Computermodelle des Denkens und Problemlösens

Ute Schmid*

3. März 2005

1 Einleitung

Der folgende Beitrag hat das Ziel, grundlegende Ansätze und Techniken des Problemlösens und Schlußfolgerns zu vermittlen, wie sie in der Künstlichen Intelligenz (KI)[1] verwendet werden. Dabei konzentriere ich mich auf diejenigen Aspekte, die auch in psychologischen Arbeiten, vor allem im Bereich der kognitiven Modellierung, zur Anwendung kommen. Zudem will ich weitere, zentrale Ansätze der KI darstellen, um Gemeinsamkeiten und Unterschiede der Sichtweisen in KI und Psychologie aufzuzeigen.

KI-Forschung ist seit Begründung der Disziplin 1956 – die Bezeichnung wurde im Sommer 1956 bei einem Workshop am Dartmouth College eingeführt – auf drei Ziele hin orientiert:

1. KI als *ingenieurswissenschaftliche Disziplin* hat das Ziel, intelligente Maschinen zu entwickeln bei denen die dem Verhalten zugrundeliegenden Algorithmen nicht notwendigerweise Ähnlichkeiten zur menschlichen Informationsverarbeitung aufweisen müssen. Allerdings können psychologische Erkenntnisse in die Algorithmen einfliessen oder diese inspirieren. Diese Orientierung, die man als "Psychonik" bezeichnen könnte, wird beispielsweise stark am MIT durch Marvin Minsky vertreten (Minsky, 1975).

2. KI als *formalwissenschaftliche Disziplin* hat das Ziel, allgemeine Sprachen und Methoden zu entwicklen, in denen intelligentes Verhalten beschreibbar oder analysierbar ist. Diese Richtung der KI wird vor allem in Stanford vertreten, wo wesentliche Beiträge zum Schlußfolgern, automatischen Beweisen und Planen erarbeitet wurden, die alle auf formaler Logik basieren. Für diese Richtung der KI steht beispielsweise John McCarthy, der auch die deklarative Programmiersprache Lisp entwickelt hat (McCarthy, 1998).

*Fakultät für Wirtschaftsinformatik und Angewandte Informatik der Otto-Friedrich-Universität Bamberg, 96045 Bamberg

[1]Die Bezeichnung "Künstliche Intelligenz" bzw. *"Artificial Intelligence"* wird inzwischen eher selten verwendet. Stattdessen spricht man von "Intelligenten Systemen". Ich bleibe jedoch bei der geläufigen Kurzbezeichnung "KI".

3. KI als *kognitionswissenschaftliche Disziplin* hat das Ziel, formale, ablauffähige Modelle – sogenannte generative Theorien (Strube, 2000) – der menschlichen Informationsverarbeitung zu erstellen. In diesem Sinne kann KI als "theoretische Psychologie" verstanden werden, die Beiträge zur Beschreibung und Erklärung kognitiver Strukturen und Prozesse sowie Methoden zur Analyse allgemeiner Beschränkungen kognitiver Systeme liefert. Diese Richtung der KI wird beispielsweise an der Carnegie Mellon University vertreten. Als Gründerväter dieser Richtung werden meist Allen Newell und Herbert Simon genannt, die vor allem durch ihre Arbeiten zum Problemlösen (Newell & Simon, 1972) auch starken Einfluss auf die Entwicklung der kognitiven Psychologie genommen haben.

Kognitionswissenschaftlich orientierte KI-Programme werden gemeinhin als "Kognitive Systeme" bezeichnet. Um ein solches System als Modell eines natürlichen kognitiven Systems zu akzeptieren, muss man die Grundannahme akzeptieren, dass Kognition als Berechnungsprozess aufgefasst werden kann. Diese Annahme wird beispielsweise von Newell mit dem Konzept der *physical symbol systems* (Newell, 1980) und von Fodor (1975) mit der Sprache des Geistes (*mentalese*) vertreten (siehe auch Strube, in press). Allerdings ist es nicht unproblematisch, zu entscheiden, ob ein KI-Programm tatsächlich nach ähnlichen Prinzipien arbeitet wie ein natürliches kognitives System. Für kognitive Systeme genügt es nicht, dass ein künstliches System ähnlich *handelt* wie ein Mensch, es muss angebotene Information auf ähnliche Weise verarbeiten wie ein Mensch. Beispielsweise sollten längere Antwortzeiten bei menschlichen Probanden mit größerem Verarbeitungsaufwand beim Computerprogramm korrespondieren, oder das Programm sollte bei verschiedenen Antwortalternativen ähnliche Präferenzen zeigen wie menschliche Probanden (Schlieder, 1999). 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 selben 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 2.2). Diese Systeme erlauben es, Beschränkungen des kognitiven Apparats, wie etwa die Kapazität des Arbeitsgedächtnisses oder Aufmerksamkeitssteuerung, zu berücksichtigen und werden vor allem in der kognitiven Psychologie eingesetzt. In der kognitiv orientierten KI gibt es eine zweite Spielart kognitiver Architekturen, die den Fokus auf die algorithmische Realisierung von allgemeinen kognitiven Fertigkeiten (etwa analoges Schliessen, Planen, Lernen) legt. Während die oben genannten Produktionssysteme gut dafür verwendet werden können, konkrete empirische Daten zu simulieren, wollen die KI-Systeme eher allgemeine psychologische Erkenntnisse abbilden, wie etwa die menschliche Fähigkeit, über verschiedene Problemlöseepisoden hinweg zu generalisieren. Beispiele für solche Systeme sind Prodigy (Veloso et al., 1995) und IPAL (Schmid & Wysotzki, 2000b). Im Gegensatz zu allgemeinen Architekturen stehen *special purpose* Modelle, die zur genaueren Spezifikation eines spezifischen kognitiven Phänomens entwickelt werden. Beispielsweise wurden im Rahmen der Forschung zum räumlichen Schließen (siehe 3.4) Modelle vorgeschlagen, die bei mehreren korrekten Antwortalternativen die von Probanden bevorzugte Alternative erzeugen (Schlieder & Behrendt, 1998).

Alle drei genannten Arten kognitiver Systeme werden insbesondere in den Bereichen Problemlösen und Denken, zum Teil auch für Sprachverstehen und Lernen, eingesetzt. Die beiden erstgenannten Bereiche sind grundlegend für den gesamten Bereich der symbolischen KI. Problemlösen und Denken wurden als zentrale Aspekte der höheren Kognition bereits von Beginn der KI-Forschung an als wesentliche Forschungsfelder identifiziert und bearbeitet. Zudem liefern die in diesen Bereichen entwickelten Techniken – intelligente Suchverfahren sowie logische Inferenz – das methodische Rückgrat für die gesamte KI.

2 Computermodelle des Problemlösens

2.1 Problemlösen als Suche im Problemraum

Das Konzept des Problemlösens als Suche im Problemraum wurde von Newell und Simon (1972) eingeführt. Probleme, die besonders gut zur Beschreibung in diesem Rahmen geeignet sind, zeichnen sich dadurch aus, dass sie aus einer Menge klar definierter Problemzustände bestehen für die eine Menge von Operatoren existiert, mit denen ein Zustand eindeutig in einen anderen Zustand überführt werden kann. Zu dieser Klasse von Problemen zählen beispielsweise das Achter-Puzzle, das Missionare-und-Kannibalen Problem, sowie der Turm von Hanoi (siehe auch Funke & Spering in diesem Band). Solche Probleme werden Transformationsprobleme genannt (Greeno, 1978) und die meisten Problemlöseverfahren der KI sind genau zur maschinellen Lösung von Problemen dieser Art geeignet.

Im folgenden werden zunächst die grundlegenden Konzepte des Problemlösens am Beispiel eines leicht modifizierten “Affe-Banane”-Problems eingeführt.[2] Danach wird

[2]Meist wird zur Illustration der Suche im Problemraum das sogenannte 8er Puzzle (Russell & Norvig, 2002) verwendet. Das Problem ist allerdings bereits so komplex, dass die vollständige Darstellung des Suchraums sehr unübersichtlich wird. Das Affe-Banane-Problem, wie es auch in Schmid (2002) dargestellt wird, ist zum einen hinreichend komplex, um alle wesentlichen Aspekte der Suche im Problemraum darzustellen,

auf Produktionssysteme als spezieller Ansatz zum maschinellen Lösen von Transformationsproblemen eingegangen. Analoges Problemlösen wird als Alternative zum Ansatz des Problemlösens durch Suche eingeführt. Abschließend wird Problemlösen im Zusammenhang mit Lernen und Wissen diskutiert.

2.1.1 Grundbegriffe

Ein *Problem* ist gegeben durch einen *Anfangszustand*, ein *Problemlöseziel* und eine Menge von *Aktionen* oder *Problemlöseoperatoren* (Nilsson, 1971). Eine *Problemlösung* besteht dann darin, eine Folge von Aktionen zu finden, um vom Anfangszustand in einen Zustand zu gelangen, in dem das Problemlöseziel erreicht ist. Im einfachsten Fall werden zur Transformation eines Zustands in einen anderen (Nachfolge-)zustand Aktionen direkt als Übergangsfunktion auf den Zuständen definiert. Eine allgemeinere und elegantere Art das Handlungsrepertoire eines Problemlöse-Agenten zu beschreiben ist die Definition von Operatoren. Operatoren sind allgemeiner formulierte Handlungsschemata, die für konkrete Zustände zu Operatoranwendungen instantiiert werden können. In Tab. 1 werden beide Varianten für das Affe-Banane-Problem dargestellt.

Ein Problemzustand wird als Beschreibung einer aktuellen Situation repräsentiert. Beispielsweise könnte eine Situation darin bestehen, dass ein Affe sich in einem Raum befindet, in dem für ihn vom Boden aus unerreichbar eine Banane an der Decke befestigt ist. Außerdem befindet sich in dem Raum eine Kiste (siehe Abb. 1). Das Problemlöseziel des Affen ist es, die Banane zu erhalten. Der Affe kann das Ziel erreichen – also das Problem lösen –, indem er der Reihe nach folgende Aktionen anwendet: zur Kiste gehen, die Kiste zu einer Position unterhalb der Banane schieben, auf die Kiste steigen und die Banane greifen. Der dadurch erreichte Zustand enthält das Problemlöseziel, ist also ein Zielzustand.

Damit es sich tatsächlich um ein Problem handelt, darf die Lösung nicht trivial erreichbar sein: Hätte der Affe nur eine einzige Aktion zur Verfügung (er kann nach Bananen geifen) und könnte das Ziel durch einmalige Anwendung dieser Aktion erreichen (er steht bereits auf der Kiste unter den Bananen), würde man also nicht von Problemlösen sprechen. 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 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* (siehe Abschnitt 2.2).

Damit ein Problem maschinell lösbar ist, müssen Zustände und Operatoren formal beschrieben werden. Beispiele für mögliche formale Repräsentationen des Affe-

zum anderen aber einfach genug, um vollständig behandelt zu werden.

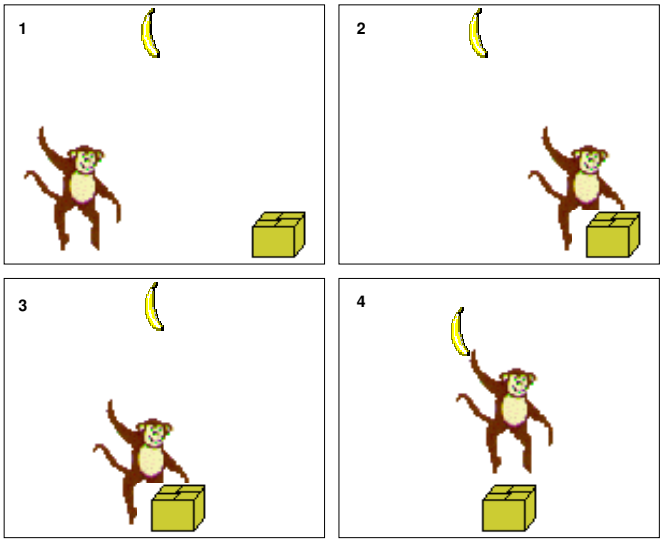


Abbildung 1: Illustration des Affe-Banane-Problems.

Banane-Problems sind in Tab. 1 gegeben.

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, werden nur einige relevante Positionen im Raum festgelegt, etwa die Raumpositionen “Links”, “Rechts” und “Mitte”.

In Tab. 1.a werden Zustände als Vektoren aus vier Komponenten repräsentiert. Der erste Eintrag gibt die Position des Affen im Raum an, der zweite Eintrag die Position der Kiste. An der dritten Stelle ist vermerkt, ob der Affe auf dem Boden (b) oder auf der Kiste (k) steht, und an letzter Stelle ob der Affe die Banane hat (ja oder nein). Das Handlungsrepertoire des Affen wird in Form von Aktionen – als Zustandsübergangsfunktion – beschrieben: Für jeden möglichen Zustand wird ein zulässiger Folgezustand angegeben. Die Aktionen entsprechen dem Pfeil zwischen Ausgangs- und Folgezustand. Beispielsweise beschreibt der erste Eintrag in der Tabelle die Aktion “gehe von links in die Mitte”. Wie man sieht, wird diese Art der Beschreibung von Aktionen schnell sehr aufwändig und unübersichtlich.

Eine elegantere Art der Darstellung ist in Tab. 1.b gegeben. Hier werden Zustände durch Propositionen beschrieben, speziell durch konjunktiv (mit “und”) verknüpfte Prädikate (Russell & Norvig, 2002). Zur Beschreibung des Affe-Banane-Problems werden das zweistellige Prädikat *pos(x, y)*, sowie die nullstelligen Prädikate *auf-boden* beziehungsweise *auf-kiste* und *hat-banane* verwendet. Der Anfangszustand wird repräsentiert durch die Aussage “Die Position des Affen ist links *und* der Affe steht auf

Tabelle 1: Formale Darstellungen des Affe-Banane-Problems (Erläuterung siehe Text).

(a) mit Aktionen

Anfangszustand: (links,rechts,boden,nein)

Ziel: (_, _, _, ja)

Bewegung im Raum:
(links,links,boden,nein) → (mitte,links,boden,nein)
(links,links,boden,nein) → (rechts,links,boden,nein)
(mitte,links,boden,nein) → (links,links,boden,nein)
(mitte,links,boden,nein) → (rechts,links,boden,nein)
(rechts,links,boden,nein) → (links,links,boden,nein)
(rechts,links,boden,nein) → (mitte,links,boden,nein)
(links,mitte,boden,nein) → (mitte,mitte,boden,nein)
(links,mitte,boden,nein) → (rechts,mitte,boden,nein)
(mitte,mitte,boden,nein) → (links,mitte,boden,nein)
(mitte,mitte,boden,nein) → (rechts,mitte,boden,nein)
(rechts,mitte,boden,nein) → (links,mitte,boden,nein)
(rechts,mitte,boden,nein) → (mitte,mitte,boden,nein)
(links,rechts,boden,nein) → (mitte,rechts,boden,nein)
(links,rechts,boden,nein) → (rechts,rechts,boden,nein)
(mitte,rechts,boden,nein) → (links,rechts,boden,nein)
(mitte,rechts,boden,nein) → (rechts,rechts,boden,nein)
(rechts,rechts,boden,nein) → (links,rechts,boden,nein)
(rechts,rechts,boden,nein) → (mitte,rechts,boden,nein)

Schieben der Kiste:
(links,links,boden,nein) → (mitte,mitte,boden,nein)
(links,links,boden,nein) → (rechts,rechts,boden,nein)
(mitte,mitte,boden,nein) → (links,links,boden,nein)
(mitte,mitte,boden,nein) → (rechts,rechts,boden,nein)
(rechts,rechts,boden,nein) → (links,links,boden,nein)
(rechts,rechts,boden,nein) → (mitte,mitte,boden,nein)

Steigen auf Kiste:
(links,links,boden,nein) → (links,links,kiste,nein)
(mitte,mitte,boden,nein) → (mitte,mitte,kiste,nein)
(rechts,rechts,boden,nein) → (rechts,rechts,kiste,nein)

Greifen der Banane:
(mitte,mitte,kiste,nein) → (mitte,mitte,kiste,ja)

(b) mit Problemlöseoperatoren

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

Ziel: 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

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. 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.

Anders als bei der ersten Variante in Tab. 1.a werden die Aktionen nicht direkt, sondern schematisch angegeben. Die Problemlöseoperatoren werden im Allgemeinen als bedingte Regeln der Form

WENN ⟨*Bedingung*⟩ *DANN* ⟨*Aktion*⟩

repräsentiert. Solche Regeln werden auch *Produktionsregeln* oder Produktionen genannt (siehe 2.2). 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 Tab. 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. 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 entspricht der Operator einer Aktion.

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, siehe Abschnitt 2.1.5) 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. 2 gegeben.

Während die Problemrepräsentation in Tab. 1.b kompakter ist als diejenige in Tab. 1.a, wird die maschinelle Lösung des Problems komplizierter, da nun in jedem Schritt Variablen entsprechend dem aktuell gegebenen Zustand belegt werden müssen. In den ursprünglichen Ansätzen zum Problemlösen, zum Beispiel beim *General Problem Solver* (Newell & Simon, 1972) wurde meist die erste Variante verfolgt, die mei-

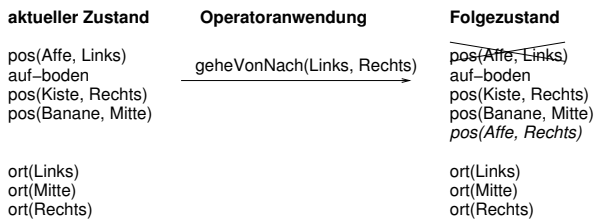


Abbildung 2: Zustandstransformation durch Operatoranwendung.

sten aktuellen Ansätze zur Handlungsplanung (siehe Abschnitt 2.1.5) basieren dagegen auf dem zweiten Prinzip.

2.1.2 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. 3 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 herunter steigen 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),

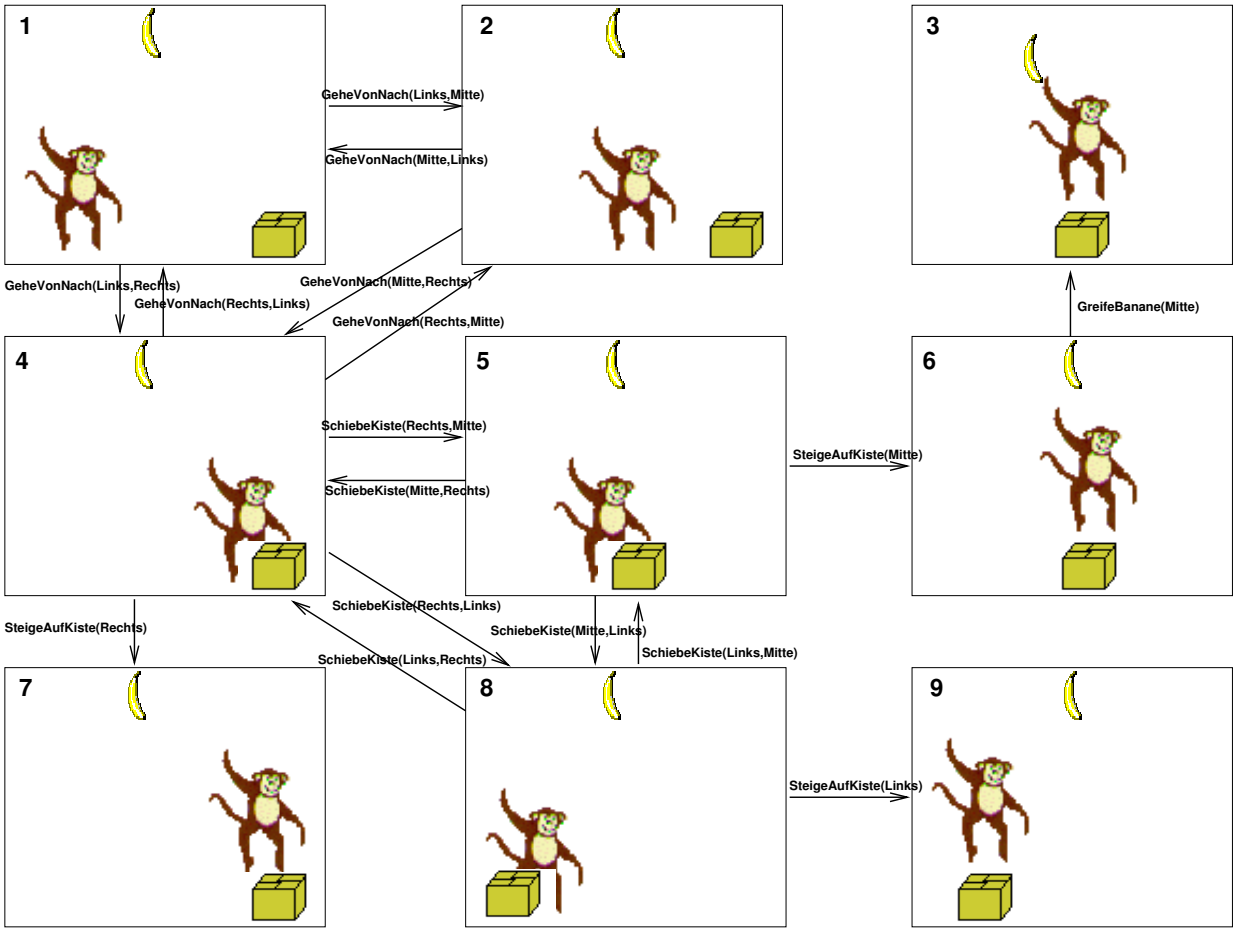


Abbildung 3: Problemraum für das Affe-Banane-Problem mit 1 = Startzustand und 3 = Zielzustand. Der optimale Pfad ist $1 - 4 - 5 - 6 - 3$.

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.

2.1.3 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.

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. 3) ist in Abb. 4 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. 4 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, ausprobiert. 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.

Ordnung auf den Operatoren:
(siehe Tab. 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.

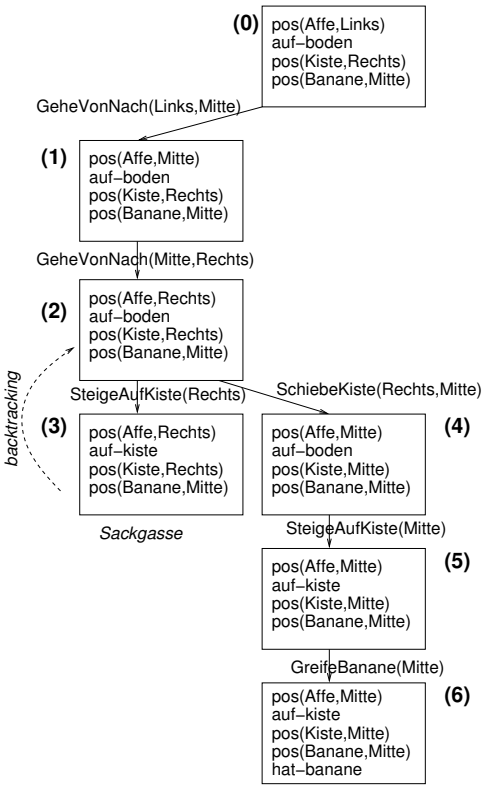


Abbildung 4: Durch Tiefensuche erzeugter Suchbaum für das Affe-Banane-Problem.

In einen Zyklus wäre der Affe beispielsweise gelaufen, wenn er in Zustand 3 in Abb. 4 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 ebenenweisen Aufbau des Suchbaums wird diejenige 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.

Hill Climbing und Bewertungsfunktionen Bei den blinden Suchstrategien werden ausgehend vom Anfangszustand Folgezustände geniert, 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, werden 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. 3) 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. 5): 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. 6). Greeno (1974) konnte zeigen, dass Probanden vor allem Schwierigkeiten mit dem Übergang von Zustand (6) zu Zustand (7) in Abb. 6 haben –

Abbildung 5: Das Problem lokaler Maxima beim *Hill Climbing*.

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).

Lösungsweg:

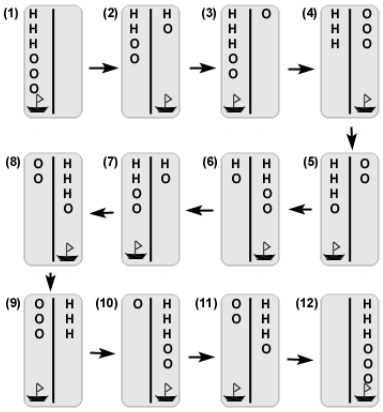


Abbildung 6: Das "*Hobbits and Orcs*"-Problem.

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 Produktionsystem (siehe 2.2) schlagen Schmalhofer und Polson (1986) vor.

Damit eine Heuristik von einem computer-implementierten 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. 7 dargestellten Achter-Puzzle (Nilsson, 1971) wären es die Anzahl von Plättchen, die bereits korrekt plaziert 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

Abbildung 7: Das Achter-Puzzle.

die Plättchen in die korrekte Reihenfolge gebracht werden als dass einzelne Plättchen schnellstmöglich an ihre Zielposition befördert werden (Nilsson, 1971). Der vor einigen Jahren durch seinen Sieg über den Schachmeister Kasparov bekannt gewordene Schachcomputer *Deep Blue* arbeitet mit einer von Schachexperten entwickelten sehr ausgefeilten Bewertungsfunktion für Spielpositionen.

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. 3) würde wie in Tab. 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. 8 veranschaulicht. Zunächst wird das globale Ziel *hat-banane* auf den *Stack* gepackt. Da das Ziel nicht direkt erreichbar ist, werden die Teilziele *auf-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 *GeheVonNach(Links, Rechts)* erfüllt werden. Damit kann das Teilziel kann vom Stack entfernt werden. Die entsprechende Operation heißt *pop*.

Der Algorithmus zur Realisierung der Mittel-Ziel-Analyse ist in Tab. 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 ei-

Tabelle 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.

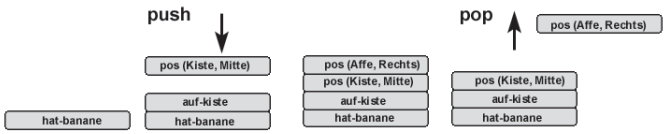


Abbildung 8: Verarbeiten von Teilzielen mit einem *Stack*.

Tabelle 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.

nem 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. 6). Erfüllt der aktuelle Zustand das Ziel, wird die Suche erfolgreich beendet, anderenfalls 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 Abschnitt 2.1.4) 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 rekursive Struktur der Regeln wird der Aufbau und die Abarbeitung des *Stack* organisiert (siehe Abb. 8 und Tab. 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 *Schiebe-Kiste(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

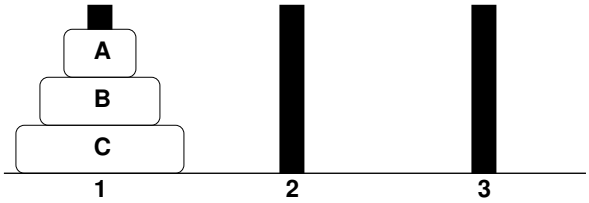


Abbildung 9: Turm-von-Hanoi-Problem mit drei Scheiben.

Aufruf von *Reduziere* endgültig verlassen und das Teilziel *auf-kiste* vom *Stack* entfernt.

Ein Problem der Mittel-Ziel-Analyse ist, dass Probleme mit voneinander abhängigen Teilzielen nicht mit dieser Strategie lösbar sind (siehe Abschnitt 2.1.5). 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 2.1.4). 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.

2.1.4 Rekursive Probleme – Der Turm von Hanoi

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 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. 9 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. 10 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 folgen-

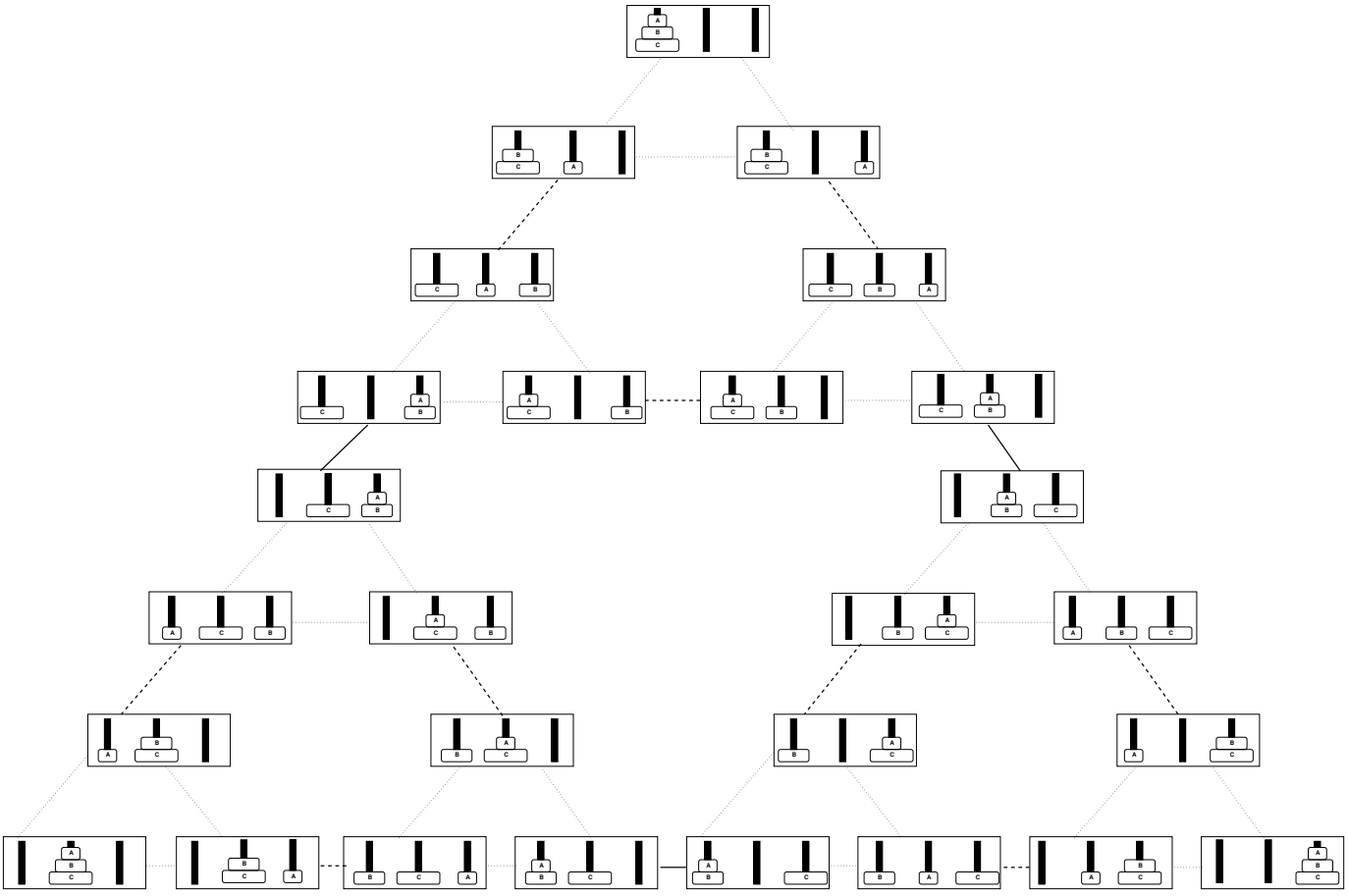
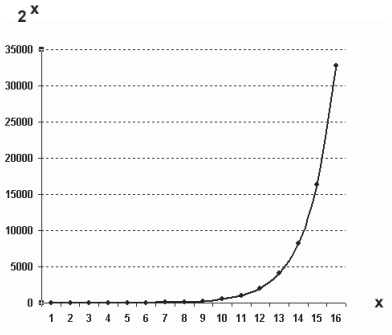


Abbildung 10: Problemraum des Turm-von-Hanoi-Problems mit drei Scheiben (Bewegungen von *A* gepunktet, Bewegungen von *B* gestrichelt, Bewegungen von *C* durchgezogen).



Scheiben	Zustände	Aktionen
1	3	1
2	9	3
3	27	7
4	81	15
5	243	31
6	729	63
7	2187	127
8	6561	255
9	19683	511
10	59049	1023

Abbildung 11: Exponentielles Wachstum beim Turm-von-Hanoi.

dermaßen 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 Scheibe *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 $2^0 + 2^1 + 2^2 + 2^3 + \ldots 2^{n-1}$ Bewegungen für n Scheiben ausgeführt werden (Schmid & Kindsmüller, 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. 11 dargestellt. Viele Probleme, die in der KI bearbeitet werden, haben ein solches exponentielles Wachstumsverhalten. Man spricht hier auch von nicht effizient berechenbaren Problemen.

Die Arbeitsweise der Mittel-Ziel-Analyse für das Turm-von-Hanoi-Problem (siehe 2.1.4) ist in Tab. 4 dargestellt.

Betrachtet man das Turm-von-Hanoi-Problem analytisch, so kann die optimale Abfolge von Aktionen nach folgender rekursiven Vorschrift 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

Tabelle 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, C 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 auf 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
```

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. 12 veranschaulicht.

Die rekursive Lösungsstruktur des Turm-von-Hanoi-Problems spiegelt sich auch im Problemraum (Abb. 10) wider: Der Problemraum des Ein-Scheiben-Problems ist neunmal 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 Über-

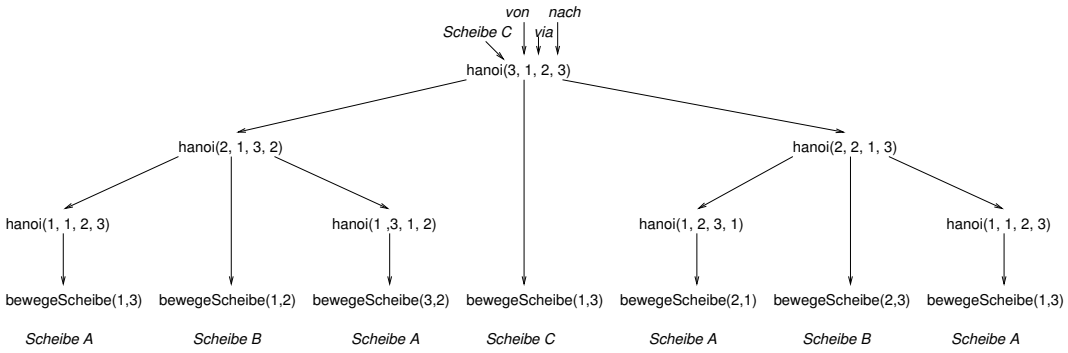


Abbildung 12: Abarbeitung der rekursiven Funktion *hanoi* für drei Scheiben.

führung eines Anfangszustands in einen Zielzustand ermitteln.

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:

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

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).

2.1.5 Problemlösen und Planen

In der KI wird zwischen Problemlösen und Planen unterschieden (Russell & Norvig, 2002). 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) oder FF (Hoffmann & Nebel, 2001), verfügen über einen Mechanismus zur automatischen Ermittlung einer Bewertungsfunktion für ein beliebiges gegebenes Problem.

Planungsalgorithmen sind im Gegensatz zu Problemlöse-Algorithmen 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. 1.b 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, wie in Tab. 1.a) 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 2.1.3), das Problem, nicht mit abhängigen Teilzielen umgehen zu können. Dieses Problem kann durch die sogenannte Sussman-Anomalie (siehe Abb. 13) verdeutlicht werden: Um die Ziele *auf(A, B)* **UND** *auf(B, C)* von dem gegebenen Zustand aus zu erreichen, könnte *B* unmittelbar auf *C* gestellt werden. Damit wäre eines der Teilziele erreicht, aber um *auf(A,B)* zu erreichen, müsste der Turm komplett wieder abgebaut werden. Andererseits könnte man zunächst *C* 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 *C* auf den Tisch gestellt wurde, zunächst *B* auf *C* und

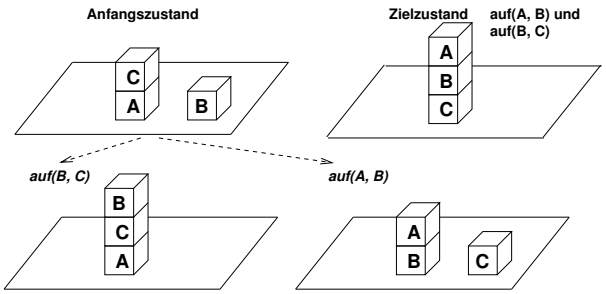


Abbildung 13: Die Sussman-Anomalie.

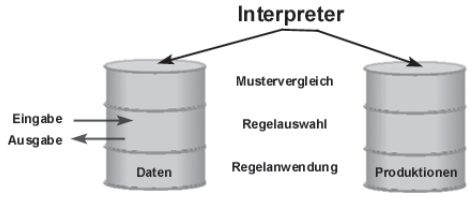


Abbildung 14: Architektur eines Produktionssystems.

dann *A* auf *B* gesetzt wird. Hierbei erfolgt ein Fokuswechsel von Ziel *auf(A, B)* auf Ziel *auf(B, C)*: Um das Ziel *auf(A, B)* zu erfüllen, muß zunächst *C* auf den Tisch gelegt werden. Dadurch wird *A* frei und damit Operator "Stelle A auf B" anwendbar. 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. 8), sondern als Menge verwaltet werden. Das erste Planungssystem, das mit solchen Abhängkeiten umgehen konnte, war NOAH (Sacerdoti, 1977). Alle modernen Planer arbeiten mit einer solchen nicht-linearen Strategie, bei der Teilziele "verschränkt" abgearbeitet werden können.

2.2 Produktionssysteme

Ein System, das die Anwendung von (bedingten) Regeln, *Produktionsregeln* oder Produktionen genannt, auf Daten steuert, heißt Produktionssystem (siehe Abb. 14). Die aktuell im Arbeitsspeicher befindlichen Daten können beispielsweise die Beschreibung eines Problemzustands sein und die gespeicherten Regeln Problemlöseoperatoren der Form "Wenn ⟨Bedingung⟩ Dann ⟨Aktion⟩". 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.

Gegeben ist eine Kaffee-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.

Abbildung 15: Lösung des "Kaffee-Dose"-Problems mit einem Produktionssystem.

Die Transformation der Daten durch Anwendung 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 und Horn (1989).

Ein sehr einfaches Produktionssystem für das "Kaffee-Dose" (*coffee can*) Problem (Gries, 1981) ist in Abb. 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⟩.

2.2.1 Mustervergleich

Im einfachsten Fall, wie bei dem Produktionssystem in Abb. 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. 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. 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.

2.2.2 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. 4 und Abb. 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 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 Breitensuche, während alle anderen Strategien eine Tiefensuche definieren (siehe 2.1.3).

2.2.3 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 2.1.1).

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 2.1.3) 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).

2.2.4 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 2.1.4) in ACT-R ist in Tab. 5 und Tab. 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

Tabelle 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))
```

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 2.2.3). Im Bedingungsteil der Regel wird jeweils das 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 2.1.3).

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 Musterverleich 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 Effekt erzielt, und einen Wert für die Kosten der Anwendung. Die Parameter

Tabelle 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!
)
```

werden aufgrund von Problemlöseerfahrungen verändert (siehe 2.4). ACT-R verfügt über einen Lernmechanismus zum Erwerb neuer Produktionsregeln, der auf Analogiebildung (siehe 2.3) basiert.

2.2.5 Weitere Kognitive Architekturen

Neben dem sicher am verbreitetsten System ACT existieren eine Reihe weiterer kognitiver Architekturen: Ein System, das als direkter Nachfolger des General Problem Solver im Umfeld der Forschergruppe um Newell (Newell, 1990) entwickelt wurde, ist SOAR (*State, Operator And Result*). SOAR ist wie ACT ein zielgesteuertes Produktionssystem. Lernen wird als *chunking* von Produktionsregeln modelliert. SOAR wird seit einiger Zeit eher in kommerziellen Bereichen, etwa zur Simulation von Flugzeugsteuerungen, eingesetzt.

Eine in Deutschland entwickelte kognitive Architektur ist das PSI-System von Dörner (2002). PSI ist ein auf einer Theorie des Handelns basierendes System, das die Interaktion eines Agenten mit einer komplexen Umgebung realisiert und motivationale und emotionale Aspekte berücksichtigt. Derzeit wird PSI auf die Interaktion mehrerer Agenten erweitert.

Das System COGENT (Cooper, Yule, Fox, & Sutton, 1998) ist ein graphisches Werkzeug für Entwurf und Analyse kognitiver Modelle. Einen Überblick über diese und weitere kognitive Modelle des Problemlösens und Lernens gibt Opwis (1992).

2.3 Analoges Problemlösen

Bisher wurde dargestellt, wie Problemlösen durch Suche im Problemraum modelliert werden kann. Der Problemlöser kann dabei auf Allgemeinwissen (siehe ACT-R, 2.2.4) zurückgreifen und Heuristiken benutzen, die ihm helfen, die Suche in Richtung des Zielzustands zu steuern (siehe Abschnitt 2.1.3). 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*.

Es gibt zahlreiche Systeme zur Modellierung von analogem Problemlösen, sowohl in der kognitiven Psychologie als auch in der KI. Eines der prominentesten Systeme ist die *Structure Mapping Engine* (SME) (Falkenhainer, Forbus, & Gentner, 1989), 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 z.B. Gentner, Ratterman, & Forbus, 1993).

2.3.1 Gentners Theorie des Strukturvergleichs

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

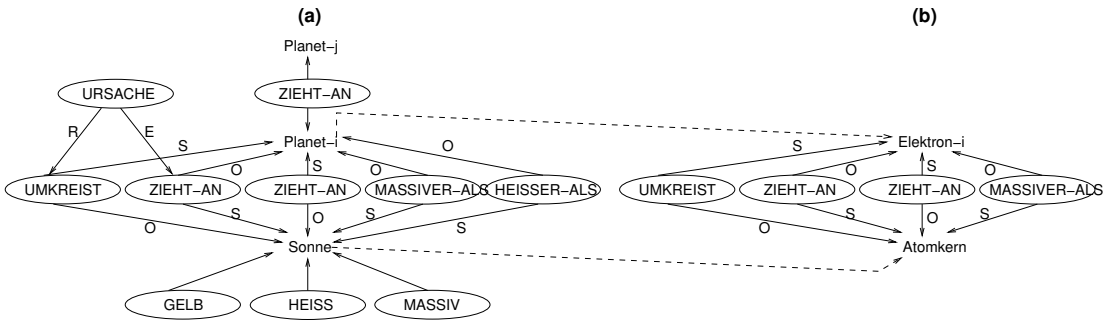


Abbildung 16: Repräsentation eines Ausschnitts von Wissen über ein Sonnensystem (a) und ein Atom (b) als Struktur aus Objekten und Relationen zusammen mit einer strukturerhaltenden Abbildung für die Rutherford-Analogie “Das Atom ist wie das Sonnensystem” (nach Gentner, 1983).

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 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 strukturerhaltende Abbildungen als Homomorphismen bezeichnet (Schmid, Wirth, & Polkehn, 1999). Beispielsweise ist dem Rutherfordschen Atommodell die Analogie “Das Wasserstoffatom ist wie unser Sonnensystem aufgebaut” zugrundegelegt (siehe Abb. 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 Be-

reich 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), außer acht 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 *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.

2.3.2 Die *Structure Mapping Engine*

Die *Strukture Mapping Engine* (SME) von Falkenhainer et al. (1989) realisiert die Annahmen der Theorie des Strukturvergleichs auf folgende Weise in einem Computer-

modell:

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 (Jain & Wysotzki, 2002).

Eine Veranschaulichung der Arbeitsweise des Algorithmus ist für die Rutherford-Analogie (siehe Abb. 16) in Tab. 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.

2.3.3 Weitere Modelle des Analogen Problemlösens

Neben der SME wurden weitere kognitive Computermodelle des analogen Problemlösens und Schließens entwickelt: Bekannt wurde vor allem ACME (*Analogical Constraint Mapping Engine*, Holyoak & Thagard, 1989) und das Nachfolgesystem LISA (Hummel & Holyoak, 1997). Bei diesen Systemen wird vor allem die Rolle von semantischen und pragmatischen Constraints beim Mapping betont. Weitere Systeme sind IAM (Keane, Ledgeway, & Duff, 1994) sowie AMBR (Kokinov & Petrov, 2000). CASCADE (VanLehn, Jones, & Chi, 1992) ist ein Modell des Problemlösens und Lernens, das auf der Selbsterklärung von Musterlösungen und analogem Transfer von Lösungen basiert. Auch im System ACT-R wird versucht, analoges Lernen aus Problemlöseerfahrung zu modellieren. In einer Variante des Systems PUPS (Anderson & Thompson, 1989) wird gezeigt, wie Programmieraufgaben durch analogen Transfer von in Schemata repräsentierten Lösungen gelöst werden können.

In der KI gibt es ebenfalls zahlreiche Analogie-Systeme. Das älteste System ist *Analogy* (Evans, 1968), das in der Lage ist geometrische Intelligenztest-Aufgaben zu lösen (siehe Abb. 17). Solche Analogien der Form "*a : b* verhält sich wie "*c : ?*" werden auch Proportionalanalogien genannt. Diese Klasse von analogen Problemen werden auch in den neueren KI-Systemen PAN (O'Hara, 1992) und Copycat (Hofstadter & The Fluid Analogies Research Group, 1995) behandelt. Ein KI-Planungssystem, das verschiedene Mechanismen des Erwerbs von Kontrollwissen (Regeln zur Steuerung der Suche nach einem Plan) sowie analoges Schließen integriert, ist PRODIGY (Veloso et al., 1995). Anders, als die oben genannten kognitiven Systeme, bei denen eine bekannte Lösung auf ein neues Problem transformiert wird, wird hier eine bekannte

Tabelle 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-i
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)) –	
***Inferenz:* ursache(zieht-an(Atomkern, Elektron-i), umkreist(Elektron-i, Atomkern))**	

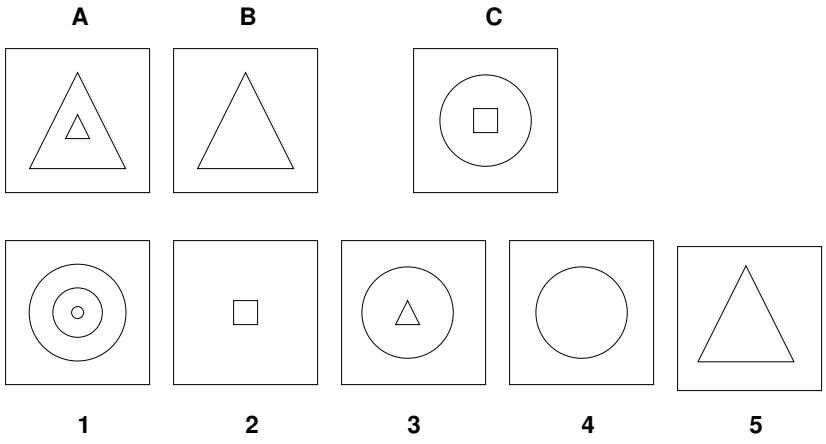


Abbildung 17: Eine Analogie-Aufgabe für das System *Analogy*.

Problemlöse-Episode im Kontext des neuen Problems “abgespielt”. Für diesen alternativen Mechanismus zum analogen Problemlösen wurde der Begriff “derivationale Analogie” geprägt (Carbonell, 1986). 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 (Aamodt & Plaza, 1994; Kolodner, 1993).

2.4 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 2.2.4). 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.

2.4.1 Verstärkung und Kombination von Regeln

Im Produktionssystem ACT-R (siehe 2.2.4) 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 2.2). 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 Ler-*

nens (*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 2.1.3) 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 alternative 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. 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”.

2.4.2 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 2.1.4) zunächst die korrekte Folge von Scheibenbewegungen durch heuristische Suche erzeugen, aber bereits nach einigen Durchläufen die in Abschnitt 2.1.4 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 (2000a).

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

2.5 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 2.2.4), sowie bereichsspezifische Strategien. In vielen Fällen ist er deshalb nicht auf die Suche nach einem Lösungsweg mithilfe schwacher Heuristiken (vergleiche Abschnitt 2.1.3) 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 2.2.4) 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 das bereits erwähnte System *Deep Blue* sind ebenfalls ein Beispiel für Expertensysteme. 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.

Expertensysteme basieren im Wesentlichen auf formal repräsentiertem Wissen über einen Bereich und sind in der Lage, mit Hilfe von Deduktionsregeln Schlüsse aus diesem Wissen abzuleiten. Die grundlegenden Techniken hierfür werden im nächsten Abschnitt dargestellt.

3 Computermodelle des Denkens

3.1 Inferenzprozesse

Schlußfolgerndes Denken oder Inferenz bezeichnet einen Prozess, bei dem aus gegebenen Fakten und Regeln Schlüsse gezogen werden. Ein System, was zum schlussfolgernden Denken in der Lage sein soll, benötigt eine Wissensbasis und einen Inferenzmechanismus. In der Wissensbasis wird das Wissen über einen Gegenstandsbereich gespeichert. Der Inferenzmechanismus greift darauf zurück, um Schlußfolgerungen zu ziehen. Während das Wissen je nach Anwendung immer wieder neu in die Wissensbasis eingebracht – erfasst und repräsentiert – werden muss, sind die Inferenzregeln allgemeingültig. In der KI wird üblicherweise mit *logischen Formalismen* gearbeitet, um Inferenzprozesse auf dem Rechner zu modellieren.

Ein bekanntes Beispiel für einen logischen Schluss ist der in Tab. 8 dargestellte Syllogismus (siehe auch Knauff, in diesem Band). Links in der Tabelle ist der Syllogismus umgangssprachlich dargestellt, daneben in Form einer logischen Repräsentation. Geht man von der Annahme aus, dass die Behauptung, dass alle Menschen sterblich sind, wahr ist, und erhält man ausserdem die Information, dass Sokrates ein Mensch ist, so kann man folgern, dass Sokrates sterblich ist. Der angewendete Inferenzmechanismus ist hier eine einzige Inferenzregel, der sogenannte *modus ponens*.

Der *modus ponens* ist eine allgemeingültige Schlussregel, das heißt, auf zwei beliebige Aussagen der obigen Form angewendet ist die erzeugte Schlussfolgerung immer korrekt. Das heißt, logische Schlüsse sind wahrheitserhaltend. Wie solche logischen

Tabelle 8: Syllogismus als Beispiel für einen logischen Schluss.

Alle Menschen sind sterblich.	$\forall$ x Mensch(x) $\rightarrow$ Sterblich(x)	Axiom/Prämisse
Sokrates ist ein Mensch.	Mensch(Sokrates)	Fakt/Beobachtung
Also ist Sokrates sterblich.	sterblich(Sokrates)	Schluss

Tabelle 9: Induktive und abduktive Inferenz.

(a) Induktion	
Sokrates ist ein Mensch.	Hintergrundwissen
Sokrates ist sterblich.	Beobachtung
Alle Menschen sind sterblich.	Hypothese
(b) Abduktion	
Alle Menschen sind sterblich.	Theorie
Sokrates ist sterblich.	Beobachtung
Sokrates ist ein Mensch.	Diagnose

Schlüsse mechanisch ausgeführt werden können, wird in Abschnitt 3.2 gezeigt. Die Wahrheit der Aussagen, auf die die Schlussregel angewendet wird, kann dagegen nicht innerhalb eines logischen Systems geprüft werden. Die Schlussregel würde beispielsweise aus den Aussagen “Alle Dozenten sind gute Tänzer.” und “Peter ist ein Dozent.” ableiten, dass Peter ein guter Tänzer ist. Diese Ableitung ist nur dann wahr, wenn beide Aussagen wahr sind.

Das bisher betrachtete Beispiel illustriert sogenannte deduktive Inferenz. *Deduktion* bezeichnet *logische* Inferenz, also den Fall, bei dem korrekte Schlüsse aus vorhandenem Wissen gezogen werden. Dabei sind die Schlüsse stets *spezieller* als die vorgegebenen Axiome (Schöning, 1992). Deduktion produziert also kein “neues” Wissen, sondern hilft, bereits Bekanntes zu explizieren. Eine andere Art der Inferenz ist die *Induktion* (siehe Tab. 9.a). Hier wird aus Hintergrundwissen und Beobachtungen (Beispielen) eine Hypothese über einen generalisierten Sachverhalt aufgebaut. Mit Induktion wird also neues, aber ungesichertes Wissen erzeugt. Induktion ist deshalb der zentrale Inferenzmechanismus beim maschinellen Lernen (Mitchell, 1997). Natürlich wäre es höchst gewagt, aus nur einer einzigen Beobachtung bereits eine generalisierte Hypothese aufzubauen, wie dies im Beispiel in Tab. 9 getan wird.

Eine dritte Form der Inferenz ist die *Abduktion* (siehe Tab. 9.b), wo aus einer gegebenen Theorie und beobachteten Sachverhalten auf Prämissen geschlossen wird. Diese Art des Schließens ist typisch für Diagnosesysteme, etwa im Bereich der Medizin. Beispielsweise kann ein Arzt aus seinem theoretischen Wissen, dass bei Grippe Fieber und Kopfschmerzen vorliegen, bei einem Patienten, der über diese Symptome klagt, schlie-

ßen, dass der Patient an Grippe erkrankt ist. Auch mit abduktiver Inferenz können keine "wahren" Diagnosen abgeleitet werden. Es existiert keine allgemeingültige logische Schlussregel, mit der aus "$A \rightarrow B$" (lies "A impliziert B") und dem Vorliegen von B die Gültigkeit der Prämisse A gefolgert werden kann!

In der Psychologie wie in der KI wird unter schlussfolgerndem Denken insbesondere deduktives Schliessen betrachtet. Im Rest des Kapitels wird deshalb speziell auf diesen Aspekt der Inferenz eingegangen.

3.2 Grundlagen der logischen Deduktion

In der Psychologie gibt es zahlreiche Belege dafür, dass Menschen ihre Schlüsse häufig nicht nach den Gesetzen der logischen Deduktion ziehen. Beispiele sind das Umdrehen einer irrelevanten Karte bei der Wason-Aufgabe (Wason & Johnson-Laird, 1972) oder die Bevorzugung einer Konjunktion von Fakten gegenüber einem einzelnen dieser Fakten als Schlussfolgerung (Tversky & Kahneman, 1983) (siehe auch Knauff, in diesem Band). Die Kenntnis von Grundlagen der deduktiven Logik ist aber dennoch auch für Psychologen sinnvoll, da logische Kalküle als normatives Modell für korrektes logisches Schliessen herangezogen werden können. Zudem basieren Repräsentationsformalismen, die auch in der Psychologie Verwendung finden – wie semantische Netze oder Schema-Hierarchien – , auf logischen Grundlagen.

Um Inferenzen automatisch, auf einer Maschine, ablaufen lassen zu können, benötigt man einen sogenannten *Kalkül*. Ein Kalkül ist eine Menge syntaktischer Transformationsregeln, die für eine bestimmte Sprache definiert sind. Das in Abschnitt 2.2 dargestellte Kaffee-Dose-Problem (Abb. 15) ist ein Beispiel für ein sehr einfaches Kalkül: Die Sprache besteht aus allen möglichen Abfolgen der Buchstaben *S* und *W*. Die Transformationsregeln ersetzen Buchstabenmuster durch andere Buchstabenmuster. Dies geschieht rein syntaktisch, ohne Berücksichtigung der Bedeutung der Buchstabenfolgen. Für die Automatisierung logischer Schlüsse ist die Sprache beispielsweise Aussagenlogik oder Prädikatenlogik erster Stufe. Ein bekanntes Kalkül ist das *Resolutionskalkül* (Robinson, 1965), das nur auf einer einzigen Regel, der Resolutionsregel, basiert.

3.2.1 Logische Repräsentation

Eine Logik ist eine formale Sprache. Wie bei einer natürlichen Sprache wird festgelegt, was die syntaktisch korrekten Ausdrücke sind, die zu der Sprache gehören. Die Menge aller syntaktisch korrekten Ausdrücke definiert die wohlgeformten Formeln (analog zu syntaktisch korrekten Sätzen einer natürlichen Sprache). Wesentlich ist, dass den Ausdrücken *Bedeutung* zugewiesen werden kann. Zu einer Logik gehört also auch eine Menge von bedeutungszuweisenden Regeln. Schließlich muß für eine Logik ein Beweiskalkül existieren, mit dem aus gegebenen Aussagen neue, wahre Aussagen abgeleitet werden können. Zwei der bekanntesten und am meisten eingesetzten Logiken sind Aussagenlogik (*propositional logic*) und Prädikatenlogik erster Stufe.

Aussagenlogik Die atomaren syntaktischen Bausteine der Aussagenlogik sind Aussagen. Mithilfe einer Menge von Junktoren (*und*, *oder*, *impliziert*, *genau dann wenn*)

Tabelle 10: Syntax der Aussagenlogik.

- $True$ und $False$ sowie Symbole $\{$P, Q, R , . . . $\}$ sind Formeln der Aussagenlogik. (atomare Formeln)
- Wenn P und Q aussagenlogische Formeln sind, dann auch
  $\neg P$ (nicht P),
  $P \wedge Q$ (P und Q), $P \vee Q$ (P oder Q)
  $P \rightarrow Q$ (P impliziert Q) $P \leftrightarrow Q$ (P genau dann wenn Q).
- Das sind alle aussagenlogischen Formeln.

Tabelle 11: Definition der Bedeutung logischer Junktoren.

P	Q	$\neg$ P	P $\wedge$ Q	P $\vee$ Q	P $\rightarrow$ Q	P $\leftrightarrow$ Q
0	0	1	0	0	1	1
0	1		0	1	1	0
1	0	0	0	1	0	0
1	1		1	1	1	1

können Aussagen zu komplexeren Aussagen verknüpft werden. Die Definition der Syntax der Aussagenlogik ist in Tab. 10 dargestellt. Die Definition ist induktiv – ausgehend von den atomaren Elementen der Sprache wird festgelegt, wie aus bereits syntaktisch korrekten Ausdrücken komplexere neue, ebenfalls syntaktisch korrekte Aussagen gebaut werden können. Der Abschlusssatz in der Definition verhindert, dass auch andere syntaktische Ausdrücke als diejenigen, die durch die Konstruktionsanleitung definiert werden, zur Menge der wohlgeformten aussagenlogischen Formeln gezählt werden können.

Beispielsweise ist die Aussage $P \rightarrow Q$ eine aussagenlogische Formel. Diese Formel könnte für die Aussage “Wenn Mensch, dann sterblich” (“Mensch sein impliziert sterblich sein”) stehen. Die Semantik aussagenlogischer Formeln ist durch den Wahrheitswert gegeben, für den sie stehen. Dieses Konzept von Bedeutung ist vielleicht zunächst etwas befremdlich. Vielleicht hilft folgendes Beispiel zur Veranschaulichung: Die Aussage “Das ist ein Baum” ist wahr, genau dann, wenn ich dabei auf einen Baum zeige und sonst falsch. Der Wahrheitswert “wahr” wird also “wahren Aussagen” zugewiesen. Die syntaktische Konstante “True” erhält immer den Wahrheitswert “wahr” (notiert mit 1), die Konstante “False” den Wahrheitswert “falsch” (notiert mit 0). Die Bedeutung der Junktoren ist in Tab. 11 gegeben. Komplexe Formeln können mithilfe der Bedeutung der Junktoren zu Wahrheitswerten verkürzt werden. Formeln, die unabhängig von den Wahrheitswerten der Atome immer zu wahr auswerten, heissen *Tautologien*.

Betrachten wir die Tautologie $P \rightarrow (Q \vee \neg Q)$. Die Disjunktion $Q \vee \neg Q$ ist immer wahr: Wenn Q wahr ist, ist $\neg Q$ falsch und umgekehrt. Eine Disjunktion, bei der ein

Element wahr ist, ist wahr. Die Implikation "Wenn P, dann wahr" ist ebenfalls immer wahr, unabhängig davon, ob P wahr oder falsch ist (siehe Spalte zum Implikationspfeil in Tab. 11). Damit ist gezeigt, dass beispielsweise die Bauernregel "Wenn der Hahn kräht auf dem Mist (P), dann ändert sich das Wetter (Q) oder bleibt wie es ist ($\neg Q$)" eine allgemeingültige Aussage ist.

Wie bei mathematischen Ausdrücken, können logische Formeln durch Nutzung von Äquivalenzen umgeformt werden. Beispielsweise gelten die Regeln von de Morgan:

$$\neg(P \wedge Q) \equiv (\neg P \vee \neg Q)$$

$$\neg(P \vee Q) \equiv (\neg P \wedge \neg Q)$$

und eine Implikation kann durch eine Disjunktion ersetzt werden:

$$(P \rightarrow Q) \equiv (\neg P \vee Q).$$

Die obige Aussage $P \rightarrow (Q \vee \neg Q)$ ist also gleichbedeutend mit der Aussage $\neg P \vee (Q \vee \neg Q)$. Solche Äquivalenzumformungen sind wesentlich für die Anwendung syntaktischer Beweiskalküle, wie das unten dargestellte Resolutionskalkül. Durch rein syntaktische Umformungen wird versucht, eine gegebene Menge von Formeln so zu transformieren, dass am Ende ein Wahrheitswert als Ergebnis steht.

Prädikatenlogik erster Stufe In der Aussagenlogik steht eine atomare Formel für eine komplette Aussage, in der Prädikatenlogik erhalten die Formeln dagegen interne Struktur. Beispielsweise ist die Aussage "Sokrates ist ein Mensch" eine Formel (P) in der Aussagenlogik, in der Prädikatenlogik erster Stufe können wir dagegenen schreiben *mensch(Sokrates)*. Dabei heißt *mensch* Prädikat und *Sokrates* ist eine Konstante, die als Argument des Prädikates dient. Die Aussage "Der Vater von Sokrates ist ein Mensch" kann ausgedrückt werden als *mensch(vater-von(Sokrates))*. Dabei ist *vater-von* eine Funktion, die für ihr Argument einen konstanten Wert zurückliefert. Für *vater-von(Sokrates)* = *Sophroniskus* kann der funktionale Ausdruck durch sein Ergebnis, ersetzt werden, die Formel kann also reduziert werden zu *mensch(Sophroniskus)*. Schließlich können Prädikate auch über Variablen definiert werden. Dies ist insbesondere interessant für All- und Existenzaussagen. Beispielsweise kann die Aussage "Alle Menschen sind sterblich" mit dem Allquantor ($\forall$) dargestellt werden als $\forall x$ *mensch(x)* $\rightarrow$ *sterblich(x)*; die Aussage, "Es existiert ein glücklicher Mensch" kann mit dem Existenzquantor ($\exists$) dargestellt werden als $\exists x$ *mensch(x)* $\wedge$ *glücklich(x)*.

Allgemein sind die Argumente von Prädikaten Terme. Atomare Formeln können dann, wie in der Aussagenlogik, durch Junktoren zu komplexeren Formeln zusammengefügt werden. Zusätzlich gibt es bei der Prädikatenlogik erster Stufe noch dei beiden genannten Quantoren – den Allquantor und den Existenzquantor. Die induktive Definition der Prädikatenlogik erster Stufe ist in Tab. 12 gegeben. Dabei ist es wesentlich, den Unterschied zwischen *Termen* und *Formeln* zu beachten. Terme werden, wie in der Mathematik üblich, zu Werten ausgewertet, etwa $2 + 5$ zu 7, Prädikate und Formeln dagegen, wie auch in der Aussagenlogik, zu ganz speziellen Werten, nämlich Wahrheitswerten.

Tabelle 12: Syntax der Prädikatenlogik erster Stufe.

- Terme:
  - Variablen $x \in X$ sind Terme.
  - Wenn f ein Funktionssymbol mit n Argumenten ist und $t_1 \ldots t_n$ Terme sind, dann ist auch $f(t_1, \ldots, t_n)$ ein Term. (Konstanten sind dabei Funktionen mit 0 Argumenten).
  - Das sind alle Terme.
- Formeln:
  - Wenn P ein Prädikatsymbol mit n Argumenten ist und $t_1 \ldots t_n$ Terme sind, dann ist $P(t_1, \ldots, t_n)$ eine Formel. (atomare Formel)
  - Wenn P und Q Formeln sind, dann auch
    $\neg P$ (nicht P),
    $P \wedge Q$ (P und Q), $P \vee Q$ (P oder Q)
    $P \rightarrow Q$ (P impliziert Q) $P \leftrightarrow Q$ (P genau dann wenn Q).
  - Wenn x eine Variable ist und P eine Formel, dann sind auch $\exists x\ P$ und $\forall x\ P$ Formeln.
  - Das sind alle Formeln.

Formel:
$(\exists$x on(x, A) $\rightarrow \neg$clear(A)$) \wedge \exists$y clear$($topof(y)$)$

Interpretation der Symbole:

- Konstantensymbol A: Block in einer Blockwelt
- Einstelliges Funktionssymbol *topof(x)*: Funktion, die den Block liefert, der direkt auf Block x liegt
- Einstelliges Prädikatsymbol *clear(x)*: wahr, wenn kein Block auf Block x liegt, falsch sonst
- Zweistelliges Prädikatsymbol *on(x, y)*: wahr, wenn Block x auf Block y liegt, falsch sonst

Mögliche Strukturen:

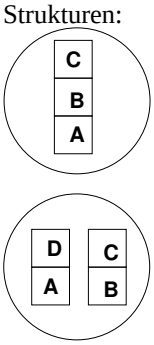


Abbildung 18: Veranschaulichung der Interpretation einer prädikatenlogischen Formel.

Wie in der Aussagenlogik wird die Bedeutung von Formeln durch Wahrheitswerte festgelegt. Allerdings kann dies hier nicht direkt geschehen, da zunächst die Bedeutung der Argumente der Prädikate bestimmt werden muß. Dies geschieht bezüglich einer Menge von Objekten mithilfe einer Interpretationsfunktion. Objektmenge und Interpretationsfunktion gemeinsam definieren eine Struktur. Die formale Definition der Semantik der Prädikatenlogik findet sich in Logiklehrbüchern (Schöning, 1992) und, etwas anschaulicher, auch in KI-Lehrbüchern (Russell & Norvig, 2002). Wir wollen die Grundidee an einem Beispiel veranschaulichen (siehe Abb. 18). Wenn wir die Bedeutung der Symbole der Formel in Abb. 18 bezüglich einer Blockwelt interpretieren, so kann die Formel der folgenden Aussage entsprechen: Wenn ein Block auf Block A liegt, so ist A nicht frei und es existiert ein Block, der auf einem anderen Block steht, aber auf dem selbst kein weiterer Block steht. An den zwei möglichen graphischen Veranschaulichungen sehen wir, dass Block *A* nicht frei ist, weil in beiden Strukturen ein Block auf *A* liegt. Ferner gilt in der ersten Veranschaulichung für *B* und in der zweiten für *A* oder *B*, dass der Block, der auf diesen Blöcken liegt, frei ist. Der erste Teil der Konjunktion ist in jeder legalen Blockwelt wahr; der zweite Teil ist in jeder Blockwelt wahr, in der ein Turm aus mindestens zwei Blöcken existiert.

Eine Struktur, in der eine Formel zu "wahr" ausgewertet werden kann, heisst auch Modell der Formel. Wenn jede mögliche Struktur ein Modell der Formel ist, heißt die Formel *gültig*, wenn mindestens ein Modell existiert, heißt die Formel *erfüllbar*. Beispielsweise ist die Formel $\forall P(x) \vee \neg P(x)$ gültig. Egal, wie wir die Formel interpretieren, ob als "Das Wetter ist regnerisch oder das Wetter ist nicht regnerisch" oder "Ein Block ist frei oder nicht frei", oder mit einer beliebigen anderen Semantik, in der klassischen Logik ist die Disjunktion eines Prädikates mit seiner Negation immer wahr. Die in Abb. 18 angegebene Formel ist erfüllbar – wir haben zwei mögliche Modelle angegeben.

Eine Formel G heißt logische Folgerung aus einer Menge von Formeln $F = \{F_1 \ldots F_n\}$, wenn jedes Modell von F auch ein Modell von G ist. Da alle Formeln in F als wahr angenommen werden, entspricht die Menge der Konjunktion $F_1 \wedge F_2 \wedge \ldots F_n$, abgekürzt notiert als $\wedge_{i=1}^{n} F_i$. Wenn gilt, dass $(\wedge_{i=1}^{n} F_i) \rightarrow G$ gültig ist (eine Tautologie ist), ist G eine logische Folgerung aus F. Äquivalent dazu ist dann die Formel $(\wedge_{i=1}^{n} F_i) \wedge \neg G$

nicht erfüllbar (ein Widerspruch).

Während die Aussagenlogik bereits in der Antike eigeführt wurde, wurde die Prädikatenlogik erst im neunzehnten und zwanzigsten Jahrhundert entwickelt. Zunächst formalisierte George Boole 1847 die Aussagenlogik. Darauf aufbauend entwarf Gottlob Frege 1879 die Prädikatenlogik. Das Konzept der Interpretation von Formeln bezüglich Objekten einer realen (oder Modell-) Welt hat Alfred Tarski dann im 20. Jahrhundert eingeführt.

Der Hauptgewinn einer formal definierten Logik ist, dass eindeutig festgelegt werden kann, wie logisch korrekte Schlussfolgerungen gezogen werden können. Darauf aufbauend können dann Algorithmen entwickelt werden, die es ermöglichen, dass logische Schlussfolgerungen automatisch, durch einen Computer, ausführbar werden. Der wesentliche Schritt zu diesem sogenannten automatischen Theorembeweisen war das von Robinson 1965 eingeführte Resolutionsverfahren. Automatische Theorembeweiser nutzen die oben angegebene Beziehung zwischen logischen Schlussfolgerungen und syntaktischen Ausdrücken aus. Wenn beispielsweise rein syntaktisch, mithilfe von Inferenzregeln, gezeigt werden kann, dass $(\wedge_{i=1}^{n} F_i) \wedge \neg G$ zu falsch auswertet, so ist gezeigt, dass G logisch aus der Formelmenge F folgt! Das heisst, G ist das bewiesene Theorem.

3.2.2 Resolutionskalkül

Ist gegebenes Wissen formal repräsentiert, etwa aussagenlogisch oder prädikatenlogisch, so können Schlüsse rein mechanisch, durch ein Computerprogramm, ausgeführt werden. Das bereits erwähnte Resolutionskalkül ist eines der bekanntesten Verfahren zum automatischen Schlussfolgern. Es besteht nur aus einer Regel, nämlich:

$$(P_1 \vee \ldots \vee P_n \vee Q) \wedge (\neg Q \vee R_1 \vee \ldots \vee R_n) \rightarrow (P_1 \vee \ldots \vee P_n \vee R_1 \vee \ldots \vee R_n).$$

Betrachtet man also zwei Klauseln, die beide als wahr angenommen werden (deshalb werden diese konjunktiv verknüpft), und taucht ein Prädikat in einer Klausel positiv und in der anderen negiert auf, so können die beiden Klauseln zu einer zusammengefügt werden, wobei die Klausel, die positiv und negativ auftauchte, gestrichen wird. Für zwei Klauseln $P \wedge \neg P$ resultiert diese Regel in der leeren Klausel. Offensichtlich kann ein Prädikat nie gleichzeitig mit seinem Gegenteil gelten – die leere Klausel repräsentiert also den Wahrheitswert “falsch” beziehungsweise einen Widerspruch!

Die Beweisidee hinter dem Resolutionskalkül entspricht also der oben angegebenen Vorgehensweise, dass $(\wedge_{i=1}^{n} F_i) \wedge \neg G$ zu falsch auswertet. Resolution basiert also darauf, einen Widerspruchsbeweis zu führen: Um zu zeigen, dass eine Formel G aus einer Menge von Formeln F logisch folgt, wird G negiert zur Formelmenge hinzugefügt und gezeigt, dass dadurch generell “falsch” abgeleitet wird – was gleichbedeutend damit ist, dass ein Widerspruch in der Formelmenge steckt.

Viele komplexere Deduktionssysteme sowie die Programmiersprache Prolog basieren auf Resolution. Damit das Verfahren angewendet werden kann, müssen Formeln in einheitlicher Form repräsentiert werden – nämlich in Klauselform. Eine Klausel ist eine Disjunktion von Prädikaten. Variablen müssen alle über Allquantoren gebunden sein. Beliebige Formeln können durch Äquivalenzumformungen (einige Beispiele wurden oben dargestellt) in Klauselform umgewandelt werden. Alle notwendigen Schritte,

um eine solche Umformung vorzunehmen werden in Logikbüchern (Schöning, 1992) und KI-Lehrbüchern (Russell & Norvig, 2002) dargestellt. Eine ausführliche, leicht verständliche Einführung der Resolution in Prädikatenlogik erster Stufe findet sich in Schmid und Kindsmüller (1996).

Im folgenden wird Resolution anhand eines Beispiels eingeführt. Wieder betrachten wir den einfachen Syllogismus von der Sterblichkeit, diesmal in prädikatenlogischer Form:

(1) $\forall(x)mensch(x) \rightarrow sterblich(x)$

(2) *mensch(Sokrates)*.

Da ein einzelnes Prädikat bereits in Klauselform (mit null Disjunktionen) ist, muss nur die erste Formel umgewandelt werden. Durch Auflösen der Implikation ergibt sich:

(1') $\forall(x)(\neg mensch(x) \vee sterblich(x))$

Die Formel enthält genau eine allquantifizierte Variable. Da Klauseln, wie oben definiert, nur allquantifizierte Variablen enthalten, kann der Quantor zur Vereinfachung weggelassen werden (die Formel ist dann implizit allquantifiziert):

(1") $\neg mensch(x) \vee sterblich(x)$

Um nun zu prüfen, ob *sterblich(sokrates)* eine gültige Aussage ist (aus der Formelmenge folgt), wird diese "Vermutung" negiert zur Klauselmenge hinzugefügt:

(3) $\neg sterblich(sokrates)$.

Der Resolutionsbeweis wird nun so geführt, dass schrittweise immer zwei Formeln betrachtet werden, auf die die Resolutionsregel angewendet wird – solange bis die leere Klausel abgeleitet wird. Ableiten einer leeren Klausel bedeutet, dass ein Widerspruch im Formelsystem steckt. Wenn mit einer negierten Formel G ein Widerspruch in der Formelmenge erzeugt wird, ist dies gleichbedeutend damit, dass die positive Formulierung von G aus der Formelmenge folgerbar ist.

Resolution kann in Form eines sogenannten Refutationsbaums dargestellt werden (siehe Abb. 19). Wenn man mit der negierten Formel *(3)* startet und Formel *(1")* hinzufügt (die Konjunktion wird hier nicht mitnotiert), so taucht das Prädikat *sterblich* in Formel *(3)* negiert und in Formel *(1")* positiv auf. Die Variable x darf durch die Konstante *Sokrates* ersetzt werden. Im Allgemeinen müssen für jeden Resolutionsschritt zwei Formeln unifiziert werden, was heißt, dass Variablen so durch Konstanten oder komplexere Terme ersetzt werden müssen, dass die in der Resolution herauszuschneidenden Prädikatausdrücke identisch werden. Nun kann der Resolutionsschritt durchgeführt werden. Die resultierende Formel kann nun zusammen mit Klausel *(2)* betrachtet werden, was in einem Widerspruch resultiert. Damit wurde nun durch Widerspruch bewiesen, dass die Formel *sterblich(Sokrates)* aus Formeln *(1)* und *(2)* logisch folgt.

Ob ein Widerspruch (in endlicher Zeit) wirklich gefunden wird, wenn ein solcher existiert, hängt unter anderem von der Reihenfolge ab, in der die Klauseln in die Resolution einbezogen werden. Je größer die Menge an Formeln (Wissen) ist, aus der

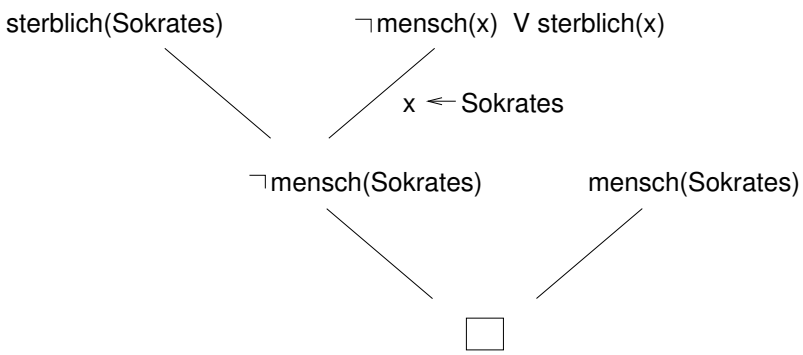


Abbildung 19: Ein Refutationsbaum.

Schlüsse gezogen werden sollen, desto komplexer wird das Problem der Suche nach geeigneten Klauselpaaren. Entsprechend wird ein Resolutionsverfahren immer zusammen mit einer Suchstrategie (vergleiche die in Abschnitt 2 dargestellten Suchverfahren) realisiert. Um das Ziehen von Schlüssen effizienter zu machen, wird ein logisches System häufig eingeschränkt, in dem die sogenannte *closed-world assumption* eingeführt wird. Basierend auf dieser Annahme kann man dann davon ausgehen, dass etwas, was nicht als wahr ableitbar ist, nicht gilt.

3.2.3 Theorembeweiser und Wissensrepräsentation

Automatisierte Beweisverfahren wie die Resolution sind immer dann nützlich, wenn aus größeren Beständen von (als gültig angenommenen) Wissensbeständen, Schlüsse gezogen werden sollen. Deduktion ist ein aktives Forschungsgebiet in der Künstlichen Intelligenz. Es gibt verschiedene Systeme, die erlauben automatische Beweise durchzuführen, wie Isabelle (Paulson, 1989), Otter (Kalman, 2001) oder Nuprl (Constable et al., 1986). Diese Systeme basieren nicht (nur) auf Prädikatenlogik erster Ordnung und arbeiten nicht (nur) mit dem Resolutionskalkül. Die logische Programmiersprache Prolog ist dagegen ein spezieller Theorembeweiser für Prädikatenlogik erster Ordnung, der auf der Resolution spezieller Klauseln, sogenannter Hornklauseln, basiert. Hornklauseln sind Klauseln, die höchstens ein positives atomares Prädikat enthalten. Die oben angegebene Formel von der Sterblichkeit ist also eine Hornklausel. Die Implikation

$$\forall(x) mensch(x) \rightarrow sterblich(x)$$

entspricht der Prolog-Regel `sterblich(X) :- mensch(X).`

Dabei repräsentiert das Symbol `:-` den Implikationspfeil (von rechts nach links geschrieben). Prolog erlaubt das effiziente Ziehen von Schlüssen einmal dadurch, dass es auf der oben genannten *closed world assumption* basiert und zum anderen durch die sogenannte SLD-Resolution. SLD steht dabei für *linear resolution with selection function for definite clauses*. Die Auswahlfunktion bestimmt, dass das Prädikat, das

im nächsten Resolutionsschritt betrachtet wird, "von oben nach unten von links nach rechts" in der Menge gegebener Fakten und Regeln gesucht wird. Dadurch wird ein Teil des Wissens über die Ableitung auf den Programmierer verlagert. Bei ungünstiger Anordnung der Regeln kann es passieren, dass ein Programm nicht terminiert (im Prinzip endlos suchen würde). Lineare Resolution meint, dass immer mit der negierten zu beweisenden Formel (Anfrage) gestartet wird. In jedem Resolutionsschritt wird genau eine Klausel aus der Menge der vorgegebenen Klauseln zusammen mit der gerade erzeugten Klausel betrachtet. Definite (Horn-)Klausel bedeutet, dass immer Klauseln mit genau einem positiven Literal bertrachtet werden – dies entspricht genau der Form der Prolog-Regeln, bei denen die eine, auf der linken Seite im Regelkopf gegebene Klausel, positiv ist und alle anderen Klauseln negativ sind. Wandelt man Implikationen, bei denen im wenn-Teil eine beliebige Menge von konjunktiv verknüpften Prädikaten steht und im dann-Teil genau ein Prädikat, in Klauselform um, so entspricht dies genau solchen definiten Klauseln.

Deduktion spielt in der KI nicht nur im Rahmen von Forschungsarbeiten, die sich speziell mit der Entwicklung von möglichst leistungsstarken automatischen Beweissystemen beschäftigen, eine Rolle, sondern ist auch eine wesentliche Grundlage für wissensbasierte Systeme (wie Expertensysteme). Dort wird Wissen in strukturierter Form repräsentiert – etwa in Form semantischer Netze. Die meisten dieser Repräsentationsformalismen basieren auf einem Logik-Kalkül. Entsprechend werden Schlußfolgerungen in wissensbasierten Systemen ebenfalls mithilfe logischer Deduktion gezogen.

Bekannt wurde KL-One (Brachman & Schmolze, 1985) – ein System, das auf sogenannter terminologischer Logik (auch als *description logic* oder formale Ontologie bezeichnet) basiert. Terminologische Logik ist eine Untermenge der Prädikatenlogik erster Stufe, die speziell zur Beschreibung von Konzeptstrukturen und Enthaltenseins-Beziehungen entwickelt wurde. Ein frühes Beispiel für ein solches System ist der "Teachable Language Comprehender" (Quillian, 1968). Ein Konzept wie Tier wird mit seinen spezifischen Eigenschaften, etwa, dass es atmet, beschrieben. Unterkonzepte wie Säugetier oder Hund erben die Eigenschaften der Oberkonzepte. Eine Schlußfolgerung, wie, dass ein Hund atmet, kann dann durch eine rekursive Regel, die die Transitivität der Vererbungsrelation beschreibt, gezogen werden. Ein großes Projekt, das auf einer Ontologie mit etwa 6000 Konzepten und etwa 60.000 beschreibenden Eigenschaften und Fakten basiert, ist CYC (Lenat, 1995).

3.3 Probleme klassischer Deduktion und Alternativen

Die Formalisierung von schlußfolgerndem Denken auf der Basis eines logischen Kalküls erlaubt die Automatisierung von Schlußfolgerungen und garantiert, dass die gezogenen Schlüsse (logisch) korrekt sind. Dabei wird allerdings davon ausgegangen, dass die vorgegebenen logischen Formeln, die das Wissen des Systems beschreiben, alle gültig sind. Die Beurteilung der Wahrheit dieser Axiome ist nicht innerhalb der Logik möglich.

Klassische Logik basiert auf dem Konzept des *tertium non datur* – etwas, das nicht wahr ist, ist falsch. Dies hat den Vorteil, dass man Widerspruchsbeweise, wie Resolutionsbeweise, führen kann. Allerdings gibt es viele Wissensbereiche, in denen Sachverhalte nicht einfach nur eindeutig wahr oder falsch sind, sondern mehr oder weniger

zutreffen können. Eine bekannter, nicht-klassischer Ansatz, der diesem Umstand gerecht wird, ist die Fuzzy-Logik. Hier werden Konzepte dadurch charakterisiert, wie stark sie als zu einer Menge gehörig, gesehen werden. Beispielsweise ist ein bestimmtes Tier nicht entweder groß oder nicht groß, sondern es gehört mit einem bestimmten Zugehörigkeitswert zur Menge der großen Dinge. Die Zugehörigkeiten können Werte zwischen 0 und 1 annehmen, wobei diese Extremfälle den klassischen Wahrheitswerten entsprechen. Der Wahrheitswert für mit Junktoren verknüpfte Zugehörigkeiten wird mit speziellen Regeln berechnet, beispielsweise:

(1) $T(A \wedge B) = \min\,(T(A), T(B))$

(2) $T(A \vee B) = \max\,(T(A), T(B))$

(3) $T(\neg A) = 1 - T(A)$.

Für Zugehörigkeitswerte von 0 und 1 entsprechen diese Regeln genau der Semantik der klassischen zweiwertigen Logik. Fuzzy-Logik eignet sich beispielsweise recht gut, um Konzepte der Prototypentheorie (Rosch, 1975) abzubilden (Zadeh, 1982).

Eine weitere Art von Unbestimmtheit, die in der klassichen Logik nicht abgebildet werden kann, ist dadurch gegeben, dass sich der Wahrheitswert einer Aussage in Abhängigkeit vom vorhandenen Wissen ändern kann. Beispielsweise würde ein Mensch, wenn man ihm mitteilt, daß Tweety ein Vogel ist, annehmen, dass Tweety fliegen kann. Er würde also mithilfe der Regel *vogel(x)* $\rightarrow$ *fliegt(x)* eine Schlußfolgerung ziehen. Bei der Regel wurde der Quantor weggelassen, da die Regel korrekt heißen müßte “im Normalfall gilt für x”. Wenn er dann aber erfährt, dass Tweety ein Pinguin ist, würde er den bereits gezogenen Schluss revidieren und nun annehmen, dass Tweety nicht fliegen kann. Schlussfolgern, bei dem bereits gezogene Schlüsse zurückgenommen werden können, heißt *nicht-monoton*. Im Gegensatz dazu ist die klassische Logik *monoton*, da ein neu gezogener Schluss keinen Einfluß auf bereits gezogene Schlüsse nehmen kann. Es gibt verschiedene Ansätze zum nicht-monotonen Schließen. Ein früher Ansatz ist die Default-Logik (Reiter, 1980). Bekannte nichtmonotone Ansätze sind Truth-Maintainance-Systeme sowie Modallogiken (Brewka, 1995).

3.4 Qualitatives Schließen

Menschliches Schließen basiert häufig auf anderen Strategien als dem Anwenden von logischen Regeln. Häufig scheinen Schlüsse auf der Basis von anschaulichen, zum Teil bildhaften Vorstellungen gezogen zu werden. Ein klassischer experimenteller Beleg hierfür ist der Symbol-Distanz-Effekt (Potts, 1972): Gegeben Aussagen der Form “Hans ist klüger als Peter”, “Peter ist klüger als Rudi”, “Rudi ist klüger als Franz” sollen Aussagen wie “Hans ist klüger als Franz” verifiziert werden. Würden Schlüsse hier mit Hilfe der Transitivitätsregel gezogen, so müßten die Reaktionszeiten um so höher sein, je häufiger die Regel angewendet werden muß – es müsste also mehr Zeit benötigen, die Aussage “Hans ist klüger als Franz” zu verifizieren, als die Aussage “Hans ist klüger als Rudi”. Das Gegenteil ist aber der Fall. Dieser Befund kann dadurch erklärt werden, dass die in den Aussagen genannten Personen in einer internen

Repräsentation linear nach dem Ausmaß der ihnen zugeschriebenen Eigenschaft angeordnet werden. Verifikation kann dann durch Betrachtung des "mentalen Bildes" erfolgen, indem geprüft wird, ob Person X links oder rechts von Person Y steht. Je weiter die Personen auseinander stehen, desto schneller geht der mentale Vergleich. Zahlreiche weitere empirische Belege liefern Experimente zum syllogistischen Schließen mit mentalen Modellen (Johnson-Laird, 1983) (siehe Knauff, in diesem Band).

In der künstlichen Intelligenz werden solche, auf eher bildhaften Repräsentationen basierenden Schlußfolgerungstechniken im Bereich des diagrammatischen Schließens (Larkin & Simon, 1987) und im Bereich des qualitativen Schließens (Kuipers, 1994) untersucht.

Ein klassisches, einfaches Kalkül zum qualitativen Schließen ist das sogenannte Allen-Kalkül (Allen, 1983). Dieses Kalkül erlaubt es, Schlußfolgerungen über die Beziehungen zwischen Zeitintervallen oder zwischen räumlichen Beziehungen in einer Dimension (Güsgen, 1989) zu ziehen. Im folgenden betrachten wir den Fall des räumlichen Schließens. Der Aufenthaltsort eines Objekts wird als Intervall mit Anfangs- und Endpunkt repräsentiert. Jedes Intervall X ist ein geordnetes Paar aus Begrenzungspunkten s_X und e_X. Dies Punkte können als reelwertige Zahlen interpretiert werden, wobei gilt: $s_X < e_X$. Da keine konkreten reellen Zahlen betrachtet werden, die das Intervall (etwa in einem euklidischen Koordinatensystem) beschreiben, basiert der Ansatz nicht auf metrischer/quantitativer Information. Es werden nur relative (ordinale) Beziehungen, also qualitative Information betrachtet. Dies entspricht in etwa der Idee eines mentalen Modells (Johnson-Laird, 1983), in dem Objektanordnungen strukturell-räumlich abgebildet werden. Repräsentiert man etwa den Sachverhalt "*Die Palme ist links vom Computer*" (siehe Abb. 20), so muß auf jeden Fall gelten, dass das Intervall, das den Aufenthaltsort der Palme repräsentiert, sich vollständig links vom Intervall, das den Computer repräsentiert, befindet, also

$$s_P < s_C, s_P < e_C, e_P < s_C, e_P < e_C$$

wobei P für Palme und C für Computer steht. Dagegen ist es irrelevant, wie weit genau Palme und Computer auseinanderstehen. Das heißt, eine qualitative räumliche Repräsentation hat in etwa die Granularität einer natürlichsprachigen Beschreibung und steht für eine ganze Klasse möglicher konkreter Realisierungen in der physikalischen Welt. Die Informationen, die im oben dargestellten Experiment zum Symbol-Distanz-Effekt gegeben wurden, können durch Verwendung der gerade eingeführten qualitativen Relation "links-von" repräsentiert werden, indem "klüger als" auf "links-von" abgebildet wird.

Das Allen-Kalkül basiert auf insgesamt 13 Basisrelationen, die paarweise disjunkt und exhaustiv sind (siehe Tab. 13). Das heißt, jede mögliche räumliche Beziehung, die zwei Objekte in einer Dimension zueinander einnehmen können, kann eindeutig repräsentiert werden.

Sind gegebene räumliche Sachverhalte durch Basisrelationen repräsentiert, so können folgende "Rechen"-Operationen ausgeführt werden:

Inverse: $I\ R^{-1}\ J \Leftrightarrow J\ R\ I$.

Schnitt: $I\ (R \cap S)\ J \Leftrightarrow I\ R\ J \wedge I\ S\ J$.

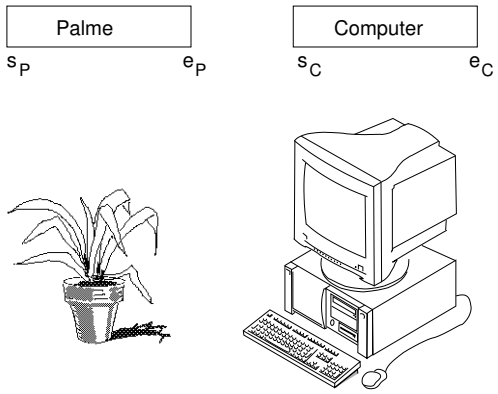


Abbildung 20: ”Die Palme steht links vom Computer” repräsentiert im Intervallkalkül und eine mögliche physikalische Realisierung.

Tabelle 13: Die 13 Basisrelationen des Allen-Kalküls in ihrer räumlichen Interpratation (nicht aufgeführt sind die Inversen *fi*, *di*, *si*, *oi*, *mi*, $\succ$; diese ergeben sich durch Vertauschung der Argumente X und Y).

Symbol	Sprachl. Beschreibung	Bildl. Beschreibung	Punkt-Ordnung
$X \prec Y$	X liegt links von Y		$s_X < e_X < s_Y < e_Y$
$X\ m\ Y$	X berührt Y von links		$s_X < e_X = s_Y < e_Y$
$X\ o\ Y$	X überlappt Y von links		$s_X < s_Y < e_X < e_Y$
$X\ s\ Y$	X liegt linksbündig in Y		$s_Y = s_X < e_X < e_Y$
$X\ d\ Y$	X liegt voll in Y		$s_Y < s_X < e_X < e_Y$
$X\ f\ Y$	X liegt rechtsbündig in Y		$s_Y < s_X < e_X = e_Y$
$X\ =\ Y$	X ist gleich Y		$s_X = s_Y < e_Y = e_X$

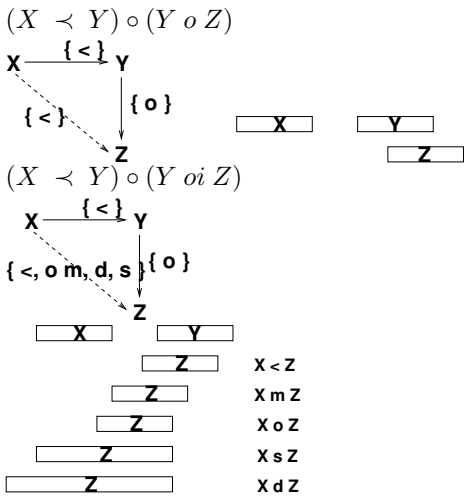


Abbildung 21: Komposition von Relationen.

Komposition: $I(R \circ S)J \Leftrightarrow \exists K : (I\ R\ K \wedge K\ S\ J)$.

Die Inversenbildung kann einfach aus der Tabelle der Basisrelationen abgelesen werden. Beispielsweise ist der zu "Die Palme steht links vom Computer" inverse Sachverhalt, dass der Computer rechts von der Palme steht. Der Schnitt zweier räumlicher Relationen engt die Menge der möglichen Aufenthaltsorte ein. Hat man etwa bislang die unspezifische Information, dass ein Buch I und eine Mappe J sowohl linksüberlappend als auch linksbündig zueinander liegen könnten und erhält man aus anderer Quelle (etwa durch eine Schlussfolgerung) die Information, dass Buch und Mappe links voneinander oder linksüberlappend liegen können, so ergibt der Schnitt aus diesen Relationen, dass Buch und Mappe nur linksüberlappend zueinander liegen können. Die für das Ziehen von Schlussfolgerungen wesentliche Operation ist die Komposition: Gegeben die Information, dass I in einer bestimmten Relation zu K steht und dass K in einer bestimmten Relation zu J steht, sollte auch klar sein, wie I und J zueinander stehen. Die Komposition kann über die Endpunktrelationen der Basisrelationen abgeleitet werden. Sie wird üblicherweise in einer Kompositionstabelle angegeben.[3]

Komposition führt im allgemeinen dazu, dass die Relationen nicht mehr eindeutig feststehen (siehe Abb. 21). Verknüpft man die Aussagen "Die Palme ist links vom Computer" und "Der Computer überlappt die Mappe von links", so ergibt die Komposition das eindeutige Ergebnis, dass die Palme links von der Mappe ist. Verknüpft man aber die Aussagen "Die Palme ist links vom Buch" und "Das Buch überlappt die Mappe von rechts", so kann eine von fünf verschiedenen Relationen gelten.

Im Rahmen des Allen-Kalküls können nun folgende Schlussfolgerungsprobleme gelöst werden: Es kann festgestellt werden, ob eine Formelmenge erfüllbar ist. Das heißt, für eine Menge gegebener räumlicher Relationen zwischen Objekten kann ge-

[3]Die Kompositionstablle ist beispielsweise in der Originalarbeit von Allen (1983) angegeben.

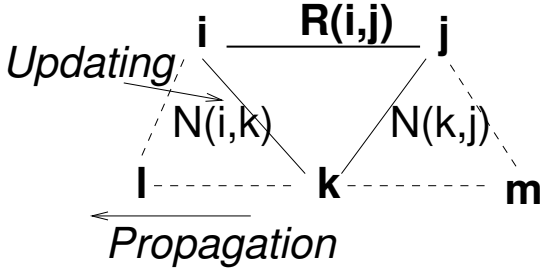


Abbildung 22: Constraint-Propagation.

prüft werden, ob es eine konsistente, physikalisch realisierbare, Anordnung in einer Dimension gibt. Zudem können Schlussfolgerungen über bisher nicht explizit gegebene Relationen zwischen Objektpaaren gezogen werden, indem für jedes Paar von Intervallen die stärkste daraus folgende Relation errechnet wird.

Die Schlussfolgerungen können mechanisch mithilfe eines *constraint propagation*-Algorithmus ermittelt werden. Grob betrachtet, funktioniert der Algorithmus folgendermaßen (siehe Abb. 22):

- Einfügen einer neuen Relation $R(i, j)$
- Für alle mit i/j direkt verbundenen Knoten k:
  Berechne die Constraints von k nach j via i und von i nach k via j.
- Falls sich Änderungen ergeben: behandle $R(k, j)$/$R(i, k)$ entsprechend.

Zur Veranschaulichung ist ein Beispiel in Tab. 14 angegeben.

Im Gegensatz zur klassischen Logik erlaubt das Allen-Kalkül das Ziehen von Schlussfolgerungen über anschaulichen Repräsentationen und entspricht daher eher der Strategie menschlichen Schließens. Allerdings ist es nicht plausibel anzunehmen, dass ein Mensch ganze Mengen möglicher gültiger Relationen bei der Inferenz berücksichtigt. In einigen empirischen Untersuchungen zum Allen-Kalkül konnte gezeigt werden, daß Menschen offensichtlich bei der Auswahl von möglichen gültigen Relationen bestimmte Relationen präferieren (Knauff, Rauh, & Schlieder, 1995). Dabei scheinen die Präferenzen interindividuell konstant zu sein. Betrachtet man etwa die Aussagen:

X ist links von Y

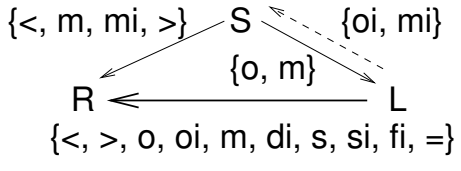
Y liegt rechtsbündig in Z

so ergibt die Komposition fünf mögliche Beziehungen zwischem X und Z (siehe Abb. 23), allerdings wählen Probanden bevorzugt die am wenigsten restriktive Beziehung "X überlappt Z von links".

Tabelle 14: Illustration des Allen-Kalküls.

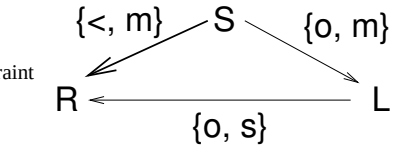
- (1) $S\ \{o, m\}\ L\ (=\ L\ \{oi, mi\}\ S)$
- (2) $S\ \{\prec, m, mi, \succ\} R$
- Berechne **Constraints(**$\{o, m\}, \{\prec, m, mi, \succ\}$**)**
  Beziehung zwischen L und R via S (Konversenbildung $L \rightarrow S$)
  - $T(oi, \prec) = \{\prec, o, m, di, fi\}$
  - $T(oi, m) = \{o, di, fi\}$
  - $T(oi, mi) = \{\succ\}$
  - $T(oi, \succ) = \{\succ\}$
  - $T(mi, \prec) = \{\prec, o, m, di, fi\}$
  - $T(mi, m) = \{s, si, =\}$
  - $T(mi, mi) = \{\succ\}$
  - $T(mi, \succ) = \{\succ\}$

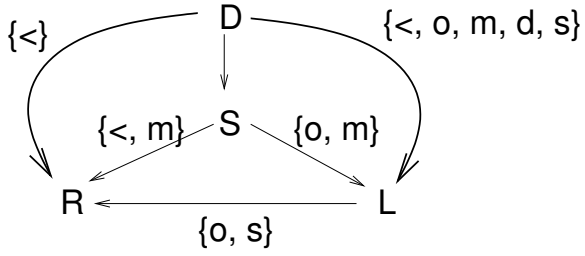
  $\hookrightarrow L\ \{\prec, \succ, o, m, di, s, si, fi, =\}\ R$



- (3) $L\ \{o, s, d\}\ R$
- Schnittmenge mit bisheriger Relation: $L\ \{o, s\}\ R$
- Einfügen des neuen Constraints $\hookrightarrow$ Propagation: neuer Constraint zwischen S und R
  $S\ \{o, m\}\ L\ \{o, s\}\ R$
  - $T(o, o) = \{\prec, o, m\}$
  - $T(o, s) = \{o\}$
  - $T(m, o) = \{\prec\}$
  - $T(m, s) = \{m\}$

  $\hookrightarrow S\ \{\prec, o, m\} R$
- Schnittmenge mit $S\ \{\prec, m, mi, \succ\}\ R$: $S\ \{\prec, m\}\ R$
- (4) $D\ \{d\}\ S$
- Neue Relationen:
  - $D\ \{\prec\}\ R$
  - $D\ \{\prec, o, m, d, s\}\ L$





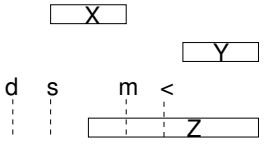


Abbildung 23: Präferierte Relation “überlappt” (o) und korrekte Alternativen.

4 Schlussbetrachtungen

Wesentliche Grundlagen für die Modellierung von Denk- und Problemlöseprozessen mit dem Computer sind Suchverfahren und Logik. Diese formalen Grundbausteine von KI-Programmen haben sicher nicht viel mit den Prozessen gemein, die menschliches Problemlösen und Denken steuern. Allerdings helfen diese formalen Grundlagen dabei, zu verstehen, welche Arten von Intelligenzleistungen im Prinzip algorithmisch lösbar sind. Grundkenntnisse in Logik und Suchverfahren können damit einen guten Ausgangspunkt zum Entwurf alternativer Problemlöse- und Schlussfolgerungsverfahren liefern, die dann eher menschlichen kognitiven Prozessen entsprechen. Allerdings ist menschliches Problemlösen und Schliessen meist sehr stark von Wissen, insbesondere von schwer formalisierbarem Alltagswissen (*common sense knowledge*) getragen. Das heißt, ein Abgleich zwischen menschlicher Informationsverarbeitung und entsprechenden Computermodellen wird immer am ehesten in eher artifiziellen Bereichen oder in Bereichen, in denen wenig spezifisches Wissen vorliegt, möglich sein.

Die in diesem Kapitel dargestellten formalen Grundlagen geben nur einen kleinen Ausschnitt der KI-Forschung in den Bereichen Denken und Problemlösen wieder. Ausführliche Informationen finden sich in Lehrbüchern der Künstlichen Intelligenz wie Russell und Norvig (2002). Einen guten Überblick über KI-Forschung in Deutschland gibt Görz (1995). Zahlreiche Hinweise zur Beziehung von Künstlicher Intelligenz und kognitiver Modellierung finden sich in Strube (1996).

Anmerkungen Abbildungen 1, 2, 3, 4, 9, 10, 12, 13 und 16 sind dem Beitrag

Schmid, U. (2002). Computermodelle des Problemlösens. In J. Müsseler und W. Prinz (Hrsg.), *Allgemeine Psychologie* (Kap. 5b, pp. 701-734). ©Elsevier GmbH, Spektrum Akademischer Verlag, Heidelberg

entnommen. Sie entsprechen dort den Abbildungen 5b-3, 5b-4, 5b-5, 5b-6, 5b-7, 5b-9, 5b-10, 5b-16 und 5b-18. Ich danke dem Spektrum-Verlag für die freundliche Erteilung des Nutzungsrechts.

Ferne danke ich Joachim Funke für die kritische Durchsicht des Manuskripts und Frau Gaby Bauer für Untersützung beim Layout.

Literatur

Aamodt, A., & Plaza, E. (1994). Case-based reasoning: foundational issues, methodological variations, and system approaches. *AI Communications*, *7*(1), 39–59.

Allen, J. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM*, *26*(11), 832-843.

Anderson, J. (1993). *Rules of the mind.* Hillsdale, NJ: Lawrence Erlbaum.

Anderson, J., & Thompson, R. (1989). Use of analogy in a production system architecture. In S. Vosniadou & A. Ortony (Eds.), *Similarity and analogical reasoning* (pp. 267–297). Cambridge, MA: Cambridge University Press.

Anderson, J. R. (1983). *The architecture of cognition.* Cambridge, MA: Havard 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: Lawrence 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.), *Kybernetik-Forschung* (Vol. 4, pp. 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*. Heidelberg: Springer.

Brachman, R., & Schmolze, J. (1985). An overview of the KL-ONE knowledge representation language. *Cognitive Science, 9*, 171-216.

Brewka, G. (1995). Nichtmonotones Schließen. In G. G. et al. (Ed.), *Einführung in die Künstliche Intelligenz (2. Aufl.)* (pp. 59–86). Bonn, Germany: Addison-Wesley.

Carbonell, J. (1986). Derivational analogy: A theory of reconstructive problem solving and expertise acquistion. In R. Michalski, J. Carbonell, & T. Mitchell (Eds.), *Machine Learning - An Artificial Intelligence Approach* (Vol. 2, pp. 371–392). Los Altos, CA: Morgan Kaufmann.

Chase, W. G., & Simon, H. A. (1973). Perception in chess. *Cognitive Psychology, 4*, 55-81.

Constable, R. L., Allen, S. F., Bromley, H. M., Cleaveland, W. R., Cremer, J. F., Harper, R. W., Howe, D. J., Knoblock, T. B., Mendler, N. P., Panangaden, P., Sasaki, J. T., & Smith, S. F. (1986). *Implementing mathematics with the Nuprl development system.* New York: Prentice-Hall.

Cooper, R., Fox, J., Farringdon, J., & Shallice, T. (1996). A systematic methodology for cognitive modelling. *Artificial Intelligence, 83*, 3-44.

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* (pp. 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.

Dörner, D. (2002). *Bauplan für eine Seele.* Reinbek, Germany: Rowohlt.

Evans, T. G. (1968). A program for the solution of a class of geometric-analogy intelligence-questions. In M. Minsky (Ed.), *Semantic information processing* (pp. 271–353). Cambridge, MA: MIT Press.

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.

Fodor, J. (1975). *The language of thought.* New York: Crowell.

Gentner, D. (1983). Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, *7*, 155-170.

Gentner, D., Ratterman, M. J., & Forbus, K. D. (1993). The roles of similarity in transfer: Separating retrievability from inferential soundness. *Cognitive Psychology*, *25*(4), 524-575.

Görz, G. (Ed.). (1995). *Einführung in die Künstliche Intelligenz (2. Auflage).* Bonn, Germany: Addison-Wesley.

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* (Vols. 5, Human information processing, pp. 239–270). Hillsdale, NJ: Lawrence Erlbaum.

Gries, D. (1981). *The science of programming.* New York: Springer.

Güsgen, H. W. (1989). *Spatial reasoning based on Allens temporal logic* (Tech. Rep.). Berkeley, CA: ICSI.

Harel, D. (1987). *Algorithmics - the spirit of computing.* Wokingham, Eng.: Addison-Wesley.

Hoffmann, J., & Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, *14*, 253-302.

Hofstadter, D., & The Fluid Analogies Research Group. (1995). *Fluid concepts and creative analogies.* New York: Basic Books.

Holyoak, K. J., & Thagard, P. (1989). Analogical mapping by constraint satisfaction. *Cognitive Science*, *13*, 295-355.

Hummel, J., & Holyoak, K. (1997). Distributed representation of structure: A theory of analogical access and mapping. *Psychological Review*, *104*(3), 427-466.

Jain, B. J., & Wysotzki, F. (2002). Self-organizing recognition and classification of relational structures. In *Proc. of the 24th Annual Meeting of the Cognitive Science Society, Fairfax, Virginia* (pp. 488–493). Mahwah, NJ: Lawrence Erlbaum.

Johnson-Laird, P. (1983). *Mental models: Towards a cognitive science of language, inference, and consciousness.* Cambridge, MA: Cambridge University Press.

Kalman, J. A. (2001). *Automated reasoning with Otter.* Paramus, NJ: Rinton Press.

Keane, M., Ledgeway, T., & Duff, S. (1994). Constraints on analogical mapping: A comparison of three models. *Cognitive Science*, *18*, 387–438.

Knauff, M., Rauh, R., & Schlieder, C. (1995). Preferred mental models in qualitative spatial reasoning: a cognitive assessment of Allen's calculus. In *Proceedings of the seventeenth annual conference of the cognitive science society* (pp. 200–205). Mahwah, NJ: Lawrence Erlbaum.

Kokinov, B., & Petrov, A. (2000). Dynamic extension of episode representation in analogy-making in AMBR. In *Proceedings of the 22nd annual meeting of the cognitive science society.* Mahwah, NJ: Lawrence Erlbaum.

Kolodner, J. (1993). *Case-based reasoning.* San Mateo, CA: Morgan Kaufmann.

Kuipers, B. (1994). *Qualitative reasoning: Modeling and simulation with incomplete knowledge.* Cambridge, MA: MIT Press.

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in soar: The anatomy of a general learning mechanism. *Machine Learning*, *1*, 11-46.

Larkin, J., & Simon, H. (1987). Why a diagram is (somtetimes) worth ten thousand words. *Cognitive Science*, *11*, 65-99.

Lenat, D. B. (1995). CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, *38*(11), 33–38.

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)* (pp. 667–677). San Francisco, CA: Morgan Kaufmann.

McCarthy, J. (1998). Programs with common sense. In M. Minsky (Ed.), *Semantic information processing* (pp. 403–418). Cambridge, MA: MIT Press.

Minsky, M. (1975). A framework for representing knowledge. In P. Winston (Ed.), *The psychology of computer vision* (pp. 211–277). New York: McGraw-Hill.

Minton, S. (1985). Selectively generalizing plans for problem-solving. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI-85)* (pp. 596–599). San Francisco, CA: Morgan Kaufmann.

Mitchell, T. M. (1997). *Machine learning.* New York: McGraw-Hill.

Möbus, C. (1988). Zur Modellierung kognitiver Prozesse mit daten- bzw. zielorientierten Regelsystemen. In H. Mandl & H. Spada (Eds.), *Wissenspsychologie* (p. 423-465). München: PVU.

Newell, A. (1980). Physical symbol systems. *Cognitive Science*, *4*, 135-183.

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* (pp. 1–56). Hillsdale, NJ: Lawrence Erlbaum.

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

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

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, pp. 268–293). Heidelberg: Springer.

Opwis, K. (1992). *Kognitive Modellierung – Zur Verwendung wissensbasierter Systeme in der psychologischen Theoriebildung.* Bern: Huber.

Opwis, K., & Plötzner, R. (1996). *Kognitive Psychologie mit dem Computer.* Heidelberg: Spektrum.

Paulson, L. C. (1989). The foundation of a generic theorem prover. *Journal of Automated Reasoning*, *5*, 363–397.

Potts, G. (1972). Information processing strategies used in the encoding of linear orderings. *Journal of Verbal Learning and Verbal Behavior*, *11*, 727-740.

Quillian, M. (1968). Semantic memory. In M. Minsky (Ed.), *Semantic information processing* (pp. 216–270). Cambridge, MA: MIT Press.

Reiter, R. (1980). A logic for default reasoning. *Artificial Intelligence*, *1*, 81-132.
Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *Journal of the ACM*, *12*(1), 23-41.
Rosch, E. (1975). Cognitive representations of semantic categories. *Journal of Experimental Psychology*, *104*, 192-233.
Rumelhart, D. E., & Norman, D. A. (1981). Analogical processes in learning. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition* (pp. 335–360). Hillsdale, NJ: Lawrence Erlbaum.
Russell, S. J., & Norvig, P. (2002). *Artificial intelligence. A modern approach (2nd edition).* Englewood Cliffs, NJ: Prentice-Hall.
Sacerdoti, E. (1977). *A structure for plans and behavior.* Amsterdam: North-Holland.
Schlieder, C. (1999). The construction of preferred mental models in reasoning. In G. Rickheit & C. Habel (Eds.), *Mental models in discourse comprehension and reasoning* (pp. 333–358). Amsterdam: Elsevier.
Schlieder, C., & Behrendt, B. (1998). Mental model construction in spatial reasoning: A comparison of two computational theories. In U. Schmid, J. Krems, & F. Wysotzki (Eds.), *Mind modelling: A cognitive science approach to reasoning, learning, and discovery* (pp. 133–162). Lengerich: Pabst.
Schmalhofer, F., & Polson, P. G. A. (1986). A production system model for human problem solving. *Psychological Research*, *48*, 113-122.
Schmid, U. (2002). Computermodelle des Problemlösens. In J. Müsseler & W. Prinz (Eds.), *Allgemeine Psychologie* (pp. 701–734). Heidelberg: Spektrum Verlag.
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* (pp. 631–636). Mahwah, NJ: Lawrence Erlbaum.
Schmid, U., & Wysotzki, F. (2000a). Applying inductive programm synthesis to macro learning. In *Proc. 5th International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2000)* (pp. 371–378). Menlo Park, CA: AAAI Press.
Schmid, U., & Wysotzki, F. (2000b). A unifying approach to learning by doing and learning by analogy. In N. Callaos (Ed.), *4th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2000)* (Vol. 1, pp. 379–384). Orlando, FL: IIIS.
Schöning, U. (1992). *Logik für informatiker.* Mannheim: BI Wissenschaftsverlag.
Shortliffe, E. H. (1976). *Computer-based medical consultations: Mycin.* New York: Elsevier.
Strube, G. (Ed.). (1996). *Wörterbuch der Kognitionswissenschaft.* Stuttgart, Germany: Klett-Cotta.
Strube, G. (2000). Generative theories in cognitive psychology. *Theory & Psychology*, *10*, 117-125.
Strube, G. (in press). Cognitive modeling. In *International encyclopedia of the social and behavioral sciences.* Oxford: Elsevier.
Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning – an introduction.* Cam-

bridge, MA: MIT Press.

Tergan, S. O. (1986). *Modelle der Wissensrepräsentation als Grundlage qualitativer Wissensdiagnostik.* Opladen: Westdeutscher Verlag.

Tversky, A., & Kahneman, D. (1983). Extensional versus intuitive reasoning: The conjunction fallacy in probability judgement. *Psychological Review, 90*(4), 293–315.

VanLehn, K. (Ed.). (1991). *Architectures for intelligence.* Hillsdale, NJ: Lawrence 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.

Wason, P., & Johnson-Laird, P. (1972). *Psychology of reasoning: Structure and content.* Cambridge, MA: Harvard University Press.

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.

Zadeh, L. (1982). A note on prototype theory and fuzzy sets. *Cognition*, *12*, 291-297.