KfK 5066
Oktober 1992

Wissensbasierte Systeme

Darstellungs- und Verarbeitungsmodelle

H. B. Keller, T. Weinberger, E. Kugele,
B. große Osterhues

Institut für Angewandte Informatik
Projekt Schadstoff- und Abfallarme Verfahren

Kernforschungszentrum Karlsruhe

Kernforschungszentrum Karlsruhe

Institut für Angewandte Informatik
Projekt Schadstoff- und Abfallarme Verfahren

KfK 5066

Wissensbasierte Systeme

Darstellungs- und Verarbeitungsmodelle

Hubert B. Keller, Thomas Weinberger,
Eugen Kugele, Bernhard große Osterhues

Kernforschungszentrum Karlsruhe GmbH, Karlsruhe

Kernforschungszentrum Karlsruhe GmbH
Postfach 3640, 7500 Karlsruhe 1

ISSN 0303-4003

Zuammenfassung:

Das Ziel einer innovativen Prozeßführung ist die Optimierung des Betriebes technischer Anlagen/Verfahren hinsichtlich einer prozeßintegrierten Vermeidung der Bildung von Schadstoffen im Sinne eines aktiven Umweltschutzes.
Neben den konventionellen Methoden der Informationstechnik sind hierzu auch neuartige Verfahren wie wissensbasierte Techniken einzusetzen. Lernverfahren können zur automatischen Wissensakquisition und -validierung herangezogen werden.

Der vorliegende Bericht beschreibt die wesentlichen Wissensrepräsentationsformen der Künstlichen Intelligenz wie Logik, Regeln, Rahmen und Constraints mit den jeweiligen Schlußfolgerungsverfahren. Die Konzepte der objektorientierten Programmierung, elementare Problemlösungsstrategien und Problemlösungstypen in Expertensystemen werden dargestellt. Das Problem der Wissensakquisition wird anhand symbolischer und neuronaler Methoden charakterisiert.

Knowledge-based Systems - Models for Representation and Processing

Abstract:

The optimization of the operation of technical systems or processes, in order to avoid the production of ecologically harmful substances, in sense of an active conservation of the environment, is the aim of an innovative process-control.
Besides the standard methods of computer science, new methods like knowledge-based techniques should be applied. For automatic knowledge acquisition and -validation learning techniques can be used.

This report describes the essential artificial intelligence knowledge representation techniques like logic, rules, frames and constraints with their related inference methods. The concepts of object-oriented programming, fundamental problem solving strategies and problem solving types in expert systems are shown. The problem of knowledge acquisition is characterised by symbolic and neural methods.

Inhaltsverzeichnis

Zusammenfassung 1

1 Die Grundidee wissensbasierter Systeme 3

2 Wissensrepräsentationsformen 5

2.1 Einleitung 5
2.2 Elementare Wissensrepräsentationsformen 7
2.2.1 Merkmalsvektoren 7
2.2.2 Parameter in algebraischen Ausdrücken 7
2.3 Logik 8
2.3.1 Klassische Logik 8
2.3.1.1 Die Aussagen- und die Prädikatenlogik 8
2.3.1.1.1 PROLOG 10
2.3.1.2 Eigenschaften von Kalkülen 10
2.3.2 Nicht-klassische Logik 11
2.3.2.1 Einleitung und Überblick 11
2.3.2.2 Fuzzy Logik 13
2.3.2.3 Beispiel der Wissensverarbeitung mit Fuzzy Logik 14
2.4 Regeln 15
2.5 Rahmen 17
2.5.1 Semantische Netze 17
2.5.2 Objekte/Frames 17
2.5.2.1 Automatische Klassifikation in KL-ONE-Sprachen 20
2.5.3 Skripten 22
2.6 Computer-Algebra 23
2.7 Constraints 25
2.8 Spezielle Schlußfolgerungsverfahren 26
2.8.1 Probabilistisches Schließen 26
2.8.1.1 Das Theorem von Bayes 27
2.8.2 Nicht-monotones Schließen 29
2.8.3 Temporales Schließen 31

3 Objektorientierte Programmierung 33

3.1 Einleitung 33
3.2 Die Grundkonzepte objektorientierter Programmierung 34
3.2.1 Überblick 34
3.2.2 Objekte und Kommunikation 34
3.2.3 Klassen und Instanzen 36
3.2.4 Vererbung und abstrakte Klassen 38
3.2.5 Polymorphismus und dynamisches Binden 41
3.3 Weiterführende Konzepte objektorientierter Programmierung 43
3.3.1 Verschiedene Formen der Vererbung 43
3.3.2 Metaklassen 44
3.3.3 Objektverwaltung 45
3.3.4 Konstruktionsprinzip 46
3.3.5 Generische Klassen 47
3.4 Resümee 47

4 Allgemeine Problemlösungsstrategien 49

4.1 Beschreiben und Vergleichen 49
4.2 Zielreduktion (Teilzielbildung) 49
4.3 Propagierung natürlicher Beschränkungen 50
4.3.1 Propagierung symbolischer Beschränkungen 50
4.3.2 Propagierung numerischer Beschränkungen 50
4.3.3 Abhängigkeitsgesteuerte Rückziehverfahren 51
4.4 Suchprobleme 52
4.4.1 Grundlegende Prozeduren zum Finden eines Pfades 53
4.4.1.1 Tiefensuche 53
4.4.1.2 Bergsteigen 54
4.4.1.3 Breitensuche 54
4.4.1.4 Strahlensuche 55
4.4.1.5 Bestensuche 55
4.4.2 Die Suche nach dem besten Weg 56
4.4.2.1 Der "British Museum Algorithmus" 56
4.4.2.2 Verzweige und Begrenze 56
4.4.2.3 Dynamisches Programmieren 57
4.4.2.4 A* 57

4.4.3 Suchprobleme bei Spielen ... 58
4.5 Die Mittel-Zweck-Analyse (Means-Ends-Analysis) ... 59
4.6 Generieren-und-Testen von Systemen ... 60
4.7 Regelbasierte Systeme ... 60
4.8 Formale (klassische) Logik ... 62
4.8.1 Logik als Theorembeweiser ... 63
4.8.2 Logik zur Planung von Operatorsequenzen ... 64
4.8.3 Logik in Beschränkungsnetzen ... 64

5 Problemlösungstypen in Expertensystemen ... 67

5.1 Der Problemlösungstyp Diagnostik ... 68
5.1.1 Übersicht über Diagnosetechniken ... 69
5.1.1.1 Wissensrepräsentation ... 69
5.1.1.2 Kontrollstrategien ... 70
5.1.1.3 Probabilistische Diagnosebewertung ... 71
5.1.2 Übersicht über Diagnostiksysteme ... 72
5.1.2.1 Statistische und fallvergleichende Diagnostiksysteme ... 72
5.1.2.2 Heuristische (assoziative) Diagnostiksysteme ... 72
5.1.2.3 Modellbasierte Diagnostiksysteme ... 73
5.1.3 Integrationsmöglichkeit verschiedener Diagnostiksysteme ... 74
5.2 Der Problemlösungstyp Konstruktion ... 75
5.3 Der Problemlösungstyp Simulation ... 77

6 Wissensakquisition ... 81

6.1 Symbolische Wissensakquisition ... 81
6.1.1 Arten des Wissenserwerbs ... 81
6.1.2 Phasenmodell des Wissenserwerbs ... 82
6.2 Wissensakquisition durch neuronale Netze ... 84
6.2.1 Die Grundelemente neuronaler Netze ... 84
6.2.1.1 Das Prozeßelement ... 84
6.2.1.2 Die Funktionen (Prozesse) des Prozeßelements ... 84
6.2.1.3 Lernfunktionen ... 86
6.2.1.4 Die Kombination von Prozeßelementen zu einem Layer ... 86
6.2.1.5 Die Kombination mehrerer Layer zu einem mehrschichtigen neuronalen Netz ... 87

6.2.2 Lernen in neuronalen Netzen 88
6.2.2.1 Verschiedene Formen des Lernens in neuronalen Netzen 89
6.2.2.2 Lernregeln 91
6.2.3 Anwendungen neuronaler Netze 93

Referenzen 97

Stichwortverzeichnis 101

Zusammenfassung

Betrachtet man einen technischen Prozeß hinsichtlich der Bildung von Schadstoffen, so muß ein *intelligenter Ansatz zur Prozeßführung* grundsätzlich eine Minimierung der entstehenden Schadstoffe anstreben, um somit die Problematik der Beseitigung von Schadstoffen quasi "an der Wurzel" anzugehen. Daher muß das oberste Ziel bei der Führung technischer Anlagen die *prozeßintegrierte Vermeidung* der Bildung von Schadstoffen sein.

Der anhaltende Fortschritt im Bereich der Hardware, welcher weiter steigende Leistung bei sinkenden Preisen bietet, und die kontinuierliche Weiterentwicklung der Softwaretechniken erschließen heute neue Möglichkeiten, den Wirkungsgrad und die Verfügbarkeit einer Anlage mit Hilfe weiter verbesserter, d. h. intelligenterer Prozeßführungssysteme noch zu steigern.

Neben den konventionellen Methoden der Informationstechnik sind hierzu auch neuartige Verfahren wie wissensbasierte Techniken einzusetzen. Lernverfahren können zur automatischen Wissensakquisition und -Validierung herangezogen werden. Das Ziel sind Systeme, die das Wissen zur optimalen Führung eines technischen Prozesses sukzessive on-line ableiten und dem Bediener zur Verfügung stellen /14/.

Hierzu bildet der vorliegende Bericht eine Einführung und einen Überblick in den Themenbereich wissensbasierter Systeme, als Teilgebiet der Künstlichen Intelligenz. Es wird nicht nur die Thematik, sondern auch die zum Verständnis notwendige Terminologie sukzessive aufgebaut.

In stark vereinfachter Weise kann man sagen, daß im wesentlichen die Vorgänge in Verbindung mit der Lösung von Problemen aus der Sicht der Informatik, hier der Künstlichen Intelligenz, beleuchtet werden. Das erste Kapitel dient hierbei der Klärung des Begriffes wissensbasiertes System.

Da bereits durch die Auswahl einer geeigneten (guten) Repräsentation häufig ein wesentlicher Schritt zur Problemlösung getan ist, werden die verschiedenen Grundformalismen zur Wissensrepräsentation, Logik, Regeln, Rahmen, Constraints und spezielle Schlußfolgerungsverfahren im zweiten Kapitel relativ ausführlich behandelt und wie bei /25/ jeweils verschiedene Ansätze vorgestellt.

Einem weiteren, derzeit im Aufschwung befindlichen Gebiet der objektorientierten Programmierung widmet sich das dritte Kapitel. Anhand der in der objektorientierten Programmierung grundlegenden Begriffe, wie Objekt, Kommunikation, Klasse, Instanz, Vererbung und Polymorphismus wird die Thematik sukzessive aufgebaut und wichtige Grundstrukturen werden aufgezeigt. Diese Darstellung wird durch die weiterführenden Konzepte wie Metaklassen, verschiedene Vererbungsformen und generische Klassen ergänzt.

Im Kapitel 4 werden die "elementaren" Problemlösungsstrategien, wie z. B. Beschreiben und Vergleichen, Teilzielbildung und Suchen vorgestellt, durch deren Anwendung zumeist als Kombination konkrete Probleme gelöst werden können /33/.

Im Kapitel 5 werden die drei nach Puppe /25/ in Expertensystemen wesentlichen Problemlösungstypen Diagnostik, Konstruktion und Simulation näher erläutert, ihre Anwendungsgebiete geschildert und Lösungsansätze aufgezeigt.

Die Problematik der Wissensakquisition sowie die verschiedenen Möglichkeiten werden im Kapitel 6 am Beispiel der Expertensysteme und anhand neuronaler Netze erläutert. Die Einsatzmöglichkeiten neuronaler Netze zur Führung technischer Prozesse wird grob skizziert.

Der Inhalt dieser Arbeit ist vor dem Hintergrund der Entwicklung eines maschinell-lernenden-Systemes zu betrachten, das (technisches) Wissen zur Führung komplexer technischer Prozesse automatisch akquiriert und validiert (vgl. /12/), und dient dabei sowohl als Einstieg, als auch als Überblick und Bestandsaufnahme zum aktuellen Stand der Forschung auf diesem Gebiet, aus der Sicht der Künstlichen Intelligenz.

In ähnlicher Weise wie in diesem Bericht, wird diese Vorgehensweise in dem vorangegangenen Bericht "Lernmodelle und Wissensverarbeitung" (/14/) und dem noch folgenden Bericht "Maschinelles Lernen - "klassische", konnektionistische und kognitive Konzepte" (/32/), einmal aus der Sicht der Lernpsychologie und zum anderen aus der Sicht des Maschinellen Lernens fortgesetzt.

Durch diese "Trilogie" wird der interdisziplinäre Charakter, den die Forschungsaktivitäten im Bereich lernender Systeme aufweisen umfassend dargestellt und die Entstehung eines neuen Konzeptes, des "schematheoretischen Ansatzes für lernfähige wissensbasierte Systeme", sowie dessen Wurzeln aufgezeigt.

1 Die Grundidee wissensbasierter Systeme

Die Grundidee wissensbasierter Systeme besteht darin, daß sie neben der rein numerischen Form (Auswertung) auch eine symbolorientierte Verarbeitung ermöglichen. Dies wird hier am Beispiel der bekannten Formel

$$U = R * I$$

erläutert.

Zur Berechnung von Spannung, Stromstärke oder Widerstand in einem einfachen elektrischen Netzwerk (bei entsprechend vorgegebenen Werten), müssen bei der konventionellen (imperativen) Programmierung alle drei numerisch zu berechnenden Fälle in der jeweiligen Umformung, d. h.

$$U = R * I \quad \text{und} \quad R = U / I \quad \text{und} \quad I = U / R$$

algorithmisch gefaßt und programmtechnisch realisiert sein.

In einem wissensbasierten System wird man eine solche Formel nur einmal, z. B. in der Form U = R * I als ein aus Termen bestehendes Faktum (Gleichung) ablegen und dem System die Umformungs- und Auswerteregeln für Gleichungen zur Gewinnung der anderen beiden Darstellungen und der Werte als Verfahren zur Verfügung stellen. Aufgrund des implementieren Verfahrens und des Faktums ist das wissensbasierte System dann in der Lage die Gleichung nach der jewiels gesuchten Größe umzuformen und zu berechnen.

Allgemein versteht man unter wissensbasierten Systemen programmtechnisch realisierte Verfahren, deren Ziel die Lösung von Aufgaben in komplexen Weltausschnitten ist (/8/).

Konventionelle Programme (imperativ, zustandsorientiert) bestehen in der Regel aus zwei unterschiedlichen Teilen, Daten und Algorithmen. Die Daten charakterisieren die Parameter des speziellen vorliegenden Problems. Algorithmen bestimmen, wie spezielle Problemarten gelöst werden.

Wissensbasierte Systeme unterscheiden sich durch ihre Organisationsform, durch die Art, wie sie Wissen enthalten und durch den Eindruck, den sie bei ihrer Ausführung erzeugen von konventionellen Programmen. Wissensbasierte Systeme simulieren die Verhaltensweise menschlicher Experten und präsentieren dem Benutzer eine menschenähnliche Fassade.

Sie sammeln die Wissensfragmente in einer Wissensdatenbank und verwenden diese Wissensdatenbank um Schlußfolgerungen bzgl. spezieller Probleme anhand eines bestimmten Schlußfolgerungsverfahrens durchzuführen.

Bzgl. dem obigen Beispiel kann gesagt werden, daß das Vorgehen bei der konventionellen Methode darin besteht, daß der Programmierer alle gewünschten Operationen vorweg erkennen und in Form von Bearbeitungsschritten einplanen und programmtechnisch realisieren muß. Die wissensbasierte Form geht über diese rein numerischen und ablaufmäßig fest programmierten Berechnungen hinaus und erlaubt eine symbolische

Manipulation der numerisch auswertbaren Formeln. Neben den reinen Berechnungsvorschriften beinhaltet das Standardreportiore an Operatoren der Sprache bzw. des Ausführungsmodells auch die symbolischen Umformungsschritte.

Die drei Dimensionen des Wissens kann man in Anlehnung an (/29/) wie folgt darstellen :

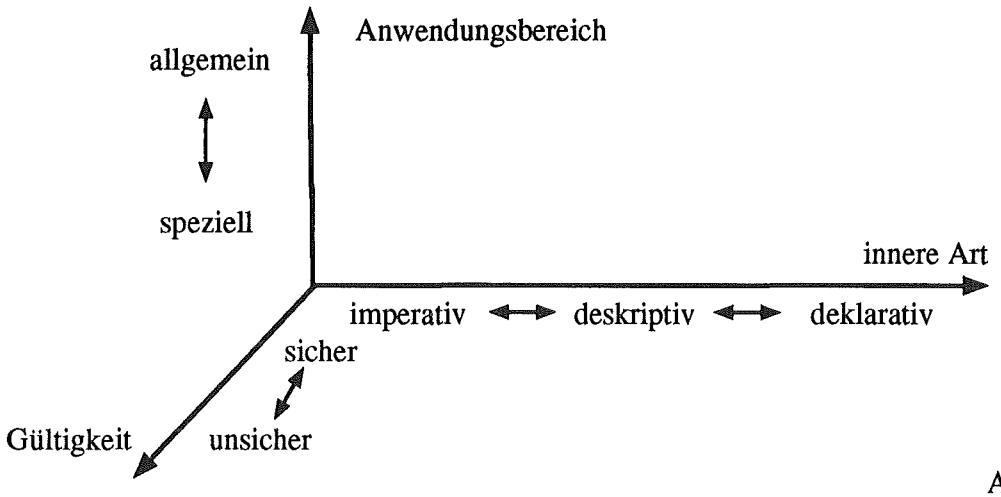

Abbildung 1

- Der Anwendungsbereich des Wissens: Dieser erstreckt sich von allgemein und auf einem breiten Bereich anwendbar, bis zu speziell und nur in einem sehr eingeschränkten Bereich anwendbar.
- Die innere Art des Wissens: Diese reicht von einer imperativen Bedeutung über Beschreibungen, bis hin zu deklarativen Angaben.
- Die Gültigkeit des Wissens: Diese liegt im Bereich zwischen sicher und unsicher.

Die Untersuchung dieser Thematik erfolgt in ähnlich strukturierter Weise, wie im ersten Bericht "Lernmodelle und Wissensverarbeitung" (/14/).

2 Wissensrepräsentationsformen

2.1 Einleitung

Charakterisiert man wissensbasierte Systeme, wie oben bereits geschehen, dadurch, daß ihr Ziel die Lösung von Aufgaben in komplexen Weltausschnitten ist, so kommt der Frage nach der geeigneten Modellierung derartiger Weltausschnitte die zentrale Rolle bei der Entwicklung solcher Systeme zu.

Ausgehend von dieser Situation sind Formalismen zur Wissensrepräsentation Mittel, mit denen formale, interne Beschreibungen der Realität konstruiert werden können. Charakteristisch für die Schwierigkeiten bei der Entwicklung wissensbasierter Systeme ist, daß die zu repräsentierende Realität - sowohl in qualitativer als auch in quantitativer Hinsicht - derart komplex ist, daß eine vollständige Repräsentation nicht erreicht werden kann.

Mit anderen Worten handelt es sich bei diesen Modellen um Approximationen der zu repräsentierenden Realität. Die Unvollständigkeit von wissensbasierten Systemen betrifft zwei Aspekte:

- Abstraktion bzgl. der Realität aufgrund von Idealisierungen. Die Auswirkungen der Abstraktion können bei sorgfältigem Vorgehen abgeschätzt werden.
- Echte Lücken, da bestimmte Fakten nicht repräsentiert werden oder nicht repräsentiert werden können. Diese Unvollständigkeit ist in ihren Auswirkungen nicht kalkulierbar und stellt daher eine Gefahr für die Adäquatheit des Systems dar (siehe auch /13/).

Prinzipiell weisen auch natürliche informationsverarbeitende Systeme, insbesondere Menschen, entsprechende Eigenschaften auf. Die Ähnlichkeit zwischen Modell und Realität und somit die Adäquatheit der Repräsentation ist nicht unabhängig von der Funktion des Modells und seinen Beziehungen zur Realität zu beurteilen (/8/).

Bei der Implementierung von Problemlösungen stellt man fest, daß es viele Gemeinsamkeiten gibt, die vor allem die Grundtechniken der Wissensrepräsentation und ihre zugeordneten Ableitungsstrategien umfassen. Diese Grundtechniken haben für die einzelnen Problemlösungstypen verschiedene Bedeutung, was die Abb. 2 verdeutlicht.

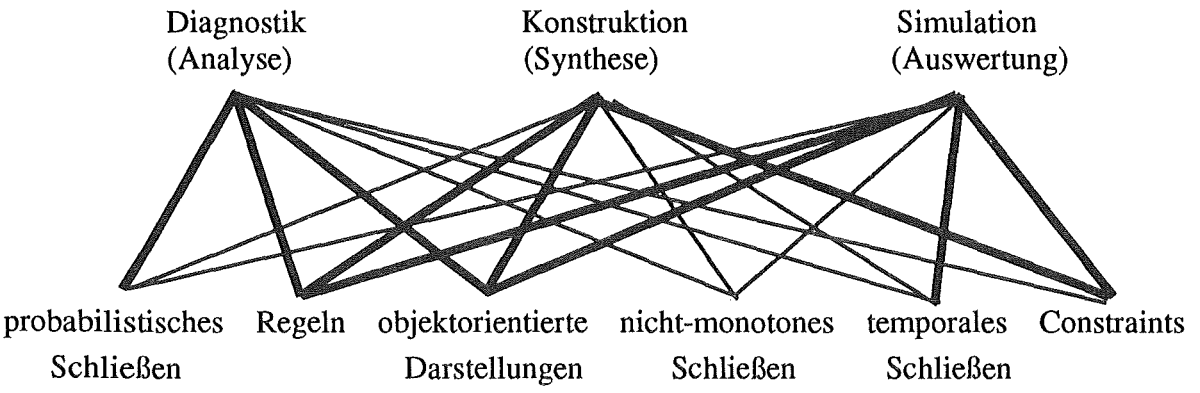

Abbildung 2

Bei lernenden Systemen, vgl. /32/, besteht eine enge Verzahnung der Wissensrepräsentationsform mit dem Lernmechanismus. Daher kann man nicht einfach zu einer Wissensbasis eine beliebige Lernkomponente hinzufügen. In diesem Zusammenhang kann man zur Auswahl einer geeigneten Wissensrepräsentationsform die folgenden Kriterien heranziehen (/26/):

- Ausdruckskraft der Repräsentation:
  Relevantes Wissen muß "einfach" ausgedrückt werden können.

- Einfachheit der Inferenzmechanismen:
  Der Berechnungsaufwand zur Durchführung vor allem häufig auftretender Inferenzen muß vertretbar sein.

- Modifizierbarkeit der Repräsentation:
  In lernenden Systemen muß das neu erlernte Wissen konsistent in die bestehende Wissensbasis integriert und altes Wissen möglicherweise revidiert werden können. Folglich muß die Wissensrepräsentationsform ein dynamisches Verhalten unterstützen.

- Erweiterbarkeit der Repräsentation:
  Die Mächtigkeit des Wissensrepräsentationsformalismus reflektiert seine generative Struktur, d. h. das, was das lernende System potentiell zu repräsentieren und damit zu lernen in der Lage ist: Der Formalismus legt den potentiell erfaßbaren Kandidaten- und Beschreibungsraum fest.

  Diese Einschränkung des von einem lernenden System Erlernbaren und der dadurch bewirkten Gerichtetheit auf das Erlernen nur ganz bestimmter, in der Repräsentationssprache ausdrückbarer Sachverhalte, wird häufig auch als **representational bias** bezeichnet.

  Dem Problem eingeschränkter Ausdrucksmächtigkeit eines Wissensrepräsentationsformalismus versucht man durch die Erweiterung des Formalismus und durch Einführung neuer Begriffe zu begegnen. Um jedoch eine Repräsentationssprache automatisch erweitern zu können, muß zunächst einmal ein Mangel an Ausdrucksmächtigkeit als solcher vom System erkannt und danach eine geeignete Erweiterung durchgeführt werden.

  Vor diesem Hintergrund muß auch ein Wechsel des Wissensrepräsentationsformalismus während des Lernvorganges oder die Darstellung des Wissens in multiplen, gleichzeitig vorhandenen Repräsentationen in Betracht gezogen werden.

Im folgenden werden die einzelnen Grundtechniken der Wissensrepräsentation in wissensbasierten Systemen näher erläutert.

2.2 Elementare Wissensrepräsentationsformen

2.2.1 Merkmalsvektoren

Merkmalsvektoren (feature vectors) beschreiben Objekte durch (zumeist numerische) Tupel einer festen und geordneten Anzahl von Merkmalsausprägungen. Die Merkmale können nur Werte aus einem bestimmten Wertebereich annehmen, der wiederum diskret oder kontinuierlich sein kann.

In Merkmalsvektoren werden, im Unterschied zu symbolischen Repräsentationen, Informationen lediglich kodiert. Das Verständnis eines Merkmalsvektors setzt daher eine vorher getroffene Vereinbarung voraus; diese Vereinbarung kann für verschiedene Teile eines Programmes unterschiedlich sein

Da die Vektoren feste Länge haben und die möglichen Wertebereiche vordefiniert sind, lassen sie sich einfach durch Konkatenation ihrer Bitrepräsentationen in Bitstrings überführen. In einem anderen Zusammenhang können diese Strings aber etwas ganz anderes kodieren; eine feste sprachliche Verabredung für die Bedeutung von Bitstrings existiert nicht, und deshalb kann man auch keine allgemeinen Inferenzmechanismen formulieren.

- Vorteile solcher Repräsentationen:
  - Ihre leichte Manipulierbarkeit.
  - Sie besitzen eine einfache Struktur.
- Nachteile:
  - Bescheidene Mächtigkeit der Repräsentation.
  - Die Kodierung muß stets neu vorgenommen werden.
  - Die strukturelle Einfachheit der Merkmalsvektoren bedingt, daß komplexe Strukturen nicht adäquat wiedergegeben werden können.

Für manche Anwendungen, wie etwa digitalisierte Bilder oder Geräusche, sind Bitstrings geradezu prädestiniert, da dort die Informationen bereits in Bitfolgen vorliegen (vgl. /26/).

2.2.2 Parameter in algebraischen Ausdrücken

Parameter in algebraischen Ausdrücken sind normalerweise Zahlen oder (gewichtete) Koeffizienten in Formeln bekannter Struktur und werden vorzugsweise zur Repräsentation numerischen Wissens verwendet.

Lernen in diesem Kontext kann als Finden und Optimieren numerischer Parameter und Koeffizienten zur Erfüllung eines Gütekriteriums (z. B. in der adaptiven Regelung) gesehen werden (vgl. /32/).

Werden nicht nur Parameter in bereits bekannten, strukturell festen algebraischen Ausdrücken, sondern auch die Struktur der algebraischen Ausdrücke als Formeln von Funktionen selbst erlernt, so spiegelt sich hierin Wissen über die Abhängigkeit zwischen Eingangsgrössen und Funktionswerten wider.

Beispiele für parametrisches Lernen sind das Optimieren von Regelkreisen sowie das Optimieren und die Adaption von Trajektorien in der Robotik. Eines der bekanntesten Beispiele ist Samuels CHECKERS Programm, das algebraische Ausdrücke und Parameterjustierung zur Evaluation von Brettkonfigurationen beim Damespiel verwendet (vgl. /26/).

2.3 Logik

Um das Ziel von Expertensystemen, die Nachbildung des Spezialwissens und der Schlußfolgerungsfähigkeit qualifizierter Fachleute zu erreichen, muß man den intuitiven Schlußfolgerungsbegriff formalisieren.

Dabei sollen subjektive Assoziationen, die mit einzelnen Begriffen verbunden sind, keine Rolle spielen, sondern alles relevante Wissen muß explizit formuliert werden. Ableitungen dürfen nur nach definierten Regeln vorgenommen werden.

2.3.1 Klassische Logik

2.3.1.1 Die Aussagen- und die Prädikatenlogik

Aus oben beschriebener Zielsetzung heraus entwickelte sich die Idee eines Kalküls, in dem man Objekte oder Zustände der realen Welt durch Aussagen des Kalküls beschreiben und daraus mittels allgemeingültiger Ableitungsregeln andere Aussagen herleiten kann, die dann wieder auf Objekte oder Zustände der Welt bezogen werden. Vorgegebene Aussagen des Kalküls heißen **Axiome**, **Fakten** oder **Annahmen**, die abgeleiteten Aussagen **Theoreme** oder **Schlußfolgerungen**.

Ein einfacher Kalkül, dessen Grundbegriffe nur aus einfachen Aussagen besteht, basiert auf der **Aussagenlogik.** Verknüpft werden diese Aussagen durch die **Verknüpfungsoperatoren** $\wedge$, $\vee$, $\neg$, $\rightarrow$. Die wichtigste **Ableitungsregel** ist der **Modus ponens**; dieser erlaubt es aus der Gültigkeit der beiden Aussagen A und $A \rightarrow B$ die Aussage B herzuleiten.

Mit der Aussagenlogik lassen sich nur konkrete Aussagen über die Welt formulieren. Zur Beschreibung allgemeiner Gesetzmäßigkeiten ist sie unzureichend, da es z. B. keine Variablen gibt. Hierfür wurde die Aussagenlogik zur **Prädikatenlogik erster Ordnung** erweitert.

In der Prädikatenlogik erster Ordnung werden statt einfacher Aussagen Prädikate verwendet, wobei ein Prädikat eine Beziehung zwischen zwei oder mehreren Variablen oder Konstanten herstellt. Als Argumente der Prädikate werden Individuen zugelassen.

Individuen können außer durch Konstanten oder Variablen auch durch **Funktionsausdrücke** dargestellt werden, die man sich als komplexe Art der Referenz von Objekten vorstellen kann. Funktionen verwendet man im mathematischen Sinne. Sie besitzen eines oder mehrere Argumente, die in direkter Relation zum Funktionsnamen stehen. Eine Funktion darf als Argument eines Prädikates auftreten, aber nicht umgekehrt (/30/).

Existenz und Allaussagen über Individuen werden durch die **Quantoren** $\forall$ und $\exists$ beschrieben. Weiterhin kommen beim Übergang von der Aussagen- zur Prädikatenlogik zwei Axiome (das Spezialisierungs- und das Quantoren-Shift-Axiom) zu den aussagenlogischen Axiomen und eine Regel zum Modus Ponens (die Generalisierungsregel) hinzu.

Stede /30/ bezeichnet die Problemlösung mittels Prädikatenlogik als Anlegen von Listen, erzeugen eines Suchbaumes und Verwenden einer "intelligenter" Suchstrategie.

Im Zusammenhang mit der "intelligenten" Suchstrategie sollte man jedoch auch erwähnen, daß nicht zuletzt aufgrund der Unstrukturiertheit der Formelmengen die Beweise zumeist lang und unübersichtlich werden und automatische Beweiser daher nicht effizient arbeiten (/28/).

Da unsicheres Wissen durch Aussagen charakterisiert ist, die solange als wahr angenommen werden, wie nicht bewiesen ist, daß sie nicht wahr sind, bzw. ihr Gegenteil nicht bewiesen ist, ist es nicht möglich in der Prädikatenlogik erster Ordnung unsicheres Wissen darzustellen (Axiome sind wahr, und können nicht revidiert werden). Da die Ableitungsregeln als korrekt bewiesen sind und daher am abgeleiteten Wissen nicht gezweifelt werden kann (/28/), ist es auch nicht möglich, unsicheres Wissen abzuleiten.

Während in der Prädikatenlogik erster Ordnung All- und Existenzaussagen nur über Individuen getroffen werden können, sind in der **Prädikatenlogik zweiter Ordnung** solche Aussagen auch über Mengen bzw. Eigenschaften von Individuen zulässig; d. h. in der Prädikatenlogik zweiter Ordnung können Variablen auch Prädikate darstellen und es kann auch über solche Prädikatenvariablen quantifiziert werden. Dadurch wird die Bildung selbstreferentieller Aussagen möglich.

Wegen ihrer mangelnden Implementierbarkeit wird die Prädikatenlogik zweiter Ordnung in der praktischen Informatik nur selten benutzt.

2.3.1.1.1 PROLOG

Bei der Wissensrepräsentationssprache PROLOG handelt es sich um ein Derivat der Prädikatenlogik erster Ordnug, bei dem die Anzahl der möglichen logischen Schlußfolgerungen verringert wird, um dadurch einer möglichen kombinatorischen Explosion entgegenzuwirken.

PROLOG basiert auf dem **Hornklauselkalkül**, einem weniger mächtigen, aber effizienter interpretierbaren Kalkül als dem Resolutionskalkül. Hierzu werden spezielle Klauseln, sogenannte Hornklauseln, zu Regeln umgeformt. Der Ableitungsmechanismus entspricht der Rückwärtsverkettung von Regeln. Die Hauptarbeit des PROLOG-Interpreters ist die **Unifikation**, d. h. der Vorgang Substitutionen zu finden, die zwei Literale mit Variablen gleichmachen.

Das Kontrollwissen von PROLOG ist in der Programmsequenz kodiert (Rechtsrekursion).

Obwohl sich die bereits erwähnte Kritik an der Prädikatenlogik erster Ordnung weitgehend auf PROLOG übertragen läßt und es für größere Expertensystementwicklungen ein zu niedriges Abstraktionsniveau besitzt, ist es als Regelsprache für kleinere Expertensysteme sehr populär, da es einen Regelinterpreter mit Rückwärtsverkettung und Unifikation zur Verfügung stellt.

2.3.1.2 Eigenschaften von Kalkülen

Alle Wissensrepräsentationsformalismen kann man verbunden mit ihren Ableitungsstrategien als Kalküle auffassen. Die Nützlichkeit eines Kalküls ergibt sich aus seinen inneren Eigenschaften und daraus, wie gut die Welt, d. h. der Problembereich, in dem Kalkül abgebildet werden kann.

Unverzichtbar ist dabei die Forderung der **Korrektheit** des Kalküls, diese ist erfüllt, wenn alle **syntaktisch** (d. h. im Kalkül) herleitbaren Schlußfolgerungen auch **semantisch** (d. h. in der Welt) folgern. Die **Mächtigkeit** (Ausdrucksstärke) bestimmt, welche Aussagen über die Welt überhaupt repräsentierbar sind. Ein Kalkül ist **vollständig**, wenn alle Schlußfolgerungen, die semantisch gelten, auch syntaktisch herleitbar sind, und **entscheidbar**, wenn für eine beliebige Aussage entschieden werden kann, ob sie aus den Axiomen folgt oder nicht.

Für Expertensysteme sehr wichtige Eigenschaften sind die **Adäquatheit** und die **Effizienz** eines Kalküls, die angeben, wie "natürlich" (d. h. einfach und elegant) Problembereiche beschrieben und wie effizient Problemlösungen hergeleitet werden können. Effizienz ist häufig nur für das Herleiten der im Anwendungskontext besonders relevanten Schlußfolgerungen erreichbar.

Die **lokale Konfluenz** ist die Eigenschaft, die gegeben ist, wenn die Anwendung irgendeines Abbildungsschrittes das Herleiten eines Theorems nicht verhindern kann. Weiterhin heißt eine Menge von Aussagen **konsistent**, wenn sich die einzelnen Aussagen nicht widersprechen.

Wünschenswert wäre es, wenn eine Wissensrepräsentationsform alle obigen Eigenschaften in maximalem Umfang erfüllen würde. Dies ist jedoch nicht möglich, so ist z. B. die relativ ausdrucksstarke Prädikatenlogik zweiter Ordnung weder vollständig noch entscheidbar, die weniger ausdrucksstarke Prädikatenlogik erster Ordnung zwar auch nicht entscheidbar aber immerhin vollständig und die Aussagenlogik sowohl entscheidbar als auch vollständig.

Eine gute Wissensrepräsentation zeichnet sich durch einen optimalen Kompromiß für ihren Problembereich aus.

Als Begründung für die Nicht-Entscheidbarkeit der Prädikatenlogik erster Ordnung betrachte man das **Postsche Korrespondenzproblem** über einem Alphabet mit mehr als zwei Elementen. Dieses Korrespondenzproblem ist nicht entscheidbar, da man für Relationen, die eine Lösung besitzen diese zwar durch systematisches Ausprobieren in endlicher Zeit finden kann; für diejenigen Relationen, die keine Lösung besitzen, weiß man jedoch a priori nicht, wie lange man suchen muß, um sicher zu sein, daß es tatsächlich keine Lösung gibt.

Die Prädikatenlogik erster Ordnung hat sich seit der Einführung des **Resolutionskalküls** (dieser arbeitet mit Klauseln und der Resolutionsregel als einziger Ableitungsregel) besonders bei der automatischen Erzeugung mathematischer Beweise bewährt.

Der Vorteil der Prädikatenlogik erster Ordnung besteht darin, daß für sie eine große Anzahl gesicherter Ergebnisse vorliegt und daß Inferenzsysteme vorliegen.

Für die Wissensrepräsentation in Expertensystemen hat die Prädikatenlogik erster Ordnung beträchtliche Nachteile hinsichtlich Adäquatheit, Effizienz und der oben bereits angesprochenen Nicht-Entscheidbarkeit. Da in vielen Anwendungsbereichen das Wissen bzw. die Daten unsicher, unvollständig oder zeitabhängig sind, kann eine Darstellung in Prädikatenlogik erster Ordnung gar nicht oder nur schlecht erfolgen. Für große Wissensbasen benötigt man adäquate, im Prädikatenkalkül erster Ordnung nicht vorhandene Strukturierungsmittel wie Hierarchien oder Kontexte.

2.3.2 Nicht-klassische Logik

2.3.2.1 Einleitung und Überblick

Neben der Aussagenlogik und der Prädikatenlogik ist die **Modallogik** der am weitestgehenden untersuchte Zweig der Mathematischen Logik. In der Modallogik werden hauptsächlich die Modalitäten der Notwendigkeit und der Möglichkeit studiert. Dabei sieht man eine Aussage A nicht mehr a priori als wahr oder falsch an. Der Wahrheitswert erfährt vielmehr eine Relativierung durch die Umstände, unter denen A in Rede steht.

Beispiel: Die Aussage A = "Es regnet" ist vom Standpunkt der Aussagenlogik entweder wahr oder falsch. Aus Erfahrung weiß man jedoch, daß sie weder notwendig wahr noch notwendig falsch ist; vielmehr gibt es Umstände (Zeit, Ort) oder man kann sich zumindest solche vorstellen, unter denen sie wahr bzw. falsch ist. In der Modallogik wird daher die Aussage "Es ist möglich, daß nicht A" und auch die Aussage "Es ist möglich, daß A" als wahr angesehen.

Als eine Variante der Modallogik wurden die **Temporären Logiken** (vgl. auch Abschnitt 2.8.3) entwickelt. Hierbei handelt es sich um ein vielversprechendes Werkzeug für die Spezifikation und Verifikation insbesondere paralleler Programme /9/.

In den sogenannten **nichtmonotonen Logiken**, in denen nichtmonotones Schließen (vgl. auch Abschnitt 2.8.2) und die Revision von bereits durchgeführten Schlüssen möglich ist, versucht man Aussagen zu formalisieren, die als wahr bzw. falsch gelten, solange nicht das Gegenteil bewiesen ist oder solange sie mit den übrigen Aussagen konsistent sind /28/.

Die von Lotfi A. Zadeh mitte der sechziger Jahre entwickelte Theorie der Fuzzy Mengen, als Verallgemeinerung der klassischen Mengentheorie, wurde zum Ausgangspunkt einer Reihe sowohl theoretischer, als auch praktischer Anwendungen. Die in diesem Zusammenhang als relevant einzustufenden Teilgebiete sind:

- Die Fuzzy Logik,
- Fuzzy Control und
- das Fuzzy Reasoning.

In jedem dieser Teilgebiete existieren mittlerweile Anwendungen, welche die Stärken der Fuzzy Mengen zum Teil eindrucksvoll demonstrieren.

Stellvertretend seien hier nur der Einsatz der Fuzzy Control zur Steuerung einer Zementbrennanlage in Dänemark, die Regelung der Vorgänge des Anfahrens, Bremsens und Beschleunigens bei der U-Bahn in Sendai (Japan) (für diese und auch weitere Anwendungen siehe /6/) und den Einsatz der Fuzzy Logik zur Kontrolle des Verbrennungsprozesses in Müllverbrennungsanlagen /24/ genannt. Viele industrielle Prozesse werden - anstelle oder zusätzlich zu einem menschlichen Bediener automatisch überwacht oder eignen sich zumindest dazu.

Die Entwicklung eines effektiven automatischen Prozeßüberwachungssystems erfordert in der Regel ein präzises mathematisches Modell des Prozesses oder des betreffenden Systems. Bie vielen komplexen Prozessen ist die Konstruktion eines solchen Modells aber entweder sehr schwierig oder sogar unmöglich. Dies ist beispielsweise durch aufretende Nicht-Lineartitäten oder zeitliche Variabilität (die Dynamik) des Prozesses oder durch eine nicht ausreichende Qualität der Messungen begründet.

In solchen Fällen kann der menschliche Bediener den technischen Prozeß besser als konventionelle Steuerungs- und Regelungssysteme überwachen und regeln, da er sich hierzu einer vagen (unpräzisen) aber stabilen heuristischen Vorgehensweise bedient.

Als Alternative und häufig auch im Gegensatz zu einem präzisen Systemmodell kann eine vage Beschreibung des technischen Prozesses oder der Art und Weise der Regelungsvorgänge häufig vom menschlichen Experten, dem Bediener formuliert werden. Eine solche sprachliche Beschreibung, häufig auch als Daumenregeln bezeichnet, bedient sich in der Regel vager oder Fuzzy Konzepten, was die Anwendung der Fuzzy Logik nahelegt.

Das gleiche Problem, daß entweder das Wissen von Experten nicht durch exakte Regeln formulierbar ist bzw. den Experten eine solche Formalisierung nicht gelingt, da es sich um

implizites Wissen handelt, tritt bei der Wissensakquisition in Expertensystemen auf (vgl. Kapitel 6). Der wesentliche Unterschied in der bisherigen Darstellung zwischen den Regeln zur Prozeßführung und den Regeln in Expertensystemen liegt in der Regelverarbeitung.

Während in der Fuzzy Control die Regelverarbeitung ausschließlich auf fuzzytheoretischen Ansätzen basiert, die ohne einen Übergang zu Methoden des "Approximate Reasoning" und des "plausiblen Schließens" nur eine einstufige Regelverarbeitung gestatten, kommen in Expertensystemen mächtige Schlußfolgerungsmechanismen, zumeist über mehrere Ableitungsschritte, zum tragen.

Der "naive" Ansatz den Schlußfolgerungsmechanismus der Fuzzy Logik ohne Modifikation auf mehrstufige Ableitungen zu Übertragen muß mißlingen, da die hierfür notwendige Voraussetzung, die Unabhängigkeit der einzelnen Fuzzy Regeln, zumeist nicht gegeben ist.

2.3.2.2 Fuzzy Logik

Grundsätzlich unterscheidet man die folgenden drei Arten von Unschärfe:

- Unschärfe aufgrund der Unvorhersagbarkeit zufälliger Ereignisse

  → Wahrscheinlichkeitstheorie

- Unschärfe bedingt durch die gegebene Information, der zwar eine exakte Begriffsbildung zugrundeliegt, die jedoch in der Praxis mehr oder weniger vage gehandhabt werden, da dies aus technischen oder ökonomischen Gründen nicht anders möglich ist

  → Fuzzy Logik.

- Die Unschärfe liegt in der subjektiven Einschätzung bestimmter sprachlicher Formulierungen begründet

  → Fuzzy Logik.

Ein wichtiger Gesichtspunkt von informativer bzw. subjektiver Unschärfe besteht darin, daß diese ein Mittel zur Bewältigung von Komplexität darstellen; d. h. sie können zur Reduktion der benötigten Informationen dienen.

Mit anderen Worten bedeutet dies, daß der Informationsverlust, der zur Reduktion der Komplexität des jeweiligen Systems auf einen handhabbaren Level notwendig ist, wird in Form von Unsicherheiten ausgedrückt. Das Konzept der Unsicherheit hängt daher sowohl mit der Komplexität, als auch mit der Information zusammen /16/.

2.3.2.3 Beispiel der Wissensverarbeitung mit Fuzzy Logik

Vorgehensweise:

1. Durch die Verbindung des sprachlichen Begriffes einer Meßgröße (Stellgröße oder Regelgröße) und deren Wertebereich (Beobachtungsbereich) mit Termen wie z. B. *sehr hoch, hoch, mittel, niedrig, sehr niedrig* und Zugehörigkeitsfunktionen zu den jeweiligen "Teilbereichen" entsteht eine **linguistische Variable**. Die Aufgabe einer linguistischen Variablen besteht darin, eine Verbindung zwischen sprachlicher Beschreibung und numerischer Darstellung der Werteintervalle von Meßgrößen herzustellen.

   Bsp.: Brennkammertemperatur Wertebereich 800 - 1000°C

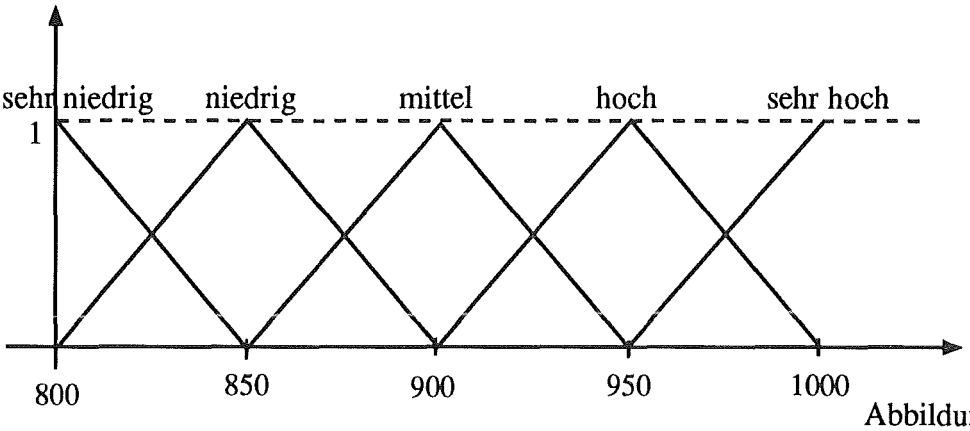

Abbildung 3

   Dieser Vorgang wird auch sehr häufig **Fuzzyfizierung** genannt (vgl. /2/).

2. Die unscharfen Informationen werden in der Fuzzy Logik in Form von (Fuzzy) Produktionsregeln verarbeitet:

   1. Die Verknüpfung verschiedener Aussagen erfolgt durch die folgenden Operatoren:

      UND: $A \wedge B := \min$ (Wahrheitswert(A), Wahrheitswert(B)),
      ODER: $A \vee B := \max$ (Wahrheitswert(A) Wahrheitswert(B)),
      NEGATION: $\neg A := 1$ - Wahrheitswert(A),
      wobei A, B Fuzzylogische Aussagen sind.

      Der Wahrheitswert einer Aussage wird durch die Zugehörigkeitsfunktion anhand der entsprechenden linguistischen Variablen ermittelt.

   2. Die Verarbeitung der Produktionsregeln erfolgt dann dadurch, daß der Aussage im Konsequenzteil der sich aus den Aussagen im Antezendenzteil resultierende Erfüllungsgrad der Regelvorbedingung zugeordnet wird.

      D. h. man geht davon aus, daß die Schlußfolgerung einer Regel immer zum gleichen Grad erfüllt ist, wie ihre Vorbedingung. Ein weiterführendes Konzept hierzu bildet das Approximate Reasoning. Dort werden die Regeln mit "Glaubensmaßen" versehen.

3. Der somit berechnete Wert (je einer pro ausgewerteter Regel) dient zur Begrenzung der entsprechenden Terme der linguistischen Variablen im Konsequenzteil der Regeln. Als Resultat der im folgenden beschriebenen Vorgehensweisen entsteht (bei einer linguistischen Variablen im Konsequenzteil) eine Fläche (allgemeiner ein |Anzahl der Variablen im Konsequenzteil| + 1 dimensionaler Raum). Hierzu gibt es mehrere alternative Vorgehensweisen. Stellvertretend seien hier nur die folgenden Beiden kurz dargestellt:

   (1) Die MAX-PROD-Inferenz:
   Die Werte der entsprechenden Zugehörigkeitsfunktion werden mit dem berechneten Erfüllungsgrad der Regelvorbedingung multipliziert, d. h. die Zugehörigkeitsfunktion wird gestaucht, da diese Werte immer aus dem Intervall [0, 1] stammen.

   (2) Die MAX-MIN-Inferenz:
   Die entsprechende Zugehörigkeitsfunktion wird ab dem Erfüllungsgrad der Vorbedingung abgeschnitten.

4. Die am häufigsten verwendete Methode um nun wieder von der somit erhaltenen unscharfen mehrdimensionalen "Menge" für die Terme zu Wertebelegungen der linguistischen Variablen zu gelangen, bildet man den Flächenschwerpunkt und dessen Projektion auf den Wertevorrat.

   Somit wurden die Werte für die einzelnen linguistischen Variablen bestimmt, die dem Erfüllungsgrad der Regelvorbedingung entsprechen. Bei dieser Vorgehensweise gilt es natürlich zu bedenken, daß die Schwerpunktbildung (und die anschließende Projektion) einen Informationsverlust darstellen.

   Dierser Vorgang wird häufig **Defuzzyfizierung** genannt (vgl. /2/).

Die oben beschriebene Vorgehensweise der Verarbeitung von unscharfem Wissen wird **Fuzzy Logik** genannt.

2.4 Regeln

Regeln dienen dazu natürlichsprachliche Wenn <Vorbedingung>-Dann <Aktion>-Sätze darzustellen. Die Vorbedingung beschreibt eine Situation, in der die Aktion ausgeführt werden soll.

Da Experten ihr Wissen häufig in Form von Regeln formulieren, sind Regeln die verbreitetste Wissensrepräsentationsform in Expertensystemen. Die Aufteilung des Wissens in vie-

le kleine eigenständige "Wissensstücke" macht eine Wissensbasis modular und damit leicht veränderbar.

Besonders attraktiv ist die Anpassungsfähigkeit des Grundformalismus an anwendungsspezifische Kompromisse zwischen Mächtigkeit und Effizienz, indem man z. B. Regeln zur Darstellung von unsicherem oder unvollständigem Wissen um Unsicherheitsangaben oder Ausnahmen erweitert.

Ein **regelbasiertes System**, auch **Produktionssystem** genannt, besteht aus einer **Datenbasis**, die die gültigen Fakten enthält, den **Regeln** zur Herleitung neuer Fakten und dem **Regelinterpreter** zur Steuerung des Herleitungsprozesses.

Für die Ableitbarkeit der Regeln gibt es zwei prinzipielle Alternativen:

- Vorwärtsverkettung (Forward-Reasoning):
  Ausgehend von einer vorhandenen Datenbasis wird aus den Regeln, deren Vorbedingung durch die Datenbasis erfüllt ist, mit Hilfe einer **Konfliktlösungsstrategie** eine Regel ausgesucht, ihr Aktionsteil ausgeführt (d. h. die Regel "feuert") und damit die Datenbasis geändert.

  Dieser Prozeß wird solange wiederholt, bis keine Regel mehr anwendbar ist.

  Ein Beispiel für eine Konfliktlösungsstrategie ist der **RETE-Algorithmus,** der die Regeln systematisch anordnet.

- Rückwärtsverkettung (Backward-Reasoning):
  Ausgehend von einem Ziel werden nur die Regeln überprüft, deren Aktionsteil das Ziel enthält. Falls Parameter (Fakten) der Vorbedingung unbekannt sind, werden sie vom Benutzer erfragt oder mit anderen Regeln hergeleitet. Hierfür verfährt man nach zu den Konfliktlösungsstrategien der Vorwärtsverkettung analogen Vorschriften.

  Die Effizienz eines rückwärtsverketteten Regelinterpreters wird durch die Formulierung des Ziels bestimmt: Je präziser das Ziel, desto kleiner der Suchbaum der zu überprüfenden Regeln und zu stellenden Fragen.

Der Hauptnachteil von Regelsystemen besteht in dem schwer verständlichen Kontrollfluß. Wegen der Unübersichtlichkeit großer Regelmengen ("Regelspaghetti") sollten Regeln durch Typisierung, durch Kontexte und vor allem durch Zuordnung zu Objekten (siehe Abschnit 2.5) strukturiert werden.

2.5 Rahmen

2.5.1 Semantische Netze

Nach /30/ stellten semantische Netze eine ideale Basis zur Darstellung von Wissen dar, da sie Objekte denotieren und die Beziehungen zwischen ihnen beschreiben.

Ein semantisches Netz setzt sich aus Kanten und Knoten zusammen, wobei die Knoten für Objekte stehen, die mittels Kanten in Verbindung gebracht werden. Die Kanten werden mit einem Label, als Auskunft über den Zusammenhang zwischen den einzelnen Objekten (daher der Name "semantisch"), und mit einem Pfeil für die Bedeutungsrichtung versehen; dadurch entsteht ein gerichteter Graph.

Durch Verwendung verschiedener Knoten- und Labeltypen kann das Anwendungsspektrum semantischer Netze deutlich erhöht werden (vgl. auch /14/).

Eine Schwiergkeit bei der Arbeit mit semantischen Netzen liegt in der möglichen Verwirrung zwischen Objekt-Klassen, die auch als Konzepte bezeichnet werden und den **Instanzen** solcher Klassen, den einzelnen Individuen.

Ein Vorteil semantischer Netze besteht darin, daß sie von der Struktur her "nur" gerichtete Graphen sind und sich dadurch mit einem Rechner gut bearbeiten lassen, da dies eine Datenstruktur ist, die von vielen Programmiersprachen unterstützt wird.

Ein Nachteil semantischer Netze liegt in derem "völligen" versagen, wenn der Faktor Zeit mit aufgenommen wird; chronologische Abläufe können mit semantischen Netzen nicht bearbeitet werden. Sie gleichen immer nur zeitlichen Momentaufnahmen.

Betrachtet man speziell semantische Netze, um ein Modell für das Verständnis natürlicher Sprache zu konstruieren, so stellt man fest, daß die Bedeutung von Worten nur dann in korrekter Weise miteinander verbunden werden kann, wenn die Worte korrekt mit der "Umwelt" verbunden sind. Dies impliziert jedoch zumeist einen zweiten Darstellungslevel, der für die Verbindung mit der Umwelt sorgt (vgl. /10/).

2.5.2 Objekte/Frames

Objekte dienen dazu, natürlichsprachliche Begriffe zu repräsentieren. Dabei geht man davon aus, daß die Bedeutung eines Begriffes am besten dadurch erklärt wird, indem man vorgibt, in welchen Beziehungen er zu anderen Begriffen steht, ähnlich wie es in einem Wörterbuch gemacht wird (vgl. /5/).

Frames sind Datenstrukturen zur Darstellung einer Menge von Fakten und Prozeduren, die Objekte jeglicher Art, Situationen und Zustände und deren Manipulation beschreiben. Wissen soll durch sie strukturiert und in zusammenhängende aber unterscheidbare Einheiten zerlegt werden können. Situationen von denen man annimmt, daß der Mensch sie als "Fra-

mes" im Gedächtnis ablegt, sollen durch Eigenschaften, genannt Slots, die mit Standardwerten belegt sind, beschrieben werden (/28/).

In einfachen Regelsystemen steckt das eigentliche "Wissen" ausschließlich in den Regeln, während die Datenbasis eine unstrukturierte und passive Menge von Fakten ist. Gegenstand dieses Abschnittes sind Formalismen, mit denen eine Menge von Fakten besser strukturiert, ökonomischer gespeichert und mit "Basiswissen" über ihre Verwendung ausgestattet werden kann. Eine erste Strukturierung erreicht man durch die Zusammenfassung aller Aussagen über ein Objekt in einer Datenstruktur, wie z. B. Records in PASCAL oder Propertylisten in LISP.

Die Kernideen objektorientierter (framebasierter) Wissensrepräsentationen sind Vererbungshierarchien, durch zugeordnete Prozeduren erfolgt die Zusammenfassung von deklarativem und prozeduralem Wissen und Erwartungswerten. Diese Grundkonzepte werden im folgenden näher erläutert:

- Vererbungshierarchien:
  Sie dienen zur ökonomischen Datenhaltung; anstatt bei jedem Objekt alle seine Eigenschaften abzuspeichern, strukturiert man die Objekte in einer Hierarchie und speichert nur individuelle Eigenschaften beim Objekt selbst ab, während allgemeine Eigenschaften den Vorgängern des Objektes in der Hierarchie zugeordnet und an alle Nachfolger "vererbt" werden.

  Bei der Abfrage von Eigenschaften eines Objektes wird der Wert zunächst im Objekt selbst gesucht, dann in seinem direkten Vorgänger in der Hierarchie etc.. Zur Flexibilitätssteigerung werden in objektorientierten Systemen häufig Vererbungshierarchien verwendet, bei denen ein Objekt die Eigenschaften mehrerer Vorgänger erbt, und in denen die selektive Unterdrückung der Vererbung einzelner Eigenschaften möglich ist.

- Zugeordnete Prozeduren:
  Aufgrund der zugeordneten Prozeduren wird der Frame-Ansatz letztlich zu einer echten Erweiterung der klassischen Logik. Durch sie wird der bisher fehlende prozedurale Aspekt ins Spiel gebracht.

  Zugeordnete Prozeduren sind kleine Programme, die einer Eigenschaft des Objektes zugeordnet sind und bei einem Lese- oder Schreibzugriff auf deren Wert ausgeführt werden.

  Durch zugeordnete Prozeduren kann ausgedrückt werden, daß eine Eigenschaft normal ist oder erwartet wird, daß es sich empfiehlt, dies oder jenes anzunehmen.

  Sie können zur ökonomischen Datenhaltung verwendet werden, indem eine zugeordnete Prozedur aus vorhandenen Parametern weitere berechnet (z. B. das Alter als Differenz aus dem aktuellen und dem Geburtsdatum).

  Außerdem können mit zugeordneten Prozeduren Wertänderungen überwacht werden. Solche überwachten Werte heißen "**active values**".

Ähnlich wie Werte können auch zugeordnete Prozeduren in der Objekthierarchie vererbt werden.

- Erwartungswerte:
  Dies sind Vorbelegungen von Werten, die meistens, aber nicht immer stimmen und daher durch korrekte Informationen überschrieben werden können.

  Der Umgang mit Erwartungswerten erfordert jedoch die Fähigkeit, Schlußfolgerungen mit allen Konsequenzen zurückziehen zu können, was nur sehr aufwendig zu realisieren ist.

In der **Frame Representation Language** (FRL) wurde der Sprachumfang von LISP durch eine Menge von Funktionen, mit denen man strukturierte Objekte definieren, ausführen und ändern sowie Einträge abfragen kann, erweitert. In FRL heißen die Objekte **Frames** und ihre Eigenschaften **Slots.** Um beispielsweise zwischen dem Erwartungswert und dem tatsächlichen Wert eines Slots unterscheiden zu können, besitzt ein Slot verschiedene **Facetten**, die jeweils einen Wert haben können. Für den Aufbau von Vererbungshierarchien gibt es in FRL einen besonderen, vordefinierten Slot für jeden Frame, der **A Kind Of** (AKO) heißt.

In anderen Frame-Sprachen wird häufig zwischen generischen Objekten (Objektklasssen) und Individuen, die Instanzen generischer Objekte sind, unterschieden. Daraus ergibt sich eine Differenzierung des AKO-Slots von FRL in echte Instanzen (Individuum → generisches Konzept, z. B. Donald Duck ist eine Ente) und in Teilmengen (generisches Konzept → generisches Konzept, z. B. ein Pinguin ist ein Vogel).

In FRL sind die Frames keine passiven Daten mehr, sondern aktive Objekte, die selbständig Berechnungen durchführen können. Damit wird im Vergleich zu konventionellen Datenbanken eine sehr ökonomische Datenhaltung ermöglicht, da nicht alle Daten explizit abgespeichert werden müssen, sondern oft durch Vererbungshierarchien und zugeordnete Prozeduren hergeleitet werden können. Da es weitgehend vermieden werden kann, Daten redundent abzuspeichern, wird die Konsistenzhaltung der Datenbasis vereinfacht, was sich zusätzlich vorteilhaft auswirkt.

Bei der objektorientierten Programmiersprache SMALLTALK besteht ein Programm aus einer Menge von Objekten, die sich Nachrichten zuschicken und diese durch Ausführen von Methoden beantworten. Methoden unterscheiden sich von zurgeordneten Prozeduren dadurch, daß sie nicht durch Wertänderungen, sondern durch Nachrichten von anderen Objekten aktiviert werden.

Objektorientierte Wissensrepräsentationen können auch als Graphen aufgefaßt werden, wobei die Objekte den Knoten und ihre Beziehungen (Slots) den Kanten entsprechen. Wegen der unterschiedlichen Bedeutung müssen die Kanten benannt werden, d. h. man erhält ein semantisches Netz.

Hierbei betrachtet man semantische Netze nicht als eigenständige Wissensrepräsentation, sondern nur als graphische Darstellung von objektorientierten Repräsentationen, da ein Knoten mit seinen Kanten zu anderen Knoten intern als Objekt mit Eigenschaften und Werten dargestellt wird.

Man kann die objektorientierte Programmierung durch einfaches Herstellen von Beziehungen, wie

Frame ↔ Rahmen, für das Verständnis einer Situation,
Slot ↔ relevante Eigenschaften der Situation,
Erwartungswert ↔ typische Ausprägung einer Situation,

in einen **kognitiven Rahmen** einordnen. Das Hauptproblem beim Wiedererkennen von Objekten liegt in den Erwartungswerten, da diese überschrieben werden können.

Eine radikale Konsequenz daraus besteht darin, daß man Erwartungswerte und Abweichungen bei der Objektdefinition verbietet. Dies wird im nächsten Abschnitt am Beispiel der KL-ONE-Sprachen untersucht.

2.5.2.1 Automatische Klassifikation in KL-ONE-Sprachen

Die Kernidee der KL-ONE-Sprachen ist die strikte Trennung von Aussagen über ein Objekt (in KL-ONE "Konzept") in **definierende** und **optionale Eigenschaften** und die Unterscheidung zwischen generischen Konzepten in der **T-Box** (Terminological Box) und den Instanzen in der **A-Box** (Assertion Box).

Eine **automatische Klassifikation** wird durch die definierenden Eigenschaften in der T-Box erreicht. Dort kann der Benutzer nicht mehr wie in der FRL einen Frame durch einen einfachen Eintrag in seinen AKO-Slot als Unterframe eines anderen Frame deklarieren, sondern muß die Eigenschaften (in KL-ONE "Rollen"), bezüglich derer ein Konzept spezieller als sein Vorgänger ist, präzisieren.

Die generischen Konzepte sind in KL-ONE entweder primitiv oder mittels anderer Konzepte definiert und durch eine Anzahl von Rollen charakterisiert. Die Rollen, die ebenfalls primitiv oder definiert sein können, repräsentieren eine Abbildung zwischen zwei Konzepten und haben eine **Wertebereichs-** und **Kardinalitätsangabe**, die bei der Definition neuer Konzepte eingeschränkt werden kann. Die Kardinalitätsangabe von Rollen in KL-ONE-Sprachen ist ein neues Merkmal, das weder in FRL, noch in der Prädikatenlogik explizit vorhanden ist.

Die wichtigsten Sprachstrukturen, die alle Mitglieder der KL-ONE-Sprachen enthalten sind:

(1) Definition von Konzepten:

- Konjunktion (Conj-Generic):
  Ein Konzept ist die Vereinigung der Eigenschaften mehrerer Konzepte.
- Wert - Beschränkung (VR-Generic):
  Die Wertemenge einer Rolle ist im Vergleich zu übergeordneten Konzepten eingeschränkt.
- Zahlen - Beschränkung (VR-Generic):
  Die Kardinalität einer Rolle ist im Vergleich zum übergeordneten Konzept eingeschränkt.

- Spezialisierung (Prim-Generic):
  Spezialisierung, ohne hinreichende Bedingung.

(2) Definition von Rollen:

- Wertdifferenzierung (VR-Diffrole):
  Eine Rolle, deren Wertemenge beschränkt ist.

- Verkettung (Role-Chain):
  Eine Rolle, die der Verkettung mehrerer Rollen entspricht.

- Spezialisierung (Prim-Role):
  Spezialisierung, ohne hinreichende Bedingung.

Die Einordnung neuer Konzepte in eine existierende Klassifikationshierarchie aufgrund ihrer Eigenschaften und Definitionen ist die Aufgabe des **Classifiers**, des Kernstücks von KL-ONE-Sprachen. Er besteht aus zwei Komponenten:

(1) Einem **Prädikat Subsumption**, das wahr ist, wenn ein Konzept ein anderes subsumiert (unterordnet), d. h. alle Rollen des einen Konzeptes die des anderen Konzeptes subsumieren, wobei eine Rolle eines Konzeptes eine Rolle eines anderen Konzeptes subsumiert, wenn sowohl der Wertebereich, als auch die Kardinalität der Rolle des übergeordneten Konzeptes umfassender sind, als beim untergeordneten Konzept.

(2) Einem **Suchalgorithmus**, der das spezifischste Konzept (SK) herausfindet, welches das einzuordnende Konzept (EK) noch subsumiert, d. h. das Prädikat (Subsumption EK SK) ist noch wahr, aber für alle Unterkonzepte SK' von SK evaluiert (Subsumption EK SK') zu falsch.

Die Komplexität des Classifiers ist sehr empfindlich hinsichtlich der Mächtigkeit der Sprachelemente. So bedingen die hier beschriebenen Sprachstrukturen bereits sehr wahrscheinlich eine überexponentielle Komplexität des Classifiers, während das Weglassen von Wertebeschränkungen für Rollen die Komplexität des Classifiers mindestens auf $O(n^2)$ reduziert. Allerdings kann auch bei einer überexponentiellen Komplexität ein Classifier in "normalen" Fällen sehr gut funktionieren. Hieraus resultiert eine Abwägung zwischen der Mächtigkeit der Sprache und der Effizienz des Classifiers.

Die KL-ONE-Sprachen haben den großen Vorteil einer funktionierenden, implementierungsunabhängigen Semantik, die den Classifier ermöglicht. Allerdings verbirgt sich darin auch der Nachteil, daß vage Zusammenhänge und Ausnahmen bei der Definition von Konzepten eliminiert wurden. Erweiterungen der KL-ONE-Konzeption machen von weiteren Boxen für Weltwissen, U-Box (Universal Box) für notwendige Bedingungen und D-Box (Default Box) für Standardannahmen Gebrauch (/8/).

2.5.3 Skripten

Nach /30/ sind Skripten eine speziell zur Darstellung von Handlungsabläufen geeignete Version der Rahmen Idee. Im folgenden wird, wie bei /31/ der Skript-Ansatz von Schank und Abelson (1977) und Graesser (1981) vorgestellt.

Demnach stellt ein **Skript** ein Schema über eine häufig erfahrene Folge von Ereignissen dar, in deren Verlauf ein oder mehrere Akteure agieren, um ein bestimmtes Ziel oder eine Reihe von Zielen zu erreichen.

Man unterscheidet personenbezogene, situationsbezogene und instrumentelle Skripten, je nachdem, ob sich die Ereignisfolgen auf bestimmte Situationen, Personen oder Wege zur Zielerreichung beziehen.

Skript-Wissen umfaßt zum einen Handlungswissen, aber auch Wissen um Eingangsbedingungen, Szenenfolgen und mögliche Ergebnisse.

Die **Variablen eines Skriptes** lassen sich zwei Kategorien zuordnen:

- Die Variablen, deren adäquate Ausführung jeweils eine Person in einer bestimmten sozialen Rolle verlangt.
- Variablen, die nach Objekten verlangen.

Ein einmal aktiviertes Skript erleichtert die Informationsverarbeitung und ermöglicht die Steuerung einer Sequenz von Handlungen. Bei der Verarbeitung skriptbezogener Informationen sind aufgrund des Skript-Wissens Inferenzen über einzelne Aspekte und Details möglich, die in den betreffenden Informationen explizit nicht enthalten sind.

Die theoretische Erweiterung und Teil-Revision des Skript-Ansatzes betrifft Annahmen über gedächtnispsychologische Realität von Skripten sowie Annahmen über grundlegende, das Skript-Wissen konstituierende Wissenseinheiten, MOPS (Memory Organization Packets) genannt. MOPS sind nach Schank die grundlegenden Organisations-Strukturen des semantischen Gedächtnisses, die episodisches Wissen in abstrahierter Form beinhalten, aber zugleich direkte Bezüge zu den einzelnen Ereignissen aufweisen.

Die Funktion von MOPS wird darin gesehen, daß diese Erwartungen bereitstellen, die die Vorhersage künftiger Ereignisse auf der Basis zuvor erfahrener strukturell ähnlicher Ereignisse erlauben. Skripten sind nach Schank eine spezielle Art von MOPS, die nur für den aktuellen Informationsverarbeitungsprozeß von Bedeutung sind, denen selbst aber keine "Gedächtnisfähigkeiten" (Memory Abilities) zukommen.

Skripten werden je nach Anforderung einer bestimmten Informationsverarbeitungssituation aus diesen grundlegenden Wissenseinheiten unter der Steuerung hierarchisch höherer Gedächtnisstrukturen konstruiert.

Die **Skript-Anwendung** stellt einen rekursiven Prozeß dar:

Skripten werden erst unter dem Aspekt einer bestimmten Anforderung an das Informationsverarbeitungssystem unter der Steuerung von Zielsetzungen und Plänen auf der Basis elementarer Wissenseinheiten (MOPS) konstruiert. Skripten stellen in diesem Sinne prozedural

orientierte mentale Organisationsstrukturen dar, die in MOPS gespeichertes Wissen der Informationsverarbeitung zugänglich machen und diese im Sinne eines instrumentellen Skriptes steuern.

Skripten erleichtern den Verarbeitungsprozeß, indem sie das Verstehen von Episoden erleichtern und damit Verarbeitungskapazität zur Verarbeitung anderer Informationen freistellen.

In einer Präzisierung des bisher Dargestellten werden skriptrelevante Informationen auf vier Ebenen unterschiedlicher Abstraktionsstufe repräsentiert. Die Informationen auf diesen unterschiedlichen Ebenen des Gedächtnisses sind inhaltlich aufeinander bezogen. Die Gedächtnisstrukturen auf diesen Ebenen reichen von Kausalketten von Ereignissen über das Wissen über "Pläne" und "Rahmen" bis hin zu zielgerichteten Verhaltensmustern für einzelne Problemsituationen. Skript-Wissen liegt demnach nicht als Liste von Ereignissen vor, sondern wird nach Maßgabe der je gegebenen Anforderungen einer Informationsverarbeitungssituation aus den jeweils relevanten Informationen einander übergeordneter Gedächtnisstrukturen konstruiert.

Aus der hier dargestellten Sicht ist Skript-Wissen ein aus elementaren abstrakten Wissenseinheiten (MOPS) konstruiertes Wissen, wobei in den Konstruktionsprozeß sowohl tatsächlich gegebene Information über ein Ereignis im Sinne eines bottom-up-Prozesses, als auch die in MOPS sowie in Plänen und Zielen repräsentierten übergeordneten Gedächtnisstrukturen im Sinne eines top-down-Prozesses einfließen.

Als Verfahren zur Generierung von Skripten werden empirische Methoden herangezogen.

2.6 Computer-Algebra

Den Begriff Computer-Algebra kann man am einfachsten dadurch erklären, indem man ihn, wie bei /3/, dem Begriff der numerischen Mathematik gegenüberstellt:

- Die numerische Mathematik befaßt sich mit algorithmischen Verfahren zur Behandlung von Problemen, deren Angaben und Lösungen (Gleitkomma-) Zahlen sind.

- Die Computer-Algebra (Symbolic and Algebraic Computation, Formelmanipulation, Buchstaben-Rechnen mit dem Computer) befaßt sich mit algorithmischen Verfahren zur Behandlung von Problemen, deren Angaben algebraische Objekte sind und deren Lösungen wieder algebraische Objekte sind.

Die algebraischen Objekte, auf denen die Probleme definiert sind und die Lösungsalgorithmen arbeiten, liegen in verschiedenen **algebraischen Bereichen**. Ein algebraischer Bereich ist dabei durch die **Menge der Objekte**, die zu ihm gehören ("Träger" des Bereiches) und die **Grundoperationen**, die man auf den Objekten ausführen kann, definiert.

Beispiele solcher algebraischer Bereiche sind:

- Die natürlichen Zahlen,
- die ganzen Zahlen,
- die Gaußschen Zahlen,
- endliche Körper und
- Polynomräume.

Nicht alle in der Mathematik untersuchten algebraischen Bereiche können im Computer behandelt werden. Für gewisse algebraische Bereiche kann man nicht einmal eine Darstellung aller Objekte des Bereiches angeben; man denke hierbei nur an die Überabzählbarkeit der reellen Zahlen.

Die erste Klasse von Problemen, die in der Computer-Algebra behandelt werden müssen ist:

- Die Darstellung der Objekte algebraischer Bereiche im Computer, insbesondere die Vereinfachung (Simplifikation) von möglichen Darstellungen eines Objektes auf eine Standardform.
- Die Konstruktion von Algorithmen, mit denen die Grundoperationen auf den Objekten im Computer ausgeführt werden soll.
- Die Übersetzung von verschiedenen Darstellungen ein und desselben Bereiches ineinander.

Typische andere, mit Computer-Algebra behandelbare Probleme sind:

- Die Dekomposition von Objekten in einfache Objekte, bzw. die Frage nach der Dekompositionierbarkeit.

  Bsp.: Primzahlzerlegung über den natürlichen Zahlen oder die Faktorisierung von Polynomen.

- Das Auffinden gemeinsamer "Teil-" oder "Oberobjekte" gegebener Objekte.
  Bsp.: Bestimmung des größten gemeinsamen Teilers.
- Die exakte Lösung von Gleichungen und Ungleichungen.

Man kann folgende generelle Aussage über die Bedeutung der Computer-Algebra treffen:

Die Bedeutung der Entwicklung neuen mathematischen, algorithmisch brauchbaren Wissens für die Computer-Mathematik, insbesondere für die Computer-Algebra, wird in Zukunft nicht abnehmen, sondern immer mehr steigen. Echte Fortschritte in der Lösung der Probleme der Computer-Algebra werden nur durch Kombination bester mathematischer Techniken mit den besten Errungenschaften der Softwaretechnologie erzielt werden können; denn ein schlechter Algorithmus (mit hoher Komplexitätsordnung) erlaubt bei der Erhöhung der

Rechnerleistung der verfügbaren Maschinen nur geringe Ausdehnung des Eingabebereiches (z. B. die Dimension einer Matrix) des Algorithmus /3/.

Ein guter Algorithmus erlaubt besseren Maschinen hingegen die Ausdehnung des Eingabebereiches um ein Beträchtliches.

2.7 Constraints

Constraints dienen zur Repräsentation von Relationen, d. h. von irgendwelchen Beziehungen zwischen Variablen. Sie eignen sich besonders zur Darstellung von lokalen Randbedingungen, die der Problemlöser in jedem Fall erfüllen muß, ohne daß damit eine konkrete Problemlösung festgelegt wird.

Durch jedes neue Constraint wird der Lösungsraum zusätzlich eingeschränkt; das Ziel von Constraintsystemen ist das Herausfinden einer Lösung unter Beachtung aller Constraints.

Eines der ersten bekannten Constraintsysteme EL hat als Anwendungsgebiet die Simulation elektrischer Schaltkreise. Alle Komponenten eines Schaltkreises sind in EL als Constraints repräsentiert und durch gemeinsame Variablen (Spannungen und Ströme) zu einem Constraint-Netz verbunden.

Während Constraints ungerichtete Zusammenhänge zwischen Variablen ausdrücken, die bzgl. jeder Variablen aufgelöst werden können (z. B. $U = R \cdot I$), so repräsentieren Regeln gerichtete Zusammenhänge (z. B. $A \Rightarrow B$).

Einen Constraint kann man häufig als eine mathematische Gleichung betrachten und ein diesen Constraint umfassendes Constraint-Netz als das zugehörige Gleichungssystem. Das Lösen des Gleichungssystems entspricht der **Constraint-Propagierung**.

Allerdings sind Constraints nicht nur auf Gleichungen beschränkt, sie können auch Ungleichungen und nicht-numerische Zusammenhänge ausdrücken. Daher eignen sich Constraints vorzüglich zur quantitativen oder qualitativen Modellierung von Systemen und physikalischen Zusammenhängen.

Constraints können sowohl als Datenbanken (für Tabellen), als Regeln oder auch als beliebige Programme implementiert werden.

Der Constraint-Propagierungs-Algorithmus hat als Eingabe ein Constraint-Netz sowie eine Teilbelegung von Variablen mit Werten und als Ausgabe eine mit den Constraints konsistente Wertzuweisung an weitere Variablen. Wenn alle Variablen genau einen Wert annehmen, liegt eine eindeutige Lösung vor, Wertemengen für Variablen entsprechen multiplen Lösungen und die Zuweisung der leeren Menge an eine Variable bedeutet eine Inkonsistenz.

Die Propagierung besteht im wesentlichen darin, daß die Beschränkung der Wertemenge einer Variablen über die mit ihr verbundenen Constraints an andere Variablen weitergegeben wird, bis keine weitere Einschränkung des Wertebereichs irgendeiner Variablen mehr möglich ist.

Techniken zur Lösung eines Constraintsystems sind die einfache Propagierung mit festen Werten oder Wertemengen, Fallunterscheidungen und die meist aufwendig zu realisierende symbolische Propagierung.

2.8 Spezielle Schlußfolgerungsverfahren

2.8.1 Probabilistisches Schließen

In der klassischen Logik geht man meist davon aus, daß eine Aussage entweder wahr oder falsch ist und berücksichtigt selten, daß man eine Aussage nur für wahrscheinlich hält oder nichts darüber weiß. Da diese Zwischentöne in Anwendungsbereichen von Expertensystemen häufig vorkommen, muß die Wissensrepräsentation und vor allem die Problemlösungsstrategie entsprechend erweitert werden, wofür die beiden Hauptansätze das probabilistische und das nicht-monotone Schließen sind.

Die Basis des probabilistischen Schließens ist die Bewertung jeder Aussage mit einer Wahrscheinlichkeit, die den Grad der Unsicherheit widerspiegelt.

Der wichtigste Problembereich für den Umgang mit Unsicherheiten ist die Diagnostik, bei der Muster (die Diagnose) anhand ihrer Eigenschaften wiedererkannt werden sollen. Die Unsicherheiten bei der Diagnostik stammen aus folgenden Quellen:

- Symptomerhebung (Feststellung der Evidenz der Symptome),
- Symptombewertung (Zuordnung der Symptome zu Diagnosen),
- Unzulänglichkeiten des Verrechnungsschemas.

Meistens werden in Expertensystemen Unsicherheiten bei der Symptomerhebung vom Benutzer und Unsicherheiten bei der Symptombewertung vom Experten geschätzt. Da verschiedene Personen meist unterschiedliche Maßstäbe haben und eine Normierung nicht möglich ist, vergrößert die Verrechnung die Gesamtunsicherheit beträchtlich.

Um keine größere Genauigkeit vorzutäuschen, als tatsächlich erreichbar ist, werden in Expertensystemen häufig vereinfachte Annahmen gemacht, indem nur einfache Unsicherheitskategorien unterschieden werden (z. B. sicher, wahrscheinlich, möglich etc.).

Auch die Verrechnungsschemata sind meist entsprechend einfach. Von Statistikern werden diese Mechanismen, aufgrund der fehlenden theoretischen Fundierung häufig "ad-hoc-Repräsentationen" genannt. Andererseits sind die Voraussetzungen zur Anwendung fundierter statistischer Verfahren zum probabilistischen Schließen in den meisten Anwendungsbereichen grob verletzt. Daher kann man auch keinen idealen Formalismus zum Umgang mit Unsicherheiten erwarten, dies spiegelt sich auch in der Vielzahl der existierenden Ansätze wider.

Während sich die verschiedenen Mechanismen im Detail stark unterscheiden, basieren sie auf demselben Grundalgorithmus zur Bewertung von Diagnosen:

(1) Starte mit den a priori Wahrscheinlichkeiten aller Diagnosen.

(2) Modifiziere für jedes Symptom die Wahrscheinlichkeit aller Diagnosen entsprechend den Symptom-Diagnose-Wahrscheinlichkeiten.

(3) Selektiere die wahrscheinlichste Diagnose.

Zur Anwendung dieses Algorithmus benötigt man Berechnungen oder Abschätzungen der Symptom-Diagnose-Wahrscheinlichkeiten aller relevanten Symptom-Diagnose-Paare und der symptomabhängigen a priori Wahrscheinlichkeiten der Diagnosen.

2.8.1.1 Das Theorem von Bayes

Das Theorem von Bayes eignet sich dazu, aus den a priori Wahrscheinlichkeiten $P(D_i)$ einer Menge von n Diagnosen und aus den bedingten Wahrscheinlichkeiten $P(S_j \mid D_i)$, d. h. der Häufigkeit des Auftretens eines Symptoms bei Vorhandensein einer Diagnose, die wahrscheinlichste Diagnose unter der Annahme der Symptome $S_1, S_2, S_3, ..., S_m$ gemäß der **Bayesschen Formel**

$$P_r(D_i \mid S_1 \& S_2 \& ... \& S_m) = \frac{P(D_i) \cdot P(S_1 \mid D_i) \cdot ... \cdot P(S_m \mid D_i)}{\sum_{i=1,...,n} P(D_j) \cdot P(S_1 \mid D_j) \cdot ... \cdot P(S_m \mid D_j)}$$

zu berechnen. P_r ist hierbei die relative Wahrscheinlichkeit einer Diagnose im Vergleich zu den anderen Diagnosen. Die Wahrscheinlichkeiten der rechten Seite der Formel können aus einer Falldatenbank ermittelt werden.

Um das Theorem von Bayes in obiger Form anwenden zu können, müssen die folgenden Voraussetzungen gelten:

- Die Symptome dürfen nur von der Diagnose abhängen und müssen untereinander unabhängig sein; dies ist der kritische Punkt!

- Die Diagnosemenge muß vollständig sein.
- Die Diagnosen müssen sich wechselseitig ausschließen (single-fault-assumption); diese Annahme ist nur bei relativ kleinen Datenmegen praktikabel.
- Man muß über weitgehend fehlerfreie und vollständige Statistiken zur Herleitung der a priori Wahrscheinlichkeiten der Diagnosen und der bedingten Symptom-Diagnose-Wahrscheinlichkeiten verfügen.
- Die Wahrscheinlichkeiten müssen konstant angenommen werden. Dies ist nicht nur problematisch, weil sich die Erscheinungsbilder der Diagnosen verändern können, sondern auch wegen der Änderung der Diagnosemethoden.

Da im allgemeinen zumindest einige dieser Voraussetzungen verletzt sind, suggeriert die Anwendung des Theorems von Bayes zumeist eine größere Genauigkeit, als sie in der Praxis möglich ist.

Im folgenden werden kurz Mechanismen beschrieben, mit denen durch Zusatzwissen die Fehlerquellen im Bayesschen Theorem verringert werden können. Es bleibt jedoch zu beachten, daß die theoretische Absicherung mit jedem Mechanismus schwieriger wird.

- Symptomkombinationen kann man durch Regeln der Art A&B&C → D berücksichtigen.
- Die Bedingung sich wechselseitig ausschließender Diagnosen kann man durch Partitionierung der Diagnosemenge abschwächen. Mehrfachdiagnosen kann man daran erkennen, daß die Diagnose nicht alle Symptome erklären kann (siehe z. B. das INTERNIST- Modell).
- Mit dem Theorem von Bayes läßt sich die Zuverlässigkeit des Ergebnisses abschätzen. Das kann man durch Angabe von Wahrscheinlichkeitsintervallen anstelle von Wahrscheinlichkeiten für Diagnosen erreichen (Dempster-Shafter-Theorie). Eine andere Möglichkeit ist, bei großer Unsicherheit nur allgemeine Grobdiagnosen zu etablieren, die bei zusätzlichem Wissen verfeinert werden.
- Im Theorem von Bayes wird ein Symptom, das mit x% für eine Diagnose spricht mit (100 - x%) gegen die Diagnose ausgewertet. Intuitiv einleuchtender erscheint eine getrennte Bewertung von positiver und negativer Evidenz einer Diagnose.
- Auch ein starres Verrechnungsschema ist eine Quelle der Unsicherheit, wenn dessen Voraussetzungen nicht erfüllt sind. Daher kann man unterschiedliche Verrechnungsschemata für die Bewertung verschiedener Diagnosen anbieten (Medi-Modell).
- Schließlich kann man auch versuchen, die Unsicherheiten ganz abzuschaffen und sie durch detailliertes Wissen, wie z. B. Ausnahmen von Regeln und kausale Modelle, ersetzen.

2.8.2 Nicht-monotones Schließen

Während in der klassischen Logik eine Ableitung zeitlich unveränderlich gültig ist (vgl. Abschnitt 4.8), können beim nicht-monotonen Schließen neue Informationen bewirken, daß Ableitungen wieder zurückgezogen werden müssen.

Dieses auf den ersten Blick unsystematische Verhalten beim Problemlösen ist jedoch in der realen Welt der Normalfall, da eine vollständige und fehlerfreie Datenerhebung entweder unmöglich oder viel zu aufwendig ist. Gewöhnlich hat man Erwartungswerte oder möchte plausible Schlüsse aus unvollständigen Daten ziehen und diese gegebenenfalls korrigieren können.

Man beachte hierbei den prinzipiellen Unterschied zwischen Ausnahmen von Regeln und negierten Aussagen in der Vorbedingung: Bei negierten Aussagen muß man beweisen, daß keine Aussage zutrifft, bevor die Regel feuern kann, während bei Ausnahmen bewiesen werden muß, daß mindestens eine Ausnahme zutrifft, um eine Regelausführung zu verhindern; d. h. die Unkenntnis über eine Aussage wird bei Ausnahmen zugunsten der Regel und bei negierten Aussagen gegen die Regel ausgewertet.

Ausnahmen basieren auf einer mindestens dreiwertigen Logik, denn eine Aussage kann wahr, falsch oder unbekannt sein. Man muß jedoch auch bei der zweiwertigen Logik nicht-monotones Schließen anwenden, wenn man aus der Tatsache, daß man ein Theorem nicht beweisen kann folgert, daß das Theorem nicht zutrifft.

Eine solche Strategie bezeichnet man als "Negation as Failure" bzw. "Closed World Assumption" und sie wird bei PROLOG angewendet.

Während bei solchen Strategien der logische Wert "unbekannt" als "falsch" repräsentiert wird und deswegen zu Revisionen führen kann, existiert "unbekannt" in der klassischen Logik überhaupt nicht, weswegen dort Ableitungen außer bei Änderung der Axiome immer gültig bleiben.

Die Formalisierung des nicht-monotonen Schließens ist sowohl ein großes theoretisches Problem, für das verschiedene Kalküle entwickelt wurden, die jedoch im allgemeinen nicht semi-entscheidbar sind und keine Hilfe für Implementierungen darstellen, als auch ein praktisches Problem bei der Entwicklung wissensbasierter Systeme.

Bei der Rücknahme von Schlußfolgerungen (Belief Revision) muß beachtet werden, daß von einer revidierten Schlußfolgerung wieder andere Schlußfolgerungen abhängen können und die Rücknahme eines Elementes in diesem Netzwerk eine Kettenreaktion auslösen kann.

Belief-Revision-Systeme oder Truth-Maintenance-Systeme (TMS), wobei TMS oder Reason-Maintenance-Systeme (RMS) unter dem Oberbegriff Belief-Revision-Systeme zusammengefaßt werden, haben die Aufgabe in den oben genannten Situationen den Zustand herzuleiten, der entstanden wäre, wenn das geänderte oder neue Faktum gleich von Anfang an berücksichtigt worden wäre.

Der einfachste Algorithmus ist die Neuberechnung aller Schlußfolgerungen aus den veränderten Eingabedaten. Eine einfache Verbesserung ist die Abspeicherung eines Fehlerprotokolls aller Schlußfolgerungen und die Neuberechnung ab dem Zeitpunkt, wo das

geänderte Faktum oder die zu ändernden Schlußfolgerungen das erste Mal verwendet wurden (Backtracking). Der Aufwand dieser beiden "brute-force" Methoden wächst natürlich rapide mit der Größe der zu verwaltenden Datenmenge und ist nur in kleinen Expertensystemen vertretbar.

Wesentlich ökonomischer ist es, sich die Neuberechnungen eines großen Teils oder der gesamten Sitzung zu ersparen und nur die Schlußfolgerungen zu korrigieren, die tatsächlich von der Änderung betroffen sind. Dazu muß man für jede Schlußfolgerung ihre Begründung mitabspeichern und bei Rücknahme eines Faktums rekursiv überprüfen, welche Begründungen dadurch ungültig werden.

Die beiden wichtigsten Ansätze zum Belief-Revision unterscheiden sich darin, was als Begründung einer Schlußfolgerung abgespeichert wird. Die Grundideen dieser beiden Ansätze werden am folgenden Beispiel illustriert, das auf einer zweiwertigen Logik ohne Ausnahmen, aber mit der Closed-World-Assumption basiert.

Gegeben seien die drei Regeln R1: A $\rightarrow$ B, R2: B $\rightarrow$ C und R3: D&$\neg$E $\rightarrow$ C. Die beiden Ansätze sind:

- Direkte Begründungen (Justification-Based-TMS, JTMS):

  Eine Begründung ist gültig, wenn alle ihre positiven (monotonen) Vorbedingungen gültig und alle ihre negativen (nicht monotonen) Vorbedingungen ungültig sind.
  Das Hauptproblem bei JTMS Ansätzen ist die Behandlung von zirkulären Begründungen, die die Rücknahme von Schlußfolgerungen in einer Schleife verhindern.

- Basisannahmen liegen einer direkten Begründung zugrunde (Assumption-Based-TMS, ATMS):

  Beim ATMS gilt B im Kontext K1 = { A } und C gilt in den Kontexten K2 = { A } und K3 = { D, $\neg$E }. Der Unterschied zwischen JTMS und ATMS wirkt sich hier so aus, daß beim ATMS die Basisannahme { A } einen Begründungskontext für { C } herstellt, während im JTMS die Schlußfolgerung { B } diese Aufgabe erfüllt.
  Der Hauptaufwand bei ATMS-Ansätzen besteht in der Verwaltung des Abhängigkeitsnetzes, während die konkrete Rücknahme eines Faktums sehr effizient durchführbar ist.

Der Hauptvorteil des ATMS ist die Möglichkeit, verschiedene Lösungen in verschiedenen Kontexten unabhängig voneinander verfolgen zu können. Dies ist z. B. in der qualitativen Simulation (vgl. Abschnitt 5.3) nützlich, in deren Zusammenhang auch das ATMS entwickelt wurde.

Interessiert man sich hingegen nur für eine Lösung, dann dürfte ein JTMS effizienter sein, da das ATMS Zwischenergebnisse beim Problemlösen wieder in ihre Basisannahmen auflösen muß. Man darf jedoch nicht übersehen, daß die allgemeine Implementierung beider Ansätzte bei großen, stark vernetzten Wissensbasen ziemlich ineffizient ist. Ausreichende Effizienz ist derzeit wohl nur in Spezialfällen möglich, z. B. wenn keine oder nur vorberechenbare Zirkularitäten zugelassen sind oder wenn die Wissensbasis nicht zu groß ist.

Für detailliertere Ausführungen zu JTMS und ATMS sei auf /25/ S. 59-63 verwiesen.

2.8.3 Temporales Schließen

Die Darstellung zeitabhängiger Informationen wurde in Expertensystemen bisher weitgehend vernachläßigt, weil dadurch die Wissensrepräsentation und die Ableitungsstrategie wesentlich komplizierter werden kann. Andererseits gibt es viele Anwendungsgebiete, bei denen Zeitangaben wichtig sind. Beispielsweise ist in der Diagnostik die zeitliche Veränderung eines Symptoms häufig aussagekräftiger, als der aktuelle Wert.

Die derzeit erfolgsversprechendste Taktik für temporales Schließen in Expertensystemen besteht darin, daß man sich auf die Repräsentation der für den Anwendungsbereich wichtigsten Zeitaspekte beschränkt.

Ein grundsätzlicher Unterschied besteht zwischen Systemen, die aus vorhandenen zeitabhängigen Daten Schlußfolgerungen ziehen und solchen, die hypothetische Situationen in der Zukunft herleiten. Erstere kann man als Zeitdatenbanken zur Verwaltung und Abfrage zeitbezogener Daten betrachten, während letztere Simulationssysteme zur Vorhersage oder Erklärung von Ereignissen sind (vgl. Abschnitt 5.3).

Ähnlich wie beim probabilistischen Schließen Aussagen mit einer Unsicherheitsangabe und beim nicht-monotonen Schließen Ableitungen mit Begründungen versehen werden, so werden beim temporalen Schließen Fakten mit einer Zeitangabe qualifiziert.

Zur genaueren Repräsentation gibt es zahlreiche Variationen. Einige davon werden im folgenden kurz erläutert:

- Basisrepräsentation:
  Diese unterscheidet man nach punkt- und intervall-basiert. Beide Repräsentationen sind aufgrund der Intervalldarstellung gleich mächtig.

- Genauigkeit:
  Am einfachsten ist die Repräsentation exakter Zeitangaben. In vielen Anwendungsbereichen sind jedoch keine genauen Zeitdefinitionen verfügbar. Ungenauigkeiten können entweder qualitativ oder quantitativ angegeben werden.

- Zeiteinheiten:
  Eine Zeitrepräsentation mit qualitativen Zeitangaben wirkt wesentlich natürlicher, wenn außer Zahlen auch Zeiteinheiten dargestellt werden. Außerdem kann die Wahl der Zeiteinheit auch ein Mittel zur Beschreibung von Ungenauigkeiten sein, beispielsweise bedeuten "vor einem Jahr" und "vor 365 Tagen" nicht unbedingt dasselbe.

- Bezug:
  Man unterscheidet drei Typen, wie eine Zeitangabe zu anderen in Bezug gesetzt werden kann, nämlich absolute Zeitskala und einfache oder mehrfache Referenzierung. Referenzereignisse haben den Vorteil, daß sie im Gegensatz zu einer absoluten Zeitskala keine totale Ordnung aller Zeitangaben erzwingen.

  Beschreibt man eine ungenaue Zeitangabe durch Bezug zu mehreren Referenzereignissen, so kann die Schnittmenge der verschiedenen Unsicherheitsintervalle genauer als jede einzelne Relation sein. Die Berechnung und Konsistenthaltung mehrfacher Referenzereignisse verkompliziert die Zeitrepräsentation im Vergleich zur Angabe

einfacher Referenzereignisse beträchtlich. Eine geeignete Implementierungstechnik ist die Darstellung der Zeitrelationen als Constraints und die Berechnung der genauesten Beziehung zwischen zwei Ereignissen als Constraint-Propagierung (vgl Abschnitt 2.7).

Da bei einer Einschränkung eines Unsicherheitsintervalles durch zusätzliche Informationen auch Widersprüche auftreten können, kann auch die Benutzung eines Truth-Maintenance-System (TMS) (vgl. Abschnitt 2.8.2) erforderlich sein.

Die Komplexität der Zeitdatenbanken bestimmt natürlich, wie aufwendig und effizient Fragen an sie beantwortet werden können.

Das wichtigste Merkmal von Zeitdatenbanken ist der zulässige Grad der Ungenauigkeit zeitlicher Beziehungen. Während einfache, exakte Beziehungen schon in einigen Expertensystemen realisiert sind, ist die Verwaltung von mehrfachen, ungenauen Zeitangaben wesentlich aufwendiger, da auch mögliche Reduktionen der Unsicherheit durch zusätzliche Zeitinformationen verwaltet werden müssen. Dazu eignen sich Techniken der Constraint Propagierung.

Zeitvariable Systeme müssen auch in der Lage sein, Schlußfolgerungen zu revidieren, wozu außer in kleinen Anwendungsgebieten ein Truth-Maintenance-System (TMS) erforderlich ist.

Für eine ausführlichere Darstellung der verschiedenen Aspekte temporalen Schließens sei auf /1/ bzw. /25/, S. 66 - 71 verwiesen.

3 Objektorientierte Programmierung

3.1 Einleitung

Der objektorientierte Ansatz verspricht für die Software-Entwicklung in den neunziger Jahren ähnliche Bedeutung zu erlangen, wie die strukturierte Programmierung im vergangenen Jahrzehnt.

Es werden zum einen technische Konzepte angeboten, die die Erweiterbarkeit und Wiederverwendung von Softwarebausteinen unterstützen, und zum anderen ein Vorgehen beim Entwurf vorgeschlagen, das es gestattet, Struktur und Komplexität des Problembereiches in "natürlicher" Weise auf die Software zu übertragen; einfache Änderungen im Anwendungsgebiet bleiben auch im Softwaresystem einfach.

Der objektorientierte Ansatz verbindet bewährte Prinzipien der Software-Entwicklung wie "abstrakte Datentypen" und "Information Hiding" mit den weniger bekannten Konzepten "Vererbung" und "dynamisches Binden" und kombiniert diese mit einer anwendungsnahen Sicht der Welt zu einem konsistenten und durchgängigen Verfahren.

Im objektorientierten Ansatz ist es durch das Klassenkonzept und das Konzept der Vererbung möglich, Gemeinsamkeiten, Verallgemeinerungen, aber auch Spezialisierungen von Komponenten mit programmiersprachlichen Mitteln zu beschreiben. Daraus ergibt sich langfristig ein Produktivitätsgewinn durch die Einsparung an Entwicklungszeit und -Kosten.

Darüber hinaus wird durch die Verwendung von standardisierten und ausgetesteten Bausteinen das zu erstellende Softwareprodukt sicherer und qualitativ hochwertiger sowie der Testaufwand geringer.

Die konsequente Anwendung der Konzepte der objektorientierten Programmierung verspricht die Erstellung von klar strukturierten, sicheren, änderbaren, erweiterbaren und leicht wartbaren Systemen, wobei vorhandene Software-Bausteine auf einfache Weise wiederverwendet werden können.

Die Grundlage für ein Verständnis der Objektorientierung und ihres Einsatzes ist ein konsistenter Satz von Basiskonzepten. Diese werden im folgenden vorgestellt und erläutert. Hierzu werden zunächst objektbasierte Systeme eingeführt und diese dann eingeschränkt, um zu objektorientierten Systemen zu gelangen. Dieser gesamte Abschnitt basiert zu großen Teilen auf /27/.

3.2 Die Grundkonzepte objektorientierter Programmierung

3.2.1 Überblick

Bei der objektbasierten Programmierung stehen nicht wie beim herkömmlichen Vorgehen Unterprogramme oder Prozeduren, sondern **Objekte** - auch **Instanzen** genannt - im Vordergrund.

Diese Objekte modellieren konkrete oder abstrakte Einheiten aus dem Anwendungsbereich. Sie bestehen zum einen aus Daten, den **Instanzvariablen**, die den veränderlichen Zustand des Objektes realisieren, und zum anderen aus Aktionen, den sogenannten **Methoden**. Diese beiden Teile werden als Einheit aufgefaßt.

Während der Laufzeit werden Methoden ausgeführt; dabei kann ein Objekt Nachrichten an andere Objekte verschicken und dadurch Aktionen beim Empfänger anstoßen. Die Beschreibung eines Objektes ist seine **Klasse**. Sie enthält die Deklaration der Instanzvariablen sowie die Beschreibung der Methoden und besitzt Typcharakter. Zu jedem Objekt gibt es genau eine Klasse; dagegen beschreibt eine Klasse im allgemeinen mehrere gleichartige Objekte, die sich in den Werten der Instanz- variablen unterscheiden können.

Die Instanzvariablen sind von außen nicht sichtbar ("Information-Hiding"), sie können nur indirekt durch Methodenaufrufe gelesen oder modifiziert werden.

Die Menge aller Klassen ist durch die **Vererbung** hierarchisch strukturiert. Durch sie können Gemeinsamkeiten und Differenzen ähnlicher Klassen knapp und übersichtlich beschrieben werden.

Der **Polymorphismus** (Vielgestaltigkeit) schließlich ermöglicht es, daß eine Instanzvariable zu unterschiedlichen Zeiten auf Objekte unterschiedlicher Klassen verweist, die dasselbe Methodenangebot, aber unterschiedliche Implementierungen besitzen.

3.2.2 Objekte und Kommunikation

Bevor man auf das prozedur- und objektorientierte Vorgehen eingehen kann, muß die Beschreibung von Daten und deren Manipulation von diesen selbst unterschieden werden. Dieser Unterscheidung entspricht der statische und der dynamische Aspekt der Software.

Bei herkömmlicher Software sind die Datenbeschreibung und die Beschreibung der Manipulation unabhängig voneinander oder nur lose gekoppelt. Die Software besteht typischerweise aus einer Reihe von Typ- und Variablendeklarationen (den Datenbeschreibungen) und einer Reihe von Prozeduren (der Beschreibung der Manipulation). Diese Zweiteilung spiegelt sich auch in dem resultierenden ablauffähigen Softwaresystem wider: Ein als Hauptprogramm ausgezeichnetes Unterprogramm wird ausgeführt. Dieses manipuliert irgendwelche Daten unter anderem dadurch, daß es andere Unterprogramme aufruft, denen es Daten zur Manipulation übergibt. Die aufgerufenen Unterprogramme verfahren vollkom-

men analog. Wegen der großen Bedeutung der Unterprogramme wird derartige Software auch als prozedurorientierte Software bezeichnet.

Die Idee bei objektbasierter Software besteht darin, sie anders zu strukturieren, indem sie in einzelne "Pakete" aufgeteilt wird, die zum einen die Beschreibung eines bestimmten Datentyps und zum anderen die Beschreibung von Manipulationen derartiger Daten enthalten. Die Daten und die zugehörigen Manipulationen werden also als Einheit aufgefaßt. Ein solches "Paket" beschreibt den Inhalt und das Verhalten von Objekten zur Laufzeit. Ein Objekt O enthält nach außen unsichtbare Daten (es hat einen Zustand) und eine Reihe von sichtbaren "Knöpfen", durch deren "Drücken" ein anderes Objekt A bestimmte Dienste von dem gegebenen Objekt O anfordern kann. Durch dieses "Drücken" werden in O Aktionen angestoßen; es führt Anweisungen aus. Dabei liest oder modifiziert es die eigenen Daten. Das "Drücken eines Knopfes" durch A wird fortan als **Methodenaufruf** oder **Senden einer Nachricht durch A** bezeichnet. Entsprechend bezeichnet man A als Absender und O als Empfänger der Nachricht, durch die auch Argumente übergeben werden können.

Methodenaufrufe dienen dazu, den inneren Zustand eines Objektes abzufragen oder ihn zu modifizieren. Wie diese Methoden, d. h. die Aktionen, realisiert sind, ist ebenso wie die inneren Daten selbst von außen nicht sichtbar. Verschiedene Objekte können auf gleiche Nachrichten durchaus unterschiedlich reagieren. Dies unterscheidet die objektorientierte Sicht von Programmiersprachen mit linearem Namensraum, in denen jeder Operation (Methode) ein vom Kontext unabhängiger eindeutiger Prozedurname zugeordnet sein muß (/18/).

Das äußere Erscheinungsbild eines Objektes ist also bestimmt durch die Gesamtheit aller von ihm angebotenen Methoden, sein **Protokoll**, mit denen sein Zustand abgefragt und modifiziert werden kann.

Von dieser äußeren ist die innere Erscheinung zu unterscheiden. Der Zustand des Objektes - seine inneren Daten - ist wiederum durch Objekte oder Verweise auf Objekte realisiert.

Die Methoden selbst haben eine prozedurale Form. Sie bestehen aus einer Sequenz von Anweisungen, durch die neue Objekte erzeugt werden oder Nachrichten an bereits existierende Objekte versandt werden. Schließlich muß es natürlich gewisse elementare, nicht weiter zusammengesetzte Objekte, wie z. B. die natürlichen Zahlen geben.

Neben den inneren Daten kann das Objekt auch noch lokale Methoden besitzen, die nur von ihm selbst, nicht aber von anderen Objekten nutzbar sind.

Das gesamte ablauffähige Softwaresystem besteht dann aus einer Menge miteinander durch Nachrichtenaustausch kommunizierender Objekte. Die Zahl der Objekte braucht dabei keineswegs konstant zu sein, es können neue Objekte erzeugt und nicht mehr benötigte entfernt werden. Die Ausführung kann durch Erzeugung eines "Ur-Objektes" und Aufruf einer seiner Methoden angestoßen werden.

3.2.3 Klassen und Instanzen

Aus obigem wird deutlich, daß ein objektbasiertes Softwaresystem aus einer Menge von untereinander durch Nachrichten kommunizierenden Objekten besteht. Häufig trifft man dabei auf Objekte, die zwar unterschiedliche Dateninhalte besitzen, aber in der Implementierung ihrer Methoden übereinstimmen. Das Gemeinsame dieser gleichartigen Objekte bezeichnet man als **Klasse**. D. h. Klassen sind Beschreibungen gleichartiger Objekte.

Gemäß dieser Definition handelt es sich bei Klassen um (statische) Software. Klassen existieren also - im Gegensatz zu den beschriebenen Objekten (ihren Instanzen) - nicht nur zur Laufzeit. Sie extrahieren das Gemeinsame aus mehreren Objekten; der Übergang von einem Objekt zu seiner Klasse - die **Klassifikation** - stellt eine Abstraktionsprozeß dar.

Klassen beschreiben Gruppierungen von Objekten in Kategorien, so daß die Objekte einer jeden solchen kategorischen Gruppierung gleiche oder ähnliche Strukturen und Verhaltensmuster aufweisen, in Form einer Schablone (/18/).

Bei einer Klasse unterscheidet man zwei Teile: Die Spezifikation (öffentlicher Teil) und den Rumpf (privater Teil). Die **Klassenspezifikation** enthält das Protokoll, d. h. die nach außen sichtbare Schnittstelle der Objekte, also Namen und Parameter der durch andere Objekte aufrufbaren Methoden. Der **Klassenrumpf** dagegen enthält alle von außen unsichtbaren Implementierungsdetails.

In Methodenrümpfen wird die Art und Weise beschrieben, in der ein Objekt dieser Klasse zur Laufzeit auf einen Methodenaufruf reagiert und seine Daten manipuliert. Daher wird im Klassenrumpf auch die Zahl und Art dieser privaten Daten festgelegt. Das innere Erscheinungsbild einer Klasse, der Klassenrumpf, ist also von der bereits beschriebenen Zweiteilung in einen Daten- und einen Anweisungsteil geprägt. Der Unterschied zur herkömmlichen Software wird erst bei der Außenansicht offenbar: Hier gibt es nur Objekte, die Nachrichten empfangen und bearbeiten, die Zweiteilung ist vollkommen unsichtbar; die Methoden sind untrennbar mit den Daten verbunden.

Der Klassenbegriff stellt also eine Weiterentwicklung des bekannten Typkonzepts dar. Um dies zu verdeutlichen werden diese beiden Konzepte explizit dargestellt:

- Unter einem Typ versteht man meist eine Menge möglicher Werte zusammen mit einem Satz anwendbarer Operationen. Durch die eigentliche Typdeklaration (z. B. type STACK = ...; in PASCAL) wird zunächst der damit im Vordergrund stehende Wertebereich festgelegt; außerdem werden im allgemeinen eine ganze Reihe zugehöriger Basisoperationen, wie etwa der Zugriff auf Komponenten, implizit definiert.

  Der Typ-Programmierer hat keine Einflußmöglichkeiten auf ihre Bedeutung und hat, da sie überall dort, wo der Typ sichtbar ist, ebenfalls sichtbar sind, keine Kontrolle über ihre Verwendung; er kann die Einhaltung einer bestimmten Semantik nicht sicherstellen.

  Der Wertebereich des Typs ist de facto das kartesische Produkt der Wertebereiche der einzelnen Komponenten. Die vom Typ-Programmierer explizit definierten Operationen (z. B. pop) gehören zwar - wie auch die Basisoperationen - mit zum Typ, sind

aber syntaktisch nur lose an die eigentliche Typdeklaration gekoppelt. Genaugenommen ist der Typ von jedem Anwender beliebig erweiterbar.

- Bei einer Klasse bilden die vom Klassen-Programmierer bereitgestellten Operationen eine auch syntaktische Einheit. Er kann ihre Bedeutung frei festlegen und kann, da es keine impliziten Operationen gibt, eine bestimmte Semantik sicherstellen.

  Die anwendbaren Operationen stehen klar im Vordergrund; der Wertebereich ist dagegen implizit durch sie definiert und kann vom Klassen-Programmierer vorgegebenen Bedingungen unterworfen sein.

Es gibt zu jedem Objekt O genau eine Beschreibung, also eine Klasse K. Man sagt dann **K ist die Klasse von O** oder **O ist eine Instanz von K**; d. h. die Instanzen einer Klasse sind die von ihr beschriebenen Objekte.

Instanzen existieren also nur zur Laufzeit in einem Softwaresystem, sie sind dynamisch. Die Erzeugung von Objekten, die **Instantiierung**, wird von einem anderen Objekt angestoßen und besteht aus zwei, nicht voneinander trennbaren Aktionen:

- Zunächst wird der benötigte Speicherplatz bereitgestellt,
- danach wird durch Durchlaufen eines sogenannten Konstruktors (z. B. "new") das Objekt initialisiert, d. h. in den gewünschten Anfangszustand versetzt.

In der "Identifizierung" eines Objektes mit seinem Speicherplatz besteht der wesentliche Unterschied des objektorientierten Ansatzes zum Ansatz relationaler Datenbanken.

Die privaten Daten eines Objektes werden in **Instanzvariablen** abgelegt. Ihre Zahl und Klassenzugehörigkeit (für Sprachen mit Typkonzept) ist für alle Instanzen einer Klasse gleich. Ihr Wert kann jedoch von Instanz zu Instanz variierern. (Besitzen Instanzen einer Klasse K Instanzvariablen der Klasse L, so wird K als **Kunde von L** bezeichnet, vgl. Abschnitt 3.2.4)

Da die Beschreibung der Methoden Teil der Klasse ist, stimmt sie für alle Instanzen überein. Da deshalb beim Eintreffen gleicher Nachrichten von verschiedenen Instanzen einer Klasse dieselben Anweisungen ausgeführt werden, ist ein unterschiedlicher, durch die Instanzvariablen dargestellter, Zustand der einzige Grund für ein unterschiedliches Verhalten dieser Instanzen. Dies soll das folgende Beispiel verdeutlichen:

Beispiel: Klasse: Person
Klassenspezifikation: Name, Geburtsdatum, Wohnort, Alter, etc.
Klassenrumpf: Berechne Alter aus dem aktuellen und dem Geburtsdatum.

Objekte: Herr Müller, Frau Meier, Herr Schmidt.

Bei der Instantiierung wird nun jedem Objekt (Müller, Meier, Schmidt) sein Alter zugewiesen.

Klassen werden hier als Software aufgefaßt, die Objekte beschreibt. Eng damit verwandt ist, Klassen als Implementierung abstrakter Datentypen zu sehen. Damit wird der "Typ-As-

pekt" in den Vordergrund gestellt und gefordert, daß die Semantik der Methoden mehr oder weniger formal und möglichst vollständig definiert wird.

Wichtig ist, daß Klassen keine abstrakten Datentypen sind. Zu einer Klasse gehört nicht nur die Spezifikation - auch der Semantik -, sondern auch eine Implementierung (Rumpf). Eine Klasse ist eine von vielen denkbaren Ausprägungen eines abstrakten Datentyps.

3.2.4 Vererbung und abstrakte Klassen

Die Grundbausteine eines objektbasierten Softwaresystems sind die Objekte. Die Tatsache, daß bei der Beobachtung dieser Objekte häufig gleichartige identifiziert werden können, kann ausgenutzt werden, indem diese nicht unabhängig voneinander beschrieben werden, sondern ihnen in Form von Klassen eine gemeinsame Beschreibung zugeordnet wird. Bei der Betrachtung von Klassen kann man vielfach eine ähnliche Beobachtung machen: Einige Klassen können gleichartig oder zumindest ähnlich sein.

Die Gemeinsamkeiten können verschiedener Natur sein. So sind beispielsweise verschiedene Stack-Klassen denkbar, die zwar in ihrer Spezifikation sowohl syntaktisch, als auch semantisch, nicht aber in ihrer Implementierung übereinstimmen (z. B. Gebrauch von Feldern bzw. Listen). Aufgrund der gleichen Semantik sind diese Klassen "austauschbar", sie können sich aber z. B. in ihrer Effizienz unterscheiden, so daß diese Klassen nur anwendungsabhängig sinnvoll einsetzbar sind.

Die Ähnlichkeit kann auch darin bestehen, daß eine Klasse ein Spezialfall einer anderen Klasse ist. Das kann so aussehen, daß die Spezialklasse ein erweitertes Methodenangebot besitzt oder schärferen Nebenbedingungen genügt; die Implementierungen können dabei trotzdem sehr ähnlich sein, der Datenteil oder die eine oder andere Methode können sogar identisch sein.

Schließlich ist es auch möglich, daß bei zwei Klassen keine ein Spezialfall der anderen ist, sie aber trotzdem in mehr oder weniger großen Teilen übereinstimmen.

Mit dem **Konzept der Vererbung** wird es möglich, Gemeinsamkeiten und Differenzen von Klassen zu beschreiben. Die Idee dabei ist, eine Klase UK von einer anderen Klasse OK abzuleiten und zu erweitern oder zu modifizieren.

Die Relation zwischen UK und OK wird durch "UK ist Unterklasse von OK" oder "OK ist Oberklasse von UK" ausgedrückt. Der erste Schritt, die "Ableitung", geschieht einfach durch die Deklaration von UK als Erben von OK. Damit wird zunächst eine Klasse UK erzeugt, die dasselbe Methodenangebot und dieselbe Implementierung wie OK besitzt. Es gilt hierbei zu betonen, daß dabei weder Quelltext noch Objektcode physikalisch kopiert wird; es genügt ein einfacher Verweis.

Diese gedachte Kopie kann nun aber durch Deklaration und Implementierung neuer Methoden oder Datenkomponenten oder dadurch, daß ein neuer Methodenrumpf den geerbten "überschreibt", ohne die Spezifikation zu ändern, erweitert oder modifiziert werden.

Die **Erben einer Klasse** sind Klassen, die deren Spezifikationen und Implementierungen übernehmen und sie erweitern oder modifizieren.

Klassen, die Erben einer Klasse sind, können genauso wie andere Klasen verwendet werden. Sie können daher auch als Oberklasse weiterer neuer Unterklassen fungieren. Damit ergibt sich eine **Vererbungshierarchie.**

Die nicht durch Vererbung entstandenen Klassen sind die Wurzeln von Bäumen, da eine Klasse beliebig viele Unterklassen, aber (in der hier beschriebenen Form der Vererbung) höchstens eine Oberklasse besitzen kann. Mit Hilfe der sogenannten Mehrfachvererbung, vgl. Abschnitt 4.3.1, kann dieses Konzept erweitert werden.

Mit der Vererbung kann eine Klasse K nicht nur als Kunde, sondern auch als Erbe verwendet werden. Kunden sind Klassen, die der Klasse K zugehörigen Instanzen Nachrichten schicken. Sie "benutzen" die Klasse. Die Kunden können ein zu K vollkommen anderes Verhalten zeigen. Erben von K hingegen sind der Klasse K ähnlich und spezialisieren oder modifizieren sie. Ein wichtiger Unterschied zwischen diesen beiden Verwendungen ist die unterschiedliche Sichtbarkeit. Der Kunde sieht lediglich das exportierte Methodenangebot (dazu gehört sinnvollerweise auch eine Beschreibung der Semantik), der Erbe dagegen hat zudem auch Zugriff auf die Instanzvariablen und die nichtexportierten Methoden.

Bewertung des Vererbungskonzeptes:

- Vorteile:
  - Durch die gute Erweiterbarkeit und Änderbarkeit werden die Einsatzmöglichkeiten und die Wiederverwendung der Klassen gefördert. Es ist möglich, von einer bereits eingesetzten Klasse OK eine Spezialisierung UK abzuleiten, ohne den unverändert bleibenden Teil des Quelltextes von OK zu duplizieren oder gar durch direkte Änderung die Klasse OK zu destabilisieren. Dies bezeichnet man als open-closed-principle.
  - Fehlerkorrekturen an einer Methode einer Oberklasse wirken sich gleichmäßig in allen erbenden Unterklassen aus. Die Konsistenz bleibt gewahrt.
- Nachteile:
  - Durch die aufgrund der Vererbung mögliche Kürze ist der Quelltext der Methoden einer Unterklasse auf sie selbst und ihre (direkten und indirekten) Oberklassen verteilt. Dies kann bei einer tiefen Vererbungshierarchie umfangreicher Klassen zu großer Unübersichtlichkeit führen, wenn keine geeigneten Programmierwerkzeuge (Möglichkeit des Flattenings) zur Verfügung stehen.
  - Die Datenkapselung einer Klasse, die gegenüber ihren Kunden besteht, ist gegenüber ihren Erben teilweise aufgehoben. Das widerspricht dem Prinzip des Information Hiding und erschwert nachträgliche Änderungen.

- Wird die Semantik von Methoden in einer Unterklasse inkompatibel zur Oberklasse geändert (die Unterklasse entspricht nicht mehr einer Untermenge), so kann es in Verbindung mit dem Polymorphismus (vgl. Abschnitt 3.2.5) zu Schwierigkeiten kommen, da dann Oberklassenvariable auf Objekte verweisen können, die der Oberklassensematik nicht genügen.

Bisher wurde behandelt, wie aus einer gegebenen Klasse durch Vererbung spezialisierte Klassen, die Unterklassen, gebildet werden können. Es ist aber auch der umgekehrte Weg möglich. Aus mehreren ähnlichen Klassen kann eine gemeinsame Oberklasse abstrahiert werden. Die Ähnlichkeit zweier Klassen KA und KB bestehe beispielsweise in einer gleichen Semantik, die Implementierungen dagegen seien unterschiedlich (wie z. B. bei den oben bereits erwähnten Stack Realisierungen).

Hier könnte jede der beiden Klassen als Unterklasse der anderen eingeführt und ihre eigene Implementierung entsprechend angepaßt werden. Dies würde jedoch eine der beiden Klassen KA oder KB vor der anderen auszeichnen. Eine andere Möglichkeit besteht darin, eine gemeinsame Oberklasse K einzuführen. Wenn die Gleichberechtigung von KA und KB beibehalten werden soll, kann K natürlich im wesentlichen nur aus der gemeinsamen Spezifikation bestehen, da wir ganz verschiedene Implementierungen angenommen hatten.

Bei K handelt es sich damit nicht um eine Klasse im bisherigen Sinn - eben weil bei K der Klassenrumpf fehlt. Man bezeichnet K als **abstrakte Klasse**. Diese Bezeichnung ist etwas unglücklich, da abstrakte Klassen nach der hier gegebenen Definition strenggenommen keine Klassen sind. So sind sie nicht instantiierbar, da für ein "Objekt" zwar das Methodenangebot, nicht aber der Rumpf vorhanden ist. Die Methoden können zwar aufgerufen, aber nicht ausgeführt werden.

Abstrakte Klassen sind damit unvollständige Beschreibungen gleichartiger Objekte.

Eine abstrakte Klasse besitzt zwar eine gewöhnliche Spezifikation, aber einen nur unvollständigen Rumpf. Sie kommt daher dem Konzept des abstrakten Datentyps näher als eine gewöhnliche Klasse. Es gibt objektorientierte Programmiersprachen, die es nicht gestatten, den Rumpf einer Methode wegzulassen. Bei derartigen Sprachen werden die abstrakten Klassen dadurch simuliert, daß man die betroffenen Methoden mit einem leeren Rumpf versieht, der dann in einer Unterklasse redefiniert wird.

Bisher besteht der Nutzen abstrakter Klassen lediglich darin, die identischen Spezifikationen mehrerer Klassen zusammenzufassen. Die Einsatzmöglichkeiten abstrakter Klassen werden durch den im nächsten Abschnitt vorgestellten Polymorphismus stark erweitert.

Es kann auch der Fall auftreten, daß die Methodenrümpfe einer abstrakten Klasse nicht vollständig, sondern nur teilweise fehlen. Die vorhandenen Rümpfe können dabei nicht-implementierte Methoden aufrufen.

Die hier beschriebenen Aspekte der Vererbung beziehen sich allein auf Klassen, also Software. Diese Klassen dienen der Beschreibung von Objekten, die wiederum Abbilder von Objekten oder Abstraktionen des Problembereiches sind. Es stellt sich damit die Frage, ob es in der realen Welt etwas gibt, das der Vererbung entspricht.

Dem ist natürlich so; denn bereits durch die Bezeichnungen Ober- und Unterklasse wird eine Bedeutung nahegelegt. Den Klassen entsprechen in der realen Welt Mengen, wobei Unterklassen auf Untermengen abgebildet werden. Faßt man Klassen als Typen auf, so sind die Unterklassen die Untertypen.

Diese Sichtweise ist richtig, wenn die Unterklasse aus der Oberklasse durch eine Erweiterung des Methodenangebotes (es werden mehr Nachrichten verstanden) oder des Datenanteils oder durch eine Verschärfung irgendwelcher Bedingungen, denen die Daten oder Methoden unterworfen sind, hervorgeht.

Faßt man die Unterklassen in dieser Weise als Untermengen auf, so ist es ein Fehler, in einem Erben eine Methode (etwa aus "Effizienzgründen") zu redefinieren und dabei die Semantik nicht zu verfeinern, sondern zu verfälschen. Denn dann verhält sich ein Unterklassenobjekt wesentlich anders als ein Objekt der Oberklasse, es gehört nicht mehr zur Untermenge.

3.2.5 Polymorphismus und dynamisches Binden

Das bisher behandelte Vererbungskonzept dient der Vereinfachung der Software-Entwicklung: Gemeinsamkeiten von Klassen werden nur einmal aufgeschrieben. Das Vererbungskonzept ist aber auch bedeutsam für das Laufzeitverhalten. Die Auswirkungen können durch die im folgenden näher erläuterten Stichwörter Polymorphismus und dynamisches Binden gekennzeichnet werden.

In Objekten und Methoden werden Variablen vereinbart, die im allgemeinen auf Objekte nicht nur einer, sondern unterschiedlicher (wenn auch nicht unbedingt beliebiger) Klassen verweisen können. Diese Vielgestalt wird als Polymorphismus bezeichnet; d. h. als **Polymorphismus** bezeichnet man die Fähigkeit einer Variablen, auf Objekte unterschiedlicher Klassen verweisen zu können.

Dieser Polymorphismus hat eine wichtige Konsequenz. Ist O eine polymorphe Variable, so kann für einen bestimmten Methodenaufruf O.M im allgemeinen zur Übersetzungszeit nicht geklärt werden, welcher der unter Umständen zahlreichen Methodenrümpfe M mit gleicher Schnittstelle gegebenenfalls ausgeführt wird. Diese Frage kann erst zur Laufzeit beantwortet werden.

Erst dann wird entschieden, welche Anweisungen bei dem Aufruf O.M ausgeführt werden. Diese späte Festlegung wird als **dynamisches Binden** bezeichnet. Beim normalen (statischen) Binden kann schon zur Übersetzungszeit ein bestimmter Methodenrumpf an den Aufruf "gebunden" werden.

Das dynamische Binden hat nichts mit dem Linken irgendwelcher Codemodule zur Laufzeit zu tun. Offensichtlich ist dynamisches Binden nur möglich, wenn die Objekte in irgendeiner Form die Information mit sich führen, zu welcher Klasse sie gehören.

Wenn eine Variable auf Objekte beliebiger Klassen verweisen kann, wenn also der Polymorphismus von der Sprache keinen Einschränkungen unterworfen wird, dann ist es möglich, daß das angesprochene Objekt O den Methodenaufruf O.M nicht versteht, weil es die Methode M gar nicht anbietet. Das wird in der Regel einen Laufzeitfehler zur Folge haben. Um das zu vermeiden, kann mit einem Typkonzept gewährleistet werden, daß der Aufruf O.M von O verstanden wird. Dazu wird der Variablen in der Deklaration ein **statischer Typ** in Form einer Klasse K zugeordnet. Dieser statische Typ, der auch eine abstrakte Klasse sein kann, legt fest, welche Methodenaufrufe zulässig sind, d. h. vom Compiler akzeptiert werden.

Das obengenannte Problem tritt nicht auf, wenn der **dynamische Typ**, d. h. die Klasse des aktuellen Objektes, eine direkte oder indirekte Unterklasse des statischen Typs K ist. (Dies kann vom Compiler sichergestellt werden, wenn eine Zuweisung OA := OB nur dann akzeptiert wird, wenn der statische Typ von OB eine direkte oder indirekte Unterklasse des statischen Typs von OA ist).

Der statische Typ bestimmt also die Zulässigkeit eines Methodenaufrufs, der dynamische Typ dagegen den auszuführenden Methodenrumpf.

Durch den Polymorphismus wird es möglich, daß eine Oberklassenvariable auch auf Instanzen einer Unterklasse verweist, deren Methodenangebot aber auf das der Oberklasse eingeschränkt ist und die sich damit als Objekte der Oberklasse darstellen. Sie sollten sich daher auch in einer Weise verhalten, die für Oberklassenobjekte akzeptabel ist. Das ist gewährleistet, wenn die Unterklasse einer Untermenge der Oberklasse entspricht. Das Vererbungskonzept sollte nur in dieser eingeschränkten Form verwendet werden.

Betrachtet man nun erneut das Konzept der abstrakten Klassen, die bisher nur als Hilfsmittel zur Beschreibung ähnlicher Klassen genutzt wurden. Zur Laufzeit waren sie nicht zu gebrauchen, da sie nicht instantiierbar sind, es also keine "abstrakten Objekte" gibt. Durch den Polymorphismus wird es aber möglich und sinnvoll, Variablen einer abstrakten Klasse zu verwenden.

Beispielsweise wird es in einem Werkzeug zur Vermögensverwaltung sicher eine Liste von Variablen geben, die den statischen Typ VERMÖGENSWERT besitzen. Diese Klasse wird abstrakt sein, da sich ihr GELDWERT für verschiedene Wertpapiertypen unterschiedlich berechnet (z. B. Berücksichtigung von Stückzinsen bei Rentenwerten). Durch den Polymorphismus wird es möglich, den Gesamtwert des Vermögens einfach als Summe der GELDWERTE aller Objekte in der Liste zu berechnen, ungeachtet dessen, daß dazu unterschiedliche Methodenrümpfe ausgeführt werden.

Software, die auf den in den Abschnitten 3.2.2 bis 3.2.5 behandelten Konzepten beruht, bezeichnet man als **objektorientiert**. Die Konzepte objektorientierter Software gehen mit der Vererbung, dem Polymorphismus und dem dynamischen Binden über die Konzepte objektbasierter Software hinaus.

Zusammenfassend gilt: Dynamisches Binden ist die Zuordnung eines Methodenrumpfes zu einem Methodenaufruf zur Laufzeit.

3.3 Weiterführende Konzepte objektorientierter Programmierung

Die im folgenden dargestellten Konzepte gehen über das Grundlegende hinaus. Sie bilden zum einen eine Verfeinerung von etwas Bekanntem, zum anderen behandeln sie auch Neues. Aufgrund der Abhängigkeit von der gewählten Implementierungssprache werden sie hier nur in aller Kürze dargestellt.

3.3.1 Verschiedene Formen der Vererbung

Als eines der Basiskonzepte wurde die Vererbung behandelt, die die Modifikation und Erweiterung vorhandener Klassen ermöglicht. Im folgenden werden drei Varianten dieses Konzepts vorgestellt:

1. Selektive Vererbung:
   Die Grundform der Vererbung wie sie bereits beschrieben wurde, besteht darin, daß dem Erben alle Instanzvariablen und Methoden der Oberklasse zur Verfügung stehen. Von diesem Erbe ausgehend können Erweiterungen oder Redefinitionen eingebracht werden. Wird von dieser einfachen Regel abgegangen, so soll das als **selektive Vererbung** bezeichnet werden.

   Für die Selektion gibt es mehrere Möglichkeiten:

   - Die Selektion kann bei der Ober- oder der Unterklasse liegen. Liegt sie bei der Oberklasse, so besitzt diese private Methoden, die ausschließlich innerhalb der Klasse bekannt sind. Sie erscheinen also weder in der Schnittstelle zum Kunden, noch in der zu den Unterklassen. Dies beinhaltet den Vorteil, daß hier Dinge eingebracht werden können, die ohne Beeinträchtigung einer abhängigen Klasse (Kunde oder Erbe) jederzeit wieder geändert oder entfernt werden können.

     Diese Selektion wiegt die Schwäche der normalen uneingeschränkten Vererbung, daß nämlich die Unterklassen viele Details der Oberklasse sehen und diese daduch gebunden wird (vgl. Abschnitt 3.2.4), etwas auf.

   - Eine andere Möglichkeit der Selektion durch die Oberklasse besteht darin, daß die Redefinition von Methoden ganz oder teilweise verboten wird. Dies verspricht eine Verbesserung der Laufzeiteffizienz, da bei einem Methodenaufruf der Zeitaufwand für die Bestimmung des auszuführenden Rumpfes wegfällt.

   - Schließlich ist auch die Einschränkung möglich, daß die Oberklasse die Unterklassen festlegen darf, daß also nicht jede beliebige Klasse die Vererbungshierarchie erweitern darf.

2. Multiple Vererbung:
Die selektive Vererbung erweitert das Vererbungskonzept durch eine "testamentarische" Einschränkbarkeit. Eine Erweiterung in eine andere Richtung stellt die multiple oder mehrfache Vererbung dar. Kurz gesagt besteht sie darin, eine Klasse als gemeinsame Unterklasse mehrerer Oberklassen zu definieren.

   Die Klassenhierarchie besteht nicht mehr aus einer Menge von Bäumen, wie bei der einfachen Vererbung, sondern aus einem gerichteten azyklischen Graphen. Die Unterklasse erhält als (erweiter- und modifizierbares) Erbe sämtliche Instanzvariablen und Methoden ihrer Oberklasse. Die dem Programmierer bei der Implementierung zu Gebote stehenden Möglichkeiten werden stark erweitert.

   Wie bereits oben dargestellt sollte die Vererbung derart eingesetzt werden, daß den Unterklassen Untermengen in der realen Welt entsprechen. Bei der multiplen Vererbung bedeutet das, daß dem Erben eine (durch die Unterklassenbildung definierte) Untermenge des Durchschnitts aller Oberklassen entspricht.

3. Vererbung und Export:
Erbt eine Klasse von einer anderen Klasse, so übernimmt sie im einfachsten Fall alle dort definierten Methoden und Attribute, und zwar unabhängig davon, ob sie exportiert, d. h. vom Kunden sichtbar, oder privat sind. Es gibt nun zwei Möglichkeiten: Entweder kann der Erbe vollkommen unabhängig von der Oberklasse definieren, welche Methoden exportiert werden und welche nicht, oder er unterliegt gewissen Einschränkungen.

   Der erste Fall (Möglichkeit des selektiven Exports geerbter Methoden) hat den Vorteil, daß die Konzepte der Vererbung und des Exports voneinander unabhängig ("orthogonal") sind.

   Im zweiten Fall besteht eine sinnvolle Regel darin, daß geerbte Methoden, die in der Oberklasse exportiert wurden, auch in der Unterklasse exportiert werden müssen, der Erbe sein Methodenangebot also höchstens erweitern darf. Diese Regel folgt aus unserer Interpretation der Unterklasse als Untermenge: Ein Objekt der Unterklasse gehört auch zur Oberklasse und muß daher auch alle für die Oberklasse gültigen Methodenaufrufe verstehen.

3.3.2 Metaklassen

Eine **Metaklasse** ist eine Klasse, deren Instanzen wieder Klassen sind; d. h. eine Klasse wird bei diesem Konzept selbst zur Instanz. Metaklassen sind z. B. in SMALLTALK eingeführt. Speziell dort wird pro Klasse vom System automatisch eine Metaklasse erzeugt. Sie enthält die zugehörigen Klassenvariablen und -methoden, wobei eine **Klassenmethode** das Verhalten von Operationen beschreibt, die ausschließlich einer Klasse und nicht den von ihr erzeugten Instanzen, zugeordnet sind. Hierunter fällt z. B. die Instantiierung von Ob-

jekten. **Klassenvariablen** sind die globalen Variablen bzgl. einer Klasse. Sie sind von allen Instanzen der Klasse lesend zugreifbar. Die Werte dieser Variablen sind für alle Instanzen einer Klasse identisch.

Eng verwandt mit dem Metaklassenkonzept ist die Vorstellung der Klassen als Objekte. Die Klassenmethoden werden dann von diesen Objekten ausgeführt. Bei dieser Sichtweise sind Metaklassen Klassen, deren Instanzen wiederum Klassen sind. Sie unterscheiden sich von "gewöhnlichen" Klassen weiter dadurch, daß es von ihnen nur je ein Objekt, nämlich die zugehörige "gewöhnliche" Klasse gibt. Der Vorteil dieses Ansatzes ist die konzeptionelle Einfachheit und Einheitlichkeit ("Alles ist ein Objekt"). Außerdem ermöglicht er es, mit Klassen als Ganzes zu arbeiten. So kann z. B. einfach überprüft werden, ob zwei Objekte derselben Klasse angehören. Andererseits kann dieser Gedanke nicht konsequent durchgeführt werden; denn dann müßten auch die Metaklassen als Objekte angesehen werden, die durch Meta-Metaklassen beschrieben werden.

Dies ist jedoch gerade der springende Punkt des Konzeptes kognitiver Schemata (vgl. /14/). Dessen zentrales Konstrukt besteht eben genau in der Tatsache, daß bis zu einer gewissen Strufe Schemata über Schemata gebildet werden können.

3.3.3 Objektverwaltung

Bei der Objektverwaltung geht es um die bestmögliche Nutzung der beschränkten Ressource Speicher. Bei prozedurorientierten Softwaresystemen ist die Speicherverwaltung prinzipiell sehr einfach: Beim Aufruf einer Prozedur wird Speicherplatz auf einem Stack angelegt, und dieser wird beim Verlassen in toto wieder freigegeben.

Komplizierter wird es, wenn durch explizites Allokieren Speicherplatz auf einem Heap angefordert wird. Dann ist es häufig nicht mehr einfach zu überblicken, wann die Freigabe erfolgen kann, da Verweise auf die Heap-Elemente dupliziert und weitergegeben werden können. Diese Heap-orientierte Speicherverwaltung ist nun der Normalfall beim objektorientierten Ansatz, so daß die Objektverwaltung dort recht aufwendig sein kann.

Ein typisches objektorientiertes Softwaresystem erzeugt während der Laufzeit immer wieder neue Objekte, andererseits "sterben" aber auch Objekte. Ein Objekt ist "tot", wenn es bei keinem "lebenden" Objekt einen Verweis auf es gibt; es kann daher nie wieder aktiv werden, d. h. eine Methode ausführen. Der komplementäre Begriff "lebendes Objekt" ist rekursiv definiert:

- Das gerade aktive Objekt und alle in der gerade aktiven Methode lokal definierten Objekte leben.
- Das gleiche gilt für die (in)direkten Aufrufer der gerade aktiven Methode.
- Alle durch Instanzvariablen eines lebenden Objekts angesprochenen Objekte leben ebenfalls.

Zu jedem lebenden Objekt gibt es eine Verweiskette, die ihren Anfang bei dem gerade aktiven Objekt hat. Da tote Objekte nie wieder aktiviert werden können, kann ihr Speicherplatz für neue Objekte wiederverwendet werden. Darauf kann nur verzichtet werden, wenn insgesamt nur wenige Objekte erzeugt werden oder sehr viel Speicherplatz zur Verfügung steht.

Kann oder soll das Problem des Speicherplatzes nicht ignoriert werden, so gibt es zwei Lösungsansätze, die sich darin unterscheiden, wer die Objektverwaltung durchführt:

(1) Im ersten Fall ist es der Programmierer, der durch "new" Speicherplatz anfordert und ihn durch "free" wieder an die Heapverwaltung zurückgibt. Mit dieser Lösung sind zwei Probleme verbunden:

- Es ist nicht immer einfach, festzustellen, wann ein Objekt nicht mehr benötigt wird. Dieses Problem setzt sich rekursiv fort.
- Enthält die Objektverwaltung einen Fehler, so wird entweder nicht mehr benötigter Speicherplatz nicht zurückgegeben oder es wird Speicherplatz freigegeben, der noch nicht "tot" ist (Problem der dangling Pointers).

Eine Variante dieses Verfahrens besteht darin, daß freiwerdender Speicherplatz nicht an die allgemeine Heapverwaltung zurückgegeben, sondern auf einem Stack gesammelt und wiederverwendet, wird.

(2) Beim zweiten Lösungsansatz ist das Laufzeitsystem für die Objektverwaltung verantwortlich, der Programmierer braucht sich um nichts zu kümmern. Da das Laufzeitsystem das Speicherabbild der Objekte kennt, ist es in der Lage, die toten Objekte zu bestimmen und ihren Platz freizugeben.

3.3.4 Konstruktionsprinzip

Durch das Konzept der Vererbung wird es möglich, Modifikationen und Erweiterungen bestehender Klassen einzubringen, ohne diese selbst anzutasten oder Quelltext zu duplizieren, also Software inkrementell zu entwickeln. Besonders die Wiederverwendung wird dadurch sehr gefördert.

Wie bereits in Abschnitt 3.2.4 dargestellt wurde, ist eine unkritische Anwendung der Vererbung nicht unproblematisch. Aufgrund der Nachteile des Vererbungskonzeptes wird es in Erwägung gezogen, das sogenannte **Konstruktionsprinzip** gegenüber dem Vererbunskonzept zu favorisieren.

Das Konstruktionsprinzip besagt das folgende: Soll eine Klasse K modifiziert oder erweitert werden, so wird eine neue Klasse L nicht als Erbe von K definiert, vielmehr enthält L eine Instanzvariable der Klasse K. Die neue Klasse L ist also nicht Erbe, sondern Kunde von K. Dabei bleibt die Datenkapselung vollständig erhalten, und die Semantik kann beliebig ge-

ändert werden. Insbesondere spricht bei diesem Ansatz nichts dagegen, von K exportierte Methoden in L verborgen zu halten.

Ein Vorschlag besteht darin, die Vererbung zu verwenden, wenn eine Inklusion zwischen neuer und alter Klasse besteht; in allen anderen Fällen scheint es besser, die alte Klasse als Kunde der neuen Klasse zu benutzen. Damit können die unzweifelhaften Vorteile der Vererbung genutzt werden, ohne allzuviele der Nachteile in Kauf nehmen zu müssen.

3.3.5 Generische Klassen

Die Vererbung wurde hier als ein Konzept vorgestellt, das die Beschreibung von Gemeinsamkeiten und Differenzen verschiedener Klassen gestattet: Übereinstimmende Methoden sind Teil der (abstrakten) Oberklasse, die Unterklassen enthalten nur die voneinander differierenden Methoden. Diese knappe Beschreibung ist aber nur möglich, wenn die Außenansicht der Klasse, d. h. die Semantik ihrer Methoden, zumindest teilweise übereinstimmt, wobei die Implementierungen unterschiedlich sein können.

Schwierigkeiten gibt es dagegen insbesondere dann, wenn die Außenansicht der Klassen zwar gleichartig ist, die Methoden aber Argumente unterschiedlicher Klassen besitzen (z. B. Matrizen über verschiedenen Ringen). Die Implementierungen der Methoden in den beiden Klassen können dabei - bis auf die unterschiedlichen Typnamen - vollständig übereinstimmen.

Derartige Gemeinsamkeiten lassen sich mit **generischen Klassen** fassen. Diese sind strenggenommen keine Klassen zur Instantiierung von Objekten, sondern Schablonen zur Ausprägung gewöhnlicher, d. h. nicht-generischer Klassen. Die Schablone kann mit generischen formalen Parametern parametrisiert sein. Bei der Ausprägung werden diese durch Klassennamen ersetzt. Damit werden - abhängig von den Parametern - unterschiedliche Klassen erzeugt, deren Methoden jedoch durch gleiche Algorithmen beschrieben sind.

Sinnvolle Einsatzmöglichkeiten für generische Klassen sind z. B. Standardklassen, Stacks, Listen, Bäume, Mengen und dergleichen.

3.4 Resümee

Die durch die Modellierungsperspektive bedingte konzeptionelle Nähe von Programm und Objekt eines problemrelevanten Realitätsausschnittes bietet viele Vorteile im Vergleich zu konventionellen Programmen, bei denen die kognitive Distanz zu der einer Problemklasse adäquaten Terminologie sehr viel größer ist.

Die objektorientierte Sichtweise erleichtert die Reduktion der Komplexität eines Problembereiches, durch ihre schrittweise abstrahierend Vorgehensweise. Dies soll jedoch nicht bedeuten, daß der objektorientierte Ansatz ein Allheilmittel darstellt. Vielmehr ist dieses Verfahren in erster Linie für Problemstellungen relevant, für die die objektorientierte Vorgehensweise einer "natürlichen" Sichtweise der Welt entspricht.

Ein Problem objektorientierter Systeme besteht in den z. B. durch indirekte Referenzen verursachten Ineffizienzen bei der Speicherung und Verarbeitung von Objekten. Des weiteren benötigt man zur Entwicklung objektorientierter Software eine geeignete Entwicklungsumgebung, die z. B. eine Benutzerschnittstelle zur komfortableren Programmierung umfaßt.

Will man das Paradigma objektorientierter Softwareentwicklung zur Realisierung eines auf der Schematheorie basierenden Systems heranziehen, so besteht das größte Manko darin, daß es in der objektorientierten Programmierung nicht möglich ist Klassen zur Laufzeit zu erzeugen.

Die Implementierung dieser Strukturen soll mittels einer eigens dafür konzipierten objektorientierten Sprache erfolgen, die es beispielsweise erlaubt, neue Klassen zur Laufzeit zu erzeugen.

4 Allgemeine Problemlösungsstrategien

Die hier vorgestellten Problemlösungsstrategien sollten als mögliche Zutaten und nicht als komplette Lösungen verstanden werden. Beim Erstellen spezieller Problemlösungssysteme sollte man niemals ausschließlich eine Strategie verwenden; stattdessen sollte man verschiedene Strategien zusammenmischen, um eine eigene, dem vorliegenden Problem angepaßte Zusammensetzung zu entwickeln. Im folgenden wird eine Übersicht über die wichtigsten elementaren Problemlösungsstrategien gegeben.

4.1 Beschreiben und Vergleichen

Die Problemlösungsstrategie Beschreiben und Vergleichen läßt sich am einfachsten am Beispiel geometrischer Analogieprobleme verdeutlichen. Geometrische Analogieprobleme sind typisch für menschliche Intelligenztests. Das Problem besteht darin, unter mehreren gegebenen Figuren eine Zielfigur X so auszuwählen, daß sich beim Übergang von einer Figur A zu einer Figur B und von einer Figur C nach X eine größtmögliche Ähnlichkeit ergibt. Mit anderen Worten: Man möchte die Regel finden, die angibt, wie C zu X wird und dabei der Regel am ähnlichsten ist, die bestimmt, wie A zu B wird.

Die Lösung für solche Probleme liegt in einer guten Formulierung der Regeln. Wenn man erst einmal solche Formulierungen hat, dann wird die Bearbeitung des Problems zu einer einfachen Angelegenheit, Regelbeschreibungen miteinander zu vergleichen und dabei ein gewisses Maß an Ähnlichkeit beizubehalten.

Gute Repräsentationen sind für die Behandlung diverser Probleme, wie z. B. das Verstehen geometrischer Analogien und das Bewegen geometrischer Körper wesentlich. Bei adäquater (guter) darstellerischer Kraft sind diese beiden Aufgaben einfach zu lösen. Ein Fehlen dieser Kraft verleitet beim Herangehen an die Problemlösung zu törichtem ad hoc Denken.

Gute Repräsentationen stellen die richtigen Dinge explizit heraus und decken natürliche Zwangsbedingungen auf.

4.2 Zielreduktion (Teilzielbildung)

Anstatt ein Problem auf einmal zu lösen, ist es häufig vorteilhaft Teilprobleme zu bilden, diese zu lösen und deren Lösungen anschließend zusammenzufügen und dieses Prinzip rekursiv anzuwenden.

Die graphische Darstellung der Beziehungen zwischen Zielen und Teilzielen kann in einem Graphen erfolgen, wobei die Ziele in einem **Zielbaum** (goal tree) angeordnet werden, die Knoten des Zielbaumes entsprechen den Zielen.

Wenn ein Ziel in Angriff genommen wird, indem mehrere weitere Teilziele geschaffen werden, dann werden diese weiteren Ziele direkt darunter angeordnet und mit Kanten verbunden. Die Ziele darunter werden in Bezug auf das Ziel darüber unmittelbare Teilziele oder Söhne genannt. Das obere Ziel wird in Bezug auf die Ziele darunter unmittelbar übergeordnetes Ziel oder Vater genannt. Der oberste Knoten ohne Vater ist das Gesamtziel oder Wurzelknoten.

Einige Ziele können nur dann erreicht werden, wenn alle ihre unmittelbaren Teilziele erreicht wurden. Die Knoten, die diesen Zielen entsprechen werden daher UND-Knoten genannt und werden durch Bögen in ihren Zweigen gekennzeichnet. Alle anderen Ziele werden erreicht, wenn irgendeines ihrer unmittelbaren Teilziele erreicht wurde. Die diesen Zielen entsprechenden Knoten werden daher ODER-Knoten genannt und bleiben unmarkiert.

Da Zielbäume immer eine Mischung aus UND- und ODER-Knoten darstellen, werden sie häufig UND/ODER-Bäume genannt. Das Verfahren Ziele über Teilzielbildung zu erreichen wird auch als **Zielreduktion** oder gleichbedeutend als **Problemreduktion** bezeichnet.

4.3 Propagierung natürlicher Beschränkungen

Für die Propagierung natürlicher Beschränkungen kann man die folgende Subklassifikation treffen.

4.3.1 Propagierung symbolischer Beschränkungen

Verfahren zur Propagierung symbolischer Beschränkungen kann man beispielsweise zur Analyse von Linienzeichnungen verwenden.

Hierbei bestimmen die symbolischen Beschränkungen die physikalische Interpretation der Linien, da in der Realität z. B. nur bestimmte Kanten- und Eckentypen miteinander in einem Kontext auftreten können.

4.3.2 Propagierung numerischer Beschränkungen

Ein weit verbreitetes Beispiel der Propagierung numerischer Beschränkungen in Netzen sind Tabellenkalkulationsprogramme.

Die numerischen Beschränkungen werden durch Formeln, deren Variablen die Einträge bestimmter Felder (Zellen) der Tabelle referenzieren und deren Operationen in einem Additions-Multiplikationsnetz repräsentiert werden können, dargestellt.

Bei der computergestützten Analyse von Bildern ist die Propagierung numerischer Beschränkungen in Matrizen einsetzbar. In der das Bild repräsentierenden Matrix nutzt man dabei aus, daß sich relevante Oberflächeneigenschaften fast überall nur langsam verändern. Auf der Oberfläche können sich Entfernung und Richtung zwar an den Kanten schnell verändern; Kantenpunkte stellen jedoch nur eine relativ kleine Anzahl der Punkte einer Bildmatrix dar. Von sich langsam ändernden Eigenschaften sagt man, daß sie eine Glättungsbeschränkung erfüllen. Glättungsbeschränkungen dieser Art können durch Relaxationsmethoden, die aus der Mathematik entnommen wurden, erreicht werden.

Das allgemeine Ziel der Methode der Propagierung numerischer Beschränkungen besteht darin, diejenigen Werte zu ermitteln, die überall mit einigen vorgegebenen Beschränkungen verträglich sind.

Da Propagierungsmethoden nur auf wenige Dinge bei enger Nachbarschaft einwirken, sagt man, daß diese Propagierungsmethoden lokale Berechnungen durchführen. Wenn Beschränkungen überall erfüllt werden, so spricht man von globaler Verträglichkeit. Damit besteht der Sinn der Propagierung numerischer Beschränkungen im Erreichen globaler Konsistenz durch lokale Berechnungen.

Die Propagierung von Beschränkungen zur Lösung numerischer oder symbolischer Probleme hat die zwei Haupteigenschaften:

- Die Verfahrensvorschriften sind einfach und leicht verständlich.
- Die erforderlichen Berechnungen können parallel auf schnellen Mikroprozessoranlagen durchgeführt werden.

4.3.3 Abhängigkeitsgesteuerte Rückziehverfahren

Gegeben sei ein Netzwerk als graphische Darstellung verschiedener "Handlungsszenarien", die auch Beschränkungen enthalten können. Alternativen bzw. Auswahlmöglichkeiten seien durch Auswahlfelder und auferlegte oder natürliche Beschränkungen durch Grenzwertfelder repräsentiert.

Auf dieser Basis soll nun ein Plan zur Erreichung eines gegebenen Zieles erstellt werden. Ein solcher Plan ist nur dann akzeptierbar, wenn sich die an den Grenzwertfeldern anliegenden Werte innerhalb der auferlegten oder natürlichen Beschränkungen befinden.

Gelingt die Problemlösung (in diesem Fall die Plangenerierung) nicht ad hoc, so muß der somit fehlerhafte Plan überarbeitet werden. Eine Möglichkeit dies zu tun besteht darin, die zuletzt getroffene Wahl und ihre Folgen zurückzunehmen, bei diesem Auswahlpunkt eine Alternative zu wählen und wieder fortzufahren, d. h. ein **Backtracking** auszulösen.

Der ganze Prozeß ähnelt dem, was ein Individuum tut, wenn es sich durch ein Labyrinth hindurcharbeitet; gelangt es in eine Sackgasse, so kehrt es wieder um und geht bei der nächsten Möglichkeit wieder vorwärts.

Jener Teil des systematischen Vorgehens, der auf Sackgassen reagiert, wird **chronologisches Rücksetzen** (chronological backtracking) genannt, um zu betonen, daß alles mit dem zeitlichen Rückziehen rückgängig gemacht wird.

Das sich beim chronologischen Rückziehen ergebende Problem besteht darin, daß unter Umständen viele der zurückgenommenen Entscheidungen mit der Sackgasse nichts zu tun haben. Dadurch kann das chronologische Rückziehen uneffektiv werden.

Eine andere Möglichkeit den fehlerhaften Plan zu überarbeiten, besteht darin, relevante Entscheidungen (im Bezug auf die Sackgasse) zurückzunehmen.

Zur Identifizierung relevanter Entscheidungen muß man sich nur merken, wie durch Propagierung von den Auswahlfeldern aus die die Sackgasse ankündigenden Grenzwertfelder erreicht werden.

Das Verfahren der Identifizierung relevanter Auswahlmöglichkeiten könnte **abhängigkeitsgesteuertes Rückziehen** (dependency-directed backtracking) genannt werden, um hervorzuheben, daß die zurückzunehmenden Entscheidungen solche sind, von denen eine Sackgasse abhängt. Das Verfahren wird auch **nichtchronologisches Rückziehen** (nonchronological backtracking) genannt, um hervorzuheben, daß nicht die Zeit bestimmt, welche Entscheidungen zurückzunehmen sind.

Der Vorteil des nichtchronologischen Rückziehens besteht darin, daß nur relevante Entscheidungen zurückgenommen werden. Damit ist das nichtchronologische Rückziehen ein wirksames Mitel, um kompatible Auswahlmöglichkeiten herauszufinden, solange es eine Möglichkeit gibt, über Abhängigkeiten hinweg den Weg zurückzuverfolgen, um relevante Entscheidungspunkte zu finden.

4.4 Suchprobleme

Suchprobleme sind allgegenwärtig und treten in allen Teilgebieten der Künstlichen Intelligenz auf. Im Gegensatz zu den im Abschnitt Zielreduktion untersuchten Entscheidungen sind diese bei Suchproblemen von Natur aus in irgendeiner Weise geordnet.

Für das Suchen gelten die folgenden allgemeinen Prinzipien:

- Mehr Wissen bedeutet weniger Suchen.
- Suchen beinhaltet die Gefahr, daß man für die Problemlösung am falschen Punkt ansetzt. Da man im allgemeinen mit vielen Aufgaben gleichzeitig beschäftigt ist, ist

die Verbesserung der Suchprozedur selten das richtige. Häufiger ist es richtig, das Verständnis des Problems zu verbessern und damit die Notwendigkeit des Suchens zu reduzieren.

Die im folgenden beschriebenen Suchalgorithmen kann man wie in Abb. 4 dargestellt kategorisieren. Sie werden am Beispiel der Suche nach einem Pfad in einem gegebenen Netzwerk mit den üblichen Bezeichnungen nacheinander behandelt.

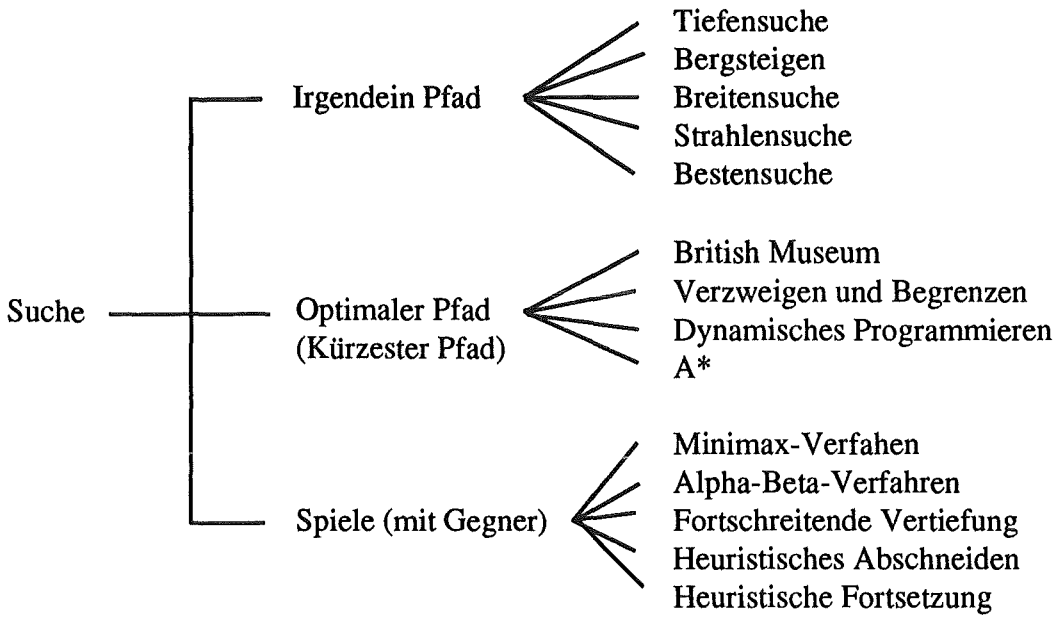

Abbildung 4

4.4.1 Grundlegende Prozeduren zum Finden eines Pfades

Wenn der gesuchte Pfad nur einmal durchlaufen werden soll oder wenn es schwierig ist, einen Pfad durch das Netz hindurchzubahnen, dann ist es angebracht, mit irgendeinem gefundenen Pfad zufrieden zu sein, auch wenn dieser nicht optimal ist.

Aus Gründen der Anschauung ist es bei der Beschreibung der einzelnen Suchstrategien hilfreich, auf der Grundlage eines gegebenen (ungerichteten) Netzes, abhängig von der ausgeführten Suchstrategie, den entsprechenden Suchbaum zu betrachten.

4.4.1.1 Tiefensuche

Dieses Suchverfahren taucht tief in den Suchbaum hinein. Es basiert auf der Annahme, daß ein Weg so gut wie jeder andere ist und daher ohne große Umstände an jedem aufgesuchten Knoten eine Alternative gewählt werden kann, um in diese Richtung fortzufahren. Andere

Alternativen auf der gleichen Ebene werden völlig ignoriert, solange es Hoffnung gibt, das Ziel mit der eingeschlagenen Richtung zu erreichen.

Tiefensuche ist gut geeignet, wenn Sackgassen nicht zusehr in die Tiefe des Suchbaumes gehen. Der Nachteil der Tiefensuche besteht darin, daß sich diese Strategie in der Tiefe eines Suchbaumes festfahren kann, so daß damit der Aufwand sehr hoch wird. Da kein Überblick über die Teilbäume besteht, ist es schwierig "intelligente" Teile zu integrieren.

4.4.1.2 Bergsteigen

Die Effektivität beim Suchen kann sich beträchtlich erhöhen, wenn es eine Möglichkeit gibt, Entscheidungen zu ordnen, so daß die vielversprechendsten zuerst untersucht werden.

Um sich beim Bergsteigen durch einen Baum aus Pfaden hindurchzubewegen, geht man wie bei der Tiefensuche vor, ordnet aber die Entscheidungen gemäß einer heuristischen Messung der verbleibenden Entfernungen bis zum Zielknoten, z. B. die Luftlinienentfernungen in einer Karte des Straßennetzes.

Eine Form des Bergsteigens wird ebenfalls bei der Parameteroptimierung genutzt. Das gegebene Problem entspricht dann einer Abstraktion, in der es anpaßbare Parameter und eine Möglichkeit, die mit jeder beliebigen Menge von Parameterwerten verbundene Qualität zu messen, gibt. An Stelle eines expliziten Ziels terminiert der Suchalgorithmus, wenn ein Knoten erreicht wird, so daß alle Nachfolger Meßergebnisse niedriger Qualität liefern.

Um sich durch eine Anordnung von Parameterwerten mittels parameterorientiertem Bergsteigen hindurchzuarbeiten, geht man einen Schritt in jede der vorgegebenen Richtungen, dann geht man zur besten gefundenen Alternative und wiederholt dieses Verfahren, bis man einen Punkt erreicht hat, der bessere Ergebnisse liefert, als alle umgebenden durch Ein-Schritt-Versuche erreichbaren Punkte.

Obgleich die Parameteroptimierung durch Bergsteigen einfach ist, ist sie mit verschiedenen Problemen belastet. Es sei hier stellvertretend nur das Vorgebirgs-Problem erwähnt, daß sich mit dem Auffinden optimaler Punkte lokaler, jedoch nicht globaler Art befaßt.

Bergsteigen ist gut geeignet, wenn es ein natürliches Maß für die Zielentfernung gibt und wenn es unter den erfolgversprechenden Entscheidungsmöglichkeiten an jedem Entscheidungspunkt wahrscheinlich eine gute Entscheidung gibt.

4.4.1.3 Breitensuche

Dieses Suchverfahren dringt gleichmäßig im Suchbaum vor. Breitensuche hält unter allen Knoten auf einer bestimmten Ebene Ausschau nach dem Zielknoten, ehe zum weiteren Vordringen die Nachfolger dieses Knoten verwendet werden.

Breitensuche ist sogar in Bäumen anwendbar, die unendlich tief oder tatsächlich unendlich sind.

Breitensuche ist gut geeignet, wenn die Anzahl der Entscheidungsalternativen nicht zu groß ist. Andererseits verschwendet man mit Breitensuche Zeit, wenn alle Wege mit mehr oder weniger gleicher Tiefe zum Zielknoten führen.

4.4.1.4 Strahlensuche

Die Strahlensuche geht, ebenso wie die Breitensuche, ebenenweise vor. Hierbei expandiert sie mehrere Teilwege und entfernt die restlichen. Im Unterschied zur Breitensuche geht die Strahlensuche nur von den besten n (n ist eine vorgegebene natürliche Zahl) Knoten in jeder Ebene weiter. Die anderen Knoten werden ignoriert.

Daher bleibt die Anzahl der untersuchten Knoten leicht zu handhaben, auch wenn es viele Verzweigungen gibt und die Suche stark in die Tiefe geht.

Die Strahlensuche ist gut geeignet, wenn es ein natürliches Maß für die Zielentfernung gibt und wenn es unter den erfolgversprechenden Teilwegen auf allen Ebenen wahrscheinlich einen guten Weg gibt. Sie kann aufgrund der Beschränkung auf eine vorgegebenen Anzahl von Teilwegen beim Suchen nach legitimen Wegen versagen.

4.4.1.5 Bestensuche

Diese Suchstrategie expandiert den besten Teilpfad. Bei der Bestensuche geht die Vorwärtsbewegung von bisher besten, noch nicht expandierten Knoten aus, unabhängig davon, wo sich dieser im teilweise expandierten Baum befindet.

Anschaulich gesprochen arbeitet die Bestensuche wie eine kooperierende Bergsteigermannschaft, die nach dem höchsten Punkt in einer Gebirgskette sucht. Die Bergsteiger dieser Bergsteigermannschaft stehen in Funkverbindung und schicken zu jedem Zeitpunkt die höchste Teilmannschaft weiter und teilen an Weggabelungen die Teilmannschaften in weitere Teilmannschaften auf.

Die von der Bestensuche gefundenen Wege sind wahrscheinlich kürzer als jene, mit anderen Methoden gefundene Wege, da die Bestensuche immer von dem Knoten fortschreitet, der dem Ziel am nächsten scheint.

Wie das Bergsteigen erfordert auch die Bestensuche eine Sortierung. Diesmal muß jedoch die gesamte Liste sortiert werden.

Die Bestensuche ist gut geeignet, wenn es ein natürliches Maß für die Zielentfernung gibt und wenn ein guter Weg auf hohen Ebenen, d. h. am Anfang der Suche schlecht erscheinen könnte.

4.4.2 Die Suche nach dem besten Weg

Die Suche nach einem Weg soll nun in der Weise optimiert werden, daß der beste Weg in Bezug auf die Anzahl der zu traversierenden Kanten oder bzgl. einem anderen, beliebigen Kostenmaß abhängig von den Kantengewichten gefunden werden soll.

4.4.2.1 Der "British Museum Algorithmus"

Unter diesem, etwas scherzhaft bezeichneten Algorithmus versteht man einen Algorithmus, der alle möglichen Wege, unter Verwendung von Breiten- bzw. Tiefensuche bestimmt, um dann unter ihnen den Besten auszuwählen. Bei einem kleinen Suchbaum ist diese Suchstrategie sinnvoll, jedoch bei einem großen Suchbaum wird dieses Verfahren sehr schnell unpraktikabel, da sich die Anzahl der Wege exponentiell zur Anzahl der Verzweigungen bzgl. der Suchbaumtiefe verhält.

4.4.2.2 Verzweige und Begrenze

Dieses Suchverfahren expandiert den am wenigsten kostspieligen Teilweg. Die Verzweige-und-Begrenze-Methode arbeitet wie folgt: Während der Suche gibt es viele unvollständige Wege, die sich für eine weitere Betrachtung anbieten. Der kürzeste davon wird um eine Ebene erweitert, wobei so viele neue unvollständige Wege wie vorhandene Zweige geschaffen werden. Diese neuen Wege werden dann zusammen mit den verbleibenden alten Wegen betrachtet, und wiederum wird der Kürzeste erweitert. Dieses Verfahren wird solange wiederholt, bis der kürzeste unvollständige Weg länger als der kürzeste vollständige Weg ist. Durch dieses Abbruchkriterium wird ausgeschlossen, daß ein noch unvollständiger Weg kürzer wird, als der bereits gefundene "kürzeste Weg"; die lokal besten Entscheidungen führen global zu einem optimalen Ergebnis.

Da zur Erweiterung immer der kürzeste Weg gewählt wurde, kann der das Ziel erreichende Weg mit Sicherheit als optimal angesehen werden.

Bei ungünstiger Baumstruktur und ungünstigem Suchziel (z. B. alle Wege zu den Blättern haben ein kleineres Gesamtgewicht als der Gesuchte und das Suchziel ist ein Blatt) ist der Aufwand genau so groß, wie beim Britisch Museum Algorithmus.

Ist der Baum groß und können schlechte Wege sehr schnell ermittelt werden, so ist die Verzweige-und-Begrenze-Suche geeignet.

In einigen Fällen kann die Verzweige-und-Begrenze-Suche durch Verwendung von Schätzungen über die restlichen Entfernungen sowie unter Berücksichtigung bereits aufsummierter Entfernungen beträchtlich verbessert werden.

Im allgemeinen sind Schätzungen jedoch nicht perfekt. Überschätzungen können dazu führen, daß der optimale Weg für immer verlassen wird. Durch Unterschätzung kann der richtige Weg nicht übersehen werden, denn wenn eine Gesamtlänge durch wiederholtes Expandieren des Weges mit der kleinsten unterschätzen Länge gefunden wurde, ist keine

weitere Arbeit erforderlich, nachdem alle geschätzten Teilwegstrecken länger als eine vollständige Weglänge sind. Diese Schlußfolgerung ist jedoch nur unter der Prämisse, daß es sich garantiert um Unterschätzungen handelt, korrekt.

Natürlich ist diese Verbesserung um so effizienter, je besser die Unterschätzung der tatsächlichen Entfernung entspricht, denn wenn es überhaupt keine Differenz gibt, dann gibt es auch keine Möglichkeit den falschen Weg einzuschlagen.

Wenn es eine gute Schätzung des unteren Grenzwertes für die verbleibende Entfernung bis zum Ziel gibt, dann ist die Verzweige-und-Begrenze-Suche mit einer Schätzung geeignet.

4.4.2.3 Dynamisches Programmieren

Die Suchstrategie des dynamischen Programmierens wird durch folgende allgemeine Regel charakterisiert: Man nimmt an, daß der Weg vom Ausgangspunkt S (der Wurzel des Suchbaumes) bis zu einem Zwischenpunkt I (einem Knoten) die Wahl der Wege von I bis zum Ziel G (einem Knoten oder einem Blatt) nicht beeinflußt. Dann ist die minimale Entfernung von S nach G über I gleich der Summe aus der minimalen Entfernung von S nach G über I gleich der Summe aus der minimalen Entfernung von S nach I und der minimalen Entfernung von I nach G.

Daher gilt für das Prinzip der dynamischen Programmierung, daß bei der Suche nach dem besten Weg von S nach G alle Wege von S nach irgendeinem Zwischenpunkt I außer dem mit der minimalen Länge ignoriert werden können.

Die dynamische Programmierung ist dann gut geeignet, wenn viele Wege gemeinsame Knoten erreichen.

4.4.2.4 A*

Der Algorithmus A* ist eine Verzweige-und-Begrenze-Suche mit einer Schätzung der verbleibenden Entfernung, kombiniert mit dem Prinzip der dynamischen Programmierung; d. h. A* vereinigt die in den Abschnitten 4.4.2.2 und 4.4.2.3 dargestellten Ideen.

Wenn die Schätzung der Restentfernung ein unterer Grenzwert der tatsächlichen Entfernung ist, dann liefert A* optimale Lösungen.

Wenn sowohl die Verzweige-und-Begrenze-Suche mit einer Schätzung als auch die dynamische Programmierung vorteilhaft sind, dann ist der Algorithmus A* geeignet.

4.4.3 Suchprobleme bei Spielen

Eine weitere Art der Suche wird bei Spielen wie Dame oder Schach durchgeführt. Die Knoten in einem Spielbaum (dieser entspricht dem Suchbaum in Spielsituationen) stellen dann natürliche Brettkonfigurationen dar und sind durch Zweige, die von einer Situation zu einer anderen führen, miteinander verbunden.

Es gibt hier eine neue Verflechtung dahingehend, daß die Entscheidungen abwechselnd von zwei als Gegner agierenden Individuen getroffen werden. Dies macht sich natürlich bei der Interpretation des Spielbaumes bemerkbar, indem die Ebenen des Spielbaumes abwechselnd jeweils einem Zug der Spieler entsprechen.

Welche Suchstrategien sind zur Bestimmung eines optimalen Zuges, unter Berücksichtigung der Gegenzüge, bis zu einer bestimmten Tiefe anwendbar?

- Der Britisch Museum Algorithmus scheidet aufgrund der kombinatorischen Explosion sicherlich aus.
- Es stellt sich die Frage, ob es eine Strategie gibt, die eine Suche steuern kann?
    - Die Beantwortung dieser Frage kann man dahingehend einschränken, daß es keine Möglichkeit gibt, die einzelnen Positionen einer Reihe von Brettstellungen zu bewerten. Man muß also zu einer etwas lokaleren Bewertungsstrategie übergehen.

Die zentrale Idee bei der Entwicklung von Suchstrategien für Spiele ist die **Minimax-Idee.**

Minimaximierung ist die Prozedur, bei der Schlüsse, was bei tieferliegenden Knoten des Suchbaumes zu tun ist, bestimmen, was bei höhergelegenen Knoten passieren sollte.

In dieser Strategie spiegelt sich im wesentlichen das abwechselnde Ziehen von Spieler und Gegenspieler, allerdings nur aus der subjektiven, gewinnorientierten Sicht eines der Beiden, wider. Der am Zug befindliche Spieler versucht mit seinem Zug sein Ergebnis zu maximieren und rechnet damit, daß sein Gegner in dessen Zug das somit erreichte minimieren, d. h. wenn möglich zunichte machen will. Als Qualitätszahl (Gütewert) dient hierbei eine statische Bewertung mit Hilfe eines Stellungsanalysators. Aufgrund der kombinatorischen Vielfalt läßt sich dieses Verfahren jedoch nur bis zu einer bestimmten Tiefe sinnvoll einsetzen.

Das **Alpha-Beta-Verfahren** hängt von der Tatsache ab, daß immer dann, wenn ein Spieler beim Minimax-Verfahen eine sichere Möglichkeit hat, festzustellen, daß ein Zug unheilvoll für ihn ist, keine Notwendigkeit besteht, diesen Zug weiter zu untersuchen. Somit wird (im Vergleich zum Minimax-Verfahren) zum einen der Suchbaum verkleinert und zum anderen die Anzahl der statischen Bewertungen verringert. Das Alpha-Beta-Verfahren erspart viel Arbeit, ohne die Gefahr heraufzubeschwören, den optimalen Spielzug zu übersehen.

Unter dem **Verfahren der fortschreitenden Vertiefung** versteht man eine Vorgehensweise, die z. B. diejenige eines Schachspielers unter Zeitdruck simuliert. Innerhalb einer vorgegebenen Zeit wird jede Situation im Spielbaum bis zur Tiefe 1, dann bis Tiefe 2, dann bis Tiefe 3 etc. analysiert, bis die Zeitbeschränkung erreicht ist.

Verbesserungen in puncto Reduzierung des Suchbaumes lassen sich durch heuristische Verfahren wie z. B. **heuristisches Abschneiden** realisieren. Sämtliche heuristischen Verfahren bringen jedoch die Gefahr mit sich, daß nicht der optimale Zug (z. B. bzgl. der Suchtiefe) gewählt werden könnte.

Das Verfahren der **heuristischen Fortsetzung** ist eine Möglichkeit den Horizonteffekt zu bekämpfen, um verzögernde Spielzüge, die das Unheil noch schlimmer machen, zu verhindern. Unter dem **Horizonteffekt** versteht man den Vorgang, der eintritt, wenn ein Spieler versucht durch verzögernde Spielzüge unabwendbare schlechte Ereignisse (z. B. den Verlust der Dame beim Schach) hinauszuschieben; beispielsweise wird durch ein als minderwertiger eingeschätztes Opfer (z. B. Bauernopfer beim Schach) ein unabwendbarer Verlust aus dem Gesichtsfeld verdrängt.

4.5 Die Mittel-Zweck-Analyse (Means-Ends-Analysis)

Die hier behandelte Problemlösungsmethode basiert auf der Unterscheidung eines aktuellen und eines Zielzustandes. Der aktuelle Zustand besteht aus einer Menge von Fakten, die das Problem und wo man sich innerhalb des Problemraumes (Verallgemeinerung des Suchbaumes) befindet, spezifizieren. Der Zielzustand ist der angestrebte Zustand nach dem Problemlösungsprozeß.

Die Mittel-Zweck-Analyse ist eine Methode für die Art von Problemen, bei denen Lösungsprozeduren gemäß ihrer Fähigkeit ausgewählt werden, die Differenz zwischen dem aktuellen Zustand und dem Zielzustand zu reduzieren.

Eine der bekanntesten, auf der Mittel-Zweck-Analyse basierenden Kontrollstrukturen ist der von Newell, Shaw und Simon in den sechziger Jahren entwickelte **General Problem Solver** (GPS).

Die zentrale Kontrollentscheidung des GPS wird auf der Basis der Mittel-Zweck-Analyse gefällt. Diese Problemlösungsstrategie wird im GPS in Verbindung mit einer Tiefensuche und Teilzeilbildung (Zielreduktion) angewandt.

4.6 Generieren-und-Testen von Systemen

Die Problemlösungsstrategie des Generierens-und-Testens wird am häufigsten verwendet, um Identifikationsprobleme, wie z. B. die Analyse von Massenspektrogrammen oder das Bildverstehen, zu lösen, die nicht mehr als einige Hundert mögliche Antworten enthalten.

Problemlöser dieser Art bestehen aus zwei Komponenten:

- Einem Generator zum Aufzählen der möglichen Lösungen und
- einem Tester, der die vom Generator vorgeschlagene Lösung bewertet, indem er sie entweder akzeptiert oder zurückweist.

Ein guter Generator sollte über drei Eigenschaften verfügen:

- Vollständigkeit, d. h. alle Lösungen werden generiert.
- Keine Redundanz, d. h. keine Lösung wird mehrfach vorgeschlagen.
- Informiertheit, d. h. er verwendet zusätzliche Informationen, um die Anzahl der möglichen Lösungen zu reduzieren.

In bezug auf Kontrollstrategien kann man sich an folgendem Leitspruch orientieren:

> Eine hochentwickelte Kontrollstrategie kann intelligente Prozeduren intelligenter, aber unintelligente Prozeduren nicht intelligenter machen.

4.7 Regelbasierte Systeme

Die meisten bisher entstandenen erfolgreichen Synthese-und-Analyse-Systeme stützen sich auf eine regelbasierte Problemlösungsstrategie.

Den Kern regelbasierter Problemlösungssysteme bilden WENN-DANN-Regeln. Die WENN-Teile (Antezedens) solcher Regeln beinhalten Kombinationen bekannter Fakten (bzw. allgemeiner Bedingungen) und die DANN-Teile (Konsequenz) geben neue Fakten an (bzw. allgemeiner Aktionen), die direkt von den bekannten Fakten abgeleitet werden können.

Dadurch wird ein regelbasiertes System zu einer Art **Deduktionssystem**; allgemeiner spricht man von einem System bestehend aus Bedingungs-Aktionen-Regeln.

Wenn alle Bedingungen in einer Regel durch die aktuelle Situation erfüllt sind, so sagt man die Regel ist **ausgelöst** (triggered). Wenn die Aktionen ausgeführt werden, so sagt man von der Regel, daß sie **gefeuert** hat (fired). Auslösen bedeutet nicht immer Feuern, denn die Bedingungen mehrerer Regeln können gleichzeitig erfüllt sein. In solchen Fällen muß eine **Konfliktlösungsstrategie** entscheiden, welche Regel tatsächlich feuern soll.

Hierfür gibt es eine Reihe von Möglichkeiten, wie z. B. die Bewertung der anwendbaren Regeln und deren anschließende Anordnung in einer Prioritätsliste.

Um Ableitungen über mehrere Schritte durchführen zu können, werden Ableitungsketten gebildet. Grundsätzlich gibt es zur Bildung und Abarbeitung von Ableitungsketten zwei Strategien:

- Vorwärtsverkettung:
  Bei der Vorwärtsverkettung gehen regelbasierte Systeme von bekannten Fakten aus und gelangen (deduktiv) zu neuen, abgeleiteten Fakten.

- Rückwärtsverkettung:
  Die zweite Möglichkeit ist die Rückwärtsverkettung, denn ein regelbasiertes System kann eine Schlußfolgerung hypothetisch ziehen und Antezedenz-Konsequenz-Regeln verwenden, um rückwärts zu hypotheseunterstützenden Fakten zu gelangen.

Die Frage, welche der beiden Ableitungsstrategien besser ist, kann man so allgemein nicht beantworten, da dies zum einen von dem zu lösenden Problem und zum anderen von den Regeln selbst abhängt.

Besteht das Ziel darin, daß alles, was von einer Reihe von Fakten abgeleitet werden kann aufgedeckt werden soll, so wird man Vorwärtsverkettung bevorzugen. Wenn andererseits das Ziel darin besteht, eine bestimmte Schlußfolgerung zu etablieren oder abzulehnen, so wird man Rückwärtsverkettung bevorzugen, um dadurch irrelevante Schlüsse und evtl. eine kombinatorische Explosion zu vermeiden.

Die graphische Darstellung von Antezedens-Konsequenz-Regelmengen kann durch **Inferenz-Netze** oder **UND/ODER-Bäume** erfolgen. Dies birgt jedoch die Gefahr, daß der Problemlösungsvorgang als reiner Suchprozeß in diesen Netzen betrachtet wird und damit mehr an einer Verbesserung der Suchprozedur "herumgespielt" wird, anstatt die Problemdarstellung zu verbessern.

In regelbasierten Systemen wird das Problem und nicht der Lösungsweg beschrieben. Sie erleichtern die Beantwortung von Fragen zu ihrem eigenen Verhalten, d. h. sie können ihre Vorgehensweise bis zu einem gewissen Grad begründen.

Die zur Identifikation verwendeten regelbasierten Systeme werden gewöhnlich in Bereichen eingesetzt, in denen Schlüsse selten sicher sind. Deshalb setzen die Entwickler von regelbasierten Systemen häufig eine Prozedur für eine Art Evidenzberechnung auf den grundlegenden Antezendens-Konsequenz-Mechanismus. Im allgemeinen assoziieren Prozeduren, die Evidenzwerte berechnen mit jedem Faktum eine Zahl zwischen 0 und 1. Diese Zahl, der **Evidenzwert**, soll wiedergeben, wie sicher eine Tatsache ist; 0 bedeutet, daß ein Faktum mit Sicherheit falsch und 1, daß es mit Sicherheit wahr ist.

Zur Berechnung von Evidenzwerten existieren bisher keine vollständig zufriedenstellenden Prozeduren, d. h. die hierbei auftretenden wahrscheinlichkeitstheoretischen Probleme sind noch nicht vollständig gelöst.

Den Vorteilen der (erzwungenen) homogenen Wissensdarstellung (vgl. auch Abschnitt 2.4), der stufenweise Wissenszunahme (durch Hinzufügen weiterer Regeln) und den mög-

lichen ungeplanten aber nützlichen Wechselwirkungen (als Seiteneffekte aus der Tatsache, daß Regeln, wann immer möglich angewandt werden können) stehen als Nachteile die flache Struktur (Schlußfolgerungen erfolgen nur auf einer Ebene), die einseitige Betrachtungsweise (Probleme werden nicht aus verschiedenen Perspektiven betrachtet) und der fehlende Zugang zu Schlußfolgerungen jenseits der Regeln gegenüber.

Unter dem in der experimentellen Psychologie verwendeten Begriff der Produktionssysteme wird die Rolle regelbasierter Systeme zur Modellierung des menschlichen Denkens untersucht.

4.8 Formale (klassische) Logik

Wie jede der bereits behandelten Problemlösungsstrategien hat Logik sowohl Vorteile, als auch Nachteile (/28/):

- Vorteile:
  - Die Ideen der Logik, die über Jahrhunderte hinweg gereift sind, sind kurz und bündig und die Logiker haben sich bis heute auf das Beweisen von Dingen konzentriert, was man mit Wissen tun kann.
  - Es gibt eine anerkannte, formale Definition der Syntax und Semantik. Durch Festlegung des Ableitunsbegriffes ist eine Automatisierung der Ableitung möglich.
  - Komplizierte Zusammenhänge lassen sich durch Zerlegung auf einfach Weise darstellen.
  - Die Trennung von Beschreibung und Bearbeitung erhöht die Übersichtlichkeit.
- Nachteile:
  - Die Kontrolle der Bearbeitung ist in der Regel nicht beschreibbar.
    (Eine Darstellung von Fakten über den Problemlösungsprozeß unterstützt z. B. der von Genesereth und Ginsberg in /7/ beschreibene MSR Interpreter.)
  - Die Beweise sind zumeist sehr lang und unübersichtlich.
  - Durch die Spezifizierung eines komplexen Problems kann eine große Menge unstrukturierter Formeln entstehen.
  - Nur in Spezialfällen sind Theorembeweiser effizient.
  - Vages, d. h. unsicheres Wissen kann in der klassischen Logik nicht ausgedrückt werden.

- Die Logik ist monoton, d. h. einmal abgeleitetes Wissen bleibt immer gültig, auch wenn das Wissen (konsistent) erweitert wird. Daß dies nicht immer wünschenswert ist, wird oft an dem Beispiel des Vogels gezeigt, für den abgeleitet wird, daß er fliegen kann, und für den dies nicht mehr abgeleitet werden darf, wenn bekannt wird, daß er einen Flügel gebrochen hat.

- Die Logik kann auch ein Prokrustesbett sein, denn die Konzentration auf Logik kann zur Konzentration auf die Mathematik der Logik führen, was die Aufmerksamkeit von nützlichen Problemlösungsverfahren, die der mathematischen Analyse widerstehen, ablenkt.

Als ein Sachgebiet konzentriert sich Logik auf die Verwendung von Wissen in einer unerbittlichen, beweisbar richtigen Weise; andere Problemlösungsstrategien konzentrieren sich auf das Wissen selbst.

Da die Thematik der Logik und speziell die Prädikatenlogik erster Ordnung im Abschnitt Wissensrepräsentation bereits aufgegriffen wurde, werden hier nur einige grundlegende Dinge, Logik betreffend angesprochen.

4.8.1 Logik als Theorembeweiser

Logik behandelt die Ableitung neuer Ausdrücke (Theoreme) aus Axiomen. Ein Theorem folgt logisch aus angenommenen Axiomen, wenn es eine Reihe von Schritten gibt, die das Theorem mit den Axiomen unter Verwendung von Inferenzregeln verknüpft. Die bekanntesten Inferenzregeln sind der **Modus ponens**, der **Modus tolens** und die diese beiden Modi umfassende **Resolution**.

Durch die Festlegung auf eine oder mehrere Inferenzregeln zum Theorembeweisen wird noch keine Aussage über eine mögliche Reihenfolge der Abarbeitung der Axiome getroffen; d. h. damit werden nur die Ableitungsregeln und nicht die Strategie zum Theorembeweisen vorgegeben.

Bei der Auswahl der Strategie bzw. bei der Generierung einer solchen steht man im wesentlichen vor den beiden folgenden Problemen:

- Kombinatorische Explosion des zugrundeliegenden Suchraumes.

- Eine Version des Halteproblems.

Ersteres ist damit begründet, daß die aus den ausgewählten Regeln generierten Suchbäume (zur weiteren Regelauswahl) ins unermeßliche wachsen und letzteres, daß dadurch ein Beenden der Suche (nach einem Beweis für das Theorem) nicht garantiert werden kann, solange es keinen solchen Beweis gibt (Semi-Entscheidbarkeit).

Das **Resolutionstheorembeweisen** verwendet die Resolution als Inferenzregel und Widerlegung als Strategie. Die Resolution erfordert eine Umwandlung von Axiomen und des negierten Terms in Klauselform (eine spezielle konjunktive Darstellung), dies ist jedoch immer möglich. Resolution erfordert ebenfalls einen Variablen-Substitutionsprozeß, der **Unifikation** genannt wird.

Logik als Problemlösungsstrategie ist ungeeignet für Probleme der folgenden Art:
Die Schreibweise der reinen Logik erlaubt nicht, solche Dinge wie heuristische Entfernungen (vgl. Abschnitt 2.2.4.1) auszudrücken oder Unterschiede festzustellen oder den Gedanken, daß eine spezielle Methode besonders schnell ist oder den Gedanken, daß ein bestimmtes Verfahren nur dann erfolgreich eingesetzt werden kann, wenn es nicht zu häufig angewendet wird. Theorembeweiser können solches Wissen nur dann darstellen, wenn dieses durch die Verwendung von Begriffen dargestellt wird, die nicht aus der reinen Logik stammen, wie z. B. den Greenschen Trick zur Herleitung eines Antwortterms im Resolutionsbeweis.

4.8.2 Logik zur Planung von Operatorsequenzen

Die bisher behandelten Axiome der reinen Logik treffen nur Aussagen darüber, was durch deren Anwendung auf ein Objekt mit diesem passiert; sie sagen nichts darüber aus, welche Veränderungen die anderen Objekte dadurch erfahren. Ohne zusätzliches Wissen gibt es keine Möglichkeit, für ein logikbasiertes Verfahren abzuleiten, welche Prädikatwerte beim Wechsel von einer Situation zu einer anderen unverändert bleiben; schließlich verändern sich einige. Dieses Dilemma wird als **Frameproblem** (vgl. /4/) bezeichnet und tritt bei logikbasierten Verfahren zur Planung von Operatorfolgen häufig auf.

Eine beliebte Methode zur Lösung des Frameproblems ist die Einführung sogenannter **Frameaxiome,** diese treffen Festlegungen darüber, wie Prädikate Operationen überleben.

Der Nachteil dieser Methode besteht darin, daß sie sehr aufwendig ist; im allgemeinen erfordert sie die Einführung einer ganzen Reihe von Frameaxiomen und damit eine Vergrösserung der Menge der Axiome inklusive der damit verbundenen Nachteile.

4.8.3 Logik in Beschränkungsnetzen

Bisher haben wir im Zusammenhang mit Logik die logischen Junktoren ($\vee$, $\wedge$, $\neg$, $\Rightarrow$) wie Funktionen behandelt, die Argumente akzeptieren und WAHR oder FALSCH liefern. Man kann logische Ausdrücke aber auch dazu verwenden, um ein Knotennetzwerk (jeder Knoten repräsentiert einen logischen Ausdruck, der sich auch aus Subausdrücken zusammensetzen kann) zu entwickeln, durch das Beschränkungen propagiert werden.

Für diesen Ansatz sprechen drei Gründe:

- Erstens stellt ein **logikorientiertes Netz aus Beschränkungen** die Tatsache heraus, daß das Theorembeweisen als eine Art der Ausnutzung von Beschränkungen gesehen werden kann.

- Zweitens kann man mit Beschränkungsnetzen beim Aufstellen oder Zurückziehen von Annahmen leichter verfolgen, was WAHR und was FALSCH ist. Deshalb sind Beschränkungsnetze mit dem Begriff **Wahrheitserhaltung** assoziiert.

  D. h. man kann in solchen Netzen leicht Begründungen für alle Schlußfolgerungen verfolgen, was den Umgang mit zurückgezogenen Annahmen erleichtert.

- Drittens können Beschränkungsnetze das Erkennen, wie abhängigkeitsgesteuertes, nichtchronologische Rückziehen (vgl. Abschnitt 4.3) die Widersprüche korrigieren helfen kann, die vorkommen, wenn Annahmen unverträglich sind. Die Methode des abhängigkeitsgesteuerten Rückziehens kann jedoch bei jeder Beweisprozedur verwendet werden und nicht nur bei solchen, die Beschränkungsnetze einbeziehen.

Die Problemlösungsstrategie der Propagierung von Beschränkungen hat jedoch auch ihre Grenzen:

- Diese Strategie ist keine vollständige Beweisprozedur, da es WAHR-Ausdrücke gibt, die sie nicht als WAHR beweisen kann.

- Sie befaßt sich nur mit der Aussagenlogik.

5 Problemlösungstypen in Expertensystemen

Um zu geeigneten Problemlösungsstrategien zu gelangen, ist es notwendig die Probleme nach verschiedenen Problemtypen zu klassifizieren. Ein erster Ansatz zur Bildung konkreter Problemtypen führte zu folgender Unterscheidung:

- Interpretation:
  Ableitung von Situationsbeschreibungen aus Sensordaten.
- Diagnostik:
  Ableitung von Systemfehlern aus Beobachtungen.
- Überwachung:
  Vergleich von Beobachtungen mit Sollwerten.
- Design:
  Konfigurationen von Objekten unter Berücksichtigung besonderer Anforderungen.
- Planung:
  Entwurf einer Folge von Aktionen zum Erreichen eines Zieles.
- Vorhersage:
  Ableitung von möglichen Konsequenzen aus gegebenen Situationen.

Da verschiedene dieser Problemtypen mit denselben Strategien gelöst werden können, ist diese Einteilung für die Zuordnung von Problemlösungsstrategien nicht optimal und man unterscheidet daher die folgenden drei wichtigsten Problemlösungstypen:

(1) Diagnostik:
Die Lösung wird aus einer Menge vorgegebener Alternativen ausgewählt.
(Hierzu gehören: Auswahl, Selektion, Klassifikation).

(2) Konstruktion:
Die Lösung wird aus kleinen Bausteinen zusammengesetzt.
(Hierzu gehören: Konfiguration, Design, Planung).

(3) Simulation:
Aus dem Ausgangszustand werden Folgezustände hergeleitet.
(Dies umfaßt die Vorhersage)

In den folgenden Abschnitten werden diese drei Problemlösungstypen näher untersucht.

5.1 Der Problemlösungstyp Diagnostik

Die Diagnostik bezeichnet den Lösungsprozeß für Probleme mit folgenden Eigenschaften:

- Der Problembereich besteht aus zwei explizit gegebenen, disjunkten Mengen von Problemmerkmalen (Symptomen) und Problemlösungen (Diagnosen) und aus typischerweise unsicherem Wissen über die Beziehung zwischen Symptomen und Diagnosen.
- Ein Problem ist durch eine eventuell unvollständig gegebene Teilmenge der Symptome charakterisiert.
- Die Lösung eines Problems besteht aus einer oder mehreren Diagnosen.
- Wenn die Qualität der Problemlösung durch Erfassung zusätzlicher Symptome verbessert werden kann, so ist es eine Teilaufgabe der Diagnostik, zu bestimmen, ob und welche zusätzlichen Symptome angefordert werden sollen.

Die wichtigsten Formen diagnostisches Wissen darzustellen und auszuwerten sind **statistisches** und **fallvergleichendes Wissen**, das auf der Auswertung von Falldatenbanken beruht, **heuristisches Erfahrungswissen von Experten** und **kausales Wissen** über Struktur und Verhalten eines zu diagnostizierenden Systems.

Typische diagnostische Problembereiche sind:

- medizinische Diagnostik,
- technische Diagnostik
  - Qualitätskontrolle (z. B. auf Prüfständen),
  - Reparaturdiagnostik (z. B. für Autos),
  - Prozeßdiagnostik und Prozeßüberwachung (z. B. in der Fertigung),
- Objekterkennung (z. B. Pilzbestimmung).

Manche Problembereiche lassen sich auch als iterative Diagnostikprobleme charakterisieren, wobei die Diagnosen einer Stufe die Symptome der nächsten Stufe bilden (z. B. Prozeßdiagnostik in der Halbleiterherstellung).

Da technische Systeme zumeist gut modelliert werden können, in ihren Anwendungen meist auch überschaubar sind und, im Vergleich zu medizinischen Systemen, mehr und genauere Meßdaten mit sicheren Regeln existieren, liegen die praktischen Erfolge von Diagnosesystemen fast ausschließlich im technischen Bereich, während die medizinischen Systeme mit ganz wenigen Ausnahmen Demonstrationsprototypen geblieben sind.

Das für die Diagnostik typische Zurückschließen von Beobachtungen auf Systemzustände bzw. Objekte, die die Beobachtungen hervorrufen, ist eine Form der **Abduktion**: Wenn eine Diagnose D das Symptom S verursacht und S wird beobachtet, dann ist D eine mögliche

Erklärung für S. Die Abduktion ist natürlich keine logisch zwingende Schlußweise wie die Deduktion, da eine Beobachtung viele Ursachen haben kann.

Die dadurch bedingte Unsicherheit in der Diagnosebewertung läßt sich durch Auswertung zusätzlicher Daten reduzieren. Weil die Symptomerhebung aufwendig und risikoreich sein kann, müssen Kosten und Nutzen diagnostischer Untersuchungen sorgfältig gegeneinander abgewogen werden.

An ein Diagnosesystem werden folgende Anforderungen gestellt:

- Diagnosebewertung mit unsicherem Wissen,
- Diagnosebewertung mit unvollständigem Wissen,
- Plausibilitätskontrolle der Eingabedaten,
- Erkennen von Mehrfachdiagnosen,
- adäquate Behandlung von Widersprüchen,
- Auswertung von Folgesitzungen.

5.1.1 Übersicht über Diagnosetechniken

Im folgenden werden diagnostische Standardtechniken zur Wissensrepräsentation, zu Kontrollstrategien und zur probabilistischen Diagnosebewertung beschrieben.

5.1.1.1 Wissensrepräsentation

Ein universelles Prinzip der Wissensrepräsentation in Expertensystemen ist die Übernahme der Fachterminologie des Anwendungsbereiches. In der Diagnostik reflektieren die Begriffe die kontinuierliche Verdichtung der Rohdaten über den diagnostischen Mittelbau zu den Enddiagnosen. Der diagnostische Mittelbau läßt sich dabei in zwei Phasen unterteilen: **Datenvorverarbeitung** und **diagnostische Auswertung.**

Bei der Datenvorverarbeitung, die meist durch einfache Regeln realisiert werden kann, werden Rohdaten durch einfache Symptomintertpretationen wie z. B. arithmetische Berechnungen, Abstraktion von quantitativen zu qualitativen Werten etc. aufbereitet.

Die eigentliche, auf unsicherem Wissen basierende diagnostische Auswertung wird oft durch ein reichhaltiges Geflecht von Zwischendiagnosen unterstützt, die eine schrittweise Verfeinerung allgemeiner Diagnoseklassen ermöglichen.

Solche verschiedenen Typen von Diagnosehierarchien sind:

- Strenge Hierarchie, d. h. ein Nachfolger hat höchstens einen Vorgänger;
- multiple Hierarchie, d. h. ein Nachfolger kann mehrere Vorgänger haben, aber es existieren keine Schleifen;
- Netzwerk, d. h. eine multiple Hierarchie mit Schleifen.

In diesen verschiedenen Typen von "Geflechten" gibt es entweder einen, mehrere oder zirkuläre Pfade zu einer Diagnose.

Mit zirkulären Pfaden (Schleifen) kann die Beschränkung vermieden werden, daß eine Diagnose von einer anderen etabliert sein muß; stattdessen kann die Diagnose, die zuerst bestätigt wurde, den Verdacht für andere verstärken. Eine Selbstbestätigung einer Diagnose über Schleifen muß dabei natürlich ausgeschlossen werden.

5.1.1.2 Kontrollstrategien

Die Kontrollstrategie hat neben der effektiven Symptomauswertung auch die Aufgabe, gegebenenfalls zusätzliche Symptome zur Überprüfung von Verdachtsdiagnosen anzufordern.

Die wichtigsten Strategien sind:

- Vorwärtsverkettung: Ausgehend von den eingehenden Symptomen werden alle anwendungsbereiten Regeln ausgewertet und somit alle möglichen Schlußfolgerungen gezogen. Die Symptomerfassung muß durch zusätzliches Wissen oder durch den Benutzer gesteuert werden.
- Rückwärtsverkettung: Ausgehend von einem Ziel (z. B. die Ursache einer Infektion zu finden) werden alle Regeln ausgewertet, die zum Erreichen des Zieles beitragen können. Falls Rohdaten zur Auswertung der Regeln unbekannt sind, so werden sie vom Benutzer erfragt.
- Establish-Refine: In einer strengen Diagnosehierarchie wird eine Diagnoseklasse zunächst durch Rückwärtsverkettung bestätigt und dann verfeinert, indem versucht wird, einen Nachfolger zu bestätigen, der wiederum verfeinert wird, etc.
- Hypothesize-and-Test: Aus den eingegebenen Symptomen werden durch Vorwärtsverkettung Verdachtshypothesen generiert, die anschließend durch Rückwärtsverkettung gezielt überprüft werden.

Im folgenden werden Vor- und Nachteile sowie Anwendungsgebiete dieser vier Strategien kurz erläutert.

Vorwärts- und Rückwärtsverkettung sind Basisstrategien zur Abarbeitung von Regeln, die für die Diagnostik nicht optimal sind, da sie eine vollständige Breiten- bzw. Tiefensuche bedeuten (vgl. Abschnitt 4.4.1).

Establish-Refine ist ideal für strenge Hierarchien, da nur eine kleine Teilmenge der Diagnosen untersucht werden muß. Allerdings kann es auch häufig nicht angewendet werden, da sich viele Gebiete nur schlecht streng hierarchisch strukturieren lassen. Die Kombination mehrerer strenger Hierarchien führt zu Heterarchien oder Netzwerken, die sich zwar schlecht mit Establish-Refine, dafür aber gut mit Hypothesize-and-Test auswerten lassen. Diese Strategie erfordert zusätzliches Wissen zur Verdachtsgenerierung, dessen Qualität die Effizienz und Korrektheit bestimmt. So ließe sich die schrittweise Verfeinerung von Establish-Refine durch Regeln zur Verdachtsgenerierung simulieren, die nach Etablierung einer Diagnoseklasse die Nachfolger verdächtigen.

Heterarchische Abhängigkeiten kann man durch zusätzliche Verdachtsregeln darstellen. Auch reines Vorwärts- und Rückwärtsverketten kann man als Spezialfall von Hypothesize-and-Test auffassen, bei dem die Verdachtsüberprüfung bzw. eine Verdachtsgenerierung fehlt.

Während Rückwärtsverkettung, Establish-Refine und Hypothesize-and-Test eine einfache Strategie zur Anforderung zusätzlicher Symptome explizit enthalten, kann diese häufig durch explizites Wissen verbessert werden.

Diagnostik-Expertensysteme mit einer vorwärts- und rückwärtsverketteten Kontrollstrategie werden auch als **regelbasierte Systeme** (vgl. Abschnitt 2.4) und solche mit Establish-Refine als **framebasierte Systeme** (vgl. Abschnitt 2.5.2) bezeichnet, da bei letzteren in der Wissensrepräsentation und dem Kontrollfluß die Bedeutung der Diagnose-Frames stärker betont wird.

5.1.1.3 Probabilistische Diagnosebewertung

Während es für die genaue Form des probabilistischen Schließens viele konkurierende Ansätze gibt (vgl. Abschnitt 2.8.1), deren objektive Einschätzung kaum möglich ist, lassen sich die unterschiedlichen Bestandteile der probabilistischen Diagnosebewertung gut charakterisieren (vgl Abb. 5).

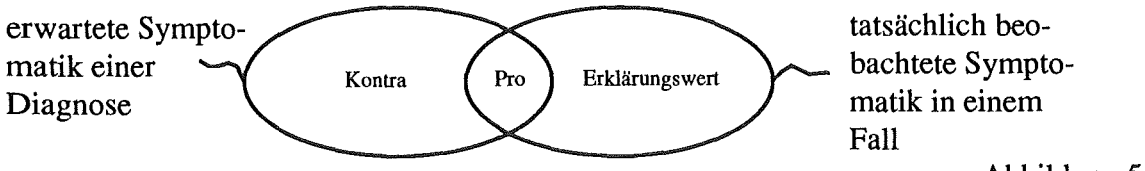

Abbildung 5

Da nur in idealen Fällen die tatsächlich beobachtete und die von der Diagnose erwartete Symptomatik vollständig übereinstimmen, muß der Grad der Übereinstimmung abgeschätzt werden. Während die Schnittmenge der erwarteten und beobachteten Symptome (Pro) für die Diagnose sprechen, schwächen erwartete, aber nicht beobachtete Symptome (Kontra) den Verdacht ab. Nicht erwartete, aber beobachtete Symptome sprechen ebenfalls gegen die Diagnose, da ihr Erklärungswert zu gering ist.

Der Erklärungswert kann aber durch Etablierung zusätzlicher Diagnosen, die bisher noch nicht erklärte Symptome abdecken, verbessert werden. Im allgemeinen wird man jedoch eine einzelne Diagnose, d. h. eine Fehlerursache, für wahrscheinlicher halten, als die Kombination mehrerer, voneinander unabhängiger Ursachen zur Erklärung der Symptome.

Während assoziative Diagnostiksysteme die Aspekte Pro und Kontra betonen, steht bei modellbasierten Systemen der Erklärungswert im Vordergrund (siehe Abschnitt 5.1.2).

Ebenfalls wichtig ist die generelle Häufigkeit (**Prädisposition**) der Diagnosen, die häufig in Abhängigkeit von Grunddaten angegeben wird. Eine hohe Prädisposition kann nur den Glauben in eine bereits verdächtige Diagnose verstärken; sie kann aber niemals aus sich heraus eine Diagnose bestätigen.

Eine Entscheidung über die Etablierung von Diagnosen und daraus resultierender Folgeaktionen, wie Reparaturen in der Gerätediagnostik oder Korrekturen in der Prozeßdiagnostik, geschieht meist durch sorgfältigen Vergleich der wahrscheinlichsten Diagnosen und Auswahl der besten Alternative (**Differentialdiagnostik**).

5.1.2 Übersicht über Diagnostiksysteme

Wie bereits oben angedeutet lassen sich die bisherigen Programme zur Lösung diagnostischer Probleme nach der Art ihres verwendeten Wissens in die folgenden drei Klassen einteilen:

5.1.2.1 Statistische und fallvergleichende Diagnostiksysteme

Die statistischen Ansätze (z. B. Bayes-Theorem oder Dempster-Shafter Theorie, vgl. Abschnitt 2.8) behandeln primär nur das Problem der Diagnosebewertung mit unsicheren Daten. Sie zeichnen sich durch eine hohe Objektivierbarkeit aus, wenn die vorhandenen Daten die statistischen Voraussetzungen erfüllen. Da der Inferenzmechanismus komplexe arithmetische Operationen erfordert, lassen sich die Ergebnisse kaum verständlich erklären. Ein einfacher, aber schlechter objektivierbarer und bisher wenig untersuchter Ansatz zur Auswertung von Falldatenbanken ist das Heraussuchen vergleichbarer Fälle aus der Datenbank zu einem neuen Fall.

5.1.2.2 Heuristische (assoziative) Diagnostiksysteme

Sie basieren auf (assoziativem) Erfahrungswissen von Experten, d. h. von Experten geschätzte Diagnosebewertungen anstelle statistisch ausgewerteter Daten. Daher ist ihre Validierbarkeit wesentlich geringer als bei statistischen Ansätzen. Ihre wesentlichen Vorteile sind:

- Sie können alle Aspekte des diagnostischen Problemlösens behandeln.
- Sie sind breit einsetzbar, da weit schwächere Voraussetzungen zu ihrer Anwendbarkeit erfüllt sein müssen, als bei statistischen Ansätzen.
- Sie zeichnen sich meist durch eine flexible Kontrollstrategie, durch die Darstellung des diagnostischen Mittelbaus und durch ein, im Vergleich zu statistischen Systemen gröberes, aber mehr Bewertungskriterien berücksichtigendes probabilistisches Verrechnungsschema aus (vgl. Abschnitt 2.8.1)

Nachteilig wirkt sich aus, daß der Wissenserwerb in solchen Systemen extrem aufwendig ist.

Beispiele "klassischer" heuristischer Diagnostiksysteme im medizinischen Bereich sind MYCIN, EXPERT, PIP und INTERNIST.

Für eine ausführlichere Darstellung des Diagnostiksystems MED2 sei auf Puppe /25/, S. 83-85 verwiesen.

5.1.2.3 Modellbasierte Diagnostiksysteme

Sie enthalten hauptsächlich kausale Ursache-Wirkung-Beziehungen der Art: Eine Störung führt zu bestimmten Symptomen.

Während in heuristischen Systemen die Diagnosen nach ihrer akkumulativen Evidenz bewertet werden, ist bei der kausalen Diagnostik die Qualität der Erklärung maßgebend, d. h. eine Diagnose ist "konsistent", wenn alle beobachteten Symptome aus ihr abgeleitet werden können.

Die einfachste Wissensrepräsentation dafür ist die direkte Zuordnung von Ursachen (Diagnosen) zu ihren Wirkungen (Symptomen), z. B. D1 $\rightarrow$ {S1, S2, S3}. Die kausale Diagnostik ist dann eine Anwendung von Algorithmen zum Finden einer minimalen Mengenüberdeckung. Bei der Verwendung komplizierter Modelle unterscheidet man zwei Typen:

- "Funktionale Modelle":
  Dies sind Modelle, die vom normalen Funktionieren des zu diagnostizierenden Systems ausgehen und Diagnosen als Veränderung des Modells auffassen, die zu den beobachteten Symptomen führen.

  Funktionale Modelle sind typisch für den technischen Bereich und weniger gut für den medizinischen Bereich geeignet, da das nötige kausale Wissen häufig nicht vorhanden ist und da die Modelle, wegen des hohen Vernetzungsgrades und den außerordentlich vielen Rückkoppelungsschleifen schon für winzige Anwendungsgebiete sehr komplex werden.

- "Pathophysiologische Modelle":
  (Pathophysiologie = Lehre von Fehlverhalten):
  Dies sind Modelle, die das Fehlverhalten eines Systems darstellen, Symptome den Diagnosen explizit zuordnen und die Zusammenhänge, z. B. indem sie Zwischenschritte und Aussagen über die Stärke von Beziehungen verwenden, modellieren.

  Diese Modelle werden häufig im medizinischen Bereich eingesetzt.

  Für weitere Ausführungen bzgl. dieser beiden hier vorgestellten Modelle sei auf /24/, S. 86-89 verwiesen.

5.1.3 Integrationsmöglichkeit verschiedener Diagnostiksysteme

Die bisherigen Diagnostikprogramme konzentrieren sich jeweils nur auf eine der vier Bewertungsarten (statistisch, fallvergleich, assoziativ oder kausal). Da sich ihre Vorzüge ergänzen, liegt es nahe, die Möglichkeiten des Zusammenspiels nach den verschiedenen Arten des verwendeten Wissens zu untersuchen.

Ein Hauptproblem dabei ist die Frage der Konsistenz zwischen den verschiedenen Wissensarten, deren Wissen sich zwar überlappt, aber nicht ineinander überführbar ist.

- Statistisches und assoziatives Wissen ähneln sich in der Darstellung probabilistischer Bewertung. Sie unterscheiden sich jedoch einerseits in der Vielfalt der angebotenen Wissensrepräsentationen, da assoziative Diagnostiksysteme Mechanismen, wie z. B. Regeln mit Symptomkombinationen und Ausnahmen bieten, die kein Äquivalent in statistischen Wissensrepräsentationen haben und andererseits in der Exaktheit der probabilistischen Bewertungen, die aus Falldatenbanken errechnet bzw. von Experten geschätzt werden.
  Die Bedeutung des Unterschiedes zwischen errechneten und geschätzten Symptom/Diagnose-Wahrscheinlichkeiten wurde, indem man zeigte, daß sich die Erfolgsquote eines auf dem Theorem von Bayes beruhenden Programmes drastisch verschlechtert, nachdem statistische Wahrscheinlichkeiten durch Schätzwerte von Experten ersetzt wurden, experimentell nachgewiesen.

- Assoziatives und kausales Wissen kann je nach Detaillierungsgrad denselben diagnostischen Mittelbau repräsentieren, oder er kann bei kausalem Wissen wesentlich verfeinert sein (gegebenenfalls auf verschiedenen Abstraktionsebenen).

  Der grundsätzliche Unterschied besteht darin, daß auch bei gut strukturierten heuristischen Systemen die probabilistischen Bewertungen meist eine große Rolle spielen, während typische kausale Systeme nur überprüfen, ob Diagnosen mit den beobachtbaren Symptomen konsistent sind, ohne dabei vergleichbares probabilistisches Wissen zu berücksichtigen.

- Die fallvergleichende Diagnostik unterscheidet sich prinzipiell von allen anderen Wissensarten, da ihr Wissen weniger auf einer Abstraktion von Einzelfällen beruht, sondern ein neuer Fall direkt mit abgespeicherten Fällen einer Datenbank verglichen wird.

Ein möglicher Ansatz zur Integration besteht darin, die Wissensarten so zu erweitern oder einzuschränken, daß keine Konsistenzprobleme entstehen. Beispiele solcher Vorgehensweisen sind:

- Die Erweiterung kausaler Modelle durch probabilistische Bewertungen. Das Hauptproblem besteht in dem Wissenserwerb, der durch die Unsicherheitsangaben für die zusätzlichen Datails kausaler Modelle noch aufwendiger als bei heuristischen Systemen wird.

- Die Kompilierung von kausalem Wissen in heuristische Regeln zur Effizienzsteigerung. Der gravierende Nachteil besteht darin, daß die resultierenden Regeln keine probabilistische Bewertung enthalten, was ihren Nutzen erheblich beschränkt.

Ein weiterer möglicher Ansatz zur Integration besteht darin, den Konsistenzanspruch zumindest teilweise aufzugeben und dafür Kriterien zu definieren, wann welche Wissensart zum Einsatz kommt. Von diesem Ansatz erwartet man eine deutliche Leistungssteigerung im Vergleich zu heutigen Systemen. Weitere Erläuterungen zu diesem Ansatz findet man bei /25/, S. 90f.

5.2 Der Problemlösungstyp Konstruktion

Der Problemlösungstyp Konstruktion unterscheidet sich von der Diagnostik dadurch, daß die Lösung nicht ausgewählt werden kann, sondern aus kleinen Bausteinen zusammengesetzt werden muß.

Zur Konstruktion gehören **Planung**, bei der eine Folge von Handlungen zum Erreichen eines Zielzustandes gesucht wird und **Design** (Konfigurierung), zum Entwurf eines Objektes, das bestimmten Anforderungen genügen muß. Die Begriffe Planung und Design werden hier, wie auch in der Umgangssprache, synonym zum Begriff Konstruktion verwendet.

Konstruktionsprobleme sind wesentlich heterogener als beispielsweise Diagnostikprobleme. Wichtig für die Auswahl von Problemlösungsstrategien ist die Unterscheidung zwischen **Zuordnungs-** und **Transformationsproblemen:**

- Bei Zuordnungsproblemen stehen die Basiselemente fest und müssen nur richtig verknüpft werden. Beispiele sind die Stundenplanerstellung und die Konfigurierung von Computern.

- Bei Transformationsproblemen wird ein Objekt von einem Anfangszustand in einen Zielzustand transformiert und die dazu notwendigen Operatoren werden in der richtigen Reihenfolge angeordnet. Beispiele sind das Planen von Experimenten und das Verschieben von Klötzchen.

Wenn es zur Lösung von Konstruktionsproblemen nur wenige Operatoren gibt, so besteht das Hauptproblem im Herausfinden der richtigen Reihenfolge der Operatoren. Wenn es viele Operatoren gibt, muß man zunächst die geeigneten Operatoren zur Erreichung des Zielzustandes bestimmen. Wenn es, wie beim Planen von bestimmten Experimenten auch unklar ist, welche Substanzen man benötigt, müssen außer den Operatoren auch die Objekte ausgewählt werden. Um den Auswahlvorgang von Operatoren und Objekten zu vereinfachen, benutzt man Begriffshierarchien. Die Endknoten dieser Hierarchien sind konkrete Objekte und primitive Operatoren. Diese Hierarchien werden in

- taxonomischen Hierarchien (A ist Untergruppe von B) oder in
- kompositionelle Hierarchien (A ist Teil von B) unterteilt,

wobei man unter einer Taxonomie die Klassifikation von Einheiten, die den Aufbau des Systems beschreiben, versteht.

Viele Konstruktionsprobleme lassen sich in ein Transformationsproblem und ein Zuordnungsproblem aufteilen, die relativ unabhängig voneinander lösbar sind.

Die primitivste Technik des Planens besteht darin, daß man mit dem Ausgangszustand beginnt und systematisch alle Sequenzen von Operatoren durchprobiert, bis man den gewünschten Zielzustand erreicht hat. Diese Vorgehen wird **einstufiges** oder **nicht-hierarchisches Planen** genannt und führt bereits bei kleinen Problemen zu einer kombinatorischen Explosion. Dieser kann man am besten durch Abstraktion begegnen, d. h. man plant zunächst eine Sequenz von abstrakten Operatoren zur Erreichung des Zielzustandes (Grobplan), die dann schrittweise verfeinert wird. Diese Methode heißt **mehrstufiges** oder **hierarchisches Planen** und entspricht dem top-down-Ansatz beim Programmieren.

Während sich Zuordnungsprobleme meistens mit Constraint-Propagierungsmethoden lösen lassen, wird, wie oben bereits geschildert, bei Transformationsproblemen oft ein Grobplan generiert und dieser schrittweise verfeinert. Eine Schwierigkeit bei Transformationsproblemen ist die **Behandlung von Interaktion zwischen den Teilproblemen**.

Während **lineare Planer** versuchen, in einem fertigen Plan auftretende Interaktionen durch Umstellung oder Korrektur der Planungsschritte zu lösen, vermeiden **nicht-lineare Planer** Entscheidungen bei unvollständigem Wissen soweit wie möglich (**Least-Commitment-Strategie**). Ein anderes, bisher noch kaum behandeltes Problem ist die Berücksichtigung von Zeitbeschränkungen im Plan.

Ein ziemlich gut verstandenes Zuordnungsproblem ist die Konfigurierung von Objekten nach einem festen Grobplan ohne Interaktionen, z. B. im Konstruktionssystem R1 zur Konfiguration von Computersystemen bei der Firma DEC.

Anspruchsvolle Planungssysteme sind z. B. die beiden MOLGENs zur Planung molekularer Experimente. Bei ihnen wurden viele Planungstechniken, wie:

- Skelett-Pläne,
  (Dies ist eine wissensbasierte Planungsmethode, bei der der allgemeine Plan aus einer, von einem Experten vorgegebenen Bibliothek von Grobplänen, ausgewählt wird.)

- Differenzenmethode,
  (Durch Feststellung der Differenz von Ausgangs- und Zielzustand wird der Grobplan direkt aus der Aufgabenstellung generiert und dann werden Operatoren zur Differenzverringerung gesucht.)

- Metaplanen,
  (Wird bereits die Planerstellung selbst kompliziert, so kann die Anwendung von Planungstechniken zur Planerstellung sinnvoll sein.)

- die Least-Commitment-Strategie und

- eine hierarchisch strukturierte Wissensbasis

eingesetzt.

5.3 Der Problemlösungstyp Simulation

Die Simulation unterscheidet sich von der Diagnostik und der Konstruktion dadurch, daß kein vorgegebenes Ziel erreicht werden soll, sondern nur die Auswirkungen von Handlungen oder Ereignissen simuliert werden.

Mit der Simulation wird das Verhalten eines Systems vorhergesagt. Sie dient oft zur Plausibilitätskontrolle von Problemlösungen, wie z. B. die Überprüfung einer Verdachtsdiagnose in einem kausalen Modell.

Die Simulation kann man als Komplement zum Planen auffassen: Während beim Planen Anfangs- und Zielzustand gegeben sind und Aktionen zum Erreichen des Zieles gesucht werden, sind bei der Simulation der Anfangszustand und die Aktionen (Prozesse) bekannt und Folgezustände werden gesucht.

Ein Zustand wird durch eine Menge von Objekten und Beziehungen zwischen den Objekten beschrieben. Eine Folge von Zuständen beschreibt das Verhalten des Systems.

Zur Berechnung des Systemverhaltens (der Verhaltensbeschreibung) gibt es die folgenden Simulationstypen:

(1) Numerische Simulation:
Das Systemverhalten wird durch eine Folge von Werten der Systemparameter zu verschiedenen Zeitpunkten angegeben. Dieser Simulationstyp eignet sich gut für Scheduling-Probleme.

Vorteile:

- Exakte Verhaltensbeschreibung,
- einfache Berechnung.

Nachteile:

- Fehlender Abstraktionsgrad, da partielles Wissen nicht dargestellt werden kann, d. h. es muß ein Zahlenwert für jeden Parameter und jede Konstante festgelegt werden.

(2) Analytische Simulation:
Das Systemverhalten wird durch algebraische Umformungen der strukturellen Beschreibung und Auflösung nach den interessanten Variablen beschrieben.

Dies ist die optimale Vorgehensweise, falls die durch die mächtigen Berechnungstechniken ausgelösten algebraischen Operationen handhabbar sind.

Vorteile:

- Exakte Verhaltensbeschreibung,
- die Darstellung von partiellem Wissen ist möglich, da z. B. für Konstante keine Zahlenwerte eingesetzt werden müssen.

Nachteile:

- Es sind mächtige Berechnungstechniken, z. B. Integration, erforderlich, deren Grenzen bei komplexen Systemen schnell erreicht werden.

(3) Qualitative Simulation:
Dieser Simulationstyp dient zur Vorhersage in Bereichen, in denen die konventionellen Simulationstypen (dies sind die Erstgenannten, die man unter dem Oberbegriff "quantitative Simulation" zusammenfassen könnte) nicht anwendbar sind.

Der Wertebereich der Parameter wird auf qualitativ interessante Werte beschränkt, z. B. kann der Wertebereich eines Parameters dadurch gegeben sein, daß er größer als ein anderer Parameter ist. Die Beschreibung der Struktur wird vereinfacht, indem die Beziehungen zwischen den Parametern nur qualitativ (z. B. durch Proportionalitäten) beschrieben werden. Dadurch können einfache algebraische Techniken verwendet werden.

Zur Beschreibung der Parameter dienen Relationen und Referenzwerte. Die Referenzwerte orientieren sich meistens an qualitativen "Landmarkewerten", die für die Simulation große Bedeutung haben.

Da die Landmarkenwerte im wesentlichen von Experten vorgegeben werden müssen, eignet sich die qualitative Simulation nur für im Prinzip gut verstandene Systeme. Beispiele für qualitative Simulationssysteme sind QSIM (Qualitive Simulation) und LONGS System (siehe /25/).

Vorteile:

- Einfach Berechnung,
- Darstellung von partiellem Wissen.

Nachteile:

- Ungenaue (qualitative) Verhaltensbeschreibung:
  Die aus der Vereinfachung der Struktur- und Verhaltensbeschreibung des zu simulierenden Systems auf qualitative Zusammenhänge resultierenden Unsicherheiten, z. B. wenn mehrere Einflüsse für einen Parameter mit unterschiedlichen Vorzeichen zusammentreffen, werden meist durch eine Fallunterscheidung und paralleles Weiterverfolgen der möglichen aktiven Prozesse behandelt. Allerdings führt die Vernachlässigung der Größenordnungen der relevanten Parameter leicht zu einer kombinatorischen Explosion.

Es gibt jedoch auch Systeme, bei denen nicht die Komponenten und ihre Parameter, sondern Prozesse die Objekte in der Strukturbeschreibung sind.

Die derzeitigen Systeme zur qualitativen Simulation behandeln zumeist sehr einfache Fragestellungen und sind kaum praxisreif.

6 Wissensakquisition

6.1 Symbolische Wissensakquisition

Die Wissensakquisition ist die Erhebung von Wissen aus verschiedenen Wissensquellen, wie Experten, Büchern, Handbüchern, Falldaten oder Lexika, sowie die anschließende Umsetzung dieses Wissens in eine operationale Wissensbasis (/11/).

Die Wissensakquisition hat bei der Entwicklung wissensbasierter Systeme nach Marchand /21/ zwei unterschiedliche Bedeutungen; Wissensakquisition beinhaltet zum einen

- alle Aktivitäten, die in der Entwurfsphase mit der Festlegung der Wissensdarstellung, der Verarbeitung und der Mensch-Maschine Kommunikation zu tun haben und zum anderen
- alle Aktivitäten, die den Aufbau und die Pflege der Wissensbasis betreffen.

Diese beiden Bestandteile der Wissensakquisition können nicht voneinander getrennt werden. Die spätere Wartung muß bei dem Entwurf berücksichtigt werden (vgl. /11/).

6.1.1 Arten des Wissenserwerbs

Bei der Entwicklung von Expertensystemen stellt die Wissensakquisition das Kernproblem dar; sie wird daher auch häufig als Flaschenhals der Expertensystemtechnologie bezeichnet.

Bei der Beseitigung dieses Flaschenhalses muß man jedoch berücksichtigen, daß z. B. eine automatische Wissensakquisition (siehe z. B. /15/ oder /32/) dieses Problem zunächst nur aufkosten eines Flaschenhalses bei der Validierung des Wissens löst.

Bei Expertensystemen unterscheidet man drei Grundarten des Wissenserwerbs:

(1) Indirekter Wissenserwerb:
Ein Wissensingenieur befragt einen Experten und formalisiert die Ergebnisse für das Expertensystem.

Diese Wissenserwerbsmethode ist aufwendig und fehleranfällig, da die Wissensextraktion von einem Experten ein schwieriges Problem darstellt und wird daher nur dann gewählt, wenn andere Methoden nicht anwendbar sind.

(2) Direkter Wissenserwerb:
Der Experte formalisiert sein Wissen selbst.

Diese Wissenserwerbsmethode erfordert komfortable Wissenserwerbssysteme, die auf einem guten Verständnis der Problemlösungsstrategien im Anwendungsbereich aufbauen müssen.

(3) Automatischer Wissenserwerb:
Das Expertensystem extrahiert sein Wissen selbständig aus Falldaten oder verfügbarer Literatur.

Während hinreichend gute textverstehende Systeme noch nicht existieren, ist das Lernen aus Falldatenbanken heute schon weiter fortgeschritten. Allerdings sind die derzeitigen Systeme noch nicht praxisreif. Diese Methode des Wissenserwerbs wird als Maschinelles Lernen bezeichnet (vgl. /23/ oder /32/).

6.1.2 Phasenmodell des Wissenserwerbs

Ein konkretes Phasenmodell des (indirekten) Wissenserwerbs bei einem Expertensystem läßt sich wie folgt darstellen (Abb. 6):

(1) Problemcharakterisierung:
Hierbei erfolgt die Identifikation der Problemlösungsstrategie und der Wissensrepräsentation. Diese Phase ist nur sehr wenig strukturierbar und sollte deshalb in intensiver Zusammenarbeit zwischen Wissensingenieur und Experte durchgeführt werden.

(2) Shell-Entwicklung:
Unter einer Expertensystem-Shell versteht man ein zur Entwicklung von Expertensystemen hilfreiches Werkzeug. In eine solche Shell muß quasi nur noch das fachspezifische Wissen eingebracht werden, um ein Expertensystem zu erhalten.

Die Shell-Entwicklung kann nur vom Wissensingenieur durchgeführt werden.

Die Shell soll den Experten in die Lage versetzen, nach kurzer Einarbeitungszeit seine Wissensbasis weitgehend selbständig aufzubauen und zu ändern. Dazu muß die Shell Problemlösungsatrategien und Wissensrepräsentationen auf einem möglichst hohen Abstraktionsniveau bereitstellen. Um dem Experten den Umgang mit einer ihm vertrauten Begriffswelt zu ermöglichen, muß die Shell über eine komfortable Wissenserwerbskomponente (einschließlich einer Erklärungskomponente zur Fehlerlokalisierung) verfügen. Während in günstigen Fällen eine passende Shell mit Wissenserwerbskomponente verfügbar ist, muß sie für andere Projekte, z. B. mit Hilfe allgemeiner Expertensystemwerkzeuge, neu entwickelt werden.

Die Bereitstellung einer geeigneten Shell für den Experten eliminiert die Fehlerquelle, die durch ein Interpretationsmodell des Wissensingenieurs bei Interviewtechniken auftritt.

(3) Aufbau der Wissenbasis:
In dieser Phase wird das Expertenwissen formalisiert; sie bildet zusammen mit

der Shell-Entwicklung, die komplexere Phase der Implementierung. Diese Trennung ist ein wesentlicher Teil der Expertensystem Methodologie, deren Hauptvorteil sich bei der Test- oder Wartungsphase, die bei in der Praxis eingesetzten Expertensystemen nie endet, zeigt.

Die Benutzung einer adäquaten Shell ermöglicht es dem Experten, sein Wissen selbst zu formalisieren und auf die vorgegebenen Wissensrepräsentationen abzubilden. Für die Einarbeitung in die Shell sollte der Wissensingenieur dem Experten mit Rat und Tat zur Seite stehen.

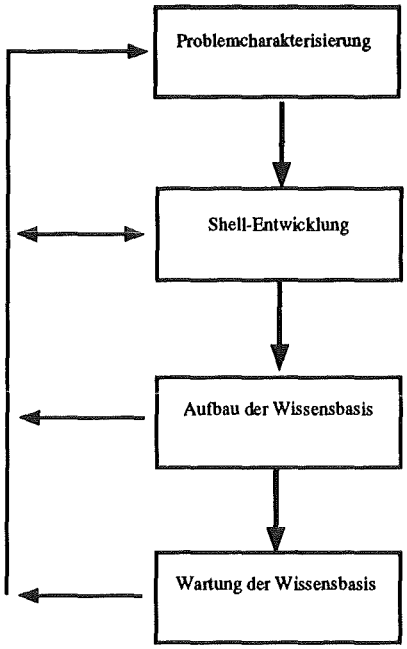

Abbildung 6

(4) Wartung der Wissensbasis:
Die Wartung eines Systems ist umso effizienter, je kürzer die Rückkopplungsschleifen zwischen der Entdeckung von Mängeln und deren Verbesserung durch den Experten mit anschließenden Tests sind. In dieser Phase sollte der Experte selbständig mit der Shell umgehen können. Der Experte kann hierbei durch automatische Analysetechniken z. B. bei der Fehlersuche unterstützt werden.

Der zu bewältigende Aufwand nimmt von Phase zu Phase zu. Diesen vier Phasen muß jedoch noch eine Phase vorgeschaltet werden, in der die Leistungsanforderungen an das zukünftige Expertensystem und die verfügbaren Ressourcen wie Hardware, Zeit, finanzielle Mittel und Mitarbeit von Experten beschrieben werden.

Im Wissenserwerb besteht das Hauptproblem bei der Entwicklung von Expertensystemen.

Das Hauptproblem beim automatischen Wissenerwerb sind die inhärente Beschränkung eines Expertensystems auf ein vorgegebenes Vokabular an Objekten des Anwendungsbereiches und die Tatsache, daß ohne umfangreiches Vorwissen kein Lernen möglich ist.

6.2 Wissensakquisition durch neuronale Netze

Neuronale Netze leisten ebenfalls eine Wissensakquisition, allerdings ist das akquirierte Wissen nicht in expliziter Form verfügbar. Eine Erklärung bzgl. den abgeleiteten Ergebnissen ist nicht möglich. Bei der Zuordnung von Handlungssequenzen (Ausgangsmuster) zu bestimmten Situationen (Eingangsmuster) sind sie sehr leistungsfähig und wenig fehleranfällig. Im folgenden werden die Konstruktions- und Verhaltensmerkmale von neuronalen Netzen kurz skizziert. Der gesamte Abschnitt basiert im wesentlichen auf McCord Nelson, Illingworth /22/.

6.2.1 Die Grundelemente neuronaler Netze

6.2.1.1 Das Prozeßelement

Das **Prozeßelement** oder der **Knoten**, die **Unit** (vgl. Abb. 9), eine Art künstliches Neuron, bildet die Grundeinheit eines neuronalen Netzes. Es erfüllt die folgenden Basisfunktionen:

- Auswertung der gleichzeitig eintreffenden Signale, indem die Stärke jedes einzelnen bestimmt wird.
- Berechnung des gesamten Input-Signals als Kombination der einzelnen Werte.
- Vergleich des gesamten Input-Signals mit einem Schwellenwert.
- Bestimmung der Ausgabe, wobei höchstens ein Ausgabe-Signal erfolgen kann; d. h. es gibt zwar mehrere Eingaben (Inputs), aber nur genau eine Ausgabe (Output).

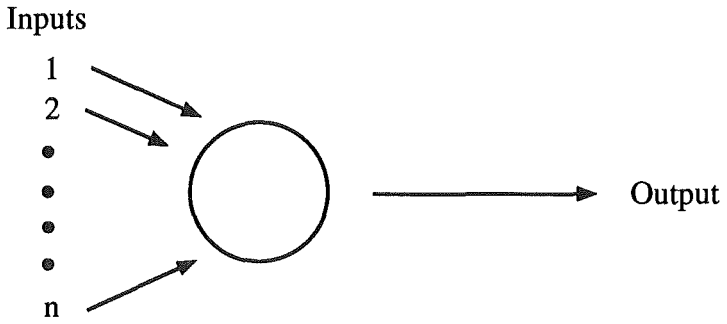

Abbildung 9

6.2.1.2 Die Funktionen (Prozesse) des Prozeßelements

Um die Relevanz eines jeden Input-Signals im Bezug auf das gesamte Input-Signal beeinflussen zu können, kann jedes Input-Signal mit einem besonderen Gewicht versehen werden (Abb. 10).

Die im folgenden beschriebenen Prozesse der Summationsfunktion, der Aktivierungsfunktion und der Übertragungsfunktion sind es in erster Linie, die die Dynamik eines neuronalen Netzes bestimmen.

Mathematisch kann man die Input-Signale und die Gewichte als Vektoren (i_1, i_2, ..., i_n) und (w_1, w_2, ..., w_n) auffassen. Die Berechnung des gesamten Input-Signals erfolgt mit Hilfe einer **Summationsfunktion**, z. B. die gewichtete Summe, d. h. dem euklidischen Skalarprodukt der beiden Vektoren, was geometrisch als ein Maß für die Ähnlichkeit dieser beiden Vektoren aufgefaßt werden kann.

Wenn die gewichtete Summe der Input-Signale größer als der Schwellenwert ist, dann erzeugt das Prozeßelement ein Signal. Falls das Ergebnis der Summationsfunktion kleiner als der Schwellenwert ist, dann wird kein Signal (oder ein hemmendes Signal) erzeugt; beide Antworten sind signifikant.

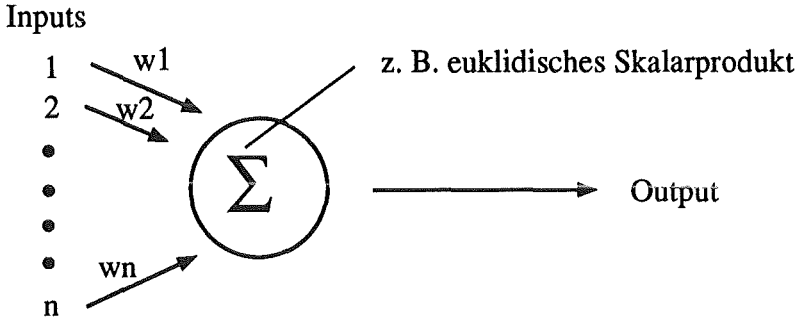

Abbildung 10

Ein weiterer Prozeß im Prozeßelement betrifft dier **Aktivierungsfunktion**. Das Ergebnis der Summationsfunktion kann als Argument für eine Aktivierungsfunktion dienen, bevor es an die Übertragungsfunktion (T in Abb. 11) übergeben wird. Der Sinn der Aktivierungsfunktion besteht darin, daß sie es ermöglicht, daß die Ausgabe in Abhängigkeit von der Zeit variiert. Die Aktivierungen früherer Zeitpunkte geben dem Prozeßelement ein Gedächtnis, mit dem beispielsweise Adaption modelliert werden kann. Allgemein ist die Aktivierungsfunktion von vorhergehenden Aktivierungen und von einem Satz von Parametern abhängig (/20/). In den meisten Fällen wird als Aktivierungsfunktion die Identität verwendet, da die Untersuchungen auf diesem Gebiet noch nicht sehr weit fortgeschritten sind. Den Wert der Aktivierungsfunktion bezeichnet man als **Aktivierungswert**.

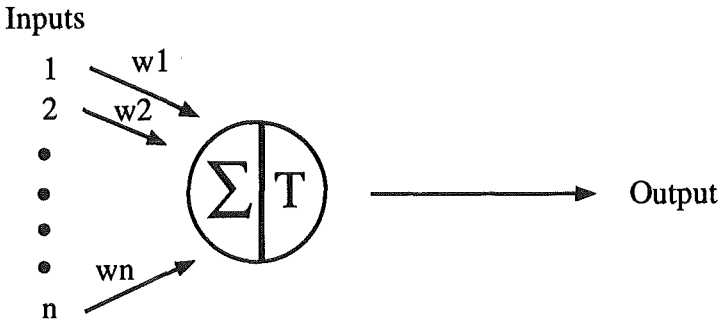

Abbildung 11

Eine wichtige Funktion ist die im allgemeinen nichtlineare **Übertragungsfunktion** T (**Schwellenwert-, Output-Funktion**). Diese Funktion ist im allgemeinen nicht-linear, da lineare Funktionen aufgrund der Tatsache, daß ihr Resultat proportional zur Eingabe ist, nur beschränkte Möglichkeiten bieten und sich daher viele Probleme, wie z. B. das exclusive-or- (XOR-) Problem, mit linearen Funktionen nicht lösen lassen (vgl. z. B. /22/ oder /20/).

Bei den normalerweise üblichen Übertragungsfunktionen (vgl. z. B. /22/ oder /20/) werden in den Anwendungen sigmoide, d. h. S-förmige, Funktionen bevorzugt werden, da sie außer der Nicht-linearität noch über gewisse Differenzierbarkeitseigenschaften verfügen.

Die Anforderungen an eine Übertragungsfunktion sind, daß diese für negative oder zu kleine positive Werte ein definiertes Minimum (entsprechend der Ruhefrequenz beim Neuron) oder 0 liefert und ab einem gewissen Schwellenwert allmählich oder abrupt einen Maximalwert, häufig 1 animmt (/20/).

6.2.1.3 Lernfunktionen

Im Bereich der Lernfunktionen liegt der Vergleich mit dem menschlichen Gehirn nahe. In gleicher Weise wie an den biologischen neuronalen Synapsen können Signale im Prozeßelement hemmend oder stimulierend in bezug auf die Erzeugung einer Ausgabe wirken.

Falls man das Prozeßelement mit einem lokalen Speicher versieht, so kann es Ergebnisse vorangegangener Berechnungen speichern und die im weiteren Verlauf verwendeten Gewichte modifizieren. Diese Fähigkeit, die Gewichte zu verändern, ermöglicht es dem Prozeßelement sein Verhalten in Abhängigkeit von den Eingaben zu verändern, d. h. zu lernen, im Sinne einer Verbesserung der Performanz.

6.2.1.4 Die Kombination von Prozeßelementen zu einem Layer

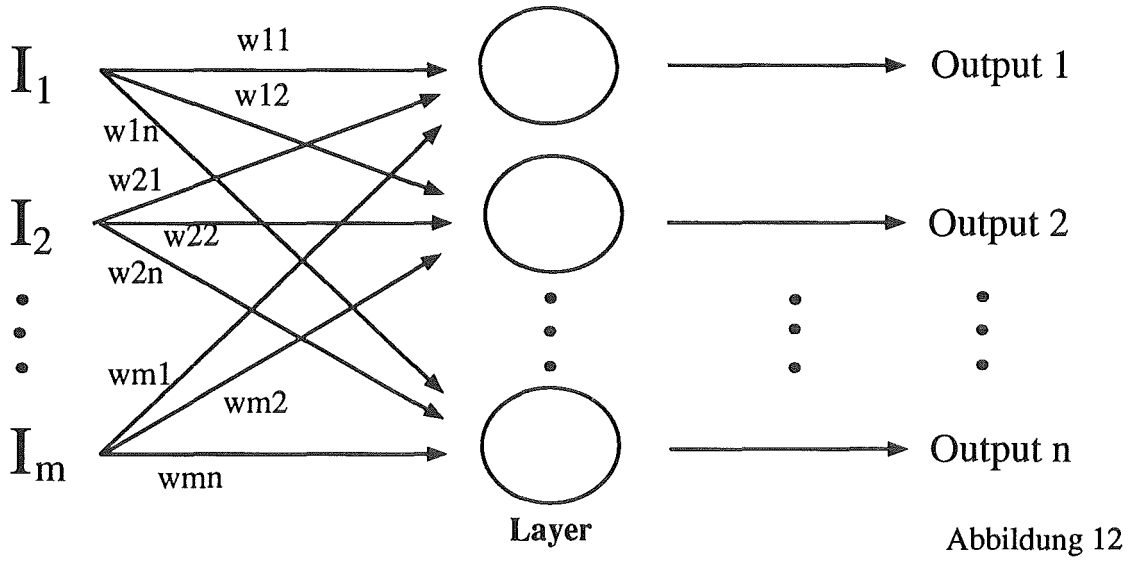

Abbildung 12

Bisher wurden immer nur einzelne Prozeßelemente betrachtet. Der nächste Schritt besteht nun darin, mehrere Prozeßelemente zu einer **Schicht (Layer)** zu kombinieren. Die Input-Signale können zu mehreren Prozeßelementen mit unterschiedlichen Gewichten verbunden sein, was dann zu einer Reihe von Ausgaben führt, je eine pro Prozeßelement (Abb. 12).

6.2.1.5 Die Kombination mehrerer Layer zu einem mehrschichtigen neuronalen Netz

Setzt man den bisher verfolgten Weg zur Konstruktion eines neuronalen Netzes fort, so muß man nun folgerichtig mehrere Schichten zu einem umfangreicheren neuronalen Netz kombinieren.

Derjenige **Layer** (Schicht oder Ebene), der die Inputs empfängt heißt **Input-Layer**. Seine Aufgabe besteht in der Pufferung der Input-Signale. Der Output des Netzwerkes wird von einem **Output-Layer** erzeugt. Sämtliche anderen Layer heißen, da sie keinen direkten Kontakt zur externen Umwelt haben **Hidden-Layer** (vgl. Abb. 13). In einem neuronalen Netz kann es beliebig viele Hidden-Layer geben.

Es liegt in der Natur der Sache, daß es unzählig viele Möglichkeiten gibt, die Knoten der einzelnen Schichten miteinander zu verbinden. Einige der bekanntesten werden im folgenden kurz beschrieben.

- Ein Netzwerk heißt **vollständig verbunden** (fully connected), wenn jeder Output eines Layer mit jedem Knoten des nächsten Layer verbunden ist.

- Das Ausgabesignal eines Knotens kann jedoch auch als Input-Signal darunterliegender Schichten dienen. In einem hierarchisch strukturierten Netzwerk unterscheidet man die folgenden Varianten:

  - **Feed forward-Netze:**
    Die Prozeßelemente eines bestimmten Layers haben keinen Einfluß auf darunterliegende Layer.

  - **Interaktive Modelle:**
    Es können sowohl Verbindungen nach oben, als auch nach unten vorhanden sein.

  Generell gilt jedoch in hierarchischen Netzwerken, daß Verbindungen nur innerhalb oder zu direkt benachbarten Layern (oberhalb oder unterhalb) erlaubt sind.

- In einem nicht-hierarchisch strukturierten Netzwerk gibt es praktisch keine Ordnung. Jedes Prozeßelement kann mit sich und mit jedem anderen Prozeßelement beliebig in Verbindung treten. Solche Netze werden auch **feedback-Netze** genannt. Netzwerke mit geschlossenen Schleifen heißen **rekursive Systeme**.

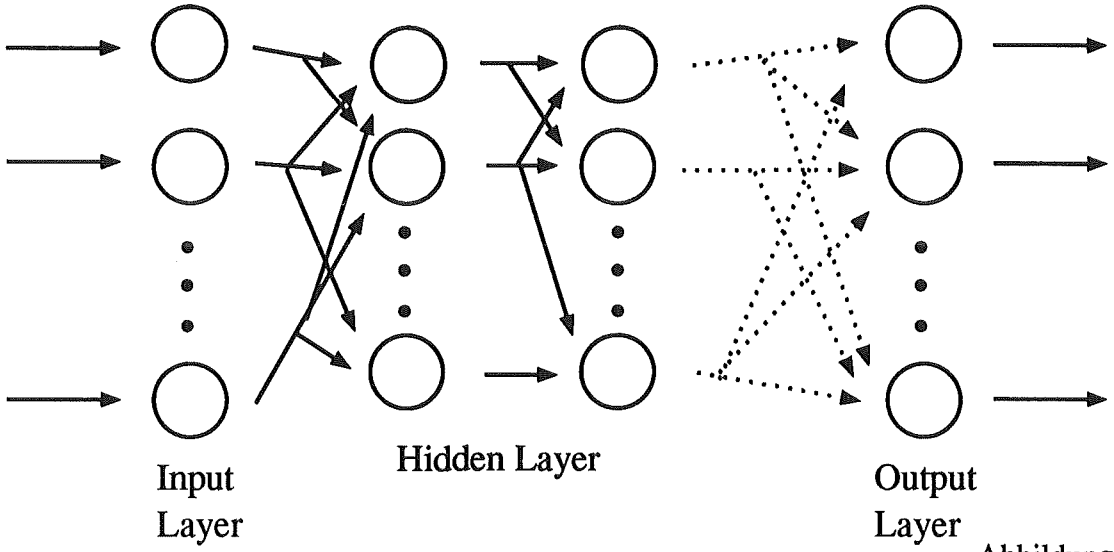

Abbildung 13

Durch das Auftreten von geschlossenen Schleifen (Rekursionen) ergeben sich interessante Möglichkeiten. Man darf jedoch nicht übersehen, daß solche Systeme, sofern sie nicht bestimmten Bedingungen genügen, zum Schwingen neigen. Die Meinungen über die Leistungsfähigkeit hierarchischer gegenüber nicht-hierarchischer Systeme gehen auseinander.

Minsky und Pappert haben gezeigt, daß man mit den bekannten Methoden der Entrekursivierung rekursive Netze, in endlicher Zeit, in entsprechende feed forward-Netze umwandeln kann (/20/).

Allgemein gilt, daß es in einem solchen Netzwerk mehr Verbindungen als Knoten gibt.

6.2.2 Lernen in neuronalen Netzen

Für das Lernen in neuronalen Netzen ist entscheidend, wie das Ausgabe- oder Reaktionsverhalten des Netzes gebildet wird bzw. geändert werden kann, und demnach auch, wie und wo das Netz etwas speichert, das sein Verhalten bestimmt. Das statische (gespeicherte) Wissen des Netzes liegt seinen Verbindungen (deren Gewichten) und in seinem Aufbau, das dynamische (aktuelle) ist in den Aktivierungswerten enthalten, die, wenn es sich um Prozeßelemente zur Ausgabe handelt, die Reaktion des Netzes auf ein präsentiertes Eingabemuster darstellen.

Das zu Erlernende (allgemein ein Muster) wird in neuronalen Netzen nicht explizit gespeichert. Vielmehr sind die Gewichte zwischen den einzelnen Prozeßelementen, die es erlauben, daß ein bestimmtes Muster immer wieder erzeugt werden kann, gespeichert. Lernen in einem neuronalen Netz bedeutet zumeist, die Gewichte richtig zu trainieren. Man geht dabei meist von einer Initialisierung mit Zufallszahlen aus und wendet dann eine Lernstrategie zur Veränderung der Gewichte des Netzes an. Dazu werden dem Netz wiederholt Muster prä-

sentiert. Es werden keine expliziten Regeln gelernt, sondern die Regeln werden anhand der Daten implizit mitgelernt (generalisiert).

Prinzipiell kann man drei Arten der Veränderung von Gewichten unterscheiden:

(1) Entwicklung neuer Verbindungen,

(2) Abbruch vorhandener Verbindungen,

(3) Veränderung der Gewichte bereits existierender Verbindungen.

Die Varianten (1) und (2) können als Spezialfälle von (3) angesehen werden, wenn dem Verbindungsabbau ein Setzen des entsprechenden Gewichtes auf 0 entspricht und dem Verbindungsaufbau einem Ändern des Gewichtes von 0 auf einen Wert ungleich 0.

Da noch keine Klarheit herrscht, nach welchen Kriterien der Auf- respektive Abbau der Verbindungen erfolgen soll, sind die Varianten (1) und (2) noch recht unerforscht.

Wie bereits dargestellt wird ein Großteil des Wissens in einerm neuronalen Netz verteilt repräsentiert; dies bedeutet, daß die Zustände einzelner Prozeßelemente bzw. ihre Verbindungen nicht mehr oder in nicht vertretbarem Aufwand in Symbole überführt werden können, die für einen Menschen verständlich sind. Die Problemlösungseigenschaften des neuronalen Netzes im Hinblick auf das spezielle Ausgangsproblem, für welches das neuronale Netz konstruiert wurde entstehen nur im Zusammenspiel der einzelnen Prozeßelemente. Der Lösungsweg kann deshalb nicht wie bei einem Expertensystem unter Bezugnahme auf die angewendeten Symbole erklärt werden. Das (Problemlösungs-) Wissen wird auf einer niedrigeren Abstraktionsebene, "unterhalb" der symbolischen Repräsentation dargestellt. Man spricht daher auch von **subsymbolischer** Verarbeitung (19/).

Eine auffallende Gemeinsamkeit der bekannten Lern-Algorithmen für neuronale Netze ist deren ausschließliches Zugreifen auf lokale Informationen, d. h. auf Informationen benachbarter Prozeßelemente. Außerdem ist interessant, daß sie zumeist ohne übergeordnete Instanz zu implementieren sind, d. h. ohne globale Ablaufsteuerung. Dadurch kann das Lernen in fast allen Fällen parallel und ohne globale Zugriffe vor sich gehen, was sich bei entsprechender Hardware sehr günstig auf den zeitlichen Aufwand auswirkt und sicherlich auch eher den Vorgängen beim Lernen des Menschen entspricht.

6.2.2.1 Verschiedene Formen des Lernens in neuronalen Netzen

Grundsätzlich kann man die verschiedenen Lernansätze nach der Art der Präsentation der zu erlernenden Muster einteilen. Hierzu unterscheidet man:

- Überwachtes Lernen oder Klassifizieren (supervised learning, learning by reinforcement):
  Bei diesem Lernmodus wird die tatsächliche Ausgabe des neuronalen Netzes mit der gewünschten Ausgabe verglichen. Die Gewichte werden dann durch das neuronale

Netz verändert, so daß bei der nächsten Iteration eine bessere Übereinstimmung mit dem Muster erreicht wird. Das Ziel sämtlicher Lernalgorithmen besteht in einer nachhaltigen Minimierung des Fehlers zwischen dem gewünschten und dem tatsächlichen Ausgabemuster durch kontinuierliche Modifizierung der Gewichte.

Beim überwachten Lernen muß das neuronale Netz vor seinem praktischen Einsatz "trainiert" werden. Das Training besteht darin, daß man dem neuronalen Netz Eingabe- und Ausgabedaten, eine Trainingsmenge, präsentiert; d. h. zu jedem gegebenen Eingabemuster präsentiert man dem neuronalen Netz das gewünschte Ausgabemuster.

Wenn der Lernvorgang abgeschlossen ist, dann werden die Gewichte "eingefroren", d. h. es kann dann nicht weiter gelernt werden und damit findet eine deutliche Unterscheidung zwischen Trainingsphase und Einsatzphase des neuronalen Netzes statt.

Beispiele überwachter Lernansätze sind:

- Selbstassoziation (Auto Association),
- Musterassoziation (Pattern Association),
- Klasseneinteilung (Classification Paradigm) und
- Fehlerpropagierung (Error (Back-) Propagation).

- Nicht überwaches Lernen oder Clustering (unsupervised learning, learning by doing): Bei diesem Lernmodus benötigen die neuronalen Netze keine externe Beeinflussung, um ihre Gewichte anzupassen. Dafür verfügen sie über eine interne Möglichkeit ihre Performanz zu überwachen. Das neuronale Netz sucht nach Regularitäten oder Tendenzen in den Eingabesignalen und führt Anpassungen bzgl. der Funktion des Netzes durch.

  Selbst wenn dem neuronalen Netz nicht mitgeteilt wird, ob es mit seinen Entscheidungen richtig liegt, so muß es doch über Information verfügen, wie es sich selbst organisieren soll.

  Ein Algorithmus zum nicht-überwachten Lernen könnte den Schwerpunkt auf die Kooperation zwischen Clustern von Prozeßelementen legen. In einem solchen Schema würden die Cluster miteinander arbeiten und versuchen, sich gegenseitig zu stimulieren.

  Ebenso könnte ein Wettbewerb zwischen den Prozeßelementen die Basis für das Lernen bilden (sogenanntes Wettbewerbslernen). Das Training miteinander in Wettbewerb stehender Cluster könnte die Antwort bestimmter Gruppen auf spezielle Reize verstärken und diese Gruppen miteinander und mit einer angemessenen Antwort verbinden.

6.2.2.2 Lernregeln

Im folgenden werden einige der bekanntesten Lernregeln für neuronale Netze kurz vorgestellt:

(1) Hebb-Regel:
Dies ist die älteste und am weitesten verbreitete Lernregel. Sie besagt: Wenn die Prozeßelemente a und b zugleich (wiederholt) stark aktiviert sind, so erhöhe die Stärke ihrer Verbindung.

(2) Die Delta-Regel:
Diese häufig verwendete Regel basiert auf der einfachen Idee, die Stärke der Verbindungen kontinuierlich zu modifizieren, um die Differenz (das Delta) zwischen dem gewünschten Ausgabewert und dem momentanen Ausgabewert eines Prozeßelements zu verringern.

Diese Regel wird auch **Widrow-Hoff-Lernregel** oder Methode der kleinsten Fehlerquadrate (least mean square learning rule) genannt.

(3) Gradienten-Abstiegs-Regel (Gradientenverfahren):
Dies ist ein mathematischer Ansatz, um den Fehler zwischen den aktuellen und den gewünschten Ausgaben zu minimieren. Die Gewichte werden um einen Betrag modifiziert, der proportional zum Wert der ersten Ableitung der Fehlerfunktion bzgl. der Gewichte ist.

Obwohl das Konvergenzverhalten sehr langsam ist, wird diese Methode sehr häufig verwendet. Die Delta-Regel ergibt sich als Spezialfall aus dieser Methode.

(4) Kohonens Lernregel:
Diese von Teuvo Kohonen entwickelte Lernregel wurde durch das Lernen in biologischen Systemen inspiriert. Sie wird nur in nicht-überwachten-Lernsituationen angewandt.

Bei dieser Lernregel besteht zwischen den Prozeßelementen ein Wettbewerb, um die Gelegenheit zum Lernen zu erhalten. Das Prozeßelement mit dem größten Ausgabewert wird zum Sieger erklärt und erhält die Möglichkeit, sowohl seine Mitbewerber zu hemmen, als auch seine Nachbarn zu stimulieren. Nur dem Sieger ist es gestattet eine Ausgabe zu erzeugen und nur der Sieger und dessen Nachbarn dürfen die Gewichte anpassen.

(5) Back Propagation Lernen:
Back-Propagation ist eine Verallgemeinerung der Delta-Regel auf Netzwerke mit beliebig vielen Layer und feed-forward-Verarbeitung. Die prinzipielle Idee ist, daß die Hidden-Prozeßelemente (d. h. jene Prozeßelemente, die weder Input- noch Output-Prozeßelemente sind) eine interne Repräsentation der Musterassoziationen durch Rückwärtspropagieren eines Fehlers vom Output-Layer in Richtung Input-Layer (error-back-propagation) erlernen.

Das Lernverfahren erfolgt in zwei Schritten: Zuerst wird das angelegte Muster in Richtung Output-Layer propagiert, um dort die Reaktion des Netzes auf das präsentierte Muster zu generieren. In der zweiten Phase erfolgt die Gewichtsänderung, abhängig vom Grad der Falschheit der Netzantworten. Die grundlegende Idee des Fehlerrücksendens hat bereits Rosenblatt (1962) formuliert.

Die Ausbreitung durch das Netz erfolgt schichtweise, d. h. die Aktivierungen der Prozeßelemente werden zuerst in jenem Hidden-Layer berechnet, der dem Input-Layer am nächsten liegt. Dann erfolgt die Berechnung der Aktivierung des nächsten, weiter beim Output-Layer gelegenen Layer, bis schließlich der Output-Layer selbst erreicht wurde.

In der zweiten Phase erfolgt die Fehlerbestimmung und die entsprechende Gewichtsveränderung, wiederum schichtweise. Dabei wird beim Output-Layer begonnen, da hier das gewünschte Muster zur Verfügung steht und mit dem tatsächlichen vom Netz produzierten Muster verglichen werden kann. Aus der Differenz dieser beiden Muster wird ein sogenanntes Fehlersignal gebildet, von dem einerseits die Änderung der Gewichte zwischen dem Output-Layer und dessen benachbartem Hidden-Layer und andererseits auch die Berechnung des neuen Fehlersignals für den nächsten Hidden-Layer abhängt.

Ist der Fehler bis zum letzten Hidden-Layer zurückgesendet und wurden dabei alle Gewichtsänderungen vorgenommen, kann wieder ein (neues) Muster angelegt und vorwärts propagiert werden. Wendet man diese Lernprozedur wiederholt an, so wird der Fehler schrittweise verringert. Es sei darauf hingewiesen, daß man andere, bereits erlernte Muster mit diesem Vorgehen zerstören kann ("Übertrainieren"). Der Lernvorgang wird, wenn der Fehler entsprechend klein geworden ist, als beendet angesehen.

Da es sich bei der Back Propagation um ein Gradientenverfahren handelt, kann sich das Netz in einem lokalen Minimum verfangen und die gestellte Aufgabe nicht fehlerfrei erlernen. Bei binären Aufgabenstellungen mag das nicht störend sein, da die Entscheidbarkeit (und nicht die Fehlerfreiheit) ausschlaggebend ist.

(6) Grossbergs Lernregel:
Grossbergs Lernregel kombiniert die Hebb-Regel mit dem biologischen Vorgang des Vergessens.

In Grossbergs Modell wird jedes neuronale Netz aus Instars und Outstars gebildet; ein Instar ist ein Prozeßelement das viele Eingaben empfängt, ein Outstar dementsprechend eines, das seine Ausgaben zu vielen anderen Prozeßelementen sendet. Hier erlauben die Verbindungen den Rückruf eines konzentrierten Bildes von einem einzigen Outstar-Knoten; das Muster wird verteilt gespeichert.
Falls ein Knoten sowohl eine hohe Input-, als auch eine hohe Outputaktivität aufweist, so werden die zugehörigen Gewichte signifikant verändert. Falls entweder der gesamte Input oder die Ausgaben kleine Werte annehmen, so werden die Veränderungen der Gewichte ebenfalls gering ausfallen und die Gewichte können auf unwichtigen Verbindungen sogar gegen 0 gehen.

Die Zeit spielt bei Grossbergs Lernmodell eine wesentliche Rolle. Falls ein Eingabereiz weggelassen wird, so wird im Laufe der Zeit, wenn das Vergessen einsetzt, auch die Antwort (in der Ausgabe) auf diesen Reiz nachlassen.

(7) Lernen in Cauchy und Boltzmann Maschinen:
Die hier verwendeten Lernprozeduren entsprechen eher stochastischen-, als deterministischen Lernprozeduren.

Ein Algorithmus für Boltzmann Maschinen kann folgendes beinhalten: In einem Temperaturgleichgewicht sind alle Zustände im neuronalen Netzwerk möglich, wobei deren relative Wahrscheinlichkeiten durch eine Boltzmann-Verteilung gegeben sind. Falls die Wahrscheinlichkeiten der Zustände im Modell denjenigen der Umwelt entsprechen, dann verfügt das neuronale Netz über ein abstraktes Modell der Umwelt.

6.2.3 Anwendungen neuronaler Netze

Bei der Führung komplexer technischer Prozesse (z. B. Müllverbrennungsanlagen) ist aufgrund der komplexen Systemzusammenhänge eine symbolische Wissensableitung bzgl. den notwendigen Eingriffsstrategien nicht mehr möglich. Diese Problematik ist in nachfolgender Abbildung anhand der Beurteilung von Situationen und der Auswahl der notwendigen Steuereingriffe dargestellt.

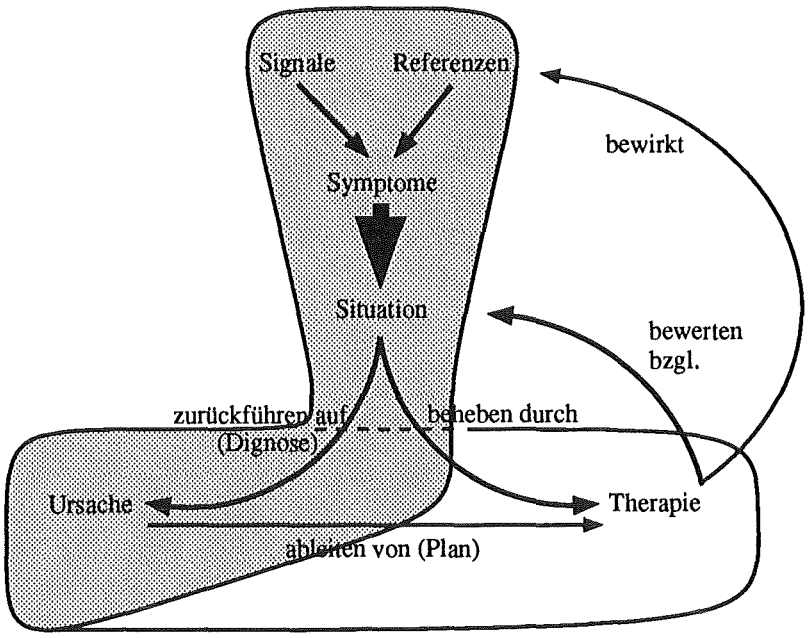

Abbildung 14

Für diesen Problemkreis bietet sich der Einsatz neuronaler Netze an, da sie das Wissen des Bedieners (mentales/heuristisches Modell bzgl. der Steuerung) anhand der verfügbaren Daten (Eingang, Ausgang, Eingriffe) lernen können und somit die Überführung des impliziten Wissens in eine explizite formale Form in manchen Fällen unnötig machen.

Die Lernfähigkeit, als herausragende Fähigkeit neuronaler oder konnektionistischer Systeme, wird dadurch erreicht, daß sie mit Hilfe einer netzähnlichen Struktur unter Verwendung von Gewichten eine Zuordnung von Elementen einer Zielmenge zu Elementen einer Eingabemenge vornehmen. Dabei können die Gewichte anhand ausgewählter Trainingssequenzen über Rückkopplungsmechanismen automatisch gelernt werden. Da der Zuordnungsprozeß über dieser netzähnlichen Struktur immer von einer großen Anzahl an Gewichten beeinflußt wird (das "Entscheidungswissen" ist verteilt), sind neuronale Systeme sehr Fehlertolerant. Da nach dem größtmöglichen Ähnlichkeitsmaß zugeordnet wird, können auch gestörte Eingabemuster dem "richtigen" Ausgabemuster zugeordnet werden.

Dem Einsatz bei der Führung technischer Systeme steht allerdings die Schwierigkeit gegenüber, daß ein neuronales Netz im allgemeinen nicht erklären kann, anhand welcher Merkmale es die Zurordnung getroffen hat. Das Wissen eines neuronalen Systems ist implizit (in den Gewichten verteilt) und nicht explizierbar. Diese Schwierigkeit könnte wiederum von **symbolischen Lernverfahren** behoben werden. Diese können bei vorgegebenen Primärattributen anhand von Trainingsbeispielen semantische Kategorien lernen und Eingaben entsprechend klassifizieren. Die Ergebnisse (Klassenzuordnungen) sind erklärbar. Transformationsfolgen im Sinne von Zeitreihen oder Situationsabläufen sind jedoch schwierig lernbar.

Eine gute Lernbarkeit, bzw. ein gutes Konvergenzverhalten und eine anschließend gute Interpolation dürfte durch ein topologieerhaltendes Netz (feature map /17/) erreicht werden können. Bei dynamischen Prozessen leitet sich der Bedienereingriff auch von der Historie ab, deshalb sind Situationsfolgen (Zeitreihen von t_k bis t_{k-i}) oder Änderungsgrößen (d/dt) einzuspeisen.

Möglichkeiten zum Einsatze neuronaler Netze sind:

- Lernen von notwendigen Prozeßeingriffen anhand erfolgreich durchgeführter Bedienerhandlungen
- Zuordnung von Prozeßsituationen und Prozeßeingriffen zu späteren Analysewerten (z. B. Dioxin in der Müllverbrennung).
- Nachbildung des dynamischen Verhaltens eines dynamischen Systems und der Möglichkeit Eingriffe durchzuführen (Simulator).
- Optimierung der Prozeßeingriffe durch eine Gewichtsanalyse (partielle Abhängigkeit der Ausgänge von den Eingängen).
- Analyse der Gewichte bzgl. der Übertragbarkeit in Regelform (Fuzzy-Regeln).

Die Struktur, um Bedienerhandlungen bei bestimmten Prozeßsituationen zu lernen, zeigt die folgende Abbildung.

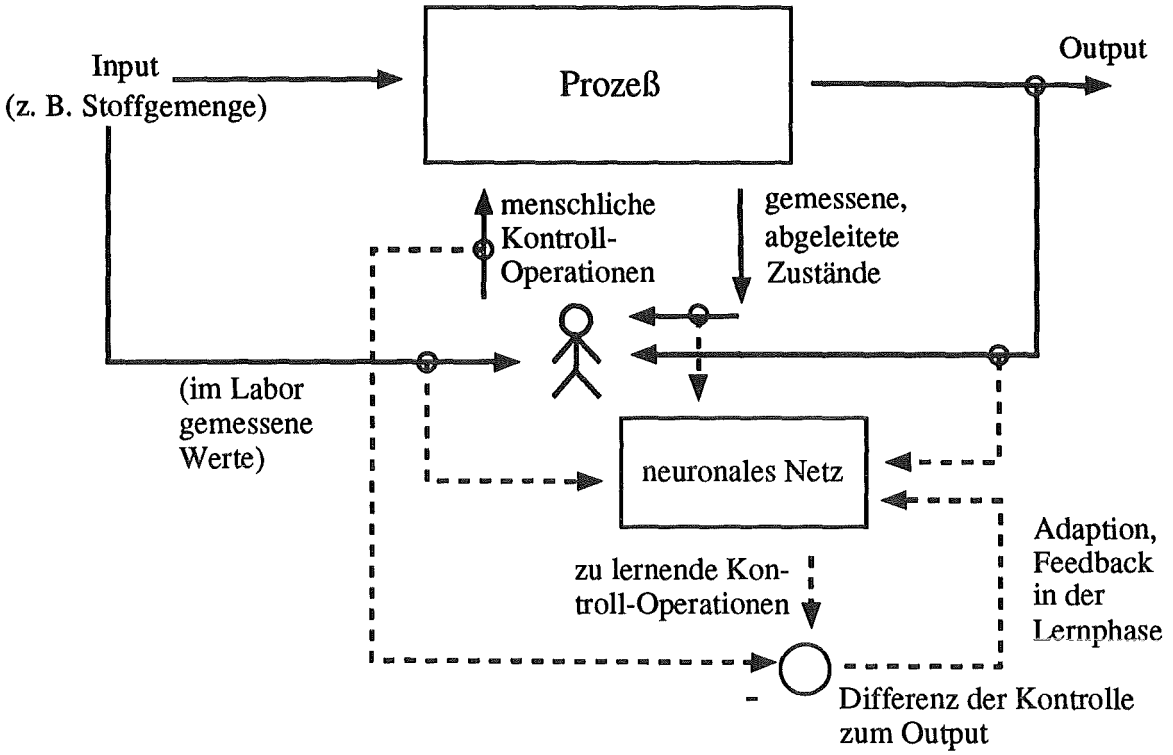

Abbildung 15

Für die beiden zuletzt genannten Bereiche existieren zur Zeit noch keine allgemein anwendbare Verfahren. Die Anwendbarkeit heuristischer Modelle und die entsprechenden KI-Methoden im allgemeinen und für die Steuerung dynamischer Systeme ist in /15/ beschrieben.

Referenzen

/1/ James F. Allen,
"Maintaining knowledge about temporal intervals,"
Communications of the ACM, Vol. 26, No. 11, S. 832 - 843, November 1983.

/2/ Constantin von Altrock
"Über den Daumen gepeilt - Fuzzy Logic: scharfe Theorie
der unscharfen Mengen,"
c't, Heft 3, S. 188 - 200, 1991.

/3/ Bruno Buchberger, Bernhard Kutzler,
"Computer-Algebra für Ingenieure,"
In: Buchberger et al., "Rechnerorientierte Verfahren,",
B. G. Teubner Verlag, Stuttgart, S. 11 - 69, 1986.

/4/ Daniel Dennett,
"Cognitive wheels: the frame problem of AI,"
In: Christopher Hookway (Ed.)
"Minds machines and evolution,"
Cambridge university Press, S. 129 - 151, 1984.

/5/ Werner Dilger,
"Expertensysteme - eine Einführung,"
Computer Magazin, 1-2, S. 30 - 36, 1991.

/6/ Ulf J. Froitzheim,
"Entfesselte Querdenker,"
highTech, November, S. 40 -47, 1990.

/7/ Michael R. Genesereth, Matthew L. Ginsberg,
"Logic Programming,"
Communications of the ACM, Volume 28, Number 9, S. 933 - 941, Sept. 1985.

/8/ Ch. Habel,
"Repräsentation von Wissen,"
Informatik Spektrum 13, S. 126 - 136, 1990.

/9/ B. Heinemann, K. Weihrauch,
"Logik für Informatiker,"
B. G. Teubner, Stuttgart, 1991.

/10/ P. N. Johnson-Laird, D. J. Herrmann, R. Chaffin,
"Only connections: A critique of semantic networks,"
Psychological Bulletin Vol. 96, No. 2, S. 292-315, 1984.

/11/ Werner Karbach, Marc Linster,
"Wissensakquisition für Expertensysteme,"
(Techniken, Modelle, Softwarewerkzeuge)
Carl Hanser Verlag, München Wien, 1990.

/12/ Hubert B. Keller, Thomas Weinberger,
"Ein kognitiver Ansatz zur Bestimmung von Ursache-Wirkungs-Beziehungen und Schlußfolgerungen in komplexen technischen Prozessen,"
Vortrag 5. Fachtagung Maschinelles Lernen, Osnabrück, 15. & 16. Juli 1992.

/13/ Hubert B. Keller, Th. Weinberger,
"Heuristische Modellierung - ein Arbeiten mit Hypothesen?"
Modellierung und Simulation im Umweltbereich, Beiträge zum Workshop Rostock, 25. & 26. Juni 1992.
Rostock, Universität Rostock, Fachbereich Informatik, S. 23 - 32, 1992.

/14/ Hubert B. Keller, Thomas Weinberger,
"Lernmodelle und Wissensverarbeitung,"
KfK - Bericht 5002, Kernforschungszentrum Karlsruhe, März 1992.

/15/ H. B. Keller, Th. Weinberger, E. Kugele, A. Jaeschke, B. Osterhues,
"Verbesserung der Betriebsführung technischer Anlagen durch den Einsatz von Methoden der Künstlichen Intelligenz - Projektbeschreibung INPRO,"
KfK - Bericht 5049, Kernforschungszentrum Karlsruhe, März 1992.

/16/ George J. Klir, Tina A. Folger,
"Fuzzy Sets, Uncertainty and Information,"
Prentice Hall, 1988.

/17/ Teuvo Kohonen,
"The Self-Organizing Map,"
Proceedings of the IEEE, Vol. 78, No. 9, S. 1464 - 1480, 1990.

/18/ Wolfgang Kreutzer,
"Grundkonzepte und Werkzeugsysteme objektorientierter Systementwicklung,"
Wirtschaftsinformatik, 32. Jahrgang, Heft 3, S. 211 - 227, 1990.

/19/ Karl Kurbel, Wolfram Pitsch,
"Eine Beurteilung konnektionistischer Modelle auf der Grundlage ausgewählter Anwendungsprobleme und Vorschläge zur Erweiterung,"
Wirtschaftsinformatik, Vieweg Verlag, Stuttgart, Heft 5, S. 355 - 364, Okt. 1991.

/20/ Richard P. Lippmann,
"An introduction to computing with neural nets,"
IEEE ASSP Magazine, S. 4 - 21, April 1987.

/21/ Hubert Marchand,
"Aus der Praxis der Wissensakquisition,"
Künstliche Intelligenz: Forschung, Entwicklung, Erfahrung
Themenheft "Wissensakquisition,"
Oldenbourg Verlag, KI 2, S. 61 - 63, 1990.

/22/ Marylin McCord Nelson, W. T. Illingworth,
"A practical Guide to Neural Nets,"
Addison-Wesley Publishing Company, Inc. N.Y., 1990.

/23/ Ryszard S. Michalski, Jaime G. Carbonell, Tom M. Mitchell,
"Machine Learning - An Artificial Intelligence Approach,"
Springer Verlag, New York, 1984.

/24/ H. Ono, T. Ohnishi, Y. Terada,
"Combustion Control of Refuse Incineration Plant by Fuzzy Logic,"
Fuzzy Sets and Systems 32, S. 193 - 206, North Holland, Amsterdam, 1989.

/25/ Frank Puppe,
"Einführung in Expertensysteme,"
Studienreihe Informatik, Springer Verlag, 1988.

/26/ Michael M. Richter, Oliver Wendel,
"Lernende Systeme,"
Vorlesungsskript Universität Kaiserslautern, WS 1990/91.

/27/ P. Schlüter, A. Behdjati, P. Fleischer, S. Bagdon,
"Objektorientierte Software-Entwicklung: Konzepte und Terminologie,"
Siemens AG, Zentralabteilung Forschung und Entwicklung, München.

/28/ Cosima Schmauch,
"Wissensrepräsentation, Grundkurs,"
In: Thomas Christaller (Ed.)
5. KIFS 87, Interne Fachberichte 202
Springer Verlag, S. 103 - 156, 1987.

/29/ Zhongzhi Shi,
"Foundations for Knowledge Systems,"
In: Y. Ci, C. Zhang, C. Sun
"New Generation Computing,"
North Holland, S. 168 - 174, 1990.

/30/ Manfred Stede,
"Einführung in die Künstliche Intelligenz,"
Heise Verlag, Hannover, 1987.

/31/ Sigmar-Olaf Tergan,
"Diagnose von Wissensstrukturen,"
DIFF, Forschungsbericht 30, 1984.

/32/ Thomas Weinberger,
"Maschinelles Lernen - "klassische", konnektionistische und kognitive Konzepte,"
KfK - Bericht in Vorbereitung.

/33/ Patrick Henry Winston,
"Künstliche Intelligenz,"
Addison Wesley, 1987.

Stichwortverzeichnis

A

Abstraktion
  Abstraktionsprozeß, 36
Adäquatheit, 10
Aktivierungsfunktion, 85
Algorithmus, 3
Analogieprobleme
  geometrische, 49
Annahmen, 8
Approximate Reasoning, 13, 14
Aussagenlogik, 8, 11, 65
Axiome, 8
A-Box, 20

B

Backtracking, 30, 51
Back Propagation, 91
Bayessche Formel, 27
Bayes-Theorem, 72, 74
  Verringerung Fehlerquellen, 28
  Voraussetzungen, 27
Belief-Revision-System, 29
Bildanalyse, 51

C

Cauchy und Boltzmann Maschinen, 93
CHECKERS, 8
Closed World Assumption, 29, 30
Clustering, 90
Computer-Algebra, 23
  Bedeutung, 24
Constraints, 25, 32
  Constraintsystem
    Lösungstechniken, 26
  Constraint-Netz, 25
  Constraint-Propagierung, 25

D

dangling Pointers, 46
Deduktionssystem, 60
Defuzzyfizierung, 15
Delta-Regel, 91
Dempster-Shafter Theorie, 28, 72
Diagnostik, 31
  Diagnosehierarchien, 70
  Diagnosesystem
    Anforderungen, 69
  Diagnostiksysteme
    Bewertungsarten, 74
    Klassen, 72
    Modelle, 73
  diagnostischer Mittelbau, 69, 73, 74
  Differentialdiagnostik, 72
  Eigenschaften, 68
  Grundalgorithmus zur Bewertung, 27
  Problembereiche, 68
  Unsicherheitskategorien, 26
  Unsicherheitsquellen, 26
dynamischer Typ, 42
dynamisches Binden, 41
dynamische Programmierung, 57

E

Effizienz, 10
Entscheidbarkeit, 10, 11
  Semi-Entscheidbarkeit, 29, 63
Expertensystem, 15, 26, 31
  Wissensakquisition, 81
    Phasenmodell, 82

F

Facette, 19
Fakten, 8
feature map, 94
feedback-Netz, 87
Feed forward-Netz, 87
Fehlerpropagierung, 90
Frame, 17, 19
Frameproblem, 64
  Frameaxiome, 64
Frame Representation Language (FRL), 19
Fuzzyfizierung, 14
Fuzzy Control, 12
Fuzzy Logik, 12
Fuzzy Mengen, 12
Fuzzy Reasoning, 12
Fuzzy-Regeln, 94

G

General-Problem-Solver (GPS), 59
Generieren-und-Testen, 60
Gewichte, 84, 88, 94
Gradienten-Abstiegs-Regel, 91
Graesser, 22
Greenscher Trick, 64
Grossbergs Lernregel, 92

H

Halteproblem, 63
Heap, 45
    Heapverwaltung, 46
Hebb-Regel, 91
Heterarchie, 71
Hierarchie
    kompositionelle, 76
    taxonomische, 76
Horizonteffekt, 59
Hornklauselkalkül, 10

I

Inferenzregel
    Abduktion, 68
    Deduktion, 69
    Modus ponens, 8, 63
    Modus tolens, 63
    Resolution, 63
Inferenzstrategie
    Rückwärtsverkettung, 16
    Vorwärtsverkettung, 16
Input-Signal, 84
Instantiierung, 37
Instanzen, 34, 37
Instanzvariablen, 34, 37, 43

J

logische Junktoren, 64

K

Kalkül, 8, 10
    Resolutionskalkül, 11
Kanten, 17, 19, 50
Klassen, 34, 36, 37, 38, 43
    abstrakte, 40
    generische, 47
    Klassenhierarchie, 44
    Klassenmethode, 44
    Klassenrumpf, 36, 40
    Klassenspezifikation, 36
    Klassenvariablen, 45
    Metaklassen, 44
    Oberklasse, 39, 40
    Unterklasse, 39, 40, 41
    Zugehörigkeit, 37
Klassifikation, 36
    automatische, 20
KL-ONE-Sprachen, 20
    Vor-/Nachteile, 21
Knoten, 17, 19, 50, 84
Kognition
    kognitiver Rahmen, 20
Kohonens Lernregel, 91
kombinatorische Explosion, 10, 63
Konfliktlösungsstrategie, 16, 60
Konstruktionsprinzip, 46
Kontext, 35
Kontrollstrategie
    Anwendungsgebiet, 70
    Establish-Refine, 70
    Hypothesize-and-Test, 70
    Rückwärtsverkettung, 70
    Vorwärtsverkettung, 70
    Vor-/Nachteile, 70
Korrektheit, 10
Kunde, 37, 39, 43, 47
Künstliche Intelligenz, 52

L

Landmarkewerte, 78
Laufzeit, 48
Laufzeitfehler, 42
Laufzeitsystem, 46
Layer, 87
    Hidden-Layer, 87
    Input-Layer, 87
    Output-Layer, 87
Least-Commitment-Strategie, 76, 77
Lernen
    parametrisches, 8
Lernfunktion, 86
linguistische Variable, 14, 15
LISP, 18
Logik, 8
    dreiwertige Logik, 29
    Fuzzy Logik, 12
    Grenzen, 64
    klassische Logik, 26
    Modallogik, 11
    nichtmonotone Logik, 12
    Temporäre Logik, 12
    Vor-/Nachteile, 62
LONGS System, 79

M

Mächtigkeit, 10
MAX-MIN-Inferenz, 15
MAX-PROD-Inferenz, 15
Means-Ends-Analysis, 59. *Siehe auch* Mittel-Zweck-Analyse
Medi-Modell, 28
Merkmalsvektoren, 7
Metapläne, 77
Methoden, 34, 43
Methodenaufruf, 35
Mittel-Zweck-Analyse, 59
MOLGEN, 76
MOPS, 22
Musterassoziation, 90

N

Nachrichten, 19, 34, 37
Negation as Failure, 29
Netzwerk
  Additions-Multiplikationsnetz, 51
  semantisches Netzwerk, 17, 19
    Vor-/Nachteile, 17
Neuron, 84
Newell, Shaw & Simon, 59
nicht-monotones Schließen, 26, 29
numerische Mathematik, 23

O

Oberklasse, 38, 43
objektbasiert, 33
Objekte, 17, 34, 35, 37, 38
  Objektverwaltung, 46
objektorientiert, 42
Output-Funktion, 86

P

Planen
  einstufiges, 76
  hierarchisches, 76
  mehrstufiges, 76
  nicht-hierarchisches, 76
  Planer
    lineare, 76
    nicht-lineare, 76
  Planungstechniken, 76
Plausibles Schließen, 13
Polymorphismus, 34, 40, 41
  polymorphe Variable, 41
Postsches Korrespondenzproblem, 11
Prädikatenlogik
  erster Ordnung, 9, 10, 11, 63
  zweiter Ordnung, 9, 11
Prädisposition, 72
probabilistisches Schließen, 26
  Basis, 26
Problem
  Problemlösungsstrategien, 49, 59, 63, 65, 67
  Problemlösungstypen, 5, 67
    Konstruktion
      Design (Konfigurierung), 75
      Planung, 75
  Problemreduktion, 50
  Problemtyp, 67
Produktionssystem, 16, 62
Programm
  konventionelles, 3
Programmierung
  objektbasierte, 34
  strukturierte, 33
PROLOG, 10, 29
Propagierung
  Abhängigkeitsgesteuertes Rückziehen, 51
  chronologisches Rücksetzen, 52
  Haupteigenschaften, 51
  lokale Berechnungen, 51
  nichtchronologisches Rückziehen, 52, 65
  numerische Beschränkungen, 50
  symbolischer Beschränkungen, 50
  Vorgehensweise, 26
Protokoll, 35

Q

Quantoren, 9

R

Reason-Maintenance-System, 29
Rechtsrekursion, 10
Regelgröße, 14
Regeln, 16
  Antezedenz-Konsequenz-Regeln, 61
  auslösen (triggern), 60
  Ausnahmen, 29
  feuern (fire), 60
  Regelinterpreter, 16
rekursive Systeme, 87
representational bias, 6
RETE-Algorithmus, 16
Rosenblatt, 92. *Siehe auch* Back Propagation
Rückwärtsverkettung, 61

S

Samuels, 8
Schank, 22
Schank & Abelson, 22
Schlußfolgerungen, 8
Schwellenwert, 85, 86
Seiteneffekt, 62
Selbstassoziation, 90
sigmoid, 86
Simulation, 77
    Simulationstypen
        analytische, 78
        numerische, 77
        qualitative, 78
Skelett-Pläne, 77
Skripten, 22
    Skript-Variablen, 22
    Skript-Wissen, 22
Slots, 19. *Siehe auch* Leerstellen (Schema)
SMALLTALK, 19, 44
Software
    inkrementelle Entwicklung, 46
    objektbasiert, 35
    objektorientiert, 42
    prozedurorientiert, 35
    prozedurorientierte, 45
    statische, 36
Speicherverwaltung, 45
Spielbaum, 58
statischer Typ, 42
Stellgröße, 14
Subsumption, 21
subsymbolische Verarbeitung, 89
Suchen
    allgemeine Prinzipien, 52
    Breitensuche, 71
    Kostenmaß, 56
    Tiefensuche, 59, 71
Suchstrategie
    Alpha-Beta-Verfahren, 58
    A*, 57
    Bergsteigen, 54
    Bestensuche, 55
    Breitensuche, 54
    British Museum Prozedur, 56
    dynamisches Programmieren, 57
    fortschreitende Vertiefung, 58
    heuristisches Abschneiden, 59
    heuristische Fortsetzung, 59
    Minimax-Idee, 58
    Strahlensuche, 55
    Tiefensuche, 53
    Verzweige und Begrenze, 56
Summationsfunktion, 85
supervised learning, 89
Symptom-Diagnose-Wahrscheinlichkeit, 27
Synapse, 86
System
    framebasiertes, 71
    objektbasiertes, 33
    objektorientiertes, 42
    regelbasiertes, 16, 60, 71
        Hauptnachteil, 16

T

Tabellenkalkulationsprogramme., 50
Teilzielbildung, 49
Temperaturgleichgewicht, 93. *Siehe auch* Cauchy und Boltzmann Maschinen
temporales Schließen, 31
    Zeitrepräsentation, 31
Theoreme, 8
topologieerhaltendes Netz, 94
trainieren, 88
Truth-Maintenance-System, 29, 32
Typkonzept, 36
T-Box, 20

U

Übertragungsfunktion, 85
UND/ODER-Bäume, 50, 61
Unifikation, 10, 64
unsupervised learning, 90
Unterklasse, 38, 43

V

Validierung, 81
Vererbung, 34, 38, 40, 47
    Erben, 39
    Mehrfachvererbung, 39
    multiple, 44
    selektive, 43
    Vererbungshierarchie, 18, 39
    Vererbungskonzept, 42
        Bewertung, 39
    Vererbung und Export, 44
Verknüpfung, 14
Verknüpfungsoperatoren, 8
Vollständigkeit, 10, 11
Vorgebirgs-Problem, 54
Vorwärtsverkettung, 61

W

Wettbewerbslernen, 90

Widrow-Hoff-Lernregel, 91. *Siehe auch*
Delta-Regel
Wissen
deklaratives, 18
Dimensionen, 4
episodisches, 22
fallvergleichendes, 68
kausales, 68
prozedurales, 18
statistisches, 68
Wissensakquisition
Bedeutungen, 81
Grundarten, 81
wissensbasiertes System, 3
Wissenserwerb, 75
Wissensingenieur, 82
Wissensrepräsentation, 5, 69
Auswahlkriterien, 6
objektorientiert
Kernideen, 18

X

XOR-Problem, 86

Z

Zadeh, 12
Zeitdatenbank, 31
Zielbaum, 50
Zielreduktion, 50, 52, 59