Zusammenspiel von Problemlösestrategien und Theorie-Entwicklung: ein Ansatz zur Simulation

Urban, Josef

Veröffentlichungsversion / Published Version
Antrag, Vorstudie / application

Zusammenspiel von Problemlösestrategien und Theorie-Entwicklung: Ein Ansatz zur Simulation

Hausarbeit
zur Erlangung des Magistergrades
im Fach
Logik und Wissenschaftstheorie
an der
Ludwig-Maximillians-Universität München

Vorgelegt von
Josef Urban

Hauptreferent:
Prof. Dr. W. Balzer

München
September 1995

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Die vorliegende Arbeit beschreibt einen Ansatz zur rechnergestützten Simulation der Entwicklung von Theorien unter dem Einfluß von Problemlösestrategien.

Im Rahmen dieser Arbeit sind deshalb folgende Aspekte besonders interessant:

1. Die computer-basierte Simulation als Methode zur Analyse der Prozesse, gemäß denen die Entwicklung wissenschaftlicher Theorien erfolgen kann.
2. Der Begriff einer wissenschaftlichen Theorie.
3. Die Problemarten, die mit solchen Theorien gegeben sind und die im Rahmen einer Forschertätigkeit zu lösen sind.
4. Die Strategien, nach denen solche Probleme gelöst werden können und die damit den Entwicklungsprozeß von wissenschaftlichen Theorien mitprägen.

Auf dem Gebiet der Wissenschaftstheorie und der Wissenschaftgeschichte existieren verschiedene Ansätze, die die Entwicklung von wissenschaftlichen Disziplinen zu beschreiben und zu erklären versuchen. Zwei wesentliche Ansätze sind der Induktivismus (F. Bacon, J. S. Mill) und der Falsifikationismus (K. R. Popper). Der Induktivismus geht davon aus, daß die Wissenschaft von elementaren Beobachtungen ausgeht und auf induktivem Wege zu einfachen Verallgemeinerungen und schließlich zu allgemeingültigen Gesetzen gelangt. Nach dem Falsifikationismus besteht die wissenschaftliche Vorgehensweise darin, daß provisorisch angenommene Hypothesen mit Hilfe einer deduktiven Logik überprüft werden und man diese Hypothesen beibehält, solange sie der Überprüfung standhalten.

Diese Ansätze können als zu unpräzise und zu praxisfern kritisiert werden. Sie beschreiben nicht notwendigerweise den tatsächlichen Ablauf der wissenschaftlichen Arbeit und lassen sich nicht durch historisch belegt Beispiele untermauern.

Gegen diese Ansätze wendet sich auch T. S. Kuhn ([Kuh67]). Nach Kuhn basiert wissenschaftliche Arbeit überwiegend auf Auffassungen, die von der jeweiligen Forschergemeinschaft allgemein akzeptiert sind. Diese als Paradigmata bezeichneten Auffassungen geben die Leitlinie für die Forschung vor und bestimmen, welche Experimente in welcher Form durchgeführt werden. Die Entwicklung der Wissenschaft wird durch revolutionäre Prozesse geprägt, in deren Rahmen der Wechsel eines solchen Paradigmas stattfinden kann. Leider bleibt der zentrale Begriff des Paradigmas bei Kuhn zu vage.

Auch bei anderen Ansätzen, wie z. B. bei L. Laudan, der die Wissenschaft als problem-lösenden Prozeß betrachtet (siehe [Lau77]), sind wesentliche Aspekte nicht präzise genug erläutert. So wäre dort etwa eine detailliertere Beschreibung der Struktur des Problemlösens wünschenswert.

Eine klarere Darstellung solcher Begriffe kann mit dem strukturalistischen Ansatz erzielt werden (siehe [BMS87])[1]. Eine empirische Theorie wird dort durch ihre mathematische Grundstruktur und durch ihren Anwendungsbereich charakterisiert. Die Angabe der Struktur einer Theorie erfolgt durch ein mengentheoretisches Prädikat. Der Anwendungsbereich wird durch die Menge der intendierten Anwendungen festgelegt, die durch die Anführung von paradigmatischen Beispielen umrissen ist.

Auf der Basis dieses formal beschriebenen Theoriebegriffs lassen sich auch andere Aspekte beschreiben, die bei der Entwicklung von Theorien eine Rolle spielen. Z. B. können damit Problemarten definiert werden, die im Verlauf einer Theorie-Entwicklung zu lösen sind. Insbesondere kommt die systematische, mathematische Beschreibungsweise des Strukturalismus der Formulierung solcher Aspekte in einer Programmiersprache sehr entgegen. Eine solche Programmierung wiederum ist die Voraussetzung für die Simulation des Entwicklungsprozesses von Theorien auf einem Computer.

Der Einsatz von Computern auf dem Gebiet der Wissenschaftstheorie wird bereits bei [Tha93] oder [Lo87] als sehr vielversprechend erachtet. Als Vorteil wird gesehen, daß mit den Programmiersprachen ein systematisches Vokabular gegeben ist, um Strukturen und Abläufe beschreiben zu können. Theorien der Wissenschaftstheorie können in ablauffähige Computerprogramme abgebildet werden. Durch einen erfolgreichen Programmablauf wird z. B. ihre interne Konsistenz getestet. Es besteht außerdem die Möglichkeit, daß auf diese Weise nicht vorhergesehene

[1] In [BMS87] wird u. a. gezeigt, daß mit dem Theoriebegriff des Strukturalismus auch der Kuhnsche Paradigmabegriff mathematisch genau beschrieben werden kann.

Konsequenzen aus den im Computerprogramm abgebildeten Hypothesen erkannt werden.

Um Simulationen auf einem Rechner durchführen zu können, wird ein Simulationsmodell benötigt, das, formuliert in einer geeigneten Programmiersprache, die relevanten Aspekte repräsentiert, die bei der Entwicklung von Theorien eine Rolle spielen. In der vorliegenden Arbeit wird bei der Beschreibung von Theorien auf den strukturalistischen Theorienbegriff zurückgegriffen.

Mit diesem Theoriebegriff können Problemarten definiert werden, die es im Verlauf der wissenschaftlichen Entwicklung zu lösen gibt. Solche Problemarten erwachsen aus der Behauptung, daß eine Anwendung zu einem Modell einer Theorie paßt und diese Passung sich konsistent zu anderen Theorien verhält.

Mit dem strukturalistischen Theoriebegriff und den Problemarten ist eine Beschreibung des statischen Zustandes einer Theorie gegeben. Auf dieser Basis können nun Strategien formuliert werden, die die Lösung der Probleme steuern. Eine solche Strategie gibt Regeln an, die bestimmen, welche Probleme als nächstes zu lösen versucht werden und welche Konsequenzen sich aus dem positiven oder negativen Ergebnis eines Lösungsversuchs resultieren. Diese Konsequenzen führen zur Veränderung der betroffenen Theorien und bewirken damit ihre dynamische Entwicklung.

Im Rahmen eines Simulationsmodells können solche Strategien aufgestellt und ihr Verlauf anhand verschiedener Beispiele verfolgt werden.

1.2 Inhalt der Arbeit

In dieser Arbeit wird ein Ansatz aufgezeigt, bei dem die Entwicklung von Theorien unter dem Einfluß verschiedener Problemlösestrategien durch den Einsatz von computer-basierter Simulation untersucht werden kann. Ein entsprechendes Simulationsprogramm wurde als Prototyp entwickelt.

Die vorliegende Ausarbeitung erläutert die verschiedenen Aspekte, die bei der Erstellung des Simulationsmodells zu berücksichtigen waren. Das Simulationsprogramm selbst wird hier nur in seiner Grobstruktur beschrieben. Die vollständige Dokumentation ist mit dem Programmtext und den darin enthaltenen Kommentaren gegeben.

In Kapitel 2 wird Simulation als das Experimentieren mit Modellen charakterisiert. Zur Erläuterung wird die Entwicklung von Simulationsmodellen im allgemeinen betrachtet und schließlich auf solche Modelle eingegangen, die auf einem Rechner realisiert werden. Bei computer-basierten Simulationen spielen Zufallszahlen eine wichtige Rolle. Deshalb werden in diesem Kapitel auch Verfahren

beschrieben, mit denen Zufallszahlen erzeugt werden. Schließlich wird eine kurze Einführung in die Programmiersprache Prolog gegeben, in der das Simulationsmodell implementiert wurde.

Die exakte Definition des verwendeten Theoriebegriffs wird in Kapitel 3 angeführt. Dort ist auch beschrieben, bis zu welchem Abstraktionsgrad dieser Theoriebegriff im Simulationsmodell repräsentiert werden konnte. Es zeigt sich nämlich, daß eine sehr detaillierte Darstellung, die z. B. die mathematischen Gesetze einer konkreten Theorie umfaßt, sehr bald die verfügbaren Kapazitäten eines durchschnittlichen PC's übersteigt.

Kapitel 4 enthält die Definitionen der Problemarten. Für die Realisierung im Simulationsmodell wurden einige grundlegende Problemarten ausgewählt. Ihre Realisierung im Simulationsmodell auf der Basis des dort abgebildeten Theoriebegriffs wird am Ende dieses Kapitels erläutert.

In Kapitel 5 wird beispielhaft eine Problemlösestrategie entworfen und ihre Realisierung beschrieben. Dabei wurde versucht, die statische Struktur der Strategie durch einzelne Regeln auszudrücken. Beim Simulationslauf entfaltet sich durch das Zusammenspiel dieser Regeln der komplexe Ablauf der Strategie.

Das achte Kapitel enthält einen Überblick über den Aufbau des Simulationsmodells und wie sich die in den vorangehenden Kapiteln beschriebenen Komponenten (Theorie, Problemarten, Strategie) in das Modell einfügen.

Im letzten Kapitel schließlich wird ein Ausblick gegeben, in welcher Art und Weise der Prototyp des Simulationsmodells weiterentwickelt werden kann.

Kapitel 2

Simulation

Nicht immer können die für eine Problemlösung benötigten Informationen durch unmittelbares Experimentieren mit dem realen System gewonnen werden, da dies zu gefährlich, zu aufwendig oder einfach unmöglich ist. Deshalb bildet man die für die gegebene Problemstellung relevanten Aspekte des realen Systems in einem Modell nach und führt Untersuchungen an diesem Modell durch. Die dabei gewonnenen Einsichten können auf das reale System übertragen werden und lassen Rückschlüsse auf dessen Verhalten zu.

Eines der Verfahren, gemäß dem die Untersuchungen an einem Modell vorgenommen werden kann, besteht in der Simulation. Unter Simulation ist das zielgerichtete Experimentieren mit Modellen zu verstehen. Das dabei verfolgte Ziel ist, Aussagen und Prognosen über das Verhalten eines bestehenden oder erst zu entwickelnden Systems zu erhalten, um mit dieser Erkenntnis ein gegebenes Problem zu lösen.

Bei der Simulation wird hierbei das Modell in eine Form gebracht, die es erlaubt, die beschriebenen Einflußgrößen zu verändern und die Auswirkungen dieser Veränderungen am Modell zu beobachten. Ein solches Modell wird als Simulationsmodell bezeichnet. Die Simulation ist mit diesem Vorgehen häufig auch in solchen Fällen noch anwendbar, in denen andere, rein mathematische, analytische Verfahren zur Untersuchung eines Modells nicht mehr einsetzbar sind.

In diesem Kapitel wird zunächst allgemein der Prozeß skizziert, der zur Bildung eines Modells führt. Es folgt eine Gegenüberstellung von analytischen und simulativen Verfahren, die sich solcher Modelle bedienen. Anschließend wird eine Klassifikation von Modellen angegeben. Diese bietet Kriterien, nach denen das in dieser Arbeit erstellte Simulationsmodell als computer-basiertes Modell charakterisiert werden kann. Die Eigenschaften von computer-basierten Modellen und deren Entwicklung wird näher betrachtet. Diese Ausführungen geschehen im wesentlichen auf der Grundlage von [Sch85] und [FG90].

Die hier vorgestellte Modellierung ist so, daß im Simulationsmodell einige Elemente als zufällige Ereignisse dargstellt werden. Deshalb wird kurz auf die Erzeugung von Zufallszahlen eingegangen.

Schließlich wird ein Überblick über Prolog gegeben, die Programmiersprache, in der das hier erstellte Simulationsmodell implementiert ist.

2.1 Modellbildung

Um ein gegebenes Problem in einem realen System zu lösen, muß in der Regel das Verhalten dieses Systems besser verstanden werden. Im Allgemeinen verfolgt der entsprechende Prozeß zur Lösung des Problems das Ziel, Erkenntnisse über ein bestehendes System zu gewinnen oder das Verhalten neu zu entwickelnder Systeme zu untersuchen. Im Detail können damit z. B. folgende Zielrichtungen verfolgt werden (vgl. auch [MM89]):

- Ein Systemablauf soll optimiert werden; z. B. durch die Ermittlung von Engpässen in einem Produktionsprozeß.
- Theorien über ein im Detail unbekanntes System sind zu überprüfen.
- Das Verhalten eines Systems (z. B. das Wetter oder die Umwelt) ist zu prognostizieren.
- Eine Entscheidungshilfe bei der Plannung eines Systems ist gesucht.

Die Modellbildung kann nun einen solchen Problemlöseprozeß unterstützen. Die Untersuchungen werden am Modell und nicht am realen System vorgenommen. Besonders interessant ist die Modellbildung dann, wenn unmittelbare Untersuchungen am realen System nicht durchführbar sind. Gründe dafür können z. B. sein:

- Das zu untersuchende System ist nicht zugänglich (z. B. Planeten).
- Die Vorgänge in der Realität laufen zu schnell oder zu langsam ab (z. B. bei Evolutionsprozessen).
- Die Kosten der Realisierung eines neuen Systems sind zu hoch (z. B. bei technischen Systemen wie einem Flugzeug).
- Die Versuche im realen System sind zu gefährlich (z. B. in der Chemie).

Modellbildung ist nun die Aktivität, die von einem realen System zu einem Modell führt. Sie ist gekennzeichnet durch Reduktion, Abstraktion und Idealisierung. Die Sicht auf ein reales System wird dabei auf die Aspekte reduziert, die das Systemverhalten wesentlich mitbestimmen und die relevant sind für die Problemstellung,

zu dessen Lösung die Modellbildung beitragen soll. Das Modell wird somit gegen seine Umwelt abgegrenzt. Diese Reduktion auf die wesentlichen Aspekte wird von einem Abstraktionsvorgang begleitet, der die konkreten Objekte des realen Systems bis zu einem gewissen Grade verallgemeinert und zusammenfaßt. Dieser Vorgang wird durch Idealisierungen unterstützt. Dabei werden reale Gegebenheiten auf gedachte, ideale Größen abgebildet, wie dies z. B. in der klassischen Mechanik mit der Einführung eines Massepunkts geschieht.

Auf diesem Weg entsteht ein abstraktes Modell, mit dem eine aufbereitete und vereinfachte Darstellung eines realen Systems gegeben ist. Eine explizite Beschreibung eines abstrakten Modells erfolgt zunächst gewöhnlich mit Hilfe einer weitgehend informellen Sprache.

Um Aussagen über das Verhalten des auf diese Weise modellierten Systems zu erhalten, kommen zwei grundsätzlich verschiedene Verfahren zum Einsatz: analytische und simulative.

Der Einsatz analytischer Verfahren setzt voraus, daß das abstrakte Modell als formales Modell vorliegt, also in einer formalen Sprache abgefaßt ist. Der mit einer solchen formalen Sprache gegebene Kalkül erlaubt es dann, neue, allgemeingültige Aussagen über das Verhalten des Modells abzuleiten. Ein Beispiel hierfür ist die Darstellung zeitkontinuierlicher abstrakter Modelle mit Hilfe von Differentialgleichungen. Eine exakte, analytische Lösung der Gleichungen kann in einfachen Fällen mit mathematischen Mitteln erreicht werden.

Oft ist die Verwendung von analytischen Verfahren jedoch nicht möglich. Ein Hauptgrund dafür besteht darin, daß für die Analyse notwendige Informationen nicht im abstrakten Modell enthalten sind, weil z. B. deren Ermittlung am realen System die Zerstörung desselben zur Folge hätte. Ein anderer Grund ist in der hohen Komplexität eines Systems gegeben. Formale, analytische Methoden können dort nicht mit vertretbaren Aufwand oder prinzipiell nicht mehr eingesetzt werden. In diesen Fällen können simulative Verfahren zu dem gewünschten Erkenntnisgewinn führen.

Simulation ist ein Verfahren, bei dem ein zweites reales System als vereinfachtes Abbild des zu untersuchenden realen Systems erstellt wird und Experimente an diesem Ersatzsystem vorgenommen werden. Dieses zweite System wird als reales Modell oder als *Simulationsmodell* bezeichnet.

Ein Beispiel für eine Simulation ist die Ermittlung des Luftwiderstandes eines Flugzeuges im Windkanal. Das abstrakte Modell des Flugzeuges ist im wesentlichen durch die Abmessungen und das Material der Außenwände bestimmt. Das Simulationsmodell schließlich besteht in einem maßstabsgetreu verkleinerten Modell, aufgebaut aus dem gleichen Konstruktionsmaterial. In Experimenten können Konstruktionsalternativen ohne größeren Aufwand an diesem Flugzeugmodell un-

tersucht und entsprechende Messungen durchgeführt werden, die dann Aussagen über die Aerodynamik des geplanten realen Flugzeuges zulassen.

Ein Simulationsmodell und das damit imitierte reale System beziehen sich also auf das gleiche abstrakte Modell.

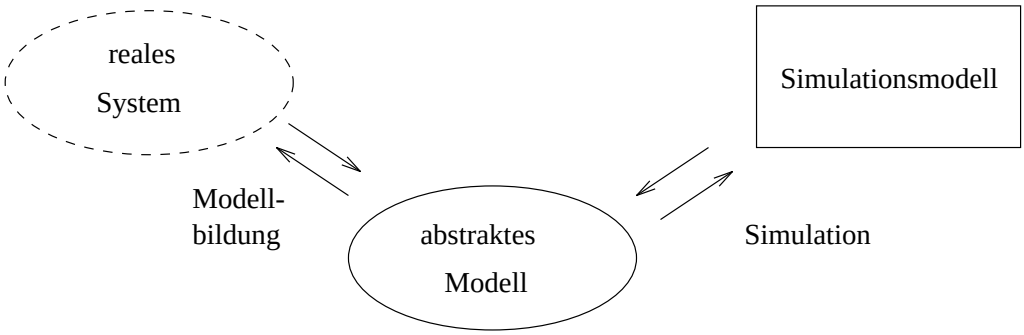


Abbildung 2.1: Modellbildung und Simulation

Der gesamte Prozeß, der von einem realen System zu einem Simulationsmodell führt, ist in Abbildung 2.1 skizziert. Hier wird der Unterschied zwischen Modellbildung und Simulation noch einmal verdeutlicht.

Modellbildung beschäftigt sich mit der Beziehung zwischen einem realen System und seinem abstrakten Modell. Dies umfaßt zum einen den Modellaufbau, der in der Abbildung 2.1 durch den Pfeil vom realen System zum abstrakten Modell dargestellt ist. Zum anderen aber, dargestellt durch den entgegengerichteten Pfeil, werden Analyseergebnisse auf das reale System angewendet. Durch Vergleich der Analyseergebnisse des abstrakten Modells mit Daten des realen Systems erfolgt die Validierung des abstrakten Modells.

Mit Simulation dagegen wird die Realisierung des abstrakten Modells in einem Simulationsmodell und das Experimentieren mit diesem Modell bezeichnet, dargestellt durch den Pfeil vom abstrakten Modell zum Simulationsmodell. Dem entgegengerichteten Pfeil folgend werden die Simulationsergebnisse gemäß dem abstrakten Modell interpretiert und können unter Umständen die Änderung des Simulationsmodells und auch des abstrakten Modells nach sich ziehen. Über diese Wechselbeziehungen zwischen Simulationsmodell und abstraktem Modell wird das Simulationsmodell verifiziert, d. h. es wird der Nachweis geführt, daß das Simulationsmodell das abstrakte Modell in befriedigender Weise realisiert.

Das abstrakte Modell spiegelt sich im Simulationsmodell nur innerhalb gewisser Toleranzgrenzen wieder. Ähnlich wie bei der Darstellung des realen Systems im abstrakten Modell, ist der Übergang vom abstrakten Modell zu einem Simulationsmodell approximativ. In der Regel sind nämlich durch die Realisierung eines Simulationsmodells zum einen wieder mehr Einflußgrößen gegeben, als im

abstrakten Modell explizit berücksichtigt. Zum anderen müssen im Simulationsmodell die im abstrakten Modell vorgenommenen Idealisierungen konkretisiert werden.

Ist für ein abstraktes Modell sowohl eine analytische Lösung als auch dessen Realisierung in einem Simulationsmodell gegeben, so wird man deshalb im allgemeinen feststellen, daß die Ergebnisse der beiden Verfahren nicht identisch sind.

Auch in ihrer Allgemeingültigkeit und Aussagekraft unterscheiden sich die Ergebnisse der beiden Verfahren. Analytische Verfahren liefern allgemeine Strukturaussagen über das Modellverhalten. In den Lösungen kommen z. B. variable Koeffizienten vor. Die damit gegebenen Aussagen gelten somit für jede beliebige Belegung dieser Variablen. Bei der Simulation dagegen wird ein konkretes reales Modell untersucht und es können damit primär nur singuläre Ergebnisse erzielt werden, die sich auf diese individuellen Modelle beziehen. Um zu fundierten Ergebnissen zu gelangen, bedarf es wiederholter Simulationsläufe, bei denen die Modellparameter mit unterschiedlichen Werten versorgt werden.

2.2 Klassifikation von Simulationsmodellen

Die nachfolgend angegebene Klassifikation versucht eine Einteilung hinsichtlich des „Mediums“, mit dem ein abstraktes Modell als Simulationsmodell realisiert wird.

Wie das obige Beispiel mit dem Flugzeugmodell zeigt, können Modelle physikalisch realisiert sein. Die Simulation erfolgt in diesem Fall mit Hilfe von materiellen Körpern, auf die die natürlichen physikalischen Gesetze einwirken.

Dieser Klasse der physikalischen Modelle steht die Klasse der ideellen Modelle gegenüber, die sich hinsichtlich ihrer Repräsentation weiter in graphische und symbolische Modelle unterteilen läßt (Abbildung 2.2).

Ideelle Modelle sind von einer direkten physikalischen Realisierung losgelöst und nur in der gedanklichen Vorstellung verankert. Eine explizite Darstellung ideeller Modelle kann durch Graphiken oder mit den Mitteln formaler Sprachen erfolgen.

Ein Beispiel für ein graphisches Modell wäre eine Landkarte, mit deren Hilfe verschiedene Fahrtrouten gemessen und auf ihre Länge hin verglichen werden können, ohne die Fahrten tatsächlich durchführen zu müssen. Symbolische Modelle sind z. B. mit jedem mathematischen Modell gegeben.

Ein wesentliches Merkmal der ideellen Modelle ist, daß mit ihnen und ihren expliziten Darstellungen Verfahren verbunden sind, durch deren Anwendung auf

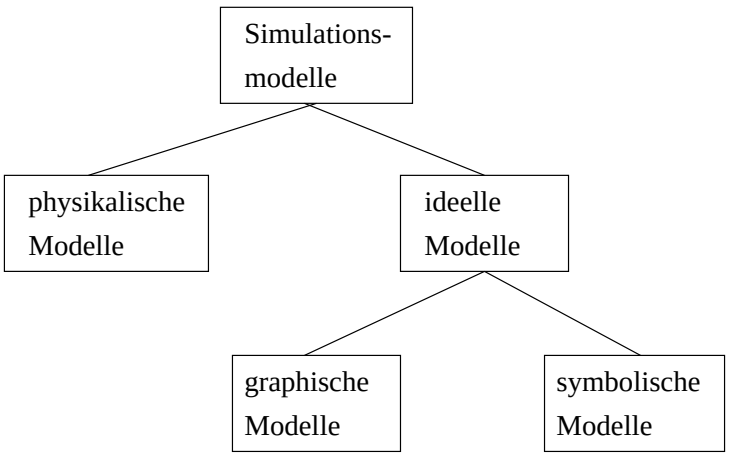


Abbildung 2.2: Klassifikation von Simulationsmodellen

das Modell weitergehende Aussagen über das dynamische Verhalten des modellierten realen Systems abgeleitet werden können. Zur Simulation von auftragsverarbeitenden Systemen, wie z. B. dem Prozessor eines Computers, kommen etwa statistische Methoden der Warteschlangentheorie zum Einsatz.

Die Anwendung solcher Verfahren, die mit symbolischen Modellen verknüpft sind, kann z. B. mit Hilfe von Papier und Bleistift erfolgen. Das wohl verbreiteste Hilfsmittel ist mittlerweile jedoch der Computer.

Als symbolisches Modell entsteht dabei ein Computer-Modell, bei dem das abstrakte Modell mit den Mitteln einer formalen Sprache, nämlich einer Programmiersprache, beschrieben ist. Das in dieser Arbeit erstellte Simulationsmodell ist demgemäß unter die symbolischen Modelle einzuordnen.

Ein abstraktes Modell, das in einer Programmiersprache beschrieben ist, wird in [FG90] als ‘algorithmisches Modell‘ bezeichnet. Das damit verbundene Verfahren, auf dem die Simulation des Modellverhaltens basiert, ist mit einem Algorithmus, einem Satz von Regeln gegeben. Das wesentliche Merkmal eines solchen Modells besteht also darin, daß seine sprachlichen Konstrukte auf einem Computer ausführbar sind. Algorithmische Modelle stellen somit eine Folge von Anweisungen dar, die von dem physikalischen Medium eines Computers bearbeitet werden können[1].

Die technischen Möglichkeiten eines Computers sind so vielfältig, daß er sogar ein generelles Hilfsmittel für jede Art von Simulation darstellen kann. So sind die graphischen Mittel eines Computers bei weitem ausreichend, um graphische Modelle zu realisieren. Durch zusätzliche Animationstechniken können sogar Si-

[1]Unter diesem Blickwinkel könnten Computermodelle auch als physikalische Modelle betrachtet werden, wie dies in [Sch85] erfolgt.

mulationsmodelle, die als physikalische Modelle zu realisieren wären, ebenfalls am Computer ablaufen[2].

Anzumerken ist noch, daß keine direkte äußere Ähnlichkeit zwischen einem Symbolmodell und dem damit modellierten realen Systems existieren muß. Bei physikalischen Modellen ist eine solche Ähnlichkeit in der Regel gegeben, da das Modell eine maßstabsgetreue Verkleinerung oder Vergrößerung des realen Systems darstellt. Bei Symbolmodellen und damit auch bei computer-basierten Simulationsmodellen dagegen gibt es keine direkte Abbildung der Modellstruktur auf das reale System. Bei computer-basierten Simulationsmodellen erfolgt die Strukturierung auch nach programmtechnischen Gesichtspunkten und muß kein Abbild des modellierten realen System sein.

2.3 Computer-basierte Simulationsmodelle

Als computer-basierte Simulationsmodelle werden nun solche Simulationsmodelle bezeichnet, die mit Hilfe eines Computers realisiert sind. Die Objekte des abstrakten Modells und deren Beziehungen untereinander werden in einer Sprache beschrieben, deren Komponenten von einer Maschine interpretiert werden können. Mit dieser maschinellen Interpretation äußert sich dann das Verhalten des Simulationsmodells.

Der Übergang vom abstrakten Modell zu einem computer-basierten Modell kann in zwei Phasen unterteilt werden. In einer Spezifikationsphase wird das abstrakte Modell in ein algorithmisches Modell überführt und eine Datenumgebung definiert, aus der heraus die Parameter des algorithmischen Modells mit konkreten Werten versorgt werden können. In der Implementierungsphase wird das algorithmische Modell zusammen mit seiner Datenumgebung als ablauffähiges Programm erzeugt. In dieser Phase sind in einer Ablaufumgebung die technischen Realisierungsdetails eines Computerprogramms zu berücksichtigen, wie z. B. die Verwaltung der Betriebsmittel eines Computers (Prozessor, Speicher, Ein- und Ausgabegeräte usw.). Erst mit dem ablauffähigen Programm ist das computerbasierte Simulationsmodell gegeben (Abbildung 2.3).

In der Spezifikationsphase erfolgt also die Algorithmisierung des abstrakten Modells. Es entsteht damit eine problemnahe Beschreibung des abstrakten Modells mit den Mitteln einer formalen Spezifikationssprache. Solche Spezifikationssprachen abstrahieren noch weitgehend von den technischen Details der eigentlichen Ziel-

[2]Steht vorrangig der Computer und damit die technische Realisierung eines Simulationsmodells im Vordergrund, so verschwinden die oben aufgeführten Grenzen der Klassifikation. Für die vorliegende Arbeit sollten diese technischen Aspekte jedoch weitgehend im Hintergrund gehalten werden.

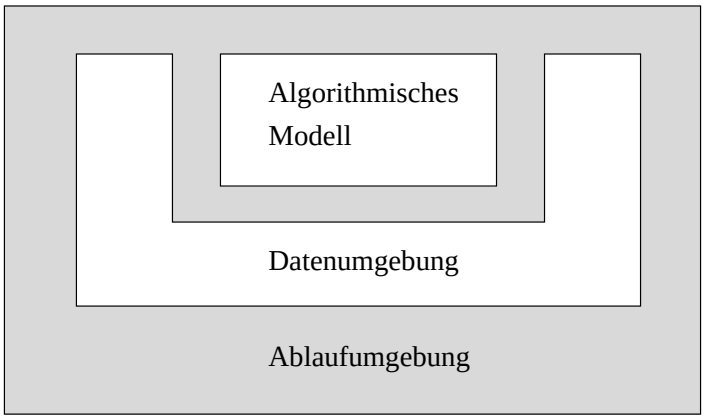


Abbildung 2.3: Computer-basiertes Simulationsmodell

maschine. Man kann sagen, daß das damit beschriebene Modell „prinzipiell“ ausführbar ist, es ist auf einer Gedankenmaschine ablauffähig (siehe [BW84]).

Eine der wesentlichen Aufgaben in dieser ersten Phase besteht nun darin, eine geeignete Strukturierung des algorithmischen Modells zu finden, die es erlaubt, die zeitlichen Abläufe des modellierten Systems und die Zeit selbst darzustellen.

Erfolgt die Modellierung so, daß sich das Simulationsmodell stetig in der Zeit verändert, so spricht man von kontinuierlicher Simulation. Das Systemverhalten ist in diesem Fall meist durch Differentialgleichungen charakterisiert, deren freie Variable die Zeit ist. Ausgehend von einem Anfangszustand kann der Zustand des Systems zu einem bestimmten Zeitpunkt errechnet werden. Eine explizite Strukturierung unter Berücksichtigung der zeitlichen Aspekte ist hier nicht notwendig.

Anders ist dies bei der diskreten Simulation, bei der sich der Zustand des Simulationsmodell sprunghaft zu diskreten Zeitpunkten ändert. Verschiedene Ansätze werden unterschieden, nach denen die Strukturierung eines diskreten Simulationsmodells vorgenommen werden kann (siehe [FG90] oder [MM89]). Einer der grundlegenden Ansätze besteht in der ereignisorientierten Simulation. Dieser Ansatz wurde auch in der vorliegenden Arbeit gewählt.

Das Verhalten des Modells wird bei diesem Ansatz durch das Eintreten ausgewählter Ereignisse bestimmt. Mit dem Eintreten eines Ereignisses erfolgt eine Veränderung des Modellzustandes und die Ermittlung des Ereignisses, das als nächstes eintreten soll. Die Simulationszeit wird auf den Eintrittszeitpunkt dieses nächsten Ereignisses fortgeschaltet und der Bearbeitungszyklus beginnt mit diesem Ereignis von neuem. Die Zeit zwischen zwei aufeinanderfolgenden Ereignissen, die im realen System verstreicht, wird bei der Simulation also übersprungen.

Um nun tatsächlich Simulationsläufe mit einem algorithmischen Modell durchführen zu können, muß eine Datenumgebung definiert werden, die die Parame-

ter des algorithmischen Modells mit konkreten Werten versorgt. Diese konkreten Werte bestimmen das Verhalten des Modells während eines Simulationslaufs.

Je nachdem, wie diese Parameterversorgung und Steuerung des Ablaufs erfolgt, wird von einem deterministischen, einem nicht-deterministischen oder einem stochastischen Simulationsmodell gesprochen. Bei deterministischen Modellen erfolgt die Versorgung der Parameter mit fest vorgegebenen Werten. Bei nicht-deterministischen Modellen dagegen werden die aktuellen Parameterwerte zufällig und bei stochastischen Modellen gemäß einer Wahrscheinlichkeitsverteilung erzeugt.

Das in dieser Arbeit erstellte Simulationsmodell kann unter die stochastischen Modelle eingeordnet werden. Wesentliche Parameter des Modells werden hier bei einem Simulationslauf mit Werten versorgt, die gemäß einer Gleichverteilung erzeugt werden (siehe Kapitel 2.4).

Der letzte Schritt zu einem lauffähigen, computer-basierten Simulationsmodell besteht in der Implementierung des Simulationsmodells. Das algorithmische Modell und seine Datenumgebung wird in eine Ablaufumgebung eingebettet und in eine maschinennahe Fassung gebracht; d. h. in einer konkreten Programmiersprache niedergeschrieben.

Damit man mit einem Modell experimentieren kann, muß es verändert werden können. Für ein computer-basiertes Simulationsmodell heißt das, daß das Simulationsprogramm modifiziert werden kann. Eine entsprechend gewählte Programmiersprache kann die Durchführung solcher Änderungen erleichtern. Die in dieser Arbeit verfolgte Idee ist, eine Programmiersprache zu verwenden, mit der bereits das algorithmische Modell und seine Datenumgebung in einer relativ problemnahen Form spezifiziert werden kann. Die Abbildung auf ein lauffähiges Programm erfolgt dann weitgehend mit Hilfe von Programmierwerkzeugen (z. B. Compiler, Interpreter), die mit einer solchen höheren Programmiersprache gegeben sind.

Der Weg vom abstrakten Modell zu einem ablauffähigen Simulationsprogramm wird damit verkürzt oder anders ausgedrückt, bereits die Spezifikation ist ausführbar. Auch Änderungen können damit auf der Abstraktionsebene des algorithmischen Modells und seiner Datenumgebung vorgenommen werden, ohne daß dies den Aufwand einer maschinennahen Implementierung nach sich zieht. Die Änderungen sind auch leichter zu verstehen und zu verifizieren, da sie in einer problemnahen Formulierung erfolgen.

Eine Programmiersprache, die diesen Ansatz unterstützt ist Prolog. Sie wurde auch in dieser Arbeit eingesetzt und wird in Kapitel 2.5 näher betrachtet.

2.4 Erzeugung von Zufallszahlen

Zur Lösung von stochastischen Simulationsproblemen werden Zufallszahlen benötigt. Ihre Erzeugung geschieht mit Hilfe von Zufallszahlengeneratoren. Es werden zwei Arten von Generatoren unterschieden, nämlich solche die echte Zufallszahlen erzeugen und solche die sogenannte Pseudozufallszahlen generieren.

Echte Zufallszahlen zeichnen sich dadurch aus, daß sie gewisse statistische Eigenschaften besitzen, wie z. B. daß sie einer bestimmten Verteilung genügen und voneinander unabhängig sind oder, daß keine Regelmäßigkeiten innerhalb der Zahlenabfolge existieren. Das Würfeln oder das Werfen einer Münze sind Beispiele für Generatoren, die solche Zahlen erzeugen. In der entsprechenden Fachliteratur sind Tabellen vorhanden, in denen bereits echte Zufallszahlen vorgegeben werden.

Für den Einsatz am Computer sind solche Methoden zur Generierung von echten Zufallszahlen nicht sinnvoll anwendbar. Sie beanspruchen zuviel Zeit oder sind wegen ihres hohen Speicherbedarfs nicht realisierbar. Deshalb werden in Computeranwendungen Zufallszahlen nach einem mathematischen Verfahren erzeugt. Damit ist jedoch kein echter Zufallsprozeß gegeben, da die erzeugte Zahlenfolge nach einem bestimmten Zyklus wieder ihren Anfangswert erreichen oder in einer konstanten Folge entarten kann. Deshalb spricht man hier auch von Pseudozufallszahlen. Die Unzulänglichkeit der Pseudozufallszahlen kann aber außer Acht gelassen werden, solange die erzeugte Zahlenfolge die relevanten statistischen Tests und damit eine gewisse Zufälligkeit in ausreichendem Maße erfüllt.

Von grundlegender Bedeutung ist die Generierung gleichverteilter Zufallszahlen. Aus ihnen können nämlich Zufallszahlen abgeleitet werden, die anderen Verteilungen genügen.

Die gebräuchlichste Methode zur Erzeugung von gleichverteilten Pseudozufallszahlen ist das Verfahren von Lehmer[3] (siehe [Knu81] oder [Ste94]). Gegeben sind natürliche Zahlen m, a, c und X_0 mit $m > 0$, $0 \leq a < m$, $0 \leq c < m$ und $0 \leq X_0 < m$. Die Folge der Zufallszahlen $\langle X_n \rangle, n = 0, 1, 2, \ldots$ wird damit nach folgender Formel erzeugt:

$$X_{n+1} = (a * X_n + c) \bmod m, \qquad n \geq 0$$

Gleichverteilte Zufallszahlen U_n im Intervall $(0; 1)$ erhält man durch

$$U_n = X_n / m.$$

Damit diese Methode befriedigende statistische Ergebnisse liefert und eine möglichst große Periode aufweist, nach der der Generierungszyklus von neuem beginnt,

[3]Das Verfahren ist auch unter der Bezeichnung lineares Kongruenzverfahren bekannt.

müssen die Werte von m, a, c und X_0 geeignet gewählt werden. m bestimmt die maximal mögliche Anzahl unterschiedlicher Pseudozufallszahlen und sollte deshalb möglichst groß gewählt werden. Eine obere Schranke für einen solchen Wert ist durch die Arithmetik des verwendeten Computersystems vorgegeben. Aufgrund der besonderen Eigenschaften der Hardwarearithmetik wird außerdem empfohlen, als Wert von m eine Zweierpotenz zu verwenden. Günstige Werte für die übrigen Parameter des Verfahrens lassen sich dann gemäß entsprechender Vorschriften (siehe z. B. [Sie91]) ermitteln.

Für die vorliegende Arbeit wurden auf diese Weise folgende Parameterwerte ermittelt:

$$\begin{aligned} m &= 2^{28} \\ a &= 16389 \\ c &= 56371447 \end{aligned}$$

Der Startwert X_0, der bei der Initialisierung des Verfahrens anzugeben ist, ist variabel. Er wird aus der aktuellen Systemzeit des Rechners ermittelt. Damit wird vermieden, daß mit jeder Serie, die mit einer Neuinitialisierung beginnt, die gleichen „Zufallszahlen“ erzeugt werden.

Eine Folge gleichverteilter Zufallszahlen $\langle Y_n \rangle$ aus einem beliebigen Intervall $(a; b)$ erhält man dann durch

$$Y_n = (b - a) * X_n + a,$$

wobei $\langle X_n \rangle$ eine Folge von gleichverteilten Zufallszahlen aus dem Intervall $(0; 1)$ ist.

2.5 Prolog als Simulationssprache

Prolog (PROgramming in LOGic) ist eine Programmiersprache, mit der Probleme in der Sprache der Logik beschrieben werden können. Die Ausführung eines Prolog-Programms kann als Suche nach einer Lösung des formulierten Problems gewertet werden.

Die Interpretation eines Prolog-Programms erfolgt damit auf einem relativ abstrakten Niveau. Auch die Programmentwicklung kann auf der Ebene einer abstrakten Gedankenmaschine erfolgen, die frei ist von den technischen Realisierungsdetails eines konkreten Computers (vgl. Seite 12).

Auch in [FG90] wird die Realisierung von Simulationsmodellen auf der Basis von Prolog vorgenommen. Zudem erwiesen sich logik-basierte Sprachen bereits in [Tha93] oder in [Lo87] als geeignetes Mittel zur Beschreibung von Theorien und von Problemlösestrategien.

Im Rahmend dieser Arbeit wird nur ein kurzer Überblick über Prolog gegeben. Damit soll ein gewisses Grundverständnis über Prolog vermittelt werden, soweit dies für die Beschreibung des hier erstellten Simulationsmodells notwendig ist. Eine ausführliche Sprachbeschreibung ist z. B. in [CM87], [SS86] oder [BS86] enthalten.

2.5.1 Syntax

Die grundlegende Datenstruktur in Prolog ist ein Term. Diese Datenstruktur bildet den syntaktischen Baustein, aus dem ein Prolog-Programm aufgebaut ist.

Ein Term kann sein (Abbildung 2.4):

- eine Konstante,
- eine Variable oder
- ein zusammengesetzter Term, eine sogenannte Struktur.

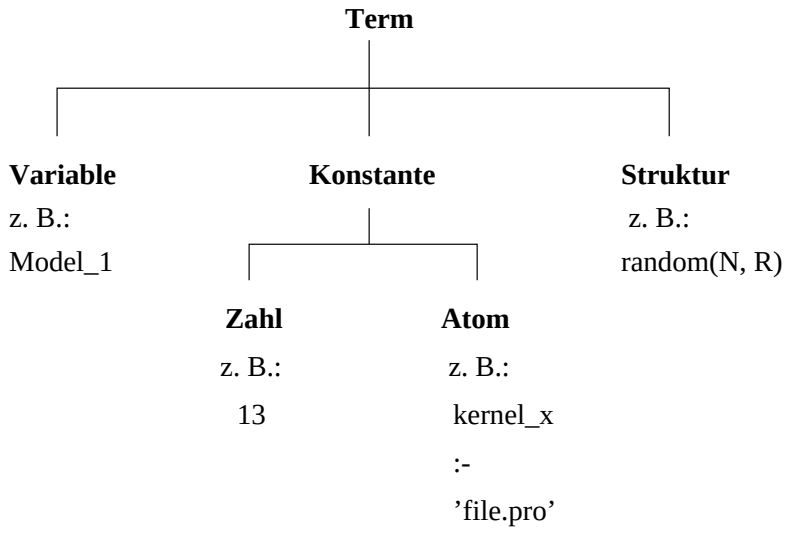


Abbildung 2.4: Die Prolog-Datenstruktur Term

Eine Konstante ist entweder eine ganze Zahl oder ein sogenanntes Atom. Ein Atom wiederum ist:

- ein Bezeichner, der mit einem Kleinbuchstaben beginnt und gefolgt wird von einer Sequenz von Buchstaben, Ziffern und Unterstrichen ('_'), oder
- eine Zeichenfolge, die aus bestimmten Symbolen (z. B. !, # oder $) besteht, oder

- eine beliebige Zeichenfolge, die in einfachen Hochkommata eingeschlossen ist.

Eine Variable ist daran zu erkennen, daß ihr Bezeichner mit einem Unterstrich oder mit einem Großbuchstaben beginnt, gefolgt von einer Sequenz von Buchstaben, Ziffern und Unterstrichen.

Eine Struktur besteht aus einem Funktor und einer nachfolgenden Liste von Argumenten. Der Funktor besitzt die syntaktische Form eines Atoms und die Argumente sind selbst wieder Terme.

Um die Lesbarkeit von Prolog-Programmen zu erleichtern, werden die obigen Grundelemente eines Terms ergänzt. Es wird die Verwendung von Operatoren ermöglicht und eine spezielle Notation für Listen eingeführt[4]. Ein Term kann damit also zusätzlich ein Operatorausdruck oder eine Liste sein.

Folgende syntaktische Formen eines Operatorausdrucks sind in Prolog möglich:

- *Term Operator*
- *Term_1 Operator Term_2*
- *Operator Term*

Ein Operator ist syntaktisch mit einem Funktor identisch. Damit ist es in Prolog möglich, daß z. B. arithmetische Ausdrücke, wie `x+y*z`, in der üblichen infix-Notation geschrieben werden können.

Dem Listen-Begriff in Prolog liegt folgende Definition zugrunde: Eine Liste ist entweder eine leere Liste oder besteht aus einem ersten Element, dem Listenkopf, gefolgt von einem Listenrumpf, der selbst wieder eine Liste ist. Demgemäß kann ein Term, der eine Liste repräsentiert, eine der folgenden syntaktischen Formen besitzen:

- []
- [*Term_1* | *Term_2*]
- [*Term*]

Die Zeichenreihe [] symbolisiert die leere Liste. Die Trennung zwischen Listenkopf und Listenrumpf wird durch die zweite Schreibweise dargestellt. Der senkrechte

[4]Listen und Ausdrücke mit Operatoren lassen sich auch durch Strukturen nachbilden. Die Operatoren nehmen hierbei die Stellung des Funktors einer Struktur ein und die Argumente des Operators erscheinen als Argumente der Struktur. Zur Darstellung von Listen in der Schreibweise einer Struktur, wird ein spezielles Symbol, nämlich der Punkt '.', als "Listen-Funktor" verwendet. Das erste Argument einer solchen "Listen-Struktur" repräsentiert ein Listenelement, während das zweite Argument den Rest der Liste darstellt. Beispiele für Listen und Operatorausdrücke in Strukturschreibweise sind: `+(x,*(y,z))` oder `.(a,.(b,.(c,[])))`

Strich gibt die Stelle an, an der die Auftrennung zwischen Kopf und Rumpf erfolgen soll. Die explizite Aufzählung der einzelnen Listenelemente ist mit der letzten Schreibweise möglich. Dort repräsentiert *Term* einen Ausdruck, in dem die einzelnen Elemente durch Kommata getrennt werden. Das Komma spielt hier die Rolle eines Operators.

Beispiele für Listen sind:

`[]`	die leere Liste
`[a, b, c]`	eine Liste mit den Elementen a, b, und c
`[model(k1,m1) | Rest]`	eine Liste, dessen erstes Element die Struktur `model(k1,m1)` ist und deren Rumpf durch die Variable `Rest` bezeichnet wird.

Ein Prolog-Programm besteht nun aus einer oder mehreren Klauseln. Eine Klausel kann sein

- ein Fakt,
- eine Regel oder
- eine Anfrage.

Fakten dienen in einem Prolog-Programm dazu, Eigenschaften von Objekten und deren Beziehungen untereinander zu beschreiben. Gesetzmäßigkeiten können durch Regeln ausgedrückt werden. Der Anstoß zur Lösungssuche und damit zur Abarbeitung eines Prolog-Programms wird schließlich durch Anfragen formuliert (siehe Abschnitt 2.5.2).

Ein Fakt besitzt eine der beiden folgenden syntaktischen Formen:

- *predicate_name.*
- *predicate_name(Term_1 , ... , Term_n).*

Die erste Form repräsentiert einen 0-stelligen Fakt, die zweite Form einen n-stelligen Fakt. Der Prädikatname *predicate_name* ist syntaktisch identisch mit einem Funktor, so daß *predicate_name(Term_1, ... , Term_n)* die Syntax einer Prolog-Struktur aufweist. Erst durch den abschließenden Punkt wird der Funktor bzw. die Prolog-Struktur zu einem Fakt. Auch die beiden anderen Klauseltypen, die Regel und die Anfrage, werden durch einen Punkt abgeschlossen.

Eine Regel besitzt folgende Syntax:

- *rulehead :- Term_1 Operator ... Operator Term_n.*

Regeln werden durch den Operator *:-* in einen Regelkopf *rulehead* und einen Regelrumpf unterteilt. Der Regelkopf ist, analog zu einem Fakt, mit einem Funktor bzw. mit einer Prolog-Struktur syntaktisch identisch. Der Regelrumpf besteht

aus einem oder mehreren Termen, die, im Falle von mehreren Termen, durch bestimmte Operatoren miteinander verbunden sind. Als Operatoren sind hier das Komma und der Strichpunkt zulässig. Wegen ihrer Bedeutung bei der Programmausführung werden die Terme des Regelrumpfs auch als Ziele (englisch: goals) bezeichnet (siehe Abschnitt 2.5.2).

Regeln und Fakten werden auch unter dem Begriff Prädikat zusammengefaßt. Beispielsweise werden die Klauseln
```
p(X, Y) :- r(X, Y, a).
q(a).
```
als Prädikat p mit Stelligkeit 2 und als Prädikat q mit der Stelligkeit 1 bezeichnet, oder in Kurzschreibweise als p/2 und q/1.

Der Begriff Prädikat kann auch für die Bezeichnung einer Menge von Klauseln eingesetzt werden, die den gleichen Namen und die gleiche Argumentanzahl besitzen. So können mit p/2 z. B. die folgenden Klauseln bezeichnet werden:
```
p(X, Y) :- r(X, Y, a).
p(X, Y) :- s(X, Y).
p(X, X).
```

Eine Anfrage hat folgende Syntax:

- *?- Term_1 Operator ... Operator Term_n.*

Die Syntax entspricht der Syntax einer Regel, bei der der Regelkopf fehlt und der Operator *:-* durch den Operator *?-* ersetzt ist.

2.5.2 Semantik

Die Bedeutung der verschiedenen Prolog-Konstrukte wird hier nur beispielhaft erläutert. Wie bereits erwähnt, ist eine vollständige Sprachbeschreibung der entsprechenden Literatur zu entnehmen.

Ein in dieser Literatur immer wieder anzutreffendes Programmbeispiel zur Erläuterung von Prolog ist die Beschreibung bestimmter Verwandtschaftsverhältnisse. Solche Verhältnisse können z. B. durch folgende Fakten beschrieben sein:

```
father(albert, victoria).
father(albert, edward).
male(albert).
male(edward).
```

Mit `father(albert, edward)` ist definiert, daß `albert` Vater von `edward` ist, und mit `male(edward)`, daß `edward` männlich ist.

Die Reihenfolge der Argumente eines Prädikats spielt eine wichtige Rolle. Im Prädikat father/2 nimmt das erste Argument die Vaterposition ein und das zweite Argument die Kindposition. Obiger Fakt ist damit nicht so zu interpretieren, daß `edward` Vater von `albert` ist. Die Bedeutung eines Prädikats und seiner Argumente wird bei der Programmierung festgelegt und sollte in der Regel durch zusätzliche Kommentare im Programmtext explizit beschrieben sein.

Ausgehend von der obigen Datenbasis, bestehend aus den beiden Prädikaten father/2 und male/1, können nun bereits Anfragen formuliert werden. Beispielsweise wird mit
```
?- father(albert, edward).
```
die Anfrage gestellt, ob `albert` der Vater von `edward` ist.

Die Ausführung dieser Anfrage besteht bei Prolog darin, daß in der gegebenen Datenbasis nach einer Lösung gesucht wird. Konkret wird nach einer Klausel gesucht, die mit dem Term in der Anfrage sowohl in seinem Prädikatsnamen und seiner Stelligkeit als auch in sämtlichen Argumenten übereinstimmt. Der Term in der Anfrage gibt das Ziel vor, nach dem in der Datenbasis gesucht wird. Die Suche in der Datenbasis verläuft von oben nach unten in der Reihenfolge, in der die einzelnen Klauseln aufgeschrieben sind. In unserem Beispiel wird auf diese Weise in der Datenbasis ein Fakt gefunden, der vollständig mit dem Term der Anfrage übereinstimmt. In einem solchen Erfolgsfall endet der Programmlauf mit der Ausgabe 'yes'. Wird keine entsprechende Klausel gefunden, endet der Programmlauf mit der Ausgabe 'no'.

Anfragen sind in der Regel etwas komplexer und bestehen aus mehreren Termen, die zudem als Argumente eine Variable enthalten können. Ein Beispiel ist:
```
?- father(albert, X), male(X).
```
Diese Anfrage wird so interpretiert, daß damit ein X gesucht wird, für das `albert` Vater ist und das zudem männlich ist. Der Komma-Operator hat hier also die Bedeutung einer logischen und-Verknüpfung[5].

Die Lösungssuche beginnt nun immer mit dem linken Teilziel einer Anfrage. Prolog sucht damit im obigen Beispiel in seiner Datenbasis zunächst nach einem zweistelligen Fakt `father`, bei dem die Argumente mit denen des Teilziels in Übereinstimmung gebracht werden können. Da die Suche in der Datenbasis von oben nach unten erfolgt, wird als erster Fakt `father(albert, victoria)` gefunden. Dieser Fakt kann mit dem Term der Anfrage in Übereinstimmung gebracht werden, da die Konstante im ersten Argument identisch ist und die Variable X an der zweiten Argumentposition mit dem entsprechenden konstanten Argument des Fakts besetzt werden kann. Die Variable wird in diesem Fall mit dem Wert der Konstante instantiiert. Man sagt dann, die Variable ist gebunden.

[5] Alternativ kann als Operator an dieser Position auch ein Strichpunkt stehen (vgl. Abschnitt 2.5.1). Dieser Operator besitzt dann die Bedeutung einer logischen oder-Verknüpfung.

Die Regeln, nach denen solche Argumente in Prolog zur Übereinstimmung gebracht werden, sind durch einen Unifikationsalgorithmus festgelegt. Dieser Algorithmus stellt fest, in welchen Teilausdrücken sich zwei Terme unterscheiden, und untersucht, ob unterschiedliche Teilausdrücke durch eine geeignete Substitution einander angeglichen werden können. Die Unifikation versucht also Terme syntaktisch einander anzugleichen.

Die Lösungssuche im Beispiel ist also für den ersten Teil der Anfrage erfolgreich. Zudem ist als Ergebnis dieser Teilanfrage die Variable X mit dem Wert `victoria` instantiiert. Nun ist noch das zweite Teilziel der Anfrage zu bearbeiten. Als Argument tritt dort die Variable X aus dem ersten Teilziel erneut auf. Durch die bisherige Lösungssuche ist diese Variable jedoch bereits gebunden, so daß im weiteren Programmablauf nach einem Fakt `male(victoria)` gesucht wird. Da ein solcher Fakt nicht in der Datenbasis vorhanden ist, endet dieser Teil der Lösungssuche ohne Erfolg.

In einem solche Fall, in dem Teilziele einer Anfrage bereits erfolgreich bearbeitet wurden und die Bearbeitung des aktuellen Teilziels erfolglos blieb, wird in Prolog nach einer anderen Lösungsmöglichkeit gesucht. Dies geschieht nach der Strategie des Backtracking. Das bedeutet, daß zum vorangehenden Teilziel der Anfrage zurückgegangen wird und alle Variablen freigesetzt werden, die bei der bisherigen Bearbeitung dieses Teilziels instantiiert wurden. Damit kann eine weitere Lösung für dieses Teilziel gesucht werden.

Im obigen Beispiel wird also für das erste Teilziel eine weitere Lösung gesucht. Dabei startet die Lösungssuche natürlich nicht wieder vom Anfang der Datenbasis, sondern wird bei der Klausel fortgesetzt, die die letzte Lösung repräsentierte. Damit wird im Beispiel der Fakt `father(albert, edward)` als weitere Lösung für das erste Teilziel gefunden und die Variable X mit `edward` instantiiert.

Mit dieser Variableninstantiierung wird nun erneut das zweite Teilziel der Anfrage bearbeitet. Die Suche nach einem Fakt `male(edward)` hat dieses Mal Erfolg. Damit kann die gesamte Anfrage erfolgreich mit der Ausgabe 'yes' beantwortet werden. Zudem werden als weitere Ergebnisse die Werte der instantiierten Variablen, die in der Anfrage enthalten sind, ausgegeben.

Die hier skizzierte Abarbeitung eines Prolog-Programms basiert auf dem sogeannten Resolutionsverfahren. Dieses Verfahren ist bei Prolog durch folgende Grundsätze charakterisiert:

1. Die Teilziele einer Anfrage werden von links nach rechts abgearbeitet. Erst wenn die Lösungssuche bei einem Teilziel erfolgreich war, wird mit dem nächsten Teilziel fortgefahren. Wird bei der Lösungssuche eine Variable gebunden, so sind damit alle Variablen gleichen Namens , die in der Anfrage enthalten sind, mit dem gleichen Wert instantiiert.

2. Die Lösungssuche bei jedem einzelnen Teilziel verläuft in der Datenbasis von oben nach unten.

3. Kann für ein Teilziel keine Lösung gefunden werden, so wird nach der Strategie des Backtracking vorgegangen.

Neben Fakten können auch Regeln in der Datenbasis enthalten sein. Mit solchen Regeln werden in Prolog allgemeingültige Gesetzmäßigkeiten ausgedrückt. Beispielsweise definiert die Regel
`son(X, Y) :- father(Y, X), male(X).`
eine Vater-Sohn-Beziehung: X ist ein Sohn von Y, wenn Y Vater von X und X männlich ist.

Die bisher aufgezeigten Grundsätze für die Lösungssuche in Prolog gelten auch, wenn Regeln in der Datenbasis enthalten sind. Zusätzlich kann nun jedoch der Fall eintreten, daß ein Teilziel der Anfrage mit dem Kopf einer Regel unifiziert werden kann. In diesem Fall wird das Teilziel der Anfrage in die im Regelrumpf enthaltenen Unterziele aufgeteilt. Die Lösungssuche muß nun für diese Unterziele erfolgreich sein, damit die Bearbeitung des Gesamtziels Ziels erfolgreich ist.

Die obige Regel läßt sich unter diesem Blickwinkel folgendermaßen lesen:

> Um die Anfrage, ist X Sohn von Y zu bearbeiten, bearbeite die beiden Anfragen, ist Y Vater von X und ist X männlich.

Die Lösungssuche setzt sich natürlich entsprechend fort, wenn eines der Ziele aus einem Regelrumpf wieder mit dem Kopf einer Regel unifiziert wird. Die Lösungssuche in Prolog läßt sich damit zusätzlich als eine sogenannte depth-first-search Strategie charakterisieren.

In Abbildung 2.5 ist eine solche Lösungssuche anhand eines einfachen Beispiels angedeutet. Die Abarbeitung der Lösungssuche wird durch einen Suchbaum dargstellt. Die Reihenfolge, in der dieser Baum bei der Abarbeitung der Anfrage durchlaufen wird, ist durch entsprechende Pfeile und Kommentare gekennzeichnet.

Anhand des Beispiels wurde nun die Abarbeitung der verschiedenen Konstrukte eines Prolog-Programms skizziert. Da bei dieser Betrachtungsweise die ausgeführten Operationen im Vordergrund stehen, spricht man hierbei von der operationalen Semantik von Prolog. Die operationale Bedeutung eines Prolog-Programms kann dann als die Menge aller Grundanfragen (Anfragen ohne Variable) charakterisiert werden, die zusammen mit den Fakten und Regeln des Programms bei Anwendung des Resolutionsverfahrens mit 'yes' beantwortet werden (siehe [SS86]).[6]

[6]Neben der operationalen Betrachtungsweise gibt es eine deklarative, bei der die Semantik

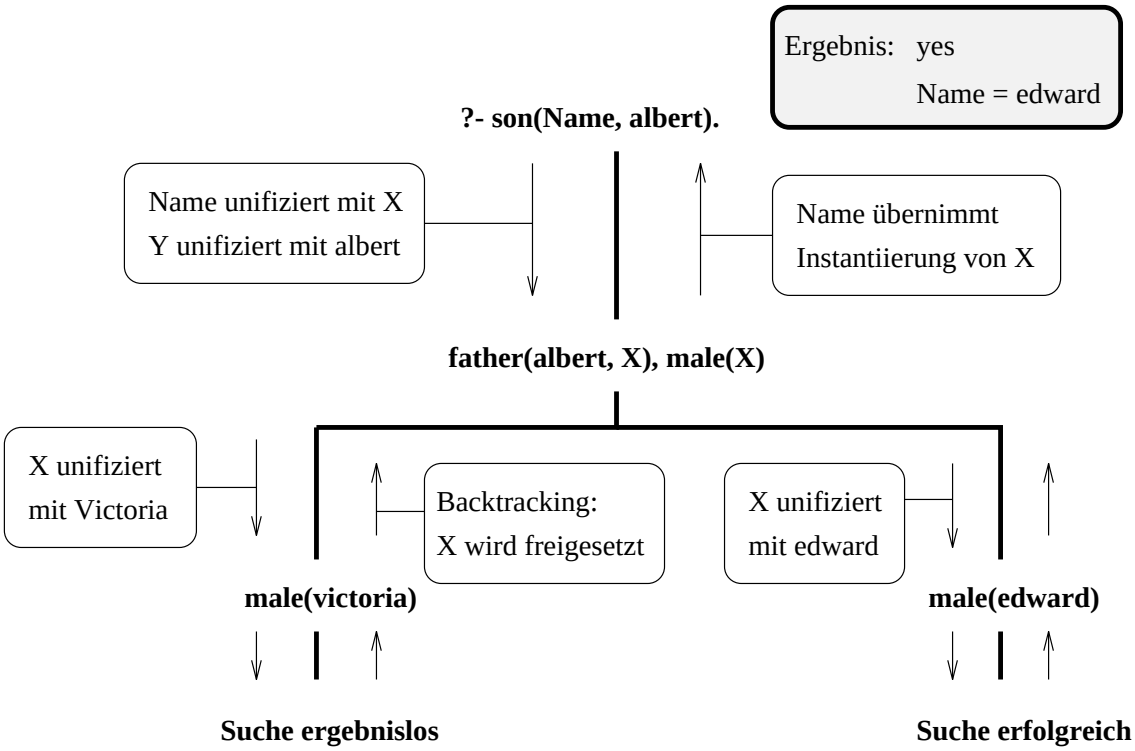


Abbildung 2.5: Ablauf der Lösungssuche für die Anfrage `?- son(Name, albert).`

von Prolog mit Mitteln der Logik beschrieben wird. Die deklarative Bedeutung eines Programms besteht demnach im minimalen Herbrand-Modell.

Prolog weist jedoch einige vordefinierte Prädikate auf, sogenannte built-in Prädikate, die sich nicht ohne weiteres deklarativ beschreiben lassen (z. B. der cut-Operator oder der not-Operator). Um ein umfassendes Verständis von Prolog zu erhalten, ist deshalb die operationale Sichtweise zu bevorzugen.

Kapitel 3

Theoriebegriff

Dieses Kapitel faßt die Definitionen zusammen, die für den Aufbau eines strukturalistischen Theoriebegriffs benötigt werden. Im wesentlichen ist dies eine Wiedergabe der entsprechenden Definitionen aus [Bal94] und [BLZ93].

Diese Definitionen sind hier noch einmal aufgeführt, um damit zum einen deutlich zu machen, was in dieser Arbeit unter Theorie verstanden wird, und zum anderen, um damit aufzeigen zu können, bis zu welchem Abstraktionsgrad dieser Theoriebegriff im Simulationsmodell abgebildet wird. Diese Abbildung ist dann auch am Ende des Kapitels beschrieben.

3.1 Theorie-Kerne

Definition 3.1 ($(k+m)$-Typen)
Für die natürlichen Zahlen k und m ($k > 0$ und $m \geq 0$) werden induktiv die $(k+m)$-Typen definiert:

1. Für jedes $i \leq k+m$ ist $\Box(i)$ ein $(k+m)$-Typ.
2. Sind τ_1 und τ_2 $(k+m)$-Typen, so sind auch $\circ(\tau_1)$ und $\oplus(\tau_1, \tau_2)$ $(k+m)$-Typen.

Definition 3.2 (Leitermenge vom Typ τ)
Für fest gegebene Mengen $u_1, \ldots, u_{k+m}$ wird die Leitermenge (vom Typ τ über $u_1, \ldots, u_{k+m}$) definiert durch Induktion über dem Aufbau von τ:

1. Hat τ die Form $\Box(i)$ mit $i \leq k+m$, so sei $\tau(u_1, \ldots, u_{k+m}) = u_i$.
2. Hat τ die Form $\circ(\tau_1)$, so sei $\tau(u_1, \ldots, u_{k+m}) = \mathbf{Pot}(\tau_1(u_1, \ldots, u_{k+m}))$

3. Hat τ die Form $\oplus(\tau_1, \tau_2)$, so sei
   $\tau(u_1, \ldots, u_{k+m}) = \tau_1(u_1, \ldots, u_{k+m}) \times \tau_2(u_1, \ldots, u_{k+m})$

Definition 3.3 (Strukturtyp)
Ein Strukturtyp ϑ ist ein Tupel $\langle k, m, \tau_1, \ldots, \tau_n \rangle$, wobei gilt:

1. k, m, n sind natürliche Zahlen mit $k, n > 0$ und $m \geq 0$.
2. Jedes $\tau_i (i \leq n)$ ist ein $(k+m)$-Typ von der Form $\square(i)$ oder $\circ(\tau_1)$.

Definition 3.4 (Struktur)
x ist eine mengentheoretische Struktur (vom Typ ϑ) gdw es $D_1, \ldots, D_k, A_1, \ldots, A_m$, $R_1, \ldots, R_n$ gibt, sodaß

1. $x = \langle D_1, \ldots, D_k, A_1, \ldots, A_m, R_1, \ldots, R_n \rangle$
2. $\forall i \leq n : R_i \in \tau_i(D_1, \ldots, D_k, A_1, \ldots, A_m)$

x ist eine Struktur gdw es ein ϑ gibt, sodaß x eine mengentheoretische Struktur vom Typ ϑ ist.

Die Mengen $D_1, \ldots, D_k, A_1, \ldots, A_m$ heißen *Basismengen*, wobei $D_1, \ldots, D_k$ *Hauptbasismengen* und $A_1, \ldots, A_m$ *Hilfsbasismengen* genannt werden.
$R_1, \ldots, R_n$ heißen *Relationen*.

Definition 3.5 (Substitution)
Für $x = \langle y_1, \ldots, y_n \rangle$ und $i \leq n$ heißt y_i die i-te Komponente von x. x wird auch als i-te Projektion bezeichnet; in Zeichen: $\pi_i(x)$.

Ist $x = \langle D_1, \ldots, D_k, A_1, \ldots, A_m, R_1, \ldots, R_n \rangle$ eine Struktur vom Typ ϑ und $j \leq n$, so heißt x' eine j-Substitution von x gdw es $R^* \in \tau_j(D_1, \ldots, D_k, A_1, \ldots, A_m)$ gibt, sodaß x' aus x durch Ersetzung von R_j durch R^* entsteht.

Definition 3.6 (Charakterisierung)
Sei $\vartheta = \langle k, m, \tau_1, \ldots, \tau_n \rangle$ ein Strukturtyp.

Mit $\mathrm{STR}(\vartheta)$ wird die Klasse aller mengentheoretischen Strukturen vom Typ ϑ bezeichnet:

$$\begin{aligned} \mathrm{STR}(\vartheta) \quad = \quad & \{\langle D_1, \ldots, D_k, A_1, \ldots, A_m, R_1, \ldots, R_n \rangle | \\ & \forall i \leq n : R_i \in \tau_i(D_1, \ldots, D_k, A_1, \ldots, A_m)\} \end{aligned}$$

Für $i \leq n$ heißt $t_i : \mathrm{STR}(\vartheta) \rightarrow \mathrm{SET}$
$t_i(D_1, \ldots, D_k, A_1, \ldots, A_m, R_1, \ldots, R_n) = R_i$ der i-te Term von ϑ.

Für $i \leq n$ heißt $E_i \subseteq \mathrm{STR}(\vartheta)$ eine Charakterisierung des i-ten Terms gdw $\forall j \leq n, j \neq i$: wenn $x \in E_i$ und x' eine j-Substitution von x, dann ist auch $x' \in E_i$.

Definition 3.7 (Klasse von potentiellen Modellen)
M_p ist eine Klasse von potentiellen Modellen (vom Typ ϑ) gdw

1. $M_p \subseteq \mathrm{STR}(\vartheta)$
2. M_p läßt sich als Durchschnitt von Charakterisierungen schreiben, d.h. es gibt $E_{i_1}, \ldots, E_{i_s}$, $1 \leq i_1, \ldots, i_s \leq n$ mit $M_p = \bigcap\limits_{1 \leq r \leq s} E_{i_r}$

Definition 3.8 (Klasse von Modellen)
M ist eine Klasse von Modellen (vom Typ ϑ) gdw

1. $M \subseteq \mathrm{STR}(\vartheta)$
2. M läßt sich nicht als Durchschnitt von Charakterisierungen schreiben, d.h. es gibt keine $E_{i_1}, \ldots, E_{i_s}$, $1 \leq i_1, \ldots, i_s \leq n$ mit $M = \bigcap\limits_{1 \leq r \leq s} E_{i_r}$

Definition 3.9 (Quasi-metrischer Raum)
Sei $\mathbb{R}^* = \mathbb{R} \cup \{\infty\}$

$\langle X, d \rangle$ heißt quasimetrischer Raum gdw

1. $X \neq \emptyset$
2. $d : X \times X \to \mathbb{R}^*$
3. $\forall a, b, c \in X$:
   (a) $d(a, b) \geq 0$
   (b) $d(a, a) = 0$
   (c) $d(a, b) = d(b, a)$
   (d) $d(a, c) = d(a, b) + d(b, c)$

   d heißt Quasi-Metrik.

Definition 3.10 (Teilstrukturen im engeren Sinn)
Ist $x = \langle D_1, \ldots, D_k, A_1, \ldots, A_m, R_1, \ldots, R_n \rangle \in \mathrm{STR}(\vartheta)$, so werden die Komponenten von x mit $D_1^x, \ldots, D_k^x, A_1^x, \ldots, A_m^x, R_1^x, \ldots, R_n^x$ bezeichnet.

Für $x, y \in \mathrm{STR}(\vartheta)$ heißt y Teilstruktur von x im engeren Sinn (in Zeichen $y \overset{e}{\sqsubseteq} x$) gdw

1. $\forall i \leq k : D_i^y \subseteq D_i^x$ und $\forall j \leq m : A_j^y \subseteq A_j^x$

2. $\forall j \leq n$, wenn τ_j nicht die Form $\square(s_j)$ hat mit $1 \leq s_j \leq k+m$, dann $R_j^y \subseteq R_j^x$

3. $\forall j \leq n$, wenn τ_j die Form $\square(s_j)$ hat mit $1 \leq s_j \leq k+m$, dann $R_j^y = R_j^x$

Definition 3.11 (Partielle Modelle)
$M_{pp}(\vartheta)$ sei die Klasse aller Entitäten der Form

$$\langle D_1, \ldots, D_k, A_1, \ldots, A_m, R_1, \ldots, R_n \rangle,$$

die aus einem $x \in \mathrm{STR}(\vartheta)$ dadurch entstehen, daß einige der R_j, für die τ_j die Form $\square(s_j)$ hat, durch das Zeichen # ersetzt werden, wobei $j \leq n$ und $1 \leq s_j \leq k+m$. M_{pp} heißt die Klasse der partiellen Modelle (vom Typ ϑ).

In $M_{pp}(\vartheta)$ wird $\sqsubseteq$ wie folgt definiert.
Für $x, y \in M_{pp}(\vartheta)$ gelte y ist Teilstruktur von x (in Zeichen $x \sqsubseteq y$) gdw es $x', y' \in \mathrm{STR}(\vartheta)$ gibt, sodaß

1. $y' \overset{e}{\sqsubseteq} x'$

2. y und x entstehen aus y' und x' durch Ersetzung einiger R_j durch #.

3. $\forall j \leq n$: wenn die $(k+m+j)$-te Komponente von x ein # ist, so auch die $(k+m+j)$-te Komponente von y.

Definition 3.12 (Theorie-Kern)
K ist ein Theorie-Kern gdw es ϑ, M, d gibt, sodaß

1. $K = \langle \vartheta, M, d \rangle$

2. $\vartheta = \langle k, m, \tau_1, \ldots, \tau_n \rangle$ ist ein Strukturtyp.

3. $M \subseteq \mathrm{STR}(\vartheta)$ ist eine Klasse von Modellen vom Typ ϑ.

4. Es gibt $N \subseteq M_{pp}(\vartheta)$, sodaß $\langle N, d \rangle$ ein quasimetrischer Raum ist und $M \subseteq N$.

3.2 Theorie-Holone

Definition 3.13 (Basis für ein Theorie-Holon)
B ist eine Basis für ein Theorie-Holon gdw es H, L, I, L^*, E gibt, so daß $B = \langle H, L, I, L^*, E \rangle$ und

1. H ist eine Menge von Theorie-Kernen ($H \neq \emptyset$)
2. $L \subseteq \{\lambda | \exists K, K' \in H(\emptyset \neq \lambda \subseteq \mathrm{STR}(K) \times \mathrm{STR}(K'))\}$
3. $I \subseteq \cup\{M_{pp}(K) | K \in H\}$ ($I \neq \emptyset$)
4. $L^* \subseteq \{\lambda^* | \exists K, K' \in H(\emptyset \neq \lambda^* \subseteq M_{pp}(K) \times M_{pp}(K'))\}$
5. $E \subseteq \cup\{M_{pp}(K) | K \in H\}$ ($E \neq \emptyset$)

Ein Element $\lambda \in L$ heißt Link oder Querverbindung (von K zu einem K'), ein Element $x \in I$ heißt intendiertes System (für ein K), ein Element $\lambda^* \in L^*$ heißt Daten-Querverbindung (von K zu einem K'), ein Element $z \in E$ heißt Datenstruktur (für ein K).

Definition 3.14 (Theorie-Holon)
T ist ein Theorie-Holon gdw es H, L , I, L^*, E, $RepI$, $RepL$, $RepU$ gibt, sodaß $T = \langle H, L, I, L^*, E, RepI, RepL, RepU \rangle$ und

1. $\langle H, L, I, L^*, E \rangle$ ist eine Basis für ein Theorie-Holon
2. $RepI \subseteq I \times H$
3. $RepL \subseteq L^* \times L$
4. $RepU : I \times H \rightarrow \mathbb{R}^+$
5. H ist endlich und $H \neq \emptyset$
6. L^* ist endlich und $\forall \lambda^* \in L^* : \lambda^*$ ist endlich,
   E ist endlich und $\forall z \in E : z$ ist endlich
7. Sei $Domain(N) = \{x \in X | \exists y \in Y(\langle x, y \rangle \in N)\}$,
   $Range(N) = \{y \in Y | \exists x \in X(\langle x, y \rangle \in N)\}$ mit $N \subseteq X \times Y$.
   $Domain(RepI) = I$, $Range(RepI) = H$,
   $Domain(RepL) = L^*$, $Range(RepL) = L$
8. $\forall x \forall K(\langle x, K \rangle \in RepI \rightarrow x \in M_{pp}(K))$
9. $\forall K, K' \in H \; \forall \lambda^* \in L^* \; \forall \lambda \in L$
   $(\langle \lambda^*, \lambda \rangle \in RepL$
   $\rightarrow [\lambda^* \subseteq M_{pp}(K) \times M_{pp}(K') \leftrightarrow \lambda \subseteq \mathrm{STR}(K) \times \mathrm{STR}(K')]);$
10. $\forall K, K' \in H \; \forall \lambda^* \in L^* \; \forall x, x'$
    $(\langle x, x' \rangle \in \lambda^* \subseteq M_{pp}(K) \times M_{pp}(K') \rightarrow \langle x, K \rangle \in RepI \wedge \langle x', K' \rangle \in RepI);$

11. H ist zusammenhängend bezüglich L, d. h. $\forall K, K' \in H$, wenn $K \neq K'$, dann gibt es ein $n \geq 2$, $K_1, \ldots, K_n \in H$ und $\lambda_1, \ldots, \lambda_{n-1} \in L$, sodaß $K = K_1, \ldots, K' = K_n$ und $\forall i \leq n$: $\lambda_i \subseteq \mathrm{STR}(K_i) \times \mathrm{STR}(K_{i+1})$ oder $\lambda_i \subseteq \mathrm{STR}(K_{i+1}) \times \mathrm{STR}(K_i)$

12. $E \subseteq I$

13. $\forall x \forall y : x \in E \wedge y \sqsubseteq x \wedge y \neq \bar{0} \rightarrow y \in E$ mit $\bar{0} = \langle \emptyset, \ldots, \emptyset, \#, \emptyset, \ldots \rangle$

Definition 3.15 (Theorie-Element)
Ein Theorie-Holon T mit nur einem Theorie-Kern heißt Theorie-Element oder isolierte Theorie.

3.3 Abbildung des Theoriebegriffs im Simulationsmodell

Die Entwicklung von Theorien soll simuliert werden. Der im Simulationsmodell verwendete Theorienbegriff ist durch die obige Definition eines Theorie-Elements beschrieben. Diese Theorie-Elemente werden jedoch nicht isoliert betrachtet, sondern eingebettet in der komplexeren Struktur eines Theorie-Holons. Es wird hier also die Entwicklung von Theorie-Holonen simuliert.

Ein Theorie-Holon wird im Simulationsmodell durch die Rechenstruktur Holon repräsentiert. Diese Rechenstruktur umfaßt zum einen Daten, die einen Holon im Simulationsmodell beschreiben und zum anderen Operationen, die auf diesen Daten ausgeführt werden können.

Die Daten dieser Rechenstruktur können noch weiter unterteilt werden in die Daten der Holon-Definition und in Meta-Daten. In den Daten der Holon-Definition spiegeln sich unmittelbar die relevanten Komponenten eines Holon gemäß der Definition 3.14 wieder. Die Meta-Daten enthalten weitergehende Informationen über diese Definitionsdaten, z. B. wird für jedes Theorie-Element ein statischer Gütegrad definiert, der ausdrückt, mit welcher Wahrscheinlichkeit in diesem Theorie-Element ein Problem gelöst werden kann. Mit den Operationen schließlich ist die Einhaltung der Eigenschaften eines Holon gemäß der Definition gewährleistet.

3.3.1 Abbildung der Definition eines Theorie-Holons

Die Rechenstruktur Holon bildet die Definition 3.14 nicht vollständig und bis in jedes Detail ab. Dies hat folgende Gründe:

- Die vorliegende Arbeit stellt einen ersten Ansatz zur Simulation dar. Es ist deshalb sinnvoll, die Komplexität des modellierten Systems soweit zu reduzieren, daß man zwar Erfahrungen mit dieser Art der Simulation sammeln kann, aber der Entwicklungsaufwand für das Simulationsmodell sich in Grenzen hält.
- Es soll der Einfluß von Problemlösestrategien auf die Entwicklung eines Theorie-Holons untersucht werden. Komponenten eines Theorie-Holons, die bei der Lösung der betrachteten Problemarten keine unmittelbare Rolle spielen, können für diesen ersten Ansatz zunächst vernachläßigt werden.
- Die vollständige Repräsentation eines konkreten Strukturtyps übersteigt die Möglichkeiten eines herkömmlichen Computers. Dessen Speicherkapazität und die eingeschränkten Fähigkeiten zur Berechnung von mathematischen Funktionen sind hier die wesentlichen Begrenzungsfaktoren.

Komponenten eines Theorie-Holons		Darstellung im Simulationsmodell
H	endliche Menge von Theoriekernen	**kernel(K)**
	$K = \langle$	
	ϑ,	nicht repräsentiert
	M,	**model(K,M)**
	$d\rangle$	**part_model(K,P)**
		metric(K,P1,P2,D)
L	theoretische Links	**theo_link(K1,K2,M1,M2)**
I	intendierte Anwendungen	nicht repräsentiert
L^*	Daten-Querverbindung	**data_link(K1,K2,P1,P2)**
E	Datenstrukturen	**datastruct(K,Z)**
$RepI$		nicht repräsentiert
$RepL$		**rep_l(Data_link, Theo_link)**
$RepU$		nicht repräsentiert

Tabelle 3.1: Abbildung der Definition eines Holons im Simulationsmodell

Tabelle 3.1 gibt einen Überblick darüber, welche Komponenten eines Theorie-Holons sich im Simulationsmodell wiederfinden und durch welche Prolog-Prädi-

kate diese Komponenten repräsentiert werden[1]. Aus den oben genannten Gründen ist der Strukturtyp ϑ, sowie die intendierten Anwendungen mit den davon abhängigen Relationen *RepI* und *RepU* nicht explizit berücksichtigt. Dadurch wird von den Basismengen und den darauf definierten Relationen eines Theorie-Kerns abstrahiert. Die Lösung von Problemen, die definitionsgemäß auf der Basis von solchen Strukturtypen erfolgt, ist damit im Simulationsmodell auf entsprechend abstrakter Ebene nachzubilden (siehe Kapitel 4).

Mit diesen Prolog-Prädikaten ist die abstrakte Beschreibung einer Menge von Prolog-Fakten gegeben. Über diese Menge von Fakten sind schließlich die Daten eines konkreten Theorie-Holons gegeben. Im Simulationsmodell sind die Argumente dieser Fakten mit symbolischen Konstanten besetzt. Mit diesen Konstanten werden die Elemente eines konkreten Holons bezeichnet. In dem Fakt `model(k1, m1)` z. B. bezeichnet `k1` einen Theorie-Kern und `m1` identifiziert ein Modell. Der Fakt ist damit so zu interpretieren, daß `m1` ein Modell des Theorie-Kerns `k1` ist.

Nachfolgend werden nun die einzelnen Prädikate, die die Definition eines Holons (Definition 3.14) im Simulationsmodell repräsentieren ausführlich beschrieben.

kernel(K) – **K** ist ein Theorie-Kern des Holons. Die einzelnen Theorie-Kerne werden durch die Konstanten **k1**, **k2** usw. bezeichnet.

model(K, M) – **M** ist ein Modell des Theorie-Kerns **K**. Die einzelnen Modelle werden durch die Konstanten **m1**, **m2** usw. bezeichnet.

part_model(K,P) – **P** ist ein partielles Modell des Theorie-Kerns **K**. Die einzelnen partiellen Modelle werden durch die Konstanten **p1**, **p2** usw. bezeichnet.

metric(K, P1, P2, D) – **D** ist der Abstand zwischen den beiden partiellen Modellen **P1** und **P2** des Theorie-Kerns **K**.

Wegen Punkt 4 in Definition 3.12 ist die Metrik auch über die Klasse der Modelle eines Theorie-Kerns definiert. Die Variablen **P1** und **P2** sind also auch mit Modellen instantiiert, die bereits als Fakten des Prädikats model/2 in der Prolog-Datenbasis enthalten sind.

Wegen Punkt 5 in Definition 3.13 ist die Metrik auch über den Datenstrukturen für einen Theorie-Kern definiert. Zusätzlich sind die Variablen **P1** und **P2** also auch mit Datenstrukturen instantiiert, die als Fakten des Prädikats datastruct/2 in der Prolog-Datenbasis vorliegen.

[1]Die Prädikate sind dort als Prolog-Strukturen angegeben, wobei die Argumente mit Variablen besetzt sind. In der nachfolgend angegebenen detaillierteren Beschreibung der einzelnen Prädikate können damit die Argumentpositionen über den jeweiligen Variablennamen angesprochen werden.

Die Variable **D** ist mit Werten zwischen 1 und 100 besetzt. Die Werte werden bei der Generierung der Holon-Daten anhand von gleichverteilten Zufallszahlen erzeugt. Der Abstand zweier partieller Modelle wird damit auf den Bereich von 1 bis 100 normiert. Die Beschränkung auf diesen Wertebereich ist eine willkürliche Festlegung, die zu Gunsten einer einfacheren Überprüfung des Abstandes im Rahmen von Problemlösungen getroffen wurde.

Drückt man diese Bedingungen in Prolog aus, so ergibt sich folgende Regel[2]:

```
metric(K, P1, P2, D) :-
    kernel(K),
    (part_model(K, P1); data_struct(K, P1); model(K, P1) ),
    (part_model(K, P2); data_struct(K, P2); model(K, P2) ),
    0 < D, D <= 100.
```

theo_link(K1, K2, M1, M2) – Es gibt eine (theoretische) Querverbindung von dem Modell **M1** des Theorie-Kerns **K1** zu dem Modell **M2** des Theorie-Kerns **K2**.

Es muß folgende Regel erfüllt sein[3]:

```
theo_link(K1, K2, M1, M2) :-
    kernel(K1), kernel(K2),
    K1 \ == K2,
    model(K1, M1),
    model(K2, M2).
```

data_link(K1, K2, P1, P2) – Es gibt eine Daten-Querverbindung von dem partiellen Modell **P1** des Theorie-Kerns **K1** zu dem partiellen Modell **P2** des Theorie-Kerns **K2**.

Wegen Punkt 5 in Definition 3.13 sind Daten-Querverbindungen auch über die Datenstrukturen für einen Theorie-Kern definiert. Zusätzlich sind die Variablen **P1** und **P2** also auch mit Datenstrukturen instantiiert, die als Fakten des Prädikats datastruct/2 in der Prolog-Datenbasis vorliegen.

[2]Die Regel ist folgendermaßen zu lesen:
D ist der Abstand zwischen den partiellen Modellen **P1** und **P2** des Theorie-Kerns **K**, wenn **K** ein Theorie-Kern ist und **P1** ein partielles Modell von **K** oder eine Datenstruktur von **K** oder ein Modell von **K** ist und **P2** ein partielles Modell von **K** oder eine Datenstruktur von **K** oder ein Modell von **K** ist und der Wert von **D** größer 0 und kleiner gleich 100 ist.

[3]Die Regel ist folgendermaßen zu lesen: Es gibt eine (theoretische) Querverbindung von dem Modell **M1** des Theorie-Kerns **K1** zu dem Modell **M2** des Theorie-Kerns **K2**,
wenn **K1** und **K2** Theorie-Kerne sind und
K1 verschieden von **K2** ist und
M1 ein Modell von **K1** ist und **M2** ein Modell von **K2** ist.

Damit muß also folgende Regel gelten[4]:

```
data_link(K1, K2, P1, P2) :-
    kernel(K1), kernel(K2),
    K1 \ == K2,
    (part_model(K, P1); data_struct(K, P1) ),
    (part_model(K, P2); data_struct(K, P2) ).
```

datastruct(K,Z) – **Z** ist eine Datenstruktur des Theorie-Kerns **K**. Die einzelnen Datenstrukturen werden durch die Konstanten **z1**, **z2** usw. bezeichnet.

rep_l(Data_link, Theo_link) – Die Daten-Querverbindung **Data_link** steht in der RepL-Relation zu der Verbindung **Theo_link**. **Data_link** ist mit einem Fakt des Prädikats data_link/4 besetzt, **Theo_link** mit einem Fakt des Prädikats theo_link/4.

Abbildung 3.1 versucht graphisch am Beispiel von 2 Theorie-Kernen einen Überblich zu vermitteln über die wesentlichen Komponenten eines Theorie-Holons und deren Zusammenhänge, soweit sie im Simulationsmodell nun abgebildet sind.

3.3.2 Die Meta-Daten eines Holons im Simulationsmodell

Die Meta-Daten der Rechenstruktur Holon enthalten zum einen Informationen, die die Verwaltung der Definitionsdaten eines Holons unterstützt (z. B. die aktuelle Anzahl der Datenstrukturen eines Theorie-Kerns). Zum Anderen sind damit Daten gegeben, die zur Steuerung der Problemlösestragien beitragen. Diese Meta-Daten sind durch folgende Prolog-Prädikate repräsentiert:

kernel_nr(N) – **N** ist die Anzahl der Theorie-Kerne im aktuellen Theorie-Holon.

datastruct_nr(K, N) – **N** ist die aktuelle Anzahl der Datenstrukturen des Theorie-Kerns **K**.

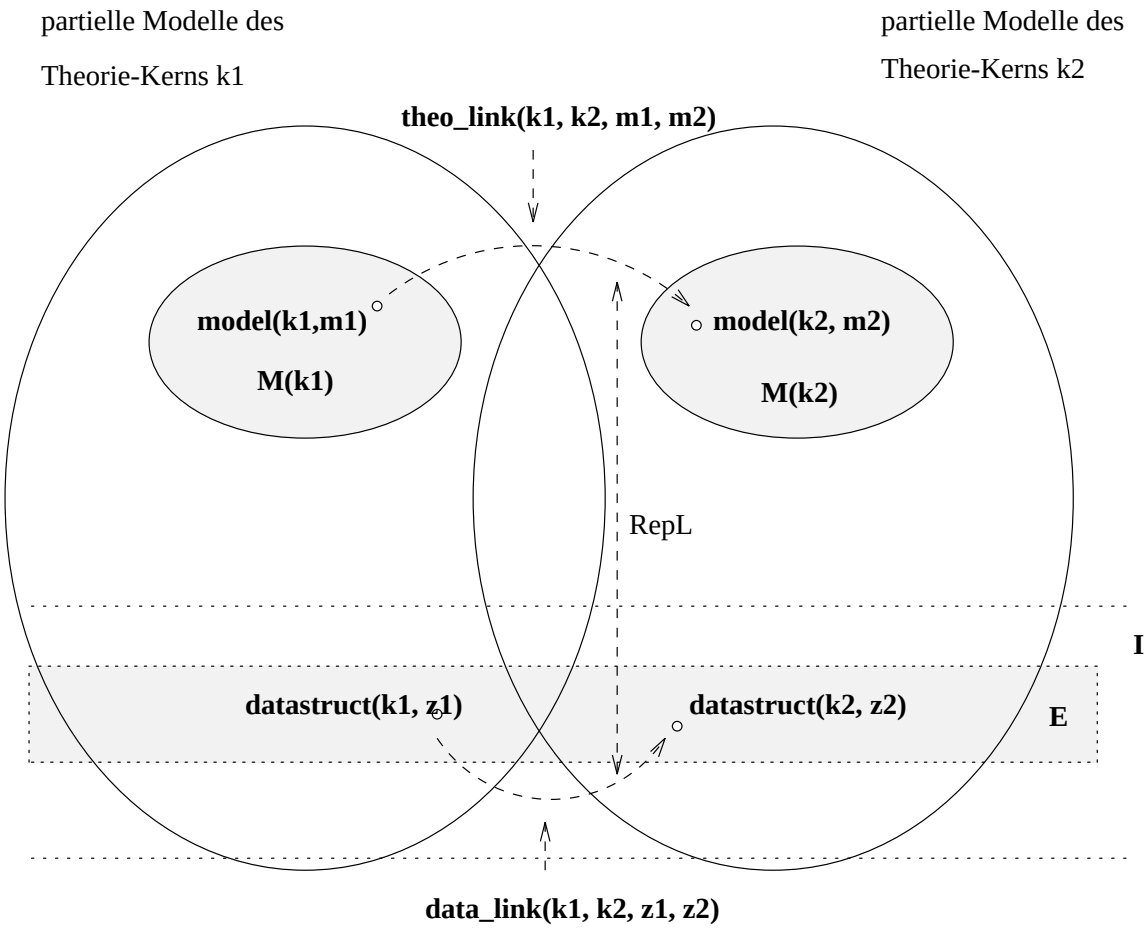
model_nr(K, N) – **N** ist die aktuelle Anzahl der Modelle des Theorie-Kerns **K** [5].

partmodel_nr(K, N) – **N** ist die aktuelle Anzahl der partiellen Modelle des Theorie-Kerns **K**.

data_link_nr(K1, K2, N) – **N** ist die aktuelle Anzahl der Daten-Querverbindungen, die vom Theorie-Kern **K1** zu dem Theorie-Kern **K2** führen.

[4] Analog wie oben zu lesen

[5] Die Zahl der abstrakten Modelle ist definitionsgemäß unendlich. Mit Hinblick auf die durchzuführenden Simulationen ist jedoch nur eine endliche Zahl solcher Modelle praktikabel.



I – intendierte Anwendungen des Holons
E – Datenstrukturen des Holons
M(k1) – Modelle des Theorie-Kerns k1
M(k2) – Modelle des Theorie-Kerns k2

Abbildung 3.1: Darstellung eines Holon im Simulationsmodell

theo_link_nr(K1, K2, N) – **N** ist die aktuelle Anzahl der (theoretischen) Querverbindungen, die vom Theorie-Kern **K1** zu dem Theorie-Kern **K2** führen.

repl_nr(K1, K2, N) – **N** ist die aktuelle Anzahl der Tupel, bestehend aus einer Daten-Querverbindungen und einer (theoretischen) Querverbindungen, die vom Theorie-Kern **K1** zu dem Theorie-Kern **K2** führen und in der RepL-Relation zueinander stehen.

static_grade(K,G) – **G** ist der statische Gütegrad des Theorie-Kerns **K**. **G** gibt die Wahrscheinlichkeit an, mit der im Theorie-Kern **K** ein elementares Passungsproblem gelöst werden kann. **G** besitzt ganzzahlige Werte zwischen 1 und 100. Siehe auch Kapitel 4.3.

count_fit(K, Total, Positiv) – **Total** ist die Gesamtanzahl der elementaren Passungsprobleme, die im Theorie-Kern **K** zu lösen versucht wurden. **Positiv** ist die Anzahl der Lösungsversuche mit positivem Ergebnis. **Total** und **Positiv** werden bei der Generierung eines Holons mit 0 vorbesetzt.

3.3.3 Die Operationen eines Holons im Simulationsmodell

Der Zugriff auf die Holon-Daten erfolgt zum einen im Rahmen einer Problemlösestrategie und zum anderen durch den Benutzer des Simulationsprogramms[6]. Er sollte ausschließlich über die Operationen der Rechenstruktur erfolgen. Damit wird die Einhaltung der Konsistenzbedingungen, die für diese Daten gelten, garantiert.

Die Schnittstelle zwischen der Rechenstruktur Holon und einer Problemlösestrategie ist im Kapitel 5 beschrieben. Im Rest dieses Kapitels werden die Operationen beschrieben, die einem Benutzer des Simulationsprogramms zur Verfügung stehen. Diese Schnittstelle ist durch die folgenden Prolog-Prädikate realisiert:

generate_holon – Anhand von Benutzereingaben werden die Prolog-Fakten eines Holons generiert.

Vom Benutzer wird die Anzahl der Theorie-Kerne abgefragt. Für jeden Theorie-Kern sind dann folgende Benutzer-Eingaben möglich:

- der statische Güte-Grad des Theorie-Kerns
- die Anzahl der Modelle
- die Anzahl der partiellen Modelle
- die Anzahl der Datenstrukturen
- die Anzahl der (theoretischen) Querverbindungen zu den übrigen Theorie-Kernen
- die Anzahl der Daten-Querverbindungen zu den übrigen Theorie-Kernen
- die Anzahl der Querverbindungen, die in der RepL-Relation zueinander stehen.

Die Auswahl der Modelle, zwischen denen die vorgegebene Anzahl von theoretischen Links und Datenlinks generiert wird, erfolgt zufallsgesteuert. Analog werden die Fakten für die Daten-Querverbindungen und für die RepL-Relation erzeugt.

[6]Wegen der großen Menge von Fakten ist z. B. eine manuelle Eingabe der Daten kaum durchführbar, da dies zu zeitaufwendig und zu fehleranfällig ist. Deshalb ist eine entsprechende Operation bereitzustellen, die die automatische Generierung dieser Daten ermöglicht.

Der Abstand zweier partieller Modelle, ausgedrückt im Prädikat metric/4, wird ebenfalls zufällig bestimmt und ein konkreter Metrikwert mit Hilfe von gleichverteilten Zufallszahlen aus dem Intervall (1;100) vorgegeben.

save_holon(Filename) – Alle Daten der Rechenstruktur Holon werden in der Datei **Filename** als Prolog-Fakten gespeichert.

load_holon(Filename) – Die in der Datei **Filename** enthaltenen Daten werden in die Datenbasis des gestarteten Simulationsprogramms geladen. Vorausgesetzt wird, daß die Datei **Filename** durch die Ausführung des Prädikats save_holon/1 entstanden ist.

summary_holon(Filename) – Eine zusammenfassende Darstellung der Holon-Daten wird in die Datei **Filename** geschrieben.

Kapitel 4

Problemarten

Eine empirische Theorie ist auf der theoretischen Ebene durch den formalen Begriffsapparat eines Theorie-Kerns und auf der nicht-theoretischen Ebene durch die Menge der intendierten Anwendungen charakterisiert. Der zentrale Punkt bei der Beschäftigung mit einer solchen Theorie ist nun das Aufstellen von empirischen Behauptungen: Es wird behauptet, daß die intendierten Anwendungen Modelle der Theorie sind.

Die empirische Behauptung wird auf der Basis der Begriffe Passung und Konsistenz formuliert. Behauptet wird, daß ein partielles Modell zu einem Modell der Theorie paßt und zwar bei Beachtung der Querverbindungen. Das partielle Modell soll sich in die durch die Querverbindungen gegebenen Abhängigkeiten zu anderen Theorien konsistent einpassen[1].

Die Begriffe der Passung und der Konsistenz induzieren Problemarten, die es in der Entwicklung einer Theorie zu lösen gibt. Die Problemarten beeinflußen selbst wieder die Theorie-Entwicklung. Für das hier entwickelte Simulationsmodell bedeutet dies, daß die Problemarten so zu beschreiben sind, daß ihr Lösungsversuch im Verlauf der Theorie-Entwicklung simuliert werden kann.

Die Problemarten lassen sich auf der Basis des strukturalistischen Theoriebegriffs formal definieren. Ihre Definitionen sind nachfolgend aufgeführt gemäß [Bal94]. Welche Problemarten im Simulationsmodell berücksichtigt werden und in welcher Form ihre Beschreibung realisiert ist, wird am Schluß dieses Kapitels angegeben.

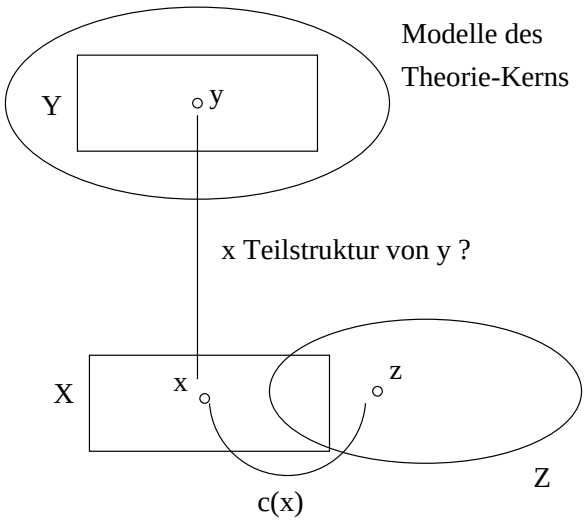
[1] Eine formale Beschreibung der empirischen Behauptung einer Theorie ist z. B. in [Bal94] oder in [BMS87] enthalten.

4.1 Passung

Mit dem Begriff der Passung wird die Beziehung zwischen einem partiellen Modell und einem Modell eines Theorie-Kerns betrachtet.

Sieht man in dem partiellen Modell eine Anwendung der Theorie, so läßt sich das mit der Passung verbundene Problem folgendermaßen beschreiben. Eine intendierte Anwendung der Theorie wird daraufhin überprüft, ob sie zu einem Modell der Theorie paßt, d. h. ob sie eine Teilstruktur eines Modells ist.

Verschiedene Variationen des Passungsbegriffs ergeben sich nun dadurch, daß zum einen auch Anwendungen, die hinreichend ähnlich zu den intendierten Anwendungen sind, auf ihre Passung mit einem Modell überprüft werden, und zum anderen dadurch, daß der Begriff der Passung auf Mengen von Modelle erweitert wird (siehe Abbildung 4.1).



Y – Teilmenge der Modelle des Theorie-Kerns
Z, X – Teilmengen der partiellen Modelle des Theorie-Kerns
c(x) – x ist ähnlich zu einem z.

Abbildung 4.1: Begriff der Passung

Definition 4.1 (Passung)
Sei T ein Theorie-Holon $T = \langle H, L, I, L^*, E, RepI, RepL, RepU \rangle$ und $K \in H$, $x \in M_{pp}(K)$, $y \in M(K)$, $X \subseteq M_{pp}(K)$ und $Y \subseteq M(K)$.

1. x **paßt ideal** zu y gdw $x \sqsubseteq y$.
2. x **paßt im Grad** ε zu y gdw
   $\exists z \in M_{pp}(K)\ (d_k(x, z) < \varepsilon \wedge z \sqsubseteq y)$ bei gegebenem $\varepsilon > 0, \varepsilon \in \mathbb{R}$

3. X **paßt ideal** zu Y gdw $\forall x \in X\ \exists y \in Y\ (x \sqsubseteq y)$

4. X **paßt stark** zu Y gdw es eine bijektive Abbildung $c : X \to Y$ gibt, so daß $\forall x \in X\ (x \sqsubseteq c(x))$.

5. X **paßt** [**stark**] **im Grad** ε zu Y gdw es ein Z gibt, so daß es eine bijektive Abbildung $c : X \to Z$ gibt und

   (a) $\forall x \in X : d_k(x, c(x)) < \varepsilon$.

   (b) Z paßt ideal [stark] zu Y.

Definition 4.2 (Passungsprobleme)
Sei T ein Theorie-Holon $T = \langle H, L, I, L^*, E, RepI, RepL, RepU \rangle$ und $K \in H$, $x \in E(K)$ und $X \subseteq E(K)$.

1. Sei $y \in M(K)$ gegeben.
   Zu überprüfen ist: paßt x ideal zu y.

2. Zu überprüfen ist: gibt es $y \in M(K)$, so daß x ideal zu y paßt.

3. Sei $\varepsilon > 0$ und $y \in M(K)$ gegeben.
   Zu überprüfen ist: paßt x im Grad ε zu y.

4. Sei $\varepsilon > 0$ gegeben.
   Zu überprüfen ist: gibt es $y \in M(K)$, so daß x im Grad ε zu y paßt.

5. Sei $y \in M(K)$ gegeben.
   Zu überprüfen ist: welches ist das kleinste $\varepsilon > 0$, für das x im Grad ε zu y paßt.

6. Sei $Y \subseteq M(K)$ gegeben.
   Zu überprüfen ist: paßt X ideal zu Y.

7. Zu überprüfen ist: gibt es $Y \subseteq M(K)$, so daß X ideal zu Y paßt.

8. Sei $\varepsilon > 0$ und $Y \subseteq M(K)$ gegeben.
   Zu überprüfen ist: paßt Y im Grad ε zu Y.

9. Sei $\varepsilon > 0$ gegeben.
   Zu überprüfen ist: gibt es $Y \subseteq M(K)$, so daß X im Grad ε zu Y paßt.

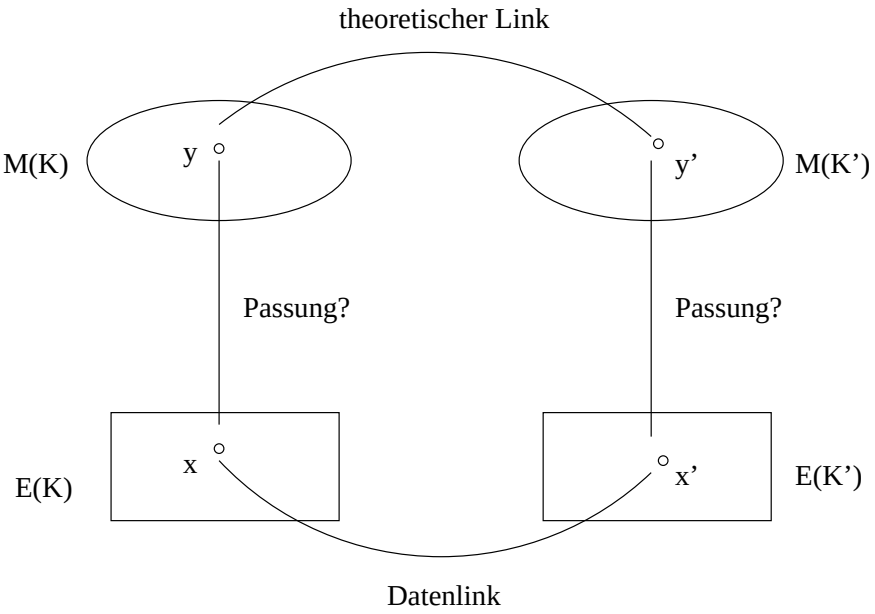
10. Sei $Y \subseteq M(K)$ gegeben.
    Zu überprüfen ist: welches ist das kleinste $\varepsilon > 0$, für das X im Grad ε zu Y paßt.

11. Sei $Y \subseteq M(K)$ gegeben.
    Zu überprüfen ist: paßt X stark zu Y.

12. Sei $\varepsilon > 0$ und $Y \subseteq M(K)$ gegeben.
    Zu überprüfen ist: paßt X stark im Grad ε zu Y paßt.

4.2 Konsistenz

Bei dem Begriff der Konsistenz werden die Abhängigkeiten zu anderen Theorie-Kernen, zu denen theoretische Links oder Datenlinks bestehen, betrachtet (siehe Abbildung 4.2). Die verschiedenen Variationen des Konsistenzbegriffs sind nachfolgend definiert.



M(K) – Modelle des Theorie-Kerns K
M(K') – Modelle des Theorie-Kerns K'
E(K) – Datenstrukturen des Theorie-Kerns K
E(K') – Datenstrukturen des Theorie-Kerns K'

Abbildung 4.2: Begriff der Konsistenz

Definition 4.3 (Konsistenz)
Sei T ein Theorie-Holon $T = \langle H, L, I, L^*, E, RepI, RepL, RepU \rangle$ und $K \in H$, $x \in E(K)$, $\lambda^* \in L^*$.

1. Sei $\langle x, x' \rangle \in \lambda^*$: x ist $\boldsymbol{\lambda^*}$**-**$\boldsymbol{x'}$**-konsistent** gdw
   $\exists \lambda \in L \; \exists y, y'$:
   $\langle y, y' \rangle \in L \wedge \langle \lambda^*, \lambda \rangle \in RepL$ und $x \sqsubseteq y \wedge x' \sqsubseteq y'$.

2. x ist **λ^*-konsistent** gdw x für alle x' mit $\langle x, x'\rangle \in \lambda^*$ λ^*-x'-konsistent ist.

3. x ist **stark λ^*-x'-konsistent** gdw
   $\forall \lambda \in L\ (\langle \lambda^*, \lambda\rangle \in RepL \rightarrow \exists y, y'\ (\langle y, y'\rangle \in \lambda \wedge x \sqsubseteq y \wedge x' \sqsubseteq y'))$.

4. x ist **stark λ^*-konsistent** gdw
   x für alle x' mit $\langle x, x'\rangle \in \lambda^*$ stark λ^*-x'-konsistent ist.

5. x ist **starr λ^*-konsistent** gdw
   $\exists \lambda : \langle \lambda^*, \lambda\rangle \in RepL \wedge \forall x'\ \exists y, y'$
   $(\langle x, x'\rangle \in \lambda^* \rightarrow \langle y, y'\rangle \in \lambda \wedge x \sqsubseteq y \wedge x' \sqsubseteq y')$.

6. x ist **(stark) L^*-konsistent** gdw
   x für alle $\lambda^* \in L^*$ (stark) λ^*-konsistent ist.

7. x ist **starr L^*-konsistent** gdw
   x für alle $\lambda^* \in L^*$ starr λ^*-konsistent ist.

8. Sei $X \subseteq E(K)$: X heißt **L^*-konsistent** gdw
   $\forall \lambda^* \in L^*\ \exists \lambda \in L\ \forall x \in X\ \forall x'$: wenn $\langle x, x'\rangle \in \lambda^*$,
   dann $\exists y, y'\ (\langle y, y'\rangle \in \lambda \wedge \langle \lambda^*, \lambda\rangle \in RepL \wedge x \sqsubseteq y \wedge x' \sqsubseteq y')$.

Definition 4.4 (Konsistenzprobleme)
Sei T ein Theorie-Holon $T = \langle H, L, I, L^*, E, RepI, RepL, RepU\rangle$ und $K, K' \in H$, $x \in E(K)$, $x' \in E(K')$,$\lambda^* \in L^*$.

1. Zu überprüfen ist: ist x λ^*-x'-konsistent.

2. Zu überprüfen ist: ist x λ^*-konsistent.

3. Zu überprüfen ist: ist x stark λ^*-x'-konsistent.

4. Zu überprüfen ist: ist x starr λ^*-konsistent.

5. Zu überprüfen ist: ist x L^*-konsistent.

6. Zu überprüfen ist: ist x stark L^*-konsistent.

7. Zu überprüfen ist: ist x starr L^*-konsistent.

8. Zu überprüfen ist: ist T λ^*-konsistent.

9. Zu überprüfen ist: ist $X \subseteq E(K)$ L^*-konsistent.

4.3 Realisierung der Problemarten im Simulationsmodell

Tabelle 4.1 gibt einen Überblick über die im Simulationsmodell berücksichtigten grundlegenden Problemarten. Sie sind grundlegend deshalb, weil die Beschreibung der anderen, komplexeren Problemarten auf sie zurückgreifen. Die ausgewählten Problemarten werden jeweils eins zu eins durch ein Prolog-Prädikat realisiert.

Problemart	**Realisierung im Simulationsmodell**
Definition 4.2, Punkt 1	**fit_problem_1(K, X, Y)**
Definition 4.2, Punkt 2	**fit_problem_2(K, X)**
Definition 4.2, Punkt 3	**fit_problem_3(K, X, Y, E)**
Definition 4.2, Punkt 4	**fit_problem_4(K, X, E)**
Definition 4.4, Punkt 1	**consist_problem_1(K1, K2, X1, X2)**
Definition 4.4, Punkt 2	**consist_problem_2(K1, K2, X1)**
Definition 4.4, Punkt 3	**consist_problem_3(K1, K2, X1, X2)**
Definition 4.4, Punkt 4	**consist_problem_4(K1, K2, X1)**

Tabelle 4.1: Abbildung der Problemarten im Simulationsmodell

Eine konkrete Problemlösung wird durch die Abarbeitung eines dieser Prolog-Prädikate simuliert. Die Problemlösung ist erfolgreich, wenn die Abarbeitung des Prädikats das Ergebnis 'yes' liefert. Der Lösungsversuch ist fehlgeschlagen, wenn das Ergebnis 'no' ist.

Im Rahmen eines solchen Lösungsversuchs ist bei jeder Problemart zu überprüfen, ob die aktuell betrachteten partiellen Modelle zueinander passen, d. h. ob das eine Teilstruktur der anderen ist. Die Realisierung dieser Überprüfung erfolgt zentral durch ein Prolog-Prädikat, nämlich durch check_fitness/3.

Da im Simulationsmodell der Strukturtyp der Theorie-Kerne nicht abgebildet ist (siehe Kapitel 3.3), wird diese Überprüfung auf der Basis von Zufallszahlen

nachgebildet. Ist also die Passung zweier partieller Modelle zu überprüfen, wird eine Zufallszahl im Intervall (1;100) gemäß einer Gleichverteilung erzeugt und mit dem statischen Gütegrad des betrachteten Theorie-Kerns verglichen. Besitzt die Zufallszahl einen Wert, der kleiner oder gleich dem Gütegrad ist, wird dies als eine positive Passung gewertet, andernfalls werden die beiden partiellen Modelle als nicht passend betrachtet. Der statische Gütegrad eines Theorie-Kerns drückt damit das Vertrauen in diese Theorie aus, Probleme dieser Art lösen zu können.

Auf der Basis dieses Zufallsmechanismus kann das Prädikat check_fitness/3 folgendermaßen beschrieben werden:

check_fitness(K, X, Y) – Zu überprüfen ist, ob das partielle Modell **X** von **K** zu dem partiellen Modell **Y** von **K** paßt. **X** und **Y** müssen als Datenstruktur, als partielles Modell oder als Modell des Theorie-Kerns **K** in der Prolog-Datenbasis vorhanden sein.

Die Ausführung des Prädikats liefert ein positives Ergebnis, falls eine der folgenden Bedingungen in der angegebenen Reihenfolge erfüllt ist:

1. **X** ist gleich **Y**.
2. Das Paar **X** und **Y** ist in der Prolog-Datenbasis bereits als passend gekennzeichnet.
3. Aufgrund des oben beschriebenen Zufallsmechanismus wird entschieden, daß **X** zu **Y** paßt.

Im dritten Fall wird zudem die Passung von **X** und **Y** in der Prolog-Datenbasis vermerkt. Die wiederholte Ausführung von check_fitness mit den gleichen Argumenten liefert damit wegen der zweiten Bedingung das gleiche Ergebnis.

Entsprechend wird in der Prolog-Datenbasis vermerkt, wenn aufgrund des Zufallsmechanismus **X** nicht zu **Y** paßt. Die Abarbeitung des Prädikats ergibt damit ein negatives Ergebnis, wenn:

1. **X** und **Y** als nicht-passend in der Datenbasis gekennzeichnet sind, oder
2. aufgrund des Zufallsmechanismus ermittelt wird, daß **X** und **Y** nicht passen.

Nachfolgend werden nun die Prädikate beschrieben, die die Definition der ausgewählten Problemarten realisieren.

fit_problem_1(K, X, Y) – Die Abarbeitung des Prädikats liefert ein positives Ergebnis, falls **X** eine Datenstruktur von **K** und **Y** ein Modell von **K** ist und **X** zu **Y** gemäß check_fitness/3 paßt.

fit_problem_2(K, X) – Die Abarbeitung des Prädikats liefert ein positives Ergebnis, falls **X** eine Datenstruktur von **K** ist und ein Modell von **K** in der Prolog-Datenbasis gefunden wird, so daß **X** gemäß check_fitness/3 zu diesem Modell paßt.

fit_problem_3(K, X, Y, E) – Die Abarbeitung des Prädikats liefert ein positives Ergebnis, falls **X** eine Datenstruktur und **Y** ein Modell von **K** ist und zudem ein partielles Modell von **K** in der Datenbasis gefunden wird, so daß der Abstand zwischen dem partiellen Modell und **X** kleiner dem Wert von **E** ist und das partielle Modell zu **Y** paßt. Der Abstand zwischen dem partiellen Modell und **X** ist durch einen Fakt des Prädikats metric/4 festgelegt.

fit_problem_4(K, X, E) – Die Beschreibung ist analog zu fit_problem_3/4, wobei **Y** ein beliebiges Modell von **K** aus der Prolog-Datenbasis sein kann.

consist_problem_1(K1, K2, X1, X2) – Die Voraussetzungen für die erfolgreiche Abarbeitung des Prädikats sind:

- **K1** und **K2** sind zwei voneinander verschiedene Theorie-Kerne.
- **X1** ist eine Datenstruktur von **K1**.
- **X2** ist eine Datenstruktur von **K2**.
- Die folgenden Fakten sind in der Prolog-Datenbasis zu finden:
  data_link(K1, K2, X1, X2).
  theo_link(K1, K2, Y1, Y2).
  rep_l(data_link(K1, K2, X1, X2), theo_link(K1, K2, Y1, Y2)).
- check_fitness muß für die beiden Paare (**X1**, **Y1**) und (**X2**, **Y2**) ein positives Ergebnis liefern.

consist_problem_2(K1, K2, X1) – Analog zu consist_problem_1/4, wobei eine beliebige Datenstruktur **X2** von **K2** in der Prolog-Datenbasis gesucht wird, für die die restlichen Bedingungen gelten.

consist_problem_3(K1, K2, X1, X2) – Die Voraussetzungen für die erfolgreiche Abarbeitung des Prädikats sind:

- **K1** und **K2** sind zwei voneinander verschiedene Theorie-Kerne.
- **X1** ist eine Datenstruktur von **K1**.
- **X2** ist eine Datenstruktur von **K2** .
- Der Fakt **data_link(K1, K2, X1, X2).** ist in der Prolog-Datenbasis enthalten.

- Für alle **Y1** und **Y2**, für die es einen Fakt
  rep_l(data_link(K1, K2, X1, X2), theo_link(K1, K2, Y1, Y2)).
  in der Prolog-Datenbasis gibt, muß gelten:
  check_fitness liefert für die Paare (**X1**, **Y1**) und (**X2**, **Y2**) ein positives Ergebnis.

consist_problem_4(K1, K2, X1) – Analog zu consist_problem_34, wobei eine beliebige Datenstruktur **X2** von **K2** in der Prolog-Datenbasis gesucht wird, für die die restlichen Bedingungen gelten.

Kapitel 5

Problemlösestrategien

Eine einfache und sehr allgemein anwendbare Problemlösestrategie ist mit dem Backtracking gegeben, so wie es bei Prolog eingesetzt wird (vergleiche Kapitel 2.5). Am Beispiel dieses einfachen Suchverfahrens läßt sich ein Schema erkennen, gemäß dem eine Problemlösestrategie strukturiert werden kann.

Dieses Schema wird zur Beschreibung der spezielleren Strategien, die die Problemlösung im Rahmen eines Theorie-Holons steuern, verwendet. Die Lösungsversuche eines Problems stellen dabei die Ereignisse dar, deren Auftreten den Zustand des Theorie-Holons verändert und die damit das Verhalten des Simulationsmodells bestimmen. Diese Art der Strukturierung entspricht dem Ansatz der ereignisorientierten Simulation (siehe Kapitel 2.3).

Im Rahmen dieser Arbeit werden nun beispielhaft einige Lösungsschritte definiert. Damit soll eine Basis geschaffen werden, auf der erste Simulationsläufe stattfinden können. Die dabei gewonnenen Erkenntnisse können genutzt werden, um diese Lösungsschritte anzupassen oder neue Schritte zu definieren.

Der Ablauf einer Strategie, der sich aus den hier definierten Lösungsschritten ergibt, unterteilt sich in Phasen, in denen lokal an einem Theorie-Kern Probleme gelöst werden. In bestimmten Situationen erfolgt dann der Wechsel zu einem anderen Theorie-Kern. Dieser Wechsel geschieht entweder entlang eines Datenlinks oder entlang eines theoretischen Links.

Begleitet wird diese auf die einzelnen Theorie-Kerne verteilte Problemlösungssuche von Veränderungen des Theorie-Holons:

- Paare von partiellen Modellen werden als passend bzw. nicht-passend markiert.
- Neue Datenstrukturen werden zusammen mit neuen Modellen aufgenommen.

- Datenlinks bzw. theoretische Links werden gelöscht oder neu eingerichtet.

In diesem Kapitel wird zunächst das allgemeine Schema einer Problemlösestrategie erläutert. Durch Anwendung dieses Schemas werden Lösungsschritte beschrieben, aus denen sich zwei Varianten einer Problemlösestrategie ergeben. Diese Beschreibung der Lösungsschritte ist mit Hinweisen auf ihre Realisierung im Simulationsmodell verbunden.

5.1 Abarbeitungszyklus einer Problemlösestrategie

Die Abarbeitung der Teilziele einer Prolog-Anfrage erfolgt nach der Lösungsstrategie des Backtracking.

Wie in Kapitel 2.5 erläutert, werden die einzelnen Teilziele einer Prolog-Anfrage von links nach rechts bearbeitet. Ist die Lösungssuche bei einem Teilziel positiv, wird der nächste Lösungsversuch bei dem in der Anfrage folgendem Teilziel gestartet. Existiert kein weiteres Teilziel, so bedeutet dies, daß eine Lösung für das letzte Teilziel der Anfrage gefunden wurde. Die Abarbeitung der gesamten Anfrage ist damit erfolgreich beendet.

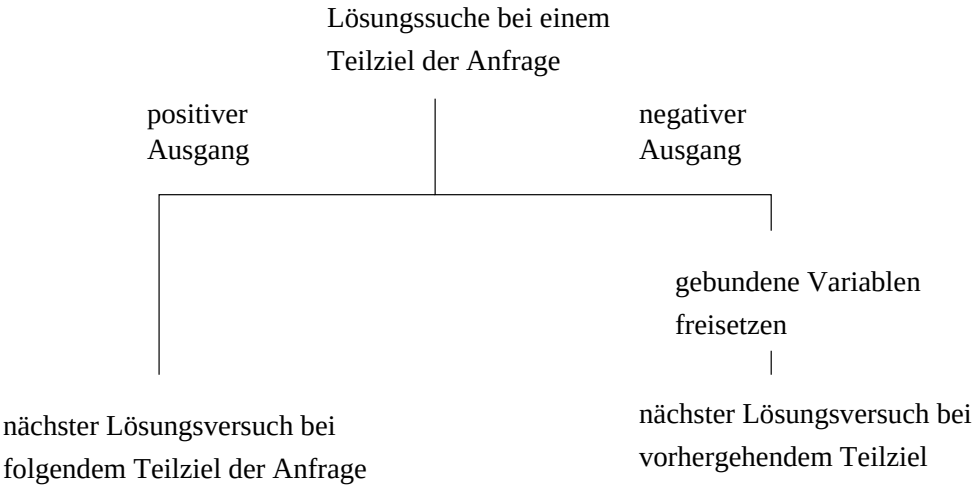


Abbildung 5.1: Lösungsschritt des Backtracking bei Prolog

Bei einem negativen Ergebnis der Lösungssuche werden die gebundenen Variablen freigesetzt und ein erneuter Lösungsversuch mit dem in der Anfrage vorangehenden Teilziel versucht. Existiert ein solches Teilziel nicht, d. h. der Lösungsversuch des ersten Teilziels in der Anfrage schlug fehl, ist die Abarbeitung der gesamten Anfrage mit einem negativen Ergebnis beendet.

Abbildung 5.1 zeigt im Überblick den Ablauf des eben skizzierten Lösungsschrittes der Backtrack-Strategie in Prolog. Eine Verallgemeinerung dieses Lösungsschrittes führt zu dem in Abbildung 5.2 dargestellten Schema.

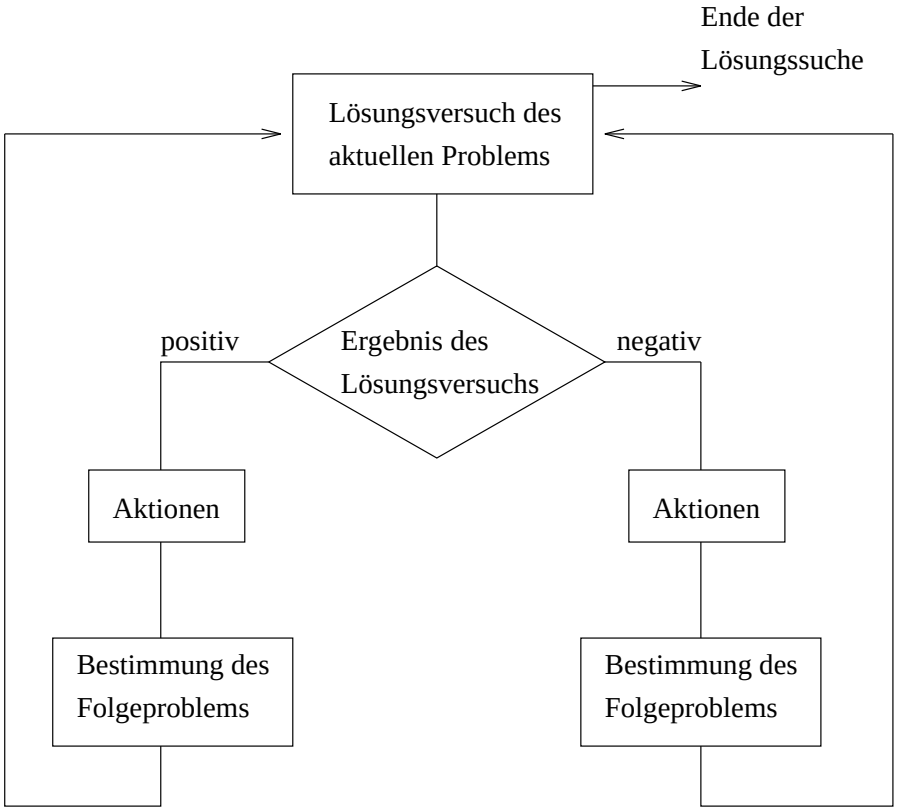


Abbildung 5.2: Schema eines Lösungsschrittes

Nach jedem Lösungsversuch werden abhängig vom Ausgang des Versuchs Aktionen ausgeführt. Diese Aktionen bewirken im allgemeinen die Zustandsänderungen des Modells. Im obigen Beispiel der Backtrack-Strategie ist z. B. das Freisetzen der gebundenen Variablen eine solche Aktion. Die Menge der auszuführenden Aktionen kann auch leer sein. Anschließend wird das Folgeproblem bestimmt und ein neuer Lösungsschritt beginnt. Läßt sich kein Folgeproblem bestimmen, wird die Lösungssuche beendet.

5.2 Strukturierung der Lösungssuche im Theorie-Holon

Das oben beschriebene Schema einer Strategie läßt sich auch auf die Lösungssuche in einem Theorie-Holon anwenden.

Die zu lösenden Probleme sind mit den Begriffen der Passung und der Konsistenz gegeben. Im Simulationsmodell ist ein Lösungsversuch durch die Abarbeitung eines der Prolog-Prädikate aus Kapitel 4.3 realisierbar.

Festzulegen ist nun, welche Aktionen in einem Lösungschritt ausgeführt werden sollen und welches Problem im nächsten Schritt zu lösen ist. Diese Festlegungen lassen sich für jeden Typ von Lösungsschritt zusammenfassend in einer Tabelle beschreiben, die gemäß dem obigen Schema strukturiert ist. Eine solche Tabelle umfaßt folgende Informationen:

- Parameter des Lösungsschrittes
- Beschreibung des aktuell zu lösenden Problems
- Die Beschreibung der Aktionen und des Folgeproblems sowohl bei positivem als auch bei negativem Ergebnis der aktuellen Problemlösung

Die auszuführenden Aktionen sind bei den hier definierten Lösungsschritten nicht nur vom Ausgang der aktuellen Problemlösung abhängig, sondern auch von Ergebnissen weiter zurückliegender Problemlösungen. In diesem Zusammenhang erweist es sich als günstig, aufeinanderfolgende Lösungsschritte, die am gleichen Theorie-Kern stattfinden, unter dem Begriff einer lokalen Lösungsphase zusammenzufassen. Die Anwendung einer Problemlösestrategie im Rahmen eines Simulationslaufes ergibt damit eine Folge von lokalen Lösungsphasen, die wiederum aus einem oder mehreren aufeinanderfolgenden Lösungsschritten bestehen (siehe Abbildung 5.3).

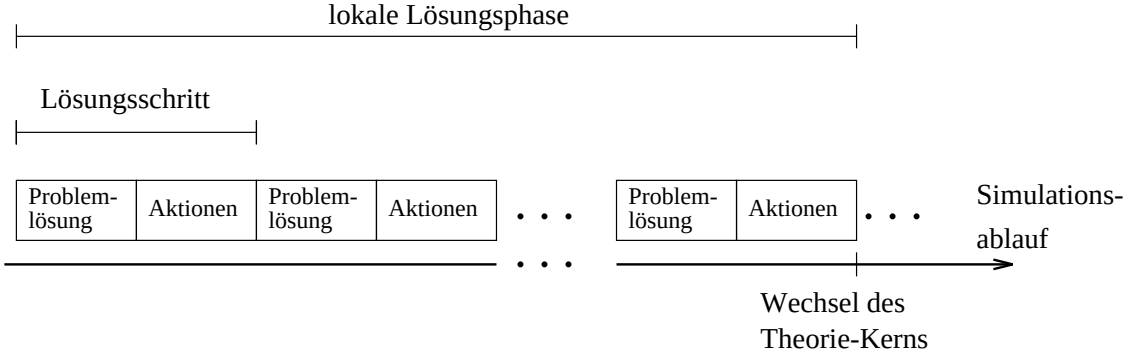


Abbildung 5.3: Phasen des Ablaufs einer Problemlösestrategie

Bei dem in dieser Arbeit erstellten Simulationsprogramm muß man sich beim Start eines Simulationslaufes für eine bestimmte Problemlösestrategie entscheiden. Im Simulationslauf kommen dann nur Lösungsschritte der ausgewählten Strategie zur Anwendung. Ein Wechsel der Strategie während eines Simulationslaufes ist nicht vorgesehen.

Bevor der Simulationslauf seinen Problemlösezyklus beginnt, müssen gewisse Parameter versorgt werden, die die Anwendung der ausgewählten Strategie steuern. Bei den hier beschriebenen Strategien müssen bei dieser Initialisierung folgende Angaben erfolgen:

- Die maximale Anzahl der im aktuellen Simulationslauf auszuführenden Lösungsschritte,
- der Theorie-Kern, bei dem die Lösungssuche beginnen soll, und
- die Anzahl der lokalen Lösungsphasen, die einem Lösungsschritt unmittelbar vorausgehen und bei der Ausführung dieses Lösungsschrittes mit betrachtet werden.

5.3 Strategie 1

Die Strategie 1 ist mit den Lösungsschritten *S1* und *S2* definiert. Sie läßt sich kurz folgendermaßen charakterisieren:

- Gelöst werden die Passungsprobleme 1 und 2 aus Definition 4.2, die durch die Prolog-Prädikate fit_problem_1/3 und fit_problem_2/2 realisiert sind.
- Der Wechsel eines Theorie-Kerns im Verlauf einer Lösungssuche erfolgt über einen Datenlink.
- Sind zwei Theorien, die über einen Datenlink miteinander verbunden sind, im Verlauf der Lösungssuche gut bestätigt worden, so werden theoretische Links zwischen diesen beiden Theorien eingeführt.

Folgende Situation muß bei Start der Lösungssuche nach Strategie 1 gegeben sein:

- Die Daten eines Theorie-Holon sind in der Prolog-Datenbasis vorhanden.
- Die Parameter der Strategie sind initialisiert.
- Eine Datenstruktur und ein Modell des Theorie-Kerns, bei dem die Lösungssuche beginnt, sind per Zufall bestimmt.
- Das erste zu lösende Problem besteht darin, zu überprüfen, ob die Datenstruktur zu dem Modell paßt. Zur Anwendung kommt damit der Lösungsschritt *S1*.

Die beiden Lösungsschritte der Strategie 1 sind gemäß der in Kapitel 5.2 festgelegten Struktur in den nachfolgenden Tabellen beschrieben. Die Beschreibung der

Aktionen, die in diesen Lösungsschritten auszuführen sind, erfolgt wegen der besseren Lesbarkeit mit Hilfe eines Pseudo-Codes[1], der sich auf folgende Prädikate und Funktionen abstützt:

confirmed(K, P) – **K** ist der Theorie-Kern der aktuellen lokalen Lösungsphase, **P** ist der Theorie-Kern der unmittelbar vorangegangenen lokalen Lösungsphase.

Das Prädikat wird erfüllt, wenn **K** und **P** in ihren jeweiligen Lösungsphasen gut bestätigt wurden. Gut bestätigt ist in diesem Zusammenhang ein Theorie-Kern, wenn in der betrachteten lokalen Lösungsphase mindestens 2 Passungen erkannt wurden und der dynamische Bestätigungsgrad des Theorie-Kerns größer 0.5 ist. Der dynamische Bestätigungsgrad eines Theorie-Kerns K ergibt sich dabei gemäß folgender Formel:

$$\text{dyn. Bestätigungsgrad von K} = \frac{\text{Anzahl der positiven Passungen bei K}}{\text{Gesamtanzahl der Passungsversuche bei K}}$$

not_confirmed(Path) – **Path** ist eine Liste der Theorie-Kerne, die in den unmittelbar vorangegangenen lokalen Lösungsphasen behandelt wurden. Die Pfadlänge ist durch den entsprechenden globalen Parameter der Strategie vorgegeben (siehe Kapitel 5.2).

Das Prädikat ist erfüllt, wenn bei den angegebenen Theorie-Kernen keine Passung in den jeweiligen lokalen Lösungsphasen ermittelt wurden.

generate_theolinks(K, P) – **K** ist der Theorie-Kern der aktuellen lokalen Lösungsphase, **P** ist der Theorie-Kern der unmittelbar vorangegangenen lokalen Lösungsphase.

Für bestehende Datenlinks von **P** nach **K** werden theoretische Links erzeugt. Die zu einem Datenlink erzeugten theoretischen Links werden zudem in die *RepL*-Relation aufgenommen.

change_kernel_by_datalink() – Der Theorie-Kern der aktuellen lokalen Lösungsphase wird verlassen und über einen Datenlink zu einem davon verschiedenen Theorie-Kern gewechselt. Die aktuelle Lösungsphase wird damit beendet und eine neue begonnen.

Existiert kein Datenlink, der von dem Theorie-Kern der aktuellen lokalen Lösungsphase zu einem anderen Theorie-Kern führt, wird spontan zu einem zufällig ausgewählten Theorie-Kern gesprungen.

[1]Die direkte Angabe der Prolog-Konstrukte aus dem Simulationsprogramm wäre mit programmtechnischen Details versehen, die hier nicht interessieren.

remove_datalinks(Path) – **Path** ist eine Liste der Theorie-Kerne, die in den unmittelbar vorangegangenen lokalen Lösungsphasen behandelt wurden. Die Pfadlänge ist durch den entsprechenden globalen Parameter der Strategie vorgegeben (siehe Kapitel 5.2).

Im Verlauf der vorangegangenen Lösungssuche erfolgte der Wechsel von einem der in **Path** enthaltenen Theorie-Kerne zum nächsten über einen Datenlink. Diese Datenlinks werden mit der Ausführung dieser Funktion gelöscht. Entsprechend wird die *RepL*-Relation aktualisiert.

add_datastruct_and_model() – Eine neue Datenstruktur und ein neues Modell wird dem Theorie-Kern der aktuellen lokalen Lösungsphase hinzugefügt.

Die Beschreibung einer Aktion in den nachfolgenden Tabellen hat z. B, dann folgende Form:

IF not_confirmed(Path)
THEN remove_datalinks(Path);

Dies ist zu lesen:

Falls das Prädikat not_confirmed(Path) erfüllbar ist, wird die Funktion remove_datalinks(Path) ausgeführt.

Das heißt, falls in den vorhergenden lokalen Lösungsphasen keine Passungen ermittelt wurde, wird der Datenlink-Pfad gelöscht, über den diese Lösungsphasen miteinander verbunden sind.

Darauf hinzuweisen ist noch, daß die Funktionen generate_theolinks, remove_datalinks und add_datastruct_and_model als Operationen der Rechenstruktur Holon realisiert sind (siehe Kapitel 3.3.3, Seite 35). Damit wird die Konsistenz der Daten eines Holons in der Prolog-Datenbasis gewährleistet.

Die Lösungsschritte *S1* und *S2* sind durch die Prolog-Prädikate **step_1(D, M)** und **step_2(D, M)** realisiert. Die Bedeutung dieser Prädikate wird unmittelbar aus den nachfolgenden Beschreibungen der Lösungsschritte deutlich.

Lösungsschritt: *S1*		
Parameter:	D	Datenstruktur des aktuellen Theorie-Kerns
	M	Modell des aktuellen Theorie-Kerns
Problem:		Im aktuellen Theorie-Kern wird überprüft, ob die Datenstruktur D zu dem Modell M paßt.
Bei positivem Ergebnis:		
Aktionen:		• IF confirmed(K, P) THEN generate_theolinks(K, P);
Folge-Problem:		Anwendung des Lösungsschritts *S2*
Bei negativem Ergebnis:		
Aktionen:		• IF not_confirmed(Path) THEN remove_datalinks(Path); • change_kernel_by_datalink();
Folge-Problem:		Anwendung des Lösungsschritts *S1*

Lösungsschritt: *S2*		
Parameter:	D	Datenstruktur des aktuellen Theorie-Kerns
	M	zu D passendes Modell des aktuellen Theorie-Kerns
Problem:		Im aktuellen Theorie-Kern wird überprüft, ob es noch weitere von M verschiedene Modelle gibt, die zu der Datenstruktur D passen.
Bei positivem Ergebnis:		
Aktionen:		• IF confirmed(K, P) THEN generate_theolinks(K, P); • IF confirmed(K, P) THEN add_datastruct_and_model();
Folge-Problem:		Anwendung des Lösungsschritts *S1*
Bei negativem Ergebnis:		
Aktionen:		• IF not_confirmed(Path) THEN remove_datalinks(Path); • change_kernel_by_datalink();
Folge-Problem:		Anwendung des Lösungsschritts *S1*

5.4 Strategie 2

Die Strategie 2 besteht aus den Lösungsschritten *S3* und *S4*. Sie stellt eine Variante von Strategie 1 dar, die sich in den folgenden Punkten unterscheidet:

- Der Wechsel eines Theorie-Kerns erfolgt über einen theoretischen Link.
- Sind zwei Theorien, die über einen theoretischen Link miteinander verbunden sind, im Verlauf der Lösungssuche gut bestätigt worden, so werden Datenlinks zwischen diesen beiden Theorien eingeführt.

Die Startsituation ist analog zur Strategie 1, nur mit dem Unterschied, daß als erster Lösungsschritt *S3* zur Anwendung kommt.

Zur Beschreibung der Aktionen, die in den Lösungsschritten *S3* und *S4* auszuführen sind, werden folgende zusätzliche Funktionen benötigt:

generate_datalinks(K, P) – **K** ist der Theorie-Kern der aktuellen lokalen Lösungsphase, **P** ist der Theorie-Kern der unmittelbar vorangegangenen lokalen Lösungsphase.

Für bestehende theoretische Links von **P** nach **K** werden Datenlinks erzeugt. Die zu einem theoretischen Link erzeugten Datenlinks werden zudem in der *RepL*-Relation berücksichtigt.

change_kernel_by_theolink() – Der Theorie-Kern der aktuellen lokalen Lösungsphase wird verlassen und über einen theoretischen Link zu einem davon verschiedenen Theorie-Kern gewechselt. Die aktuelle Lösungsphase wird damit beendet und eine neue begonnen.

Existiert kein theoretischer Link, der von dem Theorie-Kern der aktuellen lokalen Lösungsphase zu einem anderen Theorie-Kern führt, wird spontan zu einem zufällig ausgewählten Theorie-Kern gesprungen.

remove_theolinks(Path) – **Path** ist eine Liste der Theorie-Kerne, die in den unmittelbar vorangegangenen lokalen Lösungsphasen behandelt wurden. Die Pfadlänge ist durch den entsprechenden globalen Parameter der Strategie vorgegeben.

Im Verlauf der vorangegangenen Lösungssuche erfolgte der Wechsel von einem der in **Path** enthaltenen Theorie-Kerne zum nächsten über einen theoretischen Link. Diese Links werden mit der Ausführung der Funktion gelöscht. Entsprechend wird die *RepL*-Relation aktualisiert.

Lösungsschritt: *S3*		
Parameter:	D	Datenstruktur des aktuellen Theorie-Kerns
	M	Modell des aktuellen Theorie-Kerns
Problem:		Im aktuellen Theorie-Kern wird überprüft, ob die Datenstruktur D zu dem Modell M paßt.
Bei positivem Ergebnis:		
Aktionen:		• IF confirmed(K, P) THEN generate_datalinks(K, P);
Folge-Problem:		Anwendung des Lösungsschritts *S4*
Bei negativem Ergebnis:		
Aktionen:		• IF not_confirmed(Path) THEN remove_theolinks(Path); • change_kernel_by_theolink();
Folge-Problem:		Anwendung des Lösungsschritts *S3*

Lösungsschritt: *S4*

Parameter:	D	Datenstruktur des aktuellen Theorie-Kerns
	M	zu D passendes Modell des aktuellen Theorie-Kerns
Problem:		Im aktuellen Theorie-Kern wird überprüft, ob es noch weitere von M verschiedene Modelle gibt, die zu der Datenstruktur D passen.

Bei positivem Ergebnis:

Aktionen:

- IF confirmed(K, P)
  THEN generate_datalinks(K, P);
- IF confirmed(K, P)
  THEN add_datastruct_and_model();

Folge-Problem: Anwendung des Lösungsschritts *S3*

Bei negativem Ergebnis:

Aktionen:

- IF not_confirmed(Path)
  THEN remove_theolinks(Path);
- change_kernel_by_theolink();

Folge-Problem: Anwendung des Lösungsschritts *S3*

Kapitel 6

Das Simulationsprogramm im Überblick

In den vorangegangenen drei Kapiteln wurden der Begriff einer Theorie, die Problemarten und die Problemlösestrategien erläutert, jeweils in Verbindung mit ihrer Realisierung im Simulationsmodell. Im Sinne von Kapitel 2.3 ist damit das algorithmische Modell und dessen Datenumgebung beschrieben. Um ein ausführbares Simulationsprogramm zu erhalten, ist dieser Kern des Simulationsmodells in einer Ablaufumgebung einzubetten.

Eine solche Ablaufumgebung ist in dieser Arbeit zum einen durch das verwendete Prolog-System gegeben und zum anderen durch Funktionen, die über den Standard eines Prolog-Systems hinausgehen und die speziellen Anforderungen des hier erstellten Simulationsprogramms erfüllen.

Mit diesem speziell entwickelten Teil der Ablaufumgebung werden folgende Funktionsblöcke bereitgestellt:

- Ein Zufallszahlengenerator
- Eine einfache Benutzerschnittstelle
- Die Behandlung der Ergebnisdaten eines Simulationslaufes

Um einen Überblick über das gesamte Simulationsprogramm zu geben, wird nachfolgend dessen programmiertechnische Aufteilung in einzelne Programmmodule skizziert. Näher wird auf die Bedienschnittstelle und die Behandlung der Simulationsergebnisse eingegangen.

6.1 Modulstruktur des Simulationsprogramms

Um eine übersichtliche Entwicklung eines umfangreichen Programms zu gewährleisten, unterteilt man es üblicherweise in Funktionsblöcke und implementiert diese in eigenen Moduln.

In der vorliegenden Arbeit wurden die in Abbildung 6.1 dargestellten Funktionsblöcke unterschieden.

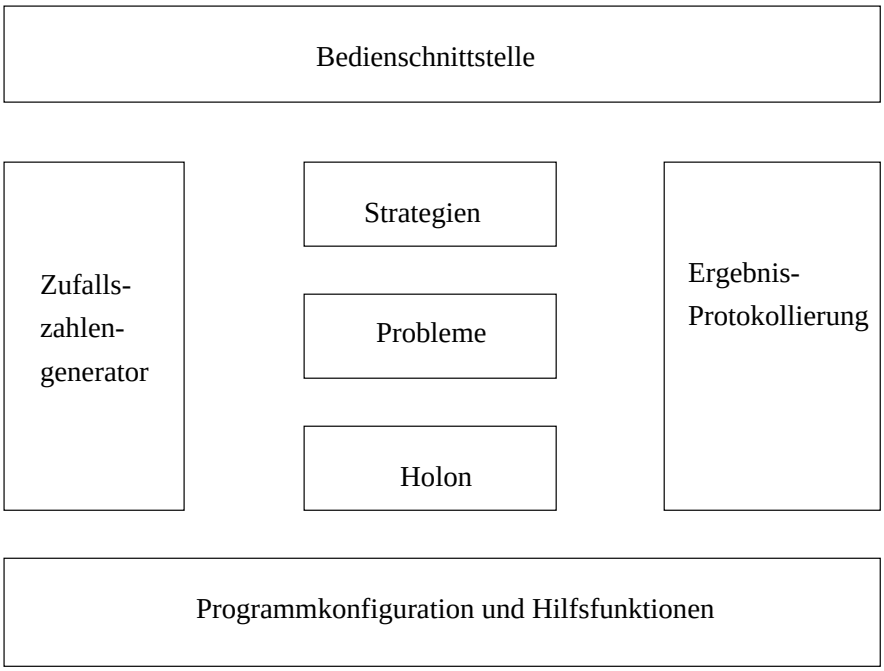


Abbildung 6.1: Modulstruktur des Simulationsprogramms

Das hier entwickelte Simulationsprogramm umfaßt also folgende Module:

Holon
: In diesem Modul sind alle Prolog-Prädikate zusammengefaßt, die die Rechenstruktur Holon realisieren (siehe Kapitel 3.3).

  Bei Start des Simulationsprogramms sind in diesem Modul noch keine Prolog-Fakten enthalten, die einen konkreten Theorie-Holon beschreiben. Solche Fakten werden erst im Programmverlauf generiert bzw. aus Dateien des Programmanwenders geladen.

Probleme
: Faßt die in Kapitel 4.3 eingeführten Prolog-Prädikate zusammen, die die Problemarten beschreiben.

Strategien
Der Modul enthält alle Prolog-Prädikate, die die in Kapitel 5 angegebenen Lösungsschritte beschreiben und deren Ausführung im Rahmen eines Simulationslaufes steuern.

Zufallszahlengenerator
Im Wesentlichen ist hier das Prädikat random/2 gemäß den Angaben aus Kapitel 2.4 definiert:

random(N, R) – **R** ist eine gleichverteilte Pseudozufallszahl zwischen 1 und **N**.

Bedienschnittstelle
Über diesen Modul werden die Interaktionen eines Programmanwenders ermöglicht.

Ergebnisprotokollierung
Mit diesem Modul werden Funktionen bereitgestellt, die die Protokollierung der ausgeführten Lösungsschritte ermöglicht. Die Lösungssuche kann nach Beendigung eines Simulationslaufes anhand der protokollierten Daten nachvollzogen werden (siehe auch Kapitel 6.2).

Programmkonfiguration und Hilfsfunktionen
Dieser Modul enthält Anweisungen, die die übrigen Module des Simulationsprogramms in ein Prolog-System lädt und das Simulationsprogramm damit in einen Initialzustand versetzt. Zusätzlich sind dort allgemeine Hilfsfunktionen zusammengefaßt, die von den übrigen Moduln benutzt werden.

6.2 Ablaufumgebung

Um das Simulationsprogramm zu starten, muß nach Aufruf des Prolog-Systems der Modul „Programmkonfiguration und Hilfsfunktionen“ geladen werden. Damit werden auch alle übrigen Module des Simulationprogramms geladen. Nun kann der Programmanwender entscheiden, welche Funktionen ausgeführt werden sollen.

Die angebotenen Funktionen sind:

- Die Generierung oder das Laden eines Theorie-Holons,
- die Durchführung eines Simulationslaufes und
- die Auswertung und Sicherung der Simulationsergebnisse.

Die beiden ersten Funktionen wurden bereits in den Kapiteln 3.3 und 5 besprochen. Deshalb wird hier nur noch auf den letzten Punkt eingegangen.

Bei jedem Simulationslauf verändert sich die Menge der Prolog-Fakten, die den aktuellen Theorie-Holon repräsentieren. Um diese Änderungen bei Ende des Simulationslaufes nachvollziehen zu können, werden entsprechende Daten mitprotokolliert.

Für jeden Simulationslauf wird festgehalten, welche Strategie gewählt und mit welchen aktuellen Parameterwerten sie gestartet wurde (siehe auch Kapitel 5.2, Seite 50).

Für jeden einzelnen Lösungsschritt, der bei einem Simulationslauf ausgeführt wird, werden folgende Informationen festgehalten:

- Der aktuelle Theorie-Kern, bei dem die Problemlösung stattfindet.
- Die Bezeichnung des aktuellen Lösungsschritts mit Angabe seiner aktuellen Parameter; d. h. welche Datenstruktur und welches Modell betrachtet wird.
- Die durchgeführten Passungsversuche mit Angabe, aufgrund welcher Entscheidung der Passungsversuch positiv bzw. negativ verlief.

Zusätzlich wird vermerkt, wenn Datenstrukturen und Modelle hinzugenommen werden oder, wenn Datenlinks oder theoretische Links eingefügt bzw. gelöscht werden.

Diese Informationen werden in der Prolog-Datenbasis als Fakten des Prädikats step/2 abgelegt, das folgendermaßen definiert ist:

step(N, D) – **N** ist die Folgenummer eines Lösungsschrittes. **D** ist die im **N**-ten Lösungsschritt protokollierte Information.

Durch Zugriff auf die Fakten dieses Prädikats kann also der Ablauf einer Lösungssuche rekonstruiert werden.

Kapitel 7

Ausblick

Mit dem in dieser Arbeit beschriebenen Simulationsmodell wurde der Versuch unternommen, ein Schema für Problemlösestrategien zu entwerfen, nach deren Regeln die Entwicklung einer Theorie erfolgen kann.

Eine Theorie ist dabei anhand des strukturalistischen Begriffs eines Theorie-Elements beschrieben, das Bestandteil eines Theorie-Holons ist. Ein Theorie-Holon besteht im Simulationsmodell aus einer vorgegebenen Anzahl von Theorie-Kernen, die über theoretische Links und Datenlinks miteinander verbunden sind. Während eines Simulationslaufes bleibt die Anzahl der Theorie-Kerne konstant. Änderungen ergeben sich lediglich auf der Ebene der Modelle, der partiellen Modelle und der Datenstrukturen sowie bei den theoretischen Links und den Datenlinks.

Die Entwicklung einer Theorie ist nachvollziehbar anhand der Veränderungen des gegebenen Theorie-Holons, die während der Ausführung des Simulationsmodells in den aufeinander folgenden Schritten der Problemlösestrategie bewirkt wurden.

Aufgabe zukünftiger Arbeiten ist nun die Anwendung des Simulationsmodells. Erst nach der wiederholten Durchführung von Simulationsläufen mit unterschiedlichen Parameterwerten und bei verschiedenen Theorie-Holonen können fundierte Aussagen über das Zusammenspiel von Problemlösestrategien und der Entwicklung von Theorien getroffen werden. Zu erwarten ist dann z. B., daß bei Verwendung einer bestimmten Strategie sich gewisse Muster in der Entwicklung einer Theorie erkennen lassen.

Aufgrund der Simulationsergebnissen läßt sich bewerten, inwieweit die Nachbildung des Theoriebegriffs und der Entwicklung einer Theorie im Simulationsmodell in der gewünschten Weise erfolgt ist und keine konzeptuellen Fehler aufweist.

Der Umgang mit dem Simulationsmodell muß auch zeigen, wie sich die bisher angebotenen Funktionen zur Bedienung des Programms und zur Auswertung

der Simulationsergebnisse bewähren. Verbesserungen könnten sich z. B. dadurch ergeben, daß der Verlauf einer Problemlösung während einer Simulation direkt anhand einer graphischen Darstellung des Theorie-Holons verfolgt werden kann.

Durch das Experimentieren mit einem Simulationsmodell ergeben sich typischerweise auch Situationen, die eine inhaltliche Veränderung des Simulationsmodells selbst erfordern.

Die Beschreibung einer Theorie und der darauf definierten Problemarten können dabei als ziemlich stabil betrachtet werden. Diese Beschreibungsform hat sich nämlich im Rahmen der strukturalistischen Rekonstruktion zahlreicher Theorien bewährt. In dem hier erstellten Simulationsmodell werden solche Änderungen hauptsächlich die Problemlösestrategien betreffen. Die Erweiterung der bereits beschriebenen Strategien oder die zusätzliche Definition neuer Strategien könnte nun z. B. folgende Ziele verfolgen:

- Weitere Problemarten, die bereits im Prototypen realisiert sind, sollen in einer Problemlösungsstrategie einbezogen werden.
- Lösungsschritte sind zu suchen, die auch das Entstehen oder das Absterben eines Theorie-Kerns beschreiben.

Eine zukünftige Weiterentwicklung des Simulationsmodells könnten z. B. darin bestehen, daß eine detailliertere Beschreibung des Theorie-Holons ermöglicht wird.

Um dies zu erreichen, müßte versucht werden, die Beschreibung des Strukturtypen eines Theorie-Kerns mit in das Simulationsmodell aufzunehmen. In eingeschränktem Umfang könnte dies dadurch geschehen, daß nur solche Strukturtypen betrachtet werden, die relativ einfache, mathematische Relationen aufweisen. Einen Anhaltspunkt, welche mathematischen Funktionen hierbei in Betracht kommen, bieten entsprechende Programmbibliotheken.

Mit der Zunahme der Komplexität des Simulationsmodells erhöht sich der Bedarf an Rechnerleistung. Dem kann durch die Wahl eines leistungsstärkeren Computers begegnet werden oder aber durch die Verteilung des Simulationsmodells auf mehrere miteinander vernetzter Computer (siehe [MM89]).

Wählt man den Ansatz der Verteilung , so bietet es sich an, das Simulationsmodell so zu strukturieren, daß die Problemlösungen, die jeweils einen Theorie-Kern eines Holons betreffen, in einem eigenen Prozessor simuliert werden. Die theoretischen Links und die Datenlinks zwischen den Theorie-Kernen ergeben sich dann durch entsprechende Kommunikationskanäle zwischen den Prozessoren.

Mit der verteilten Simulation ergeben sich neben dem technischen Aspekt der Effizienzsteigerung auch inhaltlich interessante Konsequenzen. So kann die parallel verlaufende Entwicklung verschiedener Theorie-Elemente eines Holons dann unmittelbar im Simulationsmodell abgebildet werden.

Die vorliegende Arbeit und das weitere Experimentieren mit dem entwickelten Simulationsmodell kann somit einen kleinen Beitrag dazu leisten, Einblicke in das Zusammenspiel von Problemlösestrategien und der Entwicklung von Theorien zu vermitteln und Ideen zu konkretisieren, in welcher Form die Weiterentwicklung dieses simulativen Ansatzes stattfinden kann.

Literaturverzeichnis

[Bal94] Balzer, W.: *Vorlesung: Wissenschaftstheorie für Fortgeschrittene.* Mitschrift aus einer Vorlesung an der Ludwig-Maximilians-Universität München im Wintersemesters 1993/94, 1994.

[BLZ93] Balzer, W., B. Lauth und G. Zoubek: *A Model for Science Kinematics.* Studia Logica, 52:519–548, 1993.

[BMS87] Balzer, W., C. U. Moulines und J. D. Sneed: *An Architectonic for Science.* Reidel, Dordrecht, 1987.

[BS86] Bünning, H. Kleine und S. Schmitgen: *PROLOG.* Teubner, Stuttgart, 1986.

[BW84] Bauer, F. L. und H. Wössner: *Algorithmische Sprache und Programmentwicklung.* Springer Verlag, Berlin/Heidelberg, 1984.

[CM87] Clocksin, W. F. und C. S. Mellish: *Programming in Prolog.* Springer-Verlag, Berlin/Heidelberg, 1987.

[FG90] Futó, Iván und Tamás Gergely: *Artifical intelligence in simulation.* Ellis Horwood, West Sussex, 1990.

[Knu81] Knuth, D. E.: *The Art of Computer Programming*, Band Vol. 2 Seminumerical algorithms. Addison Wesley, Reading, Massachusetts, 1981.

[Kuh67] Kuhn, Thomas S.: *Die Struktur wissenschaftlicher Revolutionen.* Suhrkamp Taschenbuch, Frankfurt a. M., 1967.

[Lau77] Laudan, L.: *Progress and its Problems.* University of California Press, Berkeley, 1977.

[Lo87] Langley, P. et al.: *Scientific Discovery.* MIT Press, Cambridge, Massachusetts, 1987.

[MM89] Mattern, F. und H. Mehl: *Diskrete Simulation - Prinzipien und Probleme der Effizienzsteigerung durch Parallelisierung.* Informatik-Spektrum, 12:198–210, 1989.

[Sch85] Schmidt, B.: *Systemanalyse und Modellaufbau.* Springer-Verlag, Berlin/Heidelberg, 1985.

[Sie91] Siegert, H.-J.: *Simulation zeitdiskreter Systeme.* Oldenbourg Verlag, München Wien, 1991.

[SS86] Sterling, L. und E. Shapiro: *The Art of Prolog.* MIT Press, Cambridge, Massachusetts, 1986.

[Ste94] Steinhausen, D.: *Simulationstechniken.* Oldenbourg Verlag, München Wien, 1994.

[Tha93] Thagard, P.: *Computational Philosophy of Science.* MIT Press, Cambridge, Massachusetts, 1993.