisse Grössenverhältnisse sind gewahrt. Dagegen besteht keine Ähnlichkeit in der absoluten Grösse (die Modellbahn ist kleiner), der Fahrgeschwindigkeit, der Passagiere, etc. Zweck dieser nur teilweise gewahrten Ähnlichkeit ist es, dass man mit der Modelleisenbahn gewisse Dinge wie Fahren, Rangieren, etc. tun kann, die auch bei einer richtigen Bahn möglich sind, ohne dass man aber dazu denselben materiellen Aufwand betreiben muss, wie das bei der richtigen Bahn notwendig ist. Ebenso ist eine Regressionsgerade einem beliebigen "je grösser x, desto grösser y" Zusammenhang ähnlich, indem tatsächlich (bei positiver Steigung) mit zunehmendem x auch y zunimmt. Die Ähnlichkeit zu einem beliebigen "je-desto" Zusammenhang ist aber natürlich beschränkt, da unendlich viele andere Zusammenhänge neben dem durch eine Gerade dargestellten möglich sind. Zweck der nur teilweise gegeben Ähnlichkeit ist es in diesem Fall, dass es möglich wird, anhand der Regressionsgeraden Gesetzmässigkeiten eines "je-desto" Zusammenhangs exemplarisch zu studieren, ohne dass man sich in den unendlichen Möglichkeiten eines beliebigen "je-desto" Zusammenhangs verliert.

In diesem zweiten Sinn betrachten wir - wie gesagt - das Simulationsprogramm als Modell der psychologischen Theorie, die mit Hilfe dieser Simulation aufgebaut wird. Das Simulationsprogramm sollte ein exemplarisches Beispiel für ein "Ding" sein, auf das alles zutrifft, was in der Theorie behauptet wird (genauso wie die Regressionsgerade ein exemplarisches Beispiel für ein Ding ist, bei dem gilt "je grösser x umso grösser y"). Dabei realisiert des Programm aber natürlich nur eine der durch die Theorie offen gelassenen Möglichkeiten - und erst noch eine relativ atypische, denn als psychologische Theorie macht diese ja Aussagen über Menschen und nicht über Computerprogramme. Zweck dieser Einschränkung auf dieses spezielle exemplarisches Beispiel ist es, dass es - wie eingangs beschrieben - so möglich wird, die Konsequenzen und logischen Verträglichkeiten des in der Theorie Behaupteten besser zu prüfen.

Wenn wir im Folgenden von einem Modell sprechen, dann haben wir immer ein Modell dieser Art im Sinn. Entsprechend werden wird die Begriffe "Modell" und "Simulation" synonym gebrauchen.

Die intuitive Vorstellung davon, was ein Modell ist, lässt sich formal strenger fassen. Unabhängig davon, welche Art von Modell man betrachtet, sind zur Beschreibung eines Modells zumindest folgende fünf Bestimmungsstücke notwendig (Fertig, 1977, Gigerenzer, 1981):

1. Der Modell-Prototyp: Der Prototyp ist das, worauf sich das Modell bezieht, das Original, das abgebildet werden soll. Hier also die Theorie des untersuchten psychischen Prozesses.
2. Das Modell-System: Das System ist das, woraus das Modell gemacht ist. Hier also der Computer und das darauf laufende Programm.
3. Das Modell-Subjekt: Subjekt ist die Person, die das Modell-System als Modell des Prototypen betrachtet. Ein Computerprogramm ist nicht von sich aus ein Modell einer Theorie, sondern wird es nur, wenn jemand es dazu benutzt.
4. Die Modell-Relation: Die Relation legt fest, welche Aspekte des Prototypen durch welche Aspekte im Modellsystem abgebildet werden.
5. Das Modell-Ziel: Der Zweck, zu dem das Modell eingesetzt wird. Aus ihm lässt sich vor allem ableiten, welche Bedingungen erfüllt sein müssen, damit das Modell den Prototypen zufriedenstellend abbildet.

Spricht man von Modellen, dann hat man im Allgemeinen nur das Modell-System im Sinn und bezeichnet es meist einfach als "Modell". Auch wir werden uns hier vor allem mit dem Modell-System beschäftigen. Dabei sind zwei Aspekte von Interesse. Einerseits der "Baukasten" (Modellontologie), mit dessen Hilfe das Modell-System aufgebaut wird (hier der Computer und eine Programmiersprache), und andererseits das "Gebäude", das man aus diesem Baukasten aufbaut (also das lauffähige Programm).

Von der Wahl des "Baukastens" hängt entscheidend ab, was im Modell überhaupt abgebildet werden kann, und damit auch, ob ein bestimmtes Ziel erreichbar ist. Für den "Baukasten" Computer wurde dies ausführlich in Kapitel 1 diskutiert. Im Kapitel 5 werden wird dann noch darauf zu sprechen kommen, was für unterschiedliche Arten solcher "Baukästen" (Programmiersprachen) zur Verfügung stehen.

Um nun genauer definieren zu können, was mit den oben erwähnten "wesentlichen Aspekten des Prozesses" gemeint ist, müssen wir uns dem "Gebäude", bzw. einzelnen "Gebäudeteilen" zuwenden und eine weitere Unterscheidung in negative, neutrale und positive Analogien einführen (Fertig, 1977).

Unter einer negativen Analogie versteht man etwas am Modellsystem, aufgrund dessen das Modell seinen Zweck nicht erfüllen kann. Nehmen wir als Beispiel an, wir wollten untersuchen, wie und warum Menschen beim Wählen von Telefonnummern Fehler machen. Eine entsprechende Theorie würde sicher Aussagen über Fehler machen, die auf Gedächtnisprobleme zurückzuführen sind. Auch das Simulationsprogramm müsste über ein "Gedächtnis" verfügen, in dem es einzelne Telefonnummer festhalten kann. Würden wir nun dieses "Gedächtnis" so konstruieren, dass sich darin beliebig viele Nummern beliebig lange fehlerfrei speichern lassen, dann hätten wir damit eine negative Analogie geschaffen. Denn gewisse Aussagen der Theorie über mögliche Fehler, nämlich Gedächtnisfehler, würden nicht auf das Modell zutreffen.

Das Gegenstück dazu sind positive Analogien. Das sind Aspekte, in denen das Modellsystem den Prototypen dem Ziel entsprechend korrekt abbildet. Baut man im oben erwähnten Beispiel ein begrenztes "Kurzzeitgedächtnis" ein, in dem nur einige wenige Nummern störungsfrei festgehalten werden können, so ergibt dies eine positive Analogie. Selbstverständlich geht es beim Bau eines Modells darum, negative Analogien zu meiden und positive zu fördern, insbesondere die wesentlichen Aspekte des zu modellierenden Prozesses in positive Analogien zu überführen.

Neben positiven und negativen Analogien lassen sich an jedem Modellsystem weitere Aspekte ausmachen, die zwar vorhanden sein müssen, damit das Modell nicht auseinanderfällt, die aber weder als negative Analogien dem Modellziel widersprechen, noch den Anspruch erheben, als positive Analogie etwas am Prototypen abzubilden. Derartige Aspekte des Modellsystems werden als neutrale Analogien bezeichnet. In unserem Beispiel wird das Programm in einer bestimmten Sprache (etwa in Pascal) geschrieben sein. Das ist ein notwendiger Aspekt des Modellsystems, denn in irgendeiner Sprache muss das Programm geschrieben werden. Aber die Wahl von Pascal schafft sicher keine positiven Analogien, denn die modellierte Theo-

rie wird kaum Aussagen darüber machen, dass die entsprechenden Prozesse (auch im menschlichen Gehirn) in Pascal programmiert sind. Es ist auch nicht zu erwarten, dass sich Pascal als negative Analogie erweist und das Modell daran hindert, seinen Zweck zu erfüllen. Es handelt sich dabei also um eine zwar notwendige, aber hinsichtlich des Modellziels neutrale Entscheidung.

Neutrale Analogien sind die Punkte, bei denen man sich im Rahmen einer MINCS sehr viel Arbeit sparen kann, indem man sie mit so wenig Aufwand wie möglich verwirklicht. Zum Beispiel würde es einen völlig unnötigen Aufwand bedeuten und aus der MINCS eine MACS machen, wollte man zuerst selbst eine geeignete Programmiersprache schreiben, anstatt eine der vorhandenen zu verwenden.

Damit dieser Spareffekt bei neutralen Analogien erreicht werden kann, ist entscheidend, dass man sich so früh wie möglich überlegt, was am zukünftigen Modell (bzw. Programm) als positive Analogie gelten soll und also mit dem notwendigen Aufwand behandelt werden muss. Prinzipiell lässt sich das aus den Zielen ableiten, die man mit der Theoriebildung verfolgt, denn die Frage, worüber man in der Theorie genau Aussagen machen möchte, legt ja fest, was am untersuchten Prozess vor allem von Interesse ist. Die genaue Zielanalyse erhält hier also nach der Festlegung der Spielwelt eine zweite wichtige Aufgabe. Die Aspekte des Programms, zu denen man in der Theorie keine Aussagen machen möchte, können dann als neutrale Analogien behandelt werden. Es empfiehlt sich, möglichst rasch festzulegen, wie man diese kostengünstig verwirklichen will, so dass man sich anschliessend der eigentlichen Hauptaufgabe widmen kann.

Die Ausscheidung neutraler Aspekte setzt, wie schon das Festlegen der optimalen Spielwelt, ein gewisses Vorverständnis des Prozesses voraus. Und entsprechend ist es auch hier möglich, dass im weiteren Verlauf der Arbeit eine vermeintlich neutrale Analogie zu einer positiven ausgebaut werden muss. Denn da ja die Theorie eine Theorie über einen empirisch gegebenen Gegenstand sein soll, ist nie ganz auszuschliessen, dass man durch die Empirie später eines Besseren belehrt wird und somit plötzlich Aspekte, die bisher aus der Theorie ausgeklammert waren, explizit in diese aufgenommen werden müssen.

Im Übrigen kann das Auftreten neutraler Analogien auch ein Hinweis darauf sein, dass die Theorie nicht vollständig ist. Wird es notwendig, im Programm Prozesse oder Strukturen zu formulieren, die in der Theorie nicht erwähnt werden, dann kann das natürlich auch bedeuten, dass man ihre Erwähnung schlicht vergessen hat. Punkte, in denen das Programm von der Theorie abweicht, indem es sich entweder anders verhält, als die Theorie aussagt (negative Analogien), oder indem die Theorie keine Aussagen dazu macht, sind also immer Anlass dazu, die Theorie zu überprüfen.

4.2 Beispiele

Allgemeingültige Regeln dafür, wie man die vermutlichen neutralen Analogien am besten behandelt, lassen sich kaum geben. Wir können zur Illustration aber wieder unsere Beispiele heranziehen:

4.2.1 Problemlösen zu zweit

Der "Prototyp" dieses Projektes waren Verständigungsprobleme, die auftreten, wenn zwei Personen gemeinsam im Gespräch eine Aufgabe zu lösen versuchen. Das Ziel der Untersuchung war es dabei, Gesprächsregeln herauszuarbeiten, die - wenn befolgt - solche Probleme verhindern helfen. In erster Annäherung heisst das also, dass das zu diesem Zweck erstellte Simulationsprogramm das Gespräch zwischen zwei Personen darstellen sollte.

Wie bekannt, geschehen in einem solchen Gespräch sehr viele unterschiedliche Dinge gleichzeitig (Grimm & Engelkamp, 1981). Einmal sind die beiden Beteiligten daran, kontinuierlich ihre Beziehung zueinander zu definieren (Schulz von Thun, 1981). Zum zweiten kann das Gespräch für jeden eine Plattform zur Selbstdarstellung bedeuten (Goffman, 1968). Zum dritten produzieren die Beteiligten sprachliche Äusserungen und verarbeiten die Äusserungen des

Partners. Zum vierten wird der ganze Dialog irgendwie gesteuert, so dass er eine bestimmte Richtung nimmt, bestimmte Dinge gesagt werden und andere nicht (Schenkein, 1978), etc. Wie man sich leicht vorstellen kann, würde es den Rahmen einer MINCS bei weitem sprengen, wollte man all dies gleichzeitig behandeln. Wir mussten uns also einschränken und festlegen, was eigentlich im Zentrum des Interesses steht.

Nach einigen Vorstudien stellte sich heraus, dass wir v.a. an Situationen interessiert waren, in denen die beiden Partner den zwischen ihnen ablaufenden Dialog nicht optimal organisieren und so ihr vorhandenes Wissen nicht voll nutzen können. Verständigungsprobleme mehr sprachlicher Natur, wie sie etwa auftreten, wenn der eine Partner eine Formulierung des anderen nicht versteht, also nicht mit seiner Repräsentation des Gesprächsgegenstands verbinden kann, standen für uns nicht im Zentrum. Wir waren auch nicht an Problemen interessiert, die auf der Beziehungsebene der beiden Partner entstehen.

Aspekte unseres Modells, die Phänomene aus der Beziehungsebene des Dialogs oder eigentliche Sprachverarbeitungsprozesse abbilden, sollten also nicht als positive Analogie gelten. Die Beziehungsebene stellte in dieser Hinsicht kein Problem dar, denn da sich Inhaltsebene und Beziehungsebene analytisch gut trennen lassen, ist es ohne weiteres möglich, ein Simulationsprogramm zu schreiben, dass die gesamte Beziehungsebene vollständig ignoriert, so dass diese nicht einmal als neutrale Analogie auftritt. (Allerdings könnte sich erweisen, dass das vollständige Ausblenden der Beziehungsebene sich später als negative Analogie rächt.) Anders ist es mit den Sprachproduktions- und Sprachverarbeitungsprozessen. Soll überhaupt so etwas wie ein Dialog ablaufen, muss der eine oder andere Partner ab und zu etwas sagen und muss sein Gegenüber das Gesagte aufnehmen. Es werden also im Programm notwendigerweise Sequenzen vorkommen, die in irgendeiner Form Sprachproduktion bzw. Sprachwahrnehmung "darstellen".

Wie diese Sprachverarbeitung aber realisiert ist, ist gleichgültig. Das erlaubte uns, die denkbar einfachste Variante zu programmieren. Wir liessen unser Programm die "Gedanken" des einen Partners bei der Sprachproduktion direkt eineindeutig in sprachliche Formulierungen übertragen und beim Gegenüber mit derselben eineindeutigen Zuordnung wieder in denselben "Gedanken" zurückübersetzen. (Prinzipiell hätte man sogar diesen Zwischenschritt weglassen und eine direkte "Gedankenübertragung" arrangieren können. Die sprachliche Zwischenform diente in der Endfassung des Programms v.a. der Darstellung, so dass es während der Dialoge überhaupt etwas zu sehen gab.)

Programmtechnisch stellt dies die billigst mögliche Lösung dar und entspricht somit dem Ziel, die MINCS nicht unnötig aufzublähen. Selbstverständlich kann es sich dabei höchstens um eine neutrale Analogie handeln, da diese Darstellung keineswegs der Komplexität menschlicher Sprachverarbeitung und Sprachproduktion entspricht. Genau genommen sind dabei allerdings nicht sämtliche Details neutral. Als positive Analogie muss gelten, dass einerseits überhaupt Information von einem Partner zum anderen gelangt, und dass andererseits dabei keine Verständigungsprobleme sprachlicher Natur auftreten. Diese Überlegung illustriert, dass die Frage, ob etwas eine positive oder neutrale Analogie darstellt, unter anderem auch eine Frage des Auflösungsniveaus ist, auf welchem man das System betrachtet. Am besten behandelt man jeden Teil des Systems, dessen Ausgestaltung eine neutrale Analogie darstellt, als eine black box. Was im Detail innerhalb dieser black box geschieht, ist für das Modell unwesentlich. Bedeutsam hingegen ist, dass die entsprechende black box überhaupt vorkommt und welche Funktion sie innerhalb des Gesamtsystems erfüllt.

4.2.2 LOGO Debugger

Klahr und Carver erleichterten sich ihre Arbeit am bereits in Kapitel 3 erwähnten "LOGO Debugger" (Carver, 1986; Klahr & Carver, 1988) ebenfalls dadurch, dass sie für ihre Zwecke irrelevante Teile des Prozesses nicht im Programm ausformulierten. Wir haben oben erwähnt, dass man sich oft bei der "Wahrnehmungskomponente" viel Arbeit ersparen kann, indem man den Input dem Programm in vorverarbeiteter Form präsentiert. Genau dies haben Klahr und Carver ausgenutzt.

Das Produkt eines LOGO Programms können verschiedene Dinge sein. Am wohl bekanntesten sind LOGO Programme, die eine Strichzeichnung erstellen. Auch Klahr und Carver verwendeten solche Programme für ihre Untersuchung. Die Aufgabe des LOGO-Debuggers bestand also konkret darin, LOGO Programme, die nicht das gewünschte Bild produzieren, zu korrigieren. Dazu muss das Simulationsprogramm zwei Dinge "wahrnehmen": Erstens das Programm (den Programmcode) selbst und zweitens, dass und inwiefern das vom Programm produzierte vom gewünschten Bild abweicht. Die erste dieser beiden Wahrnehmungsaufgaben ist für ein Simulationsprogramm mehr oder weniger trivial, die zweite hingegen nicht. Da sie nicht speziell an der Lösung dieser zweiten Aufgabe interessiert waren, machten Klahr und Carver deshalb gar keinen Versuch, sie durch das Simulationsprogramm lösen zu lassen, d.h. ihr Programm ist nicht in der Lage Bilder und Abweichungen zwischen Bildern zu "sehen". An den Stellen, an denen das Programm derartige Informationen benötigte, hält es einfach an und bittet den Benutzer, die entsprechenden Entscheidungen einzugeben. Was sie modellierten, war"... when this information is required, and what to do with it, once it is available", aber "there is no model of how this information is extracted from either memory or the environment" (Klahr & Carver, 1988, p. 375). Ihre Begründung dafür entspricht genau dem, was wir in diesem Kapitel besprochen haben: "We do not believe that these unmodeled processes are simple, unimportant, or 'hardware primitives' (..). However, the creation of a perceptual front end that could construct encodings of discrepancies between intended and actual outcomes would be a very complex task that is tangential to our central purpose..." (p. 375).

Wenn wir in Kapitel 3 vorgeschlagen haben, dass man durch eine geeignete Vereinfachung der Spielwelt die Wahrnehmungskomponente praktisch überflüssig machen kann, so gingen Klahr und Carver hier einen anderen Weg. Sie nahmen als Spielwelt die unreduzierte "Realität" - die allerdings relativ wenig komplex ist - und bürdeten die Vorverarbeitung des Inputs dem Benutzer auf. Sie umgingen damit das Problem, dass bei der Reduktion der "Realität" auf die Spielwelt Entscheidendes weggelassen wird. Dafür besteht bei ihrem Vorgehen die Schwierigkeit, dass nur schwer zu entscheiden ist, ob der Benutzer nicht Informationen einbringt, die eigentlich im modellierten Prozess generiert werden müssten. Diesem Problem versuchten Klahr und Carver dadurch zu begegnen, dass sich der Benutzer beliebig dumm stellen kann, d.h. er kann prinzipiell jede Frage des Programms mit "ich weiss nicht" beantworten (vgl. Kapitel 7).

4.2.3 Syllogismen

Johnson-Laird und Steedman wählten in ihren Untersuchungen zum syllogistischen Schliessen nochmals eine andere Möglichkeit, die Wahrnehmungskomponente als neutrale Analogie mit minimalem Aufwand zu behandeln. Ihr Schlussfolgerungsprogramm arbeitet (vermutlich; mit Sicherheit lässt sich das aus ihrem Artikel nicht ableiten) mit einer abstrakten Repräsentation der Syllogismen als Ausgangsdaten, also etwa:

```
Alle A sind B.
Einige A sind C.
---------------
Einige B sind C.
```

Das Verhalten des Simulationsprogramms verglichen sie aber mit dem Verhalten von Versuchspersonen beim Lösen konkreter Aufgaben wie etwa

```
Alle Komponisten sind Musiker.
Einige Komponisten sind Vegetarier.
-----------------------------------
Einige Musiker sind Vegetarier.
```

Grundsätzlich wäre es kein grosses Problem die Simulation "vollständig" zu machen und eine "Wahrnehmungskomponente" zu schreiben, die die abstrakte Repräsentation aus der konkreteren Form ableitet. Das Programm müsste zu diesem Zweck nur gleiche Wörter miteinander

in Verbindung bringen können. Es scheint aber, dass Johnson-Laird und Steedman sich diese Mühe nicht gemacht haben. Wie Klahr und Carver setzten sie Menschen an dieser Stelle ein. Nur geschah dies nicht im aktuellen Ablauf der Simulation. Diese wurde vollständig in Interaktion mit der abstrakten Spielwelt entwickelt. Erst als es darum ging, die Leistungen von Versuchspersonen mit denen des Programms zu vergleichen, mussten sie die Übersetzung vornehmen.

Die Art, wie erkannt wird, dass es sich um einen Syllogismus handelt, der durch eine bestimmte abstrakte Form beschrieben werden kann, wird von Johnson-Laird und Steedman also als neutrale Analogie behandelt (realisiert durch andere, als durch programmtechnische Mittel). Positive Analogie ist dabei nur, dass sie fehlerfrei erfolgt und vor dem eigentlichen Schlussfolgerungsprozess stattfindet. Allfällige Schlussfolgerungsfehler müssen nachher, bei der Bearbeitung der abstrakten Form auftreten.

Andere neutrale Analogien lassen sich in diesem Fall bei den im Schlussfolgerungsprozess verwendeten Repräsentationen finden. Die zentrale Idee von Johnson-Laird und Steedman ist, dass eine Aussage wie " Einige A sind B" in ein mentales Bild der Form

```
A
A     >     B
A     >     B
A     >     B
            B
            B
```

übersetzt wird. D.h. der Problemlöser stellt sich eine beliebige Menge von "A"'s vor die alle "B" sind. Dazu noch einige "A" die nicht "B" sind und eventuell auch noch einige extra "B", die nichts mit den "A" zu tun haben. Durch die zweite Prämisse werden dann weitere "Akteure" ins Bild aufgenommen.

Z.B. könnte "Einige B sind C" folgende Repräsentation zur Folge haben:

```
A
A     >     B
A     >     B
A     >     B         >     C
            B         >     C
            B
```

Aus diesem Gesamtbild ergeben sich dann die Schlussfolgerungen. Dieses Modell erlaubt recht erfolgreich einige typische Fehler vorherzusagen. In unserem Beispiel wäre zu erwarten, dass die Versuchsperson, die ein derartiges Bild der Situation aufbaut, fälschlicherweise annimmt, sie könne daraus "Einige A sind C" ableiten.

Neutrale Analogien sind bei diesem Modell sicher diverse Details des mentalen Bildes. Z.B. wie viele "A"s vorgestellt werden; wie die "A"s repräsentiert sind (als Liste); wie die Verbindungen von den "A"s zu den "B"s realisiert sind (als Pointer, d.h. bei jedem "A" ist die Adresse des entsprechenden "B" gespeichert; Johnson-Laird & Steedman, 1978, p. 77); etc. Die Autoren machen hier einfach von den Möglichkeiten Gebrauch, die ihnen ihre Programmiersprache (POP-2) bietet. Gewisse Aspekte dieser Details sind dann allerdings wieder als positive Analogien gedacht. Die Repräsentation der Verbindungen als pointer erlaubt es, schnell und einfach von einem "A" zu zugehörigen "B" zu gelangen, da ja beim "A" direkt die Adresse des "B" gefunden wird. Umgekehrt existiert kein direkter Weg, d.h. bei den einzelnen "B" findet sich keine Information darüber, welches "A" mit ihnen verbunden ist. Die einzige Möglichkeit dies herauszufinden, besteht darin, alle "A"s durchzugehen, bis das entsprechende gefunden ist. Diese Asymmetrie ist beabsichtigt und dient zur Erklärung, warum Versuchspersonen bei gewissen Schlussfolgerungen leichter überprüfen können, ob sie zutreffen, als bei anderen. Das

Modell von Johnson-Laird und Steedman illustriert, dass manchmal positive und neutrale Analogien eng ineinander verwoben sein können.

4.3 Zum Nachdenken

1. Überlegen Sie sich mindestens drei bis vier ganz unterschiedliche Perspektiven bzw. Zielsetzungen, unter denen man unser Morseprojekt angehen könnte. Mögliche Zugänge: Was interessiert Sie vor allem? Was wollen Sie mit den Resultaten anfangen? Was für Auftraggeber mit welchen Interessen kämen für solch ein Projekt in Frage?
2. Überlegen Sie sich, was unter diesen Zielsetzungen jeweils neutrale und positive Analogien an einem möglichen Simulationsmodell sein könnten.
3. Sollte sich herausstellen, dass Sie dabei für alle Zielsetzungen zur gleichen Aufteilung gelangen, suchen Sie nach einer weiteren Zielsetzung, die zu einer anderen Aufteilung führt.
4. Greifen Sie einen der neutralen Aspekte heraus, und überlegen Sie sich, wie Sie diesen mit ihnen bekannten Mitteln am einfachsten simulieren könnten.
5. Überlegen Sie sich, was genau an der Simulation von Punkt 4 alles neutrale Analogien sind und welche Teilaspekte Sie als positive Analogien betrachten würden.

5 Das Simulationsprogramm

5.1 Allgemeines

5.1.1 Programmierkenntnisse

Kernstück einer jeden Computersimulation ist selbstverständlich ein lauffähiges Programm. Um ein solches herstellen zu können, braucht es Programmierkenntnisse und eine gewisse Vertrautheit mit den gebräuchlichsten Techniken aus dem Bereich der Künstlichen Intelligenz. Dies sind beide Voraussetzungen, die wir hier nicht vermitteln können und auch nicht wollen, da es zu diesen Themen genügend, z.T. hervorragende Lehrbücher gibt. Gute Erfahrungen haben wir mit folgenden Lehrbüchern gemacht:

5.1.1.1 Einführungen in die Techniken der KI

Nilsson (1971) Problem solving methods in Artificial Intelligence.
Gute Einführung in die Verwendung von Suchprozessen als Problemlösemechanismus.

Nilsson (1980) Principles of Artificial Intelligence.
Stark überarbeitete und erweiterte Ausgabe des Buches von 1971; weniger ausschliesslich auf Suchprozesse ausgerichtet.

Winston, 1977 (deutsch 1987) Artificial Intelligence.
Sehr gute und umfassende Einführung.

5.1.1.2 Lehrbücher für Fortgeschrittene und Nachschlagewerke

Charniak, Riesbeck & McDermott (1980) Artificial Intelligence programming
Eigentlich als Einführung gedacht; liest sich aber erst mit Gewinn, wenn schon ein gewisser Hintergrund vorhanden ist.

Pearl (1984) Heuristics.
Umfassende Darstellung von Einsatz und Evaluation von Heuristiken in der KI.

Barr & Feigenbaum (1981) und Cohen & Feigenbaum (1982) The handbook of Artificial Intelligence, Vol. I - III.
Umfangreiches Nachschlagewerk; viele Programme aus der KI-Geschichte werden besprochen; es lohnt sich allerdings kaum, das Werk selbst anzuschaffen, da die einzelnen Artikel zu knapp sind und man deshalb meist noch auf die Originalliteratur zurückgreifen muss.

5.1.1.3 Fallstudien und einflussreiche Artikelsammlungen

Schank (1981) Inside computer understanding.
Bespricht ausführlich einige klassische Programme aus seiner Werkstatt; Anleitungen zum Nachbau reduzierter Versionen der besprochenen Programme.

Bobrow & Collins (1975) Representation and understanding, studies in Cognitive Science.
Interessante Artikelsammlung.

Winston (1975) The psychology of computer vision.
Interessante Artikelsammlung.

Man kann natürlich auch ohne diese Vorkenntnisse an ein Projekt herangehen. Ja es empfiehlt sich sogar, die Grundlagen nicht einfach in Trockenübungen durchzuarbeiten, sondern sie direkt auf ein kleines Projekt zu beziehen, da man jegliche Art von Programmieren am besten "by doing" lernt. Man muss sich in diesem Fall aber klar sein, dass solch ein erster Versuch ein Lehrstück ist, aus dem kaum viel Brauchbares resultieren wird.

Ebenfalls möglich ist es, die Mitarbeit eines erfahrenen Programmierers zu suchen. Hat dieser allerdings noch nie im Rahmen eines KI-Projekts gearbeitet, kann es zu massiven Verständigungsproblemen kommen, da sich konventionelle Informatik und KI in vielen Punkten sehr stark unterscheiden. Wahrscheinlich ist der Einstieg in KI für einen relativen Computerneuling einfacher, als das notwendige Umlernen für den erfahrenen Informatiker.

5.1.2 Programmiersprachen

Etwas ausführlicher können und möchten wir auf die Wahl des geeigneten Instruments (Computer und Programmiersprache) eingehen. Prinzipiell lassen sich Simulationsprogramme in jeder Programmiersprache auf jedem Computer verwirklichen. Dies ist ein Konsequenz davon, dass sämtliche modernen Programmiersprachen aus jedem Computer mindestens eine Turing-Maschine machen. Aber natürlich kann man sich durch die geeignete Wahl der Instrumente die Arbeit mehr oder weniger erleichtern.

Tabelle 1: Klassifikation einiger Programmiersprachen nach Grad der Unterstützung, die sie dem Programmierer gewähren

Steuerung	Datenverwaltung			
	Adressen, Stacks	Variablen	dynamische Strukturen	Beziehungs-Strukturen
prozedural	Assembler	Basic Fortran Pascal, C	(Pascal, C)	Turbo-Pascal C++
funktional	Forth	(Pascal, C, Forth)	LISP	SMALLTALK CLOS
prozess-orientiert	OCCAM	Simula		DCL
deklarativ			Prolog OPS5 PRISM	ART KEE

Neben finanziellen Erwägungen, die sicher immer eine entscheidende Rolle spielen werden, besteht bei dieser Wahl ein fundamentaler Entscheidungskonflikt zwischen Kontrolle und Komfort. Im Extremfall kann man sich auf der einen Seite für ein Instrument entscheiden, bei dem man zwar alles unter Kontrolle hat, aber dann auch alles selbst kontrollieren muss. Oder man wählt auf der anderen Seite eine Programmierumgebung, die zwar vieles selbst erledigt, wo man dann aber nur noch beschränkt bestimmen kann, wie dies geschieht. Die verfügbaren Programmiersprachen lasse sich aus dieser Sicht nach zwei Dimensionen ordnen, nämlich:

1. Wie viel Arbeit übernimmt die Programmiersprache (Programmierumgebung) im Zusammenhang mit der Steuerung des gesamten Verarbeitungsablaufes?
2. Wie viel Arbeit übernimmt die Programmiersprache (Programmierumgebung) im Zusammenhang mit der Verwaltung der Daten?

Grob ergibt sich daraus eine Einteilung in vier mal vier Klassen von Programmiersprachen (Tabelle 1). Die Klassen sind so geordnet, dass bei den Sprachen im Feld oben links der Programmierer am meisten selbst tun muss; nach unten und rechts nimmt der Anteil dessen zu, was vom Programm übernommen wird. Die Ordnung in den beiden Dimensionen ist allerdings nicht absolut zu sehen. Z.B nehmen prozessorientierte Sprachen dem Programmierer zwar

einiges ab, was er in einer funktionalen Sprache selbst erledigen müsste. Andererseits lassen sie manchmal Wünsche unerfüllt, die in einer funktionalen Sprache befriedigt würden.

Für Leser, die mit den von uns in Tabelle 1 gewählten Etiketten für die einzelnen Kategorien nicht vertraut sind, möchten wir hier eine kurze Charakterisierung versuchen.

5.1.2.1 Programmsteuerung

prozedural: Prozedurale Sprachen sind aus Befehlen aufgebaut und ein prozedurales Programm besteht aus einer Aneinanderreihung von Befehlen, die das Programm Schritt für Schritt abarbeitet. Beim Programmieren in einer prozeduralen Sprache muss man sich also überlegen, welche Schritte der Computer der Reihe nach machen soll, wobei man sich dabei auf die Ebene der dem Computer bekannten Schritte begeben muss.

funktional: Die Grundeinheit der funktionalen Sprache ist die Funktion. Eine Funktion ist eine Art selbständiges, kleines Programm, dem man Daten übergibt und von dem man Resultate zurückerhält. Der Programmierer kann sie als black-box behandeln, d.h. er braucht nicht zu wissen, was innerhalb genau abläuft. Jede Funktion organisiert sich selbst. Eine gewisse Menge von Funktionen liegt vordefiniert vor (analog den Befehlen einer prozeduralen Sprache). Darauf lassen sich Ebene um Ebene selbstdefinierte Funktionen aufbauen, die in der Lage sind, immer komplexere Aufgaben zu lösen. Auch beim Programmieren in einer funktionalen Sprache muss man sich also immer überlegen, was das Programm Schritt für Schritt tun soll. Im Gegensatz zu den prozeduralen Sprachen kann man hier aber in selbstgewählten Einheiten denken.

prozessorientiert: Die Grundeinheit ist hier der Prozess. In prozeduralen und funktionalen Sprachen spielt sich der ganze Programmablauf innerhalb eines Prozesses ab. Prozessorientierte Sprachen dagegen erlauben es, mehrere Prozesse zu formulieren, die z.T. unabhängig voneinander und parallel zueinander ablaufen können. Die einzelnen Prozesse selbst sind dann jeweils wieder prozedural (OCCAM, Simula) bzw. funktional (DCL) formuliert.

deklarativ: Bei den deklarativen Sprachen entscheidet der Computer selbst, was er Schritt für Schritt tut; der Programmierer braucht sich nicht mehr darum zu kümmern. Die Aufgabe des Programmierers ist es nur, das zu lösende Problem und das dazu notwendige Wissen in eine Form zu bringen, die vom Computer verstanden wird. Hier gibt es unterschiedliche Formen. Prolog erwartet z.B. dass man die Aufgabe als logisches Problem formuliert, derart, dass aufgrund einer Menge gegebener Sätze ein bestimmter Satz zu beweisen ist (es handelt sich also um eine "logische" Sprache im engeren Sinn). OPS5 andererseits erwartet eine Menge von Regeln der Form "Wenn das und das der Fall ist, dann tue das und das" und versucht dann das Problem zu lösen, indem es alle ihm bekannten Regeln anwendet (technisch gesehen handelt es sich um ein Produktionssystem).

5.1.2.2 Datenverwaltung

Adressen und Stacks: In adress- und stackorientierten Sprachen werden die Daten an physikalisch genau definierten Plätzen im Speicher des Computers abgelegt. Der Programmierer muss also jederzeit einmal genau wissen, welche Daten im Moment welche Plätze belegen und wo noch wie viel Platz zur Verfügung steht. Er ist auch für den Transport der Daten von Platz zu Platz verantwortlich, wenn dies im Zuge der Verarbeitung notwendig ist.

Variablen: Bei variablen-orientierten Sprachen spricht der Programmierer Daten mit einem Namen an. D.h. er definiert zuerst einmal, wie ein bestimmtes Datum heissen soll und von da an versteht der Computer, was gemeint ist, wenn dieser Name gebraucht wird. Der Programmierer braucht sich nicht mehr darum zu kümmern, wo die entsprechende Information physikalisch im Speicher lokalisiert ist, sondern das Programm führt selbst Buch darüber. Der Programmierer muss sich nur noch überlegen, wie viele Daten von welcher Art er hat, so dass das Programm im Voraus den nötigen Speicherplatz organisieren kann.

Dynamische Strukturen: Bei Sprachen mit einer dynamischen Datenstruktur spricht der Programmierer die Daten zwar auch mit Variablennamen an. Die Strukturen, die sich hinter diesen

Variablen verstecken, können aber beliebig wachsen. Ein typisches Beispiel dafür sind die Listen im LISP. Jede Liste hat wie eine Variable einen Namen, sie kann aber beliebig lange werden. Auf diese Art lassen sich dynamisch (während des Programmablaufes) Strukturen definieren, deren Umfang und genaues Aussehen beim Programmieren nicht bekannt sein müssen.

Beziehungs-Strukturen: Programmiersprachen dieser Art enthalten (meist zusätzlich zu den üblichen Datenstrukturen) vordefinierte Strukturtypen (Objekte), für die vordefinierte Beziehungen bestehen. ART z.B. kennt eine vordefinierte frame-Struktur (dort Schema genannt). Diese frames können unter anderem in einer hierarchischen Beziehung stehen, in der ein frame die slots und deren Inhalte von den übergeordneten frames erbt. Wird nun irgendwo in einem frame ein slot geändert, so erfolgt diese Änderung automatisch auch für alle diesem frame untergeordneten frames. (Ein derartiger Vererbungsmechanismus wird auch beim sogenannten "objektorientierten Programmieren" verwendet.) Eine Vielzahl weiterer Beziehungen sind denkbar. Mechanismen dieser Art erlauben den Aufbau von Wissensstrukturen mit komplexen Abhängigkeiten, ohne dass der Programmierer sich selbst darum kümmern muss, dass aus jeder Veränderung alle notwendigen Konsequenzen gezogen werden.

Jede dieser Klassen hat ihre Vor- und Nachteile, die sich aus dem schon erwähnten trade-off zwischen Komfort und Kontrolle ergeben. Generell lässt sich folgendes sagen: Je mehr Komfort eine Sprache bringt, umso schneller kann man in ihr eine erste, lauffähige Variante eines Simulationsprogramms schreiben. Allerdings hat man dabei über viele technische Details keine Kontrolle. Das führt oft dazu, dass das Programm langsam läuft und viel Platz braucht. Um dies zu kompensieren benötigt man mehr Resourcen, d.h. grössere und schneller Computer. Zudem kann es bei einer Sprache mit viel vorgegebenen Strukturen geschehen, dass diese quer zu dem liegen, was man eigentlich tun möchte, und man deshalb nur mit vielen Tricks ans Ziel kommt.

In einer Sprache mit wenig Komfort muss man sich dagegen zwar um alles selbst kümmern, man hat als Konsequenz davon aber auch alles unter Kontrolle. Diese Kontrolle bedeutet, dass man massgeschneiderte Lösungen produzieren kann, die mit wenig technischem Aufwand ablaufen können. Erreicht allerdings das Programm eine gewisse Komplexität, kann es unmöglich sein, ohne Unterstützung durch vorgegebene Strukturen diese Kontrolle noch auszuüben (es sei denn, man ist in der Lage, diese Strukturen selbst zu schaffen, was allerdings einige Erfahrung voraussetzt). Sprachen mit wenig Komfort kommen also mit weniger technischem Aufwand aus, setzen aber im Allgemeinen eine längere Einarbeitungszeit voraus.

5.1.2.3 Sprachumfang

Quer zu den bisher getroffenen Einteilungen liegt noch eine weitere Dimension, die es zu berücksichtigen gilt. In jeder der gebildeten Klassen lassen sich die Sprachen noch danach unterscheiden, wie reichhaltig sie sind, d.h. wie viele unterschiedliche vordefinierte Befehle/Funktionen bzw. Variablen/Struktur-Typen existieren. Gibt es nur wenig vordefinierte Elemente, dann kennt man diese relativ schnell, muss aber auch oft recht umständlich Konstruktionen aufbauen, da ja alles auf diese wenigen Elemente zurückzuführen ist. Existieren dagegen viele vordefinierte Elemente, hat man zwar das Problem, dass es aufwendig wird, die Sprache in ihren Finessen überhaupt zu lernen; dafür ist sie dann entsprechend ausdrucksstärker.

5.2 Beispiele

5.2.1 Elementare Lernprozesse

Wir haben im Verlauf dieses Projekts BASIC, Lisp, Forth und Assembler eingesetzt - und zwar aus den folgenden Gründen:

Als die Idee zum Projekt geboren wurde, stand nur ein PC mit einem eher wenig benutzerfreundlichen BASIC zur Verfügung. Da wir nicht warten wollten, bis weitere Mittel bewilligt waren, begannen wir unsere Vorstudien einmal mit dem, was wir hatten. Die verschiedenen Ver-

suche mit kleineren Programmen, die wir anstellten, halfen uns wesentlich, unsere Vorstellungen zu konkretisieren und die Spielwelt aufzubauen. Beides erwies sich für den Projektantrag als nützlich. Darüber hinaus programmierten wir aber auch einen ersten funktionstüchtigen Lernprozess (das zugehörige "Gedächtnis" ist im vierten Kapitel kurz beschrieben) und wir wurden dadurch auf ein Problem aufmerksam, dass uns vorher nicht bewusst gewesen war (vgl. Kapitel sechs). Es wäre durchaus möglich gewesen, in Richtung dieses ersten Ansatzes weiterzumachen und dann auch beim BASIC zu bleiben.

Da wir aber eine andere Richtung einschlugen, trafen wir auf eine typische Grenze, die sich mit variablenorientierten Sprachen wie BASIC kaum überwinden lässt. Für das in Kapitel vier beschriebene "Gedächtnis" unseres ersten Lernprogrammes war es problemlos möglich, die Grösse der entsprechenden Häufigkeitstabelle von Anfang an festzulegen. Denn wieviele mögliche Kombinationen aus Ausgangssituationen, Aktionen und Endsituationen in der Spielwelt vorkommen können, liess sich leicht berechnen. Wir wollten das Programm nun aber so erweitern, dass es in der Lage war, mit der Zeit einzelne Situationen, in denen nur ein Gegenstand sichtbar war, nach der vermuteten Position des zweiten Gegenstandes weiter zu diskriminieren. Welche neuen Situationen das Programm "erfinden" würde, liess sich nicht vorhersagen - zumindest solange nicht, bis wir den Lernprozess perfekt im Griff hatten. Und deshalb ergab sich der Bedarf nach einer Sprache mit einer dynamischen Datenverwaltung, welche die Bildung beliebig vieler neuer Situationen zulassen würde.

Unsere Wahl war Lisp auf einer möglichst schnellen Maschine. Dass wir eine funktionale und nicht eine deklarative Sprache wählten, hatte zwei Gründe. Einmal liess sich unsere Grundstruktur, nämlich das Situations-Aktions-Situations-Tripel in keiner der vorgegeben Strukturen zwanglos abbilden. Produktionsregeln z.B. beinhalten nur Ausgangssituation-Aktions-Paare und haben keinen Platz für die Erwartung, mit welcher Wahrscheinlichkeit das gewünschte Resultat auftritt. Also wählten wir eine Sprache, die genügend Flexibilität besass, um unsere eigenen Strukturen zu formulieren. Zum zweiten hatten die Versuche mit den BASIC-Programmen gezeigt, dass unser Lerner sehr viele Erfahrungen benötigte, um seine Umwelt kennenzulernen (in der Grössenordnung von mehreren hunderttausend Aktionen). Hätten wir das Ausführen einer Aktion etwa in OPS5 mit dem Feuern einer Produktionsregel dargestellt, dann hätte ein Simulationslauf viel zu lange gedauert (ausser vielleicht auf Spezialmaschinen, die ausserhalb unseres Budgets lagen).

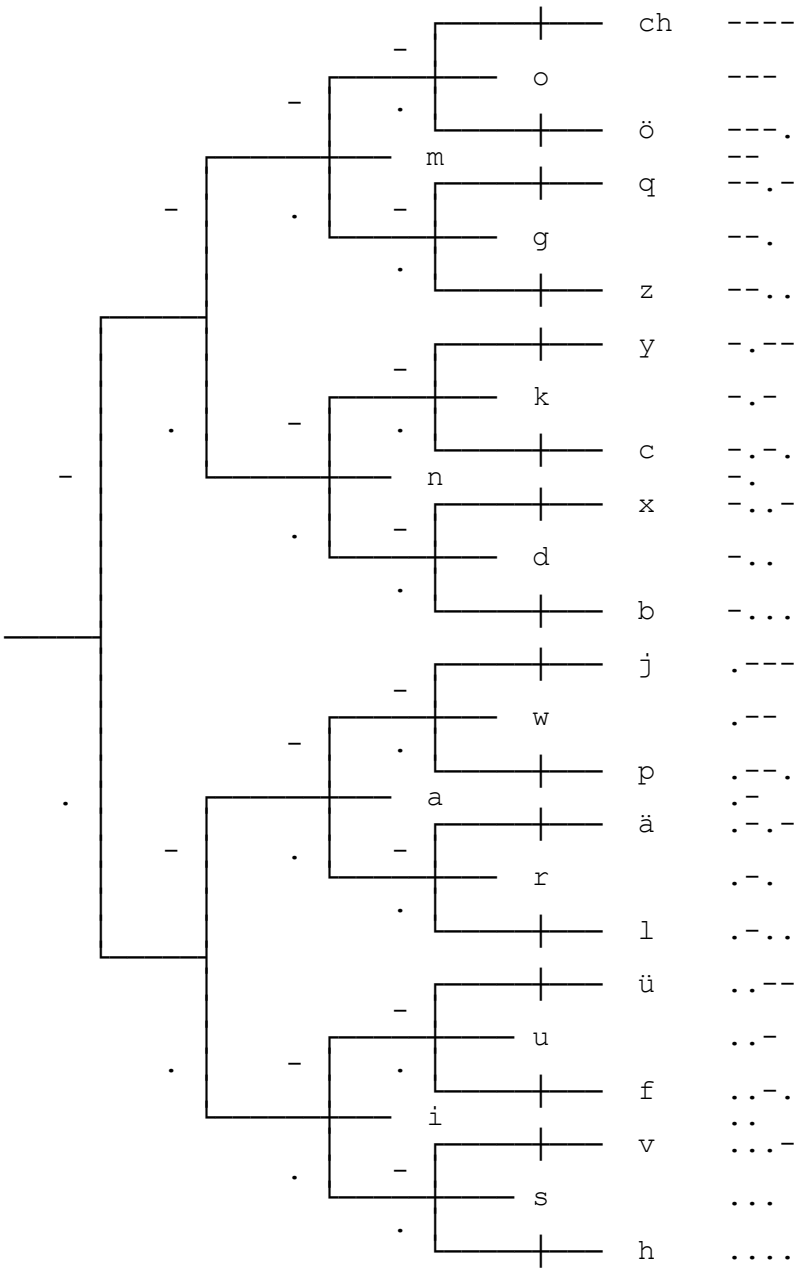
Die Hauptarbeit führten wir dann also in Lisp aus. Wie in solchen Fällen üblich, definierten wir zuerst eine gewisse Anzahl von Funktionen, die einen guten Teil der Datenverwaltung und der Programmsteuerung für unsere speziellen Strukturen übernahmen, so dass wir uns um gewisse Details nicht mehr zu kümmern brauchten. In einem gewissen Mass schrieben wir uns also unsere eigene deklarative Sprache mit einer "Beziehungs-Struktur" (Viele KI-Programmiersprachen sind auf diese Art aus Lisp entstanden, vgl. auch Allen, 1978, Winston, 1987).

In dieser selbstgebauten Sprache schrieben wir dann einige Programme, welche Prozesse darstellten, mit deren Hilfe unser simulierter Lerner in der Lage war, einige allgemeine Gesetzmässigkeiten in der Umwelt zu entdecken. Allerdings schienen alle Lernprozesse, die wir erfanden, an einem bestimmten Punkt auf unüberwindliche Schwierigkeiten zu stossen. Diesen Punkt genauer zu untersuchen war insofern schwierig, als unsere Simulationen z.T. schon mehr als eine Woche laufen mussten, bis er erreicht war. Das machte es praktisch unmöglich, schnell ein paar Varianten von Prozessen durchzutesten.

Wir programmierten deshalb in Forth (und z.T. in Assembler) eine reduzierte Version des Simulationsprogramms, die erlaubte, genau diesen Punkt zu klären. Diese reduzierte Variante lief bis zu zwanzigmal schneller - wobei etwa die Hälfte auf die Reduktion und die Hälfte auf die Verwendung einer weniger komfortablen Sprache zurückzuführen war. Dass es uns aber überhaupt möglich war, ein solches Programm in Forth zu schreiben, ist nur darauf zurückzuführen, dass wir den Prozess bereits in Lisp konzipiert und getestet hatten. Es wäre für uns unmöglich gewesen, einen Prozess dieser Komplexität ohne diese Vorarbeit direkt in Forth zu schreiben.

5.3 Zum Nachdenken

1. Die zum Decodieren von Morsezeichen notwendige Information lässt sich gut als Entscheidungsbaum darstellen. Ausgehend von einem Startknoten verzweigt sich dieser Baum immer wieder in zwei oder drei Äste, wobei der erste Ast für "Punkt" und der zweite für "Strich" steht (vgl Figur 4). Der dritte entspricht dem Zeichenende und führt zu einem Endknoten, an dem sich das decodierte Zeichen ablesen lässt. Der Decodierungsprozess kann dann als Abarbeiten eines derartigen Entscheidungsbaums verstanden werden. (Feigenbaum hat diesen Prozess bereits früh in seinem EPAM eingesetzt, das gleichzeitig in der Lage war, den Baum anhand von Beispielen zu lernen (Feigenbaum, 1963). In erweiterter Form als ATN findet die Idee heute in vielen Sprachverarbeitungsprogrammen Verwendung (Winograd, 1983).) Versuchen sie das Abarbeiten eines solchen Baumes "prozedural", d.h. als Sequenz einzelner Befehle darzustellen.
2. Versuchen sie den gleichen Prozess wie in Aufgabe 1 "funktional", d.h. als rekursiver Aufruf von Funktionen darzustellen.
3. Überlegen Sie sich, ob Prolog eine geeignete Sprache für unser Mini-Projekt wäre, indem Sie versuchen, die Aufgabe als logisches Problem darzustellen.
4. Machen sie dieselbe Überlegung für eine Produktionssystemsprache, indem sie versuchen, das zum Dekodieren der Morsezeichen notwendige Wissen in Wenn-Dann-Regeln zu fassen.



Figur 4: Ausschnitt aus dem Morsecode dargestellt als Entscheidungsbaum

6 Heuristische Empirie

6.1 Allgemeines

Beim Aufbau einer psychologischen Theorie - und damit auch beim Aufbau eines Simulationsprogramms als Modell dieser Theorie - lassen sich grob zwei Phasen unterscheiden: Theoriebildung und Theorieprüfung (Wottawa, 1977; "context of discovery" versus "context of justification", Herschel, 1830, Herzog, 1984; "Hypothesengewinnung" versus "Überprüfung von Hypothesen", Bortz, 1984). Bei einer empirischen Theorie geht es in der zweiten dieser Phasen selbstverständlich darum, Theorie und Empirie miteinander zu konfrontieren. D.h. die Bewährung gegenüber der Empirie ist das zentrale Kriterium, an dem die Theorie gemessen wird (vgl. Kapitel 8). In der vorangehenden Theoriebildungsphase dagegen steht der Bezug zur Empirie nicht derart ausschliesslich im Zentrum, sondern andere Kriterien, wie die Konsistenz der Theorie, ästhetische Aspekte (Einfachheit), etc. spielen ebenfalls eine wichtige Rolle (Losee, 1977). Damit aber eine Theorie entsteht, die sich mit einer gewissen Wahrscheinlichkeit in einer Überprüfung an der Empirie bewähren wird, ist es entscheidend, dass der empirische Bezug schon während des Aufbaus der Theorie bzw. des Simulationsprogramms im Auge behalten wird.

Diese Auseinandersetzung mit der Empirie braucht während der Aufbauphase nicht streng formalen Kriterien zu genügen. Es geht ja dabei nicht darum, andere Forscher etc. davon zu überzeugen, dass die Theorie korrekt ist (wie in der Phase der Theorieprüfung). Entscheidend ist lediglich, dass man selbst ein Gefühl dafür bekommt, ob die eingeschlagene Richtung in etwa stimmt. Das bedeutet allerdings nicht, dass man nicht auch hier selbstkritisch vorgehen soll und sich immer wieder fragen muss, wie stark man die Versuchspersonen beeinflusst, ob die gewählten Vergleichskriterien wirklich einen aussagekräftigen Vergleich ermöglichen, etc. Tut man das nicht, besteht die Gefahr, dass das endgültige Modell trotz allen Anstrengungen wenig Bezug zur Empirie hat.

Damit solch selbstkritische Reflexionen möglich sind, lohnt es sich u.a., die verschiedenen Stadien, über die sich die Theorie bzw. das Modell entwickelt, gut zu dokumentieren. Eine solche Dokumentation erleichtert den Vergleich zwischen den ursprünglichen Vorstellungen und späteren Fassungen und hilft unter Umständen Veränderungen zu entdecken, die theoretisch nicht begründbar sind, sondern nur aus einer Anpassung an das programmtechnisch Mögliche resultieren. (Diesen Hinweis verdanken wir Hilde Haider.)

Je nachdem, wie weit der Aufbau des Programms gediehen ist, können empirische Untersuchungen einen unterschiedlichen Beitrag leisten. Existieren erst sehr vage Ideen, wie überhaupt Prozesse formuliert werden könnten, die die zu behandelnde Aufgabe lösen, erhält die Beobachtung menschlichen Verhaltens vor allem die Bedeutung einer heuristischen Fundgrube. Indem man selbst die Aufgabe löst, oder andere Personen dabei beobachtet, ergeben sich Hinweise darauf, wie ein Programm sie lösen könnte. Ist dann der Aufbau des Programms soweit fortgeschritten, dass es zumindest vereinfachte Varianten der Aufgabe lösen kann, dient der Vergleich mit menschlichem Verhalten zur Verfeinerung des Programms und hilft, aus einem primär einmal lauffähigen Prozess ein psychologisches Modell zu machen.

Als Konstrukteur psychologischer Modelle verfügt man in dieser Phase über mindestens drei verschiedene Quellen empirischer Daten: (1) Die eigene Introspektion, (2) die Beobachtung und Befragung möglichst motivierter Versuchspersonen und (3) die bereits vorhandene Literatur.

6.1.1 Introspektion

Die eigene Introspektion sollte als nützliches Hilfsmittel nicht unterschätzt werden, denn schliesslich ist eine psychologische Theorie nur dann glaubwürdig, wenn der Autor sie auch auf sich selbst anwenden kann (Herzog, 1984). Darüber hinaus ist die Introspektion aber vor

allem bei kognitionspsychologischen Modellen die zugänglichste und auch die einzige jederzeit verfügbare Quelle.

6.1.2 Beobachtung und Befragung von Versuchspersonen

Allerdings ist nicht auszuschliessen, dass sich mit der Zeit die eigenen psychischen Prozesse - oder zumindest die Art, wie man sie wahrnimmt - immer mehr dem Programm angleichen. Eine wichtige Rolle spielt deshalb auch die Beobachtung, Befragung, etc. anderer Personen. Auch hier können relativ informelle Methoden eingesetzt werden.

6.1.2.1 Hochmotivierte Langzeitversuchspersonen

Als besonders hilfreich erweist es sich, wenn zwei, drei hochmotivierte Personen zur Verfügung stehen, die bereit sind, dieselbe Aufgabe, wie sie das Programm lösen sollte, immer und immer wieder durchzuspielen. Dadurch wird es möglich, diese Personen wiederholt mit einem verfeinerten Modell ihres Vorgehens zu konfrontieren und der Reihe nach unterschiedliche Aspekte im Detail auszuloten.

Damit es allerdings gelingt, solche Versuchsmitarbeiter und -mitarbeiterinnen bei der Stange zu halten, ist entscheidend, dass die Aufgabe anspruchsvoll genug ist und auch stark genug variiert werden kann, so dass jeder Durchgang für die Teilnehmer eine neue Herausforderung darstellt. Wie schon im Kapitel 3 angesprochen, ist dies bei der Gestaltung der Spielwelt zu berücksichtigen. Darüber hinaus wird es notwendig sein, die MitarbeiterInnen genau über die Ziele der Untersuchung zu informieren, so dass sie den Versuch im selben Sinn interpretieren wie der Versuchsleiter.

6.1.2.2 Vergleich von Versuchsperson und Programm

Existiert bereits eine Version eines lauffähigen Programms, so kann man das "Verhalten" des Programms mit dem Verhalten einer Versuchsperson direkt vergleichen, indem man Programm und Versuchsperson dieselbe Aufgabe lösen lässt. So lassen sich Abweichungen feststellen, mit der Versuchsperson besprechen und eventuell auch beheben. Damit ein solcher Vergleich aber sinnvoll ist, sind folgende zwei Punkte zu beachten:

Da die Simulation kaum je perfekt das Verhalten der Versuchsperson kopieren wird, ist zu erwarten, dass bei den meisten Vergleichen die Versuchsperson zumindest an einer Stelle des Ablaufs ein anderes Vorgehen wählt als das Programm. Die beiden "Verhaltens"-Protokolle des Durchgangs werden dann von der Stelle an in der Regel nicht mehr vergleichbar sein, auch wenn sich die Versuchsperson im Weiteren "programmkonform" verhält. Der Durchgang ist also ab diesem Punkt nicht mehr auswertbar. Dies lässt sich vermeiden, wenn man dafür sorgt, dass interaktive Eingriffe in den Ablauf des Simulationsprogramms möglich sind. Kommt es dann in einem Punkt zu Divergenzen zwischen dem Vorgehen des Programms und der Versuchsperson lassen sich die beiden wieder "parallelisieren", indem man das Programm veranlasst, ("gegen seine Überzeugung") dem Vorgehen der Versuchsperson zu folgen.

Am einfachsten lassen sich solche Eingriffe realisieren, wenn die Versuchsperson ihre Aufgabe am Computer-Terminal löst, so dass das Programm gleichzeitig die Versuchssteuerung übernimmt und die Aufgabe im Takt mit der Versuchsperson bearbeitet. Die Simulation durch das Programm dient dann nur der Prognose dafür, was die Versuchsperson im nächsten Schritt tun wird. Für die effektive Ausführung übernimmt das Programm aber das Vorgehen der Versuchsperson. Dies ermöglicht es, gleichzeitig ein Protokoll all der Stellen auszudrucken, an denen das Verhalten der Versuchsperson von den durch das Programm generierten Erwartungen abgewichen ist.

Zum zweiten ist zu beachten, dass ja nicht alles, was das Programm macht, eine positive Analogie darstellen muss. In gewissen Situationen mögen mehrere Möglichkeiten bestehen, wie das Programm fortfahren könnte, ohne dass die Aspekte am Modell, die als positive Analogien gelten, diese Möglichkeiten auf eine einzige einschränken. Welche Möglichkeit das Programm wählt, ist dann also willkürlich und für die Theorie nicht von Bedeutung. Bleibt die Versuchsperson an dieser Stelle innerhalb der Menge von Möglichkeiten, aus denen das Programm

willkürlich wählen würde, wird man nicht von einer echten Abweichung sprechen. Wählt man das oben beschriebene Vorgehen der computergesteuerten, parallelen Verarbeitung der Aufgabe, stellt diese Situation kein Problem dar. Damit aber im Protokoll neutrale und negative Abweichung unterschieden werden können, ist es notwendig, nicht nur die vom Programm gewählt Variante, sondern die gesamte Menge der neutralen Alternativen darzustellen.

Die festgestellten Abweichungen können anschliessend mit der Versuchsperson im Sinne einer "critical incident" Methode besprochen werden, indem man sie befragt, warum sie sich an den einzelnen Stellen nicht so wie das Programm verhalten hat.

6.1.2.3 Was ist schwierig für das Programm und was ist schwierig für die Versuchsperson

Neben den direkt feststellbaren Abweichungen im Verhalten lässt sich auch vergleichen, welche Aspekte der Aufgaben für das Programm bzw. die Versuchsperson mehr oder weniger schwierig sind. Oft lässt sich beobachten, dass bestimmte Teilprozesse, die der Versuchsperson überhaupt keine Schwierigkeiten bereiten, programmtechnisch kaum zu lösen sind (z.B. Sprachverstehen, vgl. Kapitel 1). Dies ist selbstverständlich eine Hinweis darauf, dass die entsprechenden menschlichen psychischen Prozesse anders ablaufen als die durch die Computerarchitektur und die darauf aufbauenden Programmiersprachen festgelegten.

Genausogut kann aber auch das Umgekehrte auftreten. Manche Prozesse, die für den Menschen grosse Probleme darstellen (z.B. gewisse Formen von Gedächtnisakrobatik), lassen sich auf dem Computer geradezu trivial verwirklichen. Nimmt man solche Hinweise ernst, ergeben sich oft interessante Einsichten in die Struktur psychischer Prozesse, die dann vielleicht nicht mehr direkt ins Modell aufgenommen werden können, sicher aber ihren Platz in der zugehörigen Theorie finden.

6.1.3 Bekannte negative Analogien des Computermodells

In diesem Zusammenhang ist auf zwei Punkte hinzuweisen, in denen der Computer bekanntermassen anders funktioniert als das menschliche Gehirn, und die es deshalb als mögliche Quellen negativer Analogien im Auge zu behalten gilt. Einmal ist es möglich, Prozesse auf dem Computer so zu programmieren, dass sie (ausser durch brutale Gewalt) durch nichts mehr von aussen gestört werden können. Schreiben wir z.B. ein Programm, das eine schwierige Denksportaufgabe zu lösen hat, dann wird sich dieses Programm ausschliesslich und mit voller Konzentration genau dieser Aufgabe widmen und alles andere ignorieren. Menschen sind dazu ganz offensichtlich nicht in der Lage, denn psychische Prozesse laufen nie isoliert, sondern immer in einen ganzen Lebenskontext eingebettet ab und sind aus diesem Kontext heraus störbar.

Zum zweiten verfügt ein Computer über ein im Vergleich zum Menschen praktisch unbeschränktes Arbeitsgedächtnis, in dem sich beliebig viel Information beliebig genau beliebig lange aufbewahren lässt. Programmiert man einen Computer z.B. so, dass er unter dem Variablennamen "Telefon-Oma" eine zehnstellige Nummer abspeichert, dann wird diese Nummer unter diesem Namen im Allgemeinen unverändert beliebig lange abrufbar bleiben. Auch das ist bei menschlichen psychischen Prozessen offenbar nicht der Fall.

Man kann nun argumentieren, dass es sich dabei nicht prinzipiell um negative Analogien des Computermodells handelt, denn vermutlich würden dieselben Phänomene der Störbarkeit und Vergesslichkeit, wie sie sich beim Menschen beobachten lassen, auch bei einem tatsächlich in alltäglicher Umgebung operierenden Roboter auftreten. Denn ein solcher Roboter müsste gegenüber Störungen offen sein und seine Datenbasis würde so gross, dass vermutlich ähnliche Schwierigkeiten bei ihrer Verwaltung auftreten würden, wie sie sich beim menschlichen Gedächtnis beobachten lassen. Ob dies zutrifft, ist eine offene Frage, deren Antwort für uns hier aber irrelevant ist. Denn in einer MINCS soll ja gerade die Komplexität des Simulierten so klein gehalten werden, dass derartige Phänomene, wie sie vielleicht bei einem funktionierenden Haushaltroboter auftreten würden, sicher nicht auftreten. D.h. in einer MINCS wird man immer isolierte Prozesse darstellen, die relativ wenig Gedächtnisressourcen beanspruchen, und die

man darum - sofern man nicht absichtlich etwas dagegen unternimmt - gut als unstörbare Prozesse mit perfektem Gedächtnis verwirklichen kann. Es ist also bei einer MINCS notwendig, diese beiden möglichen negativen Analogien zu beachten.

Allerdings ist es nicht gesagt, dass es unbedingt sinnvoll ist, jede Simulation gleich von Anfang an mit einem beschränkten Arbeitsgedächtnis, etc. zu versehen. Einmal gibt es einen rein technischen Grund, warum es sinnvoll ist, ein Simulationsprogramm zuerst einmal mit einem perfekten Gedächtnis etc. zu versehen: Es ist so wesentlich leichter, reine Programmierfehler zu entdecken und auszumerzen. Ist man dann sicher, dass das Programm rein technisch richtig läuft, kann man immer noch Beschränkungen einbauen.

Zum zweiten kann es aber auch (wie schon im Punkt 6.1.2.3 erwähnt) durchaus für die psychologische Theoriebildung fruchtbar sein, wenn deutliche Unterschiede zwischen dem "Verhalten" des Programms und dem Verhalten der Versuchsperson auftreten. Solche Unterschiede müssen natürlich in der Theorie berücksichtigt werden, aber unter Umständen kann es den Rahmen einer MINCS sprengen, wollte man das Programm entsprechend anpassen. (Um einen Vergleich aus der Statistik herbeizuziehen: Das Simulationsprogramm braucht durchaus nicht immer nur als "Alternativhypothese" eingesetzt zu werden, d.h. als Hypothese, von der man zeigen möchte, dass sie zutrifft, sondern es kann auch die Rolle einer "Nullhypothese" übernehmen, d.h. der Hypothese, von der man zeigen möchte, dass sie sicher falsch ist.)

6.2 Beispiele

6.2.1 Problemlösen zu zweit

Anhand unseres zweiten Beispielprojekts lassen sich ganz unterschiedliche Aspekte von Explorationsuntersuchungen illustrieren.

Einmal ist es ein gutes Beispiel dafür, dass es sich lohnt einige Arbeit in die Konstruktion der Spielwelt zu investieren. Im Gegensatz zum Lernprojekt hatten wir die Spielwelt von Anfang an dafür ausgelegt, dass ihre Beherrschung für Versuchspersonen eine einigermassen "natürliche" Aufgabe darstellte (vgl. Kapitel 3). Deshalb hatten wir keine grossen Probleme, sie speziell für solche Untersuchungen herzurichten. Die Versuchspersonen erhielten das ihrem "Vorwissen" entsprechende Zahlenfeld auf Papier ausgedruckt und waren meist schon nach wenigen Erklärungen mit Feuereifer bei der Sache. Als besonders positiv erwies sich, dass durch Variation des Vorwissen und der zwischen den einzelnen Zellen der Zahlenfelder bestehenden Beziehungen eine Vielzahl von unterschiedlichsten Aufgaben konstruiert werden konnten, die aber alle im zentralen Punkt, nämlich der Dialogsteuerung, in etwa die gleichen Anforderungen an die Teilnehmer stellten. Dadurch war es einmal möglich, die Schwierigkeit und den zeitlichen Aufwand der Aufgaben so zu variieren, dass wir nach einigen Versuchen eine Form gefunden hatten, die von den Versuchspersonen gerade als angenehme Herausforderung erlebt wurde. Zum zweiten war es möglich, denselben Versuchspersonen eine ganz Reihe von Aufgaben gleicher oder steigender Schwierigkeit zu stellen, ohne dass dies für sie langweilig wurde. Und zum dritten war es möglich, gezielt Aufgaben zu konstruieren, die bestimmte Aspekte im Dialogablauf besonders betonten.

Als Fundgrube für neue Prozessideen verwendeten wir zwei verschiedene Untersuchungsanordnungen, bei denen uns unterschiedliche Dinge interessierten.

1) Ähnlich wie im Lernprojekt waren wir einmal an möglichen Strategien zur erfolgreichen Dialogsteuerung interessiert. Dieses Ziel verfolgten wir, indem wir Paare von Versuchspersonen eine grössere Anzahl von Aufgaben bearbeiten liessen, bis sie zu eigentlichen Experten für diese Art Kommunikationsspiel geworden waren. Dann befragten wir sie nach ihren Strategien und prüften diese mit dem Programm auf deren Wirksamkeit. Im Gegensatz zum Lernprojekt war dieser Ansatz hier ausserordentlich erfolgreich. Das dürfte einmal darauf zurückzuführen sein, dass es möglich war, die Aufgaben auf ein vernünftiges Schwierigkeitsniveau einzustellen. Die Lernaufgabe war eindeutig zu schwer gewesen. Und zum zweiten war entscheidend, dass die Teilnehmer wirklich über mehrere Spiele Zeit hatten, geeignete Strategien zu entwickeln. Denn auch sie versagten im ersten Spiel meist kläglich.

2) Wir waren auch daran interessiert, zu erfahren, was in solchen Problemlösedialogen rein sprachlich als Steuerinformation ausdrückbar sein muss. Zu diesem Zweck setzten wir Gespräche zwischen jeweils einer Versuchsperson und dem Programm als simuliertem Partner ein. In diesen Gesprächen hatte sich die Versuchsperson den stark reduzierten Interaktionsmöglichkeiten des Programms anzupassen. Dabei zeigte sich, wieweit diese aus der Sicht der Versuchspersonen zum Lösen der Aufgabe genügten, und wo sie die Kommunikation als unnatürlich eingeschränkt empfanden. Dies erlaubte es uns, die Kompetenz des Programms sukzessive zu erweitern.

Als dann das Programm eine gewisse Reife erreicht hatte, gingen wir dazu über, sein Vorgehen und das von Versuchsperson Schritt für Schritt zu Vergleichen. Wir wählten Gespräche zwischen jeweils einer Versuchsperson und dem Programm. D.h. wir liessen ein Gespräch zwischen zwei simulierten Partnern ablaufen. Der eine dieser Partner diente gleichzeitig als Gesprächspartner der Versuchsperson. Den zweiten verwendeten wir, um das Verhalten der Versuchsperson zu prognostizieren.

Wie oben besprochen, mussten wir für einen sinnvollen Vergleich für jede Situation festlegen, ob die Reaktion, die das Programm konkret zeigte, die einzige war, die aufgrund der als positive Analogien geltenden Aspekte des Modells möglich war, oder ob auch noch andere denkbar waren. Z.B. konnte das Programm die Antwort auf eine Frage aufschieben, bis es mehr Informationen gesammelt hatte. Je nach Gesprächsverlauf sammelten sich darum oft mehrere solche noch unbeantwortete Fragen an, die dann aber auch alle früher oder später beantwortet wurden. Dass sie alle beantwortet wurden, sahen wir als positive Analogie; in welcher Reihenfolge dies geschah war aber willkürlich und damit stellte die Entscheidung, jetzt genau diese und keine andere offene Frage zu beantworten, eine neutrale Analogie dar.

Nachdem dies geklärt war, organisierten wir das Simulationsprogramm so, dass es in jedem Fall zuerst die Menge aller möglichen Aktionen auf die Teilmenge einengte, die durch die positiven Analogien geben war. Nennen wir sie die neutrale Zielmenge. Und erst in einem zweiten Schritt wurde dann aus dieser neutralen Zielmenge konkret eine Aktion ausgewählt.

Nach dieser Vorarbeit stellte die eigentliche Synchronisation von Versuchsperson und Prognoseprogramm kein grösseres Problem mehr dar. Jede Äusserung des programmierten Gesprächs-Partners der Versuchsperson wurde sowohl an diese (via Bildschirm) wie an den sie simulierenden Teil des Programms weitergeleitet. Dieser lief dann bis zu dem Punkt ab, an dem die neutrale Zielmenge generiert war, und stoppte dort, bis die Versuchsperson ihrerseits reagiert hatte. Ein kleines Hilfsprogramm wandelte die Äusserung der Versuchsperson in das Ziel, diese Äusserung zu produzieren, um und das Programm lief anschliessend mit diesem Ziel weiter. D.h. die willkürliche Einengung der neutralen Zielmenge wurde durch die Übernahme des Ziels der Versuchsperson ersetzt.

War das Ziel der Versuchsperson nicht in der vom Prognoseprogramm generierten neutralen Zielmenge enthalten, wurde dies auf einem parallel zum Gespräch ausgedruckten Protokoll automatisch vermerkt. So war es möglich, das Protokoll gleich anschliessend an den Versuch mit der Versuchsperson durchzusprechen und diese zu fragen, was sie bewogen hatte, an dieser Stelle so und nicht anders zu reagieren. Daraus resultieren interessante Einsichten, die sowohl zur Verbesserung des Programms wie zu Erkenntnissen über die Grenzen menschlicher Informationsverarbeitungsfähigkeiten führten. Z.B. zeigte sich, dass die menschlichen Gesprächspartner im Gegensatz zum Programm oft ganz einfach vergassen, auf noch offene Fragen zurückzukommen. Dieser Unterschied war die Folge einer der weiter oben schon angesprochenen negativen Analogien des Computers als Modell menschlicher Informationsverarbeitung. Das Programm verfügte über ein unbegrenztes Kurzzeitgedächtnis und war deshalb in der Lage, an dieser Stelle kompetenter als die Versuchspersonen zu reagieren.

Man hätte an dieser Stelle nun das Programm ändern können. Dies entsprach aber nicht unserem Ziel, Gesprächsregeln zu entwickeln, die die Menschen kompetenter machen würden. Wir leiteten daraus vielmehr versuchsweise eine erste Gesprächsregel ab, indem wir den Versuchspersonen vorschlugen, sich offene Fragen auf einem Blatt Papier zu merken und diese dann bei Gelegenheit vollständig abzuarbeiten.

Oft stellte sich übrigens beim Beobachten der Versuchspersonen die Vermutung ein, dass diese eine bestimmte, ineffiziente Strategie verfolgten. In solchen Fällen bewährte sich, dass wir die Spielwelt so konstruiert hatten, dass sich eine grosse Zahl von Aufgaben mit ganz unterschiedlichem Charakter stellen liessen. Denn dadurch war es möglich, solchen Versuchspersonen eine Anschlussaufgabe vorzusetzen, bei der die vermutete Strategie scheitern musste. Z.B. fiel uns bei einigen Versuchspersonen auf, dass sie das Programm nie direkt nach dem Feld fragten, das zu füllen die Hauptaufgabe gewesen wäre. Es war natürlich ein leichtes, Aufgaben zu konstruieren, die nur durch genau diese Frage gelöst werden konnten. Und interessanterweise brauchten viele dieser Versuchspersonen bei einer solchen Aufgabe dann auch wirklich sehr lange, bis sie die entscheidende Frage stellten.

6.3 Zum Nachdenken

1. Nehmen wir an, wir wären daran interessiert, zu untersuchen, welche Fehler in Abhängigkeit bestimmter Morse-Zeichenkombinationen auftreten, und es würde auch schon ein Programm existieren, das diesen Prozess in einer ersten Annäherung simuliert. D.h. das Programm nimmt eine Sequenz von "Punkten" und "Strichen" entgegen und produziert mit mehr oder weniger Fehlern die entsprechende Sequenz von Zeichen. Überlegen sie sich eine Anordnung, wie dieses Programm computergesteuert parallel zu einer Versuchsperson laufen könnte.
2. Welche Fragen könnte man so klären?
3. Nehmen wir an, das Programm sei so eingerichtet, dass es bei einer bestimmten Zeichenkonstellation nicht automatisch immer einen bestimmten Fehler generiert, sondern dass nur die Wahrscheinlichkeit für diesen Fehler erhöht ist. Ob der Fehler dann tatsächlich auftritt, wird durch Zufall entschieden. Wann kann man in diesem Fall von einer Abweichung zwischen Programm und Versuchsperson sprechen?
4. Welche Informationen würden sie auf einem Protokoll ausdrucken, das sie dann mit der Versuchsperson besprechen?

7 Lauftests

7.1 Allgemeines

Irgendwann kommt der Moment, wo ein lauffähiges Programm vorliegt. Und damit kommt auch die Frage, ob das Programm eigentlich das macht, was es machen sollte (Programm-Verifikation). Dabei geht es nicht darum, zu überprüfen, ob das Programm ein psychologisch valides Modell ist, sondern ob es überhaupt ein Modell der angestrebten Theorie darstellt. Es geht also um eine relativ technische Frage.

Hauptkriterium zur Beurteilung dieser Frage wird im Allgemeinen das Resultat sein, das ein Simulationslauf produziert. Soll das Programm z.B. in der Lage sein, eine Denksportaufgabe zu lösen, dann wird man es in erster Linie sicher einmal daran messen, ob es dies wirklich kann. Daneben stellt sich aber auch die Frage, wie und warum es zu einer Lösung der Aufgabe gelangt (für eine detailliertere Frageliste vgl. Cohen & Howe, 1988).

Vielleicht überrascht die Frage nach dem "Wie" etwas, denn schliesslich hat man ja das Programm selbst geschrieben und sollte somit bestens wissen, wie es abläuft. Oft ist aber der genaue Ablauf eines Simulationsdurchgangs nicht so leicht vorhersehbar. Dies trifft v.a. dann zu, wenn man mit einer der "komfortableren" Sprachen arbeitet. Verwendet man z.B. eine regelbasierte Sprache wie OPS5 (vgl. Kapitel 5), dann hat man zwar volle Kontrolle darüber, welche Regeln man ins Programm aufnimmt. Wie diese Regeln dann aber im konkreten Fall in welchem Moment und in welcher Reihenfolge zur Anwendung gelangen, ist nicht leicht vorauszusehen. Oft erlebt man einige Überraschungen, wenn man einen Simulationslauf genauer analysiert. Es kann sein, dass das Programm auf einem ganz anderen Weg zur Lösung gelangt, als man angenommen hat. Dies gilt auch für Sprachen, bei denen der Programmablauf wesentlich mehr unter der Kontrolle des Programmierers steht.

Die Frage nach dem "Wie" kann man angehen, indem man möglichst aussagekräftige "Spuren" (traces) erstellt. Zu diesem Zweck lässt man das Programm Informationen ausdrucken, die Aufschluss über den eingeschlagenen Weg geben. Welches dabei eine gute und informative Spur ist, wird sich im einzelnen Fall jeweils erst nach einigem Experimentieren zeigen. Prinzipiell denkbar sind zwei unterschiedliche Typen von Informationen: Einmal statistische Angaben darüber, wie oft das Programm an welcher Stelle vorbeikommt, und zum zweiten eine Geschichte der Zwischenresultate, über die das Programm zum Hauptresultat gelangt ist.

Die zweite Frage, die nach dem "Warum", spricht die Randbedingungen an, unter denen das Programm läuft. Hat man sich davon überzeugt, dass das korrekte Resultat auf dem "richtigen" Weg erreicht wird, bleibt immer noch offen, ob dies aus den Gründen geschieht, die man sich vorstellt. Es kann durchaus sein, dass ein Programm eine bestimmte Denksportaufgabe nicht darum lösen kann, weil es eben in der Lage ist, generell Denksportaufgaben zu lösen, sondern, dass es nur eine ganz spezielle Eigenschaft der verwendeten Testaufgabe ist, die den Erfolg ermöglicht.

Ob dies der Fall ist, kann man prüfen, indem man die verwendete Testaufgabe systematisch auf allen Dimensionen variiert, über die man generalisieren möchte. Natürlich wird dies oft nicht in vollem Umfang möglich sein, da der dazu notwendige Aufwand sehr gross werden kann. Vorsichtshalber sollte man dann aber immer genau angeben, aufgrund welcher Erfahrungen man daran glaubt, dass das Programm das tut, was es tun sollte.

Das Programm ist also - unabhängig von jeglicher psychologischer Empirie - einem dreifachen Test unterworfen. Erstens soll es an der Oberfläche das "richtige" Verhalten zeigen. Zweitens soll es auf dem "richtigen" Weg (d.h. entsprechend der Theorie, die es modelliert) dazu kommen und drittens soll es dies aus den "richtigen" Gründen tun. Dass sich das kaum auf Anhieb realisieren lässt, sondern meist eine mehrmalige Überarbeitung des Programms verlangt, dürfte klar sein. In diesem Zusammenhang stellt sich aber die Frage, wann man mit dem Überarbeiten des Programm aufhören soll und kann. Theoretisch ist die Antwort darauf klar, nämlich

dann, wenn das Programm vollumfänglich allen drei Bedingungen genügt. Praktisch wird man aber meist vorher abbrechen.

Einerseits kann es sich nämlich zeigen, dass das Programm im Verlauf der Verfeinerungen den Rahmen einer MINCS zu sprengen beginnt und man nicht die notwendigen Ressourcen hat, um es beliebig zu verbessern; und andererseits ist das Hauptziel ja nicht unbedingt ein lauffähiges Programm, sondern die Programmierarbeit soll helfen, bessere (konsistentere und vollständigere) Theorien aufzubauen. Deshalb kann es durchaus sinnvoll sein, dass man eine als notwendig erkannte Verfeinerung der Beschreibung des untersuchten Prozesses nur noch in die Theorie aufnimmt und nicht mehr im Simulationsprogramm verwirklicht. Allerdings muss man sich dabei bewusst sein, dass für diese Modifikationen, wie plausibel sie auch sein mögen, keine Garantie besteht, dass sie als Programm lauffähig wären.

7.2 Beispiele

7.2.1 Elementare Lernprozesse

Anhand unseres Vorgehens in diesem Projekt lässt sich illustrieren, wie die Frage nach dem "Wie" des Programmablaufs untersucht werden kann. Bei der Suche nach einem geeigneten Lernmechanismus waren wir auf folgendes Problem gestossen: Unser Lerner musste ja Schritt für Schritt immer wieder entscheiden, welche Aktion er als nächstes ausführen wollte. Zu Beginn der Lernphase konnte er in vollständiger Unkenntnis der Situation gar nichts anderes tun, als zufällig eine Aktion wählen. Mit der Zeit sollte er aber erkennen, dass es nur in gewissen Situationen etwas zu lernen gibt - nämlich dann, wenn Ball und Hand gleichzeitig sichtbar sind - und sich gezielt in diese begeben. (Denn wenn er weiterhin nur nach Zufall wählte, verbrachte er als Folge der speziellen Eigenschaften unserer Spielwelt annähernd fünfzig Prozent der Zeit in der Situation, in der weder der Ball noch die Hand zu sehen war.)

Wir versuchten dieses Problem der Unterscheidung von kontrollierbaren und nicht kontrollierbaren Situationen u.a. durch folgenden "Suchprozess" zu bewältigen: Das Wissen, das der Lerner erwerben muss, lässt sich als ein Netz von Situationen repräsentieren, in deren Mitte die Zielsituation liegt. Es ist deshalb naheliegend, dieses Wissen ausgehend von der Zielsituation in konzentrischen Kreisen langsam aufzubauen. Dies kann der Lerner erreichen, wenn er sich von in bereits im Netz verankerten Situationen nach "aussen" bewegt und dann den Weg zurück sucht. Konkret verwirklichten wir diese Idee über folgende drei Regeln:

1. Wenn in einer Situation kein Anhaltspunkt über den möglichen Zielweg besteht, dann mache eine Zufallsbewegung.
2. Wenn in einer Situation eine Vermutung darüber besteht, in welcher Richtung sich das Ziel befindet, diese Vermutung aber noch schlecht bestätigt ist, dann mache eine Bewegung in der vermuteten Richtung auf das Ziel hin.
3. Wenn in einer Situation bekannt ist, welche Bewegung zum Ziel führt, dann mache eine Zufallsbewegung in eine andere Richtung.

Unsere Erwartung war, dass der Lerner sich anfänglich ziemlich zufällig bewegen würde, bis er mehrmals die Zielsituation durchlaufen hatte. Anschliessend sollte er sich vor allem in Situationen aufhalten, die einen Schritt vom Ziel weg liegen und immer wieder zum Ziel zurückkehren, bis er mit diesen Verbindungen vertraut war. Schrittweise würde er dann Ring um Ring weitere Situationen an das bestehende Netz anschliessen.

Prinzipiell erreichte dieses Programm sein Ziel, denn nach einiger Zeit hatte es tatsächlich das gesamte Netz von Situationen und verbindenden Bewegungen aufgebaut. Es benötigte dafür aber nur unwesentlich weniger lang als ein anderes Programm, bei dem die Bewegungen immer zufällig gewählt wurden. Wir wollten deshalb überprüfen, ob der "Suchprozess" tatsächlich sein Wissen so aufbaute, wie wir das erwarteten. Zu diesem Zweck teilten wir alle möglichen Situationen in Klassen mit gleichem Abstand zur Zielsituation ein und druckten in regelmässigen Abständen aus, wie oft sich der Lerner in welcher Situationsklasse befunden hatte. Tabelle 2 fasst diese Spur zusammen.

Tabelle 2: Aufenthaltshäufigkeit in verschiedenen Situationstypen in Prozent der Häufigkeit in der ersten Periode.

Situations-Typ	Periode (in 10'000 Zyklen) 0-5	5-10	10-20	20-30	30-40	40-50
Ziel	100	600	780	950	440	500
Distanz 1	100	250	340	280	120	110
Distanz 2	100	100	280	310	150	150
Distanz 3	100	60	100	250	250	250
Distanz 4	100	100	30	180	180	210
Distanz 5	100	100	42	150	150	230
nur Ball	100	100	95	50	75	66
nur Hand	100	90	80	66	80	80
weder B. noch H.	100	100	95	105	110	110

Die Häufigkeiten zeigten andeutungsweise das richtige Muster. D.h. zu Beginn waren sie etwa gleichverteilt; dann kam eine Phase, in der der Lerner sich überzufällig häufig in der Zielsituation aufhielt, etc. Allerdings waren dabei die Abweichungen von der Verteilung, die sich bei zufälliger Wahl der Bewegungen ergeben hätte, äusserst gering. Eine genauere Analyse zeigt, dass die Ursache dafür in den nicht-kontrollierbaren Situationen lag. Bewegte der Lerner sich von der Zielsituation nach "aussen", so gelangte er auf den meisten Wegen sehr schnell in eine Situation, in der er einen der beiden Gegenstände aus dem Auge verlor. Er versuchte dann zwar sofort in eine kontrollierbare Situation zurückzukommen. Da er aber die Position des unsichtbaren Objektes nicht repräsentiert hatte, gelang ihm das nur zufällig. Und so verbrachte er sogar etwas mehr Zeit in unkontrollierbaren Situationen, als der Lerner, der seine Bewegungen nur zufällig wählte. Die ausgedruckte Spur zeigte also, dass unser "Suchprozess" die Frage, wie unkontrollierbare Situationen gemieden werden können, nicht löste.

Im Lernprojekt gab es auch mehrere Punkte, an denen wir gewisse Verfeinerungen unserer Vorstellungen von den ablaufenden Prozessen nicht mehr ins Programm aufnahmen. Ein Beispiel dafür ist das folgende: Wie oben beschrieben, lernte unser Lerner die Welt beherrschen, indem er explizit ein Netz aller möglichen Situationen und der verbindenden Bewegungen aufbaute. Wir nannten dies ein extensionales Weltbild. In diesem Weltbild lassen sich einige Regelmässigkeiten entdecken. Führt der Lerner z.B. die Hand durch eine Bewegung nach links vom Ball weg, kann er sie durch eine Bewegung nach rechts wieder zurückbringen - unabhängig von der genauen Situation. Wir konstruierten deshalb in einer späteren Phase des Projekt einen Prozess, der in der Lage war, solche Regelmässigkeiten zu entdecken. Das daraus resultierende Weltbild nannten wir dann intensionales Weltbild. Aufgrund dieses intensionalen Weltbilds war der Lerner nun in der Lage, gewisse Situationen zu meistern, die er vorher nicht bewältigen konnte. Bewegte er z.B. die Hand versehentlich aus dem Blickfeld, konnte er sie nun wieder gezielt zurückbringen.

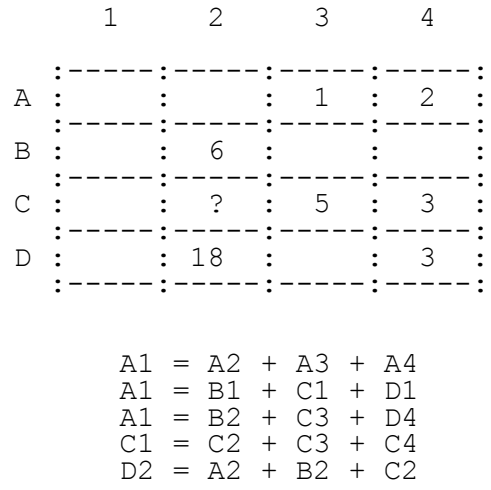
Soweit ging der Ausbau unseres Programms. Prinzipiell könnte der Lerner nun sein Netz um neue, bisher unzugängliche Situationen erweitern. Er könnte also eine neue extensionale Repräsentation aufbauen. Diese wiederum könnte Grundlage für weitere Generalisierungen bilden, also zur Erweiterung der intensionalen Repräsentation beitragen, etc. D.h. es zeichnet sich ein zyklischer Aufbau von Wissen ab, bei dem in jeder Stufe aufgrund des vorhandenen "intensionalen" Wissens neue "extensionale" Erfahrungen gesammelt werden, die dann wieder Basis für weitere "intensionale" Abstraktionen bilden, usw. (Eine gewisse Verwandtschaft mit

Piagets Akkomodations/Assimilations-Prozess ist zu erkennen; Piaget, 1974.) Diese Möglichkeit ergibt sich relativ direkt als Verallgemeinerung unseres lauffähigen Programms (das im Original etwas komplexer ist, als hier dargestellt, vgl. Kaiser & Keller, 1987a). Wir haben aber bisher nicht versucht, einen derartigen Prozess zu programmieren.

7.2.2 Problemlösen zu zweit

Die in diesem Projekt verwendeten Protokolle zum Vergleich des Verhaltens von Versuchspersonen mit dem des Programms stellen natürlich ihrerseits "Spuren" dar (vgl. 6.2.1). Die dort vorgesehene Ausgabe der "neutralen Zielmenge", d.h. der Menge aller Handlungsalternativen, die dem Programm in einer bestimmten Situation offenstehen (und unter denen es nur willkürlich wählt), kann unter Umständen eine interessante Information darstellen.

Aus diesem Projekt können wir aber vor allem ein Beispiel für die Überprüfung der Frage, "warum" ein Programm die erwünschte Leistung zeigt, herbeiziehen. Die programmierten Gesprächspartner verfügten neben der eigentlichen Dialogkomponente auch noch über eine Problemlösekomponente, damit sie überhaupt in der Lage waren, ein Zahlenquadratproblem zu lösen. Ein einfacher Teilzielbildungsprozess genügte dazu. Ausgangspunkt der Aufgabe war immer ein leeres Feld innerhalb des Zahlenquadrates. Dieses war über einige arithmetische Relationen mit anderen Feldern verbunden (vgl. Figur 3 in Kapitel 3), und wenn alle Zahlen in den Feldern, die zu einer dieser Relationen gehörten, bekannt waren, liess sich die gesuchte Zahl problemlos errechnen. Fehlten aber in einigen dieser Felder die Zahlen ebenfalls, galt es zuerst, diese Felder zu füllen (Teilziele), etc. Ein kleines Problem bestand darin, dass dieser Prozess in eine Schlaufe führen konnte. Um dies zu verhindern, führte der Problemlöseprozess eine Liste aller aktiven Ziele und Teilziele mit. Felder, die bereits auf dieser Liste waren, wurden kein zweites Mal als Teilziel zugelassen, und der Prozess behandelte die entsprechende Relation einfach, als wäre sie nicht vorhanden.



Figur 5: Beispiel eines Gesprächsgegenstandes

Dies funktionierte tadellos, solange wir mit Zahlenquadraten arbeiteten, in denen nur in horizontaler und vertikaler Richtung Zusammenhänge bestanden (Beispiel im Kapitel 3). Um unseren Versuchspersonen eine abwechslungsreichere Auswahl an Aufgaben bieten zu können, verwendeten wir aber auch Quadrate, die andere Beziehungen enthielten (wie etwa Figur 5).

Dadurch wurde das Programm vor einen neuen Typ Aufgaben gestellt. Es handelte sich also zwar nicht um einen beabsichtigten Test des Programms, die Wirkung war aber dieselbe, denn plötzlich war das Programm nicht mehr in der Lage, offensichtlich lösbare Aufgaben zu lösen. Was geschah, lässt sich anhand von Figur 5 illustrieren. Das Hauptziel ist hier C2. Nehmen wir an, das Programm bildet zuerst C1 und dann A1 als Teilziele und hält diese als offene Ziele auf der Liste fest. Expandiert es dann als zweiten Ast über A2, erreicht es früher oder später wieder A1. Da A1 aber bereits auf der Liste der offenen Ziele ist, wird dieser Ast abgebrochen

und liefert kein Resultat. Später muss der Prozess feststellen, dass der Ast über C1 ebenfalls kein Resultat liefert, da D1 fehlt. Er kommt also zum Schluss, dass das Problem unlösbar ist, obwohl über A2 und A1 sehr wohl eine Lösung bestehen würde. Als Konsequenz mussten wir den Suchprozess so modifizieren, dass er mögliche Subziele nur dann sperrte, wenn sie auf demselben Ast des Suchbaums bereits weiter oben aufgetreten waren. So erreichten wir, dass er auch mit solchen Situationen fertig wurde (Kaiser, 1987).

7.2.3 Logo Debugger

Wie in Kapitel 4 beschrieben, setzten Klahr und Carver als Wahrnehmungskomponente ihres Simulationsprogramms einen Menschen ein, d.h. ein "Gehilfe" muss auf entsprechende Fragen des Simulationsprogramms hin entscheiden, ob und wie die vom LOGO-Programm erstellte Zeichnung fehlerhaft ist. Er kann sich dabei beliebig "dumm stellen", d.h. die Fragen mit "weiss nicht" beantworten. Das Programm versucht dann das Problem genauer einzugrenzen und stellt präzisere Fragen.

Das Programm enthält zu diesem Zweck allgemeines Wissen darüber, wie man beim "debugging" am besten vorgeht. Dies sind Regeln wie "Als erstes erstelle eine Beschreibung der Abweichung des erwarteten vom tatsächlichen Resultat.", "Dann leite daraus mögliche Programmfehler ab.", "Sind mehrere Fehler möglich, verfeinere die Beschreibung der Abweichung", etc. (Klahr & Carver, 1988, p. 66). Das Programm kann auf diese Art sukzessive verfeinerte Pläne generieren und präzisere Fragen stellen.

Es besteht also eine Arbeitsteilung zwischen dem Simulationsprogramm und dem "wahrnehmenden" Gehilfen. Das Programm verfügt über allgemeines, strategisches Wissen, der Gehilfe liefert dagegen eine Art episodisches Erfahrungswissen, das ihm mehr oder weniger gut erlaubt, zu "sehen" wo der Fehler sein könnte. Interessant an diesem Vorgehen ist, dass auf diese Weise Programmierer mit viel oder wenig derartigem Erfahrungswissen simuliert werden können. Der Gehilfe braucht nur mehr oder weniger oft eine Frage mit "weiss nicht" zu beantworten.

Klahr und Carver wollten mit ihrem Programm unter anderem modellieren, dass abhängig vom diesem "Erfahrungswissen" bei der Fehlersuche unterschiedlich detaillierte Pläne generiert werden. Personen mit viel Erfahrungswissen "sehen" die möglichen Fehlerquellen mehr oder weniger direkt, Personen mit wenig Erfahrungen müssen umständlich danach suchen. Um zu überprüfen, ob sich ihr Programm tatsächlich so verhält, waren sie also an den während des Ablaufs generierten Plänen interessiert. Diese manifestieren sich im Wesentlichen dadurch, welche Ziele und Subziele in welcher Reihenfolge generiert werden. Und so illustrieren sie das "Wie" des Programmablaufs in ihrem Artikel durch die Darstellung der unter unterschiedlichen Randbedingungen generierten Zielhierarchien (Klahr & Carver, 1988). Es handelt sich hier also um eine Spur, die wesentliche Zwischenresultate bzw. -schritte auflistet.

Damit, dass Klahr und Carver die "Wahrnehmungskomponente" ihres Simulationsprogramms nicht ausmodellieren, sondern einen menschlichen Gehilfen einsetzen, ist die Gefahr verbunden, dass dieser Gehilfe unentdeckt Wissen in den Prozess einbringt, das eigentlich im Programm repräsentiert sein müsste. Oder anders gesagt: Es fragt sich, "warum" das Programm seine Aufgabe lösen kann. Ist es das Programm, das die Aufgabe löst, oder ist es der Gehilfe? Klahr und Carver testeten diese Frage durch mehr oder weniger systematische Variation des Benutzer-Inputs. (So lässt auf jeden Fall ihr Artikel vermuten, genaue Angaben fehlen.) Im Extremfall kann der Gehilfe auf alle Fragen mit "weiss nicht" antworten. Dadurch wird es prinzipiell möglich, exakt zu überprüfen, auf welche Unterstützung das Programm notwendig angewiesen ist.

7.3 Zum Nachdenken

1. Nehmen sie an, sie hätten ein Programm geschrieben, um v.a. die Kooperation parallel ablaufender Prozesse beim Decodieren der Morsezeichen zu untersuchen. Nehmen sie weiter an, sie würden drei Prozesse unterscheiden: Erkennen des Zeichenendes, Decodieren des Zeichens und Niederschreiben des Zeichens. Jeder dieser drei Prozesse benötige

je nach Zeichen unterschiedlich lange und die Resultate der einzelnen Prozesse würden in einem beschränkten Kurzzeitgedächtnis festgehalten, bis sie vom nächsten Prozess verarbeitet werden können. Überlegen sie sich eine "Spur", die es erlauben würde, dieses Ineinandergreifen zu verfolgen.

2. Nehmen sie an, sie hätten während des Aufbaus ihres Programms eine fixe Standardsequenz von Zeichen verwendet, die ihr Programm nun zufriedenstellend verarbeitet (Verarbeitungszeiten, Anzahl und Art der Fehler, etc.) Wie würden sie nun diesen Input variieren, um sicher zu gehen, dass das Programm jede "beliebige" Zeichensequenz so verarbeitet, wie sie das von ihm erwarten?
3. Überlegen sie sich Fragestellungen im Zusammenhang des Morseprojekts, bei denen eher eine "statistische" Spur (etwa wie im Beispiel unter 7.2.1.), bzw. eher eine Auflistung von Zwischenresultaten am aussagekräftigsten ist.

8 Empirische Tests

8.1 Allgemeines

Liegt schliesslich eine auf dem Simulationsprogramm basierende Theorie vor, von der man (dank ständiger, informeller empirischer Überprüfungen) überzeugt ist, dass sie das, was sie beschreiben sollte, richtig beschreibt, dann ist es selbstverständlich abschliessend notwendig, diesen Anspruch formell abzusichern. Dies geschieht prinzipiell nicht anders, als bei jeder anderen psychologischen Theorie. Wir können hier also auf die einschlägigen Publikationen zu diesem Thema verweisen (z.B. Wottawa, 1977, Bortz, 1984).

Die Tatsache, dass die Theorie zumindest partiell als Computerprogramm modelliert vorliegt, liefert einige Überprüfungsmöglichkeiten, die sonst nicht in dieser Form zugänglich sind, und auf die wir näher eingehen möchten. Am naheliegendsten ist hier einmal die bereits in Kapitel sechs besprochene Schritt für Schritt Gegenüberstellung des Verhaltens von Versuchsperson und Programm. Sofern es möglich ist, das Verhalten der Versuchspersonen sinnvoll in Schritte zu zerlegen, die im Programm eine Entsprechung haben, ergibt sich daraus eine äusserst detaillierte Bewährungsprobe für das Programm.

Ist eine derartig feine Analyse nicht möglich oder nicht erwünscht, lassen sich selbstverständlich auch die üblichen experimentellen Methoden einsetzen. Dabei bietet die lauffähige Simulation ein interessantes Hilfsmittel beim Entwurf geeigneter Experimente. Ueblicherweise steht man beim Experimentieren vor einer doppelten Aufgabe. Einerseits muss man zuerst eine geeignetes Experiment (Aufgabe, Design) finden, das überhaupt eine Aussage über den zu untersuchenden Effekt zulässt (Validitätsproblem). Z.B. müssen die verwendeten Aufgaben einen geeigneten Schwierigkeitsgrad haben. Und andererseits geht es dann selbstverständlich darum, den Effekt auch tatsächlich nachzuweisen. Mit einer Papier-und-Bleistift-Theorie steht man dabei vor dem Problem, dass man beides nur mit Hilfe von Versuchspersonen durchführen kann und somit in den üblichen Experimenten die Frage nach der Validität des Experiments und die Frage nach der Gültigkeit der Theorie hoffnungslos kontaminiert sind. Steht dagegen ein lauffähiges Simulationsprogramm zur Verfügung, lassen sich die beiden Fragen prinzipiell voneinander trennen. Denn das Programm kann dazu verwendet werden, ein geeignetes Experiment zu entwerfen, bei dem sich der Effekt zeigen muss, sofern die Theorie gültig ist.

Das Simulationsprogramm kann in diesem Zusammenhang einmal dazu verwendet werden, sehr präzise Erwartungen darüber zu formulieren, was bei Gültigkeit der Theorie in einer bestimmen Situation geschehen sollte. Dies erlaubt es, für statistische Tests auch die üblicherweise fehlende Alternativhypothese zu formulieren, so dass eine Abschätzung von Beta-Fehlern möglich wird (Cohen, 1977). Hat man z.B. ein Programm, das Denksportaufgaben löst und dabei bestimmte Fehler macht, kann man dieses die Testaufgaben lösen lassen und erhält so eine Erwartung, wie hoch der Prozentsatz von Fehlern sein sollte, die die Versuchspersonen machen, sofern das Programm ein korrektes Modell ihres Vorgehens ist.

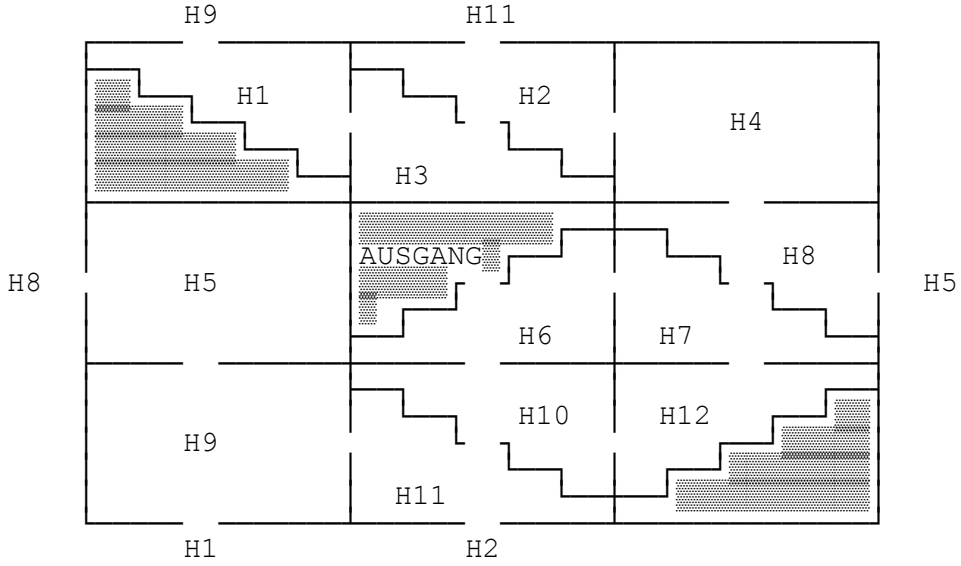
Man kann die gewählten Testaufgaben auch versuchsweise variieren, bis die Effekte, die man nachweisen möchte, besonders deutlich hervortreten. Besonders hilfreich ist dies, wenn man zu zwei konkurrenzierenden Theorien je ein lauffähiges Programm hat. Dann ist es möglich, die Aufgaben so lange anzupassen, bis eine Menge gefunden wurde, die, je nachdem welche Theorie gültig ist, zu deutlich unterschiedlichen Resultaten führen.

8.2 Beispiele

8.2.1 Elementare Lernprozesse

Aus diesem Projekt können wir ein Beispiel dafür herbeiziehen, wie es eine ablauffähige Simulation erlaubt, Aufgaben, die stark zwischen Hypothese und Alternativhypothese trennen, zu entwerfen.

Die Simulationen mit der "Ball-und-Hand"-Spielwelt gaben zur Vermutung Anlass, dass "learning by doing" nur unter ganz bestimmten Bedingungen möglich ist. Um dies zu überprüfen, entwarfen wir eine neue Spielwelt, bestehend aus einem Labyrinth von Höhlen. Die Aufgabe des Lerners bestand darin, mehrere dieser Labyrinthe zu erforschen und ihre gemeinsamen Charakteristika zu entdecken. Dies sollte es ihm ermöglichen, in Zukunft aus Labyrinthen desselben Typs schneller herauszufinden, als wenn er nur mit Hilfe von Versuch und Irrtum den Ausgang suchte.



Figur 6: Beispiel für eine "kleine Palasthöhle" (Erläuterungen im Text)

Ein Beispiel für einen ersten Entwurf eines solchen Labyrinths zeigt Figur 6. Charakteristisch für diesen Labyrinthtyp ist, dass die Höhlen einen schachbrettartigen Grundriss von drei mal drei Feldern einnehmen. Dieser Grundriss ist torusartig geschlossen, d.h. von H8 gelangt man nach "rechts" in H5 und von H9 nach "unten" in H1, etc. Gewisse Höhlen füllen ein ganzes Feld (z.B. H4). Andere teilen sich zu zweit durch eine diagonale Wand abgetrennt in ein Feld (z.B. H2 und H3). Zudem sind manche Felder oder Teilfelder leer, enthalten also keine Höhle. Der Ausgang aus dem Labyrinth führt immer in ein derartiges leeres Feld.

Ein Lerner, der von diesen Eigenschaften des Labyrinths nichts weiss, kann den Weg zum Ausgang nur durch mehr oder weniger systematisches Ausprobieren finden. Systematisch kann er vorgehen, indem er in jeder Höhle, in die er gelangt, zuerst die bisher noch nicht versuchten Durchgänge verwendet. Startet er also z.B. in H5, dann stehen ihm zwei Durchgänge (nach "links" und "unten") offen, die er noch nicht begangen hat. Er wählt z.B. zufällig den Durchgang nach "unten" und gelangt in H9. Dort findet er zwei bisher unbenutzte Durchgänge vor (nach "unten" und nach "rechts") und geht weiter nach H1, etc. Früher oder später findet er so den Ausgang.

Ein Lerner, der hingegen die typischen Merkmale dieses Höhlentyps kennt, sollte gezielter vorgehen können. In der Starthöhle H5 kann er bereits feststellen, dass zwei Ausgänge (nach "oben" und nach "rechts") fehlen. Daraus lässt sich schliessen, dass sich in diesen Richtungen leere Planquadrate befinden müssen, von denen eines den Ausgang enthalten könnte. Wenn er nun nach H9 gelangt, dann kann er über H11, etc. zu einer Umrundung des einen dieser leeren Felder ansetzen, die ihn in diesem Fall auch recht schnell zum Ausgang führt. Enthält

das erste leere Feld, das er umrundet, nicht den Ausgang, dann wird er bei der Umrundung meist schon andere gestreift haben, die er als nächstes testen kann, etc.

Wir haben geschrieben "sollte gezielter zum Ausgang finden", denn ob die Mehrinformation, über die der zweite Lerner verfügt und damit die elaboriertere Strategie, die er anwendet, tatsächlich schneller zum Ausgang führt, ist nicht ohne weiteres zu erkennen. Um sicher zu gehen, schrieben wir je ein Simulationsprogramm für die beiden beschriebenen Strategien und liessen die so simulierten Lerner in hunderten von zufällig generierten Höhlensystemen dieses Typs den Ausgang finden. Zu unserer grossen Überraschung benötigten die beiden unterschiedlichen Lerner im Durchschnitt haargenau gleich lang bis zum Ausgang, nämlich 14.2 bzw. 14.3 Schritte.

Hätten wir also diesen Höhlentyp in einem Experiment eingesetzt, dann wären unsere Versuchspersonen auch nach vielen durchlaufenen Höhlensystemen immer noch gleich schnell zum Ausgang gekommen, wie jemand, der einfach systematisch probiert. Und hätten wir dann nur auf dieses Verhaltensmass abgestellt, hätten wir schliessen müssen, dass sie nichts gelernt haben. Dieser erste Höhlentyp war also ungeeignet, um aufgrund einfach zu erhebender Daten (Länge des Wegs bis zum Ausgang) Aussagen darüber zu machen, ob die Lerner durch mehrmaliges Durchlaufen von Labyrinthen dieses Typs etwas über die zugrundeliegende Struktur lernen oder nicht.

Wir machten uns deshalb selbstverständlich daran, einen geeigneteren Höhlentyp zu konstruieren. Offenbar war im ersten Versuch die Information, die ein Lerner besass, wenn er wusste, wo ein leeres Feld ist, nicht mächtig genug, um ihm den Weg zum Ausgang deutlich zu verkürzen. Wir beschlossen deshalb, ein Höhlensystem mit einem ähnlichen Grundriss, aber mit nur einem leeren Feld, zu verwenden. Diese Änderung hatte den gewünschten Effekt, denn bei diesem Typ findet der informierte Lerner den Ausgang im Schnitt nach 7 Schritten, wogegen der uninformierte 30 Schritte benötigt.

8.2.2 Syllogismen

Das Modell von Johnson-Laird und Steedman (Johnson-Laird & Steedman, 1978, vgl. Kapitel 4) wurde explizit mit der Absicht aufgebaut, zu prognostizieren, welche richtigen und falschen Schlussfolgerungen Versuchspersonen aus bestimmten Syllogismen ziehen.

Ein Syllogismus besteht immer aus einem Prämissenpaar, wie etwa:

```
Einige Künstler sind Maler.
Einige Köche sind Maler.
-------------------------
```

Die Aufgabe der Versuchspersonen war es, zu solchen Prämissenpaaren jeweils die möglichen Schlussfolgerungen anzugeben. Prinzipiell sind dabei pro Prämissenpaar neun verschiedene Antworten möglich (wovon je nach Art der Prämissen unterschiedlich viele falsch sind). In unserem Beispiel:

1. Einige Künstler sind Köche.
2. Alle Künstler sind Köche.
3. Kein Künstler ist Koch.
4. Einige Künstler sind keine Köche.
5. Einige Köche sind Künstler.
6. Alle Köche sind Künstler.
7. Kein Koch ist Künstler.
8. Einige Köche sind nicht Künstler.
9. Die Prämissen lassen keinen Schluss zu.

Da es 64 prinzipiell verschiedene Prämissenpaare gibt, sind also 576 Kombinationen von Prämissen und Antworten möglich. Das Modell lässt erwarten, dass 213 davon überhaupt auftreten, da das Programm sie produziert, wenn es aus gegebenen Prämissen alle Schlüsse ziehen muss (132 davon sind falsche Schlüsse). Die Übereinstimmung mit dem Verhalten von

Versuchspersonen ist sehr gut. In den von Johnson-Laird und Steedman mitgeteilten Untersuchungen haben Versuchspersonen 190 dieser Schlüsse tatsächlich produziert; darüber hinaus noch acht, die das Modell nicht erwarten lässt. Das Programm (bzw. die Theorie) schneidet also in einem globalen Vergleich des Verhaltens sehr gut ab. Es produziert praktisch dieselben Resultate (inklusive Fehler), wie sie von Versuchspersonen produziert werden.

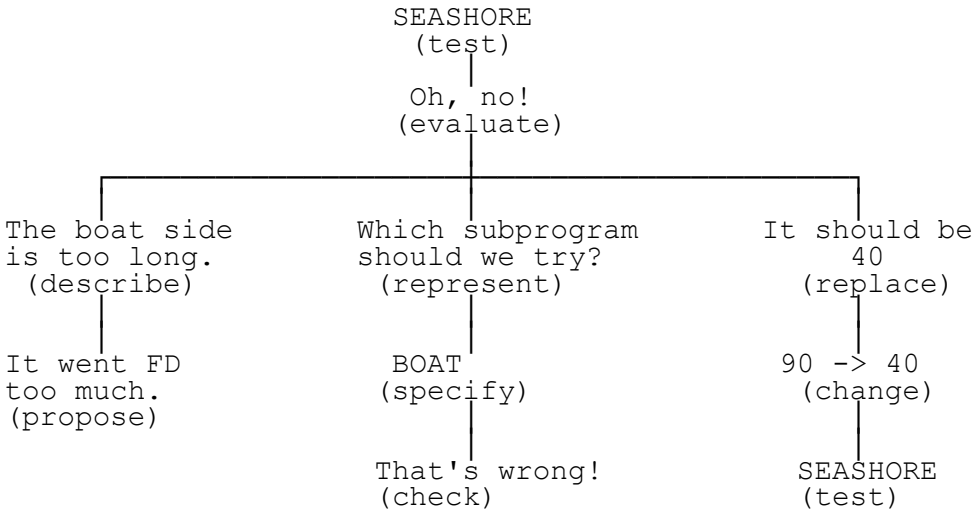
Die Autoren streben nebst diesen globalen Resultaten noch einen differenzierteren Vergleich an, indem sie Erwartungen über die relative Schwierigkeit einzelner Aufgaben formulieren. Wie bereits beschrieben (Kapitel 4), geht das Programm so vor, dass es sich zuerst ein Bild von der ersten Prämisse macht. Lautet diese z.B. "Alle A sind B", dann "stellt" sich das Programm einige A "vor" und verbindet diese mit einigen B. Durch die zweite Prämisse werden weitere "Akteure" ins Bild aufgenommen und die Schlussfolgerungen ergeben sich dann aus dem Gesamtbild. Um Aussagen über die Schwierigkeit einzelner Aufgaben machen zu können, müssen die Autoren Annahmen über die Fehleranfälligkeit einzelner Schritte in diesem Lösungsprozess machen. Sie unterscheiden im Wesentlichen drei solche Schritte: 1. Heuristischer Aufbau des "Bildes" aus den Prämissen; 2. Logischer Test des "Bildes" und allenfalls Modifikation; 3. Herauslesen einer Schlussfolgerung. Schwierigkeiten sehen sie v.a. beim zweiten und dritten Schritt. Der zweite Schritt ist denn relativ unproblematisch, wenn das ursprüngliche Bild den logischen Test besteht. Fehler treten v.a. auf, wenn eine Modifikation notwendig wird. Den dritten Schritt betrachten sie als unproblematisch, wenn die Schlussfolgerung in Richtung der im "Bild" bestehenden Verbindungen (vgl. 4.2.3) herausgelesen werden kann. Schlussfolgerungen, die sich nicht so direkt ergeben, werden entweder seltener gefunden oder der Versuch, sie herauszulösen, führt zu Fehlern.

Auch die daraus resultierenden Voraussagen decken sich gut mit den in Experimenten gewonnen Resultaten. In einem Experiment waren z.B. 80.4% der Schlussfolgerungen zu Aufgaben, in denen das Programm auf Anhieb ein korrektes "Bild" aufbaut, korrekt, im Gegensatz zu 46.5% bei Aufgaben, bei denen das Programm eine Modifikation des "Bildes" vornimmt. Entsprechende Resultate ergaben sich auch im Hinblick auf Schwierigkeiten mit dem dritten Schritt (Johnson-Laird & Steedman, 1978, pp. 83). Die Autoren basieren die empirische Evaluation also vor allem auf statistischen Vergleichen zwischen dem "Verhalten" des Programms und dem Verhalten von Versuchspersonen. Über einen Schritt für Schritt Vergleich zwischen Programm und einzelnen Versuchspersonen berichten sie im hier zitierten Artikel nicht.

Die Einführung von Annahmen zur Fehleranfälligkeit der einzelnen Schritte ist ein typisches Beispiel für einen Fall, in dem die Theorie weiter geht als das Simulationsprogramm. (Genaue Angaben fehlen allerdings dazu im Artikel; wir wollen aber um des schönen Beispiels Willen annehmen, dass dies zutrifft.) Das Programm ist so formuliert, dass es einfach aus einem gegebenen Prämissenpaar alle aus seiner Sicht möglichen Schlüsse zieht. Es beschreibt also die Kompetenz (Chomsky, 1965) von menschlichen Problemlösern, d.h. was diese tun können. Was sie dann wirklich tun (Performanz), also welche im Rahmen ihrer Kompetenz möglichen Lösungen sie dann tatsächlich produzieren oder ob sie durch irgendwelche Störungen veranlasst werden, diesen sogar zu sprengen, lässt es offen. Diese Situation ist insofern typisch, als die Tatsache, dass beim Menschen ablaufende psychische Prozesse jederzeit durch eine Unzahl von "Störungen" beeinflusst werden können, sich im geschlossen System eines Programms nur sehr schlecht abbilden lässt (vgl. 6.1.3). Im Beispiel würde es z.B. wenig Sinn machen, ins Programm einen Zufallsprozess aufzunehmen, der aus den vom Prozess generierbaren Schlussfolgerungen einige ausliest und andere unterdrückt. Das "Verhalten" das Programms würde dadurch zwar für einen Beobachter an der Oberfläche realistischer, für die Theorie wäre aber nichts gewonnen. Und so ist es oft sinnvoll, dass im Programm eine Art Gerüst oder Rahmen formuliert wird, das aufzeigt, wie der Prozess überhaupt ablaufen kann. Dies wird dann in der Theorie um Annahmen (in diesem Fall über die unterschiedliche Fehleranfälligkeit von Teilprozessen) ergänzt, so dass auf einer statistischen Ebene Vergleiche mit dem Verhalten von Versuchspersonen möglich werden.

8.2.3 Logo Debugger

Klahr & Carver (1988) setzten im Gegensatz zum vorangehenden Beispiel ihr Modell auch ein, um damit das Verhalten einzelner Versuchspersonen Schritt um Schritt abzubilden. Sie liessen ihre Schüler paarweise zusammen fehlerhafte LOGO-Programme korrigieren und baten sie dabei, möglichst oft "laut" zu denken. Anhand der so entstandenen Protokolle wurden dann für jede Korrektur-Episode (Feststellung eines Fehlers bis erneuter Testlauf des Programms) unter Verwendung der Ziele und Subziele, die auch ihr Simulationsprogramm generiert hätte, so gut als möglich ein Zielbaum rekonstruiert . Sie führen dazu ein Beispiel an: "The example begins with a test of the program SEASHORE. A negative evaluation is indicated by the comment 'Oh no!' The discrepancy was described as 'the boat side is too long,' and the bug was proposed as 'it went FD too much.' The students know that the program representation includes subprocedures, and one student asks 'Which subprogram should we try?' The other specifies the buggy subprogram as 'boat'. They edit the subprogram BOAT and scan for a FD command with a large argument. FD 90 is isolated and understood to be the incorrect move. 'It should be 40,' says one student (probably since the FD command corresponding to the other side of the boat is FD 40). The students then replace 90 with 40 and exit the editor to retest the program." (Klahr & Carver, 1988, p. 387). Figur 7 zeigt die daraus hergeleitete Zielstruktur (modifiziert nach Figur 8, Klahr & Carver, 1988, p. 387).



Figur 7: Beispiel eines Transskripts einer "debugging"-Episode

Offenbar war dies in allen Fällen ohne grosse Schwierigkeiten möglich. Aus der Fülle der so gegebenen Daten bildeten Klahr & Carver fünf verschiedene Masse (die allerdings nur zum Teil die vorhandene Information ausnutzen): (1) Der Ort im Zielbaum, an dem die Diskrepanz zwischen erwartetem und eingetretenem Resultat erstmals beschrieben wird; (2) der Ort im Zielbaum, an dem erstmals ein möglicher Fehler genannt wird; (3) der Ort im Zielbaum, an dem erstmals die mögliche Fehlerstelle im LOGO-Programm genannt wird, (4) Zeit pro Korrekturzyklus und (5) Korrekturzyklen pro Fehler. Sie wollten damit überprüfen, ob ein von ihnen durchgeführtes Training zu einer gezielteren und effizienteren Fehlerkorrektur führte, was sich auch beobachten liess: (1) Diskrepanzbeschreibung vor dem Start der Fehlersuche: 65% vor Training zu 80% nach Training, (2) Benennung des Fehlers vor dem Start der Fehlersuche: 25% zu 50%, (3) Benennung der Fehlerstelle vor dem Start der Fehlersuche: 75% zu 80%, (4) Zeit pro Zyklus: 9 Minuten zu 5 Minuten 25 Sekunden und (5) Zyklen pro Fehler: 2.9 zu 2.05. (Die ersten drei Unterschiede sind allerdings nicht signifikant, was auf die kleine Stichprobe zurückzuführen sein dürfte).

Das Ziel von Klahr & Carver ist bei dieser Analyse ein etwas anderes als das von Johnson-Laird & Steedman. Ihnen geht es nicht darum, zu beschreiben, wie ihre Versuchspersonen spontanerweise vorgehen, sondern ihr Modell beschreibt vielmehr, wie sie vorgehen müssten, wenn sie "rational" und erfolgreich ein Programm nach Fehlern durchsuchen würden. Auch ihr

Modell ist also ein "Kompetenz"-Modell, das aber nicht beschreibt, was eine Person prinzipiell kann, sondern was sie prinzipiell können muss. Entsprechend ist das Training darauf ausgerichtet den Schülern das Vorgehen, das im Simulationsprogramm implementiert ist, beizubringen. Und die Masse (1) bis (3) sind Masse dafür, wieweit das Vorgehen der Schüler in kritischen Punkten mit dem Vorgehen des Programms übereinstimmt. Masse (4) und (5) testen dann (relativ unabhängig davon), ob dadurch das Vorgehen erfolgreicher und effizienter wird.

8.2.4 Problemlösen zu zweit

Unsere Zielsetzung war in diesem Projekt dieselbe, wie sie Klahr & Carver verfolgten. D.h. wir wollten nicht modellieren, wie sich Versuchspersonen spontan in gewissen Kommunikationssituationen verhalten, sondern wir wollten untersuchen, wie sie vorgehen müssen, damit sie einen bestimmen Typ von Problem überhaupt lösen können. Und unser Ziel war es ebenfalls, ein Training zu entwickeln, das unseren Versuchspersonen dieses Vorgehen lehren würde. Während Klahr & Carver jedoch aus ihrem Modell direkt einen Lehrgang ableiteten, schalteten wir zur Konstruktion unseres Trainings einen empirischen Zwischenschritt ein. Es war nämlich anzunehmen, dass unsere Versuchspersonen spontan bereits einige Aspekte des von uns als notwendig erkannten Vorgehens zeigen würden. Diese mussten natürlich im Training nicht mehr speziell betont werden, so dass mehr Gewicht auf die Punkte gelegt werden konnte, in denen es wirklich etwas zu lernen gab.

Wir benutzten unser Modell also erst einmal als Hintergrund, vor dem wir das Vorgehen der Versuchspersonen analysierten, und zwar nicht, um das Modell zu korrigieren, sondern um im Verhalten der Versuchspersonen Abweichungen vom optimalen Vorgehen festzustellen. Einerseits verwendeten wir dazu den in Kapitel 6 beschriebenen Schritt für Schritt Vergleich, andererseits führten wir inhaltsanalytische Auswertungen der Gespräche zwischen zwei Versuchspersonen aus, ähnlich denen von Klahr & Carver (Kaiser, 1985, 1987). Daraus ergab sich deutlich, dass das Vorgehen der Versuchspersonen v.a. in zwei Punkten ungenügend war: (1) Sehr oft wurden Fragen des einen Partners und damit mögliche Lösungswege vom zweiten Partner einfach ignoriert, und (2) bei Fragen, die ein Partner nicht direkt beantworten konnte, machte kaum jemand eine Unterscheidung, ob eine Antwort prinzipiell nicht möglich war, oder ob einfach gewisse Zusatzinformationen fehlten. Beide Unterlassungen können fatale Folgen haben und entsprechend stark wurden diese Punkte im daraus abgeleiteten Training betont.

Für die Evaluation des Trainings, d.h. die Frage, ob die Versuchspersonen sich nachher an das gelehrte Vorgehen halten - und mit welchem Erfolg - konnte dann aber kein direkter Vergleich mit dem Vorgehen des Programms herangezogen werden. Dies deshalb, da der Effekt des Trainings nicht mit Gesprächen im Rahmen der Spielwelt - wo ein derartiger Vergleich möglich gewesen wäre - sondern anhand von Gesprächen über die Lösung einer mathematischen Aufgabe ("Ist die Ableitung von y=x2 tatsächlich y'=2x?") getestet wurde. In einer kleine Studie (5 Paare ohne Training gegenüber 5 Paaren mit Training) zeigte aber ein Globalvergleich, dass das Training zumindest die Häufigkeit einer erfolgreichen Lösung erhöhte (0% zu 40%).

8.3 Zum Nachdenken

1. Nehmen wir an, Sie hätten die Decodierung der Morsezeichen als Absuchen eines Entscheidungsbaums dargestellt (vgl. Kapitel 5, Aufgaben 1 und 2). Nehmen wir weiter an, sie würden die Entstehung von Fehlern dadurch modellieren, dass ihr Programm bei jedem Knoten mit einer Wahrscheinlichkeit von je 5% einen der beiden falschen Äste nimmt. Das Programm würde also ab und zu einen Punkt mit einem Strich verwechseln. Ab und zu würde es aber auch ein Zeichenende erkennen, wo keines ist, und die Suche abbrechen. Und ebenfalls besteht die Möglichkeit, dass es über das Zeichenende weitere Punkte und Striche "halluziniert". (Prinzipiell könnte es so beliebig lange Zeichen halluzinieren; nehmen sie an, dass es spätestens ein Punkt oder Strich nach dem tatsächlichen Zeichenende abbricht.) Welche Fehler sollten nach diesem Modell mit welcher Wahrscheinlichkeit beim Decodieren des Buchstaben I auftreten?

2. Nehmen wir an, sie hätten zwei Programme geschrieben. Das eine decodiere die Morsezeichen über das Absuchen eines Entscheidungsbaums (wie in Aufgabe 1) . Das zweite dagegen verwende einen eher "ganzheitlichen" Matchprozess, bei dem die wahrgenommene Folge von langen und kurzen Tönen mit Schablonen verglichen werden, die zu den einzelnen Zeichen gehören (z.B. über eine separate Regel für jedes Zeichen, vgl. Aufgabe 4, Kapitel 5). Nehmen wir weiter an, sie hätten in diesem zweiten Fall die Entstehung von Fehlern wie folgt modelliert: (1) Ein Punkt (bzw. Strich) wird fälschlicherweise mit einem Strich (bzw. Punkt) zur Übereinstimmung gebracht (5% Wahrscheinlichkeit); (2) ein Punkt oder Strich wird ausgelassen (2.5%); und (3) zwei aufeinanderfolgende Punkte, bzw. Striche in der Schablone werden fälschlicherweise mit demselben Punkt (bzw. Strich) in der Vorlage zur Deckung gebracht (2.5%). Zum Beispiel:

Vorlage	. - . -	
Fehler 1	- - . .	Verwechslung von Punkt und Strich
Fehler 2	. - -	Auslassung
Fehler 3	. . - . -	Doppelbindung

Welche Zeichen führen zu deutlich unterschiedlichen Fehlermustern, je nachdem durch welchen Prozess sie bearbeitet werden?

9 Schlussbetrachtungen

Nachdem wir nun mit einiger Ausführlichkeit dargestellt haben, wie wir uns den Einsatz kleiner Computersimulationen im Rahmen der psychologischen Forschung vorstellen, scheint es uns sinnvoll, nochmals auf die Frage zurückzukommen, was sich durch solche MINCS erreichen lässt. Welche Position nimmt eine MINCS innerhalb des gesamten Forschungsprozesses ein? Welche Art von Psychologie lässt sich so betreiben? Wir werden im Folgenden versuchen, eine Einordnung anhand verschiedenster Dimensionen vorzunehmen, die sich alle grob durch ein Gegensatzpaar ordnen lassen.

9.1 Theorie versus Modell

Man kann das Simulations-Programm selbst als eine Theorie betrachten oder als Modell zu einer Theorie. (Mit "Simulations-Programm" ist selbstverständlich nicht die physikalische Realisation des Programms gemeint; genauso wenig wie die Druckerschwärze gemeint ist, wenn man von einer "Theorie" spricht.) Dass wir es als Modell betrachten, haben wir bereits verschiedentlich gesagt und es dürfte auch implizit an mehreren Stellen klar geworden sein (v.a. bei der Behandlung der "neutralen Analogien" im Kapitel 4 und bei der Frage nach Abbruchkriterien im Kapitel 7). Oft ist auch problemlos zu erkennen, dass das Programm tatsächlich als Modell gebraucht wird. Nämlich dann, wenn in Begriffen über das Programm gesprochen wird, die in unterschiedlichster Art und Weise im Programm realisiert sein können, die aber nur in dieser abstrakteren Form eine psychologische Bedeutung haben. Wenn wir z.B. im Kapitel 7 davon gesprochen haben, dass sich der simulierte Lerner eine extensionale Repräsentation aufbaut, dann ist diese Beschreibung ein Abstraktionsniveau höher als die Prozessbeschreibung, die das Programm liefert.

Aber auch in Fällen, bei denen Theorie und Programm wesentlich näher beisammen liegen, lassen sich meist leicht Unterschiede zwischen den beiden finden. Wenn wir z.B. in der Theorie aussagen, dass der Lerner Häufigkeiten akkumuliert und daraus Schätzungen für Wahrscheinlichkeiten ableitet, dann beschreiben wir den Prozess in einer natürlichen Sprache beinahe auf demselben Abstraktionsniveau, wie er auch durch das Programm in der Programmiersprache beschrieben wird. Es handelt sich aber eben auch hier nur beinahe um dasselbe Niveau, denn z.B. wird durch die Programmiersprache festgelegt, dass die kumulierten Häufigkeiten je unter einem bestimmten Variablennamen abgespeichert werden und nur zugänglich sind, wenn dieser Name bekannt ist - was bereits wieder unter dem Niveau sein dürfte, auf dem wir psychologisch bedeutungsvolle Aussagen machen wollen. Praktisch lassen sich in jeder der verbreiteten Programmiersprache Details finden, die psychologisch gesehen keine positiven Analogien darstellen. Und praktisch in jedem Simulationsprogramm müssen Prozesse eingebaut werden, die nichts mit den psychischen Prozessen zu tun haben, über die man Aussagen machen möchte (vgl. Kapitel 4).

Dass dies nicht zwangsläufig so sein muss, behaupten z.B. Newell und Simon ("The program is the theory."). Sie gehen dabei von der "psychologischen Annahme" aus (vgl. Kapitel 1), d.h. sie nehmen an, dass auf dem richtigen Abstraktionsniveau betrachtet dieselben informationsverarbeitenden Prozesse im menschlichen Hirn und in einem Computer ablaufen. Eine Programmiersprache, die genau dieses Niveau trifft, ist dann natürlich zwangsläufig auch eine psychologische "Theoriesprache". Dass wir diese Annahme nicht so ohne weiteres teilen, wurde schon gesagt (Kapitel 1). Aber auch wenn man sich auf die etwas kritischere "erkenntnistheoretische Annahme" zurückzieht, bleibt die Möglichkeit offen, dass sich eine Programmiersprache schaffen lässt, die in jedem Aspekt psychische Prozesse beschreibt. Programme, die in dieser Sprache geschrieben wären, wären dann natürlich gleichzeitig psychologische Theorien.

Wir können diese Möglichkeit hier nicht ausschliessen. Dass wir trotzdem davon ausgehen, dass eine MINCS immer nur ein Modell einer Theorie ist, hat deshalb pragmatische Gründe. Einmal ist uns bisher noch keine derartige Programmier-/Theorie-Sprache begegnet. Und zum

zweiten - und dieses Argument ist hier entscheidend - gehört sicher keine der Sprachen, die zur Zeit im Rahmen einer MINCS einsetzbar ist (also auch keine der in Kapitel 5 erwähnten), dazu.

Man kann - und wir selbst haben das an verschiedenen Stellen getan (Kaiser, 1980, 1984) - die Theorie ihrerseits als Modell (eines Wirklichkeitsausschnitts) betrachten (diesmal im ersten Sinn der beiden im Kapitel 4 erwähnten). Ein Simulationsprogramm zu dieser Theorie wäre dann das Modell eines Modells. Dies mag eigenartig erscheinen, ist aber gar nicht so ungewöhnlich. Wenn z.B. jemand auf dem Schaltpult zu einer Modelleisenbahnanlage einen Plan dieser Anlage aufzeichnet, dann macht er sich ein Modell eines Modells mit dem Ziel, das erste Modell besser handhaben zu können. Und genau dieses Ziel verfolgen wir hier auch mit der Erstellung von Simulationsprogrammen.

9.2 Theoriebildung versus Theorieprüfung

Im ersten Kapitel haben wir definiert, dass wir Computersimulation als Instrument zur Entwicklung konsistenter und logisch vollständiger Theorien verstehen wollen. Damit ist bereits einmal gesagt, in welcher Phase des Forschungsprozesses wir die MINCS ansiedeln würden. Teilt man diesen nämlich in Theoriebildung und Theorieprüfung auf, dann gehört die MINCS ganz klar zur Theoriebildung. Ein lauffähiges Programm, ganz gleich welchen Umfangs, hat für uns keinerlei empirische Aussagekraft für die damit verbundene psychologische Theorie. (Vgl. Kapitel 1 für die Frage, welche Annahmen erfüllt sein müssten, damit dies anders wäre.)

Wir hoffen übrigens, dass dadurch, dass wir beinahe ein ganzes Buch nur über Theoriebildung geschrieben haben, auch klar geworden ist, dass uns diese Phase im Forschungsprozess wesentlich bedeutsamer erscheint, als sie üblicherweise in der psychologischen Forschung behandelt wird (vgl. auch Herzog, 1984). Eigenartigerweise wird nämlich in den meisten wissenschaftstheoretischen Diskussionen, mit deren Hilfe man aus der Psychologie eine respektable empirische Wissenschaft machen möchte, immer wieder übersehen, dass z.B. die Erfolge der Physik nicht so sehr auf ihre empirischen Methoden, als vielmehr auf die sehr gründliche Theoriebildungsarbeit zurückzuführen sind.

9.3 Theoretische Psychologie versus empirische Psychologie

Das führt uns zu einem weiteren Einteilungskriterium. Bekanntlich wird in der Physik zwischen experimenteller und theoretischer Physik unterschieden. Und genauso könnte man sich eine theoretische Psychologie vorstellen, deren Aufgabe es ist, den Raum der logischen Möglichkeiten zu kartographieren und dabei möglichst viele empirische Bruchstücke zu integrieren.

Bemüht man sich um eine genügende empirische Absicherung der MINCS (vgl. Kapitel 6 und 8), dann ist jede MINCS natürlich auch ein Stück empirische Psychologie. Da wir aber eine MINCS als Instrument sehen, um zu logisch vollständigeren und widerspruchs-freieren Theorien zu gelangen, stellt jede MINCS auch ein Stück theoretische Psychologie dar. Dieser Aspekt kann - wie zum Beispiel in unserem Lernprojekt - dominant werden.

9.4 Kompetenz versus Performanz

Die Unterscheidung in Kompetenz und Performanz wurde v.a. stark von Chomsky geprägt (Chomsky, 1965, vgl. auch Kapitel 8.2.1). Die Frage nach der "Kompetenz" ist die Frage danach, was ein bestimmtes informationsverarbeitendes System überhaupt können muss, damit es eine bestimmte Leistung erbringen kann. Unter "Performanz" wird dann verstanden, wie es diese Leistung im konkreten Fall erbringt. Zum Beispiel muss nach Chomsky jemand, der eine Sprache wie Englisch spricht, prinzipiell in der Lage sein, gewisse komplexe grammatikalische Analysen vorzunehmen, damit er jeden denkbaren Satz verarbeiten kann (Kompetenz). Ob er aber angesichts relativ einfacher Sätze immer diese vollständige Analyse durchführt, ist eine andere Frage (Performanz).

Eine Computersimulation kann selbstverständlich ein Kompetenz- oder ein Performanz-Modell sein. Boden klassifiziert Kompetenzmodelle als "computational psychology" im mathematischen Sinn. Für Performanz-Modell verwendet sie den Begriff der "computational psychology" im formalen Sinn (Boden, 1988, p. 226). Oft ist es so, dass eine MINCS gleichzeitig oder nacheinander beide Rollen erfüllt. In mehreren unserer Beispiele stand am Anfang die Frage nach einem Kompetenz-Modell. Beim Lernprojekt war dies das eigentliche Ziel, d.h. wir wollten untersuchen, welche Lernprozesse mindestens notwendig sind, damit eine bestimmte Lernleistung erbracht werden kann. Beim Problemlöseprojekt diente ein Kompetenzmodell als Hintergrund, auf dem sich das Verhalten und die Schwächen im Verhalten der Versuchspersonen besser beobachten liessen. Ähnlich gingen Klahr & Carver bei ihrem LOGO-Debugger und Young & O'Shea bei der Analyse von Subtraktionsfehlern vor. Ein echtes "reines" Performanz-Modell ist unter unseren Beispielen nur die Simulation zum syllogistischen Schliessen von Johnson-Laird & Steedman. Dies dürfte kein Zufall sein. Denn beim Aufbau eines neuen Simulationsprogramms steht man ja vor der Aufgabe, dass man zuerst einmal einen Prozess finden muss, der die gewünschte Leistung erbringt, und dies führt meist recht direkt zur Frage, welche Prozesse denn minimal notwendig sind. Prinzipiell ist es dann natürlich möglich, dieses Kompetenz-Modell zu einem Performanz-Modell zu erweitern. Oft ist es aber naheliegender, es bei der Simulation des Kompetenz-Modells bewenden zu lassen und die Performanz von Versuchspersonen im Kontrast zu diesem Modell zu beobachten.

Steht vor allem der Kompetenz-Aspekt im Vordergrund, dann ist beim Austesten des Programms besonders wichtig, ob und aus welchen Gründen es die erwünschte Leistung erbringt (Kapitel 7). Bei einem Performanz-Modell ist dagegen eher entscheidend, dass man den Kontakt zur Empirie nicht verliert (Kapitel 6).

9.5 Deskriptive versus normative Modelle

Die von uns verwendeten Beispiele legen eine weitere Einteilung nahe. Sowohl Young & O'Shea mit ihrer Analyse der Subtraktionsfehler wie Johnson-Laird & Steedman mit ihrem Modell zum syllogistischen Schliessen geht es vor allem um die Beschreibung möglicher und tatsächlicher Prozesse. Anders dagegen sind die Modelle im Problemlöseprojekt und beim LOGO-Debugger von Klahr & Carver als normative Vorgaben gedacht, wie sich Versuchspersonen verhalten müssten, damit sie die gestellte Aufgabe bewältigen können. In diesem zweiten Fall wird also nicht versucht, das Modell dem tatsächlichen Verhalten von Versuchspersonen anzupassen, sondern umgekehrt wird das Verhalten der Versuchspersonen über ein Training dem Modell angeglichen.

Diese Unterscheidung von deskriptiven und normativen Modellen ist nicht vollständig parallel zur Unterscheidung von Kompetenz- und Performanz-Modellen. Selbstverständlich geht ein normatives Modell zuerst einmal von der Kompetenz-Frage aus, d.h. von der Frage, wie man sich überhaupt verhalten muss, damit die Aufgabe lösbar ist. Es kann aber nicht dabei stehen bleiben, denn damit sich daraus nachvollziehbare Verhaltensregeln ableiten lassen, müssen auch Performanz-Aspekte berücksichtigt werden. Umgekehrt kann ein Kompetenz-Modell rein deskriptiv verstanden sein, das beschreibt, welche Prozesse garantiert bei einer Person ablaufen müssen, von der man weiss, dass sie eine bestimmte Aufgabe bewältigen kann.

9.6 Null-Hypothesen versus Alternativ-Hypothesen

Wie unter 9.4 bereits erwähnt, ist es natürlich möglich, dass man darauf verzichtet, ein Modell dem Verhalten von Versuchspersonen anzupassen, und es vielmehr als Hintergrund benutzt, vor dem sich das beobachtete Verhalten kontrastierend abhebt.

Von dieser Möglichkeit haben wir v.a. im Problemlöseprojekt Gebrauch gemacht. Dort diente ein (normatives) Kompetenz-Modell als Hintergrund, um zu beobachten, in welchen Punkten Dialogpartner bei der Steuerung des Gesprächsablaufs Fehler machen. Denn so wurde es möglich zu entscheiden, wo ein entsprechendes Training einzusetzen hat.

Aus der Statistik ist die Unterscheidung in Null-Hypothese und in Alternativ-Hypothese bekannt. Bei der Null-Hypothese handelt es sich um eine Annahme, von der man zeigen möchte, dass sie sicher nicht zutrifft; die Alternativ-Hypothese dagegen umschreibt eine Annahme, von der man erwartet, dass sie die Daten korrekt beschreibt. Ein Simulationsmodell kann beide Rollen übernehmen. Wie in der Statistik ist es aber auch hier oft einfacher, eine Null-Hypothese zu formulieren, die oft mit wenigen, groben Kennzeichnungen auskommt, als ein Modell auszuarbeiten, dass die Daten wirklich im Detail abbildet. Von daher eignet sich eine MINCS tendenziell besser zur Formulierung von Null-Hypothesen (vgl. Kapitel 6).

9.7 Symbolverarbeitende versus "subsymbolische" Prozesse

Im ersten Kapitel hatten wir erwähnt, dass sich v.a. im Bereich der Simulation von Lernprozessen zur Zeit zwei "Schulen" gegenüberstehen: "Symbol-Verarbeitung" und "Konnektionismus". Der Leserin wird vielleicht aufgefallen sein, dass es sich bei allen von uns verwendeten Beispielen um "symbolverarbeitende" Prozesse handelt. Dies ist v.a. einmal auf unsere Erfahrungen und Vorlieben zurückzuführen. Erstens haben wir wenig praktische Erfahrungen mit "konnektionistischen" Modellen und zweitens glauben wir, dass sich diese vor allem für eine Detailanalyse von Prozessen im Bereich der Mustererkennung (Wahrnehmung, Gedächtnisabruf, "pattern matching" als Teil eines Produktionssystems, etc.) eignen, nicht aber für die Darstellung zeitlich ausgedehnter und dem Bewusstsein zugänglicher Prozesse, denen unser Interesse vor allem gilt.

Prinzipiell ist es natürlich möglich, auch "konnektionistische" Modelle im Rahmen einer MINCS zu behandeln. Es wird unterdessen auch eine Vielzahl von Programmen angeboten, mit denen sich neuronale Netze relativ schnell und mühelos aufbauen lassen. (Die meisten davon sind allerdings mehr als didaktische Spielzeuge denn als echte Werkzeuge gedacht und können nur Netze von sehr beschränktem Umfang effizient bewältigen.) Nach unseren Erfahrungen ist in diesem Bereich aber die Gefahr wesentlich grösser, dass der Rahmen einer MINCS relativ rasch gesprengt wird. Zumindest die Anforderungen an eine leistungsfähige Hardware sind deutlich höher, als bei "symbolverarbeitenden" Modellen.

Tabelle 3 fasst nochmals zusammen, wo wir die Stärken einer MINCS sehen. Je mehr Sternchen der eine Pol eines Gegensatzpaares erhält, umso eindeutiger würden wir den Einsatzbereich einer MINCS jener Seite zuordnen.

Tabelle 3 : Einsatzmöglichkeiten für eine MINCS

Modell	****	Theorie
Theoriebildung	****	Theorieprüfung
Kompetenz	***	* Performanz
Null-Hypothese	***	* Alternativ-Hypothese
symbolverarbeitend	***	* subsymbolisch
Theoretische Psych.	**	** Empirische Psych.
normativ	**	** deskriptiv

10 Anhang: Lösungen

3.1 Das Problem stellt sich ganz ähnlich wie beim "LOGO Debugger" (vgl. Kapitel 4.2.2). Betrachtet man nur den Symbolcharakter von Morsezeichen, dann benötigen wir tatsächlich keine Spielwelt, denn die "Welt" der Morsezeichen ist äusserst einfach. Aber selbstverständlich bedeutet die Entscheidung, die Morsezeichen nur als Symbole zu sehen, bereits eine gewaltige Abstraktion. Denn in erster Linie sind sie natürlich einmal physikalische Erscheinungen, kurze und lange Pieptöne mit dazwischenliegenden kurzen und langen Pausen. Und es ist zu erwarten, dass v.a. die zeitliche Dimension dieser Erscheinungen beim Abnehmen von Morsezeichen eine gewaltige Rolle spielt. Die Frage ist also, mit welchem Ziel man an die Untersuchung herangeht. Will man einfach untersuchen, wie ein einzelnes Zeichen in den entsprechenden Buchstaben übertragen wird, dann genügt die symbolische Ebene und es erübrigt sich, eine eigentliche Spielwelt zu konstruieren (bzw. es liegt bereits eine geeignete Abstraktion vor, die direkt eine "Spielwelt" ergibt). Ist man dagegen daran interessiert, wie mehrere Zeichen hintereinander oder auch parallel überlappend wahrgenommen werden, wird man versuchen müssen, den physischen Aspekt der Morsezeichen irgendwie einzufangen. Dabei gibt es (wieder je nach Fragestellung) sicher Aspekte, die man weglassen kann (z.B. Tonhöhe, Hintergrundrauschen) und andere - wie eben die zeitliche Dimension - die man berücksichtigen muss.

3.2 Sicherlich würde dadurch die bereits angesprochene zeitliche Dimension vernachlässigt. Charakteristisch am Morsen ist ja, dass der Abnehmende den Fluss der Zeichen nicht kontrollieren kann und sich also nicht das nächste Zeichen erst vornehmen kann, wenn er das letzte verarbeitet hat. Will man diese "Fremdsteuerung" des Morseempfängers simulieren, muss man also dafür sorgen, dass die "Spielwelt" sich dem Wahrnehmenden in ihrem Rhythmus aufdrängen kann. Technisch könnte man das z.B. so verwirklichen, dass man für die Simulation ein Produktionssystem (vgl. Kapitel 5) verwendet, so dass der Programmteil, der die "Spielwelt" simuliert in fixen Abständen (z.B. immer nach einer bestimmten Anzahl Zyklen) neue Daten ins Arbeitsgedächtnis einbringt. Wie viel Platz dort vorhanden ist, was bei einer Überfüllung geschieht und wie sich mehrere gleichzeitig vorhandene Daten gegenseitig beeinflussen, ist dann im eigentlichen Simulationsprogramm zu entscheiden.

4.1 Denkbare Zielsetzungen wären:

a) Untersuchung des Lernprozesses: Wie wird Morsen gelernt? Warum dauert es so lange, bis die maximale Geschwindigkeit erreicht ist? Wie könnte man diese Lernzeit durch ein optimal auf die Eigenarten des Morsens abgestimmten Lehrgang verkürzen?
b) Untersuchung des Verarbeitungsprozesses: Wie werden die Morsezeichen verarbeitet? Welche Fehler treten dabei auf? Aus welchen Gründen? Könnte man den Morsecode verbessern, d.h. besser den Eigenarten des Verarbeitungsprozesses anpassen, so dass weniger Fehler gemacht werden? Gibt es Verarbeitungsstrategien, die zu einer verbesserten Leistung führen?
c) Untersuchung der Koordination von Verarbeitung und Wiedergabe: Wie ist es möglich, gleichzeitig das letzte Zeichen niederzuschreiben und das nächste zu verarbeiten? Welche Störungen treten dabei auf? Ergibt sich daraus eine theoretische oberste Geschwindigkeitslimite?

4.2

a) Es ist kaum anzunehmen, dass beim Morsen-Lernen die eigentliche Schwierigkeit darin besteht, sich zu merken, welches Muster von langen und kurzen Tönen welchem Buchstaben entspricht. Vielmehr ist zu erwarten, dass die "Automatisierung" (was auch immer das sein mag) des Gemerkten das eigentliche Problem darstellt. Wie diese Automatisierung vorsichgeht, hängt vermutlich vom Verarbeitungsprozess ab, wie er beim geübten Morser abläuft. Wie dieser Verarbeitungsprozess im Modell dargestellt ist, sollte also vermutlich eine positive Analogie darstellen. Weniger entscheidend dürfte sein, in welcher Form die Zuordnungsregeln im Gedächtnis gespeichert sind, bevor sie automatisiert werden. Und

ebenfalls relativ uninteressant dürfte sein, wie es zu dieser ersten Speicherung kommt, d.h. wie die Zuordnungsregeln gemerkt werden. Diese beiden Punkte können also vermutlich als neutrale Analogien betrachtet werden.

b) Da beim Morsen eine bestimmte Kombination kurzer und langer Töne nicht eindeutig ist, sondern die gleiche Kombination (z.B. ". - ") ein abgeschlossenes Zeichen ("a") oder der Beginn mehrerer anderer Zeichen (unter anderem "r" = ". - ." oder "w" = ". - -") bedeuten kann, besteht das Problem einerseits darin, zu erkennen, dass ein Zeichen abgeschlossen wurde (signalisiert durch eine Pause von gleicher Länge wie ein "-"), und dann zweitens das richtige Zeichen zuzuordnen. Dies gilt v.a. wenn verschlüsselter Text gesendet wird, so dass der Empfänger keine Erwartungen aufbauen kann, welches Zeichen als nächstes folgt. Nehmen wir einmal an, dass man die beiden Prozesse getrennt behandeln kann und betrachten wir nur den zweiten. Solche Erkennungsprozesse - v.a. wenn sie so gut eingeübt sind, wie bei geübten Morser - laufen in der Grössenordnung von Zehntelsekunden ab. Bei einer Übertragungsrate von 18 Wörtern pro Minute (WpM), wie sie bei geübten Morsern üblich ist, dauert die Übermittlung eines Zeichens im Durchschnitt 0.6 Sekunden (Pfisterer, 1988). Wir wollen deshalb einmal annehmen, dass sich die Verarbeitung der Zeichen nicht gross überlappt, und wir das Erkennen des einzelnen Zeichens isoliert betrachten können. Das bedeutet, dass die ganze zeitliche Dimension, soweit sie im Modell überhaupt Eingang findet, als neutrale Analogie betrachtet werden kann.

c) Es ist anzunehmen, dass Koordinationsprobleme zwischen den parallel ablaufenden Prozessen v.a. dadurch entstehen, dass diese dieselben Resourcen beanspruchen. In einer ersten Annäherung könnte man deshalb die genau Ausgestaltung der einzelnen Prozesse - soweit dadurch keine Aussagen über die benötigten Resourcen gemacht werden - als neutrale Analogien betrachten. Eine positive Analogie stellt dann aber sicher dar, wie man die Resourcen (Kurzzeitgedächtnis, etc.) und ihre Belegung durch die Prozesse formuliert.

4.3 Für unsere drei Beispiele ist v.a. der Vergleich der Varianten b) und c) interessant. Extrem formuliert sind in b) Prozesse positive und zeitliche Abläufe neutrale Analogien, wogegen die Verhältnisse in c) genau umgekehrt liegen.

4.4 Nehmen wir einmal an, wir würden in der Variante c) mit drei Prozessen arbeiten. Ein erster Prozess entdeckt Zeichenenden, ein zweiter decodiert die Zeichen in irgend eine mentale Repräsentation und ein dritter sorgt dafür, dass das Zeichen anschliessend auf ein Blatt Papier geschrieben wird. Der zweite und der dritte Prozess können als reine Übertragung des Zeichens aus einer Repräsentation in eine andere gesehen werden. Dies lässt sich am radikalsten vereinfachen, indem man alle drei Repräsentationen im Programm gleich darstellt, z.B. als String bestehend aus dem entsprechenden Zeichen ("a", ":", etc.). Die Übertragungsprozesse haben dann gar nichts zu tun, ausser Zeit und sonstige Resourcen zu verbrauchen. Sie nehmen den String und geben ihn in derselben Form wieder zurück. Da dies aber nicht sehr anschaulich ist und es unter Umständen erschwert, das Funktionieren des Programms zu kontrollieren, könnte man einen kleinen Schritt weiter gehen und drei verschiedene Repräsentationen wählen, z.B. drei verschiedene Strings ("- . . .", "b" und "bbb" für das Zeichen "B"). Die Zuordnung lässt sich leicht als Tabelle verwirklichen, so dass das Programm zum Zweck der "Übersetzung" einfach diese Tabelle konsultiert und die entsprechende Zuordnung vornimmt. Wie das technisch genau zu realisieren ist, hängt von der verwendeten Programmiersprache ab.

4.5 In diesem Übertragungsprozess ist sicher die Art, wie die Übertragung erfolgt (Konsultation einer Tabelle) als neutrale Analogie zu betrachten. Nicht neutral ist hingegen, wie sich diese Prozesse ins gesamte Modell einbetten. Einmal muss sicher als positive Analogie gelten, dass es sie überhaupt gibt, dass wir zwei verschiedene (Decodierung und Schreiben) unterschieden haben und wie viel Zeit die jeweilige Übertragung beansprucht. Diese Zeit (die nichts mit der Zeit zu tun hat, die das Programm benötigt, um die Übertragung tatsächlich auszuführen, sondern die explizit im Modell dargestellt werden muss) können wir für die beiden Prozesse je frei wählen. Allerdings werden diese Zeiten für alle Zeichen dieselben sein, da der Prozess für jedes Zeichen identisch abläuft. Auch das wird man als positive Analogie betrachten müssen. Sollte sich das als psychologisch unplausibel (d.h. als negative Analogie) erweisen, würde das

bedeuten, dass man die Übertragungsprozesse doch nicht derart radikal als neutrale Analogie behandeln kann und sie müssten detaillierter ausformuliert werden.

Tabelle 4: Entscheidungsbaum zum Morsecode als Tabelle
(Erläuterungen im Text)

a) Baum-Tabelle

Knoten-Nummer	Sprung bei "."	Sprung bei "-"	Zeichenende
1	2	3	Fehler
2	4	5	"E"
3	6	7	"T"
4	8	9	"I"
5	10	11	"A"
6	12	13	"N"
...	...		

b) Input-Tabelle

	1	2	3	4	5	6	7
	"."	"."	"-"	Pause			

5.1 Um die Aufgabe lösen zu können, müssen wir uns zuerst überlegen, wie wir das eigentliche Wissen, nämlich den Entscheidungsbaum darstellen wollen. Eine Möglichkeit ist, ihn als Tabelle zu repräsentieren. Jedem Knoten im Suchbaum soll eine Zeile der Tabelle entsprechen. Der ersten Kolonne kann entnommen werden, bei welchem Knoten, d.h. in welcher Zeile, man weiterfahren muss, wenn das nächste Wahrnehmungselement ein "Punkt" ist, der zweiten, was die Konsequenzen eines "Strichs" sind und die dritte steht für eine "Pause", also für das Zeichenende. Zeile eins steht für den Startknoten des Baums. Die ersten paar Zeilen der sind in Tabelle 4a dargestellt.

Nehmen wir an, dass das zu decodierende Morsezeichen auch als kleine "Tabelle" mit einer Zeile vorliegt, wie in Tabelle 4b dargestellt. Eine prozedurale Beschreibung des Decodierungsprozess könnte dann etwa wie folgt aussehen:

Verwende folgende Variablen:
Wahrnehmungselementnummer (anfänglich =1)
Wahrnehmungselement
Knotennummer (anfänglich =1)
Spaltennummer
Tabellenelement

1. Setze WAHRNEHMUNGSELEMENT gleich dem Zeichen in der Input Tabellen-Zeile in der Spalte WAHRNEHMUNGSELEMENTNUMMER.
2. Setze SPALTENNUMMER wie folgt:
   gleich 1, wenn WAHRNEHMUNGSELEMENT gleich "."
   gleich 2, wenn WAHRNEHMUNGSELEMENT gleich "-"
   gleich 3, sonst
3. Setze TABELLENELEMENT gleich dem Zeichen in der Baum-Tabelle in der Zeile KNOTENNUMMER und Spalte SPALTENNUMMER.
4. Ist TABELLENELEMENT ein Zeichen, dann drucke dieses Zeichen aus und stoppe.
5. Ist TABELLENELEMENT eine Zahl, dann
   setze KNOTENNUMMER gleich TABELLENELEMENT,
   erhöhe WAHRNEHMUNGSELEMENTNUMMER um 1,
   und gehe zu Punkt 1.

5.2 Auch hier müssen wir zuerst festlegen, wie wir den Entscheidungsbaum darstellen wollen. Prinzipiell könnten wir natürlich die Tabelle aus 5.1 (Tabelle 4) verwenden. Da es sich aber bei

einer funktionalen Sprache meist um LISP handeln wird, wollen wir sie hier als Liste (eine typische LISP-Struktur) definieren. Listen werden in LISP als eine Sequenz von Listenelementen zwischen zwei Klammer geschrieben; z.B. (a b c). Jedes Element in der Liste kann selbst wieder eine Liste sein, so dass tief geschachtelte Strukturen dargestellt werden können. Genau das wollen wir hier ausnutzen. Unsere Liste erhält auf der obersten Ebene folgende Form:

(( )( ) Fehler)

d.h. sie umfasst drei Elemente. Das erste wird benötigt, wenn es sich beim ersten Wahrnehmungselement um eine "." handelt, das zweite bei einem "-" und das dritte bei einer Pause. Die ersten beiden Elemente sind natürlich ihrerseits wieder Listen, die genau gleich aufgebaut sind. Stellen wir also die obersten beiden Ebenen dar, ergibt sich folgendes Bild:

((( )( ) "E")((( )( ) "T") Fehler)

Und so weiter, bis der ganze Baum dargestellt ist. Die rekursive Struktur dieses Baumes, d.h. dass jede Teilliste, aus der die Gesamtliste zusammengesetzt ist, wieder dieselbe Struktur aufweist, wie die Gesamtliste, lässt bereits erkennen, dass wir auf jeder Ebene des Baumes wieder genau dasselbe tun müssen und somit dieselbe Funktion rekursiv anwenden können. Den Input stellen wir analog auch als Liste dar, also etwa ("." "." "-" Pause). Wir benötigen folgende Funktion:

Funktion:
DECODIER-MORSE-ZEICHEN
Argument 1: Entscheidungsbaum
Argument 2: Inputliste
Wenn:
a) Das erste Element der INPUTLISTE eine "Pause" ist, dann ist das Resultat des letzte Element der Liste ENTSCHEIDUNGSBAUM.
b) Das erste Element der INPUTLISTE ein "." ist, dann ruf die Funktion DECODIER-MORSE-ZEICHEN auf; mit dem ersten Element der Liste ENTSCHEIDUNGSBAUM als erstes Argument und der INPUTLISTE ohne dem ersten Element als zweites Argument.
c) Das erste Element der INPUTLISTE ein "-" ist, dann ruf die Funktion DECODIER-MORSE-ZEICHEN auf; mit dem zweiten Element der Liste ENTSCHEIDUNGSBAUM als erstes Argument und der INPUTLISTE ohne dem ersten Element als zweites Argument.

Gestartet wird der ganze Prozess dadurch dass man die Funktion mit den vollständigen Entscheidungsbaum und der vollständigen Input-Liste aufruft. Verfolgt man dann, was geschieht, erkennt man, dass die Funktion sich immer wieder selbst auf sukzessive kleinere Teile des Baums ansetzt. Wir haben das Beispiel so konstruiert, um eine typische Eigenschaft funktionalen Programmierens zu demonstrieren. Ob dabei ein psychologisch plausibles Modell entstanden ist, wäre zu überprüfen.

5.3 Um ehrlich zu sein: Unsere PROLOG Erfahrungen reichen nicht aus, um diese Frage exemplarisch zu beantworten. Das Resultat wird ähnlich sein, wie die Produktionssystemlösung im folgenden Punkt, da man PROLOG auch als "backward chaining" Produktionssystem betrachten kann. Für eine "echte" PROLOG Lösung sollte sich der Leser/die Leserin aber an einen entsprechenden Experten/Expertin in seinem/ihrem Umfeld wenden.

5.4 In einer Produktionssystemsprache findet man einerseits Wenn-Dann-Regeln und andererseits "Fakten", auf die diese Regeln angewendet werden. Diese Fakten werden häufig als Listen dargestellt. Für eine einfache Simulation können wir einmal fünf verschiedene Typen von Fakten unterscheiden: Input, Input-Strom, Morsezeichen, Zeichen und Text. Wir kennzeichnen sie am besten dadurch, dass wir den Typ gleich als erstes Element in der entsprechenden Liste führen.

Input-Fakten sollen neben der Typenbezeichnung einfach noch ein "Wahrnehmungselement" enthalten: Punkt, Strich, Kurze-Pause oder Lange-Pause. Also zum Beispiel:

(Input Strich)
(Input Kurze-Pause)

Diese Input-Fakten werden von aussen ins Programm eingegeben und stellen die auditive Wahrnehmung dar. Der Input-Strom enthält eine ganze Liste von Wahrnehmungselementen, also:

(Input-Strom Punkt Strich Kurze-Pause)
(Input-Strom Lange-Pause Strich Strich Kurze-Pause)

Ein Morsezeichen enthält eine Liste, in der nur Punkt und Strich vorkommen, also:

(Morsezeichen Strich Strich Punkt)
(Morsezeichen Punkt)

Zeichen enthält ein Zeichen oder Sonderzeichen aus dem normalen Alphabet, also:

(Zeichen a)
(Zeichen ?)

Text enthält eine Liste bestehend aus Zeichen sowie "Abstand" für Wortabständen, also:

(Text d a s Abstand i s t Abstand e i n Abstand t e x t)

Wir benötigen nun eine erste Regel, die den Input in den Inputstrom aufnimmt:

Regel 1: Wenn
ein neues Input-Fakt vorliegt
dann hänge das Wahrnehmungselement daraus hinten an den Input-Strom.

Die zweite Regel isoliert vollständige Morsezeichen aus dem Inputstrom:

Regel 2: Wenn
das letzte Element im Input-Strom eine Kurze- oder Lange-Pause ist,
dann entferne alles, was vor dieser Pause im Input-Strom steht und bilde
damit ein neues Morsezeichen-Fakt.

Eine dritte Regel fügt Wortabstände in den Text ein:

Regel 3: Wenn
das erste Element im Input-Strom eine Lange-Pause ist,
dann entferne diese aus dem Input-Strom und hänge ein Abstand hinten an den Text.

Eine vierte Regel ist rein technischer Natur und eliminiert die verbleibenden Kurzen-Pausen aus dem Input-Strom:

Regel 4: Wenn
das erste Element im Input-Strom eine Kurze-Pause ist,
dann entferne sie.

Die fünfte Regel fügt die decodierten Zeichen in den Text ein:

Regel 5: Wenn
ein neues Zeichen vorliegt,
dann hänge es hinten an den Text.

Was nun noch fehlt, sind Regeln, die die eigentliche Decodierung vornehmen. Wenn wir diesen Prozess nicht genauer ausformulieren wollen, können wir dies lösen, indem wir für jedes Morsezeichen eine separate Regel schreiben. Das Modell, das wir so erhalten, eignet sich vor allem, um die Koordination von drei vermutlich parallel ablaufenden Prozessen zu studieren: Isolation einzelner Zeichen (Regel 2), Decodierung (die hier nicht ausformulieren Regeln) und Schreiben (Regeln 3 und 5). So wie das Modell jetzt formuliert ist, besteht eine Möglichkeit,

dass Regel 3 die Wortpause in den Text einfügt, bevor das davorliegende Morsezeichen decodiert ist. Ob dies eine positive Analogie des Modells darstellt, wäre zu untersuchen.

6.1 Verzichtet man auf die übliche handschriftliche "Ausgabe" der Zeichensequenz durch die Versuchsperson und lässt sie die erkannten Zeichen direkt am Terminal eintippen, kann der ganze Ablauf problemlos automatisiert werden. Wir benötigen dazu ein kleines Steuerprogramm, dass die zu decodierenden Zeichen z.B. aus einem file einliest oder zufällig generiert. Dieses Programm leitet dann die einzelnen Zeichen einerseits dem Simulationsprogramm zu und erzeugt andererseits akustisch für die Versuchsperson wahrnehmbar kurze und lange Töne getrennt durch die korrekten Pausen. Das Simulationsprogramm läuft bis zu der Stelle, an der das nächste Zeichen decodiert ist und wartet dort auf den Input der Versuchsperson, dieser wird dann an Stelle des vom Programm ermittelten Outputs weiterverwendet. Es sollte ohne weiteres möglich sein, diese Parallelität auch für hohe Übertragungsgeschwindigkeiten zu erreichen.

6.2 Grundsätzlich natürlich, ob das Programm das Modellziel erreicht, d.h. ob es die richtigen Fehler macht. Dazu liessen sich einfach einmal einzelne, fehlerhafte Zeichen, wie sie durch eine bestimmte Input-Sequenz provoziert werden, vergleichen. Denkbar wäre aber auch, dass wir im Programm "Folgefehler" berücksichtigen müssen, d.h. dass Fehler nicht nur davon abhängig sein können, was zuvor tatsächlich gesendet wurde, sondern auch davon, was decodiert wurde. Ein Beispiel dafür wäre etwa, dass der Empfänger bemerkt, dass er einen Fehler gemacht hat und beim Versuch, diesen zu korrigieren, gleich mehrere der folgenden Zeichen verpasst. Auch das stellt kein Problem dar, da ja das parallelisierte Programm nicht mit dem arbeitet, was es selbst "verstanden" hat, sondern mit der Reaktion der Versuchsperson.

Wie weit sich von den Versuchspersonen im Falle von Abweichungen Informationen erfragen lassen, die diese Abweichung klären, ist schwierig abzuschätzen. Macht die Versuchsperson z.B. einen Fehler nicht, der vom Programm prognostiziert wird, könnte es sein, dass die Versuchsperson z.B. die Gefahr eines solchen Fehlers durchaus bestätigt, dann aber auf einen "Trick" hinweist, der ihr hilft, ihn zu vermeiden. Macht die Versuchsperson dagegen einen Fehler, den das Programm nicht erwartet, dann wäre z.B. denkbar, dass die Versuchsperson in der Befragung darauf hinweist, dass es sich um eine generell schwierige Zeichenkombination handelt. Etc.

6.3 Das hängt unter anderem vom Ziel ab, das man mit dem entsprechenden Experiment verfolgt. Geht es darum, das Programm zu evaluieren und einfach zu testen, ob es die richtigen Fehler produziert, dann kann nur in den Fällen von einer Abweichung gesprochen werden, in denen das Programm mit hundertprozentiger Sicherheit einen oder keinen Fehler prognostiziert. Liegt ein Zufallsentscheid vor, kann im Einzelfall nicht von einer Abweichung gesprochen werden. Abweichungen können sich hier nur in der Fehlerhäufigkeit über eine längere Sequenz von Zeichen ergeben. Verwendet man das Experiment aber dazu, um mit der Versuchsperson all jene Situationen durchzubesprechen, in denen eine Abweichung auftritt, dann ist es sinnvoll, jede Situation, in der aus der Sicht des Programms eine erhöhte Fehlerwahrscheinlichkeit vorlag und in der die Versuchsperson keinen Fehler produzierte als Abweichung zu betrachten - egal ob das Programm selbst wirklich einen Fehler produziert hat.

6.4 Aufgrund der Überlegungen unter 6.3 scheint es sinnvoll, alle Situationen festzuhalten, in denen die Wahrscheinlichkeit eines Fehlers aus der Sicht des Programms erhöht war, wo aber die Versuchsperson keinen Fehler begangen hat. Und selbstverständlich sind auch alle Situationen von Interesse, in denen die Vp einen Fehler begangen hat, obwohl das Programm keinen erwarten liesse. Da wir hier untersuchen wollen, wie Fehler durch bestimmte Sequenzen von Zeichen provoziert werden, ist es notwendig, zumindest die letzten drei bis vier (oder auch mehr, je nachdem wie weit zurück eine mögliche Ursache erwartet wird) Zeichen vor dem Fehler festzuhalte. Dasselbe gilt für Zeichen nach dem Fehler, da wegen den vermutlich parallel ablaufenden Verarbeitung mehrerer Zeichen durchaus denkbar ist, dass die Störung von den folgenden Zeichen ausgeht.

7.1 Interessant ist in diesem Fall sicher, zu verfolgen, wie sich die Belegung des Kurzzeitgedächtnisses verändert. Einmal lässt sich dadurch der genaue Ablauf verfolgen und zum zwei-

ten werden so Probleme, wie z.B. die Überlastung dieses Speichers, sichtbar. Eine Möglichkeit eine Spur zu erstellen besteht also darin, in regelmässigen (simulierten) Zeitintervallen die Belegung des Kurzzeitgedächtnisses auszudrucken; z.B. als Tabelle, bei der jede Spalte für einen Speicherplatz und jede Zeile für einen Zeitpunkt steht.

Ebenfalls von Interesse könnten statistische Angaben über die Belegung des Kurzzeitgedächtnisses sein, wie etwa: Minimale Belegung, maximale Belegung, durchschnittliche Belegung, durchschnittliche Wartezeit eines Items, etc.

7.2 Da nicht zu erwarten ist, dass ihr Programm "versteht", was da gesendet wird, muss es sich also um die Simulation der Decodierung einer Zufallsfolge von Zeichen handeln. (Dies ist übrigens keine unrealistische Situation, da Texte oft verschlüsselt gesendet werden, so dass sie für den Decodierer tatsächlich zuerst einmal nicht mehr Struktur enthalten als eine Zufallssequenz (Pfisterer, 1988). Insofern ist die Sache relativ einfach, da für weitere Tests beliebige zufallsgenerierte Sequenzen verwendet werden können.

7.3 Wahrscheinlich war das eine unergiebige Frage. Uns ist jedenfalls dazu nichts eingefallen.

8.1 Es ergibt sich folgende Fehlerverteilung

Gesendetes Zeichen : I (. . Pause)

Wahrgenommenes Zeichen	Morsecode	Wahrscheinlichkeit
I	(. . Pause)	0.729
keines	(Pause)	0.05
E	(. Pause)	0.045
A	(. - Pause)	0.0405
N	(- . Pause)	0.0405
S	(. . . Pause)	0.0405
U	(. . - Pause)	0.0405
T	(- Pause)	0.0025
M	(- - Pause)	0.00225
R	(. - . Pause)	0.00225
W	(. - - Pause)	0.00225
D	(- . . Pause)	0.00225
K	(- . - Pause)	0.00225
G	(- - . Pause)	0.000125
O	(- - - Pause)	0.000125

Die Fehlerhäufigkeiten sind exakt berechnet. Sie liessen sich aber natürlich auch über eine Monte-Carlo-Simulation ermitteln (z.B. Scheid, 1968). Sie decken sich übrigens nicht mit den tatsächlich beobachteten. Z.B. tritt "S" weitaus am häufigsten auf und "U" praktisch nie (Pfisterer, 1988).

8.2 Der Buchstabe "I", für den die Wahrscheinlichkeitsverteilung aus Aufgabe 8.1 vorliegt, eignet sich nur schlecht, wie eine Gegenüberstellung der beiden Verteilungen zeigt (es sind nur Zeichen berücksichtigt, die in mindestens einer Verteilung 1% oder mehr erreichen):

Gesendetes Zeichen : I

Wahrgenommenes Zeichen	Wahrscheinlichkeit Prozess 1		Wahrscheinlichkeit Prozess 2
I	0.729		0.81
keines	0.05	>	0.000625
E	0.045		0.045
A	0.0405		0.045
N	0.0405		0.045
S	0.0405		0.045
U	0.0405	>	0.00125

Markante Unterschiede ergeben sich nur, wenn fälschlicherweise gar kein Zeichen erkannt wird, bzw. für den Fehler "U". Wenn wir also nur eigentliche Verwechslungen betrachten, zeigt sich nur bei einem von fünf relativ häufigen Fehlern ein Unterschied. Und dies obwohl z.B. der Fehler "S" auf ganz unterschiedliche Art und Weise zustande kommt: Im Prozess 1 durch Anhängen eines Punktes, im Prozess 2 durch Verdoppelung des ersten oder des zweiten Punktes. Gerade dieser Vergleich der Fehlergenese zeigt aber, dass die gleiche Fehlerwahrscheinlichkeit nur dadurch entsteht, dass im zweiten Prozess zwei unterschiedliche Fehler zum gleichen Resultat führen. Bei einem Zeichen, bei dem dies nicht möglich ist, sollten wir deshalb einen deutlicheren Unterschied haben. Dies bestätigt sich auch, wenn wir die Verteilungen für den Buchstaben "N" vergleichen:

Gesendetes Zeichen : N (- .)

Wahrgenommenes Zeichen	Wahrscheinlichkeit Prozess 1		Wahrscheinlichkeit Prozess 2
N	0.729		0.81
T	0.045	>	0.025
D	0.0405	>	0.0225
I	0.0405		0.045
K	0.0405	>>	0
M	0.0405		0.045
E	0.0025	<<	0.025
G	0.0025	<<	0.0225

Hier ergeben sich bei fünf von sieben häufigen Fehlern deutliche Unterscheide. Vergleicht man alle Buchstaben auf diese Art, dann ergibt sich folgendes Resultat: Ausgesprochen ungünstig sind I und M (nur bei 20% der möglichen Fehler lassen die beiden Prozesse deutlich unter-

schiedliche Häufigkeiten erwarten). Günstig sind dagegen A, K, N, und R (bei 70% und mehr Prozent der möglichen Fehler deutliche Unterschiede). Besonders aufschlussreich sind K und R, da hier acht von elf möglichen Fehlern deutlich unterschiedliche Häufigkeiten erwarten lassen, bei fünf davon mindestens um das zehnfache.

Genau genommen lässt sich die Frage allerdings so, wie sie gestellt ist und wie wir das hier versucht haben, nicht beantworten, da sie zu wenig Informationen über den Match-Prozess enthält. Die Vorlage muss ja mit jeder einzelnen Schablone verglichen werden (entweder der Reihe nach - sequentiell - oder mit allen gleichzeitig - parallel). Und da jeder dieser Vergleiche fehleranfällig ist, ist denkbar, dass mehrere Schablonen gleichzeitig "passen". Wir müssten deshalb zuerst noch spezifizieren, was in einem solchen Fall geschieht, bevor wir Fehlerverteilungen für den zweiten Prozess aufstellen können.

11 Anhang: Literatur

Albert, D. (Hrsg.)(1985) Bericht über den 34. Kongress der Deutschen Gesellschaft für Psychologie. Göttingen: Hogrefe.

Allen, J. (1978) Anatomy of LISP. New York: McGraw-Hill.

Ayers, A.J. (1984) Bausteine der kindlichen Entwicklung. Heidelberg: Springer.

Barr, A. & Feigenbaum, E.A. (eds) (1981) The handbook of artificial intelligence. Vol. I & II. Los Altos, Cal.: William Kaufmann.

Bennett, M. (1976) SUBSTITUTOR: A teaching program. Unpublished project report. Department of Artificial Intelligence, University of Edinburgh.

Bobrow, D.G. & Collins, A. (eds.) (1975) Representation and understanding, studies in cognitive science. New York: Academic Press.

Boden, M. (1977) Artificial intelligence and natural man. New York: Basic Books.

Boden, M. (1988) Computer models of mind. Cambridge: University Press.

Bortz, J. (1984) Lehrbuch der empirischen Forschung für Sozialwissenschaftler. Berlin, Heidelberg: Springer.

Brancazio, P.J. (1984) Sport Science. New York: Simon and Schuster.

Brooks, R.A. (1981) Symbolic reasoning among 3-d objects and 2-d models. AI Journal, 16, 285-348.

Carver, S.M. (1986) LOGO debugging skills: Analysis, instruction, and assessment. Unpublished doctoral dissertation, Department of Psychology, Carnegie-Mellon University.

Charniak, E., Riesbeck, C.K. & McDermott, D.V. (1980) Artificial intelligence programing. Hillsdale, N.J.: Lawrence Erlbaum

Chomsky, N. (1965) Aspects of a theory of syntax. Cambridge, Mass.: MIT Press.

Clocksin, W.F. & Mellish, C.S. (1981) Programming in PROLOG. New York: Springer.

Cohen, J. (1977) Statistical power analysis for the behavioral sciences. New York: Academic Press.

Cohen, P.R. & Feigenbaum, E.A.(eds) (1982) The handboook of artificial intelligence. Vol. III. Los Altos, Cal.: William Kaufmann.

Cohen, P.R. & Howe, A.E. (1988) How evaluation guides AI research. AI magazine, 9, 4, 35-43.

Davis, R. & Lenat, D.B. (1980) Knowledge-based systems in artificial intelligence. New York: McGraw-Hill.

Dörner, D., Kreuzig, H. Reither, F. & Stäudel, T. (1983) Lohausen. Vom Umgang mit Unbestimmtheit und Komplexität. Bern: Huber.

Dreyfus, H.L. (1985) Die Grenzen der künstlichen Intelligenz. Königstein: Athenäum.

Erman, L.D. & Lesser, V.R. (1980) The HEARSAY-II speech understanding system: A tutorial. In Lea, 1980, 361-381.

Feigenbaum, E. A. (1963) The simulation of verbal learning behavior. In: Feigenbaum & Feldmann, 1963, 297-309.

Feigenbaum, E. A. & Feldman, J. (eds.)(1963) Computers and thought. New York: McGraw-Hill.

Fertig, H. (1977) Modelltheorie der Messung. Berlin: Duncker & Humbolt.

Funke, J. (1984) Diagnose der westdeuschen Problemlöseforschung in einigen Thesen. Sprache & Kognition, 3, 159-172.

Gigerenzer, G. (1981) Messung und Modellbildung in der Psychologie. München, Basel: E.Reinhardt.

Goffman, E. (1968) Wir spielen alle Theater. Die Selbstdarstellung im Alltag. München.

Green, C.C. (1969) The application of theorem-proving to question-answering systems. IJCAI 1, 219-237.

Grimm, H. & Engelkamp, J. (1981) Sprachpsychologie. Handbuch und Lexikon der Psycholinguistik. Berlin: E.Schmidt.

Guzman, A. (1968) Computer recognition of three-dimensional objects in a visual scene. Tech. Rep. MAC-TR-59, AI Laboratory, Massachusetts Institute of Technology.

Herschel, J. (1830) A preliminary discourse on the study of natural philosophy. London:

Herrmann, T. (1990) Die Experimentiermethode in der Defensive? Sprache & Kognition, 9, 1-11.

Herzog, W. (1984) Modell und Theorie in der Psychologie. Göttigen: Hogrefe.

Holzkamp, K. (1973) Kritische Psychologie. Vorbereitende Arbeiten. Frankfurt a.M.: Fischer TB.

Husserl, E. (1950) Ideen zu einer reinen Phänomenologie und phänomenologische Philosophie. Haag: Martinus Nijhoff.

Johnson-Laird, P.N. (1981) Mental models in cognitive science. In: Norman, 1981, 147-192.

Johnson-Laird, P.N. & Steedman, M. (1978) The psychology of syllogisms. Cognitive Psychology, 10, 64-99

Kaiser, H. (1980) Wissenschaftstheoretische und erkenntnistheoretische Überlegungen im Rahmen der Sozialwissenschaften. Lizentiatsarbeit, Psychologisches Institut der Universität Bern.

Kaiser, H. (1985) Problemlösen zu zweit. Dissertation. Psychologisches Institut der Universität Fribourg.

Kaiser, H. (1987) Wissensaustausch im Dialog. Bern: Huber

Kaiser, H & Keller, B. (1985) Vorstudien zu einer komplexen Lernumgebung. Internes Forschungspapier. Psychologisches Institut der Universität Bern.

Kaiser, H. & Keller, B. (1986) Lernen durch den Erwerb neuer Erfahrungen. Internes Forschungspapier. Psychologisches Institut der Universität Bern.

Kaiser, H. & Keller, B. (1987a) Learning: Fitting the world model to the task. Internes Forschungspapier. Psychologisches Institut der Universität Bern.

Kaiser, H. & Keller, B. (1987b) "Version 5". Internes Forschungspapier. Psychologisches Institut der Universität Bern.

Kaiser, H. & Keller, B. (1989) Learning: Fitting the world model to the task. Can we learn every world model for every task "by doing"? In: Mandl, H., DeCorte, E., Bennett, N. & Friedrich, H.F. (eds.) Learning and instruction. European research in an international context. Vol. 2.1. Social and cognitive aspects of learning and instruction. Oxford: Pergamon Press, 219-230.

Klahr, D. & Carver, S.M. (1988) Cognitive objectives in a LOGO debugging curriculum: instruction, learning, and transfer. Cognitive Psychology, 20, 362-404.

Kluwe, R.H., Misiak, C. & Schmidle, R. (1985) Wissenserwerb beim Umgang mit umfangreichen Systemen: Lernvorgänge als Ausbildung subjektiver Ordnungsstrukturen. In: Albert, 1985, 255-257.

Kohonen, T., Mäkisara, K. & Saramäki, T. (1984) Phonotopic maps - insightful representations of phonological features of speech recognition. In: Proc. of the Seventh International Conference on Pattern recognition, Montreal, Canada. Silver Spring: IEEE Computer Society, 182-185.

Kuhn, T.S. (1967) Die Struktur wissenschaftlicher Revolutionen. Frankfurt a.M.: Suhrkamp.

Langely, P. (1985) Learning to search: From weak methods to domain-specific heuristics. Cognitive Science, 9, 217-260.

Lea, W. (ed.)(1980) Trends in speech recognition. Englewood Cliffs, N.J.: Prentice-Hall.

Locke, W.N. & Booth, A.D. (eds.)(1955) Machine translation of languages. New York: Technology Press of MIT and Wiley.

Losee, J. (1977) Wissenschaftstheorie. Eine historische Einführung. München: Beck.

McClelland, J.L., Rumelhart, D.E. et al. (1986) Parallel distributed processing. Vol II. Cambridge, Mass.: MIT Press.

McCulloch, W.S. & Pitts, W. (1943) A logical calculus of ideas immanent in neural nets. Bulletin of Mathematical Biophysics, 5, 115-137.

Michalski, R.S., Carbonell, J.G. & Mitchell, T.M. (eds.)(1983) Machine learning, an AI approach. Palo Alto: Tioga.

Minsky, M. (1975) A framework for representing knowledge. In: Winston, 1975, 211-277.

Minsky, M. & Papert, S. (1969) Perceptrons; an introduction to computational geometry. Cambridge, Mass.: MIT Press.

Newell, A., Shaw, J.C. & Simon, H.A. (1963a) Chess-playing programs and the problem of complexity. In: Feigenbaum & Feldman, 1963, 39-70.

Newell, A., Shaw, J.C. & Simon, H.A. (1963b) Empirical explorations with the logic theory machine: A case history in heuristics. In: Feigenbaum & Feldman, 1963, 109-133.

Newell, A. & Simon, H.A. (1963) GPS, a program that simulates human thought. In: Feigenbaum & Feldman, 1963, 279-293.

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

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

Nilsson, N.J. (1980) Principles of Artificial Intelligence. Palo Alto, Cal.: Tioga.

Norman, D.A (ed.)(1981) Perspectives on cognitive science. Norwood, N.J.: Ablex.

Oettinger, A.G. (1955) The design of an automatic Russian-English technical dictionary. In: Locke & Booth, 1955, 47-65.

Opwis, K. (1985) Mentale Modelle dynamischer Systeme. Forschungsbericht Nr. 30. Freiburg i.Br.: Psychologisches Institut.

Opwis, K., Spada, H. & Schwiersch, M. (1985) Erwerb und Anwendung von Wissen über ein ökologisches System. Forschungsbericht Nr. 23. Freiburg i.Br.: Psychologisches Institut.

Papert, S. (1980) Mindstorms: Children, computers, and powerful ideas. New York: Basic Books.

Pearl, J. (1984) Heuristics. Reading, Mass.: Addison-Wesely.

Pfisterer, P. (1988) Morselernen: Psychologische Studie zum Morselehrgang der Radio-Schweiz AG. Dissertation. Psychologisches Institut der Universität Fribourg.

Piaget, J. (1974) Biologie und Erkenntnis. Frankfurt a.M.: Fischer.

Reddy, R., Erman, L., Fenell, R. & Neely,R. (1976) The HEARSAY speech understanding system: An example of the recognition process. IEEE Transactions on Computers C-25:427-431.

Reichenbach, H. (1938) Experience and prediction. Chicago: Univ. of Chicago Press.

Rumelhart, D.E., McClelland, J.L (1986) On learning the past tense of english verbs. In: McClelland, Rumelhart et al., 1986, 216-271.

Reichert, U. & Dörner, D. (1988) Heurismen beim Umgang mit einem "einfachen" dynamischen System. Sprache & Kognition, 7, 12-24.

Rumelhart, D.E., McClelland, J.L. et al. (1986) Parallel distributed processing, Vol. I. Cambridge, Mass.: MIT Press.

Sacerdoti, E.D. (1974) Planning in a hierarchy of abstraction spaces. Artificial Intelligence 5, 115-135.

Samuel, A.L. (1963) Some studies in machine learning using the game of checkers. In: Feigenbaum & Feldman, 1963, 71-105.

Schank, R.C. (1975) Conceptual information processing. New York: North Holland.

Schank, R.C. (1981) Inside computer understanding. Hillsdale, N.J.: Lawrence Erlbaum.

Schank, R.C. (1982) Dynamic memory. Cambridge, Mass: Cambridge University Press.

Schank, R.C. (1984) Looking for a process model of dialogue. In: Vaina & Hintikka, 1984, 161-173.

Schank, R.C. & Abelson, R.P. (1977) Scripts, plans, goals, and understanding. Hillsdale, N.J.: Lawrence Erlbaum.

Schank, R.C. & Colby, K.M. (eds.) (1973) Computer models of thought and language. San Francisco: Freeman.

Scheid, F. (1968) Theory and problems of numerical analysis. New York: McGraw-Hill.

Schenkein, J. (ed.)(1978) Studies in the organisation of conversational interaction. New York: Academic Press.

Schulz von Thun (1981) Miteinander reden: Störungen und Klärungen. Psychologie der zwischenmenschlichen Kommunikation. Reinbeck b. Hamburg: Rowohlt.

Shannon, C.E. (1950) Programming a computer for playing chess. Philosophical Magazine (Series 7) 41:256-275.

Shortliffe, E.H. (1976) Computer-based medical consultations: MYCIN. New York: American Elsevier.

Simon, H.A. & Newell, A. (1958) Heuristic problem solving: the next advance in operations research. Operations Research, 6.

Thoman, E.B. (ed.)(1979) Origins of the infant's social responsiveness. Hillsdale, N.J.: Lawrence Erlbaum.

Vaina, L. & Hintikka. J. (eds.)(1984) Cognitive constraints on communication. Dordrecht: D.Reidel.

Waltz, D. (1975) Generating semantic descriptions from drawings of scenes with shadows. In: Winston, 1975, 19-92.

Watson, J.S. (1979) Perception of contingency as a determination of social responsiveness. In: Thoman, 1979, 33-64.

Wason, P.C. & Johnson-Laird, P.N. (1972) Psychology of reasoning: structure and content. Cambridge, Mass.: Harvard University Press.

Whitehead, A.N. & Russel, B. (1925) Principia mathematica. Cambridge, England: University Press.

Wilks, Y.A. (1973) An artificial intelligence approach to machine translation. In: Schank & Colby, 1973, 114-151.

Winograd, T. (1972) Understanding natural language. New York: Academic Press.

Winograd, T. (1983) Language as a cognitive process. Reading, Mass.: Addison-Wesley.

Winograd, T. & Flores, F. (1986) Understanding computers and cognition. Norwood, N.J.: Ablex.

Winston, P. (ed.)(1975) The psychology of computer vision. New York: McGraw-Hill.

Winston, P.H. (1977) Artificial intelligence. New York: Addison-Wesley.

Winston, P.H. (1987) Künstliche Intelligenz. Frankfurt: Addison-Wesley.

Wittgenstein, L. (1984) Philosophische Untersuchungen. Frankfurt: Suhrkamp, stw 501.

Woods, W.A. (1968) Procedural semantics for a question-answering machine. Fall Joint Computer Conference, 33, 457-471.

Wottawa, H. (1977) Psychologische Methodenlehre. München: Juventa.

Young, R.M. & O'Shea, T. (1981) Errors in children's subtraction. Cognitive Science, 5, 153-177.

Yu, V.L. et al. (1979) Antimicrobical selection by a computer - A blind evaluation by infectious disease experts. J. American Medical Association, 242, 1279-1282.

Für Peter Pfisterer

der uns klar gemacht hat, dass es dieses Buch braucht.

Im Übrigen möchten wir folgenden Personen und Institutionen für ihre Unterstützung bei der Erstellung dieses Buches danken: Dem Schweizerischen Nationalfonds zur Förderung der wissenschaftlichen Forschung für die finanzielle Unterstützung im Rahmen der Projekte Nr. 1.169-0.85 und 1.729-0.87. Prof. Dr. A.Flammer für das kontinuierliche Wohlwollen, dass er unserer Arbeit entgegenbringt. Und nicht zuletzt Hilde Haider, Thomas Rothenfluh, Heinz Süss und Michael Wolf für die kritische Durchsicht des Manuskriptes und für die vielen, konstruktiven Anregungen.