Skript zur Lehrveranstaltung

Künstliche Intelligenz

in den Studiengängen Informatik und Ingenieurinformatik

Technische Universität Ilmenau
Fakultät für Informatik und Automatisierung
Fachgebiet Künstliche Intelligenz
apl. Prof. Dr.-Ing. habil. Rainer Knauf

Wintersemester 2012/2013

Inhaltsverzeichnis

1 Einführung in die Künstliche Intelligenz

Einige Teilgebiete[1] der Künstlichen Intelligenz (KI) sind

- **Wissensrepräsentation,**
- **maschinelles Beweisen (deduktive Inferenz),**
- Sprachverarbeitung,
- Bildverarbeitung,
- Spielprogramme,
- algorithmisches Lernen (induktive und analoge Inferenz),
- Robotik,
- **KI–Sprachen** und
- **Wissensbasierte Systeme**.

Letzteres ist ein „Kerngebiet“ der KI; es findet in der akademischen Forschung und der industriellen Praxis die meiste Beachtung und Anwendung. Die meisten der anderen Gebiete entwickelten sich

- aufgrund der Tatsache, daßsie Grundlagen (z.B. Wissensrepräsentation, Deduktion, induktive und analoge Inferenz) bzw. Werkzeuge (z.B. KI–Sprachen) liefern oder
- aufgrund der inhaltlichen Verwandtschaft (Anwendung von Methoden der Wissensverarbeitung wie z.B. bei Spielprogrammen) oder
- aufgrund der Tatsache, daßes sich um sinnvolle Erweiterungen bzw. Anwendungen Wissensbasierter Systeme handelt (z.B. Sprachverarbeitung, Bildverarbeitung) oder
- (umgekehrt) aufgrund der Tatsache, daßWissensbasierte Systeme wesentliche Komponenten beitragen (wie etwa in der Robotik),

zu Teilgebieten der KI.
Der Leitgedanke der KI besteht in der Mechanisierung von „Denkprozessen“, d.h. der Anwendung von Methoden der Wissensverarbeitung. **G.W. LEIBNIZ** (wahrscheinlich der Schöpfer dieser Idee) nannte die dazu notwendigen Komponenten

- „**lingua characteristica**“ (heute würde man dies „*Wissensdarstellungssprache*“ oder schlicht „formale Wissensdarstellung“ nennen) und
- „**calculus ratiocinator**“ (ein „*Wissensverarbeitungskalkül*“, der aus einer Menge von Formulierungen in der lingua characteristica nach vorgegebenen Verarbeitungsmechanismen neue Formulierungen der lingua characteristica erzeugen kann).

Diese beiden Begriffe stehen im Mittelpunkt der Lehrveranstaltung und werden durch formale Konzepte untersetzt.
Zunächst wird eine Untersetzung auf einem (prädikaten–)logischen Niveau vorgenommen, was zielgerichtet zur Idee der Logischen Programmierung übergeleitet wird. Auf diese Weise mit einem Implementierungswerkzeug ausgerüstet können dann „problemorientiertere“ (aber ausdrucksschwächere) Wissensdarstellungs– und Wissensverarbeitungsmethoden eingeführt und umgesetzt werden.

[1] Die **fett gedruckten** Teilgebiete sind die in der Lehrveranstaltung betrachteten.

2 Logische Grundlagen

2.1 Einfache Aussagen

Gegeben sei eine Menge von Individuensymbolen und für jede natürliche Zahl n eine Menge n-stelliger Funktionssymbole und eine Menge n-stelliger Prädikatensymbole.

1. Jedes Individuensymbol ist ein variablenfreier **Term**.
2. Wenn $c_1, \ldots, c_n$ variablenfreie Terme sind und f ein n-stelliges Funktionssymbol ist, so ist $f(c_1, \ldots, c_n)$ ein variablenfreier Term.
3. Weitere variablenfreie Terme gibt es nicht.

Wenn $t_1, \ldots, t_n$ variablenfreie Terme sind und p ein n-stelliges Prädikatensymbol ist, so ist $p(t_1, \ldots, t_n)$ eine **einfache Aussage**.

2.2 Prädikate, Funktionen, Interpretation

Ein n-stelliges **Prädikat** bildet aus der Menge aller n-Tupel von Objekten eines Objektbereiches I eindeutig in die Menge der Wahrheitswerte ab: $I^n \longrightarrow \{wahr, falsch\}$.
Eine n-stellige **Funktion** bildet aus der Menge aller n-Tupel von Objekten eines Objektbereiches I eindeutig in den Objektbereich ab: $I^n \longrightarrow I$.
Eine **Interpretation** ist eine Abbildung aus der „Symbolwelt“ des PK1 in eine reale Welt:

Individuensymbole	$\longrightarrow$	*Objekte*
Prädikatensymbole	$\longrightarrow$	*Prädikate*
Funktionssymbole	$\longrightarrow$	*Funktionen*

2.3 Zusammengesetzte Aussagen

Wenn A eine Aussage ist, so ist auch $\neg A$ (die Negation von A) eine (**zusammengesetzte**) **Aussage**.
Wenn A_1 und A_2 Aussagen sind, so sind

- $(A_1 \wedge A_2)$ (die Konjunktion von A_1 und A_2),
- $(A_1 \vee A_2)$ (die Disjunktion von A_1 und A_2),
- $(A_1 \rightarrow A_2)$ (die Implikation von A_1 und A_2) und
- $(A_1 \leftrightarrow A_2)$ (die Äquivalenz von A_1 und A_2)

(zusammengesetzte) Aussagen.
Der Wahrheitswert zusammengesetzter Aussagen ist aus den Wahrheitswerten seiner Komponenten wie folgt ermittelbar:

A_1	A_2	$\neg A_1$	$A_1 \wedge A_2$	$A_1 \vee A_2$	$A_1 \rightarrow A_2$	$A_1 \leftrightarrow A_2$
$false$	$false$	$true$	$false$	$false$	$true$	$true$
$false$	$true$	$true$	$false$	$true$	$true$	$false$
$true$	$false$	$false$	$false$	$true$	$false$	$false$
$true$	$true$	$false$	$true$	$true$	$true$	$true$

2.4 Variablen und Quantifizierungen

Wenn $A(c)$ eine Aussage ist und $A(X)$ aus $A(c)$ entsteht, indem c überall durch X ersetzt wird, so sind auch $\forall X A(X)$ und $\exists X A(X)$ Aussagen. $\forall$ heißt **Allquantor**, $\exists$ heißt **Existenzquantor** und X heißt (all- bzw. existenzquantifizierte) **Variable**.
Nicht quantifizierte Variablen heißen **freie Variablen**.

Für endliche Individuenbereiche $I = \{c_1, \ldots, c_n\}$ gilt:

$\forall X A(X)$ ist äquivalent zu $\bigwedge_{i=1}^{n} A(c_i)$

$\exists X A(X)$ ist äquivalent zu $\bigvee_{i=1}^{n} A(c_i)$

2.5 Terme und Ausdrücke

1. Jede Variable ist ein **Term**.
2. Jedes Individuensymbol ist ein Term.
3. Wenn $t_1, \ldots, t_n$ Terme sind und f ein n-stelliges Funktionssymbol ist, so ist $f(t_1, \ldots, t_n)$ ein (strukturierter) Term.
4. Weitere Terme gibt es nicht.

1. Wenn $t_1, \ldots, t_n$ Terme sind und p ein n-stelliges Prädikatensymbol ist, so ist $p(t_1, \ldots, t_n)$ ein (atomarer) **Ausdruck** (eine Atomformel).
2. Wenn A ein Ausdruck ist, so ist auch $\neg A$ ein Ausdruck.
3. Wenn A_1 und A_2 Ausdrücke sind, so sind auch $(A_1 \wedge A_2)$, $(A_1 \vee A_2)$, $(A_1 \rightarrow A_2)$ und $(A_1 \leftrightarrow A_2)$ Ausdrücke.
4. Wenn A ein Ausdruck und X eine Variable ist, so sind auch $\forall X A(X)$ und $\exists X A(X)$ Ausdrücke. A heißt Wirkungsbereich des Quantors von X.
5. Weitere Ausdrücke gibt es nicht.

Zusammenhang Ausdruck — Aussage

Ein Ausdruck ohne freie Variable heißt Aussage.

Vereinbarung zur Verkürzung der Notation von Ausdrücken

1. Außenklammern können weggelassen werden.
2. Die Stärke der Bindung von Quantoren und Junktoren ist wie folgt priorisiert: $\forall$, $\exists$ und $\neg$ binden am stärksten; danach folgen (in der angegebenen Reihenfolge) $\wedge$, $\vee$, $\rightarrow$ und $\leftrightarrow$.
3. Ketten von Konjunktionen oder Disjunktionen gelten als von links geklammert.

2.6 Allgemeingültigkeit, Kontradiktorizität und Äquivalenz von Aussagen

Eine Aussage A heißt **allgemeingültig** ($ag \quad A$), falls sie für jede Interpretation wahr ist.
Eine Aussage A heißt **kontradiktorisch** ($kt \quad A$), gdw. $ag \quad \neg A$.
Zwei Aussagen A_1 und A_2 heißen **äquivalent** ($A_1 \equiv A_2$), gdw. $ag(A_1 \leftrightarrow A_2)$.

Wichtige äquivalente Umformungen

(1)	$\neg\neg A$	$\equiv$	A
(2)	$\neg(A \wedge B)$	$\equiv$	$\neg A \vee \neg B$
(3)	$\neg(A \vee B)$	$\equiv$	$\neg A \wedge \neg B$
(4)	$\neg(A \to B)$	$\equiv$	$A \wedge \neg B$
(5)	$\neg(A \leftrightarrow B)$	$\equiv$	$(A \wedge \neg B) \vee (\neg A \wedge B)$
(6)	$\neg\forall X A(X)$	$\equiv$	$\exists X \neg A(X)$
(7)	$\neg\exists X A(X)$	$\equiv$	$\forall X \neg A(X)$
(8)	$\nabla X A(X) \circ B$	$\equiv$	$\nabla X(A(X) \circ B)$
		mit	$\nabla \in \{\forall, \exists\}$ und $\circ \in \{\wedge, \vee\}$
		und	wenn X nicht in B vorkommt
	Ausnahmen (von (8)):		
(8.1)	$\forall X A(X) \wedge \forall X B(X)$	$\equiv$	$\forall X(A(X) \wedge B(X))$
(8.2)	$\exists X A(X) \vee \exists X B(X)$	$\equiv$	$\exists X(A(X) \vee B(X))$
(9)	$A \to B$	$\equiv$	$\neg A \vee B$
(10)	$A \to \nabla X B(X)$	$\equiv$	$\nabla X(A \to B(X))$
(11)	$\forall X A(X) \to B$	$\equiv$	$\exists X(A(X) \to B)$
(12)	$\exists X A(X) \to B$	$\equiv$	$\forall X(A(X) \to B)$
(13)	$A \wedge (B \vee C)$	$\equiv$	$(A \wedge B) \vee (A \wedge C)$
(14)	$A \vee (B \wedge C)$	$\equiv$	$(A \vee B) \wedge (A \vee C)$
(15)	$A \wedge A$	$\equiv$	A
(16)	$A \wedge true$	$\equiv$	A
(17)	$A \wedge false$	$\equiv$	$false$
(18)	$A \vee A$	$\equiv$	A
(19)	$A \vee true$	$\equiv$	$true$
(20)	$A \vee false$	$\equiv$	A

2.7 Folgern

Sei M eine Menge von Aussagen, A eine Aussage. A **folgt aus** M ($M \models A$), falls jede Interpretation, die zugleich alle Elemente von M wahr macht (jedes Modell von M), auch A wahr macht.
Für endliche Mengen von Aussagen $M = \{A_1, \ldots, A_n\}$ bedeutet das:
$M \models A$, gdw.

$$ag(\bigwedge_{i=1}^{n} A_i \to A)$$

bzw. (was wegen $\neg(\bigwedge_{i=1}^{n} A_i \to A) \equiv \bigwedge_{i=1}^{n} A_i \wedge \neg A$ dasselbe ist)

$$kt(\bigwedge_{i=1}^{n} A_i \wedge \neg A).$$

2.8 HORN - Klauseln

HORN-Klauseln sind Ausdrücke der Form

$$\forall X_1 \ldots \forall X_n(\underbrace{A}_{Klauselkopf} \leftarrow \underbrace{\bigwedge_{i=1}^{m} A_i}_{Klauselkörper}),$$

wobei A und die A_i quantorfreie Atomformeln ohne freie Variablen sind.

Varianten / Spezialfälle

1. **Regeln** *(vollständige HORN-Klauseln)*

$$\forall X_1 \ldots \forall X_n(A(X_1, \ldots, X_n) \leftarrow \bigwedge_{i=1}^{m} A_i(X_1, \ldots, X_n))$$

2. **Fakten** *(HORN-Klauseln mit leerem Klauselkörper)*

$$\forall X_1 \ldots \forall X_n(A(X_1, \ldots, X_n) \leftarrow true)$$

3. **Fragen** *(HORN-Klauseln mit leerem Klauselkopf)*

$$\forall X_1 \ldots \forall X_n(false \leftarrow \bigwedge_{i=1}^{m} A_i(X_1, \ldots, X_n))$$

4. **leere HORN-Klauseln**

$$false \leftarrow true$$

2.9 Resolutionsmethode und deren Hintereinanderanwendung

Sei $\{K_1, \ldots, K_n\}$ eine Menge von Fakten und Regeln (kurz: Klauseln),

$$H \equiv \bigwedge_{i=1}^{m} H_i$$

eine Frage (eine Hypothese).
Eine der Klauseln K_j sei

$$A \leftarrow \bigwedge_{k=1}^{p} B_k \quad ,$$

wobei A und die B_k Atomformeln sind und alle auftretenden Variablen bezüglich der ganzen Klausel allquantifiziert sind.
Es gebe Termeinsetzungen ϑ_1 und ϑ_2 in die Variablen von A und eines der H_i (etwa H_l mit $1 \leq l \leq m$), so dass $\vartheta_1(A) \equiv \vartheta_2(H_l)$.

$$M \equiv \bigwedge_{i=1}^{n} K_i \wedge \neg H$$

ist kontradiktorisch ($kt \quad M$), wenn M nach Ersetzen von H in M durch

$$(\bigwedge_{i=1}^{l-1} \vartheta_2(H_i)) \wedge (\bigwedge_{k=1}^{p} \vartheta_1(B_k)) \wedge (\bigwedge_{i=l+1}^{m} \vartheta_2(H_i))$$

noch immer kontradiktorisch ist.

Satz von ROBINSON

Eine Klausel„menge“ M ist kontradiktorisch ($kt \quad M$), gdw. durch wiederholte Anwendung der Resolutionsmethode in endlich vielen Schritten die Hypothese durch die leere Klausel (Symbol: $\square$) ersetzt werden kann.

Unifikation *(=„Finden von Termeinsetzungen ϑ_1 und ϑ_2“)*

Zwei **Atomformeln** $p_1(t_{11}, \ldots, t_{1n})$ und $p_2(t_{21}, \ldots, t_{2m})$ **sind unifizierbar**, gdw. sie

- die gleichen Prädikatensymbole aufweisen ($p_1 \equiv p_2$),
- die gleiche Stelligkeit aufweisen ($n = m$) und
- die Terme t_{1i} und t_{2i} jeweils miteinander unifizierbar sind.

Die **Unifizierbarkeit zweier Terme** t_1 und t_2 richtet sich nach deren Sorte:

1. t_1 **und** t_2 **sind Konstanten (Individuensymbole)**:
   Die Unifikation ist erfolgreich, gdw. t_1 und t_2 identisch sind ($t_1 = t_2$).
2. $t_1 = f_1(t_{11}, \ldots, t_{1n})$ **und** $t_2 = f_2(t_{21}, \ldots, t_{2m})$ **sind strukturierte Terme**:
   Die Unifikation ist erfolgreich, wenn
   - die Funktionssymbole identisch sind ($f_1 = f_2$),
   - die Stelligkeiten gleich sind ($n = m$) und
   - die t_{1i} und t_{2i} jeweils miteinander unifizierbar sind.
3. t_1 **ist Variable und** t_2 **ist Konstante oder strukturierter Term**:
   Die Unifikation ist erfolgreich, wenn t_1 nicht in t_2 enthalten ist. t_1 wird durch t_2 ersetzt (t_1 wird *instantiiert*).
4. t_1 **und** t_2 **sind Variablen**:
   Die Unifikation ist erfolgreich. Die Variablen werden gleichgesetzt ($t_2 := t_1$ bzw. $t_1 := t_2$).

3 Logische Programmierung in PROLOG

3.1 Einordnung des logischen Programmierparadigmas

Programmierparadigmen:

- imperativ (prozedural)
  Beschreibung von Problemlösungsalgorithmen (WIE?)
  - „rein“ imperativ
    Beschreibung von auf Daten zugreifenden Algorithmen, z.B. in FORTRAN, ALGOL, PL/1, PASCAL, FORTH, C, MODULA-2, ...
  - objektorientiert
    Datenelemente und die sie verarbeitenden Algorithmen (Methoden) bilden ein Objekt einer Klassenhierarchie, z.B. in SMALLTALK, C++, ...
- deskriptiv (deklarativ)
  „kalkülisierte“ Beschreibung von Problemen (WAS?)
  - funktional, *z.B. in LISP*
  - **logisch,** *z.B. in PROLOG*

Problembeschreibung

Notation von Aussagen über das Problemgebiet (den Diskursbereich*) mit Hilfe prädikatenlogischer Ausdrücke*

Programmabarbeitung

- durch mustergesteuerte Prozeduraufrufe
  Anhand eines „aktuellen Zustands“ (eines Ziels*) wird eine Prozedur gesucht, die das Problem durch „kleinere“ Teilprobleme ersetzt. Es wird systematisch eine Folge solcher Ersetzungsschritte generiert, bei der das „leere“ Ziel entsteht.*
- bei Bedarf auch durch explizit vorzugebende Steuerinformation
  Dieses Suchverfahren kann zielgerichtet beeinflusst werden.

3.2 Syntax

Syntax von Klauseln

Sorte	Syntax	Beispiele
Fakt	$< praed\_symb > (< term >, \ldots, < term >).$	`liefert(xy_ag,motor,opel).`

Regel	$< praed\_symb > (< term >, \ldots, < term >) :-$
	$< praed\_symb > (< term >, \ldots, < term >),$
	$\ldots \ ,$
	$< praed\_symb > (< term >, \ldots, < term >).$
		`konkurrenten(Fa1,Fa2) :-`
		`liefert(Fa1,Prod,_),`
		`liefert(Fa2,Prod,_).`

Frage	$?- < praed\_symb > (< term >, \ldots, < term >),$ $\ldots \ ,$ $< praed\_symb > (< term >, \ldots, < term >).$	`?- konkurrenten(ibm,Wer),` `liefert(Wer,_,ibm).`

Syntax von Termen

Termsorte		Syntax	Beispiele
Konstante	Name	Zeichenfolge, beginnend mit Kleinbuchstaben, die Buchstaben, Ziffern und Unterstriche enthalten kann	`otto_1`, `rainer`
		beliebige Zeichenfolge, die in “ ... “ eingeschlossen ist	`\IST NAME\`, `\hofnarr@kanzleramt.de\`
		Zeichenfolge aus Sonderzeichen	`%&&$#`
	Zahl	Ziffernfolge, ggf. mit Vorzeichen, Dezimalpunkt und Exponentendarstellung	`3`, `-5`, `1001`, `3.6E-12`
Variable	allgemein	Zeichenfolge, beginnend mit einem Großbuchstaben oder einem Unterstrich, die Buchstaben, Ziffern und Unterstriche enthalten kann	`X`, `Eingabe`, `_alter`
	anonym	Unterstrich	`_`
strukturierter Term	allgemein	$< fkt\_symb > (< term >, \ldots, < term >)$	`hauptstadt(brd)`, `freund(frau(uwe))`
	Liste	leere Liste:	`[ ]`
		$[< term > \mid < restliste >]$	`[mueller|[mayer|_]]`
		$[< term >, < term >, \ldots, < term >]$	`[mueller, mayer, schulze]`

BACKUS-NAUR – Form

$$< PROLOG - Programm > \quad ::= \quad < Wissensbasis >< Hypothese >$$

$$
\begin{array}{rcl}
< Wissensbasis > & ::= & < Klausel > | < Klausel >< Wissensbasis > \\
< Klausel > & ::= & < Fakt > | < Regel > \\
< Fakt > & ::= & < Atomformel > . \\
< Atomformel > & ::= & < Praedikatensymbol > (< Termfolge >) \\
< Praedikatensymbol > & ::= & < Name > \\
< Name > & ::= & < Kleinbuchstabe > | < Kleinbuchstabe >< Restname > | \\
 & & \text{“} < Zeichenfolge > \text{“} | < Sonderzeichenfolge > \\
< Restname > & ::= & < Kleinbuchstabe > | < Ziffer > | \_ | \\
 & & < Kleinbuchstabe >< Restname > | \\
 & & < Ziffer >< Restname > | \_ < Restname > \\
< Zeichenfolge > & ::= & < Zeichen > | < Zeichen >< Zeichenfolge > \\
< Zeichen > & ::= & < Buchstabe > | < Ziffer > | < Sonderzeichen > \\
< Buchstabe > & ::= & < Kleinbuchstabe > | < Großbuchstabe > \\
< Kleinbuchstabe > & ::= & \mathbf{a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z} \\
< Großbuchstanbe > & ::= & \mathbf{A\ |B\ |C\ |D\ |E\ |F\ |G\ |H\ |I\ |J\ |K\ |L\ |M\ |N\ |O\ |P\ |Q\ |R\ |} \\
 & & \mathbf{S\ |T\ |U\ |V\ |W\ |X\ |Y\ |Z} \\
< Ziffer > & ::= & \mathbf{0\ |1\ |2\ |3\ |4\ |5\ |6\ |7\ |8\ |9} \\
< Sonderzeichen > & ::= & + \ |\text{-}\ |* \ |/ \ |\backslash|\hat{}\ |<|>|= \ |\text{'} \ |\tilde{}|: \ |. \ |\# \ |@ \ |\$ \ |\& \ |\% \ |\S \\
< Sonderzeichenfolge > & ::= & < Sonderzeichen > | \\
 & & < Sonderzeichen >< Sonderzeichenfolge > \\
< Termfolge > & ::= & \varepsilon | < Term > | < Term > , < Termfolge > \\
< Term > & ::= & < Konstante > | < Variable > | < strukturierterTerm > \\
< Konstante > & ::= & < Name > | < Zahl > \\
< Zahl > & ::= & < Ziffer > | < Ziffer >< Zahl > \\
< Variable > & ::= & < Großbuchstabe > | < Großbuchstabe >< Restvariable > | \_ | \\
 & & \_ < Restvariable > \\
< Restvariable > & ::= & < Buchstabe > | < Ziffer > | \_ | \\
 & & < Buchstabe >< Restvariable > | \\
 & & < Ziffer >< Restvariable > | \_ < Restvariable > \\
< strukturierterTerm > & ::= & < Funktionssymbol > (< Termfolge >) | < Liste > \\
< Funktionssymbol > & ::= & < Name > \\
< Liste > & ::= & [< Termfolge >] | [< Term > | < Liste >] \\
< Regel > & ::= & < Atomformel > \text{:-} < Atomformelfolge > . \\
< Atomformelfolge > & ::= & < Atomformel > | < Atomformel > , < Atomformelfolge > \\
< Hypothese > & ::= & \text{?-} < Atomformelfolge > .
\end{array}
$$

3.3 Prolog aus logischer Sicht

Die Menge aller zu einem Sachverhalt (einem Diskursbereich) formulierten Fakten und Regeln heißt **Wissensbasis**.

Prinzip

- Formulierung einer Menge $M = \{K_1, \ldots, K_n\}$ von Fakten und Regeln (kurz „Klauseln“), d.h. einer Wissensbasis
- Formulierung einer Hypothese (Frage, Ziel) $H \equiv H_1 \wedge \ldots \wedge H_n$ (die H_i heißen **Teilziele**)
- zu zeigen: $M \models H$, d.h. $kt(\bigwedge_{i=1}^{n} \wedge \neg H)$
- dies wird gezeigt durch die Resolutionsmethode und deren Hintereinanderanwendung nach ROBINSON

Suchalgorithmus für eine Folge von Resolutionsschritten

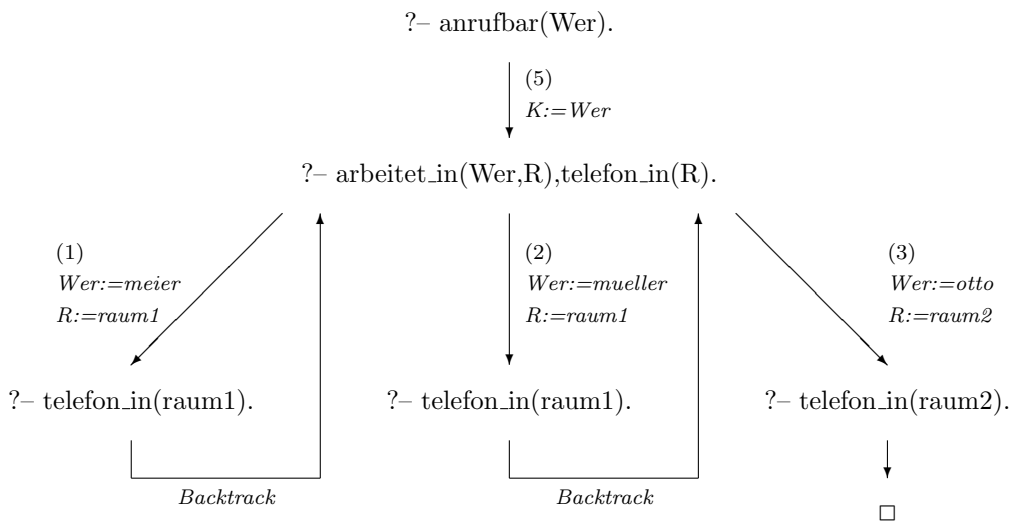
Tiefensuche mit Backtrack:

- Die Teilziele eines aktuellen Ziels werden **von links nach rechts** bearbeitet.
- Die Wissensbasis wird **von oben nach unten** nach einem mit dem 1. Teilziel unifizierbaren Klauselkopf durchsucht.
- Gibt es genau eine derartige Klausel, so wird das 1. Teilziel durch den Klauselkörper ersetzt (unter Einsetzen der Variablenersetzungen, d.h. des Unifikators).
- Gibt es mehrere derartige Klauseln, so wird ein sog. Entscheidungspunkt (place marker) eingerichtet, der momentane Bearbeitungszustand (akt. Ziel und Variablenbelegungen) im sog. Backtrack-Keller gekellert und zunächst die (in der Wissensbasis) am weitesten oben stehende Klausel angewandt, d.h. das 1. Teilziel durch deren Klauselkörper ersetzt.
- Gibt es keine derartige Klausel (mehr), so erfolgt ein Backtrack, d.h. der Algorithmus setzt zum letzten Entscheidungspunkt zurück und beginnt, ausgehend vom dort gekellerten Zustand, mit der nächsten alternativ anwendbaren Klausel.
- Das Verfahren bricht ab, wenn
  - als aktuelles Ziel die leere Klausel entsteht (Prolog's Antwort: „ja“ bzw. eine konstruktive Lösung) oder
  - das Ziel nicht leer ist, keine Klausel (mehr) anwendbar ist und kein Entscheidungspunkt (mehr) eingerichtet ist (Prolog's Antwort: „nein“).

Veranschaulichung des Suchalgorithmus' durch einen Suchbaum:
ODER-Baum der Abarbeitung, dessen Knoten das jeweils aktuelle Ziel symbolisieren und dessen gerichtete Kanten jeweils einen Resolutionsschritt symbolisieren. Die Kanten sind markiert

- mit der Nummer derjenigen Klausel, mit deren Kopf das 1. Teilziel unifiziert wurde und
- mit dem Unifikator, d.h. der Variablenersetzung.

Beispiel:
Wissensbasis:
(1) `arbeitet_in(meier,raum1).`
(2) `arbeitet_in(mueller,raum1).`
(3) `arbeitet_in(otto,raum2).`
(4) `telefon_in(raum2).`
(5) `anrufbar(K) :- arbeitet_in(K,R),telefon_in(R).`
Ziel: `?- anrufbar(Wer).`
Suchbaum:



Prolog's Antwort (konstruktive Lösung): „Wer := otto"

3.4 Prolog aus prozeduraler Sicht

Eine **Prozedur** ist die Menge aller Klauseln mit gleichem Kopfprädikat in einer Wissensbasis.

deklarative Interpretation	prozedurale Interpretation
Prädikat	Prozedur
Ziel	Prozeduraufruf
Klauselkörper	Unterprozeduren
Klauseln mit gleichem Kopfprädikat	Prozedurvarianten
Klauselkopf	Prozedurkopf
Klauselkörper	Prozedurrumpf

„Prozedurales Mitdenken" bei der Formulierung von Wissensbasen ist u.U. notwendig, um

- eine Reihenfolge konstruktiver Lösungen zu erzwingen,
- nicht terminierende (aber trotzdem logisch korrekt formulierte) Programme zu vermeiden,
- seiteneffektbehaftete Prädikate sinnvoll einsetzen zu können,
- (laufzeit-) effizienter zu programmieren und
- Möglichkeiten der gezielten Manipulation des Suchalgorithmus' nutzen zu können.

Steuerung der Abarbeitung

! (cut)

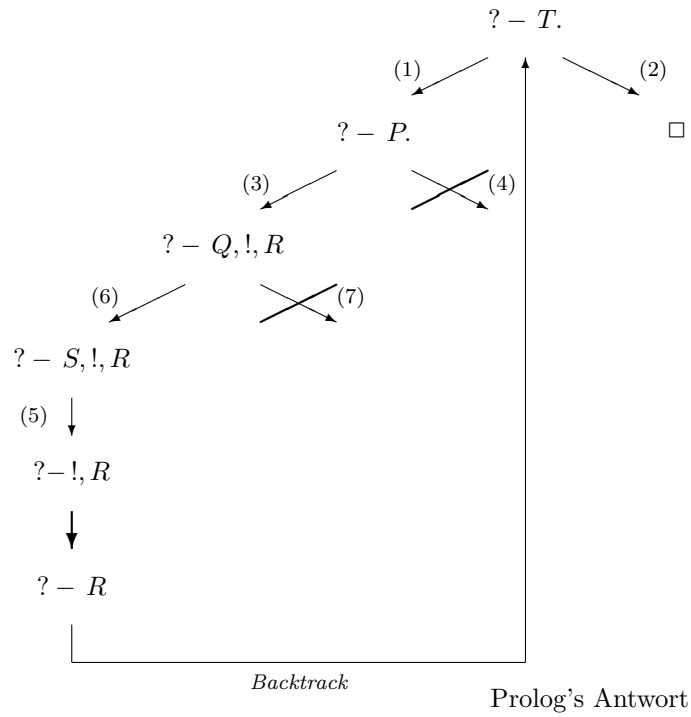
Das Prädikat „`!/0`" (cut) ist stets wahr. In Klauselkörpern als Teilziel eingefügt, verhindert es ein Backtrack der hinter dem „`!`" stehenden Teilziele zu den vor dem „`!`" stehenden Teilzielen sowie zu alternativen Klauseln mit gleichem Kopfprädikat. „!" schneidet die noch verbleibenden alternativen Lösungswege innerhalb der Prozedur, in der es vorkommt, ab.

Beispiel

Wissensbasis: (P, Q, R, S, T - symbolische Atomformeln)

(1) $T :- P.$
(2) $T.$
(3) $P :- Q, !, R.$
(4) $P :- S.$
(5) $S.$

(6) $Q :- S.$
(7) $Q.$
Ziel: $? - T.$
Suchbaum:



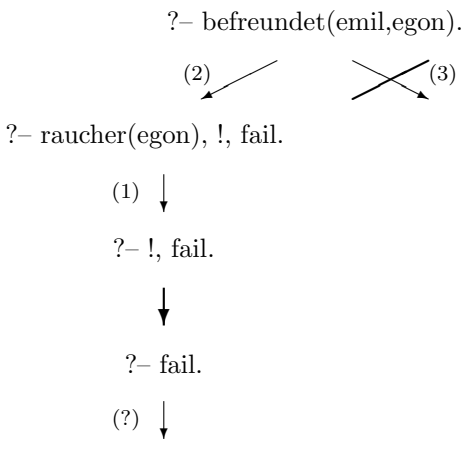
Prolog's Antwort: „ja"

fail

Das Prädikat „`fail/0`" ist stets falsch. In Klauselkörpern eingefügt, löst es ein Backtrack aus bzw. führt zum Mißerfolg (Antwort: „nein"), falls kein Backtrack mehr möglich ist.

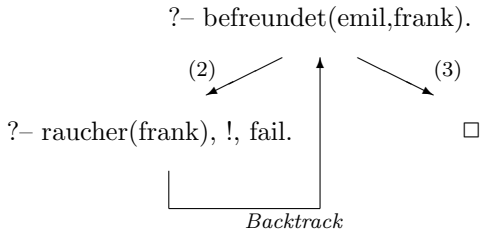
Beispiel

Wissensbasis:
(1) `raucher(egon).`
(2) `befreundet(emil,X) :- raucher(X),!,fail.`
(3) `befreundet(emil,_).`
Ziel: `?- befreundet(emil,egon).`
Suchbaum:



Prolog's Antwort: „nein"

Ziel: `?- befreundet(emil,frank).`
Suchbaum:



Prolog's Antwort: „ja“

Die sog. „cut-fail-Kombination“ verwendet man typischerweise, wenn man diejenigen Prämissen gut formulieren kann, für die ein Prädikat auf **falsch** abbilden soll.

3.5 Listen und Rekursion

1. [ ] ist eine (leere) Liste.
2. Wenn T ein Term und L eine Liste oder eine Variable ist, dann ist
   (a) $[T|L]$ eine Liste.
   (b) $T.L$ eine Liste.
   (c) $.(T, L)$ eine Liste.

   T heißt **Listenkopf** und L heißt **Listenkörper**. [2]
3. Wenn $t_1, \ldots, t_n$ Terme sind, so ist $[t_1, \ldots, t_n]$ eine Liste.
4. Weitere Listen gibt es nicht.

Eine Prozedur heißt **rekursiv**, wenn in mindestens einem Körper der zugehörigen Regeln ein erneuter Aufruf der betreffenden Prozedur steht.
Ist der Selbstaufruf die letzte Atomformel des Klauselkörpers der letzten Klausel der Prozedur (bzw. wird sie es durch vorheriges „Abschneiden“ nachfolgender Klauseln mit „`!/0`“), so handelt es sich um **Rechtsrekursion**, anderenfalls um **Linksrekursion**. Rechtsrekursion ist gegenüber der Linksrekursion bei der Abarbeitung i.allg. speicherplatz- und laufzeiteffizienter.
Eine **Differenzliste** besteht aus zwei Listen L_1 und L_2 und wird i.allg. als $[L_1, L_2]$ oder (bei Definition eines infix-Operators „-“/2) $L_1 - L_2$ notiert. Sie wird (vom Programmierer, nicht vom PROLOG-System !) als **eine** Liste interpretiert, deren Elemente sich aus denen von L_1 abzüglich der Elemente von L_2 ergeben. Differenzlisten verwendet man typischerweise, wenn häufig Operationen am Ende von Listen vorzunehmen sind.

3.6 Prolog-Fallen

3.6.1 Nicht terminierende Programme

Alternierende Zielklauseln
Ein aktuelles Ziel wiederholt sich infolge (meist indirekter) Rekursion, d.h. es kommt im Suchbaum erneut vor, wodurch der Suchbaum sich selbst als Unterbaum enthält (und folglich unendliche Tiefe hat).

Expandierende Zielklauseln
Im aktuellen Ziel wird das erste Teilziel infolge direkter Rekursion durch sich selbst und weitere neue Teilziele bei der Resolution ersetzt, wodurch es immer „länger“ (und folglich nie leer) wird.

Beide Fälle sind oft mit einem Überlaufen des Backtrack-Kellers verbunden und brechen infolge endlichen Hauptspeichers mit einer entsprechenden Fehlermeldung ab.

[2] Die Notationen 2.2. und 2.3. sind ungebräuchlich, zeigen aber den Charakter der Liste als Spezialfall des strukturierten Terms.

3.6.2 Verwendung von `not/1` und anderer Metaprädikate

Die Verwendung von Variablen beim Aufruf von Metaprädikaten führt häufig nicht zu konstruktiven Lösungen bzw. zu falschen Antworten des Prolog-Systems.

4 Wissensbasierte Systeme

4.1 Eigenschaften Wissensbasierter Systeme

1. Wissensbasierte Systeme sind in der Lage, **Darstellungen**(!) **des Wissens in symbolischer Form** getrennt von Wissensverarbeitungskomponenten **aufzubewahren**.

   > Eine scharfe geistige Trennung zwischen Wissen an sich und formalen Systemen zur Darstellung und Verarbeitung desselben ist unabdingbar für das Verständnis der Grundkonzepte der Wissensverarbeitung. Diese Trennung ist der erste Schritt zur Demystifizierung des Begriffs „Künstliche Intelligenz“.

2. Wissensbasierte Systeme können aus der abgespeicherten Information Schlüsse und Konsequenzen ableiten, die in ihnen nicht explizit repräsentiert; ihnen aber inhärent sind. Die dazu angewandten Methoden heißen **Inferenzmethoden**.

   Typischerweise ist der Inferenz Wissensbasierter Systeme ein **Nichtdeterminismus** innewohnend.

3. Eine weitere Eigenschaft Wissensbasierter Systeme ist die **Erklärungsfähigkeit**, d.h. sie sollten in der Lage sein, auf Anforderung die Kette der Folgerungsschritte (resp. Resolutionsschritte) dem Nutzer transparent und plausibel zu machen. Die Akzeptanz solcher Systeme bei Nutzern hängt wesentlich von der Qualität der Erklärungsfähigkeit ab.

4. Eine aus der Sicht des Autors gleichfalls wichtige Eigenschaft Wissensbasierter Systeme ist die **Lernfähigkeit**. Dazu gehört z.B. die Fähigkeit, anhand von Nutzerinformationen (etwa über den Erfolg oder Mißerfolg vergangener Sitzungen) neue Wissensinhalte abzuleiten.

4.2 Architektur Wissensbasierter Systeme

Die Abbildung 1 zeigt die Umsetzung der o.g. Eigenschaften durch eine Struktur und das (andeutungsweise) funktionelle Zusammenwirken der Komponenten.

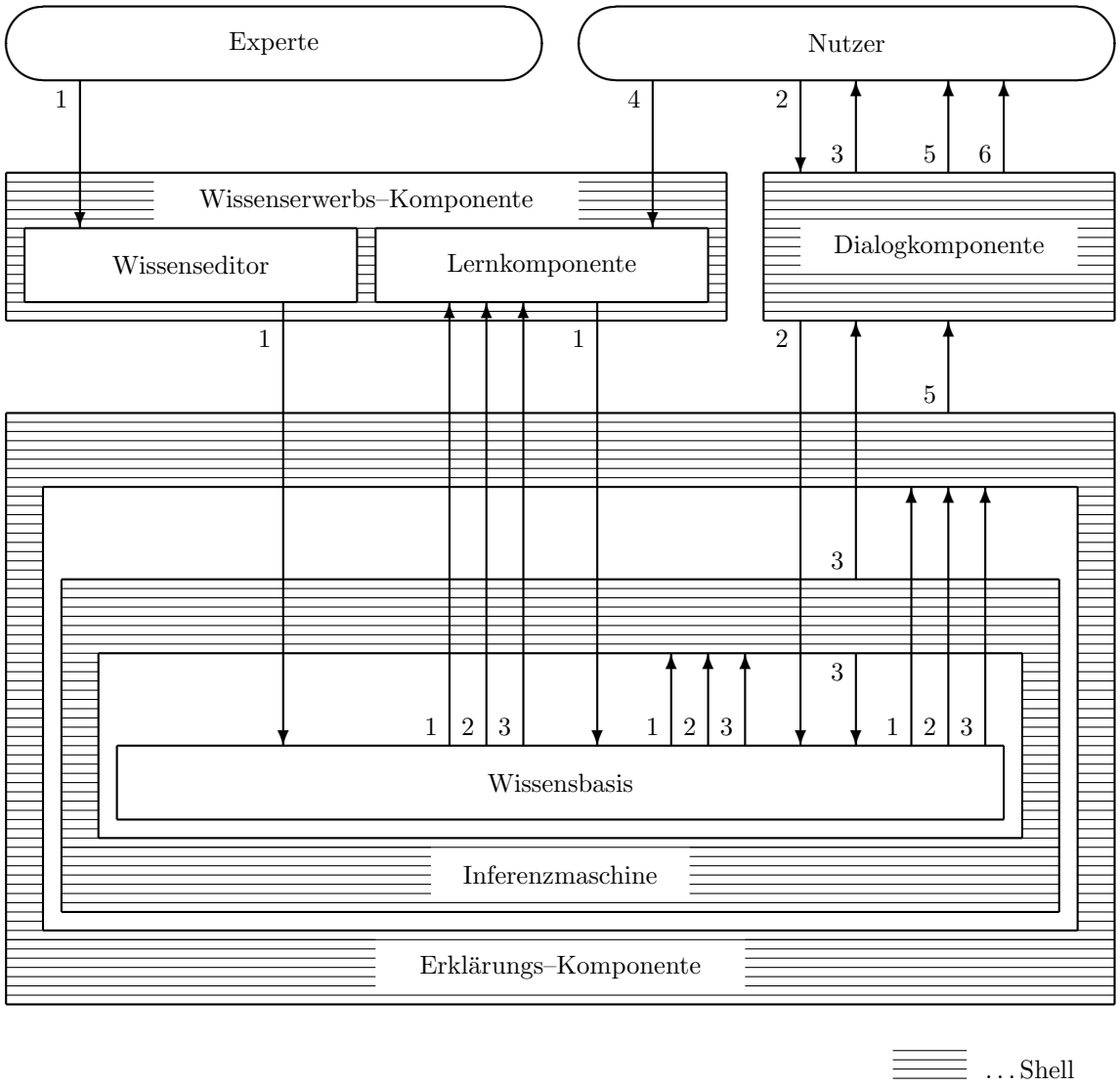
5 Wissensdarstellungen der Künstlichen Intelligenz

Typen des darzustellenden Wissens

- bereichsspezifisches Expertenwissen (welches sich während der Konsultation mit dem Nutzer nicht ändert),
- fallspezifisches Faktenwissen (welches der Nutzer während der Konsultation eingibt) und
- Zwischen– und Endergebnisse (die das System während der Konsultation ableitet).

Ebenen des darzustellenden Wissens

1. die Darstellung auf Nutzerebene (problemorientierte Darstellung),
2. die Darstellung in der (den) Programmiersprache(n), in der das Wissensbasierte System implementiert ist (implementierungsorientierte Darstellung) und
3. die computerinterne Darstellung (Bytes).



1 . . . bereichsspezifisches Expertenwissen
2 . . . fallspezifisches Faktenwissen
3 . . . Zwischen– und Endergebnisse
4 . . . Erfolgsbewertung des Nutzers
5 . . . Erklärungs–Information
6 . . . Bedien–Information (Hilfe–Komponente)

Abbildung 1: Architektur Wissensbasierter Systeme

Formen der problemorientierten Wissensdarstellung

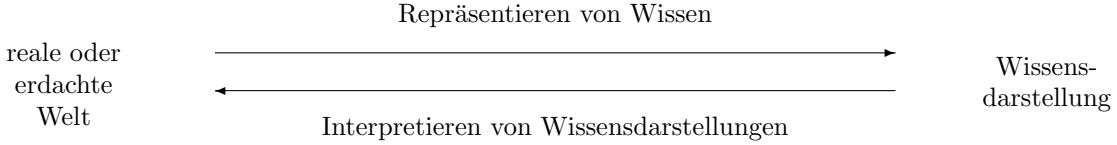
- Semantische Netze,
- Frames,
- der Prädikatenkalkül der ersten Stufe (PK1),
- Produktionssysteme und
- prozedurale Repräsentationen.

Methoden der problemorientierten Wissensdarstellung

Die **deklarativen Repräsentationsideen** basieren auf der Annahme, dass die Darstellung von Wissen unabhängig davon betrachtet werden kann, wie es verarbeitet wird. Es wird eine saubere Trennung zwischen Darstellung und Verarbeitung von Wissen vorgenommen.
Prozedurale Wissensrepräsentationen betonen den Verarbeitungsaspekt. Hierzu wird dem zu repräsentierenden Wissen zusätzlich Wissen über die Verarbeitung dieses Wissens (Metawissen) (explizit oder implizit) hinzugefügt und damit die o.g. scharfe Trennung „verwaschen“ bzw. ganz aufgehoben.

5.1 Prädikatenkalkül der ersten Stufe

Die Theorie des PK1 als Wissensdarstellungskalkül aus der Sicht der Wissensverarbeitung



Repräsentation:
Eine **Repräsentation** im PK1 ist eine eindeutige Zuordnung von Prädikaten–, Funktions– und Individuensymbolen zu Prädikaten, Funktionen und Objekten über einer Objektmenge U.
Interpretation:
Eine **Interpretation** des PK1 besteht aus einer Objektmenge U zusammen mit einer Abbildung $\mathcal{I}$, welche jedem n-stelligen Funktionssymbol eine n-stellige Funktion und jedem n-stelligen Prädikatensymbol eine n-stellige Relation über U zuweist. Eine solche Interpretation heißt auch „Interpretation $\mathcal{I}$ in U“.
Zusammenhang Prädikat – Relation:
Ein n-Tupel von Elementen aus U gehört zu der einem Prädikatensymbol p zugewiesenen Relation R, gdw. das dem Prädikatensymbol p zugeordnete Prädikat dieses n-Tupel auf *wahr* abbildet.
Sortenlogik:

1. Variablen und Konstanten haben eine Sorte S, die als Index mitgeführt wird, z.B. X_S, c_S usw.
2. Jedem n-stelligen Funktionssymbol f ist ein $(n+1)$-Tupel von Sorten $[S_1, \ldots, S_n, S_{n+1}]$ zugeordnet. Es werden nur Terme der Gestalt $f(t_1, \ldots, t_n)$ zugelassen, bei denen t_i von der Sorte S_i ist; die Sorte des gesamten strukturierten Terms $f(t_1, \ldots, t_n)$ ist dann S_{n+1}.
3. Jedem n-stelligen Prädikat p ist ein n-Tupel $[S_1, \ldots, S_n]$ von Sorten zugeordnet und es sind nur Atomformeln $p(t_1, \ldots, t_n)$ zugelassen, bei denen t_i von der Sorte S_i ist.

Semantisch gesehen wird einer Sorte S für jede Interpretation $\mathcal{I}$ eine Teilmenge des Universums U zugeordnet: $\mathcal{I}(S) \subseteq U$. Bei der Verarbeitung sortenbehafteter Prädikate und Terme müssen die vorgenommenen Substitutionen sortentreu sein.

Die Praxis der Logischen Programmierung

(Nutz-)Effekte der Beschränkung auf HORN–Klauseln:

1. Über HORN–Klauseln gibt es ein korrektes und vollständiges Ableitungsverfahren, d.h.

   $$\{K_1, \ldots, K_n\} \models H, \text{ gdw.} \{K_1, \ldots, K_n\} \vdash_* H$$

   wobei $\vdash_*$ (hier) die Ableitbarkeit nach ROBINSON durch Hintereinanderanwendung der Resolutionsmethode bedeutet.

2. O.g. Ableitungsverfahren ist algorithmisierbar.

3. In einer (PROLOG–) Wissensbasis aus HORN–Klauseln lassen sich keine Widersprüche formulieren, d.h. es gilt für jede PROLOG–Wissensbasis

   $$\bigwedge_{i=1}^{n} K_i \neq false$$

5.2 Semantische Netze

In Abbildung 2 sind 2-stellige Relationen über einer Menge von Personen in Form eines Semantischen Netzes beispielhaft illustriert.

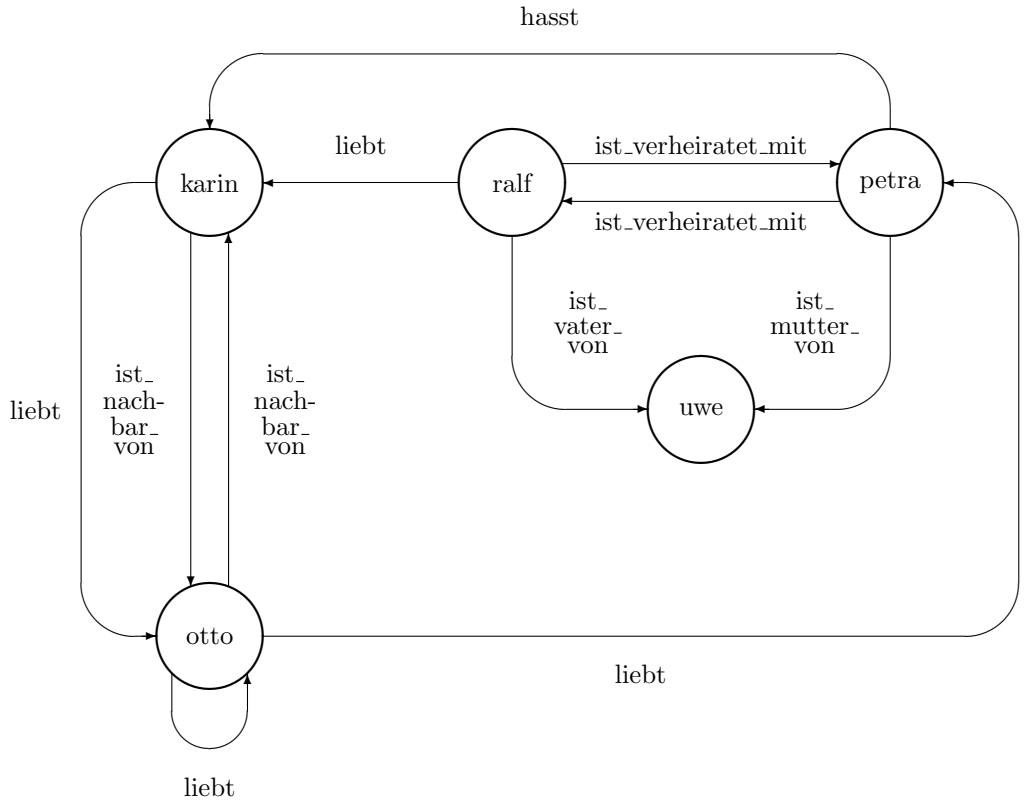


Abbildung 2: Ein Semantisches Netz

5.3 Frames

Eine allgemeine Notationsform eines Frames zeigt die Abbildung 3.
Begriffe und Eigenschaften:

Framename		
Oberkonzepte		
Unterkonzepte		
Slotname 1:	Slotwert 1	
	Merkmal 1_1:	Ausprägung 1_1
	...	
	Merkmal 1_{n_1}:	Ausprägung 1_{n_1}
...		
...		
Slotname m:	Slotwert m	
	Merkmal m_1:	Ausprägung m_1
	...	
	Merkmal m_{n_m}:	Ausprägung m_{n_m}

Abbildung 3: Notationsform eines Frames

- Wenn die Slots eines Frames (teilweise) gefüllt sind, so spricht man von einer **Instanz** eines Frames.
- **Defaultwerte** eines Slots sind vorläufige Werte, die in Ermangelung besseren Wissens als „Arbeitshypothese“ gewählt werden. Sie werden i.allg. durch das Voranstellen von *default* gekennzeichnet und können ohne weiteres „überschrieben“ werden.
- Von **generische**n **Werte**n spricht man, wenn Slotinhalte für alle Instanzen unveränderlich sind.
- Durch **Slotbedingungen** (**Constraints**) werden die Wertebereiche von Slots eingeschränkt.
- In einem Slot können auch Regelsysteme stehen, die auf Anforderung in Aktion treten, etwa um einen Slotwert zu berechnen.
- Auch „werteberechnende“ Prozeduren können Slotinhalt sein.

  Eine Stelle, welche die Bedingungen prüft, unter denen eine solche Prozedur ausgeführt wird, heißt **Dämon**. Ein Dämon bildet demnach die Schnittstelle zwischen deklarativen und prozeduralen Elementen der Wissensrepräsentation „Frame“.

  Solche prozeduralen Zusätze können in mehreren Formen auftreten, z.B. als

  - *if–needed* – Prozeduren (wenn–nötig – Prozeduren), die den Slotwert nicht automatisch, sondern nur auf Anforderung berechnen, was bei aufwendigen Berechnungen sinnvoll erscheint, oder als
  - *if–added* – Prozeduren (wenn–belegt – Prozeduren), die von einem Dämon genau dann angestoßen werden, wenn ein bestimmter Slot einen bestimmten Wert erhalten hat.
- Durch die Möglichkeit der Unterbrigung (wiederum) ganzer Frames in einem Slot sind beliebig tief strukturierte Slotinhalte denkbar.

Mit Hilfe solcher Frames kann man nun Wissen über Konzepte einer (realen oder erdachten) Welt notieren. Beispiele hierfür zeigt die Abbildung 4.

Personenkraftwagen		
Oberkonzepte: *Kraftfahrzeuge*		
Unterkonzepte: *Limousine, Caravan, Transporter*		
Motor:	*kraftstoffgetrieben*	
	Kraftstoff:	*default Benzin*
	Zylinder:	*>= 2*
	Hubraum:	*> 500 cm^3*
Getriebe:	*default Schaltgetriebe*	
	Gänge:	*default 5*
Räder:	*4*	

Limousine	
Oberkonzepte: *Personenkraftwagen*	
Unterkonzepte: *Kleinwagen, Mittelklassewagen, Wagen gehobener Klasse*	
Gesamtmasse:	*< 2 t*
Sitzplätze:	*default 5*

Abbildung 4: Beispiele für Frames

5.4 Objektorientierte Darstellungen

Grundgedanken:

1. Um komplexe Strukturen (Software, Wissen, ...) beherrschbar zu halten, sollten sie **modularisiert** werden. Die dabei entstehenden Moduln nennt man dann schlicht **Objekte**, an die man gewisse Anfragen (Botschaften) stellen (senden) kann, die ggf. beantwortet werden. Die bessere Beherrschbarkeit ergibt sich durch

   - die Erhöhung der Übersichtlichkeit mit der Blockbildung,
   - eine gewisse „Klarheit“ durch das Verbergen unwichtiger Details in den Objekten und
   - die Anwendungsunabhängigkeit durch die Möglichkeit der Datenabstraktion.

2. Eine gewisse „Kompaktheit“ der Repräsentation (von Daten, Wissen, ...) erreicht man durch die Zusammenfassung von Objekten mit gemeinsamen Eigenschaften zu Klassen. Da diese Gemeinsamkeiten innerhalb einer Klasse „vererbt“ werden können, brauchen sie nur einmal notiert werden. Dieser Vererbungsbegriff ist derselbe, wie der aus der Frame–Hierarchie bekannte.

Ein **Objekt** besteht aus lokalen Daten und einer Menge von (auf diese Daten zugreifenden) Operationen, die auch **Methoden** genannt werden. Andere als die durch die Methoden bereitgestellten Operationen können mit den Daten nicht ausgeführt werden.
Objekte kommunizieren über Botschaften (bzw. Nachrichten). Eine auszusendende Nachricht besteht i.allg. aus 3 Teilen:

- Name des Empfänger–Objektes,
- Name der auszuführenden Operation und
- eine Liste von Argumenten (die gleichfalls Objekte sind), mit denen die Operation ausgeführt werden soll.

Wenn ein Empfänger eine solche Botschaft bekommt, so muss er

1. prüfen, ob die auszuführende Operation von ihm auch bereitgestellt wird,
2. die Operation anstoßen und
3. eine Antwort an den Sender zurückmelden.

Die dadurch verursachten Effekte sind zweierlei Art:

1. lokale Effekte:
   Die Operation greift auf lokale Daten zu und verändert diese gegebenenfalls.
2. globale Effekte:
   Die Operation bewirkt, dass Botschaften an andere Objekte geschickt werden, was deren Zustand gegebenenfalls verändert.

Objekte mit gleichen Operationen und gleich strukturierten Daten werden zusammengefasst; ihre Gemeinsamkeiten werden einmalig durch eine **Klasse** definiert. Die einzelnen Objekte heißen dann **Instanzen** dieser Klasse. Die Klassen selbst sind gleichfalls Objekte.
Auch innerhalb von Klassen gibt es hierarchische Beziehungen, damit bestehende Gemeinsamkeiten kompakt notiert werden können:

> Die Klasse B ist eine Spezialisierung der Klasse A (oder A ist Oberklasse von B), wenn
>
> 1. die Klasse B zusätzliche Datenstrukturen gegenüber A aufweist oder
> 2. B zusätzliche Operationen gegenüber A aufweist,
>
> und ansonsten isomorph ist. B erbt alle Operationen und Datenstrukturen aus A. Hierbei ist es zulässig, dass gleichnamige Operationen in B anders implementiert sind als in A.

Die Ober–Unterklassen – Relationen zwischen Objekten können verschieden starken Resriktionen unterliegen:

1. In einer „strengen“ Variante wird von der Hierarchie Baumförmigkeit gefordert, d.h. jedes Objekt (mit Ausnahme der Wurzel) hat genau eine Oberklasse.

   Dies ist z.B. die Vererbungsform des „Klassikers“ objektorientierter Sprachen, SMALLTALK 80.

   Die durch diese Restriktion erzielte Effizienz erkauft man sich durch eine gewisse Schwerfälligkeit bei horizontalen Transfers in der Hierarchie. Hierzu muss man sich i.allg. „globaler Daten“ bedienen, die dem Konzept der Objektorientierung widersprechen.

2. In einer liberaleren Variante, die auch **Mehrfachvererbung** oder **multiple Vererbung** genannt wird, kann ein Objekt mehrere Oberklassen haben.

   Dies ist besonders dann von Interesse, wenn man für ein Objekt Eigenschaften verschiedener „Herkunft“ repräsentieren möchte. So kann man z.B.

   - als Oberklassen von „Licht“ sowohl „Welle“ als auch „Teilchen“ zulassen und
   - als Oberklassen von „Weizen“ sowohl „Pflanze“ als auch „Nahrungsmittel“ zulassen.

Eine in jedem Falle geforderte Bedingung ist, dass kein Objekt direkte oder indirekte Oberklasse seiner selbst sind darf. Eine graphische Darstellung der Klassenhierarchie muss demnach einen azyklischen Graphen ergeben.

5.5 Produktionssysteme

Theoretischen Grundlagen:

- HORN – Klauseln des Aussagenkalküls für die Wissensdarstellung und
- Schlussregeln für die Inferenz, wobei in allen bekannten Systemen als einzige Schlussregel der „modus ponens“ nach ARISTOTELES

  $$\frac{A \quad A \to B}{B}$$

  Anwendung findet.

Komponenten und deren Zusammenwirken:

1. eine Faktenbasis, d.h. Menge einfacher Aussagen
2. eine Menge von Produktionsregeln der o.g. Form und
3. eine Steuerkomponente, die **zyklisch**
   (a) alle auf die aktuelle Faktenbasis anwendbaren Regeln (die **Konfliktmenge**) bestimmt (sämtliche Bedingungen dieser Regeln müssen in der Faktenbasis als wahre Aussagen verzeichnet sein),
   (b) nach einer festgelegten Strategie (z.B. prioritätengesteuert) aus der Konfliktmenge eine Regel auswählt, d.h. eine **Konfliktlösung** vornimmt und
   (c) den Aktionsteil der gewählten Regel ausführt, der i.allg. darin besteht, die Faktenbasis zu verändern.

5.6 Prozedurale Repräsentationen aus dem Blickwinkel der Wissensverarbeitung

Bei dieser Repräsentationsform besteht das Wissen allein aus Prozeduren, die eine nicht explizit repräsentierte Faktenmenge, nämlich die Daten, manipuliert. Die Reihenfolge der Aufrufe der Prozeduren muss für das konkrete Problem „vorgedacht“ sein. Alle Entscheidungsprozesse, die während der Problemlösung auftreten, müssen bei der Formulierung der Prozeduren vorgesehen werden. Es gibt praktisch keine nichtdeterminierten Prozeduren, d.h. für jede Situation gibt es genau einen Lösungsweg (und nicht etwa mehrere alternativ anwendbare Prozeduren) für ein und denselben Aufruf. Den Gewinn an Effizienz erkauft man sich durch einen Verlust an universeller Verwendbarkeit.

5.7 Transformation der problemorientierten Wissensdarstellung auf die Maschinenebene

Ein „Weg“ des Wissens von der (dem „Expertenhirn“ entlockten) problemorientierten Ebene bis zur computerinternen Darstellung wird durch die Abbildung 5 illustriert.

6 Literaturhinweise

Logische Grundlagen (Prädikatenkalkül)

Asser, G. *Einführung in die mathematische Logik. Teil 2: Prädikatenkalkül der ersten Stufe.* Leipzig: Teubner Verlagsgesellschaft, 1982

Burris, S.N. *Logic for Mathematics and Computer Science* New Jersey: Prentice Hall, 1998

Bibel, W. *Deduktion – Automatisierung der Logik.* München: Oldenbourg, 1992

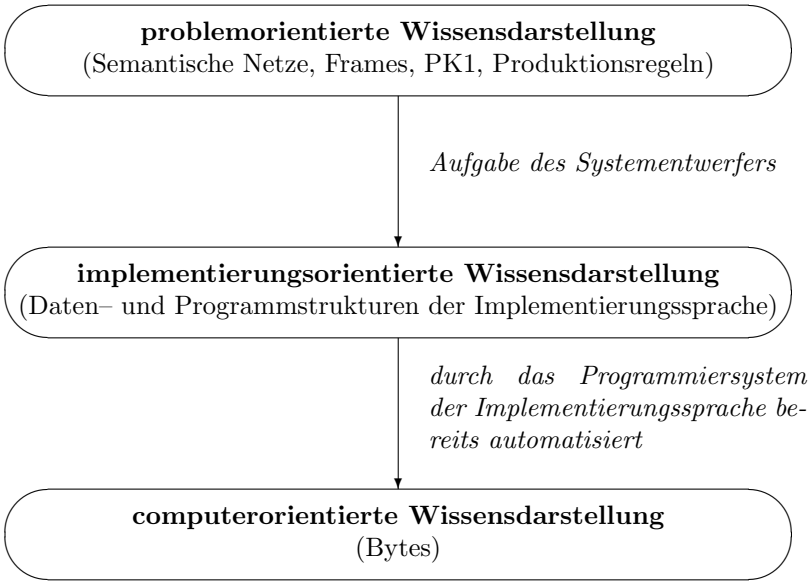


Abbildung 5: Transformationen zwischen den Ebenen der Wissensdarstellung

Gabbay, D. *Elementary Logics: A Procedural Perspective* London u.a.: Prentice Hall Europe, 1998

Genesereth, M.R.; Nilsson, N.J. *Logische Grundlagen der Künstlichen Intelligenz.* Braunschweig, Wiesbaden: Vieweg, 1989

Schmitt, P.H. *Theorie der Logischen Programmierung.* Berlin u.a.: Springer, 1992

Lehrbücher, Überblick über KI–Gebiete

Bratko, I. *Prolog: Programming for Artificial Intelligence* Addison Wesley, 2001

Clocksin, W.F.; Mellish, C.S. *Programming in Prolog.* Berlin, Heidelberg, New York: Springer, 1981,1987

Hofstadter, D.R. *Gödel, Escher, Bach: ein Endloses Geflochtenes Band* Stuttgart: Klett–Cotta, 1989

Kleine Büning, H.; Schmitgen, S. *Prolog.* Stuttgart:B.G.Teubner, 1986

Knauf, R. *Logische Programmierung und Wissensbasierte Systeme – eine Einführung.* Aachen: Shaker-Verlag, ISBN 3-86111-310-4, 1993

Luger, G.F. *Künstliche Intelligenz: Strategien zur Lösung komplexer Probleme.* München: Pearson, 2001

Richter, M.M. *Prinzipien der Künstlichen Intelligenz: Wissensrepräsentation, Inferenz und Expertensysteme* Stuttgart: B.G.Teubner, 1992

Schefe, P. *Künstliche Intelligenz: Überblick und Grundlagen* Mannheim: Bibil. Inst., 1991

Produkte und Anwendungen

Antoniou, G. *Turbo Prolog.* Düsseldorf: Data Becker, 1987

Böhringer, B.; Chiopris, C.; Futo, I. *Wissensbasierte Systeme mit Prolog.* Bonn: Addison-Wesley, 1988

Bradbury, A.; Woodward, R. *Turbo Prolog-Begleitbuch.* Maidenhead: McGraw-Hill, 1990

Dietrich, W. *Turbo Prolog.* Bonn: Addison-Wesley, 1988

Gabriel, R. *Wissensbasierte Systeme in der betrieblichen Praxis.* Maidenhead: McGraw-Hill, 1991 (Prolog für Wirtschaftsinformatik)

Grothaus, H.; Gust, H. *Turbo Prolog.* Würzburg: Vogel-Buchverlag, 1987

Hanus, H. *Problemlösen mit Prolog.* Stuttgart: B.G. Teubner, 1987

Janson *Die Programmiersprache Turbo Prolog.* München:Franzis, 1988

Kinnebrok, W. *Handbuch Turbo Prolog.* München: Oldenbourg, 1990

Kinnebrok, W. *Professionelles Programmieren mit Turbo Prolog.* München: Oldenbourg, 1988

Kinnebrok, W. *Turbo Prolog.* München: Oldenbourg, 1990

Lehner, Ch. *Prolog und Linguistik.* München: Oldenbourg, 1992

Schildt, M. *Professioneles Turbo Prolog.* Hamburg: McGraw-Hill, 1988

Schnupp, P. *Prolog - Einführung in die Programmierpraxis.* (mit IF/Prolog) München, Wien: Hanser, 1986

Schnupp, P.; Höß, K. *TerminalBuch Prolog.* München: Oldenbourg, 1989 (u.a. für Analyse u. Generierung formaler Sprachen)

Weiskamp, K.; Hengl, T. *KI-Programmierung mit Turbo Prolog.* Maidenhead: McGraw-Hill, 1989