Wissensbasierte Modellierung:

Eine kurze Einführung in Prolog

Klaus Manhart

Überarbeitete Fassung aus meiner Dissertation

„KI-Modelle in den Sozialwissenschaften“,
Oldenbourg-Verlag, München, 1995

www.klaus-manhart.de
mail@klaus-manhart.de

München, Januar 2008

1. Einleitung

Die Repräsentation von Wissen mit Wenn-dann-Regeln ist ein weit verbreiteter Formalismus, der zusammen mit logischen Darstellungsformen vorgestellt werden soll. Zunächst sind jedoch einige allgemeine Bemerkungen zu „Wissen“ und „Wissensrepräsentation“ angebracht.

Ohne ausführlich auf den Terminus „Wissen“ einzugehen (vgl. hierzu z.B. Reimer 1991 und Heyer/Krems/Görz 1988), begnügen wir uns mit einigen Stichworten zu verschiedenen Wissenskategorien und Repräsentationsformalismen. In Zusammenhang mit Expertensystemen wurde oben bereits eine Unterteilung nach *Herkunft* und *Gebrauch* des Wissens erwähnt. Eine andere Unterscheidung stammt von Barr/Feigenbaum (1981: 144), die folgende *Wissensarten* festlegen:

1. Wissen als Fakten (Eigenschaften, Beziehungen) über Objekte der Realität;
2. Wissen als Kenntnis über Handlungen und Ereignisse;
3. Wissen über bestimmte Fertigkeiten, Vorgehensweisen und Methoden;
4. Meta-Wissen, das heißt Wissen darüber, was wir wissen.

Die Wissensarten (1) und (3) in der Einteilung von Barr/Feigenbaum entsprechen der in der KI/Informatik üblichen Unterscheidung zwischen deklarativem und prozeduralem Wissen, welche bereits in Kap. 3 erwähnt wurde. Bei *deklarativem Wissen* handelt es sich um Tatsachen- oder Faktenwissen, mit dem Sachverhalte ausgedrückt werden, wie etwa bei:

```
(F1) Expertensysteme sind eine spezielle Klasse von Computer-
     programmen.
```

Prozedurales oder Verfahrens-Wissen hat die Form von Algorithmen und kann in Gestalt von Prozeduren dargestellt werden (vgl. z.B. Rechenberg 1991: 227). Das Vorhaben, eine wissenschaftliche Arbeit zu schreiben, kann prozedural wie folgt aufgefasst werden:

```
(F2) Um eine wissenschaftliche Arbeit zu schreiben, überlege dir ein
     Thema, sammle Literatur zum Thema ...
```

Wissens*repräsentation* bezeichnet das Aufschreiben von Symbolen, die in einer erkennbaren Weise einem Ausschnitt einer zu repräsentierenden Welt entsprechen (Reimer 1991: 9). Ein mächtiges und flexibles Mittel zur Repräsentation von Wissen ist die natürliche Sprache, die sich aber nicht zur Rechnerimplementierung eignet (vgl. Reimer 1991: 28-29). Maschinelle *Wissensrepräsentationsmethoden* beschreiben und erstellen Verfahren, das (Experten-)Wissen im Computer zu erfassen und in geeigneter Form zu nutzen. Die wichtigsten maschinellen Repräsentations*formen* sind:

- prädikatenlogische Formeln,
- Regeln,
- semantische Netze
- Objekte/Frames.

Diese Darstellungsverfahren sind überblicksartig beschrieben in Rich (1988), Tanimoto (1990), Winston (1987) und ausführlich in Reimer (1991). In unserem Rahmen genügen Prädikatenlogik und Regeln, so dass wir uns im folgenden darauf beschränken.

2. Regelbasierte Wissensrepräsentation

In Expertensystemen sind Produktionsregeln die verbreitetste Wissensrepräsentationsmethode, da Experten ihr Wissen oft in Regeln formulieren. Produktionsregeln haben grundsätzlich die Form „Wenn x, dann y". Der obengenannte deklarative Wissensausschnitt (F1) lässt sich etwa schreiben als:

```
(F1') Wenn etwas ein Expertensystem ist, dann ist es ein Computer-
      programm,
```

und der prozedurale Wissensausschnitt (F2) als:

```
(F2') Wenn eine wissenschaftliche Arbeit geschrieben werden soll,
      dann überlege dir ein Thema, sammle Literatur ....
```

Sowohl deklaratives als auch prozedurales Wissen lässt sich also in Regelform darstellen, wobei deren Interpretation oder Lesart aber variiert:

- Bei logischer oder *deklarativer* Lesart sind Regeln als *logische Ableitungsregeln* aufzufassen und Wenn-dann-Regeln sind zu lesen als:
  wenn *Bedingung(en)* dann *Konklusion*.
  Ein so interpretiertes regelbasiertes System ist ein *Deduktionssystem*. In diesem Fall kann der Wenn-Teil als *Antezedens* oder *Bedingung*, der Dann-Teil als *Sukzedens, Konklusion, Implikation* oder *Konsequenz* bezeichnet werden.

- Bei *prozeduraler* Lesart produzieren Regeln aus Situationen (linke Seite) Aktionen (rechte Seite) und können gelesen werden als:
  wenn *Situation(en)* dann *Aktion(en)*.
  In diesem Fall kann die rechte Seite als *Handlung* oder *Aktion* angesehen werden, mit der ein Zustand verändert wird.

Je nach Semantik kann sogar die *gleiche* Regel prozedural oder deklarativ interpretiert werden, meist ist aber die Interpretation durch die Problemstellung eindeutig determiniert.

Beispielsweise wird die Regel

```
(F3) Wenn Nackensteife und
     hohes Fieber und
     Bewußtseinstrübung
     dann besteht Verdacht auf Meningitis.
```

am besten deklarativ interpretiert, bei der die dann-Komponente als Implikation oder Deduktion aufgefaßt wird.

Hingegen wird die Regel

```
(F4) Wenn Verdacht auf Meningitis besteht,
     dann nimm sofort Antibiotika.
```

besser prozedural interpretiert, bei der die dann-Komponente eine Handlung ist (Puppe 1988: 21).

Ein wissensbasiertes System wird erst interessant durch die *Verkettung mehrerer Regeln*. Durch die Verkettung vieler Regeln können ganze *Regelnetzwerke* entstehen, die mit steigender Regelmenge immer schwerer überschaubar werden. Der Vorteil des Computers ist hier unübersehbar: mechanisches Ableiten von Hand wäre aufwendig und mit großer Wahrscheinlichkeit inkorrekt und unvollständig. Die folgende Regelmenge stellt beispielsweise bereits ein einfaches Regelnetz dar, in welchem die Regeln miteinander über Konklusion-Antezedens verkettet sind:

```
(N1) Wenn a dann b
     Wenn b dann c
     Wenn c dann d ...
```

Neben Regeln, deren Konklusion sich im Antezedens anderer Regeln findet, kann es Regeln mit gleichem Antezedens und unterschiedlicher Konklusion geben:

```
wenn a dann b,
wenn a dann c,
wenn a dann d.
```

Analog können Regeln existieren mit identischer Konklusion und unterschiedlichem Antezedens:

```
wenn b dann a,
wenn c dann a,
wenn d dann a.
```

Anhand dieser Beispiele lässt sich leicht nachvollziehen, dass bei der Wissensverarbeitung die Regeln mit bestimmten Strategien ausgewählt und manipuliert werden müssen. Diese strategische Abarbeitung der Regeln und das „Navigieren durch das Regelnetz“ übernimmt der Regelinterpreter. Man kann sich den Regelinterpreter einfach als Steuerungsprogramm vorstellen, welches die Reihenfolge der

Abarbeitung lenkt.[1] Grundsätzlich gibt es zwei strategische Alternativen bei der Manipulation von Regeln: Vorwärts- und Rückwärtsverkettung.

Bei der *Vorwärtsverkettung* (*Forward-Chaining*) geht der Regelinterpreter von der vorhandenen Faktenbasis aus, sucht eine Regel, deren Vorbedingungen erfüllt sind und fügt die Konklusion der Faktenbasis hinzu (die Regel „feuert"). Dieser Prozeß wird solange wiederholt, bis keine Regel mehr anwendbar ist. Zur Illustration ein einfaches Beispiel.

Angenommen, in der Faktenbasis steht
`a`
und es seien folgende Regeln gegeben:

```
(R1)  wenn c dann d
(R2)  wenn a und d dann f
(R3)  wenn a dann c
(R4)  wenn g dann j
```

Angenommen weiter, wir wären daran interessiert, ob `f` gilt. Ein vorwärtsverkettender Regelinterpreter würde folgende Fakten produzieren und sie der Faktenbasis hinzufügen (in Klammern die verwendeten Regeln):

```
c (R3)
d (R1)
f (R2)
```

`f` konnte also, unter Verwendung von zwei weiteren Regeln, abgeleitet werden. Da oft mehrere Regeln auf eine bestimmte Faktenmenge anwendbar sind, muss eine Konfliktlösungsstrategie die Auswahl der Regeln übernehmen. Solche Strategien können sein (Puppe 1988: 23):

- Auswahl nach Reihenfolge (die erste oder aktuellste Regel feuert),
- Auswahl nach syntaktischer Struktur (die einfachste oder komplexeste Regel feuert), oder
- Auswahl mittels Zusatzwissen (zum Beispiel Meta-Regeln, die den Auswahlprozeß steuern).

Das Gegenstück zur Vorwärtsverkettung ist die Rückwärtsverkettung. Bei der *Rückwärtsverkettung* (*Backward-Chaining*) geht der Inferenzmechanismus von der zu beweisenden Konklusion aus. Steht die Konklusion nicht in der Datenbasis, werden Regeln gesucht, deren Dann-Teil die Konklusion enthält. Gibt es eine solche Regel, wird versucht, den Wenn-Teil dieser Regel zu beweisen, der als Fakt in der Datenbasis oder wiederum im Dann-Teil einer anderen Regel stehen kann usw. Kann ein Ziel nicht bewiesen werden, kann auch eine Frage an den Benutzer gestellt

[1] In der Terminologie der Wissensverarbeitung beinhaltet die Inferenzmaschine bzw. der Regelinterpreter - meist prozedurales - Meta-Wissen, nämlich Wissen zur Bearbeitung von Wissen.

werden. Auch im rückwärtsverkettenden Fall gibt es Konfliktlösungsstrategien, welche die Regelauswahl steuern.

Ein Beispiel: Angenommen, wir wollen mit den Regeln von (N1) beweisen, dass `c` gilt. `c` steht nicht in der Faktenbasis, aber der Regelinterpeter findet die Regel

```
wenn b dann c.
```

Wir haben nun ein neues Beweisziel `b`, das ebenfalls nicht in der Faktenbasis zu finden ist. Die Inferenzmaschine sucht daher wiederum nach einer Regel, um `b` abzuleiten und findet die Regel

```
wenn a dann b.
```

Unser neues Beweisziel heißt damit `a`. Nehmen wir nun an, es wird keine Regel gefunden, die es ermöglicht, `a` abzuleiten. In diesem Fall wird „`Gilt a?`“ als Frage an den Benutzer gestellt. Antwortet der Benutzer mit „ja“, so ist damit das ursprüngliche Beweisziel `c` hergeleitet, ansonsten schlägt der Beweisversuch für `c` fehl.

Rückwärtsverkettende Systeme haben den großen Vorteil, dass *zielgerichtet* abgeleitet wird und nicht, wie bei der Vorwärtsverkettung, irrelevante Fakten deduziert werden. Beim vorwärtsverketteten Ableiten mit den Regeln (R1)-(R4) wurden beispielsweise zwei Fakten hergeleitet und in die Wissensbasis eingefügt, die gar nicht benötigt werden.

Die rückwärtsverkettende Strategie eignet sich besonders zum gezielten Erfragen noch unbekannter Fakten und findet sich in vielen existierenden Expertensystemen wie MYCIN. Im allgemeinen ist das Rückwärtsschließen besser (effizienter), es gibt aber auch Ausnahmen.

Zu den Vor- und Nachteilen von Vorwärts- und Rückwärtsverkettung vgl. z.B. Savory (1988: 46ff.) oder Puppe (1988). Neben reinen vorwärts- und rückwärtsverkettenden Systemen gibt es noch Verfeinerungen und Mischformen beider Inferenzstrategien, die in unserem Kontext aber nicht von Bedeutung sind.

3. Prädikatenlogisch basierte Wissensrepräsentation: PROLOG

Einer der ältesten Repräsentationsformalismen für Wissen ist die Prädikatenlogik erster Ordnung.[2] Sie ist die theoretisch am besten untersuchte Darstellungsmethode und häufig verwendeter Bezugspunkt für andere Wissensrepräsentationen (Puppe 1988: 16).

Ihre Attraktivität als Medium zur Wissensdarstellung liegt in der schon eingeführten Syntax und Semantik und der Bereitstellung eines formalen Schlussfolgerungsapparats. Die Prädikatenlogik kann nicht nur als abstrakter Repräsentationsformalismus benutzt werden, sondern sie ist direkt als Programmiersprache verwendbar - man spricht in diesem Fall von Logikprogrammierung. Wir führen die prädikatenlogisch basierte Wissensrepräsentation gleich im Kontext von Logikprogrammierung ein.

Logikprogrammierung ist ein direktes Ergebnis von früheren Arbeiten zum automatischen Theorembeweisen und beruht im wesentlichen auf dem Resolutionsprinzip von Robinson (1965). Das Resolutionsprinzip ist ein grundlegendes Verfahren zum mechanischen Deduzieren logischer Formeln aus Axiomen. Die Idee, Logik unmittelbar als Programmiersprache zu benutzen, geht in ihren theoretischen Grundlagen vor allem auf die Arbeiten von Kowalski (vgl. z.B. Kowalski 1988) und Colmerauer zurück (vgl. Lloyd 1993).

Vom Hauptstrom der Computersprachen weicht Logikprogrammierung insofern ab, als - zumindest in ihrer reinen Form - angestrebt wird, *Probleme* lediglich in Form logischer Axiome deklarativ zu *beschreiben*. Eine solche Axiomenmenge bildet eine Alternative zum konventionellen Programm, in dem *Algorithmen* zur Problemlösung zu programmieren sind. Das Programm kann ausgeführt werden, wenn es mit einem Problem - als zu beweisender Aussage formuliert - versorgt wird. Die Ausführung ist ein Versuch, das Problem zu lösen, das heißt, die zu beweisende Aussage mit den Annahmen im Logikprogramm herzuleiten (Sterling/Shapiro 1988: xxiii).

Für die Verwendung der Prädikatenlogik auf Computern ist eine syntaktische Variante der Logik erster Stufe relevant, die Hornclausenlogik, die wiederum ein Spezialfall der Clausenlogik ist. Die verbreitetste Logik-Programmiersprache, PROLOG (PROgramming in LOGic), kann als informatische Realisierung der Hornclausenlogik betrachtet werden. Für das weitere Vorgehen erscheint es zweck-

---

[2] Der im Text schon mehrfach verwendete Zusatz „erster Ordnung“ oder „erster Stufe“ bedeutet, dass All- und Existenzaussagen nur über Individuen getroffen werden können (für alle x: p(x)), während in der Prädikatenlogik zweiter Stufe auch über Mengen und Eigenschaften quantifiziert werden kann (für alle p: p(x)). Die ausdrucksstärkere Logik zweiter Stufe hat die unangenehme Eigenschaft, dass sie weder vollständig noch entscheidbar ist, während die Logik erster Stufe auch nicht entscheidbar, aber immerhin vollständig ist. Logiken zweiter (und höherer) Ordnung spielen in der KI und Wissenschaftstheorie eine untergeordnete Rolle.

mäßig - nach einer kurzen Vorstellung der Clausenlogik - Aufbau und Arbeitsweise von PROLOG zumindest in den Grundzügen darzulegen. Wir knüpfen dabei an das Grundmodell der Balancetheorie von Heider an. Vereinfachungen und Kürzungen werden bewusst in Kauf genommen, da der Rahmen der Arbeit eine detailliertere Darstellung nicht erlaubt.

4. Clausen, Hornclausen, PROLOG

Der grundlegende Term bei automatischen Beweisern ist der Begriff der Clause. Eine *Clause* ist eine spezielle prädikatenlogische Formel der folgenden Form:[3]
$B_1 \vee B_2 \vee ... \vee B_m \leftarrow A_1 \wedge A_2 \wedge ... \wedge A_n \qquad (m,n \geq 0)$.

In dieser Clause ist „$\vee$„ das logische „oder“ und „$\wedge$„ das logische „und“. Das logische „wenn-dann“ „$\rightarrow$„ wird hier aus Gründen, die mit PROLOG zusammenhängen, umgekehrt geschrieben; „$\leftarrow$„ ist somit zu lesen als „dann-wenn“.

Atomformeln B_i, A_j (i=1,...,m; j=1,...,n) haben die Form $p(t_1,...,t_k)$, wobei p ein k-stelliges *Prädikat* ist und $t_1,...,t_k$ Terme sind. Ein *Term* ist entweder eine *Variable* $x_1,...,x_k$, eine *Konstante* $a_1,...,a_k$ oder ein Ausdruck der Form $f(t_1,...t_k)$, wobei f ein k-stelliges Funktionssymbol (*Funktor*) und $t_1,...,t_k$ Terme sind. Ein Prädikat p bzw. Funktor f der Stelligkeit k wird auch mit p/k bzw. f/k bezeichnet.

Eine *Hornclause* (benannt nach dem Logiker Horn) ist eine spezielle Clause mit $m \leq 1$. Wir beschränken uns im folgenden auf Hornclausen, also auf Clausen der folgenden Form:
$B_1 \leftarrow A_1 \wedge A_2 \wedge ... \wedge A_n$.

$A_1, A_2,...,A_n$ sind die konjunktiv verknüpften Bedingungen, B_1 ist die Konklusion der Hornclause. Die Konklusion B_1 ist der *Kopf* der Formel, der durch „$\leftarrow$„ vom *Rumpf* (Körper) getrennt wird. Die Formel kann gelesen werden als:

B_1, wenn A_1 und A_2 und ... und A_n.

Enthält eine Hornclause Variablen $x_1,..., x_k$, so ist dies zu interpretieren als:
Für alle $x_1,...,x_k$ gilt: $B_1 \leftarrow A_1 \wedge A_2 \wedge ... \wedge A_n$.

Für die folgenden Beispiele sei der Zeichenvorrat der Hornclausenlogik nach PROLOG-Konvention festgelegt (vgl. Kleine-Büning/Schmitgen 1986):

(1) Die Menge der Prädikatsymbole P, Funktionssymbole F und Konstanten K bestehe aus Zeichenketten des lateinischen Alphabets, beginnend mit Kleinbuchstaben;
(2) Variablen V seien: $x_1,...,x_k$;

[3] Vgl. zum folgenden Kowalski (1988), Kap.1.

(3) $P \cap F \cap K \cap V = \varnothing$.

Der so festgelegte Zeichenvorrat entspricht bereits dem der Programmiersprache PROLOG mit Ausnahme von (2): in PROLOG sind Variablen Zeichenketten, die mit Großbuchstaben beginnen.

Im Hinblick auf PROLOG sind vier Fälle zu unterscheiden (vgl. Belli 1986: 60ff.):

(F1) $m = 1, n = 0$: $B_1 \leftarrow$
Dies ist eine (Atom-)Formel, die von keiner Bedingung abhängt; wir schreiben einfachheitshalber: B_1.

Wenn wir im folgenden Prädikate und Definitionen aus der Balance-Theorie von Fritz Heider verwenden, dann wäre zum Beispiel `positiv(a,b)` eine atomare (Horn-) Clause mit dem 2-stelligen Prädikat `positiv/2`, den Konstanten `a` und `b` und der Heider-Interpretation, nach der zwischen a und b eine positive Relation besteht. Ebenso ist `positiv(ehemann(inge_meier),cdu)` eine Atomformel mit der Variation, dass das erste Argument von `positiv/2` ein 1-stelliger Term ist mit dem Funktor `ehemann/1` und dem Argument `inge_meier`.

(F2) $m,n \neq 0$: $B_1 \leftarrow A_1 \wedge A_2 \wedge ... \wedge A_n$
Dies ist der übliche dann-wenn-Fall.
Beispielsweise ist
`gleichgewicht(triade(`x_1`,`x_2`,`x_3`)) ← positiv(`x_1`,`x_2`) ∧ negativ(`x_1`,`x_3`) ∧ negativ(`x_2`,`x_3`)`
eine Hornclause mit einer Konklusion und drei Bedingungen. Die Formel soll die Heider-Definition ausdrücken:
Für alle x_1, x_2, x_3: Die Triade x_1, x_2 und x_3 ist im Gleichgewicht, wenn zwischen x_1 und x_2 eine positive Relation, zwischen x_1 und x_3 eine negative Relation und zwischen x_2 und x_3 ebenfalls eine negative Relation besteht.

Eine andere Hornclause, die besagt, dass alle Entitäten, die im Gleichgewicht sind, stabil sind, wäre:
`stabil(`x_1`) ← gleichgewicht(`x_1`).`
Variablen gleichen Namens in Kopf und Rumpf sind aneinander gebunden.
Formeln des Typs (F2) bezeichnet man als Regeln. *Hornclausenlogik erlaubt also eine logische Darstellung von Regeln*.

(F3) $m = 0, n \neq 0$: $\leftarrow A_1 \wedge A_2 \wedge ... \wedge A_n$
Dies ist eine Formel, die nur aus Bedingungen besteht. Sie kann interpretiert werden als:
es ist nicht der Fall, dass gilt: A_1 und A_2 und ... und A_n.
Beispielsweise ist

← `positiv(a,b)` ∧ `negativ(a,c)`
zu lesen als: „es ist nicht der Fall, dass zwischen a,b eine Positiv- und zwischen a,c eine Negativrelation existiert“.

(F4) n,m = 0: ←
Dies ist die leere Clause, die in einem PROLOG-Programm selbst nie erscheint, die aber beweistechnisch wichtig ist.

Wir haben uns bislang immer noch im Rahmen der Hornclausenlogik bewegt. Der Schritt von der Hornclausenlogik zu ihrer informatischen Realisierung PROLOG ist aber nur mit minimalen Änderungen verbunden und lediglich bedingt durch die maschinelle Implementierung.[4]

Zunächst wird in PROLOG syntaktisch nicht unterschieden zwischen Funktoren und Prädikaten. Prädikate werden deshalb in der Literatur ebenfalls als Funktoren bezeichnet. Dies hat die Konsequenz, dass die syntaktische Elementarform von PROLOG nicht die Atomformel ist, sondern der Term: Terme sind allgemeiner als Atomformeln, da jede Verknüpfung von Termen wiederum Terme ergibt. Die rekursive Definition ermöglicht, dass PROLOG den Term als einzige syntaktische Grundform kennt.

Zusammengesetzte Terme heißen in PROLOG auch Strukturen und lassen sich als Bäume darstellen, wie etwa `g(f(X),a,h(Y,X,b))`.

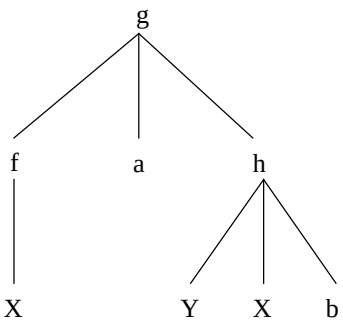

Abb. 1: Der PROLOG-Term `g(f(X),a,h(Y,X,b))` als Baumstruktur

PROLOG-Terme sind Variablen, Konstanten oder Strukturen. Variablen sind - im Unterschied zur Clausenlogik - Zeichenketten beginnend mit Großbuchstaben. Einen Sonderfall bildet die sog. anonyme Variable „_“, die immer dann benutzt wird, wenn es auf den Wert der Variable nicht ankommt. Konstanten sind Integers oder Zeichenketten, die mit Kleinbuchstaben beginnen oder in Hochkomma eingeschlossen sind. Konstanten können im übrigen als Funktoren der Stelligkeit 0 betrachtet werden.

[4] An dieser Stelle sei jedoch schon darauf hingewiesen, daß real existierende PROLOG-Interpreter Sprachkonstrukte enthalten, welche die Hornclausenlogik erweitern bzw. durchbrechen.

Wir können nun PROLOG-Programme konkreter charakterisieren. Ein PROLOG-Programm ist eine Menge von Hornclausen, die unterschieden werden in

- Fakten,
- Regeln,
- Fragen.

Das eigentliche Programm besteht nur aus Fakten und Regeln. Fakten entsprechen variablenfreien Hornclausen der Gestalt (F1), Regeln Hornclausen der Gestalt (F2) und Anfragen solchen der Gestalt (F3). Die oben gegebenen Heider-Beispiele sehen in der verbreitetsten PROLOG-Notation (Clocksin/Mellish 1987) wie folgt aus.

```
positiv(a,b).
positiv(ehemann(inge_meier),cdu).
gleichgewicht(triade(X,Y,Z)) :-
   positiv(X,Y),
   negativ(X,Z),
   negativ(Y,Z).
stabil(X) :- gleichgewicht(X).

?- positiv(a,b), negativ(a,c).
```

Im Vergleich zur Hornclausenlogik sind folgende Notationsänderungen zu verzeichnen:
- bei Regeln statt „←„ „:-“ ;
- statt „∧„ ein Komma;
- bei Anfragen statt „←„ ein „?-“;
- Variablen sind Strings, beginnend mit Großbuchstaben;
- jede Clause ist mit einem Punkt abzuschließen.

Wie lässt sich nun die Hornclausenlogik bzw. PROLOG zur Modellierung verwenden? Im allgemeinen eignen sich Fakten zur Darstellung von Eigenschaften von Objekten und Beziehungen zwischen Objekten. Regeln können zur Repräsentation von Gesetzen (Allaussagen) und zur Definition neuer, komplexerer Relationen auf der Basis existierender Beziehungen verwendet werden. Anfragen schließlich dienen zur Gewinnung von Informationen aus dem Programm. Aus dem eben Dargelegten ergeben sich etwas systematischer folgende Modellierungsmöglichkeiten mit Logik und PROLOG (für weitere Details vgl. Reimer 1991: 35-54):

Eigenschaften lassen sich mit 1-stelligen Prädikaten repräsentieren, zum Beispiel „a ist Philosoph“ mit `philosoph(a)`. n-stellige Beziehungen können durch n-stellige Prädikate ausgedrückt werden, zum Beispiel „a ist verheiratet mit b“ durch
`verheiratet_mit(a,b).`
oder „a sanktioniert b mit c“ durch
`sanktioniert_mit(a,b,c).`

Einfache zeitliche Aspekte können durch Einführung einer zusätzlichen Argumentstelle dargestellt werden, zum Beispiel „a ist mit b verheiratet zum Zeitpunkt 2“ durch `verheiratet_mit(a,b,2).`

Unsicheres Wissen kann durch Zuordnung numerischer Sicherheitsfaktoren ebenfalls durch eine weitere Argumentstelle abgebildet werden, zum Beispiel „a ist ein ziemlich guter Freund von b“ durch: `freund(a,b,0.8)`. Die Zahl 0.8 an dritter Argumentstelle soll dabei ausdrücken, dass die Aussage „ziemlich sicher“ zutrifft.

Definitionen und Allaussagen wie „alle Entitäten mit der Eigenschaft P haben die Eigenschaft Q“ können mit Regeln ausgedrückt werden. Der Allsatz „Alle Soziologen haben eine Statistikausbildung“ kann formuliert werden als:
`hat_statistik_ausbildung(X) :- soziologe(X).`

Das Analoge gilt für Definitionen. „x ist Mutter von y, wenn y Kind von x ist und x weiblich ist“ lässt sich mit folgender PROLOG-Regel repräsentieren:
`mutter(X,Y) :- kind(Y,X), weiblich(X).`

Formale relationale Eigenschaften wie Symmetrie von Prädikaten können ebenfalls in Regelform ausgedrückt werden, zum Beispiel die Symmetrie der Heiratsrelation durch:
`verheiratet_mit(X,Y) :- verheiratet_mit(Y,X).`

Angemerkt sei, dass eine PROLOG-Regel *immer* die gleiche syntaktische Grundstruktur hat, nämlich: *genau eine* Atomformel im Kopf (Konklusion) und *eine oder mehrere* Atomformeln im Rumpf (Bedingungen). Die Atomformeln im Rumpf können durch „und“ (in PROLOG: Komma) oder „oder“ (in PROLOG: Semikolon) logisch verknüpft sein.

Die genaue Syntax (und Semantik) von PROLOG ist beschrieben in Bratko (1987: 29-67), Clocksin/Mellish (1987: 21-42) oder Kleine-Büning/Schmitgen (1986: 70-78).

5. Deduktion in PROLOG

Bislang wurde nur die *Repräsentation* von Wissen mit PROLOG dargestellt, nun soll die *Wissensmanipulation*, also die Herleitung von neuem Wissen, betrachtet werden. Die Deduktion neuen Wissens kann in PROLOG als Theorembeweisen angesehen werden: PROLOG kann als automatischer Theorembeweiser interpretiert werden, der aus Axiomen Theoreme herleitet. Das Deduktionsprinzip von PROLOG soll nun konkret an einem einfachen Beispiel gezeigt werden.

Wir benutzen für die weiteren Erläuterungen die folgende einfache Wissensbasis mit der oben genannten Interpretation, bestehend aus sechs Fakten und zwei Regeln:

```
positiv(a,b).
positiv(a,d).
negativ(a,f).
negativ(b,c).
negativ(d,c).
negativ(a,c).

gleichgewicht(triade(X,Y,Z)) :-
   positiv(X,Y),
   negativ(X,Z),
   negativ(Y,Z).

stabil(X) :-
   gleichgewicht(X).
```

Eine Anfrage an eine gegebene Wissensbasis wird als *Beweisziel* oder *Goal* interpretiert.

Das Beweisziel
`?- positiv(a,b), positiv(a,c).`
besteht aus zwei konjunktiv verknüpften Termen, also zwei zu beweisenden Teilzielen. Das erste Goal kann bewiesen werden, da es mit dem entsprechenden Fakt in der Datenbasis „matcht“. Ein Term *matcht*[5] (oder *unifiziert*) mit einem anderen Term, falls

- beide Terme gleich sind - was hier der Fall ist -, oder
- falls Variablen im Spiel sind, diese in beiden Termen so gebunden werden können, dass nach der Substitution der Variablen beide Terme gleich sind (Bratko 1987: 38).

Die Substituierung einer Variablen durch einen Term (Konstante, Struktur oder andere Variable) nennt man *Instantiierung*, und die Stelle in der Datenbasis wird markiert.

Das zweite Teilziel `positiv(a,c)` ist nicht erfolgreich und scheitert, da kein entsprechender Term in der Datenbasis matcht. Mit dem Scheitern des zweiten Teilziels scheitert das ganze Beweisziel, und es wird die Antwort `no` ausgegeben. PROLOG antwortet bei Anfragen *ohne* Variablen grundsätzlich mit `yes` oder `no` . Im ersten Fall bedeutet dies einen *erfolgreichen* Beweis, das heißt, dass der angefragte Fakt logisch aus der bestehenden Wissensbasis deduziert werden kann, im zweiten Fall bedeutet dies ein Scheitern, das heißt, der Fakt folgt nicht aus der aktuell bestehenden Wissensbasis.

Bei Abfragen *mit* Variablen wird die Instantiierung der Variablen ausgegeben.
Die Anfrage

[5] Von "Pattern-Matching" (Mustererkennung); „matchen“ könnte man vielleicht am besten mit „passen“ übersetzen, wir behalten aber, wie in der Literatur üblich, den Originalterm bei.

```
?- positiv(a,X).
```
matcht als erstes mit dem Fakt `positiv(a,b)` und liefert als Output
```
X = b.
```

Die Variable `X` wird hierbei gebunden an `b` bzw. unifiziert mit `b`.
Ein anschließend eingegebenes Semikolon ; mit der Bedeutung des logischen „oder“ hebt die Variablenbindung wieder auf, und PROLOG sucht die nächste Lösung:
```
X = d.
```

Eine weitere Alternativlösung existiert nicht, und PROLOG würde auf eine entsprechende Aufforderung mit `no` antworten.

Logisch betrachtet sind Variablen in Fragen existenzquantifiziert (Sterling/Shapiro 1988: 6). Die PROLOG-Frage
```
?- positiv(a,X)
```
ist also zu lesen als: Gibt es ein `X`, so dass `positiv(a,X)` gilt; die konjunktive Anfrage
```
?- p(X), q(X)
```
als: Gibt es ein solches `X`, dass beides wahr ist, `p(X)` und `q(X)`.[6]

Wir haben uns bislang bei der Erläuterung des Beweisverfahrens auf Fakten beschränkt. Für die Anfrage
```
?- gleichgewicht(Triade).
```
ist kein Fakt vorhanden, aber ein Prädikat gleicher Stelligkeit findet sich als Konklusion einer Regel. Die Frage matcht also mit dem Kopf der obigen Regel, wobei die Variable `Triade` in der Regel instantiiert wird mit dem Term `triade(X,Y,Z).` Grundsätzlich sind Variablen in Regeln allquantifiziert.

PROLOG geht nun als *rückwärtsverkettender Regelinterpreter* vor: es versucht nach und nach, alle Bedingungen (Teilziele) der Regel mit den entsprechenden Instantiierungen zu beweisen. *Dieser Regelinterpreter ist impliziter Teil der Sprache PROLOG und muss nicht extra erstellt werden.* Das erste Teilziel `positiv(X,Y)` matcht mit `positiv(a,b)`, so dass `X` nun gebunden wird an `a` und `Y` an `b`. Wird eine Variable innerhalb einer Regel gebunden, so ist sie bei allen Vorkommnissen in der Regel an denselben Wert gebunden. Das zweite Teilziel lautet also folglich `negativ(a,Z).` `negativ(a,Z)` matcht mit `negativ(a,f)`, wobei `Z=f`. Das dritte Teilziel heißt somit jetzt `negativ(b,f).` Eine Inspektion der Faktenbasis zeigt, dass kein entsprechender Eintrag vorhanden ist, und der Beweis für das dritte Teilziel scheitert damit.

Nun setzt der für die Lösungssuche zentrale Mechanismus des *Backtracking* (Rücksetzen) ein: da das dritte Beweisziel gescheitert ist, erfolgt eine Lösung der ge-

[6] Variablen in Fakten sind ebenfalls allquantifiziert. Allerdings kommen in der Praxis Variablen in Fakten nicht vor.

bundenen Variablen `Z=f` und ein Rücksetzen zum zweiten Teilziel. Es wird nun ein erneuter, alternativer Beweis für `negativ(a,Z)` versucht. Mit `negativ(a,Z)` matcht jetzt der letzte Fakteneintrag `negativ(a,c),` wobei `Z=c`. Das dritte Teilziel lautet also `negativ(b,c)` und ist als Eintrag in der Datenbasis ebenfalls vorhanden. Somit sind sämtliche Teilziele der Regel mit den dabei erfolgten Instantiierungen bewiesen, und es wird die Antwort ausgegeben:

```
Triade = triade(a,b,c).
```

Ein nun eingegebenes Semikolon würde ein vom Benutzer initiiertes Backtracking auslösen. PROLOG würde nach einer zweiten Lösung suchen und diese mit

```
Triade = triade(a,d,c)
```

finden. Eine weitere Alternativ-Lösung existiert nicht und die Eingabe von ; würde mit `no` beantwortet werden.

Wir beschließen damit die elementare Einführung in den PROLOG-Beweismechanismus und fassen die grundlegende Arbeitsweise von PROLOG zusammen: PROLOG-Ableitungen erfolgen über (Pattern-)Matching, Top-Down-Suche, Backward-Chaining und Backtracking:

- Matching: Für jedes Beweisziel wird ein Matching versucht. Ein Term T1 matcht mit einem Term T2, wenn Funktor und Stelligkeit von T1 und T2 identisch sind, sowie die Argumente entweder gleich sind oder durch Variablenbindung (Instantiierung) gleich gemacht werden können.
- Top-Down: Für jedes Beweisziel wird in der Faktenbasis *von oben nach unten* das Matching versucht.
- Backward-Chaining: ist das Beweisziel der Kopf einer Regel (die Konklusion), so wird ausgehend von der Konklusion versucht, nach und nach die Bedingungen zu beweisen. Ist ein Teilziel wiederum ein Regelkopf, wird erst diese Regel - die eventuell wiederum weitere Regelköpfe als Teilziele enthalten kann - abgearbeitet, bevor die weiteren Teilziele geprüft werden. Damit ist PROLOG ein *rückwärtsverkettender Regelinterpreter*.
- Backtracking: Führt ein Teilziel nicht zum Erfolg, werden die Instantiierungen, die bei diesem Teilziel erfolgten, gelöst, und es erfolgt ein Zurücksetzen (Backtracking) zum vorhergehenden Teilziel. Die Eingabe eines Semikolons nach Ausgabe einer Lösung bewirkt ein vom Benutzer initiiertes Backtracking.

Ein PROLOG-Interpreter implementiert somit unmittelbar eine rückwärtsverarbeitende Top-Down Inferenzmaschine, ohne dass dies explizit programmiert werden muss. Angemerkt sei noch, dass logikbasierten Repräsentationen und insbesondere PROLOG die „Closed World Assumption“ zugrunde liegt. Diese Annahme besagt, dass eine Aussage als nicht wahr angesehen wird, wenn sie aus der *vorliegenden* Wissensbasis nicht abgeleitet werden kann.

6. Listen, Mengen und Rekursion

Bei der Repräsentation von Theorien spielen Mengen eine wichtige Rolle. Mengen werden in PROLOG in Listen repräsentiert.
Eine Liste

- ist entweder leer,
- oder besteht aus mehreren Elementen.

Die leere Liste wird dargestellt als `[]`, eine nicht-leere Liste durch `[e1,e2,e3,....,en]`. Das erste Element einer nicht-leeren Liste - `e1` - ist der *Kopf*, der Rest `[e2,e3,...,en]` ist der *Rumpf* (*Rest*, *Schwanz*) der Liste. Der Rumpf einer Liste ist wieder ein Liste, der Kopf dagegen nicht. Der Rumpf hat abermals einen Kopf `e2` und den Rumpf `[e3,...,en]` usw. Die Liste `[en]` schließlich hat den Kopf `en` und den Rumpf `[]`.

Alle Elemente einer Liste müssen vom Typ Term sein. Die Liste selbst ist auch ein Term (genauer gesagt: eine Struktur), das heißt, die Elemente einer Liste können wiederum Listen sein, deren Elemente wiederum Listen sein können etc.

Beispiele für gültige Listen sind:

```
[a,b,c]
[a,1,a,b,2,d]
[mo,di,mi,do,fr,sa]
['Beispiel',einer,liste,in,'Prolog']
[dies, ist, [eine,[verschachtelte,liste]]]
```

Da in Listen Elemente mehrfach vorkommen dürfen und der Reihenfolge nach geordnet sind, sind Listen streng genommen keine Mengen. Trotzdem können Listen natürlich zur Repräsentation von Mengen verwendet werden.

Für die Trennung einer Liste in Kopf und Rumpf steht der Listen-Separator | zur Verfügung: mit `[X|Y]` wird `X` instantiiert auf den Kopf der Liste und `Y` auf den Rumpf. In der folgenden Tabelle finden sich einige Beispiele für Unifikation/Matching zweier Listen.

Tab. 1: Drei Instantiierungsbeispiele

Liste1	Liste2	Instantiierung
`[a]`	`[X|Y]`	`X=a Y=[]`
`[a,b,c]`	`[X|Y]`	`X=a Y=[b,c]`
`[1,2,3]`	`[X,Y|Z]`	`X=1 Y=2 Z=[3]`

In Zusammenhang mit Listen und Listenoperationen - aber nicht nur damit - sind Rekursion und rekursive Definitionen von zentraler Bedeutung. Eine rekursiv definierte Regel ist eine Regel, die sich selbst wieder aufruft, das heißt, der Kopf der Regel erscheint im Regelkörper noch einmal. Ein beliebtes Beispiel ist die `member`-Relation, die prüft, ob ein Element in einer Liste enthalten ist. Beispielsweise liefert

```
?- member(a,[b,c,a]).
yes
```

weil `a` Element der Liste `[b,c,a]` ist,

```
?- member(a,[b,d]).
no
```

weil a nicht in der Liste enthalten ist.

Das Programm für die `member`-Relation kann rekursiv wie folgt definiert werden:
X ist Mitglied einer Liste L, wenn
(1) X der Kopf von L ist oder
(2) X ein Mitglied des Rests von L ist.

Dies kann in Form von zwei PROLOG-Clausen beschrieben werden:

```
member(X,[X|_]).
member(X,[_|Rest]) :-
   member(X,Rest).
```

`member(X,[X|_])` ist dabei der Rekursionsanfang, der als Faktum vor der Regel steht. Eine Anfrage ist erfolgreich, wenn `X` Kopf der Liste ist. Andernfalls scheitert die erste Clause, und es wird ein Beweis mit der zweiten Clause - einer Regel - versucht. Die Regel ruft sich rekursiv selbst wieder auf mit der Rumpfliste an zweiter Argumentstelle, und die erste Clause prüft nun wieder, ob `X` Kopf der (Rumpf-)Liste ist usw.

Generell gilt, dass auf Listenelemente nur über den Kopf zugegriffen werden kann. Um ein Element mit einer bestimmten Eigenschaft zu finden, muss also zunächst der Kopf geprüft werden. Ist die Prüfung negativ, so wird die Liste „geköpft“, das heißt, das erste Element wird entfernt, und der Kopf des Rumpfes wird geprüft. Der Prüfprozess wird solange fortgesetzt, bis das gesuchte Element gefunden ist oder die leere Liste übrig bleibt.

Nicht alle Argumente von `member/2` müssen gebundene Variablen sein. Enthält die erste Argumentstelle eine Variable, die zweite eine variablenfreie Liste, so erhalten wir als erste Lösung eine Unifikation der Variablen mit dem ersten Listenelement, als zweite eine Unifikation mit dem zweiten usw.

`member(X,[a,b,c,d]).`
würde also liefern
`X = a`
und ein anschließend ausgelöstes Backtracking mit ;
`X = b` usw.

Man sieht daran, dass einfache Prädikate wie `member/2` flexibel genutzt werden können.

Als abschließendes Beispiel definieren wir ein Prädikat, das die Gleichheit zweier Mengen prüft und später benötigt wird. Da Listen in PROLOG nur gleich sind, wenn diese die gleichen Elemente in übereinstimmender Reihenfolge enthalten, sind die Listen `[a,b,c,d]` und `[b,a,c,d]` nicht identisch in PROLOG, obwohl sie mengentheoretisch gleich sind. Eine einfache Gleichheitsprüfung von Listen ist für Mengen also nicht geeignet.

Wir müssen deshalb ein entsprechendes Prädikat definieren: das Prädikat `gleiche_menge(X,Y)` soll wahr sein, wenn `X` und `Y` mengentheoretisch gleich sind. Wir setzen hier voraus, dass weder `X` noch `Y` doppelte Vorkommnisse von Elementen enthalten, also echte Mengen sind - was wiederum mit einem eigens definierten Prädikat geprüft werden könnte.

Wir definieren das Prädikat `gleiche_menge(X,Y)` unter Ausnutzung der mengentheoretischen Grundbeziehung:
$X = Y$ gdw $X \subseteq Y$ und $Y \subseteq X$.

Dass die Listen X und Y (mengentheoretisch) gleich sind, lässt sich damit einfach unter Reduktion auf die Teilmengen-Relation definieren:

```
gleiche_menge(X,Y) :-
   teilmenge(X,Y),
   teilmenge(Y,X).
```

Die rekursive Definition der Teilmengen-Relation lautet:
(1) Die leere Menge ist Teilmenge jeder Menge.
(2) X ist Teilmenge von Y, wenn der Kopf von X Mitglied von Y ist und der Rumpf Teilmenge von Y ist.

```
teilmenge([],Y).
teilmenge([K|X],Y) :-
   member(K,Y),
   teilmenge(X,Y).
```

Das letzte Beispiel eignet sich zur Exemplifizierung von drei wichtigen Eigenschaften von PROLOG, die es für die Modellierung so interessant machen:

1. Die semantische Lücke zwischen Quellsprache (hier: Mengenlehre) und PROLOG als Zielsprache ist gering. Die Definition der mengentheoretischen Gleichheit in PROLOG kann - zumindest auf der oberen Ebene - genauso geschrieben werden wie in der Quellsprache.

2. Es muss kein Algorithmus festgelegt werden, der bestimmt, welche Schritte zur Verifizierung der Mengengleichheit ausgeführt werden müssen. Vielmehr erfolgt die Definition der Mengengleichheit rein beschreibend.

3. Komplexe Prädikate lassen sich auf weniger komplexe, immer einfachere Prädikate zurückführen, so dass sich die oben erwähnte hierarchische Strukturierung ergibt: das Prädikat `gleiche_menge/2` wird unter Rückgriff auf `teilmenge/2` definiert und dieses unter Rückgriff auf `member/2`. Wie im nächsten Kapitel gezeigt wird, können Prädikate auch als Prozeduren interpretiert werden, so dass das Beispiel den oben eingeführten Begriff der Prozedurabstraktion verdeutlicht.

Mit PROLOG-Listen sind alle gebräuchlichen mengensprachlichen Operationen durchführbar. Typische Operationen auf Mengen sind Durchschnitts-, Vereinigungs-, Differenzmengenbildung oder die Bildung des Kreuzprodukts zweier Mengen. Das Kreuzprodukt würde eine Menge von geordneten Paaren liefern, in PROLOG eine Liste von zweielementigen Listen: `[[a,b],[a,c],[a,d],...]`. Die Elemente der inneren Listen (also die der Listenpaare) sind hier als geordnet aufzufassen und die Elemente der äußeren Liste als ungeordnet. Soll dieser Unterschied deutlicher hervorgehoben werden, kann die Kreuzproduktliste auch so dargestellt werden: `[(a,b),(a,c),(a,d),...]`.

Allgemeine, nicht ausschließlich mengenspezifische Listenoperationen sind zum Beispiel: Listen zusammenhängen, Listenelemente entfernen, Bestimmung des letzten Listenelements, Länge einer Liste bestimmen, Listenpermutationen, Listenumkehr oder Sortieren von Listen. Prädikate zur Definition dieser Operationen finden sich zum Beispiel in Kleine-Büning/Schmitgen (1986) oder Belli (1986).

7. Interpretation und Beweisverfahren von PROLOG-Programmen

Wir haben bislang die deklarative Sicht der Logikprogrammierung eingenommen und werden diese im folgenden auch weitgehend beibehalten. In der deklarativen oder logischen Bedeutung *beschreibt* ein Logikprogramm einen Wissensausschnitt mit Fakten und Regeln. Fakten und Regeln werden als eine Menge von Axiomen verstanden, die Anfrage als ein vermutetes Theorem und PROLOG versucht, dieses Theorem zu beweisen und aus den Axiomen abzuleiten (Bratko 1987: 21). Die deklarative Bedeutung bestimmt also, ob ein Ziel wahr ist, und falls ja, für welche Variablenwerte.

Die Regel
```
p(X) :- q(X), r(X)
```
ist damit unter logischer Betrachtungsweise zu lesen als:
```
p(X), wenn q(X) und r(X).
```
oder

`p(X)` ist wahr, wenn `q(X)` wahr ist und `r(X)` wahr ist.

Die Regeln können dabei als Axiome betrachtet werden, die zur Definition von *neuen und komplexeren Relationen* unter Verwendung von bereits *bestehenden, einfachen Relationen* verwendet werden.

Soweit möglich, sollte PROLOG deklarativ gesehen werden, und es sollte insbesondere die bevorzugte Interpretation des Modellbauers sein. Obwohl die deklarative Betrachtungsweise elegant, einfach und vorteilhaft ist, ist in der Praxis aber oft die prozedurale Betrachtung nicht zu vermeiden.[7] Die prozedurale Bedeutung von PROLOG-Programmen bestimmt, *wie* eine Ausgabe erhalten wird, das heißt, wie das PROLOG-System Fakten und Regeln auswertet. Prozedural interpretiert ist eine Anfrage eine „ereignisgesteuerte Aktivierung einer Prozedur" (Schnupp/Nguyen Huu 1987: 60):

Anfragen lösen einen Prozeduraufruf bzw. eine Sequenz von Prozeduraufrufen aus, wobei der Funktor eines Terms den *Prozedurnamen* darstellt und die Argumente die *Übergabeparameter*. Eine Frage löst dann den Aufruf eines Faktes oder des Kopfes einer Regel aus. Im Fall des Aufrufes einer Regel heißt die Ausführung der Prozedur, alle Prozeduren im Regelrumpf der Reihenfolge nach auszuführen.

Die obengenannte Regel ist prozedural damit wie folgt zu lesen:
Um die Prozedur `p(X)` zu lösen, rufe die (Unter-)Prozedur `q(X)` auf und dann die (Unter)Prozedur `r(X)`, bzw. mit dem Aufruf `p(a)`:
Um die Prozedur `p(a)` zu lösen, rufe `p(X)` auf mit dem Parameter `a` usw.

Die prozedurale Bedeutung ist die Sicht, die der PROLOG-Interpreter einnimmt. Sie bestimmt, wie - nach welchem Suchalgorithmus - der implizite Inferenzmechanismus arbeitet. Wir haben oben den grundlegenden Ableitungsvorgang dargestellt und PROLOG als *implizites Backward-Chaining-System* vorgestellt, ohne für beides eine Begründung gegeben zu haben. Wir holen die Begründung hier nach. Das Ableitungsverfahren und damit die prozedurale Interpretation von PROLOG beruht auf dem Resolutionsalgorithmus, einem Ableitungsprinzip, das zum mechanischen Beweisen von Theoremen benutzt werden kann.

Das *Resolutionsprinzip* (Robinson 1965) besagt folgendes: Zwei Clausen können resolviert werden, wenn zu einer Atomformel der einen Clause die gleiche Atomformel in der anderen Clause *negiert* vorkommt. Die Vereinigung der beiden alten Clausen ohne das komplementäre Paar von Atomformeln erzeugt eine neue Clause, die Resolvente.

---

[7] Manche Autoren (z.B. Schnupp 1987) interpretieren PROLOG fast nur prozedural. Diese Betrachtungsweise ist typisch für Programmierer, die aus den prozeduralen/algorithmischen Sprachen FORTRAN, PASCAL, C o.ä. kommen.

Zum Beispiel resolvieren die beiden Clausen
p und
$(q \vee \neg p)$
zur Resolvente q, was dem Modus Ponens entspricht.

Ein wichtiger Satz lautet, dass eine Clausenmenge M genau dann widerspruchsvoll ist, wenn die Resolution auf M zur leeren Clause (F4) führt (vgl. Kleine-Büning/Schmitgen 1986). Dieser Satz wird zur maschinellen Herleitung benutzt. Der PROLOG-Inferenzmechanismus führt - logisch betrachtet - einen Beweis durch Widerspruch: (F3) besagt nämlich, dass PROLOG-Anfragen als negierte Fakten zu verstehen sind. Die zu beweisende Anfrage wird als negiertes Theorem nun solange resolviert, bis die leere Clause (F4) hergeleitet ist. In diesem Fall ist gezeigt, dass die Negierung des Theorems zu einem Widerspruch führt und das Theorem damit gültig ist.

Bei prädikatenlogischen Formeln müssen Variablen beziehungsweise Terme beim Resolvieren durch Variablensubstitution so gleichgesetzt werden, dass ein Widerspruch erzeugt wird - was unter Umständen sehr aufwendig ist. Die theoretische Basis für das Resolvieren in der Prädikatenlogik ist das Theorem von Herbrandt und Gegenstand der Unifikationstheorie.

Im Rahmen dieser Arbeit ist eine weitergehende Behandlung des Resolutionsprinzips nicht möglich. Eine ausführliche und genauere Darstellung geben Kleine-Büning/Schmitgen (1986), Clocksin/Mellish (1987) und insbesondere Lloyd (1993). Ein zusammenfassender Überblick findet sich in Manhart (1988).

Die Aussage mancher Autoren, nach der PROLOG eine deklarative, nicht-prozedurale Sprache ist, ist also nicht ganz richtig (z.B. Brent 1986: 275). Vielmehr kann PROLOG auf die eine oder andere Weise interpretiert und gelesen werden. Das Besondere und Schöne an PROLOG als Modellierungsinstrument ist nun, dass man das algorithmische Denken - so man will und es die Problemstellung zulässt - vergessen kann: „PROLOG enthält den Keim einer großartigen Idee. Ein PROLOG-Programmierer legt nicht fest, wie der Computer seine Aufgaben erledigen soll, sondern er gibt vielmehr eine Beschreibung der Aufgabe als Folge von Bedingungen, die erfüllt werden müssen. Kurz gesagt, in LISP muss man das 'Wie' der Datenverarbeitung spezifizieren, während man in PROLOG nur das 'Was' anzugeben braucht - es ist dann Aufgabe der Maschine, das 'Wie' zu bestimmen. Der letztgenannte Ansatz ist deshalb von Vorteil, weil er den Programmierer davon befreit, sich über die Details der Algorithmen den Kopf zu zerbrechen, mit denen er die Aufgaben zu bewältigen trachtet. Stattdessen braucht er sich nur um eine präzise Aufgabenstellung zu kümmern“ (Harmon/King 1986: 103).

Angemerkt sei noch, dass neben dem hier vorgestellten „pure PROLOG“ jeder PROLOG-Interpreter Erweiterungen zur Verfügung stellen muss, die mit dem logischen Programmiermodell brechen. Dies sind insbesondere (vgl. GMD 1987):

- *Arithmetik:* arithmetische Prädikate schließen PROLOG an eine maschinennahe Arithmetik an, stehen aber außerhalb des logischen Programmiermodells. Beispielsweise wertet `is/2` arithmetische Ausdrücke aus:
  ```
  ?- X is 3+5
  X = 8
  ```
- *Meta-Logik:* Meta-logische Prädikate liegen *überhalb* der gegebenen PROLOG-Sprache und erweitern als Meta-Prädikate die Verwendungsmöglichkeiten von PROLOG. Ein solches Meta-Prädikat ist beispielsweise `var(X)`. Dieses Prädikat ist wahr, wenn `X` eine ungebundene Variable ist. Entsprechend ist das Gegenstück `nonvar(X)` wahr, wenn `X` gebunden ist.
- *Extra-Logik:* Extra-logische Prädikate liegen *außerhalb* des prädikatenlogischen Modells. Ein wesentlicher Typ sind die Ein-/Ausgabe-Prädikate `read/1` und `write/1`.
- *Ablaufsteuerung:* Hierzu zählt vor allem der cut `!/0`, der den Ablauf eines PROLOG-Programms ändert und Backtracking verhindert. Zur Ablaufsteuerung wird manchmal auch das `not/1` gerechnet, da es in den meisten PROLOG-Systemen nicht im logischen Sinn implementiert ist, sondern als „negation by failure“.
- *Datenbankzugriffe:* Hierzu zählen vor allem Prädikate zum Hinzufügen (`assert/2`) und Löschen (`retract/2`) von Fakten und Regeln aus der Datenbasis.

Der vorgestellte kleine Ausschnitt von PROLOG muss hier genügen und dürfte hinreichend sein. Für eine weitergehende Behandlung von PROLOG vgl. Clocksin/Mellish (1987). Eine umfassendere Einführung mit Bezug zur theoretischen Logikprogrammierung und fortgeschrittenen Programmiertechniken ist Sterling/Shapiro (1988). In diesem Buch werden auch theoretische Aspekte ausführlich behandelt, wie zum Beispiel Fragen zu Korrektheit und Vollständigkeit von PROLOG-Programmen. Die mathematischen Grundlagen der Logikprogrammierung finden sich in Lloyd (1993). Praktisch orientierte Lehrbücher sind Belli (1986), Schnupp (1986) und Schnupp/Nguyen Huu (1987).

8. Wissensbasierte Modelle

Nachdem die grundlegenden Elemente regelbasierter Systeme und prädikatenlogischer Wissensdarstellung in PROLOG referiert wurden, ist die Relation dieser beiden Formalismen zu Theorien einerseits und Computermodellen andererseits leicht herzustellen.

Ganz allgemein kann der Zusammenhang zwischen Theorien und wissensbasierten Systemen wie folgt gesehen werden: In wissensbasierten Systemen wird das

bereichsspezifische Wissen menschlicher Experten repräsentiert. Wissenschaftliche Theorien enthalten ebenfalls bereichsspezifisches Wissen und Wissenschaftler, die mit einer Theorie arbeiten oder diese gut kennen, können als Experten für diese Theorie betrachtet werden.

Theorien eines bestimmten Wissensgebietes können somit als wissensbasierte Systeme formalisiert werden. Wir sprechen in diesem Fall von *wissens- oder regelbasierten Computermodellen.*

Wissensbasierte Modelle sind mindestens in jenen Bereichen gut anwendbar, in denen konventionelle Simulationstypen nicht brauchbar sind. Sowohl verbale als auch formalisierte strukturelle Theorien sind deshalb zunächst Kandidaten für wissensbasierte Modelle. Inwieweit es Sinn macht, auch quantitative Theorien wissensbasiert zu modellieren, kann hier nicht behandelt werden.

Wenn wir die Wissensrepräsentation auf PROLOG beziehen, dann entsprechen aus der Sicht des Modellierers PROLOG-Fakten den Daten einer Theorie und bereits abgeleiteten Schlussfolgerungen aus der Theorie. Die Daten in Form von PROLOG-Fakten können zum Beispiel Relationen sein oder komplexere Strukturen wie Bäume, Graphen etc.

PROLOG-Regeln eignen sich zur Darstellung von Definitionen und Gesetzen. Eine in PROLOG repräsentierte Theorie wäre eine Menge von PROLOG-Regeln, eventuell erweitert um Fakten. PROLOG-Fragen sind Anfragen an die Theorie bezüglich der Gültigkeit von Theoriedaten und Theoremen. Die folgende Tabelle listet die wichtigsten Parallelen stichwortartig auf.

Tab. 2: Parallelen zwischen (Sozial-)Wissenschaft und Expertensystemen

(Sozial-)Wissenschaft	*Expertensysteme*
Wissenschaftler	Experte
Gesetz	PROLOG-Regel(n)
Definition	PROLOG-Regel(n)
Daten	PROLOG-Fakten
Theorie	Menge von PROLOG-Regeln und Fakten
Frage nach Gültigkeit	PROLOG-Frage
Schlußfolgerung	PROLOG-Inferenz (Resolution)

Betrachten wir die Vorzüge regelbasierter Modelle, so kann ein regelbasiertes Computermodell zunächst natürlich als ganz gewöhnliches Computermodell angesehen werden, so dass die oben genannten Eigenschaften auch auf diese Programme zutreffen. Die Nutzung von Wissensverarbeitung und Expertensystem-Architektur beseitigt jedoch weitgehend die mit den traditionellen Computermodellen verknüpften Nachteile und bringt eine Reihe von gewichtigen, zusätzlichen Vorteilen. Wir fassen diese in sieben Punkten zusammen.

1. *Symbolische, nicht-numerische Repräsentation*
   Ein erster, ganz allgemeiner Vorteil wissensbasierter Modellierung ist die *Loslösung* vom Zwang zum *Quantifizieren* und der Notwendigkeit der *Anwendung numerischer Verfahren*. In qualitativen Theorien sind nicht-numerische Wissensstrukturen enthalten, die in wissensbasierte Systeme bzw. symbolisch orientierte Programmiersprachen besser abgebildet werden können. Dadurch erschließt sich den Sozialwissenschaften ein wesentlich breiteres Anwendungsgebiet für maschinelle Modellierungsformen als mit numerischen Instrumenten.

2. *Weitgehend deklarative, nicht-algorithmische Repräsentation*
   Lindenberg (1971: 85) deutet die Turing-These so, dass, wenn wir bei der Theorienbildung in Algorithmen statt in Propositionen denken, uns wenigstens der Computer zu Hilfe kommt. Dem lässt sich in einem wissensbasierten Verfahren entgegnen: Wenn wir eine deklarative Sprache wie PROLOG verwenden, dann kommt uns der Computer zu Hilfe, *und* wir können weiter in Propositionen denken. Der wissensbasierte Ansatz erlaubt eine weitgehende Abkehr vom *prozedural-algorithmischen Denken* des Modellierers. Zwar muss zwangsläufig *jedem* Computerprogramm ein Algorithmus unterliegen, wie eben dargelegt wurde, ist es aber in PROLOG möglich, Wissen rein deklarativ zu beschreiben und die algorithmische Bearbeitung dem PROLOG-internen Inferenzmechanismus zu überlassen. Der Vorteil ist, dass deklarative Aspekte einfacher zu verstehen und zu formulieren sind und man sich um die prozeduralen Details nicht zu kümmern braucht (Bratko 1987: 64). Die obengenannte semantische Lücke oder Schere zwischen der ursprünglichen Repräsentation einer Theorie (Logik, Mengenlehre, Graphentheorie, natürliche Sprache) und ihrer programmierten Form wird dadurch eindeutig reduziert. Ohne sich um den Algorithmus zu kümmern, können Modelle in einer Art „rapid prototyping“ sogar unmittelbar in Regeln umgesetzt werden.

3. *Trennung Theorie von Steuerung*
   Durch die klare Trennung von Wissensbasis und Steuersystem werden theoretisch bedeutsame Annahmen und Regeln klar von anderen Programmteilen, die zur Steuerung dienen, abgehoben. Damit lässt sich die Forderung von Frijda (1967) nach deutlicher Abschottung theoretisch relevanter von theoretisch irrelevanten Teilen in wissensbasierten Modellen eindeutig besser erfüllen als in konventionellen Modellen.

4. *Begründung von Schlussfolgerungen*
   Wissensbasierte Modelle können aufgrund ihrer Architektur ihr eigenes Verhalten erklären und begründen. Für jede Schlussfolgerung lässt sich zeigen, durch welche Fakten und Regeln diese zustande kam. Mit wissensbasierten Modellen, die fähig sind, ihre eigenen Schlüsse zu rechtfertigen, lassen sich die hinter dem Modell steckenden Regeln und Annahmen jederzeit belegen und kritisieren. Diese Begründungen können benutzt werden zum Nachweis der

Korrektheit der Ableitungen, zur Nachvollziehbarkeit der Lösungswege oder allgemein zur besseren Einsicht in die Funktionsweise des Modells. Der Vorwurf der Undurchsichtigkeit von Computermodellen wird damit weitgehend aufgehoben.

5. *Modularität und Flexibilität*
   Während in konventionellen Modellen das Wissen in Funktionen oder Prozeduren „verpackt" ist und Wissenselemente nur umständlich hinzugefügt bzw. entfernt werden können, impliziert in wissensbasierten Systemen die klare Schnittstelle von Wissensbasis und Steuersystem weitgehende Modularität und Flexibilität: Wissensstücke lassen sich einfach hinzunehmen (zusätzliche Regeln einführen), ändern (bestehende Regeln modifizieren) oder entfernen (Regeln streichen). Dies begünstigt zweifellos den experimentellen Charakter der Modelle.

6. *Ableitung als Deduktion und Schlussrechtfertigung als Erklärung*
   In logikbasierten Regelsystemen entspricht der Begriff der Ableitung unmittelbar dem logischen Deduktionsbegriff und der Begriff der Erklärung unmittelbar dem H-O-Schema. Eine Aufforderung an das System, ein Ereignis zu erklären, korrespondiert - unter der Bedingung, dass die Regel ein Gesetz repräsentiert - direkt dem nomologischen Erklärungsschema: die Aufforderung, einen Schluß zu erklären, löst die Ausgabe der (Gesetzes-)Regel(n) aus und der Fakten, aus denen das zu erklärende Ereignis deduziert wurde.

7. *Mechanische Deduktion*
   Die zentrale Eigenschaft und Stärke wissensbasierter Modelle ist, dass die logischen Eigenschaften einer (qualitativen) Theorie maschinell untersucht und Folgerungen automatisch deduziert werden können (Glorie/Masuch/Marx 1992: 80-82). Dieser Nutzen kommt beim wissensbasierten Ansatz wesentlich prägnanter und klarer zur Geltung als bei konventionellen numerischen Modellen. In wissensbasierten Modellen kann automatisches Ableiten die Fallen des intuitiven Schließens und die Fehler der verbalen Theoriekonstruktion eliminieren. Diese Fehler stammen primär von den Grenzen der Rechenkapazität des menschlichen Gehirns (wie dem kleinen Kurzzeitgedächtnis) und von psychologischen Verzerrungen, die den Inferenzprozeß stören. Beispielsweise wird Information selektiv verarbeitet, „wünschbare" und „interessante" Konklusionen werden leichter gezogen und Ableitungen sind von den Werten des Forschers abhängig. Im Gegensatz zum intuitiven, „gehirnbasierten" Schluß-folgern können computerbasierte Deduktionen zuverlässig, reproduzierbar und wertneutral gemacht werden. Der Computer lässt sich - anders als Menschen - weder durch psychologische Faktoren noch durch eine große Menge von Daten oder Regeln (oder beidem) verwirren. In folgenden Bereichen kann der wissens-verarbeitende Computer beispielsweise besser sein als der Mensch:

- Beim Ableiten vollständiger und korrekter Konklusionen aus einer großen Datenmenge;
- Beim Entdecken versteckter Inkonsistenzen in verbalen Theorien;
- Beim Prüfen, ob bestimmte Annahmen von anderen abhängen (logische Unabhängigkeit);
- Beim Untersuchen kausal komplexer Bereiche und komplexer Interaktionen in Theorien;
- Beim Entdecken kontraintuitiver Propositionen.

Metaphorisch gesprochen kann das wissensbasierte Modell die Rolle eines idealen Forschungsassistenten spielen: „For the expert social scientist a KBS (knowledged based system, Anm.d.Verf.) may play a role analogous to that of a good research assistent, checking the logic of the researcher, tracing out the implications of changing an assumption, answering questions about its reasoning, and identifying the source of particular bits of knowledge“ (Brent 1986: 270-271).

Trotz vieler Gemeinsamkeiten zwischen Expertenwissen, Expertensystemen und Theorien gibt es allerdings auch Unterschiede. Reale Expertensysteme müssen mit viel komplexerem Wissen umgehen, als dies bei den meisten Theorien der Fall sein dürfte. Die Wissensrepräsentationsformalismen praktisch ausgerichteter Expertensysteme gehen weit über Prädikatenlogik hinaus und beinhalten zum Teil komplexe Wissensstrukturen.[8]

Moderne Expertensysteme verwenden vielfach hybride - das heißt unterschiedliche - Repräsentationsformalismen innerhalb einer Problemstellung. Bei den unten behandelten Theorien hat die Datenbasis hingegen eine sehr einfache und einheitliche Struktur. Die empirische Faktenbasis besteht aus sehr vielen, aber einfachen 2-stelligen Relationen. In diesem Fall macht es keinen Sinn, das Expertensystem so zu konstruieren, dass es mit der Abarbeitung von Regeln nach Benutzerdaten fragt oder die Anfrage nach Daten in der Erklärungskomponente rechtfertigen zu lassen.

Die Stärke von Expertensystemen wird in der Literatur oft in der maschinellen Behandlung unsicheren und vagen Wissens gesehen. Haas (1990) betont zum Beispiel gerade den Vorteil der Behandlung unsicheren Wissens für sozialwissenschaftliche Expertensysteme. Versucht man explizit vorliegende Theorien in solche Systeme zu transferieren, spielen Vagheiten, Unsicherheiten und Daumenregeln - zumindest bei den hier behandelten Modellen - keine Rolle. Für die Darstellung von Unsicherheit in sozialwissenschaftlichen Expertensystemen möchten wir auf Haas (1990) und Benfer/Brent/Furbee (1991) verweisen.

Unser Ziel ist es nicht, aus Theorien *vollständige* Expertensysteme zu machen, sondern bestimmte *Aspekte* der Expertensystem-Technologie zur Modellierung zu ver-

[8] In der KI gibt es eine Diskussion darüber, inwieweit Prädikatenlogik überhaupt zur Darstellung von Alltagsswissen geeignet ist.

wenden. Von den typischen Komponenten sind für unsere Zwecke nicht alle bedeutsam. Beispielsweise werden wir auf die Installation einer Wissenserwerbs-Komponente verzichten. Wir sind auch nicht daran interessiert, möglichst benutzerfreundliche Systeme mit schönen Oberflächen zu generieren, sondern die Modelle sollen als Werkzeuge für den Wissenschaftstheoretiker und Substanzwissenschaftler zur Theorienbildung und -analyse eingesetzt werden können.

9. Wissensbasierte Modelle in den Sozialwissenschaften

Wissensbasierte Modelle wurden in den Humanwissenschaften vor allem in der kognitiven Psychologie verwendet. Bekannte Vertreter und Initiatoren dieses Ansatzes sind Allan Newell und Herbert Simon (vgl. z.B. Newell/Simon 1972). Ziel dieser KI-orientierten Modellbildung ist es, mentale Prozesse mit wissensbasierten Systemen zu „rekonstruieren“ und auf diese Weise besser verstehbar zu machen. Den theoretischen Hintergrund bildet die Annahme, dass kognitive Prozesse wie Problemlösen, Planen oder Sprachverstehen als wissens- oder regelbasierte Vorgänge der Symbolverarbeitung aufgefasst werden können.

Wissensbasierte Systeme werden dabei oft direkt als psychologisch interpretierbare Modellvorstellung für die Architektur des menschlichen kognitiven Apparats angesehen (Opwis 1992: 80-86). Beispielsweise spiegeln Regeln den assoziativen Charakter weiter Bereiche menschlichen Wissens wieder und eignen sich - durch Hinzufügen bzw. Löschen von Regeln - gut zur Modellierung von Lern- und Vergessensvorgängen. Die strukturellen Bestandteile eines solchen Systems - Faktenbasis und Regeln - lassen sich als Strukturtheorien des Gedächtnisses interpretieren: der Faktenbasis entspricht das Kurzzeitgedächtnis, den Regeln das Langzeitgedächtnis.

Bezüglich der Speicherung und des Abrufs von Informationen können eine Vielzahl gedächtnispsychologischer Vorstellungen in diesem Ansatz rekonstruiert werden. In der kognitiven Psychologie und in anderen Teilgebieten der Psychologie haben sich deshalb wissensbasierte Systeme aufgrund ihrer hervorstechenden Eigenschaften zum bevorzugten formalen Modellierungsinstrumentarium entwickelt. Ein Überblick über diesen Ansatz mit vielen Beispielen findet sich in Opwis (1992).

In den Sozialwissenschaften ist die Motivation für die Verwendung KI-basierter Modelle eine andere. Erste sozialwissenschaftlich relevante KI-Modelle entstanden bereits in den sechziger und siebziger Jahren, wobei die Rezeption dieser Modelle aber weitgehend auf die KI-Gemeinde beschränkt blieb. Sie unterscheiden sich auch insofern von dem hier betrachteten Modelltyp, als sie sich (noch) nicht explizit auf Wissen oder Regeln beziehen. Zu diesen frühen KI-Modellen zählt das Programm von Gullahorn/Gullahorn (1963) zur Interaktionstheorie von Homans (1961). Ein anderes frühes Beispiel ist die IDEOLOGY MACHINE von Abelson (1973).

Abelson betrachtet den ideologischen Konflikt des kalten Krieges aus der Sicht des KI-Wissenschaftlers und konstruiert ein Skript, das mentale Prozesse eines rechtsradikalen Ideologen repräsentiert. Eine detaillierte, zusammenfassende Beschreibung gibt Boden (1987).

In den achtziger Jahren erkannten und diskutierten einige Sozialwissenschaftler die Relevanz *regelbasierter* Systeme für Modellierungszwecke. Vordergründigstes Ziel wissensbasierter Modellierung ist die Möglichkeit der Formalisierung und Explikation *verbaler* theoretischer Repräsentationen, die sich einer quantitativ-mathematischen Modellierung entziehen. Beispielsweise betrachtet Brent (1986) wissensbasierte Systeme als „qualitativen Formalismus" zur Repräsentation nicht quantitativ-mathematisch formalisierbarer Theorien: „To sociologists who have found traditional mathematical models to be appropriate and useful for only a limited range of social phenomena, the greatest appeal of KBS's (knowledged based systems, Anm.d.Verf.) is their claim to be applicable to problems not having tractable solutions based on mathematical reasoning" (Brent 1986: 258). Ein Überblick über Anwendungen regelbasierter Systeme in den Sozialwissenschaften findet sich in Benfer/Brent/Furbee (1991).[9] Wir wollen einige typische Modelle betrachten.

Thorson/Sylvan (1982) benutzen ein Produktionssystem um Kennedys Entscheidungen während der Kuba-Krise zu modellieren. Kennedys Entscheidungsprozesse werden als Wenn-dann-Regeln repräsentiert, und das Programm wird zunächst mit Ereignissen getestet, die tatsächlich während der Kuba-Krise auftraten. Das Modell reagierte ähnlich wie Kennedy in der historischen Situation. Die Autoren führten anschließend Experimente durch um zu sehen, wie Kennedy bei Ereignismengen reagiert hätte, die historisch nicht auftraten.

Sylvan/Glassner (1985) benutzen ein Logikprogramm, um theoretische Aussagen von Simmel (1955) zu prüfen und übersetzen 107 Passagen aus Simmels „Conflict and the Web of Group Affiliations" in 223 Regeln. Es wird geprüft, ob aus der Theorie Kontradiktionen ableitbar sind und ob es möglich ist, die zentrale Behauptung von Simmel, nämlich die Konflikt-Kohäsions-Hypothese, zu deduzieren. Es zeigt sich, dass aus den formalisierten Regeln zwar keine Kontradiktionen abgeleitet werden können, aber ebensowenig kann die zentrale Hypothese bewiesen werden. Mit dem Programm kann also belegt werden, dass die - übersetzten - verbalen Annahmen von Simmel nicht hinreichend sind, um seine zentrale Behauptung deduzieren zu können.

Banerjee (1986) versucht mit Hilfe eines PROLOG-Programms die Entstehung von sozialen Strukturen zu erklären. Sozialstruktur ist für Banerjee einfach ein Muster

---

[9] Benfer/Brent/Furbee (1991) geben ebenfalls einen Überblick über Verwendungsmöglichkeiten sozialwissenschaftlicher Expertensysteme *außerhalb* von Modell- und Theorienbildung im engeren Sinn. Hierzu gehören zum Beispiel statistische Expertensysteme zur Datenanalyse oder Systeme zur Unterstützung der Auswahl bestimmter Forschungsmethoden.

von Wiederholungen sozialer Handlungen. Zur Erklärung sozialer Strukturen benutzt Banerjee Schema-Konzepte aus der KI, in denen die Muster sozialer Handlungen gespeichert sind. Die Annahme ist, dass Akteure in sozialen Situationen passende Schemata suchen, aktivieren und dann ausführen. Diese Konzepte werden in einem PROLOG-Programm operationalisiert. Benfer/Brent/Furbee (1991: 26) weisen darauf hin, dass in diesem Programm eine Theorie entstanden ist, die vorher nicht explizit vorhanden war: „Banerjee's work is an example of the use of artificial intelligence programming strategies where the AI program becomes the theory itself".

Die Forschungsgruppe um Michael Masuch (Center for Computer Science in Organization and Management, Amsterdam) übersetzt verschiedene organisationssoziologische Theorien in Prädikatenlogik und Clausenform. Unter Nutzung eines Theorembeweisers können alle logisch möglichen Schlussfolgerungen aus den Annahmen abgeleitet werden. Glory/Masuch/Marx (1990) präzisieren auf diese Weise die Struktur von Mintzbergs Kontingenztheorie und entdecken während des Formalisierungsprozesses an unterschiedlichen Stellen terminologische Probleme.

Ein anderes Beispiel aus dieser Gruppe (Péli/Bruggeman/Masuch/Nualláin 1992) ist die prädikatenlogische Übersetzung der Organisationsökologie von Michael Hannan und John Freeman. Die Anwendung des Theorembeweisers zeigt, dass der modellierte Teil - mit einer Ausnahme - konsistent ist. Weiter werden eine Reihe neuer Theoreme entdeckt, während einige postulierte Konklusionen aus den Annahmen nicht hergeleitet werden können.

Die Beispiele betonen verschiedene Seiten und Ziele wissensbasierter Modellierung (vgl. Benfer/Brent/Furbee 1991): Thorson/Sylvan (1982) ***simulieren*** tatsächliche und fiktive historische ***Ereignisse***, Sylvan/Glassner (1985) und die Masuch-Gruppe prüfen die ***logische Konsistenz und Vollständigkeit existierender Theorien*** und in Banerjee (1986) ***entsteht mit dem Programm eine neue Theorie.*** Ein wichtiger inhaltlicher Anreiz für die regelbasierte Darstellung ist insbesondere auch die ***qualitative Modellierung von Klassikern der Soziologie***, wie sie in Sylvan/Glassner (1985) und dem folgenden Beispiel erfolgt. Diese Modelle, sowie die der Masuch-Gruppe, kommen unserem folgenden Ansatz am nächsten.

Brent (1986) will am Beispiel einer klassischen, aber hochgradig „dunklen Theorie" - Goffmans Interaktions-Modell des „dramaturgical framework" - demonstrieren, dass ein weiter Bereich umgangsssprachlich-theoretischer Propositionen in PROLOG-Clausen übersetzt werden kann. Er transferiert einen Teil des Goffman-Textes - der „bislang allen Formalisierungsversuchen widerstand" - nach PROLOG um zu demonstrieren, dass der nach PROLOG übersetzbare Bereich sozialwissenschaftlichen Wissens größer ist als der für traditionelle mathematische Modelle und weniger Strukturierung verlangt. Der folgende Auszug ist ein Beispiel,

wie eine verbale Definition in eine PROLOG-Regel übersetzt wird (Brent 1986: 264).

Tab. 3: Eine umgangssprachliche Proposition von Goffman und die entsprechende Repräsentation als PROLOG-Regel.

Umgangssprache	*PROLOG*

performance refer(s) to all the activity of an individual which occurs during a period marked by his continuous presence before a particular set of oberservers and which has some influence on the observers.	`is_a(X,performance) :-`
	`  is_a(X,activity),`
	`  occurs_during(X,Period),`
	`  marked_by(Period),`
	`  contin_pres_of(Obsvs),`
	`  influences(X,Obsvs).`

Brent überträgt die Kernsätze der umgangssprachlichen Theorie direkt in PROLOG. Dieses Vorgehen ist typisch für die meisten Modellbauer. Ein unmittelbares, lineares Satz-für-Satz-Übersetzen von der Umgangssprache in eine formale Sprache erscheint aber problematisch. Verbale Theorien - und hier ist die Goffman-Theorie ein Beispiel par excellence - sind oft in einer blumig-vagen und hochambigen Prosa gehalten, die bei der direkten Übertragung in das Programm ihre Mängel in die Formalisierung mitschleppen: „Moving from sentence to sentence, trying to find some translation for each phrase, one would end up with an uncontrollable number of concepts in the formal language. Many of these concepts would overlap, reproducing the ambiguities of the natural language“ (Glorie/Masuch/Marx 1990: 85-86).

Wir vertreten deshalb die Auffassung, dass man vor der Implementierung den logischen Aufbau der Theorie explizit machen und klären sollte. Konkret empfiehlt sich, sich zunächst einmal Gedanken zu machen über die Grundbegriffe, die verwendeten Relationen und deren Stelligkeit, sowie die postulierten Zusammenhänge, um dann zu unterscheiden zwischen Definitionen, Gesetzen, Daten und Theoremen.

In PROLOG ist alles in Fakten und Regeln zu formalisieren, so dass zum Beispiel PROLOG-intern zwischen Definitionen und Gesetzen kein Unterschied besteht. Die Aussage von Goffman in Tab. 3 ist offensichtlich eine Definition, was aber weder in der umgangssprachlichen Darstellung noch in PROLOG expliziert wird. Eine logische Rekonstruktion der Theorie würde dies explizit offen legen. Die strukturalistische Wissenschaftstheorie leistet genau die gewünschte logische Explikation empirischer Theorien, auf deren Basis die Computermodellierung erfolgen kann.[10]

[10] Im übrigen wird auch in kommerziellen und praxisnahen Expertensystemen das Wissen des Experten normalerweise nicht unmittelbar in die Maschine transferiert. Der „Wissenstransfer-Experte“ oder Knowledge-Engineer strukturiert und „rekonstruiert“ vielmehr erst das Expertenwissen, um es dann systematisch aufbereitet zu implementieren.

LITERATUR

Abelson, R.P. (1973). The Structure of a Belief System. In R.C. Schank/K.M. Colby (eds.), *Computer Models of Thought and Language*. San Francisco: Freeman.

Banerjee, S. (1986). Reproduction of Social Structure. An Artificial Intelligence Model. *Journal of Conflict Resolution*, 30, 2, S. 221-252.

Barr, A./Feigenbaum, E.A. (1981). *The Handbook of Artificial Intelligence* Vol. 1, Los Altos: Kaufman.

Belli, F. (1986). *Einführung in die logische Programmierung mit Prolog*. Mannheim: Bibliographisches Institut.

Benfer, R.A./Brent, E.E./Furbee, L. (1991). *Expert Systems*. Newbury Park CA: Sage.

Boden, M. (1987). *Artificial Intelligence and Natural Man* (2.Aufl.). New York: Basic Books.

Bratko, I. (1987). *Prolog. Programmierung für Künstliche Intelligenz*. Bonn: Addison-Wesley (zuerst 1986: Prolog Programming for Artificial Intelligence. Reading, Mass: Addison-Wesley).

Brent, E.E. (1986). Knowledged-Based Systems: A Qualitative Formalism. *Qualitative Sociology*, 9, 3, S.256-282.

Clocksin, W.F./Mellish, C.S. (1987). *Programming in Prolog* (3. Aufl.). Berlin: Springer.

Frijda, N.H. (1967), Problems of Computer Simulation. In J.M.Dutton/W.H.Starbuck (eds.), *Computer Simulation of Human Behavior*. New York: Wiley, S.610-618.

Glory, J.C./Masuch, M./Marx, M. (1990). Formalizing Organizational Theory: A Knowledge-Based Approach. In M. Masuch (ed.), *Organization, Management, and Expert Systems. Models of Automated Reasoning*. Berlin: de Gruyter, S. 79-104.

GMD Gesellschaft für Mathematik und Datenverarbeitung (1987). *Die Sprache Prolog*. GMD F2G2.

Gullahorn, J.T./Gullahorn, J.E. (1963). A Computer Model of Elementary Social Behavior. *Behavioral Science*, 8, S.354-362 (deutsch in: R.Mayntz (ed.), *Formalisierte Modelle in der Soziologie*. Neuwied/Berlin: Luchterhand, S. 233-248).

Haas, J. (1990). Treatment of Uncertainty in Social-Science Expert Systems. In H.Czap/ W.Nedobity (eds.), *Terminology and Knowledge Engineering* Vol.1. Frankfurt a.M.: Indeks, S.62-76.

Harmon, P./King, D. (1986). *Expertensysteme in der Praxis. Perspektiven, Werkzeuge, Erfahrungen*. München: Oldenbourg.

Heyer, G./Krems, J./Görz, G. (eds.) (1988). *Wissensarten und ihre Darstellung. Beiträge aus Philosophie, Psychologie, Informatik und Linguistik*. Berlin: Springer.

Homans, G.C. (1961). *Social Behaviour: Its Elementary Forms*. New York: Harcourt.

Kleine-Büning, K./Schmitgen, S. (1986). *Prolog. Grundlagen und Anwendungen*. Stuttgart: Teubner.

Kowalski, R. (1988). *Logic for Problem Solving* (9. Aufl.). New York: North Holland.

Lindenberg, S. (1971). Simulation und Theoriebildung. In H.Albert (ed.), *Sozialtheorie und soziale Praxis*. Meisenheim: Hain, S.78-113.

Lloyd, J.W. (1993). *Foundations of Logic Programming* (2. Aufl.). Berlin: Springer.

Manhart, K. (1988). Prolog und Logik. Der Beweismechanismus von Prolog und seine logischen Grundlagen. Teil 1: *mc* 2/88, S.41-43; Teil 2: *mc* 3/88, S.87-89.

Newell, A./Simon, H.A. (1972). *Human Problem Solving*. Englewood Cliffs, N.J.: Prentice-Hall.

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

Péli, G./Bruggeman, J./Masuch, M./Nualláin, B. (1992). A Logical Approach to Organizational Ecology: Formalizing the Inertia-Fragment in First-Order Logic. CCSOM Reprint 92-74 (Center for Computer Science in Organization and Managment).

Puppe, F. (1988). *Einführung in Expertensysteme*. Berlin: Springer.

Rechenberg, P. (1991). *Was ist Informatik? Eine allgemeinverständliche Einführung*. München: Hanser.

Reimer, U. (1991). *Einführung in die Wissensrepräsentation*. Stuttgart: Teubner.

Rich, E. (1988). *KI-Einführung und Anwendungen*. Hamburg: McGraw-Hill (zuerst 1983: Artificial Intelligence. New York: McGraw-Hill).

Robinson, J.A. (1965). A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the ACM*, 12, 1, S.23-41.

Savory, S.E. (1988). *Grundlagen von Expertensystemen*. München: Oldenbourg.

Schnupp, P./Nguyen Huu, C.T. (1987). *Expertensystem-Praktikum*. Berlin: Springer.

Sterling, L./Shapiro, E. (1988). *Prolog. Fortgeschrittene Programmiertechniken*. Bonn: Addison-Wesley.

Sylvan, D./Glassner, B. (1985). *A Rationalist Methodology for the Social Sciences*. New York: Blackwell.

Tanimoto, S.L. (1990). *KI: Die Grundlagen*. München: Oldenbourg.

Thorson, S.J./Sylvan, D.A. (1982). Counterfactuals and the Cuban Missile Crisis. *International Studies Quarterly*, 26, 4, S.539-571.

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