Klaus Manhart

Prolog und Logik

Der Beweismechanismus von Prolog und seine logischen Grundlagen (Teil 1)

Prolog ist die derzeit höchste Programmiersprache, die ganz allgemein für die Bearbeitung nichtnumerischer Problemstellungen eingesetzt werden kann und sich insbesondere in der Artificial Intelligence – etwa bei der Entwicklung von Expertensystemen – wachsender Beliebtheit erfreut.

Seinen Aufstieg verdankt Prolog unter anderem der Entscheidung, daß bei dem ehrgeizigen japanischen Forschungsprojekt, die nächste „intelligente" Computergeneration zu entwickeln, Prolog als Basissprache eingesetzt wird. Für die Zukunft kann die Bedeutung von Prolog also kaum überschätzt werden. Aber bereits heute werden zahlreiche Expertensysteme und Programme, die natürliche Sprache verarbeiten, in Prolog geschrieben und nicht mehr mit dem „traditionellen" Artificial-Intelligence-Instrument Lisp.

Prolog beruht auf der klassischen Prädikatenlogik und steht für Programming in Logic. In der Tat wird eine Variante dieser Logik direkt als Programmiersprache verwendet, so daß sich grundlegende Unterschiede ergeben zu prozeduralen, algorithmischen Sprachen wie Fortran oder C, aber auch zu funktionalen wie Lisp. Prolog ist, zumindest in seiner reinen Form, völlig „anti-algorithmisch". Damit ist gemeint, daß der Programmierer dem System zur Lösung eines Problems keinerlei Anweisungen im herkömmlichen Sinn mehr vorzuschreiben braucht, sondern die Struktur des Problems lediglich zu beschreiben hat. Prolog sucht sich seine Lösung „eigenständig" mit einem Deduktionsmechanismus, dessen Grundlagen und Arbeitsweisen dieser Artikel beschreibt.

Ein kleines Programm-Beispiel

Die Interaktion mit einem Prolog-Interpreter oder Compiler besteht im Grunde in nichts anderem als der Eingabe von Wissen und dessen Nutzbarmachung in Form von Fragen. Das folgende Beispiel zeigt ein – allerdings sehr kleines und primitives – Prolog-Programm (die Numerierung gehört nicht zum Programm):

```
(1)  kind_von(wolfgang_amadeus_mozart,
leopold_mozart).
(2) maennlich(leopold_mozart).
(3) vater_von(X,Y) :- maennlich(X),
kind_von(Y,X).
```

Das Programm besteht aus drei Formeln, die auch Clausen genannt werden und immer mit einem Punkt abzuschließen sind. Die Clausen (1) und (2) sind Fakten, mit denen dem System die für ein Problemgebiet relevanten Sachverhalte mitgeteilt werden und die vom Benutzer als wahr anerkannt werden. Im vorliegenden Fall sollen die beiden Fakten besagen, daß Wolfgang Amadeus Mozart das Kind von Leopold Mozart ist und daß Leopold Mozart männlich ist.

Die Underliner „_" haben keine besondere Bedeutung sondern dienen lediglich der besseren Lesbarkeit. Clause (3) ist eine Regel. Regeln drücken allgemeine, für das Problemgebiet wichtige Zusammenhänge aus, im obigen Beispiel:

Für alle X und Y gilt: Wenn X männlich ist und Y das Kind von X ist, dann ist X der Vater von Y.

Man kann erkennen, daß der Dann-Teil einer Regel – auch Kopf genannt – in Prolog zuerst hingeschrieben werden muß und von den Bedingungen – dem Rumpf der Regel – durch die Zeichenfolge :- getrennt wird. Der Kopf darf nur aus einem Element bestehen, während der Rumpf mehrere z. B. durch „und" verknüpfte Bedingungen beinhalten darf; durch „und" verknüpfte Bedingungen sind mit Kommas zu trennen. Schließlich kommen in Regeln meist noch Variable vor. Variable sind Zeichenketten, die mit Großbuchstaben beginnen und Platzhalter für konkrete Objektnamen sind. Der Grund für die Notationsweise von Regeln wird weiter unten deutlich.

Die Fakten bilden zusammen mit den Regeln die Wissensbasis eines Programms, an die Fragen gestellt werden können. Fragen beginnen in Prolog mit ?-. Die Frage

```
?- maennlich(leopold_mozart).
```

beantwortet das System mit

```
Yes
```

Die Frage

```
?- maennlich(wolfgang_amadeus_mozart).
```

wird mit

```
No
```

beantwortet.

Prolog prüft dabei, ob der durch eine Frage ausgedrückte Sachverhalt logisch mit der Wissensbasis vereinbar ist. Genauer gesagt, versucht es zu zeigen, daß ein Theorem (die Frage) logisch aus den Axiomen (Fakten und Regeln) folgt. Eine Frage wird deshalb auch als Ziel bezeichnet, das zu beweisen ist. Beispielsweise ist das zweite Beweisziel

```
maennlich(wolfgang_amadeus_mozart)
```

in der realen Welt zwar richtig, kann aber aus den vorliegenden Fakten und Regeln nicht bewiesen werden. Aufgrund seiner Eigenschaft, Theoreme aus Axiomen abzuleiten, kann man Prolog auch als automatischen Theorembeweiser bezeichnen.

Interessantere Ableitungen ergeben sich, wenn das System nicht nur mit Ja oder Nein reagiert, sondern Antworten generiert. Mit

```
?- vater_von(X,wolfgang_amadeus_mozart).
```

kann etwa gefragt werden, wer der Vater von W. A. Mozart ist. Wie unschwer zu erraten ist, antwortet das System mit

```
X = leopold_mozart
```

Die Frage

```
?- vater_von(X,Y).
```

wird mit

```
X= leopold_mozart
Y= wolfgang_amadeus_mozart
```

beantwortet. Bei den letzten beiden Fragen ist zu erkennen, daß Prolog richtige Antworten gibt, die nicht explizit in der Wissensbasis stehen.

Die Sprache der Prädikatenlogik

Die Kenntnis der logischen Grundlagen und der Lösungswege ist eine unabdingbare Voraussetzung für das ernsthafte Arbeiten mit Prolog. Die nicht immer einfachen theoretischen Ausführungen werden dabei durch Beispiele illustriert.

Die Prädikatenlogik erster Stufe, auch Quantorenlogik genannt, geht in ihren Wurzeln bis auf Aristoteles zurück und wurde in ihrer heutigen Form in der Hauptsache von Gottlob Frege und Bertrand Russel begründet. Die Beschäftigung mit Logik war bis zur Einvernahme bestimmter Logik-

konzepte durch die Artificial Intelligence ein eher esoterisches Gebiet der Mathematik und Philosophie ohne konkrete Anwendungsmöglichkeiten, abgesehen vielleicht von Informatikgrundlagenforschung. Mit dem Siegeszug der Artificial Intelligence und der Entwicklung von Logikprogrammiersprachen änderte sich dies jedoch grundlegend.

Bei der Umsetzung eines Problems in die präzise logische Sprache hat man meist eine umgangssprachliche Formulierung im Kopf. Tatsächlich besteht zwischen Logik und natürlicher Sprache eine enge Beziehung, es existieren aber auch viele Unterschiede und Schwierigkeiten, wenn man versucht, natürliche Sätze in die asketische logische Sprache zu transformieren.

Das wichtigste Grundprinzip der klassischen Logik ist das Bivalenzprinzip. Es besagt ganz einfach: Jeder Satz ist entweder wahr oder falsch. Theoretisch teilt die Logik dieses Prinzip mit (Aussage-)Sätzen der natürlichen Sprache, ist dort aber nur mit erheblichen Einschränkungen und Schwierigkeiten anwendbar. Fast alle natürlichsprachlichen Sätze sind nämlich mehr oder weniger vage. Den Satz „Europa ist groß" wird man beispielsweise kaum als wahr oder falsch schlechthin betrachten. In einem gegebenen Kontext, bei gegebenen Maßstäben, Definitionen, Erwartungen, Hörern etc. können Sätze dieser Art aber durchaus wahr oder falsch sein.

Eine der wichtigsten Anwendungsvoraussetzungen der klassischen Logik ist deshalb:
Jedes Prädikat ist hinreichend scharf.

Ein Prädikat ist die Eigenschaft, die einem Objekt zugeschrieben ist (im Beispiel: „..ist groß") bzw. die Relation, die zwischen Objekten besteht.
Die klassische Logik fordert somit, daß man immer entscheiden kann, ob ein Prädikat („...ist groß") auf ein bestimmtes Objekt („Europa") zutrifft oder nicht. Erwähnenswert hierzu ist noch, daß in neuerer Zeit Logiken entwickelt wurden, die sowohl das Bivalenzprinzip fallenlassen (drei- und mehrwertige Logiken) als auch die Forderung der Schärfe der Prädikate (Fuzzy Logiken). Für Prolog hat dies allerdings keine Bedeutung.
Eine weitere Gemeinsamkeit von Logik und natürlicher Sprache ist: beide haben ein Vokabular, eine Syntax und eine Semantik. Anders als in der natürlichen Sprache sind in der Logik aber Zeichenvorrat, Syntax und Semantik präzise definiert.

Die Prädikatenlogik ist induktiv aufgebaut

Ausgehend von den kleinsten Bestandteilen werden immer komplexere Ausdrücke definiert. Der Zeichenvorrat der Prädikatenlogik besteht dabei aus folgenden Symbolen, wobei in Hinblick auf Prolog die Notation von der Konvention abweicht:

- ein Prädikat ist eine (endliche, nichtleere) Folge von Zeichen des lateinischen Alphabets ohne Leerzeichen und mit Unterstrich _, beginnend mit einem Kleinbuchstaben (Beispiele: a, b, pred, klein_buch_stabe). Man unterscheidet 0-,1-,2-,...,n-stellige Prädikate und gibt damit die Zahl der Argumente an, die ein Prädikat verlangt. In der natürlichen Sprache ist beispielsweise das Prädikat
  ... liebt ...
  2stellig, da es zwei Argumente erfordert, das Prädikat
  ... groß
  ist einstellig.
- eine Variable ist definiert wie ein Prädikat, außer daß der erste Buchstabe der Zeichenfolge mit Großbuchstaben beginnt (Beispiele: X, Y, Variable, Pred).
- Konstanten und Funktoren sind definiert wie Prädikate.
- logische Konstanten: ¬ (nicht), & (und), ∨ (oder), => (wenn-dann), <=> (genau dann, wenn)
  - Quantorzeichen: ∀, ∃
  - Sonderzeichen: Klammern, Doppelpunkt, Komma.

Aus diesen Grundbausteinen werden nun in der Syntax Terme, Atomformeln und komplexe Formeln (molekulare Formeln) gebaut.
Terme sind induktiv definiert:

(T1) Konstanten und Variable sind Terme.
(T2) Ist $\underline{f}$ ein Funktor und sind $\underline{t}_1,...,\underline{t}_n$ Terme, so ist $\underline{f}(\underline{t}_1,...,\underline{t}_n)$ ein Term. Dabei sind $\underline{t}_1,...,\underline{t}_n$ die Argumente von $\underline{f}$.

Die unterstrichenen Symbole gehören nicht zur zu definierenden Sprache, sondern dienen hier und im folgenden immer lediglich als Platzhalter für die Ausdrücke der Sprache. Terme kann man in der Umgangssprache mit Objektnamen vergleichen. Terme der letzten Art entsprechen dabei Konstrukten wie „der Verfasser von Ulysses", „die Summe von zwei und drei", in Logikschreibweise sieht das so aus:
verfasser_von(ulysses).
summe_von(zwei,drei).
Terme bezeichnen immer Objekte, die nicht wahr oder falsch sein können; das sollte man nie vergessen.
Der nächste Schritt besteht nun darin, daß man aus Termen und Prädikaten Atomformeln bildet:

(A) Ist $\underline{p}$ ein n-stelliges Prädikat und sind $\underline{t}_1,..,\underline{t}_n$ Terme, so ist $\underline{p}(\underline{t}_1,...,\underline{t}_n)$ eine Atomformel.

Beispiele für Atomformeln sind:
maennlich(a)
liebt(X,Y)
schreibt(verfasser_von(ulysses),buch). Variablenfreie Atomformeln stehen in der Umgangssprache für einfache Sätze ohne logische Verknüpfungen wie bei „Der Verfasser von Ulysses schreibt ein Buch". Es sei angemerkt, daß in Prolog syntaktisch nicht unterschieden wird zwischen Atomformeln und Termen als Funktoren mit Argumenten; verfasser_von(ulysses) ist syntaktisch auch eine Atomformel.
Schließlich können aus Atomformeln mit den logischen Konstanten und Quantoren komplexe Formeln gebaut werden:

(F1) Ist $\underline{F}$ eine Atomformel, so ist $\underline{F}$ auch eine Formel.
(F2) Sind $\underline{F}$ und $\underline{G}$ Formeln, so sind auch die folgenden Ausdrücke Formeln (mit abnehmender Bindungsstärke der logischen Konstanten in dieser Reihenfolge):
¬$\underline{F}$ (Negation), $\underline{F}$&$\underline{G}$ (Konjunktion), $\underline{F}\vee\underline{G}$ (Disjunktion), $\underline{F}$=>$\underline{G}$ (Implikation), $\underline{F}$<=>$\underline{G}$ (Äquivalenz).
(F3) Ist $\underline{F}$ eine Formel und $\underline{X}$ eine Variable, so sind auch $\forall\underline{X}:\underline{F}$ und $\exists\underline{X}:\underline{F}$ Formeln.

$\forall\underline{X}$ ist der Allquantor, $\exists\underline{X}$ der Existenzquantor. Formeln der Gestalt (F3) heißen All- bzw. Existenzformeln.
Für die unterstrichenen Symbole sind dabei wieder zulässige Ausdrücke der Prädikatenlogik einzusetzen. Es sei angemerkt, daß sich die Aussagenlogik als der Teil der Prädikatenlogik definieren läßt, der keine Funktionskonstanten, keine Individuenkonstanten und nur 0stellige Prädikatsymbole enthält (n = 0 in Definition (A)).

Ein Beispiel

Es soll nun an einem kleinen Beispiel veranschaulicht werden, wie man komplexe Formeln aufbaut. X und Y seien Variable und somit auch Terme, maennlich, kind_von und vater_von seien Prädikate. Nach (A) sind dann maennlich(X), kind_von(Y,X), vater_von(X,Y) Atomformeln. Wendet man (F2) an, so ergibt sich die komplexe Formel maennlich(X) & kind_von(Y,X) und bei nochmaliger Anwendung von (F2)
maennlich(X) & kind_von(Y,X) =>
vater_von(X,Y).
In dieser Formel kommen alle Variablen frei vor, d. h. sie sind an keinen Quantor gebunden. Wendet man mit (F3) den All-

quantor an, so erhält man eine geschlossene Formel, in der alle Variablen durch die Quantoren gebunden sind:

∀X:∀Y: maennlich(X) & kind_von(Y,X)
=> vater_von(X,Y).

Allgemein nennt man eine Variable gebunden, wenn sie im Wirkungsbereich eines Quantors liegt. Der Wirkungsbereich eines Quantors erstreckt sich dabei auf jene Formeln, deren Präfix er ist. Nur geschlossene Formeln ohne freie Variablen können wahr oder falsch sein. Die obige Formel kann natürlich mit weiteren Formeln zu noch komplexeren verknüpft werden.
Nimmt man zu der Allformel die beiden eingangs aufgeführten Fakten hinzu, so läßt sich z. B. logisch ableiten:

vater_von(leopold_mozart,
wolfgang_amadeus_mozart).

Wie lassen sich nun solche Schlüsse rechtfertigen?
Prinzipiell sind hierzu zwei Vorgehensweisen möglich:

- Entweder, man stellt einen sog. Kalkül mit einer Menge von Regeln auf und leitet aus den vorliegenden Formeln das zu beweisende Theorem mit den Regeln ab. Ein Beispiel für eine einfache Regel, die in den meisten Logikkalkülen Verwendung findet, ist der Modus Ponens, der besagt: aus F und F=>G darf G abgeleitet werden.
- Eine andere Möglichkeit besteht darin, einen Beweis semantisch zu führen, d. h. zu zeigen, daß, wenn die vorliegenden Formeln wahr sind, auch der Schluß zwingend wahr sein muß.

Bei der zweiten Möglichkeiten, auf die wir uns hier beschränken, muß geklärt werden, was es heißt: „eine Formel ist wahr". Hierzu müssen den Grundausdrücken Bedeutungen zugeordnet werden. Bislang sind die eingeführten Ausdrücke ja Zeichenketten ohne Bedeutung. Weder die zulässigen Terme wolfgang_amadeus_mozart noch xcfsfaf haben innerhalb der Prädikatenlogik eine Bedeutung.
Daß wolfgang_amadeus_mozart für uns mit einem bestimmten Menschen verbunden ist, liegt nur daran, daß diese Zeichenfolge in der Umgangssprache auf einen bekannten Komponisten verweist. Wir haben bislang immer stillschweigend angenommen, daß Prädikate und Konstanten auf ihre umgangssprachlichen Äquivalente verweisen, ohne eigentlich eine Rechtfertigung dafür zu haben. In der Semantik werden die Ausdrücke der Sprache über die Begriffe Interpretation und Modell auf bestimmte mengentheoretische Strukturen abgebildet.

Was eine Interpretation ist

Eine Interpretation ordnet Individuenkonstanten (z. B. sokrates oder xcfsfaf) Individuen der realen Welt zu (z. B. der Konstanten wolfgang_amadeus_mozart den konkreten Menschen W. A. Mozart), Prädikaten (geordnete) Mengen von Individuen (z. B. dem Prädikat ist_menschlich die Menge aller Menschen, dem 2stelligen Prädikat toetet eine Menge von geordneten Paaren aus Mördern und Ermordeten etc.).
Formal besteht eine Interpretation aus einem (nicht-leeren) Definitionsbereich D, über den die Variablen laufen und einer Funktion I, die

(1) den Individuenkonstanten Elemente aus D zuordnet:
$\underline{a} :\rightarrow I(\underline{a}), I(\underline{a}) \in D$
für alle Konstanten a;
(2) n-stelligen Funktoren n-Tupel von Elementen aus D zuordnet:
$\underline{f} :\rightarrow I(\underline{f}), I(\underline{f}) \subseteq D^n$ für alle Funktoren f
$\subseteq$ bezeichne die Teilmengenrelation, D^n das n-fache Cartesische Produkt von D);
(3) n-stelligen Prädikaten n-Tupel von Elementen aus D zuordnet:
$\underline{p} :\rightarrow I(\underline{p}), I(\underline{p}) \subseteq D^n$ für alle Prädikatsymbole p.
(4) Atomformeln Wahrheitswerte zuordnet:
$\underline{F} :\rightarrow I(\underline{F}), I(\underline{F}) \in$ {wahr,falsch} für alle Atomformeln F.

Mit Hilfe dieser Festlegungen kann nun der Wahrheitsbegriff für Atomformeln definiert werden; Atomformeln sind die kleinsten Einheiten, denen Wahrheitswerte zukommen:
Unter einer Interpretation I ist eine Atomformel $\underline{p}(\underline{a}_1,...,\underline{a}_n)$ genau dann wahr, wenn gilt:
$<I(\underline{a}_1),...,I(\underline{a}_n)> \in I(\underline{p})$.
Beispiel: toetet(kain,abel) ist wahr unter folgender Interpretation:
(i1) I(kain) = Kain
Dabei ist kain eine Zeichenfolge der Sprache, die vor der Interpretation ohne Bedeutung ist und Kain der konkrete Mensch.
(i2) I(abel) = Abel
(i3) I(toetet) = {...<Kain,Abel>, ...}
Dem Ausdruck toetet ordnet die Interpretation eine Menge von geordneten Paaren zu, wobei das erste Element der Mörder, das zweite der Ermordete ist. Diese Menge enthält u.a. auch das Paar <Kain,Abel>, nicht aber <Abel,Kain>.
Dann ist toetet(kain,abel) wahr, da <Kain, Abel> in der Menge I(toetet) enthalten ist.
Die Wahrheitswerte komplexer Formeln werden aus den Basisausdrücken berechnet und sind wie folgt festgelegt
(„gdw" ist die Abkürzung für „genau dann wenn"):
$\underline{F}\&\underline{G}$ ist wahr unter I gdw $I(\underline{F}) = I(\underline{G})$ = wahr.
$\underline{F} \vee \underline{G}$ ist wahr unter I gdw $I(\underline{F})$ = wahr oder $I(\underline{G})$=wahr.
$\neg\underline{F}$ ist wahr unter I gdw $I(\underline{F})$ = falsch.
$\underline{F} => \underline{G}$ ist falsch unter I gdw $I(\underline{F})$ = falsch.
$\underline{F} <=> \underline{G}$ ist wahr unter I gdw $I(\underline{F}) = I(\underline{G})$.
$\exists X:\underline{F}$ ist wahr unter I gdw es existiert ein $d \in D$ so daß bei $X/d\underline{F}$ wahr ist (X/d steht für: Ersetze alle Vorkommnisse von X durch d).
$\forall X:\underline{F}$ ist wahr unter I gdw für alle $d \in D$ gilt: F ist wahr bei X/d.
Fügt man zu den obigen Interpretationen noch hinzu:
(i4) I(bruder_von) =
{... <Kain,Abel>,
<Abel,Kain>,...},
so ist
toetet(kain,abel) & bruder_von(kain,abel)
wahr. Diese Formel wäre aber nicht wahr unter der Interpretation:
I(kain) = Abel
I(abel) = Kain;
Man nennt alle Interpretationen, die eine gegebene geschlossene Formel wahr machen, ein Modell dieser Formel. Beispielsweise ist die Interpretation (i1)-(i4) ein Modell für die obige konjunktiv verknüpfte Formel. Hat eine Formel mindestens ein Modell, d. h. gibt es mindestens eine Interpretation, die die Formel wahr macht, so ist sie erfüllbar.
Formeln, die nicht erfüllbar sind, sind widerspruchsvoll, z. B. ist
∃X: maennlich(X) & ¬ maennlich(X),
(die Formel behauptet, es gibt jemanden, der männnlich und nicht männlich ist) unter keiner Interpretation erfüllbar und somit widerspruchsvoll.
Schließlich läßt sich der logisch-sematische Folgerungsbegriff wie folgt festlegen: eine Formel F folgt aus einer Formelmenge M, genau dann, wenn unter allen Interpretationen, die M wahr machen, auch F immer wahr ist.
Wir haben somit die für unsere Belange wichtigsten Konzepte der Prädikatenlogik in einer freilich sehr lückenhaften und für einen Logiker recht oberflächlichen Weise vorgestellt, die uns in diesem Rahmen aber genügen sollen. Die Prädikatenlogik in der bislang aufgebauten üblichen Syntax eignet sich u. a. wegen Ineffizienz schlecht für Computerübertragung; alle Versuche, Logik als Computersprache zu benutzen mündeten in eine spezielle syntaktische Variante der Prädikatenlogik. Der zweite und letzte Teil dieses Beitrags im nächsten Heft beginnt deshalb mit der Clausenlogik.

Klaus Manhart

Prolog und Logik

Der Beweismechanismus von Prolog und seine logischen Grundlagen (Teil 2)

Prolog ist die derzeit höchste Programmiersprache, die ganz allgemein für die Bearbeitung nicht-numerischer Problemstellungen eingesetzt werden kann. Im zweiten und letzten Teil werden Clausenlogik, Hornclauseln, Syntax und die Resolutionslogik behandelt.

In der Clausenlogik werden die prädikatenlogischen Formeln in eine sogenannte Normalform umgewandelt. Hier gibt es nur mehr negierte und nicht-negierte Atomformeln, die ausschließlich mit „und" und „oder" verknüpft sind. Jede prädikatenlogische Standardformel läßt sich in einem algorithmischen Prozeß in Clausenlogik überführen.

Als Beispiel die Formel
(F0) ∀X: ∀Y: p(X,Y) & q(Y) =>
∃Z: r(Z) & p(X,Z)

Sie kann als prädikatenlogische Repräsentation der folgenden Regel aufgefaßt werden: Für alle X,Y gilt: Wenn X Kind von Y ist und Y männlich ist, dann gibt es ein Z, so daß Z weiblich und X Kind von Z ist, oder etwas natürlicher formuliert:
Zu jedem Kind eines Vaters gibt es eine Mutter (statt der „sprechenden" Prädikate kind_von etc. verwenden wir hier kurze Ausdrücke).
Die Umwandlung prädikatenlogischer Formeln in Formeln der Clausenlogik geschieht mit folgenden Schritten:

– Ersetze Implikationen und Äquivalenzen unter Benutzung der Gesetze:
F <=> G ist äquivalent zu
F => G & G => F;
F => G ist äquivalent zu ¬F ∨ G.

(Die Beweise für diese sowie die in den folgenden Schritten angegebenen Äquivalenzen sind trivial.)
Aus der obigen Formel (F0) wird somit
(F1) ∀X: ∀Y: ¬(p(X,Y) & q(Y)) ∨
∃Z: r(Z) & p(X,Z).

– Schiebe Negationssymbole möglichst weit nach innen unter Nutzung der Gesetze:
¬(¬F) äquivalent zu F;
¬(F & G) äquivalent zu ¬F ∨ ¬G;
¬∀X: p(X) äquivalent zu ∃X: ¬p(X);
¬∃X: p(X) äquivalent zu ∀X: ¬p(X).
Somit ergibt sich (F1) zu
(F2) ∀X: ∀Y: (¬p(X,Y) ∨ ¬q(Y)) ∨
∃Z: r(Z) & p(X,Z).

– Eliminiere Existenzquantoren durch Skolemisierung:
Existenzquantoren sind für automatische Theorembeweiser hinderlich. Nach dem Logiker Skolem lassen sich „erfüllbarkeitsäquivalente" existenzquantorfreie Formeln herstellen durch Ersetzung der existenzquantifizierten Variablen mit einer neuen Konstanten, oder, falls die Variable im Bereich eines Allquantors liegt, durch Ersetzung mit (Skolem-)Funktionen. Die Argumente von Skolemfunktionen bilden dabei sämtliche allquantifizierten Variablen, in deren Bereich die existenzquantifizierten liegen.
Beispielsweise müßte die Formel
∀X: ∃Y: p(X,Y) transferiert werden zu:

∀X: p(X,f(X)).
Aus (F2) wird somit
(F3) ∀X: ∀Y: (¬p(X,Y) ∨ ¬q(Y)) ∨
(r(g(X,Y)) & p(X,g(X,Y))).

– Erzeuge die Pränexform:
Da nun in der Formel nur mehr Allquantoren vorkommen, können diese gedanklich nach vorne geschoben und weggelassen werden. Alle nun noch vorkommenden Variablen bleiben aber (implizit) allquantifiziert:

(F4) (¬p(X,Y) ∨ ¬q(Y)) ∨ (r(g(X,Y)) &
p(X,g(X,Y)).

– Konvertiere die Formel in eine Konjunktion von Disjunktionen (konjunktive Normalform) mit Hilfe der Distributivgesetze:

(F & G) ∨ H ist äquivalent zu (F ∨ H) &
(G ∨ H);
F ∨ (G & H) ist äquivalent zu (F ∨ G) &
(F ∨ H).
(F5) ((¬p(X,Y) ∨ ¬q(Y)) ∨ r(g(X,Y))) &
((¬p(X,Y) ∨ ¬q(Y)) ∨ p(X,g(X,Y)).

– Erzeuge die endgültige Clausenform:
Es liegen jetzt nur mehr Formeln vor, deren Hauptoperator das logische „und" ist. Man bezeichnet jedes einzelne Konjunktionsglied als eine Clause. Innerhalb der Clausen erscheinen nur mehr disjunktiv verknüpfte Formeln. Da bei „v" als einzigem Verknüpfungsoperator Klammerung und Reihenfolge innerhalb einer Clause irrelevant sind, werden Klammern weggelassen und die Formeln in einer Clause können zu einer (implizit disjunktiv verknüpften) Menge zusammengefaßt werden. Alle Clausen zusammen bilden schließlich eine (implizit konjunktiv verknüpfte) Menge von Clausen ((F6) siehe *Bild 1*).
Die Elemente einer Clause heißen Literale, negierte Elemente sind negative, unnegierte positive Literale. Eine Clause ist also eine Menge von positiven oder negativen Literalen.
Prolog liegt eine spezielle Variante von Clausen zugrunde, sog. Horn-Clausen (nach dem Logiker Horn). Hornclausen sind Clausen mit höchstens einem positiven Literal. Beispielsweise sind die beiden obigen Clausen Hornclausen, nicht aber {¬p(X), q(X), s(X)}. Nach einem Theorem von Horn läßt sich jede prädikatenlogische Formel als Hornclause darstellen; diese bilden die grundlegende syntaktische Struktur von Prolog.

Zusammenhang zwischen Hornclausen und Prolog-Syntax

Ein Prolog-Programm ist eine endliche Menge von Hornclausen, die in Prolog-Syntax geschrieben werden; anstelle der Mengennotation tritt die Satznotation:
Besteht eine Clause aus genau einem positiven Literal, so wird nur diese geschrieben und mit einem Punkt abgeschlossen. Clausen mit genau einem positiven Literal sind Fakten.
Besteht eine Clause aus mehreren negativen Literalen und einem positiven Literal, so steht das positive Literal immer ganz links und wird von den negativen durch :- getrennt, die negativen Literale werden (ohne Negationszeichen) durch Kommata getrennt und mit einem Punkt abgeschlossen. Clausen mit einem positiven und mindestens einem negativen Literal sind Regeln.

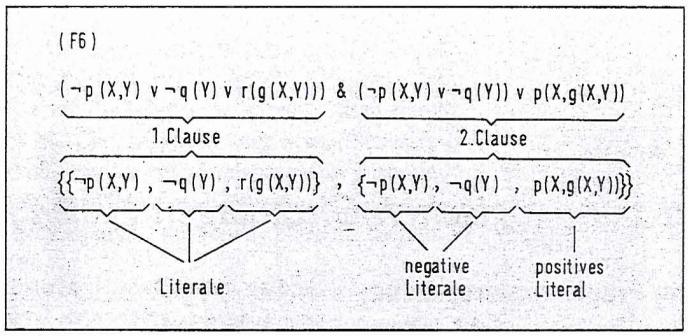


Bild 1. Eine implizit konjunktiv verknüpfte Menge von Clausen

Bild 3. Der Resolvierungsprozeß beim Modus Ponens

1)	Standard-Logik:	p
	Clausenform:	{p}
	PROLOG-Notation:	p.
2)	Standard-Logik:	p ⇒ q
		(äquivalent zu: q v ¬p)
	Clausenform:	{q,¬p}
	PROLOG-Notation:	q :- p.
3)	Standard-Logik:	∀X: p(X) & r(X) ⇒ q(X)
		(äquiv. zu: ∀X: (¬p(X) v q(X) v ¬r(X)))
	Clausenform:	{¬p(X), q(X), ¬r(X)}
	PROLOG-Notation:	q(X) :- p(X), r(X).
4)	Standard-Logik:	¬∃X: p(X)
		(äquiv. zu: ∀X: ¬p(X))
	Clausenform:	{¬p(X)}
	PROLOG-Notation:	?- p(X)

Bild 2. Gegenüberstellung von Standard-Logik, Clausenform und Prolog-Notation an einigen Beispielen

Als letzte Möglichkeit verbleiben schließlich noch Clausen mit ausschließlich negativen Literalen. Clausen mit negativen Literalen werden mit ?- beginnend und durch Kommata getrennt geschrieben. Clausen mit ausschließlich negativen Literalen sind – aus Gründen, die unten dargelegt werden – Fragen. Normalerweise ist dies eine einelementige Menge. Clausen mit mehreren negativen Literalen sind – wie man sich leicht überlegen kann – konjunktiv verknüpfte Fragen. Einige Beispiele zeigt *Bild 2.*

Aus (2) und (3) des Bildes 2 ist ersichtlich, daß die Prolog-(Satz-)Notation eine Horn-Clause in eine umgekehrt geschriebene Implikation umwandelt, so daß die Prolog-Notation immer als umgekehrter Wenn-Dann-Satz gelesen werden kann. Beispiel (4) zeigt, daß eine Prolog-Frage einem negierten Literal entspricht bzw. in der Standardlogik einer negierten Existenzformel (es gibt kein X, so daß X die Eigenschaft p hat). Der Grund hierfür wird sogleich deutlich.

Es sei darauf hingewiesen, daß man bei der praktischen Arbeit mit Prolog meist nicht so vorgeht, eine Wissensbasis prädikatenlogisch zu erstellen und sie in dem beschriebenen Prozeß in Prolog-Hornclausen zu überführen. Die Wissensbasis läßt sich meist direkt in Form von Prolog-Sätzen schreiben. Oft lassen sich durch die Einführung spezifischer „ad-hoc-Regeln" auch Skolemfunktionen vermeiden.

Der Beweismechanismus von Prolog

Nun zum zentralen Kern von Prolog und dem eigentlichen Ziel dieses Aufsatzes, der Vorstellung der Resolutionslogik. Der Beweismechanismus von Prolog und der meisten heutigen Deduktionsprogramme beruht auf dem Resolutionsprinzip von Robinson (1965):

Ist eine Interpretation I ein Modell für eine Clausenmenge C = {A,B}, so daß es ein Literal L gibt mit L∈A und L∈B, dann ist I auch Modell für C' = {(A\{L}) ∪ (B\{¬L})}. („\" ist das Operationszeichen für Differenzmengenbildung.)

Ist z. B. C = {{p},{q,¬p}} wahr, dann sind die Voraussetzungen des obigen Satzes erfüllt, da p∈{p} und ¬p∈{q,¬p}, und somit ist auch C' = {{p}\{p} ∪ {q,¬p}\{¬p}} = {q} wahr.

Der Schluß

{p}
{q,¬p}
{q}

ist natürlich nichts anderes als der klassische Modus Ponens, da {q,¬p} aussagenlogisch p => q ist.

Man nennt q die Resolvente und stellt den Resolvierungsprozeß zweckmäßigerweise dar, wie es *Bild 3* zeigt.

Die Ableitungsmethode in der Resolutionslogik beruht auf einem Widerspruchsbeweis: Man negiert die zu beweisende Formel – also das Theorem – und zeigt, daß es kein Modell gibt, das alle Formeln erfüllt. Der Hintergrund ist folgender:

Die Axiome sind als wahr vorausgesetzt. Folgt das Theorem aus den Axiomen, so müßte die Negation des Theorems unverträglich mit den Axiomen sein. Man sucht also einen logischen Konflikt zwischen dem negierten Theorem und den Axiomen. Semantisch ausgedrückt: Gibt es kein Modell, das alle Formeln erfüllt, so muß die negierte Formel falsch sein, die unnegierte mithin wahr. Konkret bedeutet dies nun, eine Clause herzuleiten, die nie erfüllbar ist und dies ist die leere Clause [ ]: eine Clausenmenge M ist nämlich widerspruchsvoll genau dann, wenn aus M die leere Clause ableitbar ist.

Um eine Formel s zu beweisen, ist diese also zu negieren, in die übrige Clausenmenge einzufügen und die leere Clause herzuleiten. Aus diesem Grund ist eine Prolog-Frage implizit eine negierte Clause, die mit den anderen Clausen aus der Datenbasis resolviert wird.

Prolog bedient sich beim Resolvieren der linearen Input-Resolution, nach der das Beweisziel – also das negierte Literal – resolviert wird mit einer Clause aus der Wissensbasis, die Resolvente dieses Prozesses wieder resolviert wird mit einer Clause aus der Wissensbasis usw. Die Auswahl der Resolutionskandidaten in der Datenbasis erfolgt dabei nach den Strategien:

– der Rumpf von Regeln wird von links nach rechts bearbeitet;
– Clausen werden in der Reihenfolge von oben nach unten durchsucht.

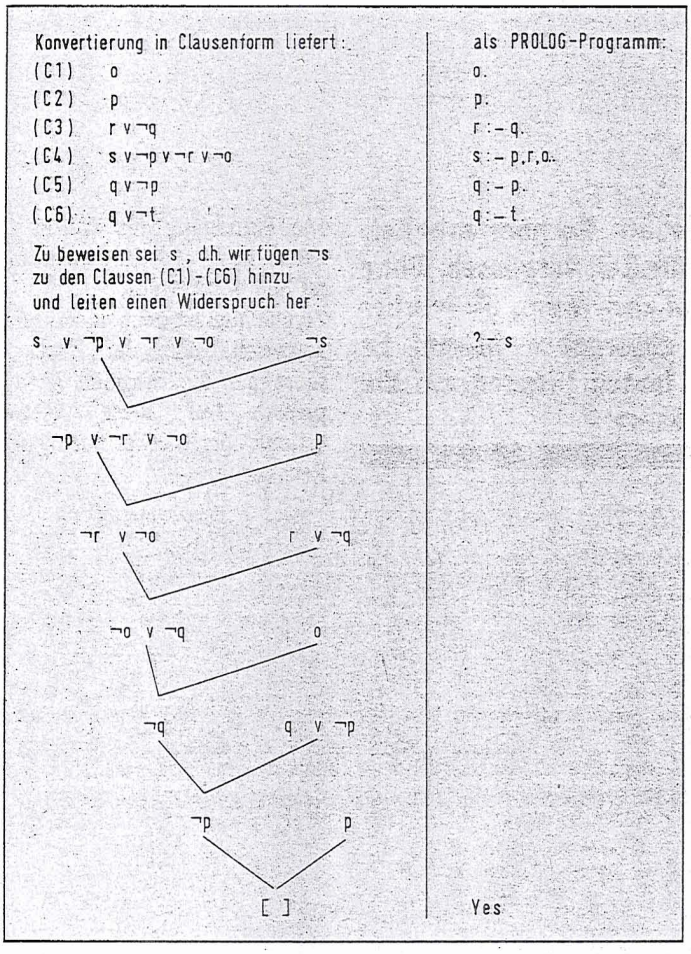
Angenommen, es seien folgende aussagenlogische Formeln gegeben:

(A1) o
(A2) p
(A3) q => r
(A4) p & r & o => s
(A4) p v t => q

Bild 4 zeigt die Konvertierung in Clausenform, das zugehörige Prolog-Programm und den Resolutionsbeweis für s. Aus diesem Beispiel ist unmittelbar nachprüfbar, daß die Anordnung der Clausen die Effizienz einer Lösung beeinflussen kann.

Bei prädikatenlogischen Formeln kommen nun zusätzlich Variable vor, die je nach Bindung einen Widerspruch erzeugen können oder nicht. Beispielsweise läßt sich bei den Clausen

praed(X).
praed(Y).



◁ **Bild 4. Ein Resolutionsbeweis in der Aussagenlogik**

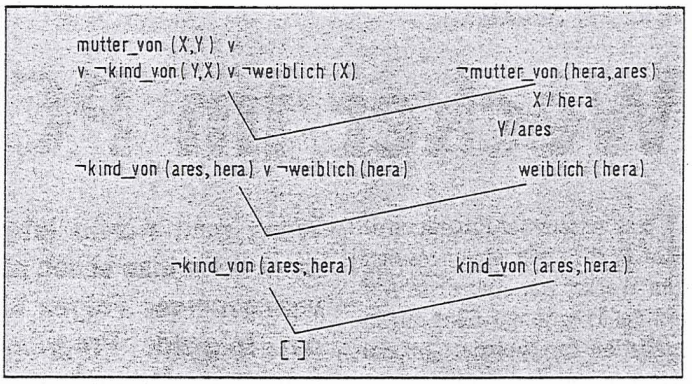


Bild 5. Prädikatenlogischer Resolutionsbeweis bei Ja/Nein-Fragen

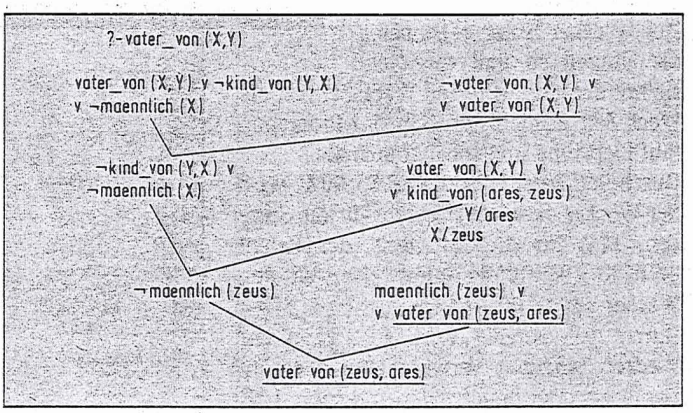


Bild 6. Prädikatenlogischer Resolutionsbeweis bei Ergänzungsfragen

für die Belegung X=Y=a die leere Clause herleiten, nicht jedoch für X=a, Y=b.

Die theoretische Basis für das Resolvieren in der Prädikatenlogik ist das Theorem von Herbrand: es besagt, daß bei der Herleitung der leeren Clause die Interpretation über einen eingeschränkten Bereich, das Herbrand-Universum, ausreichend ist. Hier kommt nun der Unifikationsalgorithmus mit ins Spiel. Der Unifikationsalgorithmus ist ein Matching-Prozess, der 2 Literale vergleicht und prüft, ob es eine Variablensubstitution gibt, bei der die Formeln identisch werden. In beiden Literalen müssen natürlich das Prädikatsymbol und die Zahl der Argumentstellen übereinstimmen. Die Substitutionsregeln sind einfach: Variable können andere Variable, Konstante oder Funktionen ersetzen, sofern die Funktion nicht eben diese Variable als Argument besitzt (ansonsten: Gefahr von Endlossubstitutionen!). Ohne auf weitere Details einzugehen, sei gleich das Familienrelationenbeispiel präsentiert; diesmal soll zur Abwechslung die griechische Mythologie herhalten:

```
kind_von(ares,zeus).
kind_von(ares,hera).
maennlich(zeus).
maennlich(ares).
weiblich(hera).
mutter_von(X,Y) :-
kind_von(Y,X), weiblich(X).
vater_von(X,Y) :-
kind_von(Y,X), maennlich(X).
?- mutter_von(hera,ares).
```

Bild 5 zeigt den prädikatenlogischen Resolutionsbeweis (X/hera bedeutet, daß X mit hera unifiziert wird). Bei Ergänzungsfragen funktioniert der Beweisalgorithmus mit einer kleinen Modifikation: man fügt die ursprüngliche Frage ihrer Negation hinzu und markiert sie, so daß sie vom Resolutionsalgorithmus nicht verwendet wird. Ihre Argumente werden aber durch die stattfindenden Resolutionen gebunden. Das Ende des Verfahrens ist dann erreicht, wenn nur noch genau diese markierte Clause übrigbleibt; die Argumentbindungen stellen dann das Ergebnis dar: *Bild 6* zeigt einen prädikatenlogischen Resolutionsbeweis bei Ergänzungsfragen.

Abschließend sei noch darauf hingewiesen, daß der Resolutionskalkül die beiden wichtigsten Eigenschaften für Logikkalküle erfüllt: zum einen ist die Resolution korrekt, d. h. es werden keine falschen Aussagen hergeleitet, zum anderen ist sie vollständig bezüglich der Widerlegung, d. h. sind Clausen inkonsistent, so ist es immer möglich, die leere Clause abzuleiten.

Bei einem in eine Programmiersprache umgesetzten Resolutionskalkül wie Prolog ist die Vollständigkeit aber durch die informatische Interpretation nicht immer gegeben, insbesondere dann nicht, wenn ein Beweiskandidat nicht terminiert. Dies ist z. B. bei dem folgenden abschließenden Programm der Fall, bei dem durch die Auswahlstrategie für geeignete Resolventen eine Endlosschleife entsteht:

```
p(X) :- q(X).
q(X) :- p(X).
q(a).
?- p(X)
```

Um p(X) zu beweisen, ist nämlich q(X) zu beweisen. q(X) könnte zwar mit X=a bewiesen werden, steht aber hinter der Regel q(X) :- p(X), so daß diese zuerst aufgerufen wird. Um q(X) zu beweisen ist aber wiederum p(X) zu beweisen usw. Das Programm erreicht aufgrund der Anordnung der Clausen also nie den Kandidaten q(a) und gerät in eine Endlosschleife. Das Problem läßt sich zwar leicht beheben, indem man q(a) an erster oder zweiter Stelle setzt, zeigt aber auch, daß theoretisch herleitbare Lösungen in bestimmten Fällen nicht ermittelt werden können. Beim praktischen Programmieren können solche Probleme i.a. vermieden werden, indem Fakten grundsätzlich vor Regeln aufgeführt werden.

Die Literaturliste befindet sich aus Platzgründen auf S. 3.