

Kapitel 5b: Computermodele des Problemlösens

Ute Schmid

Anzahl der Anschläge (mit Leerzeichen): ca. 120263

Anschrift der Autorin:

Fachbereich 6 – Mathematik/Informatik
Universität Osnabrück
Albrechtstrasse 28
D-49076 Osnabrück

Tel.: ++49-541-969-2558, Fax: -2779
email: schmid@informatik.uni-osnabrueck.de
Homepage: <http://www.inf.uos.de/schmid/>

Deep Blue versus Kasparov – Maschinelle versus menschliche Intelligenz. Im Jahr 1957 prophezeite Herbert Simon – einer der Gründerväter der Künstlichen Intelligenz –, dass in etwa zehn Jahren ein Computer Schachweltmeister sein wird. Dreißig Jahre später, im Frühjahr 1997, war es dann so weit: *Deep Blue* besiegte den damaligen Schachweltmeister Garry Kasparov. Ist es gelungen, menschliche Intelligenz auf einem Computer nachzubilden? Jein! *Deep Blues* Mikroprozessoren können 200 Millionen Züge pro Sekunde berechnen; seine Datenbanken verfügen über unzählige Endspiele und Eröffnungen; mit einer ausgeklügelten Funktion wird die Stärke von Spielpositionen bewertet; verschiedene Spielstrategien wurden programmiert. Kasparov hat ein Gehirn mit etwa 10^{10} Neuronen und etwa 10^{14} Verschaltungen zwischen Neuronen; in seinem Gedächtnis hat er aufgrund langjähriger Spielpraxis viele Endspiele und Eröffnungen gespeichert; aufgrund seiner Erfahrung kann er die Stärke von Spielpositionen bewerten; er kann verschiedene Spielstrategien anwenden.



Das strategische Spiel Schach stellt eine spezielle Problemlöseaufgabe dar, bei der es darum geht, jeweils den Spielzug auszuführen, der am meisten Vorteil verschafft, um das Spiel am Ende zu gewinnen. Computermodelle des Problemlösens, in der Künstlichen Intelligenz wie in der Kognitiven Psychologie, basieren auf der Grundannahme der Suche in einem Problemraum. Beim Schach entsprechen die Zustände des Problemraums den Spielpositionen. Ein Zustand kann durch Ausführung eines Zuges in einen Folgezustand überführt werden. Die wesentliche Stärke eines Computers im Vergleich zum Menschen hat wenig mit Intelligenz zu tun – ein Computer kann sehr schnell sehr viele Daten verarbeiten – das heißt, er kann einen größeren Ausschnitt des Problemraums in schnellerer Zeit durchsuchen. *Deep Blue* ist Kasparov darin überlegen, dass er ausgehend von der aktuellen Spielposition wesentlich mehr Züge im voraus berechnen kann. Mit *brute force* – also “blinder Gewalt” – kann er viele mögliche Züge und Gegenzüge generieren und dann den besten auswählen. Ein menschlicher Schachexperte wird dagegen strategisch vorgehen und nur solche Züge betrachten, von denen er annimmt, dass sie zum Erfolg führen – er sucht also unter Einbezug von Wissen.

Aber, die Fachwelt konstatiert, dass *Deep Blue* mehr kann als seine Vorgänger. Kasparov zollte *Deep Blue* Respekt und forderte den Nachweis, dass er wirklich gegen einen Computer und nicht gegen einen Menschen gespielt hatte. Die “menschliche” Spielweise von *Deep Blue* hat ihre Ursache in der ausgeklügelten Bewertungsfunktion, die vor allem den Analysen des Schach-Großmeisters Joel Benjamin zu verdanken ist. Wie Kasparov ist *Deep Blue* also in der Lage, Spielpositionen sinnvoll zu bewerten. Die Analyse, welche Wissensstrukturen notwendig sind, um ein Problem erfolgreich zu lösen, ist eine wesentliche Aufgabe der kognitiven Modellierung. Für den sehr eingeschränkten, aber ohne Zweifel komplexen Bereich des Schachs dient *Deep Blue* als positives Beispiel dafür, dass ein Computerprogramm, das wenigstens in einigen Aspekten menschliches Problemlösen imitiert, auch zu einem gewissen Grade intelligentes Verhalten zeigen kann.

<i>5b – Computermodelle des Problemlösens</i>	3
<i>Textbox Deep Blue versus Kasparov – Maschinelle versus menschliche Intelligenz</i>	2
5b-1 Einleitung: Computermodellierung kognitiver Prozesse	5
5b-2 Problemlösen als Suche im Problemraum	6
5b-2.1 Problemzustände	7
5b-2.2 Problemlöse-Operatoren	8
5b-2.3 Problemraum	9
5b-2.4 Das Turm-von-Hanoi Problem	9
5b-3 Suchstrategien	13
5b-3.1 Uninformierte Suche	13
<i>Textbox Rekursive Probleme</i>	14
5b-3.2 Hill Climbing und Bewertungsfunktionen	15
5b-3.3 Mittel-Ziel Analyse	17
5b-4 Produktionssysteme	19
<i>Textbox Problemlösen und Planen in der KI</i>	20
5b-4.1 Mustervergleich	21
5b-4.2 Konfliktlösung	21
5b-4.3 Regelanwendung	22
5b-4.4 Das Produktionssystem ACT	23
5b-5 Modellierung spezieller Aspekte des Problemlösens	24
5b-5.1 Analoges Problemlösen	24
<i>Textbox Kognitive Architekturen und Spezielle Modelle</i>	25
5b-5.1.1 Gentners Theorie des Strukturvergleichs	26
5b-5.1.2 Die <i>Structure Mapping Engine</i>	27
5b-5.2 Erwerb von Problemlösefertigkeiten	28
5b-5.2.1 Verstärkung und Kombination von Regeln	28
5b-5.2.2 Erwerb von Strategien	29
5b-5.3 Problemlösen und Wissen	30
5b-6 Ausblick	30
Kernsätze	31
Schlüsselbegriffe	32
Weiterführende Literatur	33
Einschlägige Journals	34
Webseiten	34
Literatur	35

Abbildungsverzeichnis

5b-1 Computermodellierung als Forschungsthema in Kognitiver Psychologie und Künstlicher Intelligenz	5
5b-2 Ein Affe-Banane Problem: Anfangszustand (1), Zwischenzustand, nachdem der Affe zur Kiste gelaufen ist (2), Zwischenzustand, nachdem der Affe die Kiste unter die Banane geschoben hat (3) und Zielzustand, nachdem der Affe auf die Kiste gestiegen ist und die Banane genommen hat (4)	7
5b-3 Zustandstransformation	9
5b-4 Problemraum für das Affe-Banane Problem	10
5b-5 Turm-von-Hanoi Problem mit drei Scheiben	11
5b-6 Problemraum des Turm-von-Hanoi Problems mit drei Scheiben (Bewegungen von A gepunktet, Bewegungen von B gestrichelt, Bewegungen von C durchgezogen)	12
5b-7 Exponentielles Wachstum beim Turm-von-Hanoi	12
5b-8 Abarbeitung der rekursiven Funktion <i>hanoi</i> für drei Scheiben	13
5b-9 Durch Tiefensuche erzeugter Suchbaum für das Affe-Banane Problem	15
5b-10 Das Problem lokaler Maxima beim <i>Hill Climbing</i>	16
5b-12 Das Achterpuzzle	16
5b-11 Das “ <i>Hobbits and Orcs</i> ” Problem	17
5b-13 Verarbeiten von Teilzielen mit einem <i>Stack</i>	17
5b-14 Architektur eines Produktionssystems	21
5b-15 Lösung des “Kaffe-Dose” Problems mit einem Produktionssystem	22
5b-16 Repräsentation eines Ausschnitts von Wissen über ein Sonnensystem (a) und ein Atom (b) als Struktur aus Objekten und Relationen zusammen mit einer strukturerehaltenden Abbildung für die Rutherford Analogie “Das Atom ist wie das Sonnensystem” (nach Gentner, 1983)	27

Tabellenverzeichnis

5b-1 Formale Darstellung des Affe-Banane Problems	8
5b-2 Lösung des Affe-Banane Problems mit Mittel-Ziel Analyse	18
5b-3 Mittel-Ziel Analyse	18
5b-4 Lösung des Turm-von-Hanoi Problems mit Mittel-Ziel Analyse	19
5b-5 Repräsentation des deklarativen Wissens für ein Turm-von-Hanoi Problem in ACT-R	23
5b-6 Ausschnitt der Repräsentation des prozeduralen Wissens für ein Turm-von-Hanoi Problem in ACT-R	24
5b-7 Herstellung der Rutherford Analogie mit der SME (Elemente der besten globalen Zuordnung sind fett markiert)	28

5b-1 Einleitung: Computermodellierung kognitiver Prozesse

Computermodellierung kognitiver Prozesse ist nicht allein Forschungsthema der Psychologie, sondern liegt im Schnittbereich zwischen Kognitiver Psychologie und Künstlicher Intelligenz (KI) – als interdisziplinärer Beitrag zur **Kognitionswissenschaft** (Strube, Habel, Hemforth, & Konieczny, 2000) (siehe Abb. 5b-1). Im Rahmen der Kognitiven Psychologie ist die Erstellung von prozessorientierten Modellen ein die empirische Forschung ergänzender methodischer Zugang zur menschlichen Informationsverarbeitung (Strube, 2000). Empirische Untersuchungen dienen der *Überprüfung* von theoretischen Annahmen. Computermodelle dienen der *Präzisierung* theoretischer Annahmen: Ein ablauffähiges Programm muss **vollständig** sein, das heißt, sein Verhalten muss für alle Bedingungen, die auftreten können, spezifiziert sein, und es muss **konsistent** sein, das heißt, es darf keine widersprüchlichen Anweisungen enthalten.

Im Rahmen der Künstlichen Intelligenz geht es darum, Algorithmen zur maschinellen Lösung von Problemen zu entwickeln. Entscheidend ist vor allem, dass ein Computerprogramm bestimmte Probleme automatisch lösen kann und nicht unbedingt, dass die Art, in der das Problem gelöst wird, Ähnlichkeit mit menschlicher Informationsverarbeitung aufweist. Genauere Kenntnis menschlicher Kognition kann aber auch für die KI-Forschung notwendig sein: (1) zur Verbesserung der Mensch-Computer Interaktion im Sinne verständlicher Dialog-Schnittstellen; (2) zur Integration von problemlöserrelevantem Wissen, zum Beispiel bei **Expertensystemen** und strategischen Spielen; und (3) als Ideengeber für den Entwurf von Algorithmen für Problembereiche, für die noch kein Ansatz zur automatischen Problemlösung existiert.

Die *Forschungsmethoden* und *-ziele* von KI und kognitiver Psychologie sind also unterschiedlich: In der KI geht es vor allem darum, Informationsverarbeitungsprozesse zu formalisieren und als (effiziente) Programme auf einem Computer zu implementieren. Typische Programmiersprachen der KI sind Prolog und Lisp (Schmid &

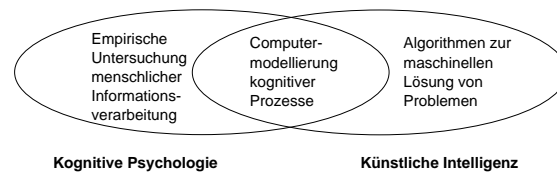


Abb. 5b-1: Computermodellierung als Forschungsthema in Kognitiver Psychologie und Künstlicher Intelligenz

Kindsmüller, 1996). In der kognitiven Psychologie geht es dagegen darum, Theorien und Modelle menschlicher Informationsverarbeitung zu formulieren und empirisch zu überprüfen. Die *Inhaltsbereiche*, die in der KI-Forschung abgedeckt werden, überlappen sich dagegen stark mit denen der kognitiven Psychologie. Gemeinsame Inhaltsbereiche sind insbesondere Wahrnehmung (Bildverstehen), Wissensrepräsentation, Sprachverstehen und -produktion, Schlussfolgern (Inferenz), Problemlösen und Lernen. Einen Überblick über die Bereiche der KI-Forschung gibt beispielsweise das Handbuch der Künstlichen Intelligenz (Görz, Rollinger, & Schneeberger, 2000). In diesem Kapitel betrachten wir Computermodelle des Problemlösens und konzentrieren uns auf Ansätze, die in der *kognitiven* Modellierung verwendet werden.

Bei weitem nicht jedes Computerprogramm, das Probleme löst, die auch Menschen lösen können, simuliert menschliche Informationsverarbeitung. Die auf dem Computer ablaufenden Prozesse können mit den Annahmen, die über menschliche kognitive Prozesse formuliert wurden, auf **funktionaler** oder auf **struktureller** Ebene verglichen werden. Eine Methode zur Prüfung der funktionalen Äquivalenz ist der **Turing-Test** (Turing, 1950): Dabei kommuniziert ein Beurteiler mit zwei verschiedenen Terminals. Einer der Terminals ist mit einem Computersystem verbunden, auf dem ein Simulationsprogramm läuft; der andere Terminal ist mit einem Computersystem verbunden, an dem ein Mensch sitzt. Wenn der Beurteiler nicht entscheiden kann, welche Reaktionen vom Menschen und welche vom Simulationsprogramm gegeben werden, so hat das Simulationsprogramm den Turing-Test bestanden. Der Schachcomputer *Deep Blue* (siehe Textbox *Deep*

Blue versus Kasparov) hat den Turing-Test auf gewisse Art bestanden, als Weltmeister Kasparow die Vermutung äußerte, dass bei *Deep Blues* Sieg über ihn menschliche Intervention im Spiel war.

Wenn, wie beim Turing-Test, nur geprüft wird, ob ein Programm für bestimmte Eingaben Ausgaben erzeugt, die mit menschlichen Antworten übereinstimmen (funktionale Übereinstimmung), ist jedoch noch nichts darüber gesagt, ob diese Ausgaben durch ähnliche Prozesse generiert werden (strukturelle Übereinstimmung). Um ein Computermodell hinsichtlich der ablaufenden *Prozesse* mit menschlicher Informationsverarbeitung zu vergleichen, sollte geprüft werden, inwieweit empirisch erhobene Daten – beispielsweise Reaktionszeiten oder Fehlerraten – mit den entsprechenden Antwortmustern des Computermodells übereinstimmen. Aber selbst, wenn der Vergleich von Simulationsprogramm und menschlicher Informationsverarbeitung auf detaillierten Prozessanalysen basiert, ist Vorsicht geboten: Bei der Umsetzung eines Prozessmodells in ein Computerprogramm hat man viele Freiheitsgrade und im Allgemeinen müssen bei der Programmierung Zusatzannahmen getroffen werden, die unabhängig vom zugrundeliegenden Modell sind (Cooper, Fox, Farringdon, & Shallice, 1996). Es empfiehlt sich also, Modellannahmen und Zusatzannahmen im Programm möglichst klar zu trennen und beim Vergleich von Daten und Modell anzugeben, welche Verarbeitungsschritte das Antwortmuster des Programms erzeugt haben.

Eine Möglichkeit die Freiheitsgrade bei der Erstellung von Computermodellen theoretisch begründet einzuschränken ist es, Modelle im Rahmen einer **kognitiven Architektur** zu erstellen (VanLehn, 1991). Eine kognitive Architektur – auch *unified theory of cognition* genannt – meint die explizite Vorgabe von elementaren Mechanismen der Informationsverarbeitung, von denen angenommen wird, dass sie über alle möglichen Aufgaben hinweg (z. B. Problemlösen, Sprachverarbeitung, Denken, Mustererkennung) stabil bleiben. Zu diesen elementaren Prozessen gehört die Steuerung der Interaktion mit der Umgebung, die Repräsentation von Gedächtnisinhalten, sowie eine Strategie zur Auswahl von alternativen Regeln. Verschiedene Modelle, die im Rahmen der sel-

ben kognitiven Architektur realisiert werden, unterscheiden sich also nur in ihren spezifischen Annahmen. Die Art ihrer technischen Realisierung ist durch die gegebene Architektur beschränkt. Bekannte Beispiele für kognitive Architekturen sind die Produktionssysteme ACT und Soar (siehe 5b-4). Im Gegensatz zu solchen allgemeinen Architekturen stehen *special purpose* Modelle, wie beispielsweise das System SME zum analogen Problemlösen (siehe 5b-5.1).

5b-2 Problemlösen als Suche im Problemraum

Bei der Computermodellierung von Problemlöseprozessen werden vor allem **geschlossene Probleme** mit **Interpolationsbarriere** (siehe Knoblich, Kap. 5a) betrachtet, da sich diese Probleme relativ einfach formalisieren lassen. Ein Problem wird dabei als Beschreibung einer aktuellen Situation, **Problemzustand** genannt, repräsentiert. Für ein leicht modifiziertes “Affe-Banane” Problem (siehe Knoblich, Kap. 5a) könnte eine Situation zum Beispiel darin bestehen, dass der Affe sich in einem Raum befindet, in dem für ihn vom Boden aus unerreichbar eine Banane an der Decke befestigt ist (siehe Abb. 5b-2). Außerdem befindet sich in dem Raum eine Kiste. Eine **Problemlösung** besteht dann aus einer Folge von Aktionen, Operatoranwendungen genannt, mit der der Anfangszustand in einen Zielzustand überführt werden kann. Probleme, deren Lösung auf diese Art beschrieben werden kann, werden auch **Transformationsprobleme** genannt (Greeno, 1978). Das Problemlöseziel des Affen ist es, die Banane zu erhalten. Der Affe kann das Ziel erreichen, indem er zunächst zur Kiste geht, dann die Kiste zu einer Position unterhalb der Banane schiebt, auf die Kiste steigt und die Banane greift.

Im Allgemeinen können auf einen Problemzustand verschiedene Operatoren angewendet werden. Beispielsweise kann der Affe auf die Kiste steigen, wenn die Kiste rechts und nicht unter der Banane steht, oder er kann sie unter die Banane schieben. Nicht jede Abfolge von Operatoranwendungen führt zu einem Zielzustand. Häufig gibt es nur *eine* “optimale” Lösung, die den Problemlöser mit der geringstmöglichen Anzahl von

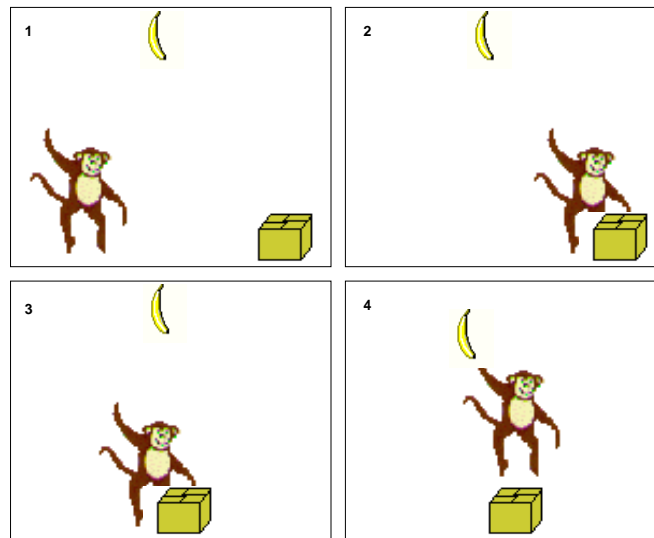


Abb. 5b-2: Ein Affe-Banane Problem: Anfangszustand (1), Zwischenzustand, nachdem der Affe zur Kiste gelaufen ist (2), Zwischenzustand, nachdem der Affe die Kiste unter die Banane geschoben hat (3) und Zielzustand, nachdem der Affe auf die Kiste gestiegen ist und die Banane genommen hat (4)

Operatoranwendungen zum Ziel führt. Die Strategie, nach der ein Problemlöser in einem gegebenen Problemzustand einen Operator auswählt, den er anwenden will, bedingt, ob er das Problem lösen wird, ob er es optimal lösen wird und wie schnell er eine Lösung findet. Solche Strategien werden durch Suchverfahren beschrieben. Systeme zur Transformation von Zuständen durch Operatoranwendungen, bei denen die Auswahl von Operatoren auf einer festgelegten Strategie basiert, heißen **Produktionssysteme**.

Im folgenden werden die Grundbegriffe für die Modellierung von Problemen und Problemlösungen eingeführt. In den nachfolgenden Abschnitten werden dann verschiedene **Suchstrategien** und das Problemlösen mit Produktionssystemen dargestellt.

Ein **Problem** ist gegeben durch einen **Anfangszustand**, einen gewünschten **Zielzustand** und eine Menge von **Problemlöseoperatoren** (Nilsson, 1971). Damit ein Problem maschinell lösbar ist, müssen Zustände und Operatoren formal beschrieben werden. Ein Beispiel für die formale Repräsentation des Affe-Banane Problems aus Abb. 5b-2 ist in Tab. 5b-1 gegeben.

5b-2.1 Problemzustände

Häufig (vor allem in der KI-Planung, siehe Textbox *Problemlösen und Planen in der KI*) werden Zustände durch Propositionen (siehe Hemforth & Konieczny, Kap. 4b) beschrieben, speziell durch konjunktiv (mit “und”) verknüpfte Prädikate (Russell & Norvig, 1995). Zur Beschreibung des Affe-Banane Problems können das zweistellige Prädikat $pos(x, y)$, sowie die nullstelligen Prädikate $auf-boden$ beziehungsweise $auf-kiste$ und $hat-banane$ verwendet werden. Der Anfangszustand in Tab. 5b-1 wird repräsentiert durch die Aussage “Die Position des Affen ist links *und* der Affe steht auf dem Boden *und* die Position der Kiste ist rechts *und* die Position der Banane ist in der Mitte”. Jedes Element der Aussage, beispielsweise $pos(Affe, Links)$, entspricht einem Fakt, der im aktuellen Zustand gültig ist. Die konjunktive Verknüpfung wird dabei abgekürzt mit Kommata notiert.

Repräsentiert werden nur die Aspekte eines Zustands, die relevant für die Problemlösung sind. Beispielsweise ist es egal, welche Farbe die Kiste hat. Im Prinzip kann die Kiste an jeder möglichen Stelle im Raum stehen, die Banane kann an jeder möglichen Stelle an der Decke befestigt sein und der Affe kann zu jeder Position im Raum gehen. Um das Problem formal zu repräsentieren, wer-

Tab. 5b-1: Formale Darstellung des Affe-Banane Problems

Anfangszustand: pos(Affe,Links),auf-boden, pos(Kiste,Rechts), pos(Banane,Mitte)

Zielzustand: pos(Affe, Mitte), auf-kiste, pos(Kiste, Mitte), hat-banane

Raumpositionen: ort(Links), ort(Mitte), ort(Rechts)

Operatoren:

GeheVonNach(x,y):

Anwendungsbedingung: ort(x), ort(y), pos(Affe, x), auf-boden

Auswirkung: ADD pos(Affe,y); DEL pos(Affe, x)

SchiebeKiste(x,y):

Anwendungsbedingung: ort(x), ort(y), pos(Affe, x), pos(Kiste, x), auf-boden

Auswirkung: ADD pos(Affe, y), pos(Kiste, y); DEL pos(Affe, x), pos(Kiste, x)

SteigeAufKiste(x):

Anwendungsbedingung: ort(x), pos(Affe, x), pos(Kiste, x), auf-boden

Auswirkung: ADD auf-kiste; DEL auf-boden

GreifeBanane(x):

Anwendungsbedingung: ort(x), pos(Affe, x), pos(Banane, x), auf-kiste

Auswirkung: ADD hat-banane

den nur einige relevante Positionen im Raum festgelegt, etwa die Raumpositionen *Links*, *Mitte* und *Rechts* in Tabelle 5b-1. Die Prädikate *ort(Links)*, *ort(Mitte)* und *ort(Rechts)* gelten in jeden Zustand. Solche Prädikate werden statisch genannt.

Im Anfangszustand hat der Affe die Banane nicht. Im Prinzip hätte dafür ein eigenes Prädikat eingeführt werden können. Dies ist unnötig, wenn für das Problemlösen angenommen wird, dass alle nicht explizit als gültig angegebenen Fakten falsch sind. Ist also das Prädikat *hat-banane* nicht in einer Zustandsbeschreibung enthalten, kann davon ausgegangen werden, dass dieses Prädikat nicht gilt, also der Affe die Banane nicht hat. Diese **Abgeschlossenheitsannahme** (*closed world assumption*) wird bei der formalen Repräsentation von Sachverhalten durch logische Ausdrücke häufig zugrundegelegt.

Das **Problemlöseziel** beim Affe-Banane Problem ist, dass der Affe die Banane hat – also ein Zustand, in dessen Beschreibung das Prädikat *hat-banane* enthalten ist. Der in Tab. 5b-1 angegebene Zielzustand wird repräsentiert durch die Aussage: “Die Position des Affen ist in der Mitte und der Affe steht auf der Kiste und die Position der Kiste ist in der Mitte und der Affe hat die Banane.”

5b-2.2 Problemlöse-Operatoren

Problemlöseoperatoren werden im Allgemeinen als bedingte Regeln der Form

WENN (Bedingung) DANN (Aktion)

repräsentiert. Solche Regeln werden auch **Produktionsregeln** oder Produktionen genannt (siehe 5b-4). Beispielsweise kann der Affe nur dann auf die Kiste steigen, wenn er sich am selben Ort wie die Kiste befindet, also etwa wenn Affe und Kiste in der Mitte des Raumes sind. Genau dies drückt die **Anwendungsbedingung** (*precondition*) für den Operator *SteigeAufKiste(x)* in Tabelle 5b-1 aus. Der Operator enthält einen Platzhalter (Variable) *x*. Diese Variable kann mit einem der möglichen Orte *Links*, *Mitte* oder *Rechts* belegt (instantiiert) werden. Gleiche Variablen müssen für den gesamten Operator gleich belegt sein (siehe 5b-4.1). Der Modellierung liegt zudem die vereinfachende Annahme zugrunde, dass verschiedene Variablen (zum Beispiel *x* und *y* im Operator *GeheVonNach(x, y)*) verschieden belegt sein müssen. Sind alle Variablen in einem Operator belegt, so bezeichnet man den Operator als Aktion. Anstelle von Operatoren werden bei der Modellierung von Problemlöseprozessen oft direkt alle möglichen Aktionen angegeben (siehe Textbox

aktueller Zustand	Operatoranwendung	Folgezustand
pos(Affe, Links) auf-boden pos(Kiste, Rechts) pos(Banane, Mitte)	$\xrightarrow{\text{geheVonNach(Links, Rechts)}}$	pos(Affe, Links) auf-boden pos(Kiste, Rechts) pos(Banane, Mitte) pos(Affe, Rechts)
ort(Links) ort(Mitte) ort(Rechts)		ort(Links) ort(Mitte) ort(Rechts)

Abb. 5b-3: Zustandstransformation

Problemlösen und Planen in der KI).

Wenn die Anwendungsbedingung eines Operators im aktuellen Zustand erfüllt ist, dann kann die entsprechende Aktion ausgeführt werden. Im Anfangszustand ist nur die Anwendungsbedingung des Operators $GeheVonNach(x, y)$ erfüllt. Die Ausführung des Operators resultiert in einem neuen Zustand. Wie die Transformation des aktuellen Zustands in einen Folgezustand vonstatten geht, wird durch die **Operator-Auswirkung** (*effect*) beschrieben. Häufig (vor allem in der KI-Planung) werden hierfür sogenannte ADD-DEL Listen verwendet. Diese beschreiben, welche Fakten nach Anwendung des Operators zusätzlich gelten (ADD) und welche Fakten des aktuellen Zustands nicht mehr gelten (DEL). Ein Beispiel für die Transformation des Anfangszustands in einen Folgezustand ist in Abb. 5b-3 gegeben.

5b-2.3 Problemraum

Die Formulierung eines Problems durch Anfangszustand, Zielzustand und Operatoren definiert einen **Problemraum** (Newell & Simon, 1972). Ein Problemraum ist ein Graph mit allen möglichen Zuständen, die das Problem annehmen kann, als Knoten. Zwischen Zuständen, die durch Operatoranwendung ineinander überführbar sind, existiert eine Kante. Der Problemraum für das Affe-Banane Problem ist in Abb. 5b-4 gegeben.

Der Affe kann sich vom Anfangszustand aus von Links in die Mitte oder von Links nach Rechts begeben. Er kann dann von Rechts zur Mitte gehen, oder auch zurück nach Links. Im Prinzip könnte der Affe endlos hin und her laufen, ohne das Problem zu lösen. Wenn ein Zustand von einem anderen Zustand aus wieder erreichbar ist, spricht man von einem **Zyklus**. Der Affe kann aber auch, wenn er nach Rechts gelaufen ist, auf die

Kiste steigen, da dann die Anwendungsbedingung des Operators $SteigeAufKiste(x)$ erfüllt ist. Allerdings haben wir dem Affen keinen Operator mitgegeben, mit dem er wieder von der Kiste heruntersteigen kann. Wenn er an einer Position auf die Kiste steigt, von der aus die Banane nicht erreichbar ist, so ist er in eine **Sackgasse** gelaufen und kann das Problem nicht lösen.

Der Affe kann sein Ziel, die Banane zu haben, durch Anwendung von vier Operatoren erreichen:

GeheVonNach(Links, Rechts),
SchiebeKiste(Rechts, Mitte),
SteigeAufKiste(Mitte),
GreifeBanane(Mitte).

Dies ist die kürzestmögliche, also **optimale** Lösung. Es sind alternative, nicht optimale Lösungen des Problems möglich, beispielsweise:

GeheVonNach(Links, Mitte),
GeheVonNach(Mitte, Rechts),
SchiebeKiste(Rechts, Links),
SchiebeKiste(Links, Mitte),
SteigeAufKiste(Mitte),
GreifeBanane(Mitte).

Allgemein ist eine **Problemlösung** definiert als eine Abfolge von Operatoren, die einen gegebenen Anfangszustand in den gewünschten Zielzustand überführt. Bezogen auf den Problemraum ist eine Problemlösung ein Pfad, also eine Folge von Kanten, vom Anfangszustand zum Zielzustand. **Problemlösen** ist definiert als die Suche nach einem solchen Pfad.

Im Allgemeinen hat ein Problemlöser den vollständigen Problemraum nicht zur Verfügung – weder als externe noch als mentale Repräsentation. Im nächsten Abschnitt wird dargestellt, wie mit Hilfe von Suchstrategien Teile des Problemraums erzeugt und exploriert werden können.

5b-2.4 Das Turm-von-Hanoi Problem

Der Legende nach sind die Mönche von Hanoi seit Jahrhunderten damit beschäftigt, einen Turm aus 64 Scheiben von einer Position auf eine andere zu versetzen. Nach Meinung der Mönche soll

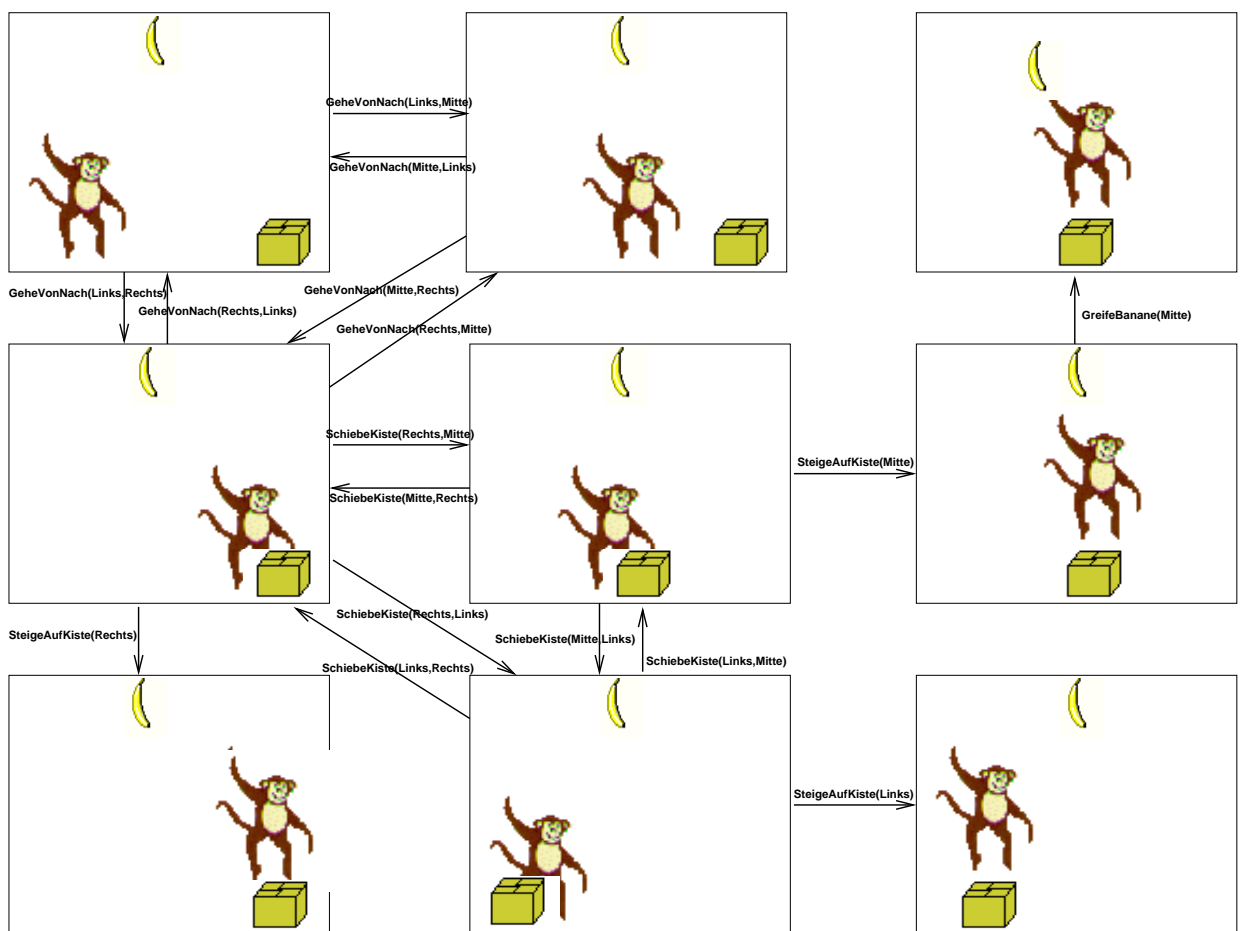


Abb. 5b-4: Problemraum für das Affe-Banane Problem



Abb. 5b-5: Turm-von-Hanoi Problem mit drei Scheiben

die Welt untergehen, wenn der Turm komplett versetzt ist. Alle Scheiben des Turmes haben unterschiedliche Größen. Es gibt drei Positionen (Stifte) – einen Ausgangsstift (1), auf dem der Turm ursprünglich steht, einen Zielstift (3), auf dem der Turm am Ende stehen soll, und einen weiteren Stift (2), auf dem Scheiben zwischengelagert werden können. Eine Veranschaulichung für das Drei-Scheiben Problem ist in Abb. 5b-5 gegeben. Scheiben dürfen nur nach folgender Regel bewegt werden: Es kann nur die oberste Scheibe auf einem Stift versetzt werden und eine Scheibe darf nur auf einen Stift gesetzt werden, auf dem keine Scheibe liegt oder dessen oben liegende Scheibe größer als die zu versetzende Scheibe ist.

Der Problemraum ist in Abb. 5b-6 dargestellt. Bereits für drei Scheiben kann das Problem 27 mögliche Zustände annehmen, und es werden mindestens sieben Operatoranwendungen benötigt, um den Turm vom Start- auf den Zielstab zu setzen. Die Anzahl möglicher Zustände und die minimale Anzahl von Operatoranwendungen wachsen **exponentiell**. Allgemein kann das Wachstumsverhalten folgendermaßen bestimmt werden:

Zustände: 3^n bei 3 Stiften und n Scheiben

Operatoranwendungen: $2^n - 1$.

Die Zustandszahl 3^n ergibt sich durch das wahrscheinlichkeitstheoretische Gesetz der Variation. Die Ermittlung der minimal notwendigen Zahl der Züge lässt sich folgendermaßen veranschaulichen: Für das Ein-Scheiben Problem muss die Scheibe genau einmal – von Stift (1) nach Stift (3) – bewegt werden. Für zwei Scheiben wird zunächst die kleinere auf Stift (2) gelegt, dann die größere auf Stift (3) und schließlich die kleinere auf Stift (3). Die größere Scheibe wird also einmal, die kleinere zweimal bewegt. Bei drei Scheiben muss Schei-

be A viermal, Scheibe B zweimal und Scheibe C einmal bewegt werden. Jede Scheibe muss 2^i mal bewegt werden: die größte 2^0 (also ein-) mal, die zweitgrößte 2^1 (also zwei-) mal, die drittgrößte 2^2 (also vier-) mal, die viertgrößte 2^3 mal und so fort. Insgesamt müssen $2^0 + 2^1 + 2^2 + 2^3 + \dots + 2^{n-1}$ Bewegungen für n Scheiben ausgeführt werden (Schmid & Kinds Müller, 1996).

Die Welt wird also, selbst wenn die Legende stimmt, noch lange nicht untergehen. Wenn die Mönche alle zehn Sekunden eine Scheibe versetzen, brauchen sie zur Lösung des 64-Scheiben Problems immerhin fünf *Trillionen* Jahre (Harel, 1987). Das exponentielle Wachstumsverhalten ist in Abb. 5b-7 dargestellt. Viele Probleme, die in der KI bearbeitet werden, haben ein solches exponentielles Wachstumsverhalten. Man spricht hier auch von nicht effizient berechenbaren Problemen.

Betrachtet man das Turm-von-Hanoi Problem analytisch, so kann die optimale Abfolge von Aktionen nach folgender **rekursiven Vorschrift** (siehe Textbox *Rekursive Probleme*) ermittelt werden:

```

hanoi(n, von, via, nach) =
WENN n = 0
DANN tue nichts
SONST
    hanoi(n-1, von, nach, via)
    bewegeScheibe(von, nach)
    hanoi(n-1, via, von, nach).

```

Die Scheibenzahl n korrespondiert mit der Größe der Scheibe. Für das Drei-Scheiben Problem entspricht $n = 3$ der Scheibe C, $n = 2$ der Scheibe B und $n = 1$ der Scheibe A. Die Stifte sind der Anfangsstift $von = 1$, der Hilfsstift $via = 2$ und der Zielstift $nach = 3$. Die Funktion *hanoi* regelt, wie oft und von welchem Stift auf welchen eine Scheibe versetzt werden soll. Für $n = 3$ wird die Aktion *bewegeScheibe* einmal, nämlich vom Anfangsstift ($von = 1$) zum Zielstift ($nach = 3$) ausgeführt. Gleichzeitig ruft sich *hanoi* für die nächste Scheibe zweimal selbst auf. Scheibe $n - 1$, also B, muss einmal vom Anfangsstift zum Hilfsstift transportiert werden und – nachdem Scheibe C bewegt wurde – vom Hilfsstift auf den Zielstift. Die Abarbeitung der rekursiven Funktion ist in Abb. 5b-8 veranschaulicht.

Die rekursive Lösungsstruktur des Turm-von-Hanoi Problems spiegelt sich auch im Problem-

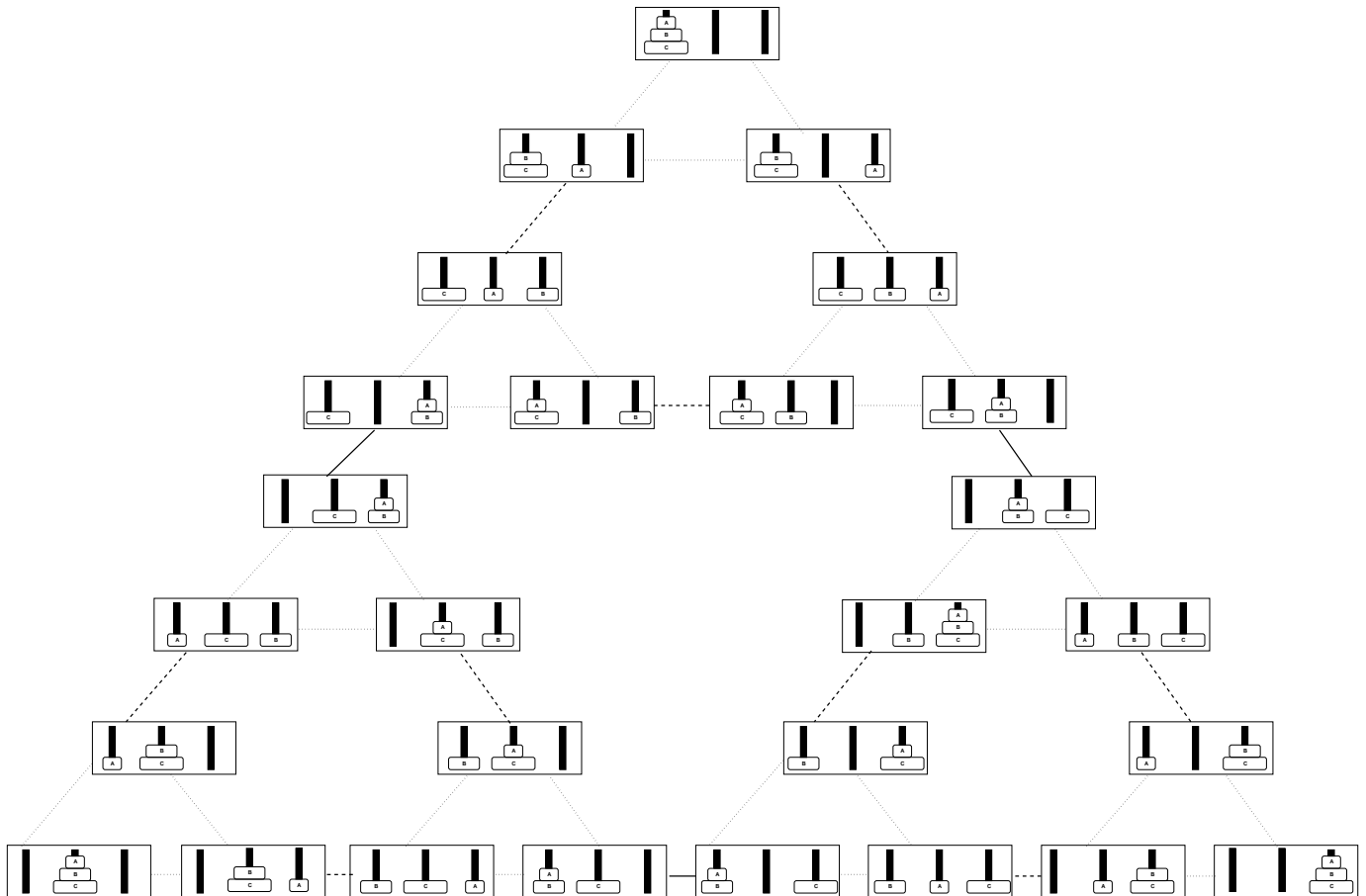


Abb. 5b-6: Problemraum des Turm-von-Hanoi Problems mit drei Scheiben (Bewegungen von A gepunktet, Bewegungen von B gestrichelt, Bewegungen von C durchgezogen)

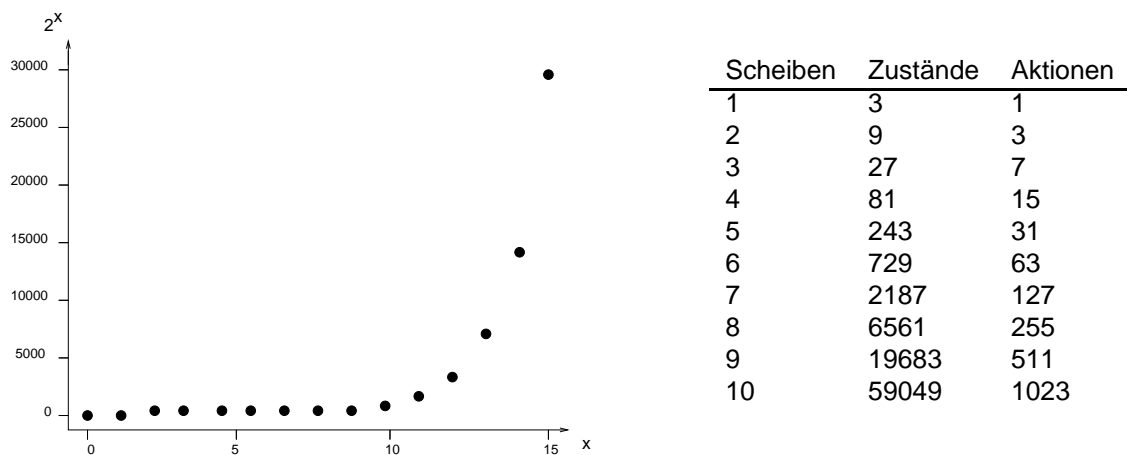
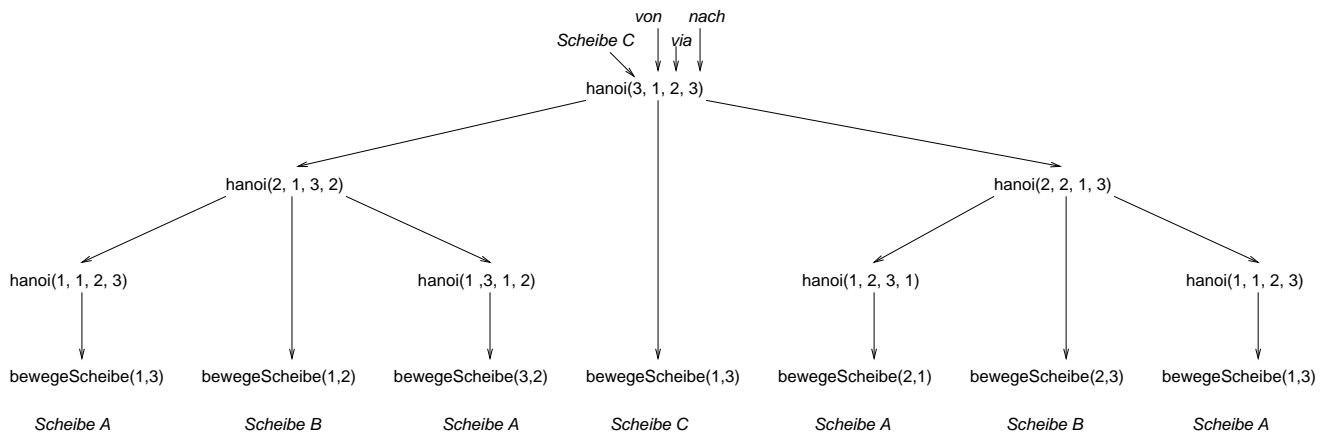


Abb. 5b-7: Exponentielles Wachstum beim Turm-von-Hanoi

Abb. 5b-8: Abarbeitung der rekursiven Funktion *hanoi* für drei Scheiben

raum wider: Der Problemraum des Ein-Scheiben Problems ist achtmal enthalten (alle kleinen Dreiecke); der Problemraum des Zwei-Scheiben Problems ist dreimal enthalten (das obere, linke und rechte Dreieck). Die möglichen Bewegungen der größten Scheibe *C* verbinden die drei Zweier-Problemräume, ebenso verbinden die möglichen Bewegungen der Scheibe *B* die Einer-Problemräume.

Ist die Lösungsvorschrift für ein Problem bekannt, so spricht man nicht mehr von Problemlösen. Ein Problemlöser muss die korrekte Abfolge von Operatoren zur Überführung eines Anfangszustands in einen Zielzustand ermitteln.

5b-3 Suchstrategien

Auf einen gegebenen Zustand sind im Allgemeinen mehrere Operatoren anwendbar. Der Problemlöser muss sich also entscheiden, welche Aktion er ausführen soll. Diese Auswahl wird durch eine Suchstrategie gesteuert.

5b-3.1 Uninformierte Suche

Die einfachste Suchstrategie ist die zufällige Auswahl von Operatoren, also eine **Versuch-und-Irrtum** Strategie (*trial and error*). Damit bewegt man sich jedoch sehr unsystematisch im Problemraum fort. Eine sogenannte "blinde" oder uninformierte Suchstrategie ist die **Tiefensuche** (Winston, 1992). Basierend auf einer Ordnung auf den

Operatoren wird in jedem Zustand der erste anwendbare Operator ausgewählt. Führt die Suche in eine Sackgasse, wird die letzte Operatoranwendung zurückgenommen (*Backtracking*) und der nächstmögliche Operator ausprobiert. Damit die Suche nicht in unendliche Pfade führt, wird zusätzlich ein Mechanismus zur Erkennung und Vermeidung von Zyklen benötigt. Welche Teile des Problemraums bei der Suche generiert und exploriert werden, kann durch einen **Suchbaum** dargestellt werden. Ein Beispiel für einen durch Tiefensuche erzeugten Suchbaum für das Affe-Banane Problem (siehe Abb. 5b-4) ist in Abb. 5b-9 gegeben.

Die Ordnung der Operatoren soll die Präferenzen des Affen widerspiegeln: Am liebsten mag er die Banane greifen, am zweitliebsten auf die Kiste steigen, dann die Kiste schieben und am uninteressantesten ist das Herumlaufen. Bevorzugt hält sich der Affe in der Mitte des Raumes, also nahe bei den Bananen auf, am zweitliebsten Rechts, und am wenigsten gerne Links. Egal, welche Ordnung vorgegeben wird, wird mit der Tiefensuche eine Lösung gefunden, wenn das Problem lösbar ist. Allerdings bedingt die Anordnung der Operatoren, (1) wie viele Zustände des Problemraums exploriert werden müssen, und (2) wie viele Operatoranwendungen die gefundene Lösung enthält. In dem Suchbaum in Abb. 5b-9 landet der Affe zunächst in einer Sackgasse, weil er zunächst schon rechts auf die Kiste steigt. Diese Aktion wird dann zurückgenommen und der nächstmögliche Operator, also die Kiste zu schieben, auspro-

Rekursive Probleme. Rekursion meint Selbstbezüglichkeit, also die Definition eines Problems unter Bezug auf sich selbst. Die wohl bekannteste rekursive Definition einer mathematischen Funktion ist die der Fakultät:

$$faku(x) = \begin{cases} 1 & \text{wenn } x = 0 \\ x \cdot faku(x - 1) & \text{sonst.} \end{cases}$$

Die Fakultät einer natürlichen Zahl x berechnet sich durch Multiplikation von x mit der Fakultät der Zahl $x - 1$. Beispielsweise gilt:

$$faku(0) = 1$$

$$faku(1) = 1 \cdot faku(0) = 1 \cdot 1 = 1$$

$$faku(2) = 2 \cdot faku(1) = 2 \cdot 1 \cdot 1 = 2$$

$$faku(3) = 3 \cdot faku(2) = 3 \cdot 2 \cdot 1 \cdot 1 = 6$$

$$faku(4) = 4 \cdot faku(3) = 4 \cdot 3 \cdot 2 \cdot 1 \cdot 1 = 24.$$

Um also die Fakultät der Zahl 4 auszurechnen, muss man auf das Ergebnis $faku(3)$ zurückgreifen. Ist dieses Ergebnis nicht bekannt, so muss nun zunächst $faku(3)$ berechnet werden. Dafür wird aber die Kenntnis von $faku(2)$ benötigt, hierfür $faku(1)$ und hierfür $faku(0)$. Für $faku(0)$ ist das Ergebnis direkt definiert. Das Ergebnis 1 kann an die wartende Funktion $faku(1)$ zurückgegeben werden, die nun berechnet werden kann und ihr Ergebnis an $faku(2)$ weitergibt. Diese gibt ihr Ergebnis an $faku(3)$ weiter und schließlich kann $faku(4)$ fertig berechnet werden.

Bei rekursiven Definitionen ist es wichtig, dass es einen Fall gibt, für den das Ergebnis direkt ermittelt werden kann. Ansonsten würde man bei der Berechnung endlos “in die Tiefe” steigen und nie zu einem Abschluss kommen. Man spricht dann von Nicht-Termination einer Funktion.

Vielen Problemen liegt eine rekursive Lösungsstruktur zugrunde. Das bekannteste dieser Probleme ist sicher das Turm-von-Hanoi Problem. Generell gilt, wenn ein Problem nicht durch eine rekursive Berechnungsvorschrift lösbar ist, dann ist es nicht algorithmisch, also durch ein Computerprogramm, lösbar. Eine ausführliche Beschreibung verschiedener Formen rekursiver Funktionen und eine Diskussion von Rekursion als Problemlösewerkzeug findet sich in Vorberg und Göbel (1991).

Ordnung auf den Operatoren:

(siehe Tab. 5b-1)

- (1) GreifeBanane,
- (2) SteigeAufKiste,
- (3) SchiebeKiste,
- (4) GeheVonNach

Bevorzugung von Orten:

- (1) Mitte,
- (2) Rechts,
- (3) Links

Die statischen Prädikate $ort(Links)$, $ort(Mitte)$ und $ort(Rechts)$ werden bei den Zustandsbeschreibungen der Übersichtlichkeit halber weggelassen.

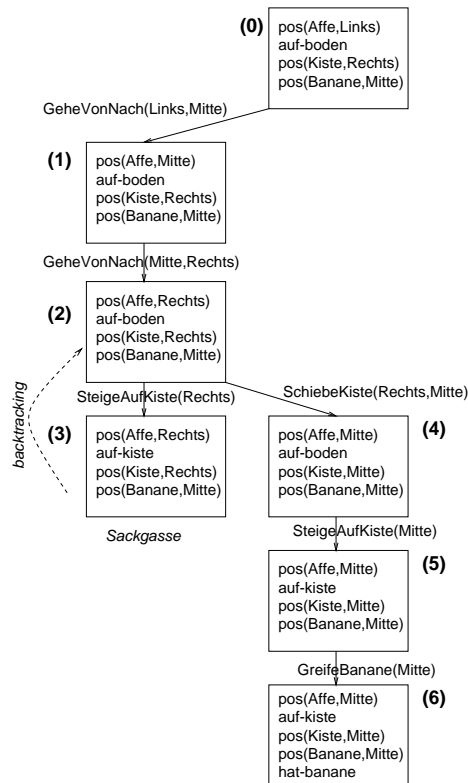


Abb. 5b-9: Durch Tiefensuche erzeugter Suchbaum für das Affe-Banane Problem

biert. Dieser Pfad führt schließlich zum Ziel. Allerdings enthält die Lösung einen unnötigen Schritt, weil der Affe nicht direkt nach rechts, sondern zunächst in die Mitte läuft, die mit der Tiefensuch-Strategie gefundene Lösung ist also nicht optimal.

In einen Zyklus wäre der Affe beispielsweise gelaufen, wenn er in Zustand 3 in Abb. 5b-9 wieder in die Mitte oder nach Links gegangen wäre. Damit wäre ein Zustand erzeugt worden, der schon einmal weiter oben auf dem Pfad erreicht worden ist. Dies kann durch einfache Prüfung auf Identität des aktuellen Zustands mit den bereits auf dem Pfad vorhandenen Zuständen erkannt werden und die entsprechende Aktion kann durch *backtracking* verworfen werden.

Eine alternative Strategie zur Tiefensuche ist die **Breitensuche** (Winston, 1992). Hier wird nicht zuerst ein Pfad in die Tiefe verfolgt, sondern der Baum wird ebenenweise aufgebaut. Auf jeden Zustand werden alle möglichen Aktionen angewendet, es wird also nicht nur ein sondern alle zulässigen Folgezustände erzeugt. Durch den ebenweisen Aufbau des Suchbaums wird diejeni-

ge Lösung gefunden, die mit der geringsten Anzahl von Operator-Anwendungen zum Ziel führt. Im Allgemeinen wird bei der Breitensuche jedoch ein größerer Teil des Problemraums generiert als bei der Tiefensuche, die Suche ist also mit mehr Aufwand verbunden.

5b-3.2 Hill Climbing und Bewertungsfunktionen

Bei den blinden Suchstrategien werden ausgehend vom Anfangszustand Folgezustände generiert, solange bis ein erreichter Zustand das Problemlöseziel erfüllt. Die Suche ist also nicht auf das Problemlöseziel hin ausgerichtet. Alternativ zur Operatorauswahl nach einer beliebigen Ordnung kann in jedem Zustand derjenige Operator angewendet werden, der den Unterschied des aktuellen Zustands zum Zielzustand am meisten verringert (Unterschiedsreduktion). Die Suche wird damit durch eine **heuristische Bewertung** gesteuert. Suchverfahren, bei denen die Operatorauswahl aufgrund heuristischer Bewertungen erfolgt, wer-

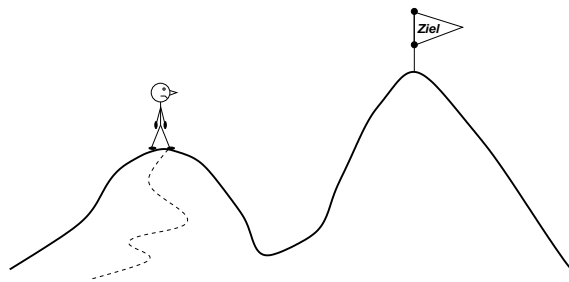


Abb. 5b-10: Das Problem lokaler Maxima beim *Hill Climbing*

den als **heuristische Suchstrategien** bezeichnet.

Eine auf Unterschiedsreduktion basierende Strategie ist das *Hill Climbing* (**„Bergsteigen“**). Wenn man sich das Ziel als einen Berggipfel vorstellt, wählt man den nächsten Schritt immer so, dass man möglichst viel an Höhe gewinnt, da dadurch die Distanz zum Ziel am meisten verringert wird. Beispielsweise verringert beim Affe-Banane Problem (siehe Abb. 5b-4) das Hin-und-Her-Laufen im Raum den Unterschied zum Zielzustand nicht, das Schieben der Kiste in die Mitte unter die Bananen dagegen schon. Allerdings basiert die Entscheidung beim *Hill Climbing* immer nur auf dem aktuellen Zustand, ist also lokal. Durch die Vernachlässigung der globalen Struktur des Problems kann es passieren, dass man sich *„verrennt“* (siehe Abb. 5b-10): Selbst wenn jede Operatoranwendung den Problemlöser näher ans Ziel bringt, kann es sein, dass er in einem Zustand landet, von dem aus er sich erst wieder vom Ziel entfernen (also *„absteigen“*) müsste, um tatsächlich ans Ziel zu gelangen. *Hill Climbing* kann also zu lokalen Maxima führen.

Es gibt empirische Hinweise, dass menschliche Problemlöser eine Strategie der Unterschiedsreduktion anwenden, beispielsweise bei der Lösung von sogenannten *„Fluss-Überquerungs“*-Problemen, wie dem *„Hobbits and Orcs“* Problem (Greeno, 1974). Bei diesem Problem, das manchmal auch als *„Missionare und Kannibalen“* Problem bezeichnet wird, geht es darum, dass eine Gruppe von *Hobbits* und *Orcs* mit einem Boot einen Fluss überqueren wollen, wobei die *Orcs* nie in Überzahl sein dürfen (genaue Beschreibung siehe Abb. 5b-11). Greeno (1974) konnte zeigen,

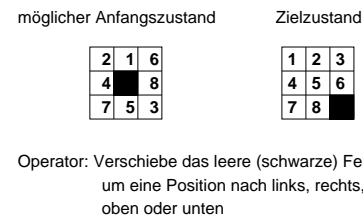


Abb. 5b-12: Das Achterpuzzle

dass Probanden vor allem Schwierigkeiten mit dem Übergang von Zustand (6) zu Zustand (7) in Abb. 5b-11 haben – hier rudert nicht nur ein Passagier zurück zum Ausgangsufer, sondern es müssen zwei Passagiere mit zurückgenommen werden. Es muss also ein Zustand hergestellt werden, der sich stärker vom Ziel unterscheidet, als ein bereits vorliegender Zustand. Eine Modellierung der Lösung von Überquerungsproblemen mit einem Produktionssystem (siehe 5b-4) schlagen Schmalhofer und Polson (1986) vor.

Damit eine **Heuristik** von einem computerimplementierten Suchprogramm zur Steuerung der Suche verwendet werden kann, muss die Heuristik formal definiert werden. Eine sehr einfache Bewertungsfunktion wäre, jeweils die Anzahl der im aktuellen Zustand schon erfüllten Komponenten des Problemlöseziels zu ermitteln. Beim *Hobbits-and-Orcs* Problem wären das also die Anzahl der Passagiere am rechten Flussufer. Beim in Abb. 5b-12 dargestellten Achter-Puzzle (Nilsson, 1971) wären es die Anzahl von Plättchen, die bereits korrekt platziert sind. Je mehr Wissen über die Struktur eines Problems verfügbar ist, um so differenzierter kann die Bewertung von Zuständen erfolgen und um so wahrscheinlicher wird es, dass der bei der Suche eingeschlagene Pfad zum Erfolg führt. Beispielsweise ist bei der Lösung des Achter-Puzzles wichtiger, dass die Plättchen in die korrekte Reihenfolge gebracht werden als dass einzelne Plättchen schnellstmöglich an ihre Zielposition befördert werden (Nilsson, 1971). Der Schachcomputer *Deep Blue* (siehe Textbox *Deep Blue versus Kasparov*) arbeitet mit einer von Schachexperten entwickelten sehr ausgefeilten Bewertungsfunktion für Spielpositionen.

Anfangszustand: Drei Hobbits (H) und drei Orcs (O) befinden sich am linken Ufer eines Flusses.

Zielzustand: Alle sechs befinden sich am rechten Ufer des Flusses.

Operator: Mit einem Boot (b) können mindestens ein und maximal zwei Passagiere gleichzeitig von einem Ufer zum anderen transportiert werden. **Anwendungsbedingung:** An keinem Ufer dürfen mehr Orcs als Hobbits sein (da die Orcs sonst die Hobbits auffressen).

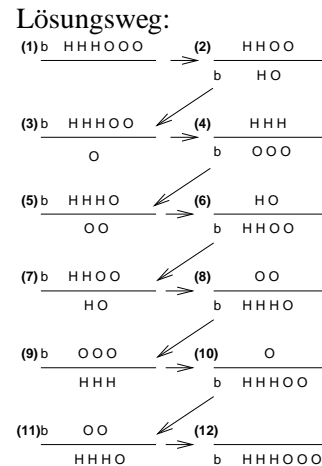


Abb. 5b-11: Das “Hobbits and Orcs” Problem

5b-3.3 Mittel-Ziel Analyse

Eine spezielle Methode der Unterschiedsreduktion ist die **Mittel-Ziel Analyse**. Diese Suchstrategie wurde in den sechziger Jahren von Newell und Simon im Rahmen des *General Problem Solvers* (GPS) entwickelt (Newell & Simon, 1972), einem Programm zur Simulation menschlicher Problemlöseprozesse.

Die Idee der Mittel-Ziel Analyse ist, basierend auf einem Vergleich vom aktuellen Zustand mit den Problemlösezielen, den Operator – also das Mittel – auszuwählen, der den Unterschied zu den Zielen am stärksten reduziert. Die Mittel-Ziel Analyse angewendet auf das Affe-Banane Problem (siehe Abb. 5b-4) würde wie in Tab. 5b-2 dargestellt arbeiten.

Ausgehend vom globalen Problemlöseziel werden so lange **Teilziele** (Unterziele) erzeugt, bis eines der Teilziele direkt durch Anwendung eines Operators zu erreichen ist. Dieses Teilziel ist dann abgearbeitet und kann vergessen werden. So werden allmählich alle erzeugten Teilziele erfüllt, bis schließlich das globale Ziel erreicht werden kann. Das Verarbeitungsprinzip, nach dem die Teilziele abgearbeitet werden, ist “was als Letztes erzeugt wurde, wird als Erstes betrachtet” (*last in first out*). Die entsprechende Speicherstruktur heißt **Stack** (Stapel, “**Kellerspeicher**”) und ist in Abb. 5b-13 veranschaulicht. Zunächst wird das globale Ziel *hat-banane* auf den *Stack* gepackt. Da das Ziel nicht direkt erreichbar ist, werden die Teilziele *auf-*

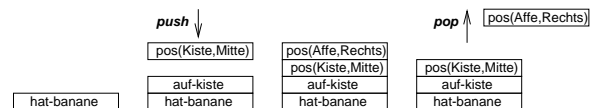


Abb. 5b-13: Verarbeiten von Teilzielen mit einem *Stack*

kiste und *pos(Kiste, Mitte)* auf den *Stack* gepackt. Die Operation, mit der etwas auf einen *Stack* gelegt wird, heißt *push*. Danach wird das Teilziel *pos(Affe, Rechts)* – also dafür zu sorgen, dass sich Affe und Kiste am selben Ort befinden – auf den *Stack* gepackt. Betrachtet werden kann immer nur das oberste Element des *Stacks*. Das nun oben liegende Teilziel kann durch die Aktion *GeheVon-Nach(Links,Rechts)* erfüllt werden. Damit kann das Teilziel vom *Stack* entfernt werden. Die entsprechende Operation heißt *pop*.

Der Algorithmus zur Realisierung der Mittel-Ziel Analyse ist in Tabelle 5b-3 angegeben. Der Algorithmus besteht aus drei Regeln – *Transformiere*, *Reduziere* und *Wende-An* – die sich wechselseitig aufrufen. Die Suche wird durch Aufruf der Regel *Transformiere* mit dem Anfangszustand und dem globalen Ziel gestartet. Das Ziel kann dabei aus einem einzigen Fakt bestehen – wie *hat-banane* beim Affe-Banane Problem – oder aus mehreren Fakten, beispielsweise, dass alle *Hobbits* und *Orcs* auf der rechten Uferseite sind (siehe Abb. 5b-11). Erfüllt der aktuelle Zustand das Ziel, wird die Suche erfolgreich beendet, anderenfalls

Tab. 5b-2: Lösung des Affe-Banane Problems mit Mittel-Ziel Analyse

Mein Ziel ist, die Banane zu haben.

Um die Banane zu haben, muss ich sie greifen (den Operator *greifeBanane* anwenden).

Um die Banane zu greifen, muss ich

(a) auf der Kiste stehen (den Operator *SteigeAufKiste* anwenden) und

(b) die Kiste muss unter der Banane stehen.

Also ist mein neues Teilziel, die Kiste unter die Banane zu schieben (den Operator *SchiebeKiste* anwenden).

Um die Kiste unter die Banane zu schieben, muss ich am selben Ort sein wie die Kiste.

Also ist mein neues Teilziel, zur Kiste zu gehen.

Diese Aktion kann ich ausführen. Ich kann also jetzt zur Kiste gehen.

Das Teilziel, zur Kiste zu gehen, ist erreicht.

Jetzt kann ich die Kiste unter die Banane schieben.

Das Teilziel, die Kiste unter die Banane zu schieben, ist erreicht.

Jetzt kann ich auf die Kiste steigen.

Das Teilziel, auf der Kiste zu stehen, ist erreicht.

Jetzt kann ich die Banane greifen.

Mein Ziel ist erreicht.

wird die Regel *Reduziere* aufgerufen. Hier wird nach einem Operator gesucht, der den Unterschied zwischen aktuellem Zustand und Ziel verringert. Existiert kein solcher Operator, wird die Suche erfolglos abgebrochen, anderenfalls wird der Regel *Wende-An* aufgerufen. Wenn der Operator auf den aktuellen Zustand anwendbar ist, wenn also die Anwendungsbedingungen des Operators im aktuellen Zustand erfüllt sind, wird der entsprechende Folgezustand generiert und erneut die Regel *Transformiere* aufgerufen – in der nun der neue Folgezustand mit dem Ziel verglichen wird. Ist der Operator nicht auf den aktuellen Zustand anwendbar, so wird als neues Teilziel gesetzt, den Unterschied zwischen den Anwendungsbedingungen des Operators und dem aktuellen Zustand zu reduzieren, es wird also wieder die Regel *Reduziere* aufgerufen.

Das wechselseitige Aufrufen der Regeln wird als verschränkte Rekursion (siehe Textbox *Rekursive Probleme*) bezeichnet. Ruft beispielsweise *Reduziere* die Regel *Wende-An* auf, so bleibt diese Regel so lange “aktiv”, bis tatsächlich ein Operator angewendet wird und der entsprechende Unterschied entfernt wurde. Durch die rekur-

Tab. 5b-3: Mittel-Ziel Analyse

Transformiere: Vergleiche den aktuellen Zustand mit dem Ziel

WENN der Zustand das Ziel erfüllt

DANN halte an und melde Erfolg

SONST **Reduziere** den Unterschied zwischen Zustand und Ziel.

Reduziere: Finde Operator, um den Unterschied zwischen Zustand und Ziel zu verringern

WENN kein solcher Operator verfügbar ist

DANN halte an und melde Fehler

SONST **Wende** den gefundenen Operator auf den aktuellen Zustand **an**.

Wende-An: Wende einen Operator auf den aktuellen Zustand an

WENN der Operator auf den aktuellen Zustand anwendbar ist

DANN wende den Operator an und **Transformiere** den dabei entstandenen Folgezustand in das Ziel.

SONST **Reduziere** den Unterschied zwischen dem Zustand und den Anwendungsbedingungen des Operators.

sive Struktur der Regeln wird der Aufbau und die Abarbeitung des *Stack* organisiert (siehe Abb. 5b-13 und Tab. 5b-2): Wenn in *Reduziere* festgestellt wird, dass das Teilziel *auf-kiste* durch Anwendung des Operators *SteigeAufKiste(Mitte)* erfüllt werden kann, so gelangt dieses Teilziel auf den *Stack*. Der Operator kann nicht unmittelbar auf die aktuelle Situation angewendet werden, also ruft *Wende-An* wiederum *Reduziere* auf, das nun wiederum das Teilziel, dass die Kiste erst einmal in die Mitte geschoben werden muss (*pos(Kiste, Mitte)*), erzeugt, es oben auf den *Stack* packt und *Wende-An* für den Operator *SchiebeKiste(Rechts, Mitte)* aufruft. Erst nach erfolgreicher Erledigung der Teilziele *pos(Affe, Rechts)* und *pos(Kiste, Mitte)* durch Anwendung der entsprechenden Operatoren kann die Aktion *SteigeAufKiste(Mitte)* ausgeführt werden. Dann erst wird der entsprechende Aufruf von *Reduziere* endgültig verlassen und das Teilziel *auf-kiste* vom *Stack* entfernt.

Die Arbeitsweise der Mittel-Ziel Analyse für das Turm-von-Hanoi Problem (siehe 5b-2.4) ist in Tab. 5b-4 dargestellt.

Ein Problem der Mittel-Ziel Analyse ist, dass Probleme mit voneinander abhängigen Teilzielen nicht mit dieser Strategie lösbar sind (siehe Textbox *Problemlösen und Planen in der KI*). Allerdings ist das Finden einer Problemlösung oder gar einer optimalen Problemlösung mit einer vollständigen Strategie – also einer Strategie, die auch mit abhängigen Teilzielen umgehen kann – oft sehr aufwendig. Viele Probleme bestehen aus einer sehr großen Anzahl von Zuständen (siehe 5b-2.4). Entsprechend groß können die Suchbäume werden, die auf der Suche nach einer Problemlösung erzeugt werden. Die Annahme der Unabhängigkeit von Teilzielen kann zwar dazu führen, dass manche Probleme nicht gelöst werden können, dafür ist sie aber effizient, weil immer nur ein Ziel im Auge behalten werden muss.

5b-4 Produktionssysteme

Ein System, das die Anwendung von (bedingten) Regeln, **Produktionsregeln** oder Produktionen genannt, auf Daten steuert, heißt Produktionssystem (siehe Abb. 5b-14). Die aktuell im Arbeitsspeicher befindlichen Daten können bei-

Tab. 5b-4: Lösung des Turm-von-Hanoi Problems mit Mittel-Ziel Analyse

Transformiere: Anfangszustand (Scheiben A, B, C auf Stift 1) nach Ziel (Scheiben A, B, C auf Stift 3)
Reduziere: C ist nicht auf 3
Wende-An: Bringe C nach 3
Reduziere: C ist nicht frei, weil B darauf liegt
Wende-An: Entferne B von C
Reduziere: B ist nicht frei, weil A darauf liegt
Wende-An: Entferne A von B;
A kann auf 3 gelegt werden
B ist nun frei
B kann auf 2 gelegt werden
C ist nun frei
Reduziere: C kann nicht auf 3 gebracht werden, weil A darauf liegt
Wende-An: Entferne A von 3;
A kann auf 2 gelegt werden
C kann auf 3 gelegt werden
Transformiere: Zustand (A und B auf 2, B auf 3) nach Ziel
Reduziere: B ist nicht auf 3
Wende-An: Bringe B nach 3
Reduziere: B ist nicht frei, weil A darauf liegt
Wende-An: Entferne A von B;
A kann auf 1 gelegt werden
B ist nun frei
B kann auf 3 gelegt werden
Transformiere: Zustand (A ist 1, B und C auf 3) nach Ziel
Reduziere: A ist nicht auf 3
Wende-An: Bringe A nach 3;
A kann auf 3 gelegt werden
A ist auf 3
Transformiere: Zustand (A, B und C auf 3) nach Ziel
Ziel erreicht

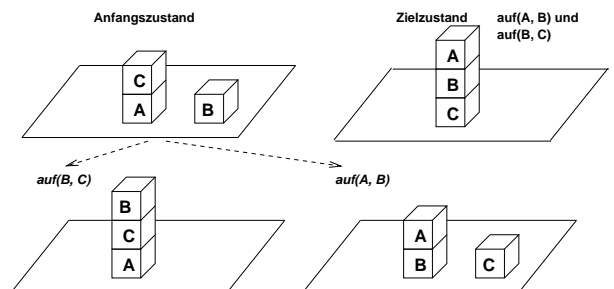
spielsweise die Beschreibung eines Problemzustands sein und die gespeicherten Regeln Problemlöseoperatoren der Form “Wenn (Bedingung) Dann (Aktion)” (siehe 5b-2.2). Die Steuerung der Regelanwendung erfolgt durch einen **Interpreter**. Zunächst werden Ausgangsdaten über eine Eingabe-Schnittstelle in den Arbeitsspeicher gebracht. Danach werden solange Regeln auf die Daten im Arbeitsspeicher angewendet, bis entweder keine Regel mehr anwendbar ist oder eine Regel zur Anwendung kommt, deren Aktion ein Kommando zum Stoppen der Abarbeitung ist. Danach werden die aktuellen Daten als Ergebnis ausgegeben.

Die Transformation der Daten durch Anwen-

Problemlösen und Planen in der KI. In der KI wird zwischen Problemlösen und Planen unterschieden (Russell & Norvig, 1995). Planen kann dabei als Oberbegriff gesehen werden. Problemlösealgorithmen basieren auf speziellen, eingeschränkten Repräsentationen für Zustände und Operatoren und können auf zusätzliches Wissen, zum Beispiel auf eine speziell auf ein Problem hin zugeschnittene Bewertungsfunktion, zurückgreifen. Der bekannteste Problemlösealgorithmus in der KI ist A^* (Nilsson, 1971). Dieser Algorithmus findet garantiert den kürzesten Lösungsweg. A^* arbeitet mit einer Bewertungsfunktion, die sowohl verschiedene Kosten einzelner Operatoren als auch eine Abschätzung der minimalen Distanz des aktuellen Zustands vom Ziel berücksichtigt. Einige aktuelle Planungssysteme, zum Beispiel HSP (Bonet & Geffner, 1999), verfügen über einen Mechanismus zur automatischen Ermittlung einer Bewertungsfunktion für ein beliebiges gegebenes Problem.

Planungsalgorithmen sind im Gegensatz zu Problemlöse-Algorithmus allgemein gehalten: Sie basieren auf einer für alle möglichen Probleme uniformen Repräsentationssprache und arbeiten unabhängig von problemspezifischem Wissen. Für die Modellierung des Affe-Banane Problems in Tab. 5b-1 wurde die für Planer typische Repräsentation verwendet, in der Operatoren mit Variablen definiert werden. Beim Problemlösen würde es dagegen ausreichen, alle möglichen Aktionen (also alle instantiierten Operatoren) anzugeben, mit dem Nachteil, dass dann sehr viele Aktionen auf ihre Anwendbarkeit hin überprüft werden müssen. Operatoren mit Variablen können in Abhängigkeit von der gerade gegebenen Situation belegt werden. Steht der Affe zum Beispiel gerade rechts, macht es wenig Sinn, die Aktion *GeheVonNach(Links, Rechts)* anzuwenden. Durch das Prädikat $pos(Affe, x)$ kann der Operator dagegen unmittelbar mit $x = Rechts$ belegt werden.

Eines der ersten Planungssysteme, das noch bis heute von Relevanz ist, ist STRIPS (Fikes and Nilsson, 1971). Im Rahmen dieses Systems wurde eine Repräsentationssprache für Zustände und Operatoren vorgeschlagen, die noch bis heute Grundlage vieler Planungssysteme ist. Allerdings hatte der ursprüngliche Algorithmus, genau wie die Mittel-Ziel Analyse (siehe 5b-3.3), das Problem, nicht mit abhängigen Teilzielen umgehen zu können.



Dieses Problem kann durch die sogenannte Sussman-Anomalie verdeutlicht werden: Um die Ziele $auff(A, B)$ UND $auff(B, C)$ von dem gegebenen Zustand aus zu erreichen, könnte B unmittelbar auf C gestellt werden. Damit wäre eines der Teilziele erreicht, aber um $auff(A, B)$ zu erreichen, müsste der Turm komplett wieder abgebaut werden. Andererseits könnte man zunächst B auf den Tisch legen und dann A auf B setzen, aber wieder ist damit nur ein Teilziel erfüllt. Das Problem bei STRIPS und der Mittel-Ziel Analyse ist, dass jeweils *ein* aktuelles Ziel fokussiert wird und zunächst alle Aktionen ausgeführt werden, die dieses Ziel herstellen. Durch diese lineare Strategie sind manche Probleme nicht lösbar. Die Sussman-Anomalie ist leicht auflösbar, wenn, nachdem Block B auf den Tisch gestellt wurde, zunächst B auf C und dann A auf B gesetzt wird. Hierbei erfolgt ein Fokuswechsel von Ziel $auff(A, B)$ auf Ziel $auff(B, C)$: A wurde auf den Tisch gestellt, um den Operator "Stelle A auf B" anwenden zu können, also um das Ziel $auff(A, B)$ zu erfüllen. Nachdem A und B frei sind, kann aber auch der Operator "Stelle B auf C" angewendet werden. Algorithmisch kann dies leicht erreicht werden, indem die Teilziele nicht in einem Stack (siehe Abb. 5b-13) sondern als Menge verwaltet werden. Das erste Planungssystem, das mit solchen Abhängigkeiten umgehen konnte, war NOAH (Sacerdoti, 1977). Alle modernen Planer arbeiten mit einer solchen nicht-linearen Strategie, bei der Teilziele "verschränkt" abgearbeitet werden können.

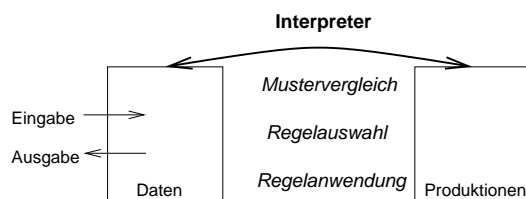


Abb. 5b-14: Architektur eines Produktionssystems

dung von Regeln erfolgt in sogenannten **Match-Select-Apply Zyklen**:

Mustervergleich (*match*): Suche alle Produktionsregeln, deren Bedingungsteil mit den Daten verträglich ist.

Auswahl (*select*): Wähle – nach einer vorgegebenen **Konfliktauflösungs-Strategie** – eine dieser Regeln aus.

Anwendung (*apply*): Wende die Regel auf die Daten im Arbeitsspeicher an.

In jedem Zyklus kommt also eine Regel zur Anwendung, die die Daten im Arbeitsspeicher verändert. In den folgenden Abschnitten werden die drei Komponenten des Interpreter-Zyklus genauer besprochen. Eine einfache Anleitung zur Implementation eines Produktionssystems in der Programmiersprache Lisp gibt (Winston & Horn, 1989).

Ein sehr einfaches Produktionssystem für das “Kaffe-Dose” (*coffee can*) Problem (Gries, 1981) ist in Abb. 5b-15 dargestellt. Eine Reihe schwarzer und weißer Bohnen kann verkürzt werden, in dem nach festen Regeln, die als Produktionsregeln repräsentiert werden, Paare benachbarter Bohnen durch eine einzelne Bohne ersetzt werden. Anstelle von “Wenn ⟨Bedingung⟩ Dann ⟨Aktion⟩” schreiben wir verkürzt ⟨Bedingung⟩ → ⟨Aktion⟩.

5b-4.1 Mustervergleich

Im einfachsten Fall, wie bei dem Produktionssystem in Abb. 5b-15, meint Mustervergleich, zu prüfen, ob der Bedingungsteil einer Regel mit einem Ausschnitt der Daten übereinstimmt. Im Allgemeinen kann der Bedingungsteil einer Regel Variablen enthalten (siehe Tab. 5b-1). Mustervergleich (*pattern matching*) meint dann, zu prüfen,

ob ein vorgegebenes Muster mit den Daten verträglich ist. Verträglichkeit heißt, dass die Variablen im Muster so belegt werden können, dass der entstandene Ausdruck mit einem Ausschnitt der Daten übereinstimmt.

Der Bedingungsteil der Regel *SteigeAufKiste* in Tab. 5b-1 ist beispielsweise mit Daten-1 verträglich, aber nicht mit Daten-2:

Muster: $ort(x), pos(Affe, x), pos(Kiste, x), auf-boden$

Daten-1: $pos(Affe, Mitte), auf-boden, pos(Kiste, Mitte), pos(Banane, Mitte)$

Daten-2: $pos(Affe, Links), auf-boden, pos(Kiste, Rechts), pos(Banane, Mitte)$.

Wenn Variable x im Muster mit *Mitte* belegt ist, dann kommen alle Ausdrücke des Musters in Daten-1 vor. Für Daten-2 gibt es keine Belegung für x , die diese Bedingung erfüllt. Würde x mit *Links* belegt, so ist der Fakt $pos(Kiste, Links)$ nicht mit den Daten verträglich; für $x = Rechts$ ist der Fakt $pos(Affe, Rechts)$ unverträglich; für $x = Mitte$ sind die Fakten $pos(Affe, Mitte)$ und $pos(Kiste, Mitte)$ unverträglich.

5b-4.2 Konfliktlösung

Das Ergebnis des Mustervergleichs liefert die Menge aller Regeln, die auf die aktuellen Daten anwendbar sind. Ist diese Menge leer, so hält das System an. Enthält die Menge genau eine Regel, so wird diese auf die Daten angewendet. Enthält die Menge mehr als eine Regel, so muss eine Entscheidung getroffen werden, welche dieser Regeln zur Anwendung kommen soll. Dies geschieht im Allgemeinen dadurch, dass man eine Präferenz-Ordnung auf den Regeln definiert (vergleiche Abb. 5b-9 und Abb. 5b-15). Diese Präferenz-Ordnung kann auf verschiedene Weise, also durch verschiedene **Strategien zur Konfliktlösung**, definiert werden (Davis & King, 1977), zum Beispiel:

Erste Übereinstimmung: Nimm die erste Regel, die mit den Daten verträglich ist. Dabei ist “erste” bezüglich der Anordnung der Regeln im Produktionsspeicher festgelegt.

Höchste Priorität: Nimm die Regel mit dem höchsten Prioritätswert. Der Prioritätswert kann dabei beispielsweise über einen “Stärkewert” der Regel ermittelt werden, der dynamisch verändert werden

Gegeben ist eine Kaffe-Dose, in der schwarze (S) und weiße (W) Bohnen in einer festen Reihenfolge angeordnet sind, beispielsweise: *W W S S W W S S*.

Gegeben sind folgende Regeln:

S W → *S*

W S → *S*

S S → *W*

Das Ziel ist, am Ende möglichst wenige Bohnen zu haben. Die Konfliktlösungs-Strategie sei, immer die oberste anwendbare Regel auszuwählen.

W W S S W W S S
W W S S W S S
W W S S S S
W S S S S
W S S S S
W S S
S S
W

Abb. 5b-15: Lösung des “Kaffe-Dose” Problems mit einem Produktionssystem

kann (Anderson, 1983), oder er kann problemspezifisch definiert werden.

Spezifischste Bedingung: Nimm die Regel, die die spezifischsten Anwendungsbedingungen hat.

Aktualität: Nimm die Regel, die sich auf ein Datenelement bezieht, das erst kürzlich (*most recently*) erzeugt worden ist.

Neuigkeit: Nimm eine Regel, die noch nicht angewendet wurde (mit einer noch nicht betrachteten Variablenbelegung).

Zufall: Wähle zufällig eine Regel.

Keine Wahl: Exploriere alle anwendbaren Regeln.

Die letztgenannte Strategie definiert eine Breiten- suchung, während alle anderen Strategien eine Tiefensuche definieren (siehe 5b-3.1).

5b-4.3 Regelanwendung

Ist eine Regel ausgewählt, so wird sie auf die Daten im Arbeitsspeicher angewendet. Man sagt auch, die Regel “feuert”. Dadurch wird der Inhalt des Arbeitsspeichers verändert. Beispielsweise transformiert die Anwendung eines Problemlöse-Operators den aktuellen Zustand in einen Folgezustand (siehe 5b-2.2).

Es gibt zwei grundlegende Verarbeitungsstrategien für Regeln: **Vorwärtsverkettung** und **Rückwärtsverkettung**. Vorwärtsverkettung meint, dass ausgehend von den aktuellen Daten im Arbeitsspeicher jeweils der Aktionsteil einer Regel – also die rechte Regelseite – ausgeführt wird. Dadurch werden die Daten schrittweise verändert. Man spricht hier auch von **datengesteuerter** (*data-driven, bottom-up*) Regelanwendung. Ist neben den aktuellen Daten ein Ziel vorgegeben, so existiert üblicherweise eine Regel, der

Form “WENN ⟨Ziel erreicht⟩ DANN halt”. Vorwärtsverkettung bewirkt also, dass die Daten schrittweise in Richtung Zielzustand transformiert werden. Rückwärtsverkettung meint dagegen, dass ausgehend von einem gegebenen Ziel immer die Regel ausgeführt wird, deren rechte Seite das Ziel unmittelbar herstellen kann. Ist die linke Seite der Regel – die Anwendungsbedingungen oder Teilziele – im aktuellen Zustand nicht erfüllt, so werden die dort genannten Bedingungen als Teilziele eingeführt und eine Regel gesucht, die diese Teilziele unmittelbar herstellt. Rückwärtsverkettung bewirkt also, dass ausgehend vom Problemlöseziel das Problem so lange in Teilziele zerlegt wird, bis Teilziele gefunden sind, die gelten oder im aktuellen Zustand unmittelbar erfüllt werden können. Man spricht hier auch von **zielgesteuerter** (*goal-driven, top-down*) Regelanwendung.

Die Mittel-Ziel Analyse (siehe 5b-3.3) kombiniert beide Strategien: Ein Zustand wird vorwärts in einen Folgezustand transformiert, wenn ein Operator, der die Distanz zum Ziel minimiert, angewendet werden kann. Ansonsten wird als neues Ziel gesetzt, die Anwendungsbedingung des Operators zu erfüllen, also rückwärts das Ziel in Teilziele zerlegt. Ein Planungssystem, das mit einem Wechselspiel aus Vorwärts- und Rückwärtsverkettung arbeitet, ist Prodigy (Veloso et al., 1995). Die Programmiersprache Prolog basiert auf Rückwärtsverkettung (Opwis & Plötzner, 1996). Eine ausführliche Darstellung von daten- und zielgesteuerten Regelsystemen zur kognitiven Modellierung gibt Möbus (1988).

5b-4.4 Das Produktionssystem ACT

Das System ACT (*adaptive character of thought*) ist eine kognitive Architektur, die im Wesentlichen als Produktionssystem realisiert ist. Bekannt wurde vor allem die Systemversion ACT* (Anderson, 1983), die aktuelle Version ist ACT-R (Anderson, 1993) sowie dessen Erweiterung um Komponenten für Informationsaufnahme (“*perception*”) und Handlungsausführung (“*motor performance*”) ACT-R/PM (Anderson & Lebière, 1998).

ACT-R besteht aus einer Rahmentheorie über Repräsentation, Anwendung und Erwerb von Wissen. Dabei werden zwei Arten von Wissen – deklaratives und prozedurales Wissen – angenommen. **Deklaratives Wissen** (*know that*) korrespondiert dabei mit Faktenwissen, beispielsweise, dass ein Kanarienvogel singen kann oder dass sieben plus vier elf ergibt. Es wird in Form sogenannter *Chunks* als Strukturen, die Fakten enthalten (vergleiche Schemata, Kapitel 3c), repräsentiert. **Prozedurales Wissen** (*know how*) korrespondiert mit Fertigkeiten, also wie deklaratives Wissen zur Lösung von Problemen angewendet werden kann. Es wird in Form von Produktionsregeln repräsentiert. Während deklaratives Wissen verbalisierbar und dem Bewusstsein zugänglich ist, repräsentiert prozedurales Wissen Automatismen.

Ein Beispiel für die Repräsentation des Turm von Hanoi Problems mit vier Scheiben (siehe 5b-2.4) in ACT-R ist in Tab. 5b-5 und Tab. 5b-6 gegeben (Anderson & Lebière, 1998, Kap. 2). ACT-R ist in der Programmiersprache Lisp implementiert. Das deklarative Wissen besteht aus *Chunks*, die den Anfangszustand (*current*) und den Zielzustand (*goal*) beschreiben, sowie Fakten-Wissen über die Reihenfolge der natürlichen Zahlen eins bis vier. Die *Chunks* sind dabei durch abstrakte Typen beschrieben. Beispielsweise legt der Typ *disk* fest, dass eine Scheibe durch ihre Größe und den Stift, auf dem sie steht, beschrieben wird. Zusätzlich wird angegeben, in welchem Problemzustand dieser Fakt gilt.

Durch Anwendung von Produktionsregeln wird der aktuelle Anfangszustand in einen Folgezustand – der dann *current* ist – überführt, solange bis der Zielzustand erreicht ist. ACT-R ist als zielgesteuertes Produktionssystem realisiert (siehe 5b-4.3). Im Bedingungssteil der Regel wird jeweils das

Tab. 5b-5: Repräsentation des deklarativen Wissens für ein Turm-von-Hanoi Problem in ACT-R

```
(chunk-type disk size peg state)
(chunk-type peg name)
(chunk-type tower-task largest current goal)
(chunk-type move-disk disk to from other test at)
(chunk-type countfact first then)
(chunk-type encode-configuration size state)
(add-dm
  (disk1c isa disk size 1 peg c state current)
  (disk2c isa disk size 2 peg a state current)
  (disk3c isa disk size 3 peg b state current)
  (disk4c isa disk size 4 peg b state current)
  (disk1g isa disk size 1 peg b state goal)
  (disk2g isa disk size 2 peg a state goal)
  (disk3g isa disk size 3 peg c state goal)
  (disk4g isa disk size 4 peg c state goal)
  (fact01 isa countfact first 0 then 1)
  (fact12 isa countfact first 1 then 2)
  (fact23 isa countfact first 2 then 3)
  (fact34 isa countfact first 3 then 4)
  (fact45 isa countfact first 4 then 5)
  (a isa peg name a)
  (b isa peg name b)
  (c isa peg name c)
  (goal isa tower-task largest 4)
  (current isa chunk))
```

aktuelle Problemlöseziel und die im aktuellen Zustand gültigen Fakten geprüft, im Aktionsteil werden Teilziele als neue aktuelle Ziele gesetzt und Fakten verändert. Die Spezialanweisungen *!push!* und *!pop!* regeln den Auf- und Abbau des Ziel-*Stacks* (siehe 5b-3.3).

Die Regel *start-tower* kommt beispielsweise zur Anwendung, wenn das aktuelle Ziel ist, ein Problem *tower-task* mit vier Scheiben zu lösen. Die Variable *=size* kann durch Mustervergleich mit dem *Chunk* (*goal isa tower-task largest 4*) belegt werden. Über den Fakt (*fact34 isa countfact first 3 then 4*) wird die Variable *=new* mit 3 belegt.

In ACT-R sind sowohl *Chunks* als auch die Produktionsregeln mit Parametern versehen: Jeder *Chunk* hat einen Aktivierungswert, der die Wahrscheinlichkeit angibt, mit der er im nächsten Verarbeitungszyklus verwendet wird. Produktionsregeln haben drei Parameter: Einen Stärkewert, der die Wahrscheinlichkeit angibt, mit der diese Regel zur Anwendung kommt, eine Wahrscheinlichkeit dafür, dass die Produktion einen bestimmten

Tab. 5b-6: Ausschnitt der Repräsentation des prozeduralen Wissens für ein Turm-von-Hanoi Problem in ACT-R

```
(p start-tower
"IF the goal is to solve a tower task of
  size =size and =size is greater than 1
THEN set a subgoal to move disk =size checking
  disk =new and change the goal to solve a
  tower task of size =new"
=goal>
  isa tower-task
  largest =size
  - largest 1
  current t
  goal t
=fact>
  isa countfact
  first =new
  then =size
==>
=goal>
  goal nil
  largest =new
=newgoal>
  isa move-disk
  disk =size
  test =new
  !push! =newgoal
(p move
"IF the goal is move disk of size n to peg x
  and all smaller disks have been checked
THEN move disk n to peg x and pop the goal"
=goal>
  isa move-disk
  test 0
  from =from
  to =to
  disk =size
=disk>
  isa disk
  size =size
==>
=disk>
  peg =to
  !pop!
```

Effekt erzielt, und einen Wert für die Kosten der Anwendung. Die Parameter werden aufgrund von Problemlöseerfahrungen verändert (siehe 5b-5.2). ACT-R verfügt über einen Lernmechanismus zum Erwerb neuer Produktionsregeln, der auf Analogiebildung (siehe 5b-5.1) basiert.

Neben dem sicher am verbreitetsten System ACT existieren eine Reihe weiterer kognitiver Architekturen, von denen einige in Box 4 aufgeführt sind. Einen Überblick über kognitive Modelle des Problemlösens und Lernens gibt Opwis (1992).

5b-5 Modellierung spezieller Aspekte des Problemlösens

5b-5.1 Analoges Problemlösen

Bisher wurde dargestellt, wie Problemlösen durch Suche im Problemraum modelliert werden kann. Der Problemlöser kann dabei auf Allgemeinwissen (siehe ACT-R, 5b-4.4) zurückgreifen und Heuristiken benutzen, die ihm helfen, die Suche in Richtung des Zielzustands zu steuern (siehe 5b-3.2 und 5b-3.3). Die Suche nach einer Problemlösung kann auf Irrwege führen und ist häufig mit hohem (kognitivem) Aufwand verbunden. In vielen Fällen weist ein Problem Ähnlichkeit zu einem bereits bekannten Problem auf. Wird dies erkannt, so kann Suche vermieden und stattdessen versucht werden, die Lösung des bekannten Problems auf das neue Problem zu übertragen. Man spricht dann von **analogem Problemlösen** (siehe Knoblich, Kap. 5a).

Es gibt zahlreiche Systeme zur Modellierung von analogem Problemlösen (siehe Textbox *Kognitive Architekturen und spezielle Modelle*). Einen Überblick gibt Weber (1994). In der KI wird anstelle von analogem Problemlösen häufig der Spezialfall des **fallbasierten Schließens** betrachtet, bei dem nur Probleme eines fest vorgegebenen Wissensbereichs miteinander in Beziehung gebracht werden (Kolodner, 1993). Im folgenden wird die *Structure Mapping Engine (SME)* (Falkenhainer et al., 1989) genauer dargestellt, die die Struktur-Vergleichs Theorie von Gentner (1983) realisiert. Wir konzentrieren uns auf den Prozess der Analogiebildung und vernachlässigen Fragen des Abrufs eines geeigneten bereits bekannten Problems (siehe Knoblich, Kap. 5a).

Kognitive Architekturen und Spezielle Modelle.

Eine Auswahl von bekannten kognitiven Architekturen:

ACT: (*Adaptive Character of Thought*): Zielgesteuertes Produktionssystem, das in deklarative und prozedurale Wissensstrukturen unterteilt ist; analoges Problemlösen, Lernen als Veränderung von Stärkewerten und der Erwerb von Produktionsregeln aus deklarativem Wissen werden modelliert; siehe Anderson und Lebière (1998).

Modellierungsbeispiel: Steuerung eines komplexen Regelsystems (Wallach & Tack, 1998).

COGENT: Ein graphisches Werkzeug für Entwurf und Analyse kognitiver Modelle; siehe Cooper, Yule, Fox, und Sutton (1998).

PSI: Ein auf einer Theorie des Handelns basierendes System, das die Interaktion eines Agenten mit einer komplexen Umgebung realisiert; siehe Schaub (1997).

SOAR: (*State, Operator And Result*): Zielgesteuertes Produktionssystem, das aus GPS (General Problem Solver) und GOMS (Card, Moran, & Newell, 1983) entstanden ist; Lernen wird als *chunking* von Produktionsregeln modelliert; siehe Newell (1990).

Modellierungsbeispiel: Wissenserwerb zur Lösung von Maschinen-Belegungs Problemen (Nerb, Ritter, & Krems, 1999).

Eine Auswahl von Modellen zum Analogen Problemlösen:

CASCADE: Modell des Problemlösens und Lernens, das auf der Selbsterklärung von Musterlösungen und analogem Transfer von Lösungen basiert; siehe VanLehn, Jones, und Chi (1992).

LISA: Modell des analogen Problemlösens, das die Steuerung von Abruf und Mapping über semantische und pragmatische Randbedingungen beschreibt; siehe Hummel und Holyoak (1997).

PAN: KI-Modellierung von Re-Deskriptionseffekten (Umrepräsentation einer Wissensstruktur unter Verwendung alternativer Relationen) beim analogen Schließen; siehe O'Hara (1992).

PRODIGY: KI Planungssystem, das verschiedene Mechanismen des Erwerbs von Kontrollwissen sowie analoges Schließen integriert. Als Analogiemechanismus wird die *derivational analogy* (Carbonell, 1986) verwendet; siehe Veloso et al. (1995).

SME: Modell der Theorie des Strukturvergleichs von Gentner (1983); siehe Falkenhainer, Forbus, und Gentner (1989).

5b-5.1.1 Gentners Theorie des Strukturvergleichs

Gentner (1983) geht von der Annahme aus, dass Wissensbereiche (*domains*) als relationale Strukturen repräsentiert werden. Zur Veranschaulichung ist in Abb. 5b-16(a) ein Ausschnitt des Wissens über das Sonnensystem dargestellt. Ein Wissensbereich besteht aus:

Objekten: Als Grundelemente, beispielsweise *Sonne*, *Planet-i*, *Planet-j*.

Attributen: Als einstellige Relationen über Objekten, die Objekteigenschaften beschreiben, beispielsweise *gelb(Sonne)*, *heiss(Sonne)*, *massiv(Sonne)*.

Relationen: Zur Beschreibung von Beziehungen zwischen Objekten (Relationen erster Ordnung), beispielsweise *zieht-an(Sonne, Planet-i)*, oder zur Beschreibung von Beziehungen zwischen Relationen (Relationen höherer Ordnung), beispielsweise *ursache(zieht-an(Sonne, Planet-i), umkreist(Planet-i, Sonne))*.

Ein Wissensbereich wird also als propositionales Netzwerk (siehe Buchner & Brandt, Kap. 3c) repräsentiert. In der graphischen Darstellung wird dabei von den Relationen ausgehend mit gerichteten Kanten (Pfeilen) auf deren Argumente verwiesen. Die Position eines Arguments in einer Relation wird durch eine Markierung (*label*) angegeben. Beispielsweise wird für die Proposition *zieht-an(Sonne, Planet-i)* die Kante zur Sonne mit *S* (Subjekt) und die Kante zum Planeten mit *O* (Objekt) markiert.

Die Grundannahme der Theorie des Strukturvergleichs besagt, dass Analogie eine Abbildung (*mapping*) des Wissens eines Bereichs (**Basis**, *base* oder *source*) auf einen anderen Bereich (**Ziel**, *target*) ist. Dabei wird angenommen, dass die Beziehungen zwischen Objekten in der Basis auch im Zielproblem gelten, während von den Objekteigenschaften abstrahiert wird. In der Mathematik werden strukturhaltende Abbildungen als Homomorphismen bezeichnet (Schmid, Wirth, & Polkehn, 1999). Beispielsweise ist dem Rutherford'schen Atommodell die Analogie "Das Wasserstoffatom ist wie unser Sonnensystem aufgebaut"

zugrundegelegt (siehe Abb. 5b-16). Werden das Objekt *Sonne* auf das Objekt *Atomkern* und das Objekt *Planet-i* auf das Objekt *Elektron-i* abgebildet, kann aus dem vorhandenen Wissen über das Sonnensystem etwa inferiert werden, dass das Elektron den Atomkern umkreist wie ein Planet die Sonne. Zudem kann durch das Übertragen einer Relation zweiter Ordnung inferiert werden, dass das Elektron den Atomkern umkreist, weil es vom Atomkern angezogen wird. Wissen aus einem Bereich kann also benutzt werden, um Beziehungen, die in einem anderen Bereich gelten, zu erklären. Nicht übertragen werden dagegen Attribute wie beispielsweise, dass der Atomkern so heiß ist wie die Sonne.

Im Allgemeinen sind für den Basis-Bereich zahlreiche Beziehungen zwischen den Objekten bekannt, insbesondere auch Relationen höherer Ordnung wie Ursache-Wirkungs-Beziehungen. Für den Ziel-Bereich sind dagegen nur die Basisobjekte, eventuell mit einigen Attributen, und einige Relationen (erster Ordnung) bekannt.

Die Theorie des Strukturvergleichs betont also die *syntaktische Struktur* eines Wissensbereichs. Semantische Ähnlichkeiten, also eine Übereinstimmung von Objekten bezüglich ihrer Eigenschaften, werden, anders als etwa im Ansatz von Hummel und Holyoak (1997), außeracht gelassen. Die Abbildung von Wissen eines Bereichs auf einen anderen wird nach Gentner (1983) durch folgende **Randbedingungen** (*constraints*) gesteuert:

Abbildung erster Ordnung: Objekte aus dem Basis-Bereich können auf anders benannte Objekte aus dem Ziel-Bereich abgebildet werden. Aber Relationen aus dem Basis-Bereich müssen auf gleichnamige Relationen aus dem Ziel-Bereich abgebildet werden.

Das heißt, *Sonne* kann auf *Atomkern* abgebildet werden, aber *zieht-an(x, y)* muss auf *zieht-an(x', y')* abgebildet werden.

Isomorphie: Eine Menge von Objekten aus dem Basis-Bereich muss eineindeutig auf die Objekte aus dem Ziel-Bereich abgebildet werden.

Das heißt *Sonne* kann nur *entweder* auf *Atomkern* oder auf *Elektron-j* abgebildet werden, aber nicht auf beide Objekte und es können nicht *Sonne* und *Planet-i* beide auf

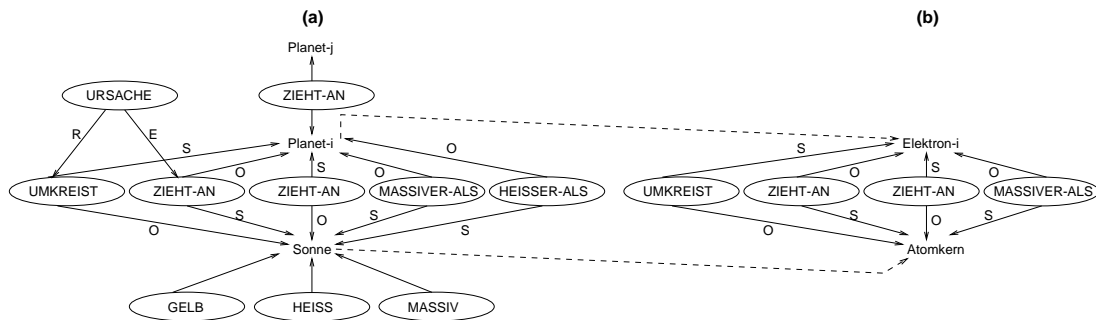


Abb. 5b-16: Repräsentation eines Ausschnitts von Wissen über ein Sonnensystem (a) und ein Atom (b) als Struktur aus Objekten und Relationen zusammen mit einer strukturerhaltenden Abbildung für die Rutherford Analogie “Das Atom ist wie das Sonnensystem” (nach Gentner, 1983)

Atomkern abgebildet werden. Mathematisch spricht man hier von einer bijektiven Abbildung.

Strukturelle Konsistenz: Für eine gegebene Abbildung von Objekten aus dem Basis-Bereich auf Objekte aus dem Ziel-Bereich muss gelten, dass die Abbildung einer Relation verträglich mit den Argumenten erfolgen muss. Mathematisch ist dies die Bedingung für eine homomorphe Abbildung.

Das heißt, wenn *Sonne* auf *Atomkern* abgebildet wurde und *Planet-i* auf *Elektron-i*, dann *muss* die Relation *zieht-an(Sonne, Planet-i)* auf *zieht-an(Atomkern, Elektron-i)* abgebildet werden. Eine Abbildung auf *zieht-an(Elektron-j, Atomkern)* wäre beispielsweise nicht struktur-verträglich.

Systematizität: Es werden bevorzugt möglichst große relationale Gefüge in Kontrast zu einzelnen Relationen abgebildet, insbesondere solche Relationen, die über Relationen höherer Ordnung miteinander verbunden sind (*systematicity principle*).

Die genannten Randbedingungen gelten dabei sowohl für die Abbildung von Relationen auf bereits bekannte Relationen im Ziel als auch für die Inferenz von Relationen, also deren Übernahme ins Ziel.

5b-5.1.2 Die Structure Mapping Engine

Die *Structure Mapping Engine* (SME) von Falkenhainer et al. (1989) realisiert die Annahmen der

Theorie des Strukturvergleichs auf folgende Weise in einem Computermodell:

Paarweise Zuordnung: Erzeugung aller möglichen paarweisen Zuordnungen zwischen Objekten von Basis zu Ziel und Relationen von Basis zu Ziel sowie einer Liste der Relationen in der Basis, die nicht im Ziel existieren (und die möglicherweise auf das Ziel übertragen werden).

Globale Zuordnung: Kombination der paarweisen Zuordnungen zu maximal konsistenten Strukturen sowie Übernahme von Basis-Relationen ins Ziel (*candidate inferences*).

Bewertung und Auswahl: Für jede globale Zuordnung wird eine Bewertung (*structural evaluation score*) bestimmt und die am besten bewertete Zuordnung wird ausgewählt. Zentraler Bestandteil der Bewertung ist die relative Größe der globalen Zuordnung.

Formal entspricht dieses Vorgehen der Identifikation einer maximalen Clique in einem Kompatibilitätsgraphen (Schädler & Wysotzki, 1999).

Eine Veranschaulichung der Arbeitsweise des Algorithmus ist für die Rutherford-Analogie (siehe Abb. 5b-16) in Tab. 5b-7 dargestellt. Die größte globale Zuordnung ergibt sich, wenn *Sonne* auf *Atomkern* und *Planet-i* auf *Planet-j* abgebildet werden. Die mit den Relationen *zieht-an(Sonne, Planet-i)* und *umkreist(Planet-i, Sonne)* verbundene Relation *ursache* wird als **Inferenz** in den Ziel-Bereich übernommen.

Tab. 5b-7: Herstellung der Rutherford Analogie mit der SME (Elemente der besten globalen Zuordnung sind fett markiert)

1 Sonne – Atomkern	2 Sonne – Elektron-i
3 Planet-i – Atomkern	4 Planet-i – Elektron-i
5 Planet-j – Atomkern	6 Planet-j – Elektron-j
7 umkreist(Planet-i, Sonne) – umkreist(Elektron-i, Atomkern)	
8 zieht-an(Sonne, Planet-i) – zieht-an(Atomkern, Elektron-i)	
9 zieht-an(Sonne, Planet-i) – zieht-an(Elektron-i, Atomkern)	
10 zieht-an(Planet-i, Sonne) – zieht-an(Atomkern, Elektron-i)	
11 zieht-an(Planet-i, Sonne) – zieht-an(Elektron-i, Atomkern)	
12 massiver-als(Sonne, Planet-i) – massiver-als(Atomkern, Elektron-i)	
13 heisser-als(Sonne, Planet-i)	
14 ursache(zieht-an(Sonne, Planet-i), umkreist(Planet-i, Sonne)) – <i>Inferenz: ursache(zieht-an(Atomkern, Elektron-i), umkreist(Elektron-i, Atomkern))</i>	

5b-5.2 Erwerb von Problemlösefertigkeiten

Macht ein Problemlöser Erfahrung mit einem Problembereich, so kann er dabei Wissen erwerben, das ihm das nachfolgende Lösen ähnlicher Probleme erleichtert (*learning by doing*). Lernen aufgrund von Problemlöseerfahrung betrifft vor allem prozedurales Wissen (siehe 5b-4.4). Im folgenden werden zwei Aspekte des Wissenserwerbs dargestellt: **Beschleunigungseffekte** (*speed-up effects*), die durch Erhöhung des Stärkewerts von Produktionsregeln oder durch Kombination von Regel zu komplexeren Einheiten beschrieben werden können, und der Erwerb von **Problemlösestrategien** als Aufbau neuer Regeln oder Schemata.

5b-5.2.1 Verstärkung und Kombination von Regeln

Im Produktionssystem ACT-R (siehe 5b-4.4) können die mit Produktionsregeln assoziierten Parameter – Stärke, Effekt-Wahrscheinlichkeit und Kosten-Wahrscheinlichkeit – in jedem Interpreter-Zyklus verändert werden (*tuning*). Die Erhöhung dieser Parameterwerte bewirkt, dass die entsprechende Regel bevorzugt zur Anwendung kommt (vergleiche Konfliktresolution in Abschnitt 5b-4). Eine detaillierte Darstellung der entsprechenden Funktionen zur Parameterveränderung und deren Auswirkung auf die Wahrscheinlichkeit und Kosten der Anwendung von Regeln findet sich in Anderson (1993). Durch Erhöhung der Stärkewerte kann das sogenannte **Potenzgesetz des Lernens** (*power law of practice*) modelliert werden (Newell & Rosenbloom, 1981). Mit zunehmender Übung werden Probleme aus einem Bereich zunehmend schneller gelöst, was dadurch beschrieben werden kann, dass bei der Suche weniger *Backtracking* (siehe 5b-3.1) notwendig ist, weil die lösungsrelevanten (verstärkten) Regeln direkt angewendet werden. In der KI wird der Erwerb von Problemlösefertigkeiten beispielsweise durch Verstärkungslernen (*reinforcement learning*) modelliert (Sutton & Barto, 1998). Gelernt wird dabei die Wahrscheinlichkeit mit der eine Aktion in einem bestimmten Zustand ausgeführt wird.

Eine zur Veränderung von Stärkewerten alter-

native Möglichkeit, das Potenzgesetz des Lernens nachzubilden, ist die Verknüpfung oder **Komposition** (*composition*) von Produktionsregeln (Anderson, 1983), die auch als *Operator-Chunking* (Laird, Rosenbloom, & Newell, 1986) bezeichnet wird. Werden zwei Operatoren wiederholt nacheinander angewendet, so können sie zu einem komplexeren Operator zusammengefasst werden. Dies geschieht durch Verknüpfung der Bedingungs- und Aktionsteile:

Produktion 1: WENN $\langle B_1 \rangle$ DANN $\langle A_1 \rangle$

Produktion 2: WENN $\langle B_2 \rangle$ DANN $\langle A_2 \rangle$

Verknüpfung: WENN $\langle B_1 \circ B_2 \rangle$ DANN $\langle A_1 \circ A_2 \rangle$.

Bei logisch repräsentierten Operatoren (wie in Abb. 5b-1 gezeigt) erfolgt die Verknüpfung durch Vereinigung der Anwendungsbedingungen sowie der Vereinigung der ADD- und DEL-Effekte. In der KI-Planung wird diese Art des Lernens als Makro-Bildung bezeichnet. Das Lernen von Makros wird jedoch seit den neunziger Jahren kaum mehr in Planungssystemen berücksichtigt. Eine wesentliche Ursache hierfür ist das sogenannte *utility problem* (Minton, 1985): Zwar reduziert sich die Anzahl von notwendigen Interpreter-Zyklen bei Verwendung von Makros, da ja komplexere Aktionen innerhalb *einer* Regelanwendung ausgeführt werden, gleichzeitig erhöht sich aber der Aufwand beim Mustervergleich, da nun neben den gegebenen elementaren Produktionen auch noch eine Menge von Makros auf ihre Anwendbarkeit hin überprüft werden muss. Ohne eine vernünftige Heuristik zur Entscheidung, unter welchen Bedingungen Regeln komponiert werden sollen, wird das System leicht mit Makros “überschwemmt”.

5b-5.2.2 Erwerb von Strategien

Die Veränderung von Stärkewerten und Regel-Verknüpfung beschreiben Lernen als Optimierungsprozess. Eine “höhere Stufe” des Lernens ist der Erwerb von neuen Problemlösestrategien. Hier wird Wissen über die Lösungsstruktur eines Problembereichs aufgebaut. Beispielsweise zeigen Anzai und Simon (1979), dass Probanden beim Lösen eines Turm-von-Hanoi Problems (siehe 5b-2.4) zunächst die korrekte Folge von Scheibenbewegungen durch heuristische Suche erzeugen,

aber bereits nach einigen Durchläufen die in Abschnitt 5b-2.4 angegebene rekursive Lösungsstrategie erworben haben. Die rekursive Regel beschreibt insbesondere die **Zielstruktur** des Problems. Wie solche rekursiven Regeln aus Problemlöseerfahrung aufgebaut werden können, beschreiben Schmid und Wysotzki (2000).

Strategien können entweder als **Lösungsschema** (Novick & Holyoak, 1991) oder über aus solchen Schemata generierten Produktionsregeln (Anderson, 1993) repräsentiert werden. Die wechselseitige Überführbarkeit von Schemata und Regeln diskutieren Rumelhart und Norman (1981). Der Aufbau von **Lösungsschemata** wird üblicherweise im Kontext von analogem Problemlösen beschrieben. Beispielsweise können die Strukturen “Sonnensystem” und “Wasserstoffatom” zu “Zentralkraftsystem” generalisiert werden, wobei Sonne und Atomkern zu einem allgemeinen Konzept “zentrales Objekt” und Planeten und Elektronen zu “periphere Objekte” verallgemeinert werden. Allgemein meint Abstraktion oder **Generalisierung**, dass nur die strukturell gemeinsamen Teile von zwei Bereichen erhalten bleiben – also Objekte und Relationen wegfallen können. Konstante Größen, die sich in Basis und Ziel unterscheiden, werden durch eine Variable ersetzt. Werden nur Objekte durch Variablen ersetzt, spricht man von Generalisierung erster Ordnung; werden auch Relationen (Prädikate oder Funktionen) durch Variablen ersetzt, spricht man von Generalisierung höherer Ordnung. Beispielsweise beschreibt Anderson (1993) wie der Lisp-Ausdruck (+ 712 91) in Analogie zu dem Ausdruck (* 2 3) erzeugt werden kann. Eine Generalisierung wäre: “Um in Lisp zwei Zahlen zahl-1 und zahl-2 miteinander durch eine mathematische Operation math-op zu verknüpfen, schreibe (math-op zahl-1 zahl-2)”.

Generalisierung über beispielhafte Erfahrung entspricht einem **induktiven Schluss** von Beispielen auf eine allgemeine Regel. Dabei kann es sich um Regeln zur Zuordnung von Objekten zu semantischen Kategorien handeln (siehe Waldmann, Kap. 3b) oder um Regeln, die beschreiben, auf welche Art ein Problem aus einem bestimmten Wissensbereich gelöst werden kann. Induktion wird in der KI vor allem im Bereich Maschi-

nelles Lernen (Mitchell, 1997) erforscht. Zum Erwerb von Regeln (Kontrollwissen) wird beispielsweise die Methode des erklärungs-basierten Lernens verwendet. Wie Kontrollwissen zur Lösung von sogenannten Blockwelt-Problemen aus Beispiellösungen induziert werden kann, beschreiben Martín und Geffner (2000).

5b-5.3 Problemlösen und Wissen

In **komplexen Problembereichen**, wie dem Erstellen von Computerprogrammen, dem mathematischen Beweisen oder dem Steuern komplexer Systeme, ist der Weg vom Novizen zum Experten ein langwieriger Prozess. Ein **Experte** verfügt über reichhaltiges Erfahrungswissen: Umfassendes Faktenwissen über einen Problembereich, das er effizient organisiert und damit schnell zugreifbar hat, typische Lösungsmuster für Standardprobleme (Fallbasiertes Schließen), automatisiertes Handlungswissen (vergleiche prozedurales Wissen, Abschnitt 5b-4.4), sowie bereichsspezifische Strategien. In vielen Fällen ist er deshalb nicht auf die Suche nach einem Lösungsweg mithilfe schwacher Heuristiken (vergleiche Abschnitt 5b-3) angewiesen.

Zur empirischen Untersuchung des Wissenserwerbs in komplexen Bereichen werden häufig **Intelligente Tutorielle Systeme (ITS)** eingesetzt. Damit kann gleichzeitig eine systematische und individuelle Vermittlung von Wissen und Fertigkeiten in einem Bereich erfolgen, und Annahmen über den Erwerb von Problemlösefertigkeiten, die in der tutoriellen Komponente realisiert sind, können überprüft werden. Im Kontext der ACT-Theorie (siehe 5b-4.4) wurden einige tutorielle Systeme entwickelt, beispielsweise für den Erwerb von Programmierkenntnissen in Lisp (Anderson, Conrad, & Corbett, 1989; Weber, 1996). Ein bekanntes Beispiel zur Untersuchung von Expertenwissen sind die Experten-Novizen Vergleiche im Bereich Schach von Chase und Simon (1973).

In der KI wurde, vor allem in den achtziger Jahren, versucht, Expertenwissen in **Expertensystemen** umzusetzen. Bekannt wurde beispielsweise das System Mycin zur medizinischen Diagnose (Shortliffe, 1976). Schachcomputer wie *Deep Blue* (siehe Textbox *Deep Blue versus Kasparov*) sind ebenfalls ein Beispiel für Experten-

systeme. Problematisch am Aufbau von Expertensystemen ist vor allem die **Wissensdiagnose** (Tergan, 1986). Häufig werden direkte Verfahren wie Interviews oder Struktur-Lege-Techniken verwendet, um Aufschluss über das Wissen eines Experten zu erhalten. Dabei ist man darauf angewiesen, dass Experten ihr Wissen verbalisieren können. Expertentum zeichnet sich aber gerade durch einen hohen Anteil an prozeduralem und strategischem Wissen aus, das nicht unbedingt dem bewussten Reflektieren zugänglich ist. Ein alternativer Zugang ist es, Experten mit typischen Problemen zu konfrontieren und zu versuchen, die verwendeten Problemmerkmale und Lösungsstrategien durch indirekte Methoden zu ermitteln. Beispielsweise haben Block et al. (1974) Experten in der medizinischen Diagnostik Röntgenbilder vorgelegt und mithilfe von Klassifikationsverfahren ermittelt, auf welche Merkmale die Mediziner ihr Urteil über eine vorliegende Herzerkrankung stützen.

5b-6 Ausblick

Computermodelle des Problemlösens beschreiben Problemlösen als Suche nach einer Folge von Aktionen, mit denen ein gegebener Anfangszustand in einen gewünschten Zielzustand überführt werden kann. Der Suchprozess kann durch eine Heuristik gesteuert werden – beispielsweise wird immer bevorzugt der Operator ausgewählt, der den Unterschied des aktuellen Zustands zum Ziel am stärksten reduziert. Die Transformation von Zuständen durch Anwendung von bedingten Regeln wird häufig durch Produktionssysteme modelliert. Alternativ zur Modellierung von Problemlösen als Suche kann Problemlösen durch Analogiebildung zu bereits bekannten Problemen beschrieben werden. Löst man mehrere Probleme aus einem Bereich, so wird dabei Problemlösewissen erworben. Dadurch kann die Suche nach Lösungen beschleunigt werden. Durch Generalisierung können neue Problemlösestrategien aufgebaut werden, die den Suchprozess durch eine bereichsspezifische Steuerung von Regelanwendungen ersetzen. Der Weg vom Novizen zum Experten kann in komplexen Problembereichen sehr langwierig sein. Die Entwicklung von Expertensystemen setzt die Dia-

gnose des Problemlösewissens, über das Experten verfügen, voraus.

In diesem Kapitel wurde Problemlösen als spezieller Prozess der symbolischen Informationsverarbeitung beschrieben. Teilweise wird kritisiert, dass dieses Herangehen wesentliche Aspekte menschlichen Problemlösens ignoriert. Menschen lösen Probleme eingebettet in ihre Umgebung, sie können auf Umwelteinflüsse reagieren und das Lösen von Problemen wird durch aktuelle Motive und Bedürfnisse (siehe Langens & Puca, Kap. 2a) beeinflusst. Diese Aspekte des Problemlösens werden beispielsweise unter dem Stichwort "situiertes Handeln" (*situated action*) diskutiert (Norman, 1993).

Kernsätze

Computermodellierung kognitiver Prozesse liegt im Schnittbereich von kognitiver Psychologie und Künstlicher Intelligenz. Computermodellierung ist eine methodische Ergänzung zur empirischen Forschung, die es ermöglicht, Annahmen über kognitive Prozesse präzise zu formulieren.

Bei der Computermodellierung von Problemlöseprozessen werden vor allem Transformationsprobleme betrachtet, also Probleme, bei denen ein gegebener Anfangszustand durch Anwendung einer Folge von Operatoren in den gewünschten Zielzustand überführt werden kann.

Ein Problem wird formal durch Anfangszustand, Zielzustand und eine Menge von Operatoren repräsentiert. Operatoren werden im Allgemeinen als bedingte Regeln der Form "Wenn (Bedingung) Dann (Aktion)" repräsentiert, solche Regeln werden auch als Produktionsregeln bezeichnet.

Die Formulierung eines Problems durch Anfangszustand, Zielzustand und Operatoren definiert einen Problemraum mit allen möglichen Problemzuständen als Knoten; ein Zustand, der durch Anwendung eines Operators in einen anderen Zustand überführt werden

kann, ist mit diesem durch eine Kante verbunden.

Eine Problemlösung ist definiert als eine Abfolge von Operatoren, die einen gegebenen Anfangszustand in den gewünschten Zielzustand überführt, und Problemlösen meint die Suche nach einer solchen Abfolge.

Die Suche nach einem Lösungsweg kann durch Heuristiken gesteuert werden. Ein Beispiel für eine heuristische Suchstrategie ist die Mittel-Ziel-Analyse, bei der bevorzugt solche Operatoren angewendet werden, die den Unterschied zwischen gegebenem Zustand und zu erfüllenden Teilzielen am stärksten reduzieren.

Ein Produktionssystem steuert die Anwendung von Produktionsregeln auf Daten durch *Match-Select-Apply* Zyklen – also durch die Identifikation aller anwendbaren Regeln durch Mustervergleich, die Auswahl einer dieser Regeln auf Basis einer Strategie zur Konfliktlösung und die Veränderung der Daten durch Anwendung der ausgewählten Regel.

Die kognitive Architektur ACT ist als zielgesteuertes Produktionssystem realisiert; Wissen ist in deklaratives Wissen (Fakten) und prozedurales Wissen (in Form von Produktionsregeln) unterteilt.

Mit der *Structure Mapping Engine* wird analoges Problemlösen als strukturverträgliche Abbildung eines bekannten Wissensbereichs auf einen neuen Bereich modelliert; dabei werden Beziehungen, die im bekannten Bereich gelten, auf den neuen Bereich übertragen.

Aufgrund von Problemlöseerfahrung in einem Bereich kann die Auswahl und Anwendung von Aktionen optimiert werden und es können Lösungsstrategien erworben werden, wodurch weitere Probleme aus diesem Bereich effizienter gelöst werden können.

Intelligente Tutorsysteme können zur Vermittlung von Problemlösefertigkeiten in komplexen Wissensbereichen und gleichzeitig zur

Analyse des Lernprozesses eingesetzt werden.

Hauptproblem beim Entwurf von Expertensystemen ist die Wissensdiagnostik, also die Analyse des Wissens, das einen menschlichen Experten zur effizienten und erfolgreichen Problemlösung befähigt.

Schlüsselbegriffe

Abbildung erster Ordnung (*first order mapping*): Im Kontext des **analogen Problemlösens** eine Zuordnung von Objekten eines Basis-Bereichs auf einen Ziel-Bereich, so dass die Beziehungen zwischen den Objekten erhalten bleiben. Werden zusätzlich Relationen übertragen, spricht man von einer Abbildung höherer Ordnung.

Abgeschlossenheitsannahme (*closed world assumption*): Annahme, dass alles, was in einer gegebenen Situation nicht explizit als gültig bekannt ist, in dieser Situation auch nicht vorliegt.

ACT (*adaptive character of thought*): Kognitive Architektur, die als **Produktionssystem** realisiert ist; Wissen ist in eine deklarative (Faktenwissen) und eine prozedurale (Fertigkeiten) Komponente unterteilt.

Anfangszustand (*initial state*): **Problemzustand**, mit dem ein Problemlöser anfänglich konfrontiert ist.

Anwendungsbedingung (*precondition*): Fakten, die in einem Zustand erfüllt sein müssen, damit ein **Operator** anwendbar ist.

Basis-Problem (*base/source problem*): Im Rahmen des **analogen Problemlösens**, ein Problem, dessen Lösung bereits bekannt ist; beziehungsweise ein Bereich, über den bereits Wissen vorhanden ist.

Bergsteigen (*hill climbing*): Heuristische **Suchstrategie**, die darauf basiert, immer den Operator auf den aktuellen Zustand anzuwenden, der die Distanz zum Ziel am stärksten minimiert.

Bewertungsfunktion (*evaluation function*): Funktion zur Bewertung der Güte eines Problemzustands, beispielsweise dessen Distanz zum Ziel.

Heuristik (*heuristic*): Daumenregel zur Steuerung der Suche nach einer Problemlösung.

Kellerspeicher (*stack*): Speicherstruktur, nach dem Prinzip “*last in first out*”; neue Elemente werden oben auf den *Stack* gelegt (*push*), das oberste Element wird nach erfolgreicher Bearbeitung vom *Stack* entfernt (*pop*).

Kognitionswissenschaft (*cognitive science*): Interdisziplinäres Forschungsgebiet zum Gegenstandsbereich Kognition, auf der Basis von Theorien, Methoden, und empirischen Ergebnissen der Philosophie, Linguistik, Kognitiven Psychologie, Künstlichen Intelligenz und Neurowissenschaften.

Kognitive Architektur (*cognitive architecture; unified theory of cognition*): Rahmen zur Erstellung spezifischer Computermodelle, mit festvorgegebenen Grundprinzipien der Informationsverarbeitung.

Konfliktlösung (*conflict resolution*): Anwendung einer Strategie zur Auswahl einer **Produktionsregel** aus einer Menge anwendbarer Regeln im Kontext eines **Produktionssystems**.

Match-Select-Apply Zyklus (*match-select-apply cycle*): Beschreibung der Arbeitsweise des Interpreters eines **Produktionssystems**; durch Mustervergleich werden alle **Produktionsregeln**, die auf die aktuellen Daten anwendbar sind, identifiziert; eine dieser Regeln wird ausgewählt (siehe **Konfliktlösung**) und auf die Daten angewendet.

Mittel-Ziel-Analyse (*means-end analysis*): Heuristische **Suchstrategie**, die darauf basiert, Teilziele zu generieren und den Unterschied zwischen einem Zustand und diesen Teilzielen durch Anwendung von Operatoren zu reduzieren.

Operator (*operator*): Abstrakte Beschreibung einer möglichen Handlung, die in einem gegebenen Problemzustand ausführbar ist.

Problem (*problem*): Im Kontext der Computermodellierung von Problemlöseprozessen ist ein Problem definiert durch einen **Anfangszustand**, einen **Zielzustand** und eine Menge von **Operatoren**.

Problemlösen (*problem solving*): Im Kontext der Computermodellierung von Problemlöseprozessen ist Problemlösen definiert als die Suche nach einer Folge von Operatoranwendungen (siehe **Suchstrategie**) zur Überführung eines **Anfangszustands** in einen **Zielzustand**.

Problemlösen, analoges (*analogical problem solving*): Lösung eines gegebenen **Ziel-Problems** durch Übertragung von Wissen eines bereits bekannten **Basis-Problems**.

Problemlösung (*problem solution*): Im Kontext der Computermodellierung von Problemlöseprozessen ist eine Problemlösung definiert als eine Folge von Operatoranwendungen, mit der ein **Anfangszustand** in einen **Zielzustand** überführt werden kann.

Problemraum (*problem space, state space*): Menge aller möglichen Zustände, die ein Problem annehmen kann. Zustände, die durch Operatoranwendung ineinander überführbar sind, werden durch Kanten verbunden.

Problemmzustand (*problem state*): Beschreibung einer aktuellen oder möglichen Situation, in der sich ein Problemlöser befindet, beispielsweise durch alle relevanten Fakten, die in dieser Situation gelten.

Produktionsregel (*production rule*): Bedingte Regel der Form WENN (Bedingung) DANN (Aktion).

Produktionssystem (*production system*): Ein System zur Anwendung von **Produktionsregeln** auf Daten. Die Anwendung wird durch einen Interpreter gesteuert, der in sogenannten **Match-Select-Apply Zyklen** arbeitet.

Regelanwendung, datengesteuert/zielgesteuert (*data-driven/goal-driven processing*): Verarbeitung von den Daten ausgehend hin zu einem gegebenen Ziel (üblicherweise durch vorwärtsgerichtete Anwendung durch **Produktionsregeln** realisiert) bzw. Verarbeitung vom Ziel ausgehend (üblicherweise durch rückwärtsgerichtete Anwendung von **Produktionsregeln** realisiert).

Rekursion (*recursion*): Die Definition eines Problems unter Rückgriff auf sich selbst.

Sackgasse (*impasse*): Zustand, auf den kein Operator anwendbar ist.

SME (*structure mapping engine*): Computermodell zum analogen Problemlösen durch strukturverträgliche Abbildung eines bekannten Wissensbereichs auf einen neuen Bereich.

Suchbaum (*search tree*): Repräsentation der Teile des Problemraums, die bei Anwendung einer **Suchstrategie** erzeugt und exploriert werden.

Suchstrategie (*search strategy*): Im Kontext des **Problemlösens** ein algorithmisches Verfahren, mit dem festgelegt wird, in welcher Reihenfolge nachfolgende **Problemmzustände** für einen gegebenen Zustand erzeugt werden; das Verfahren terminiert, wenn ein bestimmter Zustand (zum Beispiel der **Zielzustand**) erreicht ist; mittels der Suchstrategie werden bestimmte Teile eines **Problemraums** exploriert, dabei entsteht ein **Suchbaum**; siehe **Suchstrategie**

Suchstrategie, heuristische (*heuristic search*): **Suchstrategie**, bei der die Erzeugung von nachfolgenden Problemmzuständen nach einer **Heuristik** erfolgt; die Güte eines Problemmzustands wird durch eine **Bewertungsfunktion** eingeschätzt und jeweils der am besten bewertete Zustand wird weiterbetrachtet; siehe **Bergsteigen, Mittel-Ziel-Analyse**.

Suchstrategie, uninformierte (*blind search*): Systematische **Suchstrategie** zur schrittweisen Erzeugung von nachfolgenden Problemmzuständen; Standardverfahren sind Tiefensuche und Breiten-suche.

Ziel-Problem (*target problem*): Im Rahmen des **analogen Problemlösens**, ein Problem, dessen Lösung noch unbekannt ist; beziehungsweise ein Bereich, über den nur wenig Wissen vorhanden ist.

Zielzustand (*goal state*): **Problemmzustand**, in dem alle Problemlöseziele erfüllt sind.

Zyklus (*cycle*): Eine Folge von Operatoranwendungen führt von einem gegebenen Zustand zu diesem Zustand zurück.

Weiterführende Literatur

Gardner, H. (1989). *Dem Denken auf der Spur: Der Weg der Kognitionswissenschaft*. Stuttgart: Klett-Cotta. (Original erschienen 1987 (rev. ed.): “The mind’s new science: A history of the cognitive revolution”, New York, NY: Basic Books) – (Allgemeinverständliche Einführung in die Entwicklung des Gebietes Kognitionswissenschaft.)

Görz, G., Rollinger C.-R., & Schneeberger, J. (Hrsg.) (2000). *Handbuch der Künstlichen Intelligenz*. München: Oldenbourg. – (Sammlung von Überblickstexten zu verschiedenen Bereichen der KI, insbesondere ist Kapitel 2, “Kognition”, von G. Strube et al. von Interesse.)

Johnson-Laird, P. N. (1996). *Der Computer im Kopf: Formen und Verfahren der Erkenntnis*. München: dtv. (Original erschienen 1993 (2nd ed.): “The computer and the mind: An introduction to cognitive science”, London: Fontana Press) – (Allgemeine Darstellung von Ansätzen der Computermodellierung für

- verschiedener Themenbereiche der kognitiven Psychologie.)
- Kurzweil, R. (1993). *Das Zeitalter der Künstlichen Intelligenz*. Hanser. (Original erschienen 1992: "The Age of Intelligent Machines", Cit Press) (Reichlich bebildeter einführender und unterhaltsamer Überblick über Geschichte und aktuelle Themen der KI mit vielen Bezügen zu benachbarten Disziplinen.)
- Opwis, K., & Plötzner, R. (1996). *Kognitive Psychologie mit dem Computer - Ein Einführungskurs zur Simulation geistiger Leistungen mit Prolog*. Heidelberg: Spektrum Akademischer Verlag. – (Einführung in die Entwicklung von Computermodellen.)
- Osherson, D. (general Ed.). *An Invitation to Cognitive Science*. Cambridge, MA: MIT Press. Vol 3: Smith, E. E. and Osherson, D. (Eds.) (1995). Thinking. Vol 4: Scarborough, D. & Sternberg, S. (Eds.) (1998). Methods, Models, and Conceptual Issues. – (Sammlung von kognitionswissenschaftlichen Arbeiten zu ausgewählten Problemen.)
- Russell, S. J. and Norvig, P. (1995). *Artificial Intelligence. A Modern Approach*. Englewood Cliffs, NJ: Prentice-Hall. – (Das Standard-Lehrbuch für den Einstieg in die Künstliche Intelligenz.)
- Schmid, U. & Kindsmüller, M. (1995). *Kognitive Modellierung. Eine Einführung in die logischen und algorithmischen Grundlagen*. Heidelberg: Spektrum Akademischer Verlag. – (Einführendes Lehrbuch in die Methoden der Künstlichen Intelligenz für Psychologen.)
- Stillings, N. A., Weisler, S. E., Chase, C. H., Feinstein, M. H., Garfield, J. L., & Rissland, E. L. (1995). *Cognitive Science: An Introduction* (2nd Edition). Cambridge, MA: MIT Press. – (Recht aktueller Überblick über den Forschungsbereich der Kognitionswissenschaft.)
- Strube, G. (in Druck). Cognitive Modeling. In N. Smelser & P. B. Baltes, *International Encyclopedia of the Social and Behavioral Sciences*. Oxford: Elsevier. – (überblicksdarstellung von Fragen der kognitiven Modellierung.)
- Strube, G., Becker, B., Freksa, C., Hahn, U., Opwis, K., & Palm, G. (Eds.) (1996). *Wörterbuch der Kognitionswissenschaft*. Stuttgart: Klett-Cotta. – (Nachschlagewerk, das alle Gebiete der Kognitionswissenschaft abdeckt. Zentrale Begriffe werden in längeren Artikeln erläutert.)
- Winston, P. H. (1992). *Artificial Intelligence* (3rd Edition). Reading, MA: Addison-Wesley. – (Ein KI-Lehrbuch, das an einigen Stellen Bezüge zur Kognitionswissenschaft herstellt.)

Einschlägige Journals

Cognitive Science

Cognitive Science Quarterly

Cognitive Systems Research

Artificial Intelligence

Webseiten

<http://www.gk-ev.de> – Seite der Deutschen Gesellschaft für Kognitionswissenschaft mit weiterführenden Links zu Tagungen, Institutionen und Personen

<http://www.cognitivesciencesociety.org> – Seite der *Cognitive Science Society* mit weiterführenden Links

<http://www.kuenstliche-intelligenz.de/fb1/> – Seite des Fachbereich 1 (Künstliche Intelligenz) der Gesellschaft für Informatik (GI)

<http://www.elsevier.nl/homepage/> – Seite von Elsevier, für die Journals *Artificial Intelligence*, *Cognitive Science* und *Cognitive Systems Research*

<http://www.iig.uni-freiburg.de/cognition/csq/> – Seite für das Journal *Cognitive Science Quarterly*

<http://cogprints.soton.ac.uk/> – *Cog-Prints*, Archiv für kognitionswissenschaftliche Literatur

<http://www.cs.reading.ac.uk/people/dwc/ai.html> – *The World Wide Web Virtual Library: Artificial Intelligence*

<http://transit-port.net/AI.CogSci.Robotics/index.html> – *AI, Cognitive Science, and Robotics Research Groups and Resources*

<http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/html/air.html> – *Artificial Intelligence Repository* (Sammlung von Software) der *Carnegie-Mellon University*

<http://act.psy.cmu.edu/> – *ACT Research Homepage*

Danksagung.

Ich bedanke mich herzlich bei Gerhard Strube, Heike Pisch, Knut Polkehn und Julia Nitschke für Kommentare und kritische Durchsicht einer früheren Version des Manuskripts.

Literatur

- Anderson, J. (1993). *Rules of the mind*. Hillsdale, NJ: Lawrence Erlbaum.
- Anderson, J. R. (1983). *The architecture of cognition*. Cambridge, MA: Harvard University Press.
- Anderson, J. R., Conrad, F. G., & Corbett, A. T. (1989). Skill acquisition and the LISP tutor. *Cognitive Science*, 13, 467-505.
- Anderson, J. R., & Lebière, C. (1998). *The atomic components of thought*. Mahwah, NJ: Erlbaum.
- Anzai, Y., & Simon, H. (1979). The theory of learning by doing. *Psychological Review*, 86, 124-140.
- Block, H., Böck, G., Cobet, H., Kukla, F., Richter, K., Stanke, G., Ulbrich, P., Unger, S., & Wysotzki, F. (1974). Klassifizierungsprozesse bei der Auswertung von Röntgenaufnahmen. In F. Klix, H. Sydow, & F. Wysotzki (Eds.), (p. 157-178). Berlin: VEB Deutscher Verlag der Wissenschaften.
- Bonet, B., & Geffner, H. (1999). Planning as heuristic search: New results. In *Proc. European Conference on Planning (ECP-99)*, Durham, UK. Springer.
- Carbonell, J. (1986). Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In R. Michalski, J. Carbonell, & T. Mitchell (Eds.), *Machine Learning - An Artificial Intelligence Approach* (Vol. 2, p. 371-392). Los Altos, CA: Morgan Kaufmann.
- Card, S. K., Moran, R. P., & Newell, A. (1983). *The psychology of human computer interaction*. Hillsdale, NJ: Erlbaum.
- Chase, W. G., & Simon, H. A. (1973). Perception in chess. *Cognitive Psychology*, 4, 55-81.
- Cooper, R., Fox, J., Farringdon, J., & Shallice, T. (1996). A systematic methodology for cognitive modelling. *Artificial Intelligence*.
- Cooper, R., Yule, P., Fox, J., & Sutton, D. (1998). COGENT: An environment for the development of cognitive models. In U. Schmid, J. F. Krems, & F. Wysotzki (Eds.), *A cognitive science approach to reasoning, learning and discovery* (p. 55-82). Lengerich, Germany: Pabst Science Publishers.
- Davis, R., & King, J. J. (1977). An overview of production systems. In E. Elcock & D. Michie (Eds.), *Machine intelligence* (Vol. 8, pp. 300-332). Chichester, England: Ellis Horwood.
- Falkenhainer, B., Forbus, K., & Gentner, D. (1989). The structure mapping engine: Algorithm and example. *Artificial Intelligence*, 41, 1-63.
- Fikes, R. E., & Nilsson, H. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2, 189-205.
- Gentner, D. (1983). Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, 7, 155-170.
- Görz, G., Rollinger, C.-R., & Schneeberger, J. (Eds.). (2000). *Handbuch der Künstlichen Intelligenz*. München: Oldenbourg.
- Greeno, J. (1974). Hobbits and orcs: Acquisition

- of a sequential concept. *Cognitive Psychology*, 6, 270-292.
- Greeno, J. G. (1978). Natures of problem-solving abilities. In W. K. Estes (Ed.), *Handbook of learning and cognitive processes* (Vol. 5, Human information processing, pp. 239–270). Hillsdale, NJ: Erlbaum.
- Gries, D. (1981). *The science of programming*. New York: Springer.
- Harel, D. (1987). *Algorithmics - the spirit of computing*. Wokingham, Eng.: Addison-Wesley.
- Hummel, J., & Holyoak, K. (1997). Distributed representation of structure: A theory of analogical access and mapping. *Psychological Review*, 104(3), 427-466.
- Kolodner, J. (1993). *Case-based reasoning*. San Mateo, CA: Morgan Kaufmann.
- Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in soar: The anatomy of a general learning mechanism. *Machine Learning*, 1, 11-46.
- Martín, M., & Geffner, H. (2000). Learning generalized policies in planning using concept languages. In *Proc. 7th International Conference on Knowledge Representation and Reasoning (KR 2000)* (p. 667-677). Morgan Kaufmann.
- Minton, S. (1985). Selectively generalizing plans for problem-solving. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI-85)* (pp. 596–599). Morgan Kaufmann.
- Mitchell, T. M. (1997). *Machine learning*. McGraw-Hill.
- Möbus, C. (1988). Zur Modellierung kognitiver Prozesse mit daten- bzw. zielorientierten Regelsystemen. In H. Mandl & H. Spada (Eds.), *Wissenspsychologie*. München: PVU.
- Nerb, J., Ritter, F. E., & Krems, J. (1999). Knowledge level learning and the power law: A SOAR model of skill acquisition in scheduling. *Kognitionswissenschaft*, 8, 20-29.
- Newell, A. (1990). *Unified theories of cognition*. Cambridge, MA: Harvard University Press.
- Newell, A., & Rosenbloom, P. (1981). Mechanisms of skill acquisition and the law of practice. In J. Anderson (Ed.), *Cognitive skills and their acquisition*. Hillsdale, N.J.: Erlbaum.
- Newell, A., & Simon, H. A. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentice Hall.
- Nilsson, N. (1971). *Problem-solving methods in artificial intelligence*. McGraw-Hill.
- Norman, D. (1993). Cognition in the head and in the world: An introduction to the special issue on situated action. *Cognitive Science*, 17, 1-6.
- Novick, L. R., & Holyoak, K. J. (1991). Mathematical problem solving by analogy. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 17(3), 398-415.
- O'Hara, S. (1992). A model of the redescription process in the context of geometric proportional analogy problems. In *Proc. International Workshop on Analogical and Inductive Inference (AII '92)*, Dagstuhl Castle, Germany (Vol. 642, p. 268-293). Springer.
- Opwis, K. (1992). *Kognitive Modellierung – Zur Verwendung wissensbasierter Systeme in der psychologischen Theoriebildung*. Göttingen: Huber.
- Opwis, K., & Plötzner, R. (1996). *Kognitive Psychologie mit dem Computer*. Heidelberg: Spektrum.
- Rumelhart, D. E., & Norman, D. A. (1981). Analogical processes in learning. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition* (p. 335-360). Hillsdale, NJ: Lawrence Erlbaum.
- Russell, S. J., & Norvig, P. (1995). *Artificial intelligence. A modern approach*. Englewood Cliffs, NJ: Prentice-Hall.
- Sacerdoti, E. (1977). *A structure for plans and behavior*. Amsterdam: North-Holland.
- Schädler, K., & Wysotzki, F. (1999). Comparing structures using a Hopfield-style neural network. *Applied Intelligence*, 11, 15-30.
- Schaub, H. (1997). Selbstorganisation in konnektionistischen und hybriden Modellen von Wahrnehmung und Handeln. In G. Schiepek & W. Tschacher (Eds.), *Selbstorganisation in Psychologie und Psychiatrie*. Wiesbaden: Vieweg.
- Schmalhofer, F., & Polson, P. G. A. (1986). A pro-

- duction system model for human problem solving. *Psychological Research*, 48, 113-122.
- Schmid, U., & Kindsmüller, M. (1996). *Kognitive Modellierung. Eine Einführung in die logischen und algorithmischen Grundlagen*. Heidelberg: Spektrum.
- Schmid, U., Wirth, J., & Polkehn, K. (1999). Analogical transfer of non-isomorphic source problems. In *Proc. 21st Annual Meeting of the Cognitive Science Society (CogSci-99), August 19-21, 1999; Simon Fraser University, Vancouver, British Columbia* (p. 631-636). Morgan Kaufmann.
- Schmid, U., & Wysotzki, F. (2000). Applying inductive program synthesis to macro learning. In *Proc. 5th International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2000)* (p. 371-378). AAAI Press.
- Shortliffe, E. H. (1976). *Computer-based medical consultations: Mycin*. New York: Elsevier.
- Strube, G. (2000). Generative theories in cognitive psychology. *Theory & Psychology*, 10, 117-125.
- Strube, G., Habel, C., Hemforth, B., & Konieczny, L. (2000). Cognition. In G. Görz, C.-R. Rollinger, & J. Schneeberger (Eds.), *Handbuch der Künstliche Intelligenz* (p. 19-72). München: Oldenbourg.
- Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning – an introduction*. MIT Press.
- Tergan, S. O. (1986). *Modelle der Wissensrepräsentation als Grundlage qualitativer Wissensdiagnostik*. Opladen: Westdeutscher Verlag.
- Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, 59, 433-460.
- VanLehn, K. (Ed.). (1991). *Architectures for intelligence*. Hillsdale, NJ: Erlbaum.
- VanLehn, K., Jones, R. M., & Chi, M. T. H. (1992). A model of the self-explanation effect. *Journal of the Learning Sciences*, 2, 1-59.
- Veloso, M., Carbonell, J., Pérez, M. A., Borrajo, D., Fink, E., & Blythe, J. (1995). Integrating planning and learning: The Prodigy architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 7(1), 81-120.
- Vorberg, D., & Göbel, R. (1991). Das Lösen rekursiver Programmierprobleme: Rekursionsschemata. *Kognitionswissenschaft*, 1, 83-95.
- Wallach, D. P., & Tack, W. H. (1998). Wissenserwerb und Performanz bei der Regelung komplexer Systeme. *Kognitionswissenschaft*, 7, 118-123.
- Weber, G. (1994). *Fallbasiertes Lernen und Analogien - Unterstützung von Problemlöse- und Lernprozessen in einem adaptiven Lernsystem*. Weinheim: PVU.
- Weber, G. (1996). Episodic learner modeling. *Cognitive Science*, 20, 195-236.
- Winston, P. (1992). *Artificial intelligence, 3rd edition*. Reading, MA: Addison-Wesley.
- Winston, P., & Horn, B. (1989). *Lisp*. Reading, MA: Addison-Wesley.

ACHTUNG

Die Abbildung in der einführenden Textbox *Deep Blue versus Kasparov* ist der New York Times entnommen. Hierfür muss geklärt werden, ob Copyright Ansprüche bestehen. Quelle:

http://www.rci.rutgers.edu/~cfs/472_html/Intro/NYT_Intro/ChessMatch/ComputerDefeats.html