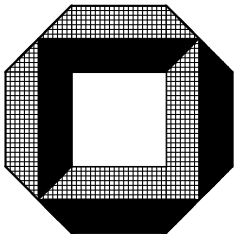
Proseminar

Künstliche Intelligenz



Universität Karlsruhe (TH)
Fakultät für Informatik
Institut für Algorithmen und Kognitive Systeme

Prof. Dr. J. Calmet
Dipl.-Inform. A. Daemi

Wintersemester 2003/2004

Institut für Algorithmen und Kognitive Systeme
Fakultät für Informatik
Universität Karlsruhe
Am Fasanengarten 5
76 128 Karlsruhe

Künstliche Intelligenz: Aussagenlogik, Prädikatenlogik und Inferenzmechanismen

Veronika Khaeva
Matthias Thoma

Inhaltsverzeichnis

Kapitel 2

Aussagen und Prädikatenlogik

2.1 Einleitung

Diese Ausarbeitung des Themas "Aussagenlogik, Prädikatenlogik und Inferenzmechanismen" wurde im Rahmen des Proseminars Künstliche Intelligenz im WS2003/2004 unter der Betreuung von Anusch Daemi erarbeitet. Es behandelt die fundamentalen Grundlagen sowohl der Aussagenlogik als auch der Prädikatenlogik. Zu beiden Themenkomplexen werden verschiedene Inferenzmechanismen vorgestellt.

2.2 Definition Aussagenlogik

Die Aussagenlogik (propositional logic, sentential logic) oder auch klassische Junktorenlogik ist die Grundlage jeder "logischen" Beschäftigung mit sprachlichen Konstrukten. Aussagen und ihre Relationen zueinander sind Gegenstände der Aussagenlogik. Die Sprache der Aussagenlogik besteht aus **Aussagenvariablen** und **Junktoren**.

Die Aussagenlogik basiert auf dem Konzept der **Booleschen Algebra** und dient als Basis für **Prädikatenlogik**. Aussagen (auch positive **Literale** oder **atomare Formeln** genannt) werden durch Negation zu **negativen Literalen**. Aussagen kombinieren sich durch die Booleschen Funktionen **Negation**, **Konjunktion**, **Disjunktion**, **Implikation** und **Äquivalenz** zu Formeln. Da eine Formel mehrere Literale besitzen und jedes dieser Literale wiederum wahr oder falsch sein kann, gibt es unter Umständen eine beträchtliche Anzahl an möglichen Kombinationen, um den Wahrheitswert der Formel zu bestimmen. Diese Kombinationen werden auch **Belegungen** genannt und übersichtlich in **Wahrheitstabellen** dargestellt. Die Aussagenvariablen abstrahieren von der syntaktischen Form der Aussagen; Interessante Fragestellungen sind dabei, ob eine gegebene Formel **tautologisch**, also wahr unter jeder möglichen Belegung

der Aussagevariablen, oder **unerfüllbar**, also falsch unter jeder möglichen Belegung ist. In Kalkülen für die Aussagenlogik werden mit Hilfe von **Inferenzregeln** Formeln aus gegebenen Formeln hergeleitet. Aus A und A -> B kann z.B. mit der Regel **Modus ponens** auf B geschlossen werden. Die Aussagenlogik ist entscheidbar; es existieren Algorithmen, die feststellen, ob eine beliebige gegebene Formel tautologisch oder unerfüllbar ist.

In der Aussagenlogik gibt es zahlreiche Äquivalenzen, die es ermöglichen Formeln aus der einen in eine andere schriftliche Repräsentation umzuwandeln, ohne dass dabei der Wahrheitswert verfälscht wird. Hierzu gehören die **Idempotenz**, **Kommutativität**, **Assoziativität**, **Absorption**, **Distributivität**, **Doppelnegation** sowie die Regeln der **Tautologie**, **Unerfüllbarkeit** und die von **deMorgan**. Mit Hilfe der oben genannten Booleschen Funktionen sowie den Äquivalenzregeln können diverse Normalformen gebildet werden zu denen die Konjunktive (KNF) und Disjunktive Normalform (DNF) gehören.

2.3 Inferenzmechanismen der Aussagenlogik

2.3.1 Modus ponens

Mit Ausdrücken der Aussagenlogik können wir interessante Dinge anstellen. Wir brauchen nicht zu wissen, ob ihre Teilaussagen wahr oder falsch sind. Setzen wir von beliebigen Ausdrücken voraus, dass sie wahr sind, können wir aus ihnen neue ableiten, die mit Sicherheit wahr sind. Um zu neuen Ausdrücken zu gelangen, die sich logisch aus den anfänglichen Ausdrücken ergeben, verwenden wir Regeln. In formalen Systemen und Kalkülen werden aus gegebenen Sätzen S1 mit Hilfe formaler, meist rein syntaktischer Regeln neue Sätze S2 abgeleitet. Diese Regeln werden als **Inferenzregeln** bezeichnet. Sie definieren eine logische Beziehung (oder Relation) zwischen S1 und S2, die meist als S1⊣S2 notiert wird. Beispiele für Inferenzregeln sind der **Modus ponens**, der **Modus tollens** und die **Resolutionsregel**.

Inferenzregeln geben an, wie aus Formeln in der Wissensbasis neue Formeln erzeugt – abgeleitet –werden können. Korrekte Inferenzregeln erzeugen nur solche Formeln, die aus der Wissensbasis logisch folgen. Inferenzregeln werden durch Schemata beschrieben.

Es seien $F_1, ...F_n$ und G Formeln und R $= \frac{F_1,...F_n}{G}$ stellt eine Inferezregel dar, die F_i werden als **Prämissen**, G als **Konklusion** der Regel bezeichnet. Eine alternative Schreibweise ist $R = F_1, ...F_n \therefore G$.

nferenzregeln sind keine Formeln der Aussagenlogik. Sie spezifizieren Muster in Formelmengen und sind als Ableitungs- bzw. Schlussfiguren aufzufassen. Wir sagen, dass das Muster $F_1, ...F_n$ liegt in einer Formelmenge M genau dann vor, wenn es eine Substitution *sub* gibt, so dass $sub(\{F_1, ...F_n\}) \subseteq M$. Die **Regel R** kann auf die Formelmenge **M angewendet werden**, wenn **das Muster** $F_1, ...F_n$ **in M vorliegt**.

Eine (Formel-) Substitution ist eine Funktion sub:$L_{AL} \rightarrow L_{AL}$,die jede Formel F auf eine Formel $H := sub(F)$ abbildet. $L_{AL}-$ ist die Menge der Formeln der Aussagenlogik. Jedes Vorkommen einer Teilformel A_i in F wird durch die

entsprechende Formel H_i ersetzt. Ist M eine Menge von Formeln, dann ergibt die Substitution die Formelmenge $sub(M) = \{sub(F)|F \in M\}$.

Jede atomare Formel wird immer wieder durch die gleiche Formel ersetzt, diese Eigenschaft heisst Uniformität. Die Substitution ist durch die Zuordnung von Formeln zu den atomaren Formel A_i, eindeutig bestimmt.

- Es seien $A_1, ...A_n$ atomare Formeln und $H_1, ...H_n$ Formeln dann ergibt sich:
  $sub(A_1) = H_1, ..., sub(A_n) = H_n$
  Für die komplexen Formeln ergibt sich dann:
  $sub(\neg F) = \neg sub(F)$
  $sub((F \vee G)) = (sub(F) \vee sub(G))$
  $sub((F \wedge G)) = (sub(F) \wedge sub(G))$
  $sub((F \rightarrow G)) = (sub(F) \rightarrow sub(G))$
  $sub((F \leftrightarrow G)) = (sub(F) \longleftrightarrow sub(G))$

Modus ponens(lat) bedeutet **Modus der Behauptung**. Der Modus ponens ist eine Inferenzregel, die es erlaubt, **von einer Aussage A** und einer **Implikation** $A \rightarrow B$ (wenn A, dann B) **auf** B **zu schliessen** (wenn A und $A \rightarrow B$ dann B). Damit wird eine ausgesprochen natürliche Form des Schliessens formalisiert, die auch im Alltagsleben weit verbreitet ist und in verschiedenen Kalkülen angewendet wird.

Beispiel:

Wenn es regnet, gehe ich nicht nach draussen ($A \rightarrow B$). Es regnet (A). Also: Ich gehe nicht nach draussen (B).

Beispiel:

- $M_1 = \{A, A \rightarrow B, A \rightarrow C\}$
  $sub_1(A) = A$, $sub_1(B) = B$ $\quad sub_1(\{A, A \rightarrow B\}) = \{A, A \rightarrow B\} \subseteq M_1$
  $sub_2(A) = A$, $sub_2(B) = C$ $\quad sub_2(\{A, A \rightarrow B\}) = \{A, A \rightarrow C\} \subseteq M_1$
- $M_3 = \{A \wedge B, (A \wedge B) \rightarrow \neg(C \vee D)\}$
  $sub_3(A) = A \wedge B$, $sub_3(B) = \neg(C \vee D)$
  $sub_3(\{A, A \rightarrow B\}) = \{A \wedge B, (A \wedge B) \rightarrow \neg(C \vee D)\} \subseteq M_3$

Wenn es eine Substitution sub gibt, so dass $sub(\{F_1, ...F_n\}) \subseteq M$,dann kann die Formel $sub(G)$ aus M mit R in einem Schritt abgeleitet werden.

- Andere Schreib-/Sprechweisen:
  $sub(G)$ ist mit R direkt ableitbar aus M
  $sub(G)$ ist mit R direkt ableitbar aus $sub(F_1)$, ..., $sub(F_n)$
  $M \vdash_R sub(G)$ bzw $\{sub(F_1), ..., sub(F_n)\} \vdash_R sub(G)$
  Beispiel (Modus ponens)
  $MP = \frac{A, A \rightarrow B}{B}$

- $M_1 = \{A, A \to B, A \to C\}$ $\quad M_1 \vdash_{MP} B \quad M_1 \vdash_{MP} C$
- $M_2 = \{A \wedge B, (A \wedge B) \to \neg(C \vee D)\}$ $\quad M_2 \vdash_{MP} \neg(C \vee D)$

2.3.2 Resolution

Die Resolution ist ein von Robinson Anfang der sechziger Jahre entwickelter Logikkalkül. Die Resolution basiert auf einer einzigen Inferenzregel der Form: $(P \vee Q) \wedge (\neg\ Q \vee R) \Rightarrow (P \vee R)$. Durch die Kombination dieser Regel mit der Unifikation und der Faktorisierung ergibt sich die Resolutionsregel für die Prädikatenlogik . Resolution ist ein vollständiger Widerlegungskalkül für Formeln in Klauselform, d.h., für jede inkonsistente Menge von Klauseln existiert eine Resolutionsherleitung der leeren Klausel, die dem logischen falsum entspricht. Das bedeutet, aus einer unerfüllbaren Klauselmenge kann mit dem Resolutionskalkül immer die leere Klausel $\square$ abgeleitet werden. Eine Anzahl von Verfeinerungen der Resolution wurde entwickelt, u.a. die Hyperresolution und die Subsumption, die erfolgreich im automatischen Beweisen angewandt werden. Die SLD-Resolution, die hocheffizient zu implementieren, aber lediglich für Hornklauseln vollständig ist, spielt eine wichtige Rolle in der Logikprogrammierung . Auf der SLD-Resolution basiert auch die Programmiersprache Prolog .

Ist K eine **Klausel** (Disjunktion von Literalen) mit $K = (\bigvee_{k=1}^{m} L_k)$, dann nennen wir $K = \{L_1, ...L_m\}$ die Mengendarstellung von K

Ist F eine aussagenlogische Formel in konjunktiver Normalform (KNF), mit $F = (\bigwedge_{i=1}^{n}(\bigvee_{k=1}^{m,i} L_{i,k}))$, dann nennen wir $F = \{\{L_{1,1}, ..., L_{1,m1}\}, ..., \{L_{n,1}, ..., L_{n,mn}\}\}$ die Mengendarstellung von F.

- **Definition (Resolvente)**

Seien K_1 und K_2 Klauseln, dargestellt als Mengen von Literalen ((negierte) atomare Formel). Sei L ein Literal mit $L \in K_1$ und $\overline{L} \in K_2$, R heisst Resolvente von K_1 und K_2, falls $R = (K_1 - \{L\}) \cup (K_2 - \{\overline{L}\})$

- **Resolution**

Falls $K_1 = \{L\}$ und $K_2 = \{\overline{L}\}$, so ist die Resolvente leer ($R = \varnothing$). Die leere Menge wird in diesem Fall als leere Klausel bezeichnet und durch $\square$ symbolisiert.

Gegeben: Eine Formel in KNF

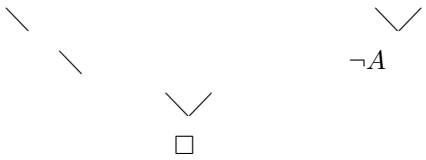
- $(\neg A \vee \neg B \vee \neg D) \wedge \neg E \wedge (\neg C \vee A) \wedge C \wedge B \wedge (\neg G \vee D) \wedge G$

  Resolutionsableitung als Baum von Klauseln:

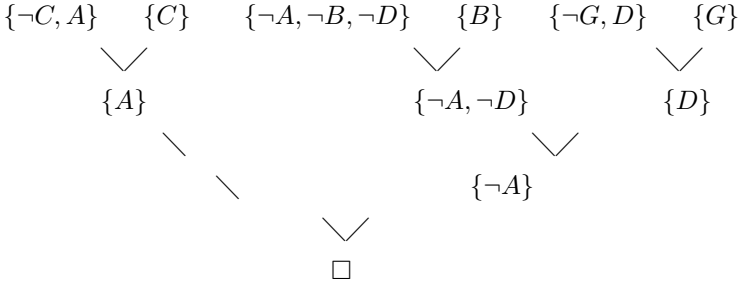
  $(\neg C \vee A) \quad C \qquad (\neg A \vee \neg B \vee \neg D) \quad B \qquad (\neg G \vee D) \quad G$

  $\quad\quad\diagdown\diagup \qquad\qquad\qquad\diagdown\diagup \qquad\qquad\diagdown\diagup$

  $\quad\quad A \qquad\qquad\qquad \neg A \vee \neg D \qquad\qquad D$



Resolutionsableitung als Baum von Klauseln in der Mengendarstellung:



2.4 Definition Prädikatenlogik

2.4.1 Unterschied zu Aussagenlogik

lat. praedicare - aussagen / griech. logike - Vernunftlehre

In der Prädikatenlogik ist es erlaubt, anstelle von Aussagen über elementare Einheiten wie in der Aussagenlogik allgemeine Aussagen über Individuen und Beziehungen zwischen Individuen zu machen. Dazu verwendet man logische Operatoren, Variablen und zusätzlich die beiden Quantoren $\exists$(Existenzquantor) und $\forall$(Allquantor).

Der Wahrheitswert einer Aussagenverbindung hängt nur von den Wahrheitswerten ihrer Bestandteile ab. Die einfachen Aussagen werden dabei als unzerlegbar angesehen, ihre innere Struktur ist nicht von Interesse. Dieser Verzicht charakterisiert die Aussagenlogik und unterscheidet sie von der Prädikatenlogik. Im Gegensatz dazu steht die Prädikatenlogik, welche sich mit den Strukturen von Aussagen befasst, ohne die inhaltlichen Bedeutungen der vorkommenden Mengen, Funktionen und Prädikate in Betracht zu ziehen.

Prädikatenlogik ist eine Erweiterung der Aussagenlogik.Die Prädikatenlogik unterscheidet von der Aussagenlogik grösserer Ausdrucksstärke.

Definition: Das Vokabular der Prädikatenlogik besteht aus

1. einer abzählbaren Menge von Variablen: x_i mit i = 1, 2, ...
2. einer abzählbaren Menge von Prädikatensymbolen der Form: P, Q, R, $P_i, ...$
3. einer abzählbaren Menge von Funktionssymbolen der Form: f, g, h, $f_i, ...$
4. der Junktoren: $\neg, \vee, \wedge, \rightarrow$
5. den Quantoren $\forall, \exists$
6. Klammern: (,)

Aussagen sind die kleinsten bedeutungstragenden Einheiten der Aussagenlogik. Prädikatenlogik berücksichtigt die interne Struktur von Aussagen und erlaubt es, zusätzlich bestimmte Beziehungen zwischen Öbjektenßum Ausdruck zu bringen. Sind möglich (z.B. Datenbankbereich) systematische Beziehungen zwischen Aussagen und Fragen.

Beispiele:

- Welche Stadt ist die zweitgrösste Stadt der Bundesrepublik Deutschland? Hamburg ist die zweitgrösste Stadt der Bundesrepublik Deutschland.
- Hamburg liegt südlich von München.

- 8 ist Kubikzahl.
  16 ist eine Quadratzahl.
  8 ist die Hälfte von 16.
  →Es gibt eine Kubikzahl, die die Hälfte einer Quadratzahl ist.
  27 ist eine Kubikzahl.
  54 ist keine Quadratzahl.
  27 ist die Hälfte von 54.
  →Nicht alle Kubikzahlen sind die Hälfte einer Quadratzahl.

Jedem Funktions- und jedem Prädikatensymbol ist eindeutig eine Stelligkeit k ≥ 0 zugewiesen. Funktionssymbole der Stelligkeit 0 heissen Konstanten. Bei nullstelligen Funktionssymbolen sind keine Klammern erforderlich. Falls die Stelligkeit nicht aus dem Kontext hervorgeht, wird sie als Index (Superskript) geschrieben:p_i^k, f_i^k

Der Index i (Subskript) dient gegebenfalls (wie bei der Aussagenlogik) der Unterscheidung von Prädikaten- bzw. Funktionssymbolen.

- **Induktive Definition der Terme: T-1**Jede Variable ist ein Term.

  T-2 Falls f ein Funktionssymbol der Stelligkeit k ist, und falls $t_1, ...t_k$ Terme sind, so ist $f(t_1, ...t_k)$ ein Term.

  T-3 Dies sind alle Terme.

  Induktive Definition der Formeln:

  F-1 Falls P ein Prädikatensymbol der Stelligkeit k ist, und falls $t_1, ...t_k$ Terme sind, so ist $P(t_1, ...t_k)$ eine Formel.

  F-2 Falls F und G Formeln sind, so sind auch folgende Konstrukte Formeln:

  $$\neg F, (F \vee G), (F \wedge G), (F \rightarrow G)$$

  F-3 Falls x eine Variable und F eine Formel, so sind auch $\exists xF$ und $\forall xF$ Formeln.

  F-4 Dies sind alle Formeln.

  Beispiele:

  Terme:

  x $\qquad$ $f_i^2(x, y)$ $\qquad$ $g(x, y)$

  $quadrat(2), mutter(peter), ...$

  "peter" wird als null-stelliges Funktionssymbol (Konstante) eingeführt.

  "2" kann ebenfalls als Konstante verwendet werden.

  Formeln:

  $P(x, y, z)$ $\qquad$ $\neg P(a, b) \wedge Q(b, c)$ $\qquad$ $\exists xR(x)$ $\qquad$ $\forall yP2(x, b)$

  $südlich(münchen, hamburg)$

  $\forall x \forall y(südlich(x, y) \rightarrow \neg südlich(y, x))$

  $\forall x \forall y(südlich(x, y) \rightarrow nördlich(y, x))$

  Definition (gebundene / freie Variable):

1. Ein Vorkommen der Variablen x ist gebunden in der Formel F, falls x in einer Teilformel von F vorkommt, die die Gestalt $\exists xF$ oder $\forall xF$ hat. In diesem Fall wird x durch den $\exists$(Existenzquantor) bzw. den $\forall$(Allquantor) gebunden.

2. Anderenfalls ist das Vorkommen von x frei in F.

3. Eine Formel ohne freies Vorkommen von Variablen heisst geschlossen. Geschlossene formeln werden auch als Aussagen bezeichnet.

4. Die Matrix einer Formel F ist die Formel F^*, die aus F dadurch hervorgeht, dass alle Quantoren und die direkt hinter ihnen stehenden Variablen gestrichen werden.

Beispiel:

$F = \exists xP(x, f(x, y)) \vee \neg\exists y\forall xQ(y, g(x, z))$

Formel/Teilformel	Nur gebundene Vorkommen	Nur freie Vorkommen
$\exists xP(x, f(x, y))$	x	y
$\neg\exists y\forall xQ(y, g(x, z))$	x, y	z
F	x	z

Interpretation in der Aussagenlogik:

Atomare Aussagen werden durch Belegungen A_i Wahrheitswerte zugewiesen. Die Belegungen A_i werden zu Wahrheitswertzuweisungen für komplexe Formeln der Sprache L_{AL} fortgesetzt.

- **Interpretation in der Prädikatenlogik:**

Die atomare Bestandteile der prädikatenlogischer Ausdrücke werden durch Entitäten über einer Grundmenge (Domäne) interpretiert. Freie Variablen durch Objekte der Grundmenge, Funktionssymbole durch Funktionen über der Grundmenge. Prädikatensymbole durch Prädikate(Relationen) über der Grundmenge. Komplexe Ausdrücke werden hierauf aufbauend interpretiert.

→Auch in der Semantik der Prädikatenlogik spielen Wahrheitswerte eine zentrale Rolle.

→Interpretationen führen nicht direkt zu Wahrheitswerten sondern indirekt über die Interpretation einer Grundmenge (Domäne).

Man kann die Prädikatenlogik auf der Aussagenlogik reduzieren. Wenn alle Prädikatensymbole einer prädikatenlogischen Sprache null-stellig sind, so können diese als Entsprechungen für die atomaren Formeln einer aussagenlogischen Sprache angesehen werden. -Terme, Variablen und Funktionssymbole, aber insbesondere auch Quantoren sind in einer derartigen prädikatenlogischen Sprache überflüssig. Die prädikatenlogische Semantik wird zur aussagenlogischen Semantik reduziert. Wenn in einer prädikatenlogischen Sprache weder Variablen noch Quantoren auftreten, ist ebenfalls eine Reduktion auf die Aussagenlogik möglich. Die atomaren Formeln einer derartigen prädikatenlogischen Sprache können als atomare Formeln einer aussagenlogischen Sprache angesehen werden. Auch in diesem Fall wird die prädikatenlogische Semantik zur aussagenlogischen Semantik reduziert.Aber die zusätzliche Ausdrucksstärke der Prädikatenlogik geht dabei verloren.

2.4.2 Anwendungsbeispiel (Knowledge Engineering)

Unter Knowledge Engineering versteht man die Gewinnung von Wissen und einbindung dieses Wissens in ein Expertensystem. Im Vordergrund steht dabei im wesentlichen nicht die Erklärung von Expertenleistung, sondern der Transfer von Expertenwissen auf maschinelle Systeme. Häufig versteht man unter Knowledge Engineering den gesamten Erstellungs und Wartungsprozess eines wissensbasierten Systems (inkl. Machbarkeitsstudien, Auswahl von Werkzeugsystemen, Wissensakquisition bis hin zur Integration eines fertigen Systems in die Einsatzumgebung, Wartung und ggb. Erweiterung). [Ruck87] beschreibt die Aufgabe eines Knowledge Engineers wie folgt

"Wesentliche Aufgabe des Knowledge Engineers ist das Herausarbeiten von Wissen und seine Übertragung in geeignete Wissensrepräsentationsschemata, die von den entsprechenden Ableitungsmechanismen verarbeitet werden können. In Zusammenarbeit mit dem Benutzer und dem Fachexperten baut der Knowledge Engineer anwendungsspezifische Expertensysteme auf. Er fragt Sach- und Erfahrungswissen vom Fachgebietsexperten ab und überträgt es in ein Expertensystem."

andere Beschreibungen sind

"Der Begriff Knowledge Engineering wurde geprägt, um den Prozess zu beschreiben, wie menschliche Expertise extrahiert und derart kodifiziert wird, dass sie in einem Rechnerprogramm verarbeitet werden kann" [Crasemann/Krasemann 1988, S. 43]

Das deutsche "Wissen" unterscheidet sich vom englischen "Knowledge". Das Wissen zielt auf wahre Aussagen über die Welt und Knowledge umschliesst auch Kenntnisse, Fertigkeiten und Fähigkeiten. Wissen ist ein Potential des Menschen und Knowledge des künstlich-intelligenten Systems.

Die wesentlichen Grundprobleme des Knowledge Engeneering sind:

1. Erhebung von Wissen aus verschiedenen Wissensquellen um einen bestimmten Anwendungsproblem zu bewältigen.
2. Wie sollte dieses Wissen geordnet werden?
3. Mit welchen Maßstäben oder Kriterien könnte ein Bestand an Wissen bewertet werden?

Bei Formalisierung wird eine natürlich-sprachliche Beschreibung der Welt in eine formal-sprachliche Beschreibung überführt.

Beim Wissenserwerb sind verschiedene Vorgehensmodelle möglich.

Beispiele:

Phasenmodell - nach Puppe:

1. Klassifizierung unter Problemlösungstyp.
2. Zuordnung zur Lösungsstrategie.

3. Abbildung des Wissens in Wissensrepräsentation und Steuerstrategie.
4. Implementation von Wissensrepräsentationssprache und Steuerungssystem.

Puppe (nach Software- Engineering-Phasen):

1. Problemidentifizierung (Beteiligte, Ressourcen, Ziele und Problemcharakteristika).
2. Knappe bzw. vage Problemformulierung.
3. Detaillierte Anforderungsdefinition.
4. Spezifikation.
5. Kodierung.

Buchanan:

1. Problemdefinition (Einordnung in Problemklasse, erste Vorstellungen zum Verarbeitungsmodell).
2. Erhebung (Wissenserwerb).
3. Konzeptionalisierung (Definition des Versrbeitungsmodells, Wissensrekonstruktion: Objekte, Relationen, Ereignisse).
4. Formalisierung (Aufbau der Wissensbasis).
5. Implementierung der Inferenzmaschine.
6. Kodierung.
7. Validierung.
8. Wartung.

Harmon/ Mauss/ Morrissey:

1. Vorstudie.
2. Aufgabenanalyse.
3. Prototypenentwicklung.
4. Systementwicklung.
5. Erprobung.
6. Implementierung.
7. Wartung.

MIKE- Ansatz:

1. Erhebung.

2. Interpretation.
3. Formalisierung.
4. Entwurf.
5. Implementation.

Wesentliche Erfolge von Knowledge Engineering:

1. Wissenserwerb ist immer Konstruktion von etwas Neuem und ist als Wissensmodellierungsprozess zu verstehen.
2. Bei der Entwicklung wissensbasierter Systeme ist ein Verständnis verschiedener Wissensarten und ihrer Rolle erarbeitet worden.
3. Aus Problemlösungsmethoden lässt sich der Wissenserwerbsprozess zielgerichtet durchführen.
4. Ontologien erlauben Fachbegriffe präzise zu fassen und sind wichtig für die Kommunikation zwischen Fachexperten und Wissensingenieuren.
5. Ontologie ist eine formale. explizite Spezifikation von Beziehungen, die in einem Weltausschnitt von bedeutung sind.

Nach [Norv03] lässt sich der Prozess des Knowledge Engineering in sieben unterschiedliche Schritte unterteilen:

- Beschreibung der Aufgabe
- Erwerb des notwendigen Wissens
- Festlegung des Vokabular von Prädikaten, Funktionen, Konstanten.
- Kodierung des allgemeinen Wissens
- Kodierung eines spezifischen Anwendungsfalles
- Testen der Inferenzprozedur
- Debugging der Wissensbasis

Beschreibung der Aufgabe: Der Knowledge Engineer legt fest was die Wissensbasis leisten soll. Welche Fragestellungen sie behandeln soll und welches Wissensgebiete dafür notwendig sein werden.

Erwerb des notwendigen Wissens: In diesem Schritt wird alles notwendige Wissen erworben. Entweder durch den Knowledge Engineer selbst oder durch hinzugezogene (externe) Experten. Eine formelle Beschreibung ist in diesem Schritt noch nicht notwendig.

Festlegung des Vokabular von Prädikaten, Funktionen, Konstanten.: Die wesentlichen Elemente des im dritten Schritt noch informell beschriebenen Wissens wird in Prädikate, Funktionen und Konstanten überführt. Das Resultat

dieser Überlegungen wird **Ontologie** genannt.

Kodierung des allgemeinen Wissens: In diesem Schritt werden die Axiome für das Vokabular und damit auch die Bedeutung der jeweiligen Sätze festgelegt. Dies ermöglicht es auf mögliche Inkonsistenzen und Fehler zu stoßen und diese zu bereinigen (durch einen Rückfall zum vorangegangenen Schritt)

Kodierung eines spezifischen Anwendungsfalles: Die Kodierung des allgemeinen Wissens zeigt wie gut (oder schlecht) die gewählte Ontologie ist. Hier werden einfache atomare Sätze über Instanzen der Ontologie kodiert.

Testen der Inferenzprozedur: Beim Testen der Inferenzprozedur wird die Leistungsfähigkeit der Inferenzprozedur getestet. Inferenzen werden ausgenutzt um Antworten auf gegeben Fragestellungen zu erhalten.

Debugging der Wissensbasis: Es wird geprüft ob (alle) Antworten korrekt sind. Identifizierung fehlender oder schwacher Axiome.

2.4.3 Beispiel: Digitaler Schaltkreis

Das folgende Beispiel - das sich wiederum an [Norv03] orientiert - entwickelt eine Wissensbasis für digitale Schaltkreise. Zur Illustration wird das in Abbildung 2.1 dargestellte 2 aus 3 Schaltkreis verwendet. Die entsprechende Wahrheitstabelle ist in Tabelle 2.1 dargestellt.

Z	Y	X	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Tabelle 2.1: Wahrheitstabelle: 2 aus 3 Schaltung

Im folgenden werden die sieben Schritte des Knowledge Engineering für dieses Beispiel entwickelt.

Erwerb des notwendigen Wissens

Ein **digitaler Schaltkreis** besteht aus *und* (and), *oder* (or) und *nicht* (not) **Gattern**. Die ersten beiden Gatter (und, oder) haben je mindestens zwei Eingänge und einen Ausgang (also insgesamt mindestens je drei **Anschlüsse**). Die Negation nur je einen Eingang und einen Ausgang. Ein- und Ausgänge werden untereinander über Leitungen verbunden und nehmen **Signale** auf bzw. geben welche ab. Somit besteht das Vokabular aus folgenden Elementen:

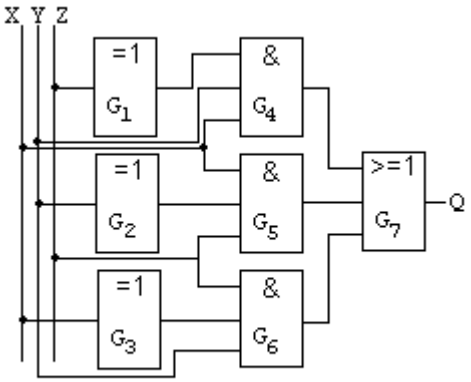


Abbildung 2.1: Die 2 aus 3 Schaltung gibt immer dann einen High Pegel an Q durch, wenn an genau zwei der drei Anschlüsse X,Y,Z ebenfalls ein High Pegel anliegt. Liegt an keinem, nur an einem oder an allen drei ein High an, so liegt an Q ein Low Pegel an.

- Digitaler Schaltkreis
- Gatter
- Anschlüsse
- Signalen

Diese Wissensbasis ist für das Beispiel ausreichend - könnte aber für einige andere Anwendungsfälle zu eng umfasst sein. Die Wissensbasis enthält zum Beispiel keinerlei Informationen über Spannungen oder andere Details die beim Debugging von Gattern notwendig sein könnten.

Festlegung des Vokabulars

Nachdem das notwendige Wissen feststeht ist nun bekannt das das Wissen aus Schaltkreisen, Gattern, Anschlüssen und Signalen besteht. Diese müssen jetzt mit Prädikaten, Funktionen und Konstanten bezeichnet werden.

Ein Schaltkreis besteht aus einem oder mehreren Gattern. Diese Gatter müssen untereinander stets unterscheidbar sein. Deshalb werden sie mit den Konstanten G_1, G_2,... bezeichnet. Die unterschiedlichen Gatter einer Schaltung gehören jedoch stets zu einem Gattertyp - ist also entweder ein und-Gatter, ein oder-Gatter oder ein nicht-Gatter. Diese Gattertypen werden durch die Konstanten *UND*, *ODER* bzw. *NICHT* repräsentiert. Zusätzlich existiert eine Funktion *Typ* - die den Gattertyp eines beliebigen Gatters bestimmt. In der Schaltung ist $\text{Typ}(G_1) = \text{Typ}(G_2) = \text{Typ}(G_3) = \text{NOT}$ bzw. $\text{Typ}(G_4) = \text{Typ}(G_5) = \text{Typ}(G_6) = \text{UND}$.

Das Signal eines Ausganges wird durch das Prädikate **Signal** beschrieben - Ein- bzw. Ausgänge durch die Prädikate **Ein** und **Aus**. Es gibt die beiden

Signale High-Pegel und Low-Pegel die durch die Konstanten **1** bzw. **0** dargestellt werden. Des weiteren können je ein Ein- und ein Ausgang miteinander verbunden werden. Dies geschieht mittels des Prädikates **Verbunden**.

Kodierung des allgemeinen Wissens

Die folgenden Aussagen sind allgemeingültig für alle digitalen Schaltungen. Wenn die Anschlüsse x und y miteinander verbunden sind, so haben sie den selben Signalpegel.
$\forall x, y : Verbunden(x, y) \rightarrow Signal(x) = Signal(y)$

An einem Signal liegt entweder High- oder Lowpegel an.
$\forall x : Signal(x) = 1 \vee Signal(x) = 0(1 \neq 0)$

Das Prädikat *Verbunden* ist kommutativ.
$\forall x, y : Verbunden(x, y) \Leftrightarrow Verbunden(y, x)$

Informationen über die einzelnen Typen:
$\forall g : Typ(g) = ODER \Rightarrow Signal(Aus(1, g)) = 1 \Leftrightarrow \exists n : Signal(Ein(n, g)) = 1$
$\forall g : Typ(g) = UND \Rightarrow Signal(Aus(1, g)) = 0 \Leftrightarrow \exists n : Signal(Ein(n, g)) = 0$
$\forall g : Typ(g) = NICHT \Rightarrow Signal(Aus(1, g)) \neq Signal(Ein(1, g))$

Kodierung eines spezifischen Anwendungsfalles

Der spezifische Anwendungsfall ist das in Abbildung 2.1 dargestellte JK-FlipFlop. Zuerst werden die Typen der einzelnen Gatter gemäß der Schaltung festgelegt

Typ(G_1) = NICHT
Typ(G_2) = NICHT
Typ(G_3) = NICHT
Typ(G_4) = UND
Typ(G_5) = UND
Typ(G_6) = UND
Typ(G_7) = ODER

dann die Verbindungen der Gatter untereinander

Verbunden(Aus(1, G_1), Ein(1,G_4)
Verbunden(Aus(1, G_2), Ein(2,G_5)
Verbunden(Aus(1, G_2), Ein(2,G_6)
Verbunden(Aus(1, G_4), Ein(1,G_7)
Verbunden(Aus(1, G_5), Ein(2,G_7)
Verbunden(Aus(1, G_6), Ein(3,G_7)

Testen der Inferenzprozedur
In diesem Schritt werden Fragen an die Inferenzprozedur gestellt. Zum Beispiel die Frage welche Kombinationen von x,y bzw z. einen High Pegel am Ausgang erzeugen, also

$\exists x, y, z Signal(Ein(G_1, 1)) = Signal(Ein(G_5, 3)) = Signal(Ein(G_6, 1)) = z \wedge$

$Signal(Ein(G_1, 1)) = Signal(Ein(G_4, 2)) = Signal(Ein(G_6, 3)) = y \wedge$
$Signal(Ein(G_2, 1)) = Signal(Ein(G_5, 1)) = Signal(Ein(G_4, 3)) = x \wedge$
$Signal(Aus(G_7, 1)) = 1$

Diese Anfrage würde zu den Substitutionen

$$\{x/1, y/1, z/0\} \ \{x/0, y/1, z/1\} \ \{x/1, y/0, z/1\}$$

(vergl. Wahrheitstabelle) führen.

Debugging der Wissensbasis
Das Debuggen der Wissensbasis geschieht durch das stellen verschiedenster sinnvoller Anfragen an das System - durch den Vergleich mit vorher bekannen Werten lässt sich dann ermitteln ob die Wissensbasis bzw. alle Axiome korrekt sind. Ein weglassen des Axioms $1 \neq 0$ würde dazu führen das die Inferenzprodur keine einzige Frage mehr beantworten kann.

2.5 Inferenz in der Prädikatenlogik

In der Prädikatenlogik gibt es im Grunde die selben Inferenzregeln wie bei der Aussagenlogik. Die Inferenzen werden in der Form

$$\frac{\textbf{Vorraussetzungen}}{\textbf{Folgerung}}$$

notiert. Der bereits aus der Aussagenlogik bekannte **Modus Ponens** muß (wie alle anderen Inferenzregeln aus der Aussagenlogik auch) um Quantoren erweitert werden.

Modus Ponens $\frac{a \quad (a \Rightarrow b)}{b}$

$$\frac{mensch(sokrates) \quad (\forall x{:}(mensch(x) \Rightarrow sterblich(x))}{sterblich(sokrates)}$$

In der Prädikatenlogik kommt oft eine Inferenzregel namens **Generalisierter Modus Ponens** verwendet. Für atomare Sätze p_i, p_i' und q mit Substitution θ der Form $\forall i : subst(\theta, p_i) = subst(\theta, p_i')$ gilt:

$$\frac{p_1, p_2, \ldots, p_n, \ (p_1 \wedge p_2 \wedge p_n \Rightarrow q)}{subst(\theta, q)}$$

2.5.1 Unifikation

Zwei Terme können durch geeignete Substitutionen syntaktisch identisch gemacht werden. Diese Substitutionen heissen **Unifikator**. Identische Terme sind natürlich ohne Substitution **unifizierbar**.

$Unifikation(p, q) = \theta \; mit \; Subst(\theta, p) = subst(\theta, q)$

Das nachfolgende Beispiel zeigt die Funktionsweise eines Unifikationsalgorithmuses anhand einer fiktiven Datenbasis, die auf der Anfrage *Kennt(Mark, x)* basiert.

$Unifikation(Kennt(Mark, x), Kennt(Mark, Dieter)) = \{x/Dieter\}$
Hier wird einfach das x aus dem ersten Term durch Dieter substituiert.

$Unifikation(Kennt(Mark, x), Kennt(y, Karl)) = \{y/Mark, x/Karl\}$
Hier wird einfach das y aus dem zweiten Term durch Mark substituiert und das x durch Karl.

$Unifikation(Kennt(Mark, x), Kennt(y, Vater(y)) = \{y/Mark, x/Vater(Mark)\}$
Hier wird einfach das y durch Mark substituiert und das y in Vater(y) somit zu Mark und x wird durch Vater(Mark) ersetzt.

$Unifikation(Kennt(Mark, x), Kennt(x, Frederike) =$ **nicht möglich**
Nicht möglich da x nicht gleichzeitig die Werte Mark und Frederike annehmen kann. Das Problem kann durch eine Umbennung einer der Variablen ermöglicht werden.

$Unifikation(Kennt(Mark, x), Kennt(y, Frederike) = \{y/Mark, x/Frederike)\}$

Wie bereits angedeutet kann es mehr als eine solche Substitution geben. Deshalb ist vor allem der sogennante **Most General Unifier** (MGU) oder Allgemeinster Unifikator von Interresse. Der MGU ist ein Unifikator θ aus einer Literalmenge L, für den gilt das es zu jedem anderen Unifikator $\acute{\theta}$ aus L einen Substitutor τ gibt mit $\theta\tau = \acute{\theta}$. Es ist bekannt das jede unifizierbare Menge von Literalen einen MGU besitzt (Unifikationssatz nach Robinson).

Das folgende Beispiel

$Unifikation(Kennt(Mark, x), Kennt(y, z)$

hat u. a. die folgenden beiden Unifikatoren

$\{y/Mark, x/z)\}$
$\{y/Mark, x/Mark, z/Mark)\}$

die zu den Resultaten Kennt(Mark, z) bzw. Kennt(Mark, Mark). Wie man sieht kann das zweite aus dem ersten gewonnen werden. Der erste Unifikator ist also allgemeiner als der andere.

2.5.2 Forward Chaining

Forward Chaining (Vorwärtsverkettung) ist ein relativ einfaches Inferenzverfahren. Es werden alle Regeln mittels Modus Ponens ausgewertet deren Vorbedingungen erfüllt sind und der Schluß dem bekannten Wissen hinzugefügt. Dieser

Schritt wird solange ausgeführt bis entweder keine Regeln mehr angewant werden kann oder die Anfrage bereits beantwortet worden ist.

function FOL-FC-ASK(KB, α) **returns** a substitution or *false*

repeat until *new* is empty
- *new* $\leftarrow \{\}$
- **for each** sentence r **in** KB **do**
  - $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-APART(r)
  - **for each** θ such that $(p_1 \wedge \ldots \wedge p_n)\theta = (p'_1 \wedge \ldots \wedge p'_n)\theta$ for some $p'_1, \ldots, p'_n$ in KB
    - $q' \leftarrow$ SUBST(θ, q)
    - **if** q' is not a renaming of a sentence already in KB or *new* **then do**
      - add q' to *new*
      - $\phi \leftarrow$ UNIFY(q', α)
      - **if** ϕ is not *fail* **then return** ϕ
- add *new* to KB

return *false*

Abbildung 2.2: Ein Forward Chaining Algorithmus (nach [Norv03])

Das folgende Beispiel soll die Funktionsweise des Forward Chaining anhand der Familienverhältnisse in Abbildung 2.2 verdeutlichen:

Vater(Michael, Frederike)
Vater(Georg, Malte)
Vater(August, Georg)
Vater(August, Mike)
Vater(August, Fred)
Vater(August, Silke)
Mutter(Maria, Frederike)
Mutter(Frederike, Georg)
Mutter(Fransika, Mike)
Mutter(Fransika, Fred)
Mutter(Fransika, Silke)
Mutter(Anna, Fransika)

$Vater(x, y) \Rightarrow Elternteil(x, y)$
$Mutter(x, y) \Rightarrow Elternteil(x, y)$
$Elternteil(x, y) \Rightarrow VorfahreVon(x, y)$
$Elternteil(x, y) \wedge VorfahreVon(y, z) \Rightarrow VorfahreVon(x, z)$

Damit würde Forward Chaining (der kürze wegen nur für August) die folgenden Elternteil Beziehungen schließen:
Vater(August, Georg), Vater(August, Mike), Vater(August, Fred), Vater(August, Silke)

und die folgenden VorfahreVon Beziehungen

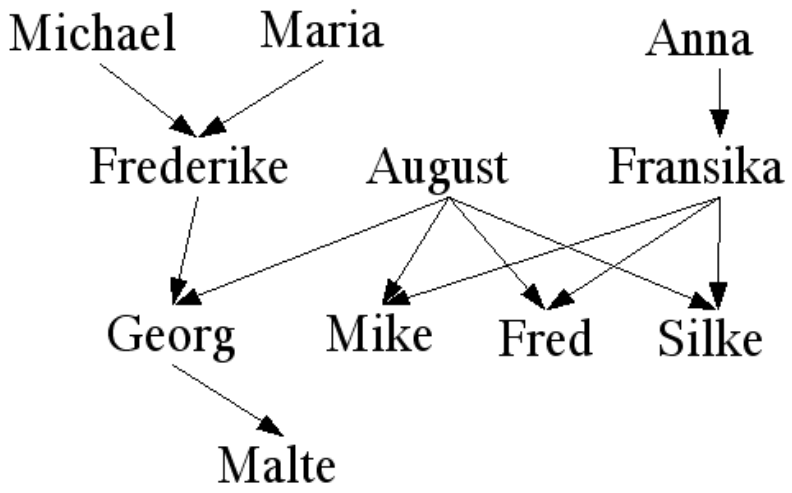


Abbildung 2.3: Familienverhältnisse zur Demonstration von Forward/Backward Chaining.

Vater(August, Georg), Vater(August, Mike), Vater(August, Fred), Vater(August, Silke)

Die Anfrage ob Georg ein Nachfahre von Maria ist wird mittels Forward Chaining wie folgt aufgelöst:
$Mutter(Frederike, Georg) \Rightarrow Elternteil(Frederike, Georg) \Rightarrow Vorfahre(Frederike, Georg)$
$Mutter(Maria, Frederike) \Rightarrow Elternteil(Maria, Frederike)$
$Elternteil(Maria, Frederike) \wedge VorfahreVon(Frederike, Georg) \Rightarrow VorfahreVon(Maria, Georg)$

2.5.3 Backward Chaining

Backward Chaining (Rückwärtsverkettung)funktioniert entgegengesetzt dem Forward Chaining. Es wird von der Fragestellung ausgegangen und untersucht ob sie sich aus der gegebenen Wissensbasis herleiten lässt. Backward Chaining wird unter anderem in der Programmiersprache Prolog verwendet.

Die Idee hinter Backward Chaining ist einfach:

- Ist die Anfrage bereits in der Wissenbasis hält der Algorithmus.
- Finde eine Regel deren Konklusionen zu der Anfrage passen.
- Ersetze die variablen der Regel entsprechend.
- Beweise die Atome die in den Prämissen der Regel auftauchen.

Auf das vorangegange Beispiel bezogen, würde die Frage ob Frederike ein Vorfahre von Malte ist wie folgt gelöst: Zuerst wird die Regel $Elternteil(x, y) \Rightarrow VorfahreVon(x, y)$ getestet:

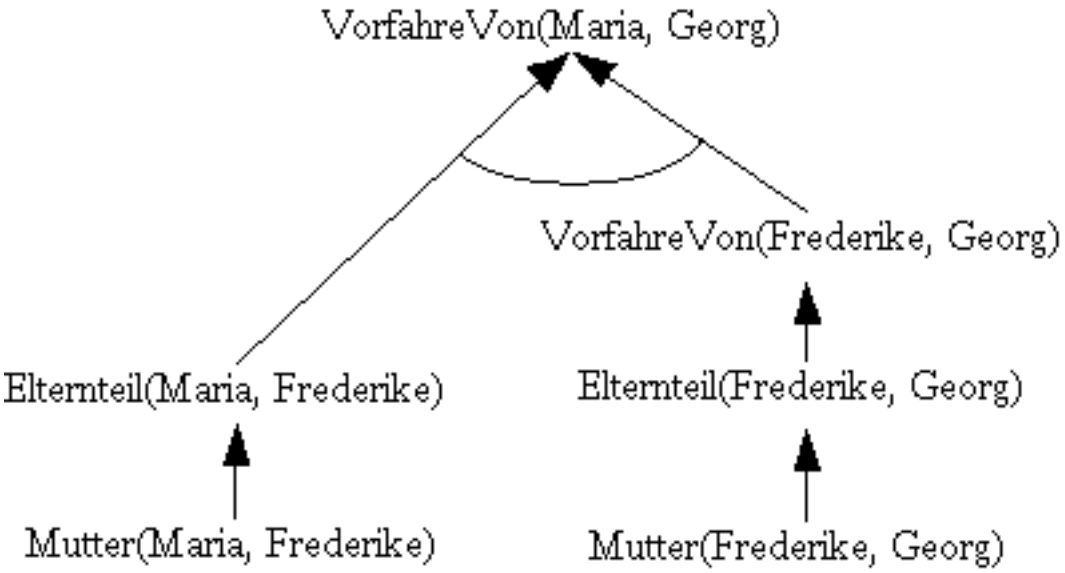


Abbildung 2.4: Forward Chaining: Ist Georg ein Nachfahre von Frederike?

$Elternteil(Frederike, Malte) \Rightarrow Nichtbeweisbar.$

dann
$Elternteil(x, y) \wedge VorfahreVon(y, z) \Rightarrow VorfahreVon(x, z)$
also
$Elternteil(Frederike, y) \wedge VorfahreVon(y, Malte) \Rightarrow VorfahreVon(x, z)$
somit müssen nun $Elternteil(Maria, y)$ und $VorfahreVon(y, Malte)$ bewiesen werden.

$Elternteil(Frederike, y) \Rightarrow Mutter(Frederike, Georg)$
und
$VorfahreVon(y, Malte) \rightarrow Elternteil(y, Malte) \rightarrow Vater(Georg, Malte)$

Abbildung 2.5 zeigt einen einfachen Backward Chaining Algorithmus. Er erwartet ein Ziel, eine Anfrage und liefert eine Liste von Substitutionen zurück.

2.5.4 Resolution

Die bereits aus der Aussagenlogik bekannte Resolution lässt sich auch auf die Prädikatenlogik übertragen. Leider ist Resolution etwas aufwändiger als das Forward-/Backward Chaining, verwendet Unifikation und verläuft in fünf Schritten:

- Herstellung der Disjunktiven Normalform (DNF)

function FOL-BC-ASK(KB, goals, θ) **returns** a set of substitutions
  inputs: KB, a knowledge base
    goals, a list of conjuncts forming a query
    θ, the current substitution, initially the empty substitution $\{\}$
  local variables: *ans*, a set of substitutions, initially empty

  if *goals* is empty **then return** $\{\theta\}$
  $q' \leftarrow$ SUBST(θ, FIRST(*goals*))
  for each r **in** KB where STANDARDIZE-APART(r) $= (p_1 \wedge \ldots \wedge p_n \Rightarrow q)$
    and $\theta' \leftarrow$ UNIFY(q, q') succeeds
    ans $\leftarrow$ FOL-BC-ASK(KB, $[p_1, \ldots, p_n | \text{REST}(goals)]$, COMPOSE($\theta', \theta$)) $\cup$ *ans*
  return *ans*

Abbildung 2.5: Ein Backward Chaining Algorithmus (nach [Norv03])

- Umbennung von Variablen
- Herstellung der Skolemform
- Quantorenelimination
- Umwandlung in eine Konjunktion von Disjunktionen

daraufhin wird die Resoltionsregel ausgeführt:

$$\frac{l_1 \vee \ldots \vee l_k, m_1 \vee \ldots \vee m_n}{SUBST(\theta, l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)}$$

mit $Unify(l_i, \neg m_j) = \theta$

Diese fünf Schritte werden im folgenden anhand des einfachen Beispielsatzes ([Norv03]) erläutert: "Jeder der alle Tiere liebt wird von jemandem geliebt".

$\forall x(\forall y Tiere(y) \Rightarrow Liebt(x, y)) \Rightarrow (\exists y Liebt(y, x))$

Herstellung der Disjunktiven Normalform (DNF)

Implikationen entfernen
$\forall x(\neg\forall y \neg Tiere(y) \vee Liebt(x, y)) \vee (\exists y Liebt(y, x))$

Negationen nach innen ziehen
$\forall x(\exists y \neg(\neg Tiere(y) \vee Liebt(x, y)) \vee (\exists y Liebt(y, x))$
$\forall x(\exists y \neg\neg Tiere(y) \wedge Liebt(x, y)) \vee (\exists y Liebt(y, x))$
$\forall x(\exists y Tiere(y) \wedge \neg Liebt(x, y)) \vee (\exists y Liebt(y, x))$

Umbennung von Variablen
$\forall x(\exists y Tiere(y) \wedge \neg Liebt(x, y)) \vee (\exists z Liebt(z, x))$

Skolemisierung
$\forall x(Tiere(F(x)) \wedge \neg Liebt(x, F(X))) \vee Liebt(G(x), x))$

Quantorenelimination
$(Tiere(F(x)) \wedge \neg Liebt(x, F(X))) \vee Liebt(G(x), x))$

Umwandlung in eine Konjunktion von Disjunktionen
$(Tiere(F(x)) \vee Liebt(x, F(X))) \wedge (\neg Liebt(x, F(x)) \vee Liebt(G(x), x))$

Jetzt kann die Resolutionsregel angwandt werden:

$$\frac{l_1 \vee ... \vee l_k, m_1 \vee ... \vee m_n}{SUBST(\theta, l_1 \vee ... \vee l_{i-1} \vee l_{i+1} \vee ... \vee l_k \vee m_1 \vee ... \vee m_{j-1} \vee m_{j+1} \vee ... \vee m_n)}$$

Daraus können die beiden Klauseln
$(Tiere(F(x)) \vee Liebt(G(x), x))$ und $(\neg Liebt(u, v) \vee \neg Tötet(u, v))$

gefolgert werden, durch Elimination der komplementären Klauseln ergibt sich als Resolvente

$(Tiere(F(x)) \vee \neg Tötet(u, v)$

Diese Resolutionsstrategie nennt sich Binäre Resolution - es gibt noch weitere Strategien

- Unit Resolution
  Ein Satz ist eine sogenannte Unit Clause (entspricht einem Literal), nicht volländig, vollständig für Horn-Klauseln.
- Set of Support
  Resolutionen eliminieren, Wissen in "set of support"S und einen Rest R unterteilen. Ein Satz ist aus der Menge S. Set of Support ist vollständig gdw. R konstistent ist.
- Subsumption
  Die Wissensbasis möglichst klein halten. Sätze entfernen die redunant sind. Vollständig.

2.6 Mächtigkeit und Effizienz der Verfahren

Vollstandigkeitssatz: (Kurt Gödel): In der Prädikatenlogik kann jeder Satz, der aus einer Wissensbasis folgt auch aus ihr inferenziert werden, d. h.

$Wenn\ WB \models \alpha,\ dann WB \vdash \alpha$

Die Folgerung ist semi-entscheidbar (wenn Sätze aus einer Prämisse folgen kann sie gezeigt werden, wenn sie das nicht tun, ist das nicht unbedingt möglich)

Forward Chaining ist wahrheitserhaltend und auch völlständig für definite Prädikatenlogische Klauseln (Horn-Klauseln). In der Regel ist Forward Chaining effizienter als Backward Chaining. Der vorgestellte Algorithmus kann noch durch einige Änderungen verbessert werden, so kann z. B. das Zeit intensive Pattern Matching vermieden werden.

Backward Chaining ist eine rekursive Tiefensuche. Der Speicherbedarf wächst dementsprechend linear. Weiterhin ist Backward Chaining unvollständig, da endlosschleifen auftreten können. Dieses Problem kann durch die Prüfe des aktuellen Zieles gegen alle anderen Ziele auf einem Stack umgangen werden. Backward Chaining ist in der Regel zeitaufwändiger als Forward Chaining.

Die Resolution ist der mächtigste Inferenzmechanismus. Sie ist widerspruchsvollständig (refutation-complete), d. h. wenn eine Menge von Sätzen nicht erfüllbar ist wird die Resolution einen Widerspruch hervorbringen.

2.7 Zusammenfassung

Die Aussagenlogik besteht lediglich aus **Aussagenvariablen** und **Junktoren**. Sie kennt lediglich die beiden Wahrheitswerte WAHR und FALSCH. Es existieren einige Inferenzverfahren für die Aussagenlogik, u. a. den Modus Ponens, den Modus Tollens und die Resolution.

Die Prädikatenlogik (Prädikatenlogik erster Stufe) kann als eine Erweiterung der Aussagenlogik mit bei weitem größerer Ausdrucksstärke betrachtet werden. Im Gegensatz zur Aussagenlogik können auch allgemeine Aussagen über Objekte und über Beziehungen zwischen diesen Objekten formuliert werden. Für die Prädikatenlogik existieren mehrere Inferenzverfahren, so z. B. das Forward Chaining, das Backward Chaining und die Resolution.

Eine praktische Anwendung der Prädikatenlogik in der Künstlichen Intelligenz ist das Knowledge Engineering. In dem Wissen durch prädikatenlogische Sätze ausgedrückt wird und durch Inferenz logische Schlußfolgerungen gezogen werden.

Literaturverzeichnis

[1] Russel S., Norvig P.: *Artificial Intelligence – A Modern Approach*, Second Edition, Prentice Hall, 2003.

[2] Günter G., *Einführung in die Künstliche Intelligenz*, Addison Wesley, 1995.

[3] Tanimoto S.L., *KI: Die Grundlagen*, Oldenbourg, 1990.

[4] Bauer F.L., Wirsing M., *Elementare Aussagenlogik*, Springer, 1991.

[5] Lifeline, *http://www.ifi.unizh.ch/CL/*, Institut für Computerlinguistik, Universität Zürich

[6] G.Görz, *Handbuch der künstlichen Intelligenz*, Oldenbourg Verlag, München, 2000.

[7] Cawsey, A.: *Künstliche Intelligenz – im Klartext*, Prentice Hall, 2002.

[8] Goos, G.: *Vorlesungen über Informatik – Band 1*, Springer, 2000.

[9] Ruckert, M.: *Berufsbild: Knowledge Engineer*, Computer Magazine, 1987, Seite 33.

[10] Gottlob, G.: *Artificial Intelligence und Wissensrepräsentation*, www.dfki.uni-kl.de/ aabecker/Mosbach/ Gottlob-Einfuehrung.pdf

[11] Crasemann, C.; Krasemann, H.: *Der Wissens-Ingenieur - ein neuer Hut auf altem Kopf - Zur Diskussion gestellt*, Informatik Spektrum Band 11, Heft 1, Februar 1988, Seite 43.