Theoretische Grundlagen der Wissensverarbeitung

Dr. habil. Udo Hebisch
Institut für Informatik
TU Clausthal [1]

9. November 1992

[1] In LaTeX gesetzt von Arno Hildebrand

Vorwort

Dieses Skript wurde von mir für all diejenigen in LaTeX gesetzt[1] , denen es, aus welchen Gründen auch immer, nicht möglich war, die Vorlesung *Theoretische Grundlagen der Wissensverarbeitung* zu hören. Ich hoffe, daß damit allen Lesern die Möglichkeit gegeben wird, der Vorlesung *Wissensbasierte Systeme* zu folgen!
Ich möchte mich an dieser Stelle recht herzlich bei Dr. habil. Udo Hebisch bedanken, der so freundlich war, dieses Skript Korrektur zu lesen, so daß sich hoffentlich wenige inhaltliche Fehler eingeschlichen haben. Rechtschreibfehler möge der Leser bitte übersehen, da viele Teile dieses Skriptes zu später Stunde von mir gesetzt wurden!
Für Hinweise auf Fehler innerhalb dieses Skriptes wäre ich sehr dankbar, damit diese in neueren Auflagen entfernt werden können! Allen Lesern wünsche ich auf diesem Wege viel Erfolg bei der Arbeit mit diesem Skript und hoffe, daß es beim Hören der folgenden Vorlesungen seinen Beitrag zum Verständnis leistet.
Im Wintersemester 1992/93 wurde diese Vorlesung dann zum zweiten Mal gehalten. In der Zwischenzeit waren einige Fehler im Skript an den Tag getreten, die in dieser zweiten Auflage[2] nun korrigiert wurden. Weiterhin wurde aus aktuellem Anlaß das neue Kapitel *Die Behandlung unsicheren Wissens* eingeschoben, wofür an anderen Stellen gekürzt wurde.[3] Dieses Kapitel befaßt sich u.a. auch mit den Grundlagen der Fuzzy-Logik, mit deren Inhalten sich in der letzten Zeit auch eine Projektgruppe des Instituts für Informatik auseinandersetzt.
Gedankt sei an dieser Stelle all denen, die durch ihre konstruktive Kritik an diesem Skript zu seiner Verbesserung beigetragen haben.

Arno Hildebrand

[1] im Zeitraum Oktober 1991 – Februar 1992
[2] im Zeitraum Oktober – November 1992
[3] Dieses Skript gibt einen Gesamtüberblick über beide Semester.

Inhaltsverzeichnis

Kapitel 1

Wissen und seine Repräsentation

1.1 Aussagen, Überzeugungen und Wissen

Definition 1.1

i) Eine *Aussage* ist ein Satz einer (formalen oder natürlichen) Sprache, von dem es sinnvoll ist zu sagen, ob er wahr oder falsch ist (in einem zu repräsentierenden Teil der realen Welt).

ii) *Überzeugungen* (eines Wissensträgers) sind die Aussagen, von denen der Wissensträger annimmt, daß sie in der realen Welt wahr sind.

iii) *Wissen* sind diejenigen Überzeugungen, die auch tatsächlich wahr sind.

Beispiel 1.2 Der zu repräsentierende Teil der Welt sei folgenderweise beschrieben:
Zwei Personen A und B benutzen einen Kühlschrank gemeinsam.
Ereignis E1: A stellt Milch in den Kühlschrank.
Ereignis E2: B holt Milch aus dem Kühlschrank und verbraucht sie.
Sätze über diese Welt in natürlicher Sprache sind dann etwa:

i) Es ist Milch im Kühlschrank. (Überzeugung von A !)

ii) Es ist keine Milch im Kühlschrank. (Überzeugung von B !)

iii) Der Kühlschrank ist leer. (Überzeugung von B !)

iv) Der Kühlschrank ist prim. (Syntaktisch korrekt, semantisch sinnlos, keine Aussage)

Wenn E2 vor E1 eintritt, dann ist i) Wissen von A, aber ii) kein Wissen von B. Ob iii) Wissen von B ist, können wir nur unter zusätzlichen Annahmen entscheiden, etwa einer "closed world assumption":
Was nicht aus der Repräsentation der Welt hervorgeht, ist auch nicht wahr in ihr.

Anmerkung 1.3

i) Das Problem, wie für eine Aussage entschieden werden kann, ob sie "tatsächlich wahr" ist, bleibt in Definition 1.1 offen. Es gibt auch formale Aussagen über die reale Welt, von denen (noch) nicht entschieden ist, ob sie wahr oder falsch sind, z.B. die große Fermatsche Vermutung (Überzeugung von Fermat!)
$$\forall n, x, y, z \in N (n \geq 3 \Rightarrow x^n + y^n \neq z^n).$$

ii) Wissensträger kann eine Person, eine Personengruppe oder sogar die ganze Menschheit sein.

1.2 Symbolische Wissensrepräsentation

Definition 1.4 Unter symbolischer *symbolischer Wissensrepräsentation* versteht man das Aufschreiben (oder Speichern) von Symbolen, den Repräsentationsstrukturen, die bzgl. einer *Interpretationsvorschrift* einem Ausschnitt der zu repräsentierenden Welt entsprechen.

Anmerkung 1.5

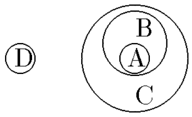
i) Eine Repräsentation erfaßt einige Merkmale der zu repräsentierenden Welt.

ii) Eine Repräsentation erfaßt nie alle Merkmale der repräsentierten Welt.

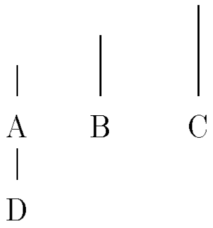
iii) Nicht alle Merkmale einer Repräsentation stehen zwangsläufig für Merkmale der repräsentierten Welt.

iv) Es gibt auch nichtsymbolische Repräsentationen von Wissen, z.B. durch Bilder, Landkarten, neuronale Netze. Diese werden aber im Rahmen dieser Vorlesung nicht betrachtet (vgl. [Hel91]).

Beispiel 1.6 Die zu repräsentierende Welt bestehe aus Kreisen unterschiedlicher Größe und Lage in der Ebene, etwa



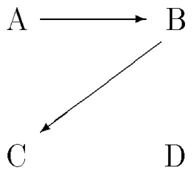
Die Repräsentation erfolge
a) durch Striche:



Interpretation: Strichlänge $\simeq$ Größe des repräsentierten Kreises
"Links von" $\simeq$ "ist enthalten in"

Die transitive Relation "ist enthalten in" für Kreise wird durch eine ebenfalls transitive Relation "Links von" für Striche repräsentiert.

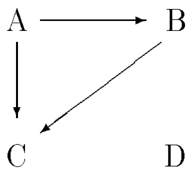
b) durch gerichtete Graphen:



Interpretation: Kantenfolge von x nach y $\simeq$ x ist enthalten in y

Die Größe der Kreise wird also nicht repräsentiert. Die Relation "es gibt eine Kantenfolge von x nach y" für Knoten ist ebenfalls transitiv.

c) durch gerichtete Graphen:



Interpretation: Kante von x nach y $\simeq$ x ist enthalten in y

Die Größe der Kreise wird auch hier nicht repräsentiert. Die transitive Relation "ist enthalten in" für Kreise wird durch eine nicht transitive Relation "es gibt eine Kante von x nach y" für Knoten repräsentiert.

d) durch gerichtete Graphen:

$$\begin{tikzcd} A \arrow[d, shift right] & C & D \\ B \arrow[u, shift right] & & \end{tikzcd}$$

Interpretation: Kante von x nach y $\simeq$ x hat gleiche Größe wie y

Die Lage der Kreise wird überhaupt nicht, ihre Größe nur unvollständig repräsentiert.
In keiner der Repräsentationen a) - d) wird die Form der Objekte, nämlich Kreise, dargestellt. Dagegen sagen die Längen der Kanten und Positionen der Knoten in b) - d) nichts über die repräsentierten Objekte aus.

Definition 1.7 Ein *Wissensrepräsentationsmodell* legt durch eine *Wissensrepräsentationssprache* (z.B. Prolog, OPS5, KL-One) die konstruierbaren Repräsentationsstrukturen (z.B. logische Formeln, Produktionsregeln, Netze) und die darauf ausführbaren Operationen (z.B. Hinzufügen u. Löschen, Beschriften von Kanten) fest.

Bemerkung 1.8
Da die natürliche Sprache wegen zu hoher Komplexität, hoher Redundanz und Mehrdeutigkeit als Wissensrepräsentationssprache ungeeignet ist, wird nach geeigneten formalen Sprachen gesucht. So kann etwa der Satz

Menschen haben Köpfe (im Gegensatz zu Computern) verschieden gedeutet werden:

Alle Menschen haben Köpfe. (So wie sie Beine haben?)
Alle Menschen haben einen Kopf. (So wie sie einen Heimatplaneten haben?)
Jeder Mensch hat einen eindeutig bestimmten Kopf, der nur ihm gehört.

Präziser, aber auch umständlicher geschieht dies durch eine formale Sprache:

$$\begin{array}{lcl} \forall m \in M \; \exists k \in K : hat(m,k) & \wedge & \forall m1, m2 \in M \; \forall k1, k2 \in K : \\ (hat(m,k1) \wedge hat(m,k2) & \Rightarrow & k1 = k2) \wedge \\ \forall m1, m2 \in M \; \forall k \in K : & & (hat(m1,k) \wedge hat(m2,k) \Rightarrow m1 = m2) \end{array}$$

1.3 Wissensarten

Die am leichtesten zu formalisierende Art symbolischen menschlichen Wissens ist das *begriffliche Wissen.*

Definition 1.9 Es sei G eine Menge von *Gegenständen (Objekten)* und M eine Menge von *Merkmalen (Attributen)* (die ein Gegenstand besitzen kann oder auch nicht). Ein *Konzept (Begriff)* ist ein Tripel (N, E, I) aus einem eindeutigen *Namen* N, einer *Extension* $E \subseteq G$ und einer *Intension* $I \subseteq M$.
Die Extension (der *Begriffsumfang*) ist die Menge aller Objekte, die zu dem Begriff gehören, die Intension (der *Begriffsinhalt*) ist die Menge aller Merkmale, die ein Objekt besitzen muß, um zum Begriff zu gehören.

Beispiel 1.10 Tiere aus Literatur, Film und Fernsehen

G = {Fury, Lassie, Flipper, Clarence, King-Kong, Moby Dick, Fritz the Cat,
Garfield,Snoopy, Tom, Jerry, Miss Piggy, Kermit, Bambi, Maja,
Onkel Otto, Idefix}.
M = {Zeichentrickfigur (ZF), literarische Figur (LF), Figur einer Fernsehserie (TVF), real lebendes Wesen (L), Katze (K), Meeressäuger (M),
Hund (H), Säugetier (S)}.

Das Zutreffen eines Merkmales auf ein Objekt wird durch folgende Tabelle festgelegt.

Name	ZF	LF	TVF	L	K	H	M	S
Fury			X	X				X
Lassie		X	X	X		X		X
Flipper			X	X			X	X
Clarence			X	X	X			X
King-Kong		X						X
Moby Dick		X		X			X	X
Fritz	X				X			X
Garfield	X		X		X			X
Snoopy	X		X			X		X
Tom	X		X		X			X
Jerry	X		X					X
Miss Piggy			X					X
Kermit			X					
Bambi	X							X
Maja	X	X	X					
Onkel Otto	X						X	X
Idefix	X					X		X

Einige Begriffe sind dann etwa:
(Katze, {Clarence, Fritz, Garfield, Tom}, {K,S})
(Hund, {Lassie, Snoopy, Idefix}, {H,S})
(Säugetier, $G\backslash$ {Kermit, Maja}, {S})
(Lassie, {Lassie}, {LF,TVF,L,H,S})

Im Sinne der folgenden Definition ist der letzte Begriff ein Individualkonzept, alle anderen sind Klassenkonzepte. Es gelten die Relationen *Lassie instanz_von Hund* und *Hund is_a Säugetier*.

Definition 1.11 Seien G und M wie in Def. 1.9

i) Ein Konzept (N, E, I) heißt *Individualkonzept*, wenn $\mid E \mid= 1$ gilt, sonst *Klassenkonzept*.

ii) Zwischen einem Individualkonzept $(N, \{g\}, I)$ und einem Klassenkonzept (N', E', I') besteht genau dann die *Instanz_von-Relation*, wenn $g \in E'$ (und daher auch $I' \subseteq I$) gilt. Die Merkmale des Klassenkonzepts werden also auf die Individualkonzepte vererbt.

iii) Zwischen einem Klassenkonzept $K = (N, E, I)$ und einem Klassenkonzept $K' = (N', E', I')$ besteht die *Is_a-Relation* genau dann, wenn $E \subseteq E'$ (und daher auch $I' \subseteq I$) gilt. K heißt dann *Unterbegriff* von K'.

Durch Gruppierung von Klassenkonzepten kann man zu Klassen ,,höherer Ordnung" aufsteigen:

Definition 1.12 Ein *Gruppenklassenkonzept* ist ein Konzept $(\mathcal{N}, \mathcal{E}, \mathcal{I})$, wobei $\mathcal{E}$ selbst aus Klassenkonzepten besteht.

Beispiel 1.13 In Beispiel 1.10 läßt sich eine Gruppenklasse (Tierarten, {Hund, Katze, Meeressäuger},$\mathcal{I}$) bilden, wobei $\mathcal{I}$ aus einer neuen Merkmalsmenge noch zu definieren wäre. Man muß dieses Konzept aber von $(Tier, G, I)$ unterscheiden, das auf einer niedrigeren Stufe gebildet wird.

Bemerkung 1.14 Es gibt auch nicht-objektorientierte Arten begrifflichen Wissens:
- *Massenkonzepte* (z.B. Sand, Luft, Wasser)
- *Ereignisse*, speziell *Handlungen* (z.B. Sonnenfinsternis, Vorlesung, vgl. $E1$ und $E2$ aus Bsp. 1.2)
- u.a. (z.B. ,,Wetter", ,,Würde des Menschen")

Daneben werden im folgenden noch weitere Wissensarten betrachtet:

- *Wissen um regelhafte Zusammenhänge* : ,,wenn Faktum A zutrifft, dann auch Faktum B", etwa ,,wenn der Mond zwischen Sonne und Erde tritt, dann gibt es eine Sonnenfinsternis."
- *Wissen über einschränkende Bedingungen (Constraints)*: ,,wenn eine Person einen Führerschein besitzt, dann ist ihr Alter mind. 18 Jahre."
- *unvollständiges Wissen*: ,,Alter der Person ist unbekannt"
- *unsicheres Wissen*: ,,Person ist möglicherweise ein Student"
- *widersprüchliches Wissen* (eigentlich kein Wissen!): ,,Person ist 12 Jahre alt und besitzt einen Führerschein".
- *ungenaues Wissen*: ,,Person ist groß (legt eher einen Bereich für einen exakten Wert fest)"
- *Wissen über Modalitäten eines Sachverhaltes*: d.h. ob der Sachverhalt notwendig, möglich, unmöglich, obligatorisch, geboten oder verboten ist.
- *prototypisches Wissen*: Aussagen, die ,,in der Regel" zutreffen und solange als korrekt angenommen werden, bis gegenteiliges Wissen vorliegt. Solche Aussagen heißen *Default-Aussagen*.

1.4 Formale Begriffsanalyse

Die formale Begriffsanalyse wurde von R. Wille eingeführt (vgl. [Wil87]) und dient zur formalen Darstellung von Begriffen und ihren Abhängigkeiten.

Definition 1.15 *Ableitungsoperatoren* Ein Kontext (G, M, κ) besteht aus:
- einer Menge G von *Gegenständen*,
- einer Menge M von *Merkmalen* und
- einer *Inzidenzrelation* $\kappa \subseteq G \times M$.
Dabei bedeutet $g\kappa m$, daß der Gegenstand g das Merkmal m besitzt (vgl. Bsp. 1.10)
Für jeden Kontext definiert man *Ableitungsoperatoren* zwischen $\mathcal{P}(G)$ und $\mathcal{P}\ (M)$ gemäß
$X' = \{m \in M \mid g\kappa m \text{ für alle } g \in X\} (\forall X \subseteq G)$,
$Y' = \{g \in G \mid g\kappa m \text{ für alle } m \in Y\} (\forall Y \subseteq M)$.

Satz 1.16 Sei (G, M, κ) ein Kontext. Für alle $X_i \subseteq G, Y_i \subseteq M$ gilt dann:

(1) $X_1 \subseteq X_2 \Rightarrow X_1' \supseteq X_2'$,
(2) $Y_1 \subseteq Y_2 \Rightarrow Y_1' \supseteq Y_2'$,
(3) $X_1 \subseteq X_1''$,
(4) $Y_1 \subseteq Y_1''$,
(5) $X_1 = X_1'''$,
(6) $Y_1 = Y_1'''$,
(7) $(\bigcup X_i)' = \bigcap X_i'$,
(8) $(\bigcup Y_i)' = \bigcap Y_i'$,
(9) $(\bigcap X_i)' \supseteq \bigcup X_i'$,
(10) $(\bigcap Y_i)' \supseteq \bigcup Y_i'$.

Beweis:
(1): $X_1' = \{m \in M \mid g\kappa m \text{ für alle } g \in X_1\}$
$\uparrow$
$X_2' = \{m \in M \mid g\kappa m \text{ für alle } g \in X_2 \supseteq X_1\}$,
also $X_2' \subseteq X_1'$.
(2): folgt dual!

(3): $X_1'' = \{g \in G \mid g\kappa m \text{ für alle } m \in X_1'\}$
$g \in X_1 \Rightarrow$ (nach Definition von X_1') $g\kappa m$ für alle $m \in X_1'$,
damit gilt also $X_1 \subseteq X_1''$.
(4): folgt dual!

(5): $X_1 \subseteq X_1'' \overset{(1)}{\Rightarrow} X_1' \supseteq X_1'''$
Setze $Y_1 \overset{!}{=} X_1' \Rightarrow X_1' = Y_1 \subseteq Y_1'' = X_1'''$,
also $X_1' = X_1'''$.
(6): folgt dual!

(7): $m \in (\bigcup X_i)' \Leftrightarrow g\kappa m$ für alle $g \in \bigcup X_i$
$\Leftrightarrow g\kappa m$ für alle $g \in X_i$ und alle $i \in I$
$m \in \bigcap X_i' \Leftrightarrow m \in X_i'$ für alle $i \in I$
$\Leftrightarrow g\kappa m$ für alle $g \in X_i$ und alle $i \in I$
also $(\bigcup X_i)' = \bigcap X_i'$.
(8): folgt dual!

(9): $m \in \bigcup X_i' \Leftrightarrow$ es existiert $i \in I$ mit $m \in X_i'$
$\Leftrightarrow$ es existiert $i \in I$ mit $g\kappa m$ für alle $g \in X_i$
$\Rightarrow g\kappa m$ für alle $g \in \bigcap X_i \Leftrightarrow m \in (\bigcap X_i)'$
also $(\bigcap X_i)' \supseteq \bigcup X_i'$
(10): folgt dual!

Definition 1.17 Sei (G, M, κ) ein Kontext. Ein Paar (A, B) heißt *Begriff* (des Kontextes (G, M, κ)), wenn $A \subseteq G, B \subseteq M$ und $A' = B$, sowie $B' = A$ gilt. Dabei heißt A *Umfang* und B *Inhalt* von (A, B).
Für Begriffe (A_1, B_1) und (A_2, B_2) desselben Kontextes nennt man (A_2, B_2) *Oberbegriff* von (A_1, B_1), in Zeichen $(A_1, B_1) \sqsubseteq (A_2, B_2)$, wenn gilt $A_1 \subseteq A_2$ (vgl. auch Def. 1.9).

Satz 1.18 Sei (G,M,κ) ein Kontext und B(G,M,κ) die Menge seiner Begriffe. Dann ist $\sqsubseteq$ partielle Ordnung auf B(G,M,κ) und man hat $(A_1, B_1) \sqsubseteq (A_2, B_2)$ genau dann, wenn $B_1 \supseteq B_2$ gilt. Es ist sogar (B(G,M,κ),$\sqsubseteq$) ein vollständiger Verband, der *Begriffsverband* von (G,M,κ).

Genauer: Sind (A_j, B_j) aus $B(G, M, \kappa)$ für $j \in J$, dann gilt:

$$\inf_{j \in J}(A_j, B_j) = (\bigcap A_j, (\bigcup B_j)'') \in B(G, M, \kappa)$$
$$\sup_{j \in J}(A_j, B_j) = ((\bigcup A_j)'', \bigcap B_j) \in B(G, M, \kappa)$$

Beweis:
Wegen $(A_1, B_1) \sqsubseteq (A_2, B_2) \Leftrightarrow A_1 \subseteq A_2$ ist $\sqsubseteq$ partielle Ordnung auf $B(G, M, \kappa)$.

Weiterhin gilt $A_1 \subseteq A_2 \overset{(1)}{\Rightarrow} A_1' \supseteq A_2'$, also $B_1 \supseteq B_2$ und
$B_1 \supseteq B_2 \overset{(2)}{\Rightarrow} B_1' \subseteq B_2'$, also $A_1 \subseteq A_2$,
daraus folgt $A_1 \subseteq A_2 \Leftrightarrow B_1 \supseteq B_2$.

Seien $(A_j, B_j) \in B(G, M, \kappa)$ für $j \in J$ sowie $B \overset{!}{=} (\bigcup B_j)'' \subseteq M$ und $A \overset{!}{=} \bigcap A_j \subseteq G$
Es folgt $B' = (\bigcup B_j)''' = (\bigcup B_j)' = \bigcap B_j' = \bigcap A_j = A$ und
$A' = B'' = (\bigcup B_j)'''' = (\bigcup B_j)'' = B$, also
$(A, B) \in B(G, M, \kappa)$.

Ersichtlich gilt $(A, B) \sqsubseteq (A_j, B_j)$ für alle $j \in J$ und aus $(C, D) \in B(G, M, \kappa)$ mit $(C, D) \sqsubseteq (A_j, B_j)$ für alle $j \in J$ folgt $C \subseteq \bigcap A_j = A$, also $(C, D) \sqsubseteq (A, B)$.
Damit ist $(A, B) = \inf_{j \in J}(A_j, B_j)$ gezeigt. Die Behauptung über $\sup_{j \in J}(A_j, B_j)$ folgt analog.

Definition 1.19 Sei (G,M,κ) Kontext. Eine Menge $X \subseteq M$ *impliziert* eine Menge von Merkmalen $Y \subseteq M$, wenn $X' \subseteq Y'$ gilt.
Man schreibt dann $X \Rightarrow Y$.

Bemerkung 1.20

i) Es ist $X \Rightarrow Y$ gleichwertig zu $Y \subseteq X''$, denn aus $X' \subseteq Y'$ folgt $Y \subseteq Y'' \subseteq X''$ und hieraus wiederum $X' = X''' \subseteq Y'$.

ii) Es gibt Algorithmen, die sämtliche Implikationen eines endlichen Kontextes effektiv berechnen.

Weitere Literatur findet sich im Literaturverzeichnis unter [Wil87] [Rei91].

Kapitel 2

Prädikatenlogik

Im folgenden wird eine (Klasse von) Sprache(n) PL1 der Prädikatenlogik erster Stufe definiert.

2.1 Syntax der Prädikatenlogik erster Stufe (PL1)

Es werden zunächst Terme, die Objekte repräsentieren, und Formeln, die Aussagen über Objekte darstellen, von PL1 definiert.
Zur Vertiefung dieses Kapitels vergleiche man [BB87],[CM81],[GN89],[HK89],[KBS86], [Kow79],[Llo84] und [Ric89] im Literaturverzeichnis.

Definition 2.1 Das Alphabet $\mathcal{A}$ von (PL1) besteht aus den folgenden paarweise disjunkten Teilmengen.
- einer Menge $\mathcal{F} = \{f, g, \ldots, f_1, \ldots\}$ von *Funktionssymbolen*,
- einer Menge $\mathcal{P} = \{p, q, \ldots, p_1, \ldots\}$ von *Prädikatssymbolen*,
- einer Menge $\mathcal{V} = \{x, y, \ldots, x_1, \ldots\}$ von *Variablensymbolen*,
- der Menge $\mathcal{J} = \{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$ *logischer Junktoren*,
- der Menge $\mathcal{Q} = \{\forall, \exists\}$ der *Quantoren*,
- der Menge $\mathcal{H} = \{(,),,\}$ von *Hilfssymbolen*,

Jedem Funktions- und Prädikatssymbol ist dabei eine *Stelligkeit* $n \in I\!N_0$ zugeordnet. Die nullstelligen Funktionssymbole heißen *Konstante*, die nullstelligen Prädikatssymbole *Aussagenvariable*.

Bemerkung 2.2

i) Es ist $\mathcal{F} = \emptyset$ zulässig. Für $\mathcal{F} \neq \emptyset$ ist es zweckmäßig, $\mathcal{F}$ als abzählbar unendlich vorauszusetzen. Analog für $\mathcal{V}$.

ii) Läßt man $\mathcal{F}$,$\mathcal{V}$ und auch $\mathcal{Q}$ weg, und sind alle Prädikatssymbole nullstellig, so spricht man von einer *Aussagenlogik*.

Definition 2.3 Die Menge $\mathcal{T}$ der *Terme* wird wie folgt gebildet :

i) $\mathcal{V} \subseteq \mathcal{T}$,

ii) $t_1, \ldots, t_n \in \mathcal{T}$, $f \in \mathcal{F}$ n-stellig $\Rightarrow f(t_1, \ldots, t_n) \in \mathcal{T}$

iii) $\mathcal{T}$ ist die kleinste Menge mit i) und ii).

Ein Term heißt *Grundterm*, wenn er keine Variablen enthält.

Definition 2.4 Die *atomaren Formeln (Atome)* sind von der Form $p(t_1, \ldots, t_n)$ für $p \in \mathcal{P}$ n-stellig und $t_1, \ldots, t_n \in \mathcal{T}$.
Variablenfreie Atome heißen *Grundformeln*.

Beispiel 2.5 $\mathcal{F} = \{c, f, g\}$, c nullstellig, f einstellig, g zweistellig, $\mathcal{P}$ =$\{p, q\}$, p einstellig, q nullstellig, $\mathcal{V} = \{x, y\}$
Terme : $\underline{\text{c}}$, $\underline{\text{f(c)}}$, $\underline{\text{g(c,c)}}$, g(c,f(y)), ... Grundterme sind unterstrichen
Atome : $\underline{\text{q}}$, $\underline{\text{p(c)}}$, p(x), p(f(x)), ... Grundformeln sind unterstrichen

Definition 2.6 Die *Formeln* werden wie folgt gebildet:

i) Jedes Atom ist eine Formel.

ii) Ist A eine Formel, dann auch $\neg A$.

iii) Sind A und B Formeln, dann auch $(A \wedge B), (A \vee B), (A \rightarrow B), (A \leftrightarrow B)$.

iv) Ist A eine Formel und $x \in \mathcal{V}$, dann sind auch $\forall x A$ und $\exists x A$ Formeln. In diesen Formeln heißt die Variable x dann *gebunden*.

v) Nur die durch Iteration von i) - iv) gebildeten Zeichenketten sind Formeln.

Eine Formel, in der jede Variable gebunden ist, heißt *geschlossen*.

Definition 2.7 Ein *Literal* ist ein Atom A (*positives Literal*) oder seine Negation $\neg A$ (*negatives Literal*). Eine *Klausel* ist eine geschlossene Formel der Form $\forall x_1 \ldots \forall x_k (L_1 \vee \ldots \vee L_n)$, wobei die L_i Literale sind und die $x_1, \ldots, x_k$ sämtliche Variablen, die in ihnen vorkommen. Sie wird auch als $\{L_1, \ldots, L_n\}$ notiert. Enthält sie höchstens ein positives Literal, so heißt sie eine *Horn-Klausel.*
Existiert in einer Hornklausel ein positives Literal, so heißt sie eine *Programmklausel*, sonst eine *Zielklausel (Goal)*. Ein *logisches Programm* ist eine endliche Menge von Programmklauseln.

Beispiel 2.8 $\mathcal{F}$,$\mathcal{P}$,$\mathcal{V}$ wie in Beisp. 2.5

Literale: $\underline{p(x)}$,$\neg p(f(c))$,$\underline{q}$,$\neg q$, ... positive Literale sind unterstrichen
Klauseln:$\underline{p(c)}$ oder $\{p(c)\}$, Programmklauseln,
$\forall x(\neg p(x) \vee p(f(x)))$ oder $\{\neg p(x), p(f(x))\}$, Programmklauseln,
$\forall x \forall y : (\neg p(x) \vee \neg p(f(y)))$ oder $\{\neg p(x), \neg p(f(y))\}$, Zielklauseln,
$\forall y(p(y) \vee q)$ oder $\{p(y), q\}$, keine Horn - Klausel
Prolog: $p(c)$.
$p(f(X)) : -p(X)$. (:- steht für $\leftarrow$)
?- $p(X), p(f(Y))$.
Antwort : yes, X = c, Y = c.

2.2 Semantik

Nachdem Terme und Formeln als zulässige Zeichenketten über dem Alphabet $\mathcal{A}$ von PL1 definiert wurden, ist nun ihre Bedeutung festzulegen.

Definition 2.9

i) Eine *Interpretation* I in (eine Menge) $U \neq \emptyset$ (*Universum* oder *Modellwelt*) ist eine Abbildung I von $\mathcal{F} \cup \mathcal{P}$ in die Menge der Relationen über U, die jedem n-stelligen Funktionssymbol $f \in \mathcal{F}$ eine Funktion $I(f) : U^n \to U$ und jedem n-stelligen Prädikatssymbol $p \in \mathcal{P}$ eine Relation $I(p) \subseteq U^n$ zuordnet.

ii) Eine *(Variablen-)Belegung* bzgl. der Interpretation I in U ist eine Abbildung $\beta : \mathcal{V} \to U$. Eine *Variation von* β *im Argument* $x \in \mathcal{V}$ ist eine Belegung $\beta' : \mathcal{V} \to U$ mit $\beta'(y) = \beta(y)$ für alle $y \in \mathcal{V} \backslash \{x\}$.

Bemerkung 2.10 Man setzt β bzgl. I gemäß $\beta(f(t_1, \ldots, t_n)) = I(f)(\beta(t_1), \ldots, \beta(t_n))$ zu einer Abbildung $\beta : \mathcal{T} \to U$ fort.

Definition 2.11 Es sei I eine Interpretation in U. Eine Belegung $\beta : \mathcal{V} \to$ U *erfüllt* genau dann *eine Formel* F, wenn einer der folgenden Fälle eintritt.

i) F $\equiv$ p(t_1,..,t_n) und es gilt I(p)($\beta(t_1)$,..,$\beta(t_n)$),

ii) F $\equiv \neg$A und β erfüllt A nicht,

iii) F $\equiv$ A $\wedge$ B und (β erfüllt A und β erfüllt B),

iv) F $\equiv$ A $\vee$ B und (β erfüllt A oder β erfüllt B),

v) F $\equiv$ A $\to$ B und (β erfüllt $\neg$A oder β erfüllt B),

vi) F $\equiv$ A $\leftrightarrow$ B und (β erfüllt A $\to$ B und β erfüllt B $\to$ A),

vii) F $\equiv \exists$x A und es existiert eine Variation β' von β im Argument x, die A erfüllt.

viii) F $\equiv \forall$x A und alle Variationen β' von β im Argument x erfüllen A.

Beispiel 2.12 Es seinen $\mathcal{F}, \mathcal{P}, \mathcal{V}$, wie in Bsp. 2.5 und 2.8
$U = I\!R, I(c) = 0 \in I\!R, I(f)(x) = x + 1$ für alle $x \in I\!R$,
$I(p)$: "ist eine natürliche Zahl. "
$I(p(c))$: "0 ist eine natürliche Zahl. "
$I(p(f(x)) : -p(x))$: "ist x eine natürliche Zahl, dann auch $x + 1$."
Beide Formeln sind erfüllt für jede Belegung $\beta : \mathcal{V} \to \mathcal{U}$. Ändert man dagegen $I(c) = 0.5 \in I\!R$, dann ist $F \equiv p(c)$ für keine Belegung erfüllt, $p(f(x)) : -p(x)$ aber immer noch für jede.

Definition 2.13

i) Eine Formel F ist *wahr* für eine Interpretation I in U, wenn alle Belegungen $\beta : \mathcal{V} \to U$ sie erfüllen und *falsch*, wenn keine Belegung sie erfüllt. Im ersten Fall heißt (I, U) ein *Modell* für F.

ii) *Tautologien* oder *allgemeingültige Formeln* F (in Zeichen $\models F$) sind Formeln, die für alle Interpretationen wahr sind, *Kontradiktionen* oder *Widersprüche* sind Formeln, die für keine Interpretation und keine Belegung erfüllt sind.

iii) Formeln F und G heißen *logisch äquivalent* (in Zeichen $F \Leftrightarrow G$), wenn $(F \leftrightarrow G)$ eine Tautologie ist.

iv) Eine Formel F heißt *semantische Folgerung* aus einer Formelmenge Σ, wenn für jede Interpretation jede Belegung, die alle Formeln aus Σ erfüllt, auch F erfüllt (in Zeichen $\Sigma \models F$).

Bemerkung 2.14

i) Geschlossene Formeln sind für jede Interpretation entweder wahr oder falsch, unabhängig von der Belegung.

ii) Widersprüche sind die Negationen der Tautologien.

iii) Formeln F und G sind genau dann logisch äquivalent, wenn $\{F\} \models G$ und $\{G\} \models F$ gelten.

iv) Eine Formel F ist genau dann eine semantische Folgerung aus $\{F_1, \ldots, F_n\}$, wenn $F_1 \wedge \ldots \wedge F_n \to F$ eine Tautologie ist; F ist genau dann keine Folgerung, wenn $(F_1 \wedge \ldots \wedge F_n) \wedge \neg$F eine erfüllbare Interpretation besitzt.

v) Es gibt kein algorithmisches, stets terminierendes Verfahren, das für eine beliebige Formel F aus PL1 feststellt, ob F eine Tautologie ist. Die Prädikatenlogik erster Stufe ist *nicht entscheidbar*. Es gibt aber Algorithmen, die für jede Tautologie F aus PL1 in endlich vielen Schritten einen Beweis der Allgemeingültigkeit von F erbringen. PL1 ist *semi-entscheidbar*.

vi) Auf den folgenden Äquivalenzen beruht das Prinzip des Widerspruchsbeweises. Dabei sind $F_1, \ldots, F_n$ Voraussetzungen und F das zu beweisende Theorem.

$\{F_1, \ldots, F_n\} \models F$
$\Leftrightarrow \models F_1 \wedge \ldots \wedge F_n \to$ F
$\Leftrightarrow \models \neg(F_1 \wedge \ldots \wedge F_n) \vee F$
$\Leftrightarrow F_1 \wedge \ldots \wedge F_n \wedge \neg$ F ist widersprüchlich

Logisch äquivalente Formeln können syntaktisch sehr verschieden sein. Man versucht also, möglichst ,,einfache" Darstellungen zu finden.

Definition 2.15

i) Eine Formel ist in *pränexer Normalform*, wenn sie die Form $Q_1x_1 \dots Q_kx_kF$ hat, wobei die Q_i aus $\{\forall, \exists\}$ sind und F quantorenfrei ist.

ii) Eine quantorenfreie Formel ist in *konjunktiver Normalform*, wenn sie die Form $F_1 \wedge \dots \wedge F_n$ hat, wobei die F_i jeweils Disjunktionen von Literalen sind.

iii) Sie ist in *disjunktiver Normalform*, wenn sie die Form $F_1 \vee \dots \vee F_n$ hat, wobei die F_i jeweils Konjunktionen von Literalen sind.

Bei der Umformung einer gegebenen Formel in eine dieser Normalformen sind folgende logische Äquivalenzen nützlich :
1) $\neg\neg A \Leftrightarrow A$
2) $\neg(A \vee B) \Leftrightarrow \neg A \wedge \neg B$
3) $\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B$
4) $(A \leftrightarrow B) \Leftrightarrow (A \rightarrow B) \wedge (B \rightarrow A)$
5) $(A \rightarrow B) \Leftrightarrow (\neg A \vee B)$
6) $\forall xA \Leftrightarrow \neg\exists x \neg A; \exists xA \Leftrightarrow \neg\forall x\neg A$
7) $(A \rightarrow \forall xB) \Leftrightarrow \forall y(A \rightarrow B')$; $(A \rightarrow \exists xB) \Leftrightarrow \exists y(A \rightarrow B')$
8) $\forall xA \rightarrow B \Leftrightarrow \exists y(A' \rightarrow B)$; $\exists xA \rightarrow B \Leftrightarrow \forall y(A' \rightarrow B)$
Dabei sei y in 7) bzw. 8) eine Variable, die weder in A noch in B vorkommt, und B' bzw. A'entstehen aus B bzw. A durch Umbenennung von x in y.

Mit diesen Äquivalenzen kann man jede Formel F aus PL1 in eine äquivalente Formel G überführen, die in pränexer Normalfall ist. Entsprechend kann man jede quantorenfreie Formel in konjunktive oder disjunktive Normalform überführen. Schließlich kann man noch Existenzquantoren eliminieren:

Sei $F \equiv \forall x_1 \dots \forall x_k \exists y G\ (..y..)$
$\uparrow$ $\uparrow$ $\uparrow$ Formel in der y vorkommt
nur $\forall$ - Quantoren

Wähle ein *neues* k-stelliges Funktionssymbol f (also f $\notin \mathcal{F}$!) und bilde
$Sk(F) \equiv \forall x_1 \dots \forall x_k G(..f(x_1, \dots, x_k) \dots)$
Dabei wird y überall durch $f(x_1, \dots, x_k)$ ersetzt. Diese Transformation $F \rightarrow Sk(F)$ heißt *Skolemisierung*. Es gilt:

Satz 2.16 Eine Formel F in pränexer NF ist genau dann eine Kontradiktion, wenn $Sk(F)$ eine Kontradiktion ist.

Wie die folgenden Äquivalenzen zeigen, kann man den in Bemerkung 2.14 vi) skizzierten Widerspruchsbeweis auch mit einer Formel in konjunktiver Normalform führen.
$G \equiv F_1 \wedge \dots \wedge F_n \wedge \neg F$ Kontradiktion
$\Leftrightarrow$ Pränexe NF G' von G ist Kontradiktion
$\Leftrightarrow$ $Sk(G') \equiv H$ ist Kontradiktion
$\Leftrightarrow$ Konjunktive NF $H_1 \wedge \dots \wedge H_m$ von H ist Kontradiktion.

Beispiel 2.17 $F \equiv \forall x(p(x) \rightarrow (q(x) \rightarrow \exists y \forall z r(x, y, z)))$
$\Leftrightarrow \forall x(p(x) \rightarrow (\exists y \forall z q(x) \rightarrow r(x, y, z)))$ mit (7)
$\Leftrightarrow \forall x \exists y \forall z(p(x) \rightarrow (q(x) \rightarrow r(x, y, z)))$ mit (7)
$\Leftrightarrow \forall x \exists y \forall z(\neg p(x) \vee (\neg q(x) \vee r(x, y, z))) \equiv G$ mit (5)
Sk(G) $\equiv \forall x \forall$ z ($\neg$p(x) $\vee\neg$q(x) $\vee$ r(x,f(x),z))
In Klauselform : $\{\neg$p(x), $\neg$q(x), r(x,f(x),z)$\}$, also sogar Programmklausel.

In Prolog - Notation :

r(x,f(x),z) :- p(x),q(x). ,,Regel"
,,Kopf" ,,Rumpf"

Wie in diesem Beispiel werden im folgenden alle Variablen einer Klausel als allquantifiziert betrachtet.

2.3 Unifikation

Hierbei geht es darum, Terme oder Literale, die syntaktisch verschieden sind, durch geeignete Belegungen der in ihnen vorkommenden Variablen ,,gleich zu machen", zu unifizieren.

Definition 2.18 Eine *Substitution* σ ist eine Abbildung $\sigma : \mathcal{V} \to \mathcal{T}$ mit $\sigma(v) \neq v$ nur für endlich viele $v \in \mathcal{V}$, etwa $v_1, \ldots, v_n$. Man schreibt dann $\sigma = \{v_1 \mid t_1, \ldots, v_n \mid t_n\}$ für $t_i = \sigma(v_i), i = 1, .., n$. Ist t_i stets Grundterm, dann heißt σ *Grundsubstitution*. Eine Substitution, die eine Permutation der Variablen ist, heißt *Variablenumbenennung*.

Bemerkung 2.19

i) Man kann jede Substitution $\sigma : \mathcal{V} \to \mathcal{T}$ gemäß $\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$ zu einer Abbildung $\sigma : \mathcal{T} \to \mathcal{T}$ fortsetzen.

ii) Sind dann σ und τ zwei derartige Substitutionen, so auch $\sigma \circ \tau : \mathcal{T} \to \mathcal{T}$.

iii) Im folgenden betrachten wir jede Substitution $\sigma : \mathcal{V} \to \mathcal{T}$ sogar für beliebige Literale gemäß

$$\sigma(p(t_1, \ldots, t_n)) = p(\sigma(t_1), \ldots, \sigma(t_n))) \text{ bzw. } \sigma(\neg p(t_1, \ldots, t_n)) = \neg p(\sigma(t_1), \ldots, \sigma(t_n)).$$

Definition 2.20 Seien $t_1, \ldots, t_n$ Terme oder sogar Literale.

i) Eine Substitution σ mit $\sigma(t_1) = \sigma(t_2) = \ldots = \sigma(t_n)$ heißt *Unifikator* von $t_1, \ldots, t_n$.

ii) Ein Unifikator σ von $t_1, \ldots, t_n$ heißt dann *allgemeinster Unifikator* von $t_1, \ldots, t_n$, wenn für jeden Unifikator τ von $t_1, \ldots, t_n$ eine Substitution ρ mit $\tau = \rho \circ \sigma$ existiert.

Bemerkung 2.21 Sind σ und τ in ii) allgemeinste Unifikatoren, dann ist ρ eine Variablenumbenennung.

Beispiel 2.22

i) $L_1 \equiv p(X, Y), L_2 \equiv p(X, b), L_3 \equiv p(f(a), Z)$, $\sigma = \{X \mid f(a), Y \mid b, Z \mid b\}$ Grundsubstitution mit $\sigma(p(X, Y)) = \sigma(p(X, b)) = \sigma(p(f(a), Z)) = p(f(a), b)$.

ii) $p(X, Y), p(f(Z), b)$, $\sigma = \{X \mid f(Z), Y \mid b\}, \tau = \{X \mid f(b), Y \mid b, Z \mid b\}$, $\rho = \{Z \mid b\}$.

iii) $p(X, f(Y)), p(X, g(Y))$ sind nicht unifizierbar, ebensowenig wie $p(X, Y), p(X, f(Y))$.

iv) $f(X, Y), f(Z, Z)$ haben allgemeinste Unifikatoren $\{X \mid Z, Y \mid Z\}, \{X \mid Y, Z \mid Y\}, \{Y \mid X, Z \mid X\}$.

Satz 2.23 Der folgende Unifikationsalgorithmus entscheidet für jede endliche Menge von Literalen, ob sie einen Unifikator besitzt. Ist dieses der Fall, so berechnet er einen allgemeinsten Unifikator.
Sei E eine endliche Menge von Literalen.
0. Enthält E ein positives und ein negatives Literal, dann existiert kein Unifikator.
1. Enthält E Atome mit verschiedenen Prädikatssymbolen, dann ex. kein Unifikator.
2. Setze $i := 0$ und $\sigma_0 := id_{\mathcal{V}}$.
3. Ist $\mid \sigma_i(E) \mid = 1$, dann ist σ_i allg. Unifikator.
4. Wähle $L_1, L_2 \in \sigma_i(E)$ beliebig. Besitzen L_1 und L_2 an der ersten Stelle (von links her), an der sie sich unterscheiden, verschiedene Funktionssymbole, dann existiert kein Unifikator.
5. Steht an dieser Stelle dagegen eine Variable X und ein Term t, der X enthält, dann ex. kein Unifikator (,,occur-check"!)
6. Setze $\sigma_{i+1} := \{X \mid t\} \circ \sigma_i, i := i + 1$ und gehe zu 3.

2.4 Syntaktische Ableitungen

Definition 2.24

i) Eine *(n-stellige) Ableitungsregel* (besser ein Regelschema) mit den *Prämissen* $G_1,..,G_n$ und der *Konklusion* G ist ein syntaktisches Schema der Gestalt:

$$\frac{\begin{array}{c} G_1 \\ . \\ . \\ G_n \end{array}}{G} \qquad \text{bzw.} \qquad \frac{G_1 \dots G_n}{G}$$

mit prädikatenlogischen Formeln $G, G_1, \dots, G_n$. Ein *Regelsystem* $\mathcal{R}$ ist eine Menge von Ableitungsregeln.

ii) Eine Ableitungsregel auf gegebene Formeln *anwenden* heißt, eine Instanz des Schemas mit diesen Formeln zur Deckung bringen und daraus die Konklusion als entsprechende Instanz zu erhalten. Dabei soll stets entscheidbar sein, ob das Schema auf diese Formeln anwendbar ist, und die Konklusion soll berechenbar sein. Eine solche Regelanwendung heißt ein *Ableitungsschritt*.

Beispiel 2.25 Die zweistellige Ableitungsregel
(S):

$$\frac{A \to B \vee P \qquad P \wedge C \to D}{A \wedge C \to B \vee D}$$

heißt *Schnittregel*. Dabei ist P ein Atom, A und C sind Konjunktionen und B und D Disjunktionen von Atomen.
Anwendung:

$$\underbrace{p(a) \wedge q(y,a) \wedge r(y,b)}_{A} \to \underbrace{p(b)}_{B} \vee \underbrace{s(x)}_{P} \qquad \underbrace{r(y,b)}_{C} \wedge \underbrace{s(x)}_{P} \to \underbrace{p(b) \vee s(a)}_{D}$$

$p(a) \wedge q(y,a) \wedge r(y,b) \to p(b) \vee s(a)$

Beispiel 2.26 Die einstellige Ableitungsregel
(Subst):

$$\frac{G}{\sigma(G)}$$

heißt *Substitutionsregel*. Dabei ist G eine quantorenfreie Formel und σ eine Substitution.

Definition 2.27 Es seien $P_1,..,P_n$ (n ≥ 1) Atome, A ein Konjunktion und B eine Disjunktion von Atomen, sowie σ eine allgemeinste Unifikation von $\{P_1,..,P_n\}$.

Dann heißt die Ableitungsregel
(Fak):

$$\frac{A \to B \vee P_1 \vee .. \vee P_n}{\sigma(A \to B \vee P_1)} \text{ bzw. } \frac{P_1 \wedge .. \wedge P_n \wedge A \to B}{\sigma(P_1 \wedge A \to B)}$$

Faktorisierungsregel.

Definition 2.28 Es seien $P_1, \ldots, P_n, Q_1, \ldots, Q_m$ Atome, A und C Konjunktionen, sowie B und D Disjunktionen von Atomen. Weiterhin sei π eine Umbenennung der Variablen von $Q_1 \wedge \ldots \wedge Q_m \wedge C \rightarrow D$ derart, daß $A \rightarrow B \vee P_1 \vee \ldots \vee P_n$ und $\pi(Q_1 \wedge \ldots \wedge Q_m \wedge C \rightarrow D)$ keine gemeinsamen Variablen enthalten und σ ein allg. Unifikator von
$\{P_1, \ldots, P_n,\ \pi(Q_1),..,\pi(Q_m)\}$.

Dann heißt die Ableitungsregel
(Rob):

$$\frac{A \rightarrow B \vee P_1 \vee \ldots \vee P_n \qquad Q_1 \wedge \ldots \wedge Q_m \wedge C \rightarrow D}{\sigma(A \wedge \pi(C) \rightarrow B \vee \pi(D))}$$

Resolution (nach Robinson, 1965).

Beispiel 2.29
$r(X,a) \wedge s(Y) \rightarrow \underbrace{r(g(X,a),Y)}_{P_1}$ $(B\ und\ C\ sind\ leer)$ $\quad \underbrace{r(X,X)}_{Q_1} \rightarrow \underbrace{t(X)}_{D}$
$\pi = \{X \mid Z, Z \mid X\}$,
$r(X,a) \wedge s(Y) \rightarrow r(g(X,a),Y), \quad r(Z,Z) \rightarrow t(Z)$,
$\sigma = \{Y \mid g(X,a), Z \mid g(X,a)\}$,
$r(X,a) \wedge s(g(X,a)) \rightarrow t(g(X,a))$.

Definition 2.30 Es sei $\mathcal{R}$ ein Regelsystem, X eine (endliche) Formelmenge, G eine Formel.
Eine *Ableitung von G aus X mit $\mathcal{R}$* ist induktiv definiert.

i) Für jedes $G \in X$ ist (G) eine *Ableitung von G aus X mit $\mathcal{R}$*.

ii) Ist $(C_1, \ldots, C_k)$ eine Ableitung von C_k aus X mit $\mathcal{R}$ und $R \in \mathcal{R}$ eine n-stellige Ableitungsregel, so daß durch Anwendung von R auf die Formeln $G_1, \ldots, G_n \in X \cup \{C_1, \ldots, C_k\}$ die Konklusion G entsteht, dann ist $(C_1, \ldots, C_k, G)$ eine Ableitung von G aus X mit $\mathcal{R}$.

Gibt es eine Ableitung von G aus X mit $\mathcal{R}$, so heißt *G aus X mit $\mathcal{R}$ ableitbar*, in Zeichen : $X \vdash_{\mathcal{R}} G$.
Man sagt auch, daß G eine *syntaktische Folgerung* aus X ist und $\vdash_{\mathcal{R}}$ ein *Ableitungsoperator*.

2.5 Korrektheit und Vollständigkeit

Definition 2.31 Ein Regelsystem $\mathcal{R}$ heißt
korrekt, falls gilt: $X \vdash_{\mathcal{R}} G$ impliziert $X \models G$,
vollständig, falls gilt: $X \models G$ impliziert $X \vdash_{\mathcal{R}} G$,
widerlegungsvollständig, falls gilt "X widersprüchlich" impliziert $X \vdash_{\mathcal{R}} \Box$ ("leere Klausel"),
für alle Formelmengen X und alle Formeln G (einer bestimmten Formelklasse).

Satz 2.32 Das Regelsystem $\mathcal{R} = \{$(S),(Subst)$\}$ ist korrekt für Klauseln.
Beweis: Nach der Definition von $\vdash_{\mathcal{R}}$ reicht es, die Korrektheit eines einzelnen Ableitungsschrittes jeweils nach (S) bzw. (Subst) zu zeigen.
(Subst):

$$\frac{G}{\sigma(G)}$$

für eine Klausel G und eine Substitution σ, also G $\vdash_{(Subst)} \sigma$(G).
Sei (I,U) ein Modell von G. Dann ist G für jede Variablenbelegung $\beta : \mathcal{V} \rightarrow U$ erfüllt, insbesondere also für die durch σ vermittelte Belegung $\beta' : \mathcal{V} \rightarrow U$. Dies heißt aber, daß $\sigma(G)$ für β' erfüllt ist. Da $\sigma(G)$ mit G eine geschlossene Formel ist, ist $\sigma(G)$ dann schon wahr für (I,U). Dies zeigt $G \models \sigma(G)$.

(S) :

$$\frac{A \rightarrow B \vee P \qquad P \wedge C \rightarrow D}{A \wedge C \rightarrow B \vee D},$$

also $X = \{A \rightarrow B \vee P, P \wedge C \rightarrow D\} \vdash_{(S)} A \wedge C \rightarrow B \vee D$.

Sei (I, U) ein Modell für X, d.h. $A \rightarrow B \vee P$ und $P \wedge C \rightarrow D$ sind für jede Belegung $\beta : \mathcal{V} \rightarrow U$ erfüllt. Sei β solch eine Belegung. Hieraus folgt
(β erfüllt nicht A oder β erfüllt B oder β erfüllt P) und (β erfüllt nicht P oder β erfüllt nicht C oder β erfüllt D).

Nun kann man zwei Fälle unterscheiden:
1.Fall β erfüllt P. Dann tritt nach dem zweiten Teil der Vorausssetzung wenigstens
eine der folgenden Möglichkeiten ein.
β erfüllt **nicht** $C \Rightarrow \beta$ erfüllt nicht $A \wedge C$
β erfüllt $D \Rightarrow \beta$ erfüllt $B \vee D$.
In jedem dieser beiden Fälle gilt aber β erfüllt $A \wedge C \rightarrow B \vee D$.
2.Fall β erfüllt **nicht** P. Wiederum gibt es zwei Möglichkeiten:
β erfüllt **nicht** $A \Rightarrow \beta$ erfüllt nicht $A \wedge C$
β erfüllt $B \Rightarrow \beta$ erfüllt $B \vee D$.
Auch hier erfüllt β stets $A \wedge C \rightarrow B \vee D$.

Also ist $A \wedge C \rightarrow B \vee D$ für jede beliebige Belegung $\beta : \mathcal{V} \rightarrow U$ erfüllt, d.h. (I, U) ist auch Modell von $A \wedge C \rightarrow B \vee D$.

Bemerkung 2.33

i) Insbesondere ist $\mathcal{R}$ korrekt für Horn-Klauseln.

ii) Der Beweis zur Korrektheit von (Subst) zeigt sofort die Korrektheit von (Fak).

iii) Analog ergibt sich die Korrektheit von (Rob) für Klauseln.

Damit ist die übliche PROLOG - Interpretationsregel "Resolution mit Unifikation" korrekt für PROLOG - Programme.

Wesentlich aufwendiger ist der Nachweis der Widerlegungsvollständigkeit (vgl. 1.49 bis 1.55 in [HK89]).

Satz 2.34 $\mathcal{R} = \{$(S),(Subst)$\}$ ist widerlegungsvollständig für Klauseln.

Eine Analyse der in [HK89] angegebenen Beweise zeigt insbesondere die Widerlegungsvollständigkeit der PROLOG - Interpretation (Rob) für Horn-Klauseln.

Beispiel 2.35 Gegeben seien die Aussagen :

1. Wer lesen kann, der ist gebildet.

2. Delphine sind nicht gebildet.

3. Einige Delphine sind intelligent.

Behauptung: 4) Es gibt jemanden, der intelligent ist und nicht lesen kann.

Repräsentation in PL1:
1) $\forall x(l(x) \to g(x)) \equiv F_1$
2) $\forall x(d(x) \to \neg g(x)) \equiv F_2$
3) $\exists x(d(x) \wedge i(x)) \equiv F_3 \Rightarrow Sk(F_3) \equiv d(\text{flipper}) \wedge i(\text{flipper})$
4) $\exists x(i(x) \wedge \neg l(x)) \equiv F_4 \Rightarrow \neg F_4 \equiv \neg i(x) \vee l(x)$
In Klauselform :
1)$\{\neg l(x), g(x)\}$
2)$\{\neg d(x), \neg g(x)\}$ keine Programmklausel !!
3) $\{d(flipper)\}$,$\{i(flipper)\}$ also zwei Klauseln
4) $\{\neg i(x), l(x)\}$
Resolutionsbeweis der Behauptung:

$$(S)\ \frac{(1)\ \{\neg l(x), g(x)\} \qquad (2)\ \{\neg d(x), \neg g(x)\}}{(5)\ \{\neg l(x), \neg d(x)\}}$$

$$(Rob)\ \frac{(5)\ \{\neg l(x), \neg d(x)\} \qquad (3)\ \{d(flipper)\}}{(6)\ \{\neg l(flipper)\}}$$

$$(Rob)\ \frac{(3)\ \{i(flipper)\} \qquad (4)\ \{\neg i(x), l(x)\}}{(7)\ \{l(flipper)\}}$$

$$\frac{(6)\ \{\neg l(flipper)\} \qquad (3)\ \{l(flipper)\}}{\{\ \}\ \text{oder}\ \Box}$$

Problem: Wie finde ich möglichst schnell die zu schneidenden Klauseln und Literale.

Beispiel 2.36 Zeige die Widersprüchlichkeit der Aussage aus dem Paradoxon von B. Russel
,, Der Barbier rasiert eine Person genau dann, wenn sie sich nicht selbst rasiert."
PL1 : ∀X: (rasiert(barbier,X) $\leftrightarrow \neg$ rasiert (X,X))
Klauselform : {¬rasiert(barbier,X), ¬rasiert(X,X)}, {rasiert(Y,Y),rasiert(barbier,Y)}
$\sigma = \{X \mid \text{barbier}, Y \mid \text{barbier}\}$
Es folgt dann hieraus:
{¬rasiert(barbier,barbier),rasiert(barbier,barbier)}
Der Widerspruch wird ohne Faktorisierung nicht gefunden, da jede Klausel aus zwei Literalen besteht.

2.6 Bemerkungen zu PROLOG und PL1

Im Prinzip arbeitet jeder PROLOG - Interpreter nach 2.5 korrekt und widerlegungsvollständig, denn die PROLOG - Interpretationsregel ist ein Spezialfall der Schnittregel:
PROLOG :
$P :- A_1, \ldots, A_n$

?- $P, C_1, \ldots, C_m$
———————————

?- $A_1, \ldots, A_n, C_1, \ldots, C_m$

Im Resolutionskalkül:

$A_1 \wedge \ldots \wedge A_n \rightarrow P \qquad P \wedge C_1 \wedge \ldots \wedge C_m \rightarrow \text{false}$
———————————————————-

$A_1 \wedge \ldots \wedge A_n \wedge C_1 \wedge \ldots \wedge C_m \rightarrow \text{false}$

Aber die Suchstrategien ,,Klauseln von oben nach unten" und ,,Literale von links nach rechts" werden eventuell notwendige Resolutionen nicht finden und die Operation $\wedge$ ist nicht mehr kommutativ und idempotent!

Beispiel 2.37

"Fakten"
p(a,c).
p(c,b).
"Regeln"
p(X,Z) :- p(X,Y), p(Y,Z).
p(X,Y) :- p(Y,X).

Der Interpreter läuft bei jeder (!) Reihenfolge dieser Klauseln in eine Schleife !

Beispiel 2.38

PROLOG - Interpretation als Resolution

$a :- b, c, d.$	$a \vee \neg b \vee \neg c \vee \neg d$
$a :- e, f.$	$a \vee \neg e \vee \neg f$
$b :- f.$	$b \vee \neg f$
$e.$	e
$f.$	f
$a :- f.$	$a \vee \neg f$
——–	
?-$a, e.$ *)	$\neg a \vee \neg e$ (negierte Anfrage !!)
?-$b, c, d, e.$	$\neg b \vee \neg c \vee \neg d \vee \neg e$
?-$f, c, d, e.$	$\neg f \vee \neg c \vee \neg d \vee \neg e$
?-$c, d, e.$	$\neg c \vee \neg d \vee \neg e$
Keine Unifikation möglich $\Rightarrow$ Backtracking zu *.) !!	
?-$e, f, e.$	$\neg c \vee \neg f \vee \neg e$
?-$f, e.$	$\neg f \vee \neg e$
?-$e.$	$\neg e$
?-Leere Klausel, also Antwort: Yes	

Es gibt Sprachelemente in PROLOG, die PL1 verlassen, z.B. E/A - Operationen, Arithmetik, "Prädikate",die andere Prädikate manipulieren(z.B. assert(),retract(), sie sind notwendig zur Erstellung lernfähiger Programme) und den "cut"- Operator.

Beispiel 2.39
$a :- b, c.$
$b.$
(Schlechter Programmierstil. Besser: Fakten vor Regeln, da dann Schleifen vermeidbar sind.)
$c :- d, !, e.$
$c.$
$d.$
$b :- f.$ $\leftarrow$ Wird ohne den cut - Operator niemals durchlaufen!!
$f.$

?-a.
?-b, c.
?-c.
?-$d, !, e$.
?-$!, e$. keine Alternativen für c testen
?-e. $\leftarrow$ Sackgasse, daher Backtracking zur zweiten Anfrage!
?-f, c.
?-c.
?-$d, !, e$.
?-$!, e$.
?-e. keine Alternative, also endgültige Antwort: No, obwohl a logische Folgerung des Programmes ist.

Aber: f könnte mit "assert(c)" das Faktum c. hinzufügen. Dann würde der Interpreter "c" und damit "a" verifizieren.

Beispiel 2.40 Gebrauch und Mißbrauch des cut - Operators

anzahl_der_Eltern(adam,0):-!.
anzahl_der_Eltern(eva,0):-!.
anzahl_der_Eltern(X,2).
?-anzahl_der_Eltern(eva,2) $\rightarrow$ yes.
Alternative:
anzahl_der_Eltern(adam,N):-!, N=0
anzahl_der_Eltern(eva,N):-!, N=0
anzahl_der_Eltern(X,2):-!.
?- anzahl_der_Eltern(X,0).
X = adam,
X = eva
?-anzahl_der_Eltern(X,Y).
$\rightarrow$ .. X = _33, Y = 2.
Noch besser:
mensch(adam).
mensch(eva).
mensch(kain).
mensch(abel).
anzahl_der_Eltern(adam,0).
anzahl_der_Eltern(eva,0).
anzahl_der_Eltern(X,2):-mensch(X), X $\neq$ adam, X $\neq$ eva.

Beispiel 2.41 Die Negation "not X" hat in PROLOG nicht die übliche Semantik! Sie wird definiert gemäß:
not X :- call(X),!,fail.
not X.
mit call(X): Metaprädikat(nicht mehr PL1) und
fail: nullstelliges Prädikat, das stets zum Backtracking führt.
a.
b.
?- c $\rightarrow$ no.
?- not c. $\rightarrow$ yes.
?- not a. $\rightarrow$ no.
PROLOG beweist Anfragen also unter einer "closed world assumption".

2.7 Repräsentation verschiedener Wissensarten in PL1

Es wird nicht verlangt, daß die Darstellung in PROLOG erfolgt. Man vergleiche auch [Ric89].

Für alle zu repräsentierenden Objekte sind eindeutige Namen (=Terme) einzuführen, z.B. bopriek,..
Sogar: Verschiedene Namen bezeichnen verschiedene Objekte.
Dies ist nicht selbstverständlich, so bezeichnet etwa "Abendstern" und "Morgenstern" dasselbe Objekt, nämlich den Planeten Venus, und die Gleichheit muß erst erkannt werden (vgl. Bem. 4.4 iii).
Eigenschaften von Objekten werden i.a. durch Prädikate repräsentiert, z.B. rot(rose_1),gelb(teerose_1),..
Besser ist es aber, auch Eigenschaften als Objekte aufzufassen, sie zu *vergegenständlichen*, um über sie Aussagen machen zu können, z.B. farbe(rose_1,rot),..
Aber auch "Farbe" kann, wie etwa Größe, als Eigenschaft angesehen werden, die verschiedene *Werte* annehmen kann, z.B. rot,gelb,blau (s.u.)
Häufig wird daher das Wissen über Attribute von Objekten als Tripel abgelegt.

fakt(objekt,attribut,wert) oder wahr(objekt,attribut,wert)

Beispiel 2.42

z.B.: wahr(rose_1,bluetenfarbe,rot).
wahr(rose_1,laenge,30).
wahr(bopriek,gnaselt,drauslapümpfe).

Diese einheitliche Struktur des Faktenwissens ermöglicht die einfache Beantwortung von Anfragen:

?-wahr(rose_1,bluetenfarbe,rot).
?-wahr(rose_1,bluetenfarbe,X).
?-wahr(rose_1,Y,_).
?-wahr(Z,_,_).

Auf diese Weise können z.B. auch Kontexte gemäß Def 1.15 abgelegt werden:

Beispiel 2.43 Als Beispiel diene das Wissen über den Farbkreis:

wahr(grundfarbe,rot,ja).
wahr(grundfarbe,gelb,ja).
wahr(grundfarbe,blau,ja).
wahr(mischfarbe,violett,ja).
wahr(mischfarbe,orange,ja).
wahr(mischfarbe,gruen,ja).
wahr(mischfarbe,weiss,ja).
.
.

wahr(rot,wellenlaenge,von_bis(..,..)).
Durch Angabe eines Bereiches (für die Wellenlänge) könnte etwa auch *ungenaues Wissen* dargestellt werden. Für einen präziseren Umgang mit ungenauem Wissen wurde die *Fuzzy-Logik* (vgl. Kap 7.3) entwickelt.
.
.
farbmischung(rot,gelb,orange). oder *Listen:*
farbmischung([rot,gelb],orange).
komplementärfarbe(X,Y):-farbmischung(X,Y,weiß); farbmischung(Y,X,weiß).

Beispiel 2.44 Beispiel für Individualkonzepte, Klassenkonzepte, Gruppenkonzepte
instanz_von(rot,grundfarbe).
.
.
.
instanz_von(weiss,mischfarbe).
is_a(grundfarbe,farbe).
is_a(mischfarbe,farbe).
gruppenelement(grundfarbe,farbtyp).
gruppenelement(mischfarbe,farbtyp).

Also: *Begriffliches Wissen* kann gut in PL1 oder sogar PROLOG repräsentiert werden.
Ebenso Wissen über *regelhafte Zusammenhänge* oder *einschränkende Bedingungen*, etwa
die Transitivität der is_a - Relation: is_a(K1,K2) :- is_a(K1,K3),is_a(K3,K2).
oder die Vererbung von Eigenschaften auf Unterbegriffe und Instanzen:
wahr(K1,Attribut,Wert):-wahr(K2,Attribut,Wert), is_a(K1,K2).
wahr(Objekt,Attribut,Wert):-wahr(Klasse,Attribut,Wert),instanz_von(Objekt,Klasse).

Widersprüchliches Wissen ist im Rahmen von PL1 nicht zu repräsentieren, da sich aus einem Widerspruch jedes Faktum folgern läßt.
Es gibt aber Erweiterungen von PL1 zu sogenannten mehrwertigen Logiken, die neben den Wahrheitswerten „wahr" und „falsch" (mindestens) einen weiteren Wahrheitswert vorsehen, der als „inkonsistent" zu interpretieren ist. Dies erfordert eine neue Definition der formalen Semantik!
Die Repräsentation von *unvollständigem Wissen* ist in PL1 möglich:
¬komplementärfarbe(blau,rot).
∃X: komplementärfarbe(blau,X).
Probleme treten auf, wenn von einer abgeschlossenen Welt ausgegangen wird, wie etwa in PROLOG, die Wissensbasis aber unvollständig ist.
Auch hierfür wird eine geeignete Erweiterung von PL1 zu einer *epistemischen Logik* vorgeschlagen. Sie ermöglicht Aussagen darüber, was einer Wissensbasis bekannt und was unbekannt ist. Dazu wird ein neuer einstelliger Junktor K (für „kown") eingeführt mit der Semantik „es ist bekannt, daß".

Mit der obigen Wissensbasis ist dann etwa
?- K komplementärfarbe(blau,orange)
mit „nein" zu beantworten und als „es ist unbekannt, ob orange die Komplementärfarbe von blau ist" zu interpretieren.
Die Anfrage :
?- $\exists X$: komplementärfarbe(blau,X).
ergibt dann die Antwort „yes" während
?- $\exists X$: K komplementärfarbe(blau,X).
als Antwort „no" liefern muß.
Übrigens beschränkt man sich meistens darauf, den Junktor K nur in Anfragen zu verwenden, in der Wissensbasis liegen nur Formelklauseln aus PL1.
Die Repräsentation von *unsicherem Wissen* erfolgt meistens durch Hinzunahme von Sicherheitsfaktoren
$s \in [0,1]$ zu den Fakten (und Regeln!):

Also etwa Fakten als 4-Tupel

wahr(bopriek,gnaselt,drauslapümpfe,0.9).

oder Regeln

fakt(X,S):-wahr(X1,S1),..,wahr(XN,SN),SR = 0.75,

wobei X durch X1 bis XN und S durch S1 bis SN, sowie SR, bestimmt wird.

Bei der Berechnung des abgeleiteten Sicherheitsfaktors S werden unterschiedliche Wahrscheinlichkeitsmaße angewandt (vgl. [Som92]).

Die Darstellung *prototypischen Wissens* mit Ausnahmen ist in der klassischen PL1 nicht möglich, da deren Behandlung *nicht monotone Inferenzmechanismen* erfordert.

Definition 2.45 Ein Ableitungsoperator $\vdash_{\mathcal{R}}$ heißt *monoton*, falls aus $\Sigma \vdash_{\mathcal{R}} F$ und $\Sigma \subseteq \Sigma'$ stets $\Sigma' \vdash_{\mathcal{R}} F$ folgt, andernfalls heißt $\vdash_{\mathcal{R}}$ *nicht-monoton*.

Die nach Definition 2.30 definierten Ableitungsoperatoren sind stets monoton!

Oft wird zur Darstellung prototypischen Wissens ein neuer einstelliger Junktor M (für *Modaloperator*) eingeführt mit folgender Semantik:
MF ist genau dann wahr, wenn aus der aktuellen Wissensbasis die Formel ¬F nicht abgeleitet werden kann.
M bedeutet also etwa ,, es ist konsistent anzunehmen, daß"

Beispiel 2.46
$\forall X\ (vogel(X) \wedge M$ kann_fliegen$(X) \rightarrow$ kann_fliegen$(X))$.
$\forall X\ (pinguin(X) \rightarrow \neg$kann_fliegen$(X))$.
Für einen (normalen) Vogel X kann dann kann_fliegen(X) abgeleitet werden. Ist aber zusätzlich bekannt, daß X ein Pinguin ist, so kann dies nicht mehr abgeleitet werden (Nicht-Monotonie!)

Diese Thematik wird auch in den Büchern [Sie90], [Kre91], [Kro87] und [Sme88] des Literaturverzeichnisses behandelt.

Die Verwaltung solcher Default - Aussagen und Neuberechnung von Inferenzen bei Veränderung der Wissensbasis ist Aufgabe sogenannter *Begründungsverwaltungssysteme* (,,truth maintenance systems").

Ebenfalls eine Erweiterung der PL1 erfordert die Darstellung von *Wissen über Modalitäten*. Auch hier werden neue *Modaloperatoren* eingeführt *(Modallogik)*.

Definition 2.47 Eine *modallogische Sprache* ML besteht aus einer prädikatenlogischen Sprache PL durch Hinzunahme zweier einstelliger logischer Junktoren "$\Box$" und "$\Diamond$". Mit F sind dann auch ($\Box$F) und ($\Diamond$F) Formeln von ML. Bzgl. der Semantik wird
($\Box$F) als "F gilt notwendigerweise" und
($\Diamond$F) als "F gilt möglicherweise" interpretiert.

Man verabredet noch, daß ($\Diamond$F) äquivalent zu $(\neg(\Box(\neg F)))$ ist. Eine exakte Semantik wird in §9 von [Ric89] angegeben.

Definition 2.48 Eine *Kripkeinterpretation* (*Kripkestruktur*) für ML ist von der Form $I = (G, (I_g, U_g)_{g \in G})$, wobei $(G, \leq)$ ein gerichteter Graph ist und (I_g, U_g) für jedes $g \in G$ eine gewöhnliche prädikatenlogische Interpretation ist.
$U = \bigcup_{g \in G} U_g$ heißt das *Universum von I.*
Für jedes $g \in G$ stellt dann U_g eine (Modell-)Welt dar und $g_1 \leq g_2$ bedeutet, daß U_{g2} von U_{g1} aus (direkt) *erreichbar* ist.
Ist dann U_g die aktuell beschriebene Welt, so wird $(\Box F)$ als "wahr" angesehen, wenn F in allen Welten wahr ist, die von U_g aus erreichbar sind. Dagegen wird $(\Diamond F)$ genau dann als "wahr" angesehen, wenn F in mindestens einer von U_g aus erreichbaren Welt wahr ist.

Interpretiert man die U_g als Welten zu bestimmten Zeiten, so gelangt man zur *temporalen Logik* oder *dynamischen Logik*, einer Erweiterung der PL1, die zur Beschreibung von *Ereignissen* bzw. *Handlungen* dient (vgl. Bsp. 1.2.)
Ihre wesentliche Eigenschaft ist, daß der Wahrheitswert einer Aussage als zeitabhängig betrachtet wird.

Beispiel 2.49 Der folgende Ausschnitt aus einem (PASCAL-) Programm werde zur Laufzeit zu drei verschiedenen diskreten Zeiten betrachtet.

```
... a:=3; b:=-3; c:=0; c:=b; b:=b - a;
                     ↑     ↑        ↑
                     t0    t1       t2
```

Der Wahrheitswert der Formel $F \equiv (a + b = c \wedge a \geq 0) \rightarrow (b \geq 0)$ zu diesen Zeiten ist jeweils false, true bzw. wieder false.
(Dynamische Logik wird u.a. bei der Verifikation von Programmen eingesetzt !)

Neben den in Definition 2.47 angegebenen Modaloperatoren mit der neuen Bedeutung *($\Box F$)* "F gilt immer (in der Zukunft)" und *($\Diamond F$)* "F gilt irgendwann (in der Zukunft) einmal" gibt es in der temporalen Logik weitere einstellige und zweistellige Junktoren:
($\circ F$) "F gilt zu dem Zeitpunkt, der unmittelbar auf den Referenzzeitpunkt folgt".
F *atnext* G "F wird gelten, wenn G zum nächstenmal gilt".
F *until* G "F gilt zu allen zukünftigen Zeitpunkten bis zu dem Zeitpunkt, an dem G gilt".
F *while* G "F gilt, solange G gilt".

Es gelten aber folgende Äquivalenzen :
F until $G \leftrightarrow [G$ atnext $(F \rightarrow G)] \wedge \circ \Diamond G$
F while $G \leftrightarrow \neg G$ atnext $(F \rightarrow \neg G)$

Damit lassen sich etwa folgende Aussagen formalisieren :
$F \rightarrow (\Box G)$ "Wenn F gilt, dann gilt in Zukunft immer G".
$F \rightarrow (\Diamond G)$ "Wenn F gilt, dann wird irgendwann einmal G gelten".
$\Diamond(F \wedge \circ\neg F)$ "Irgendwann einmal wird F gelten und sofort danach ¬F".
$\Diamond\Box F$ "Irgendwann einmal wird F beständig gelten".
$\Box\Diamond F$ "F wird unendlich oft gelten".
Mit analogen Junktoren kann man auch Aussagen über Gültigkeiten von Formeln in der Vergangenheit darstellen. In einfachen Anwendungen kann man zeitliche Aspekte auch durch zusätzliche Argumente in den Prädikaten der PL1 berücksichtigen.

Beispiel 2.50 *Zeitliche Aspekte in Prädikaten*
$s(X,T)$: "Der Roboter Robbie sieht das Objekt X zum Zeitpunkt T"
$g(X,T)$: "Robbie greift X zum Zeitpunkt T"
$T1 < T2$: "Der Zeitpunkt T1 ist früher als T2"

$\forall T \exists X\, s(X,T)$: "Robbie sieht immer irgendetwas"
$\forall T \forall X (s(X,T) \rightarrow \exists T1((T < T1) \wedge g(X,T1)))$:
"Wenn Robbie etwas sieht, ergreift er es später einmal."

2.8 Prädikatenlogik höherer Stufen

Man unterscheidet eine Hierarchie von Logikkalkülen verschiedener Stufen, die mit höherer Stufe immer mächtiger werden:

$\vdots$

PL2

$\boxed{\text{PL1}}$

Hornklauseln

Aussagenlogik

Die Unterschiede der verschiedenen Stufen der Prädikatenlogik liegen in den Werten, die von quantifizierten Variablen angenommen werden dürfen.
1.Stufe : $\forall$x Werte von x aus dem Universum U.
2.Stufe : $\forall$x Werte von x aus U oder $\mathcal{P}$(U).
3.Stufe : $\forall$x Werte von x aus U oder $\mathcal{P}$(U) oder $\mathcal{P}(\mathcal{P}$(U)).
.
.
.

Beispiel 2.51 *für 2.Stufe : Induktionsaxiom für $I\!N_0$*

$$\forall T(T \subseteq I\!N_0 \wedge 0 \in T \wedge (\forall n(n \in T \rightarrow \mathrm{succ(n)} \in T)) \rightarrow T = I\!N_0)$$
$$\uparrow$$
$$T \in \mathcal{P}(I\!N_0)$$

Kapitel 3

Repräsentation durch Produktionsregeln

3.1 Produktionsregelsysteme

Man vergleiche zu diesem Kapitel auch [Rei91], [Ric89] §4, [CM85], [KLN87], [Nil82] und [Win89] aus dem Literaturverzeichnis.

Definition 3.1 Ein *Produktionsregelsystem* (WB, R, S) besteht aus
- einer *dynamischen Wissensbasis* $WB = \{F_1, \dots, F_n\}$ von *Fakten* F_i
- einer Menge $R = \{R_1, \dots, R_m\}$ von *(Produktions-)Regeln* R_j und
- einer *Kontrollstrategie* S.

Bemerkung 3.2

i) In der Wissensbasis wird das Faktenwissen über einen bestimmten Wissensbereich abgelegt. Es kann durch die Regeln verändert werden, daher ist die Wissensbasis dynamisch.

ii) Das Repräsentationsformat der Fakten ist nicht fest vorgeschrieben, es können z.B. logische Formeln, Tripel (Objekt, Attribut, Wert) in Form von Listen oder anderen Strukturen sein.

iii) Die Produktionsregeln sind von der Form
(*) WENN Vorbedingung DANN Folgerung / Aktion.

Dabei ist die Vorbedingung eine logische Bedingung, die von den aktuellen Fakten der Wissensbasis erfüllt sein kann oder nicht. Folgerung bzw. Aktion ist i.a. eine Prozedur, die z.B. die dynamische Wissensbasis durch Hinzufügen oder Löschen von Fakten verändert oder Ein- bzw. Ausgaben durchführt.

iv) Die Bedeutung von (*) ist also: Wenn die Wissensbasis die Vorbedingung erfüllt, dann *darf* die Folgerung gezogen werden bzw. die Aktion ausgeführt werden. Ob dieses dann auch tatsächlich geschieht, darüber entscheidet die Kontrollstrategie.

v) Zur Erzielung lernfähiger Programme ist es manchmal auch möglich durch Aktionen die Regeln zu verändern.

Beispiel 3.3 Gegeben sind vier Würfel $W1, W2, W3, W4$, die auf einem Tisch liegen. Sie sollen zu dem Turm "$W3$ liegt auf $W4$ auf $W2$ auf $W1$ auf Tisch" aufeinandergelegt werden.
$WB = \{F_1, \dots, F_{12}\} = \{$wuerfel($W1$),..,wuerfel($W4$),liegt_auf($W1$,Tisch),..,liegt_auf($W4$,Tisch), frei($W1$),..,frei($W4$)$\}$

$R = \{R_1\} = \{$WENN wuerfel(X),wuerfel(Y),X $\neq$ Y,frei(X),frei(Y),liegt_auf(X,Tisch)
DANN loesche_faktum(frei(Y)),loesche_faktum(liegt_auf(X,Tisch)),
ergaenze_faktum(liegt_auf(X,Y))$\}$

Durch wiederholte Anwendung dieser Regel (Kontrollstrategie S : "Wende $R1$ an") ist die Wissensbasis in den Zielzustand WB' zu überführen.
$WB' = \{$wuerfel($W1$),..,wuerfel($W4$),frei($W3$),liegt_auf($W3, W4$),..,liegt_auf($W1$,Tisch)$\}$.
Dieses ist die *Endebedingung*, die dem Regelsystem übergeben wird.

Das allgemeine Verhalten eines beliebigen Produktionsregelsystems wird durch den folgenden Algorithmus beschrieben :

Definition 3.4 Sei (WB, R, S) ein Regelsystem und Z eine Endebedingung. Der *Kontrollalgorithmus* lautet dann :

1. DATEN := WB;
2. until DATEN erfüllt Z do
   begin
      Bestimme die *Konfliktmenge* K aller ausführbaren Regeln $R_i \in R$.
      Ist $K = \emptyset$, dann melde einen Mißerfolg und beende den Algorithmus.
      Wähle gemäß S eine (oder mehrere) Regel(n) aus K aus und wende sie auf DATEN an.
   end.

Bemerkung 3.5

i) Die Schleife in Def. 3.4. heißt auch "recognize - act - cycle".

ii) Die Kontrollstrategie S, deren Ziel es ist, $K \neq \emptyset$ auf $\mid K \mid= 1$ zu reduzieren, bestimmt im wesentlichen die Effizienz des Regelsystems. Die wichtigsten Strategien sind:

   1 Wähle eine Regel $R_i \in K$ zufällig aus [*Nicht-Determinismus!*] *(while-1.-Strategie)*

   2 Wende alle Regeln aus K in feststehender Reihenfolge [oder *parallel*] an *(while-all-Strategie)*

   3 Durch *Meta-Regeln* wird festgelegt, welche Regel der anderen vorzuziehen ist.

   4 Einführung einer Präordnung $\leq$ (Priorisierung) auf den Regeln.

      a) Es wird diejenige Regel $R_i \in K$ ausgewählt, die bzgl. $\leq$ am kleinsten (am größten) ist. Z.B. Reihenfolge des Aufschreibens bei PROLOG !

      b) Es werden alle Regeln R_i aus K entfernt, zu denen ein $R'_j \in K$ mit $R'_j < R_i (R_i < R'_j)$ existiert. Danach wird 1. oder 2. angewandt.

iii) Einige Beispiele für Präordnungen auf Regelmengen sind:

   1 Umfang der Vorbedingung, d.h. Anzahl der einzelnen Bedingungen (kann eingeschränkt werden auf vergleichbare Bedingungen).

   2 Spezifität der Vorbedingung, d.h. wenn die Vorbedingung von $R1$ die Vorbedingung von $R2$ impliziert, dann wird $R1$ vorgezogen.

   3 Aktualität, d.h. erstmaliges / letztmaliges "Feuern" der Regel bestimmt ihre Priorität.

   4 Zuverlässigkeit, falls Regeln mit Sicherheitsfaktoren versehen sind.

   Weitere Präordnungen entstehen durch Ordnungen auf den Fakten:
   Sämtliche Fakten werden in einer Prioritätsliste angeordnet (vgl. PROLOG !). Die Priorität einer Regel ergibt sich aus den Prioritäten der Fakten, auf die die Vorbedingung Bezug nimmt(vgl. OPS5).

iv) Die Regelmenge R kann gemäß $R = Rk_1 \cup \ldots \cup Rk_e$ in disjunkte *Regelklassen* zerlegt werden, von denen jeweils nur eine oder einige *aktiv* sind. Meta - Regeln können dann jeweils Klassen ein- oder ausschalten.

v) Ebenfalls durch *heuristisches Meta-Wissen* läßt sich die Effizienz bei der Berechnung von K steigern:
   - leichter zu prüfende Bedingungen sind zuerst zu prüfen,
   - allgemeinere sind vor spezielleren Bedingungen zu prüfen,
   - unwahrscheinlichere Bedingungen sind zuerst zu prüfen.

Bei der Berechnung der Konfliktmenge unterscheidet man zwei unterschiedliche Verfahren, ebenso bei der Anwendung der Regeln:

Definition 3.6 *Sei (WB,R,S) ein Regelsystem, Z eine Endebedingung.*

i) Läßt sich der Regelinterpreter bei der Berechnung von K von WB leiten, spricht man von *datengesteuerter* Regelanwendung oder *Vorwärtsverkettung*, orientiert er sich dagegen an Z, spricht man von *zielorientierter* Regelanwendung oder *Rückwärtsverkettung.*

ii) Die Anwendung der ausgewählten Regel(n) geschieht *rückrufbar*, wenn alle Veränderungen von WB und alle nicht gewählten Alternativen protokolliert werden, um sie eventuell später zu verfolgen*(Backtracking)*, andernfalls *nicht rückrufbar*.

Beispiel 3.7 *Fortsetzung von 3.3.*
Vorwärtsverkettung (die Variablenbelegung ist hier willkürlich gewählt, um Sackgassen zu erzeugen !!)

Schritt	Variablenbelegung	Löschungen in WB	Ergänzungen in WB
1	X=W2,Y=W1	frei(W1),liegt_auf(W2,Tisch)	liegt_auf(W2,W1)
2	X=W3,Y=W2	frei(W2),liegt_auf(W3,Tisch)	liegt_auf(W3,W2)
3	X=W4,Y=W3	frei(W3),liegt_auf(W4,Tisch)	liegt_auf(W4,W3)

Keine Regel mehr anwendbar, Zielzustand nicht erreicht ⇒ Backtracking zu 3!
In Schritt 3 gibt es keine alternative Variablenbelegung, daher ist sogar Backtracking zu 2 nötig.

Schritt	Variablenbelegung	Löschungen in WB	Ergänzungen in WB
$2'$	X=W4,Y=W3	frei(W3),liegt_auf(W4,Tisch)	liegt_auf(W4,W3)

Backtracking zu $2'$

Schritt	Variablenbelegung	Löschungen in WB	Ergänzungen in WB
$2''$	X=W4,Y=W2	frei(W2),liegt_auf(W4,Tisch)	liegt_auf(W4,W2)
$3''$	X=W3,Y=W4	frei(W4),liegt_auf(W3,Tisch)	liegt_auf(W3,W4)

Rückwärtsverkettung (die Variablenbelegung ist jeweils eindeutig !)

Schritt	Variablenbelegung	Löschungen in WB'	Ergänzungen in WB'
1	X=W3,Y=W4	frei(W4),liegt_auf(W3,Tisch)	liegt_auf(W3,W4) WB''
2	X=W4,Y=W2	frei(W2),liegt_auf(W4,Tisch)	liegt_auf(W4,W2) WB'''
3	X=W2,Y=W1	frei(W1),liegt_auf(W2,Tisch)	liegt_auf(W2,W1) $WB'''' = WB$

$R1$ nicht mehr anwendbar, aber es ist ein Zustand erreicht, der mit WB übereinstimmt, d.h. das Problem ist gelöst.

3.2 Darstellung verschiedener Wissensarten

Dieses Thema wird auch in [Rei91] des Literaturverzeichnisses betrachtet

In Produktionssystemen werden *Objekte* (Individualkonzepte) mit ihren *Attributen* in der dynamischen Wissensbasis dargestellt und daher in einem eigenen Repräsentationsformat (z.B. logische Formeln, semantische Netze, Frame-Strukturen).
Dagegen werden *regelhafte Zusammenhänge* und *einschränkende Bedingungen* zwischen den Objekten leicht direkt durch Produktionsregeln dargestellt:

WENN instanz_von(X,fluegel),instanz_von(Y,vogel),
hat_teil(Y, X),hat_eigenschaft(X,gebrochen),
hat_eigenschaft(Y,flugfähig)
DANN loesche_faktum(hat_eigenschaft(Y,flugfähig)),
ergaenze_faktum(not hat_eigenschaft(Y,flugfähig)).

Stellt man das "negative" Wissen not hat_eigenschaft(Y,flugfähig) aber durch hat_eigenschaft(Y,flugunfähig) dar, so kann man die Konsistenz der WB an dieser Stelle durch eine eigene Produktionsregel überwachen:

WENN hat_eigenschaft(Y,flugfähig), hat_eigenschaft(Y,flugunfähig)
DANN melde("Widerspruch bei Flugunfähigkeit von Y")

Auch die Behandlung von *Prototypen* mit *Default - Werten* und *Ausnahmen* ist durch Regeln leicht realisierbar.

WENN instanz_von(X,vogel), not ausnahme(X,flugfähig)
DANN ergaenze_faktum(hat_eigenschaft(X,flugfähig))

Diese Regeln repräsentieren den (proto-)typischen flugfähigen Vogel.
Die korrekte Behandlung von Ausnahmen garantiert dann z.B.

WENN instanz_von(X,pinguin)
DANN ergaenze_faktum(ausnahme(X,flugfähig)).

Auch die Beschreibung von *Klassenkonzepten* (und analog *Gruppenklassen*) lassen sich durch Regeln darstellen, etwa
"Ein Vogel ist ein eierlegendes Tier mit zwei Flügeln"

WENN instanz_von(X,vogel)
DANN ergaenze_faktum(hat_eigenschaft(X,eierlegend)),
ergaenze_faktum(is_a(vogel,tier)),
ergaenze_faktum(instanz_von(linker_fluegel(X),fluegel),
ergaenze_faktum(instanz_von(rechter_fluegel(X),fluegel),
ergaenze_faktum(hat_teil(X,linker_fluegel(X))),
ergaenze_faktum(hat_teil(X,rechter_fluegel(X))).

In einigen Repräsentationsformaten sind auch irgendwann einmal die Klassennamen "tier" und "vogel" zu initialisieren. (Ähnlich wie die Variablendeklaration in prozeduralen Sprachen!)
Die *Vererbung von Eigenschaften* eines Klassenkonzeptes auf seine Unterklassen läßt sich z.B. wie folgt darstellen :

WENN hat_eigenschaft($K1, E$), is_a($K2, K1$),
DANN ergaenze_faktum(hat_eigenschaft($K2, E$)).

Die Repräsentation *unsicheren Wissens* verlangt zunächst die Darstellung von Sicherheitsfaktoren für die Fakten in der WB. Dann können auch den Regeln solche Fakten zugeordnet werden.
Das folgende Beispiel entstammt dem Expertensystem *MYCIN* zur Diagnose bakterieller Infektionen.

WENN instanz_von(X,organismus),
hat_eigenschaft(X,gram_negativ),
hat_eigenschaft(X,staebchenfoermig),
hat_eigenschaft(X,anaerob),
DANN ergaenze_faktum(instanz_von(X,bakterium)).
SICHERHEITSFAKTOR = 0.6

Der dem "neuen" Faktum zugewiesene Sicherheitsfaktor errechnet sich wie folgt :

$$S_{\text{neu}} = S_{\text{alt}} + S_{\text{regel}} * S_{\text{vorbed.}} * (1 - S_{\text{alt}}).$$

Dabei ist
S_{alt} : der "alte" Sicherheitsfaktor des Faktums, falls dies schon in der Wissensbasis existiert, sonst Null.
S_{regel} : 0.6
$S_{vorbed.}$: der Sicherheitsfaktor der gesamten Vorbedingung, der sich als Produkt der vier Faktoren der beteiligten Fakten ergibt.

Ein *Widerspruch* zwischen Produktionregeln besteht, wenn zwei Regeln in der Konfliktmenge zur Herleitung widersprüchlicher Fakten benutzt werden können. Diese Situation ist zu vermeiden, was aber oft nur schwer überprüfbar ist!

Kapitel 4

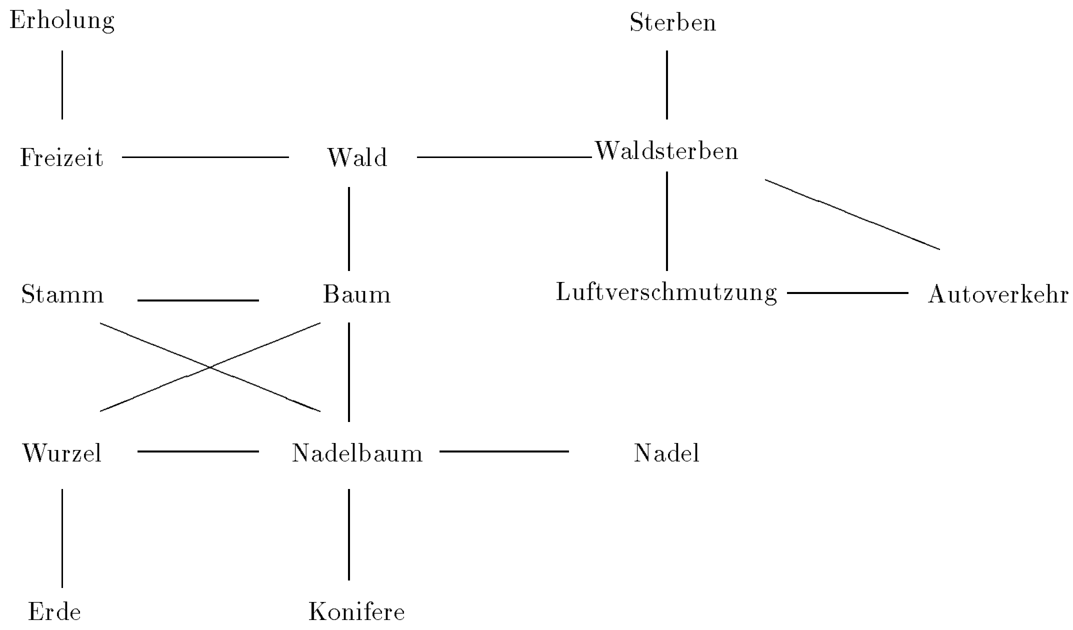
Semantische Netze

4.1 Einführung

In den Büchern [Rei91], [Ric89], [CM85], [Win89] werden ebenfalls semantische Netze behandelt.

Semantische Netze oder assoziative Netze entstanden als Repräsentationsformate experimenteller Untersuchungen des menschlichen Gedächtnisses durch Kognitionspsychologen. Konzepte, die durch assoziative Beziehungen miteinander verbunden sind, sollen auch im Modell verbunden sein.
Solche Formate, bei denen alle Eigenschaften eines Objektes "unmittelbar" bei diesem Objekt abgespeichert sind, bezeichnet man als *objekt-zentriert* oder *objektorientiert*.

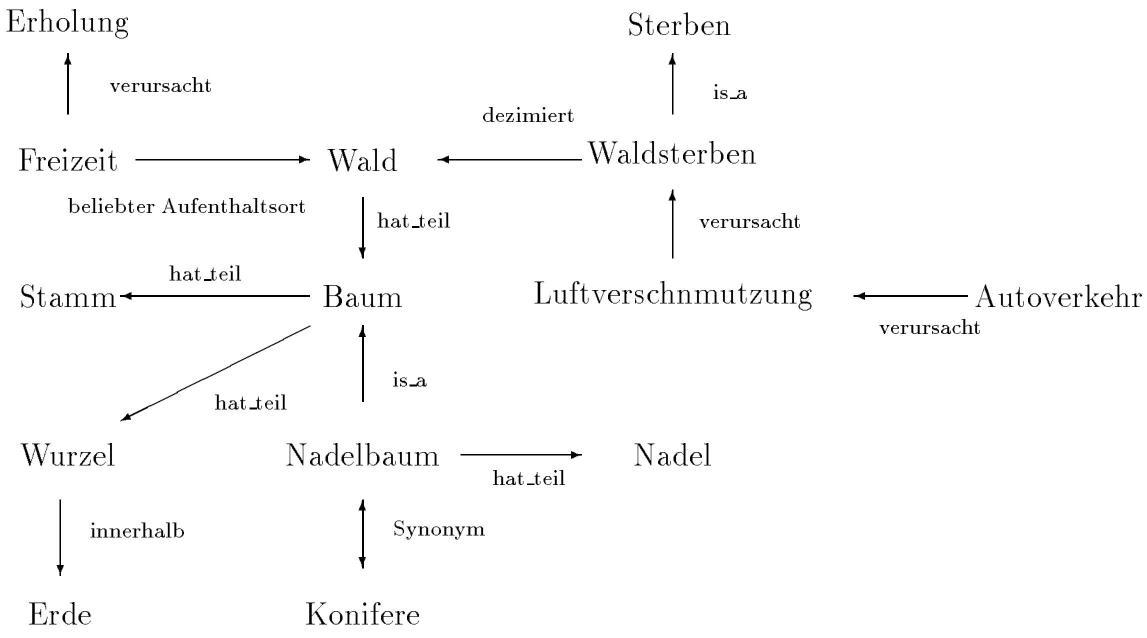
Beispiel 4.1 *Ein assoziatives Netz*



Meistens werden die (ungerichteten) Kanten noch durch Zahlen bewertet, die die "Stärke" der assoziativen Beziehung ausdrücken sollen.

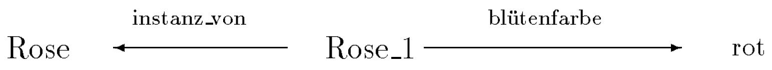
Definition 4.2 Ein *semantisches Netz* (K,σ,T,τ) ist ein endlicher gerichteter Graph, bestehend aus :
- einer Menge K von *Konzepten*, den Knoten des Graphen,
- einer Relation $\sigma \subseteq K \times K$, die die Kanten (a,b) $\in \sigma$ des Graphen beschreibt,
- einer Menge T von *Kantentypen* und
- einer Funktion $\tau : \sigma \rightarrow T$, die jeder Kante einen Typ zuordnet.

Beispiel 4.3
Ein semantisches Netz:



Bemerkung 4.4

i) Es ist bei der Wissensdarstellung mittels semantischer Netze üblich, Eigenschaften zu vergegenständlichen (vgl. 2.7.) und sie somit als Knoten zu repräsentieren, z.B.:



Man unterscheidet daher drei Arten von Knoten: *Eigenschaftsknoten*, *Individualknoten* und *Klassenknoten*, die beiden letzten faßt man zu *Konzeptknoten* zusammen.

ii) In den meisten Fällen werden gewisse *domänenunabhängige* Kantentypen vom System zur Verfügung gestellt, z.B. is_a, instanz_von, hat_teil, verursacht, Synonym. Dann ist ebenfalls ihre Semantik mit definiert, z.B. Symmetrie oder Transitivität bzw. allgemeiner ihre Vererbungsmechanismen (vgl. 4.2).

iii) Man trifft noch folgende Vereinbarungen:

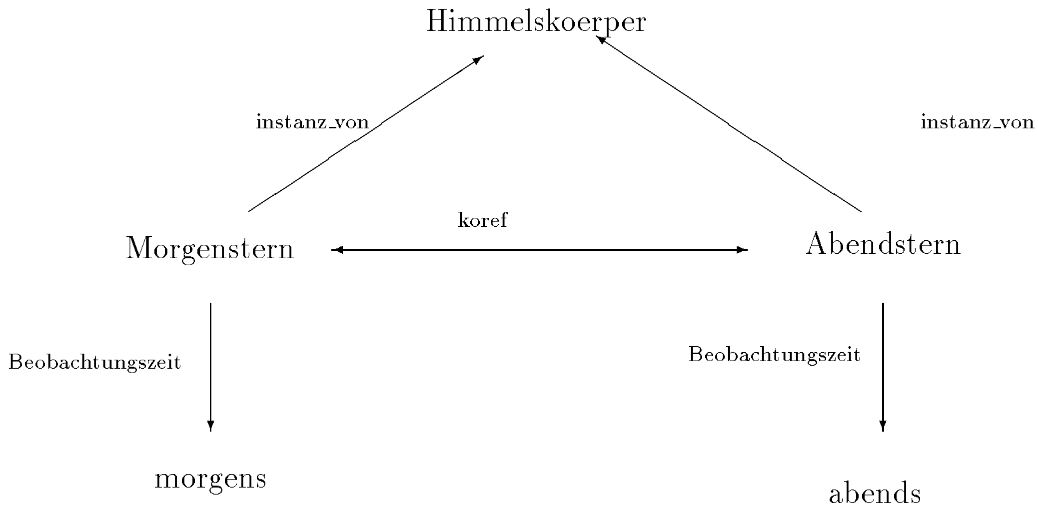
a) Wird ein Individualkonzept dargestellt, dann ist damit seine Existenz behauptet.

b) Klassenkonzepte dürfen aber leer sein, z.B. "Drachen".

c) Alle in einem Netz repräsentierten Aussagen sind implizit konjunktiv verknüpft.

d) Es treten keine zwei Knoten mit demselben Namen auf.

e) Zwei Knoten mit unterschiedlichen Namen sind verschieden. Um die Gleichheit zweier Objekte mit verschiedenen Namen auszudrücken, führt man einen speziellen Kantentyp "koref" ein:



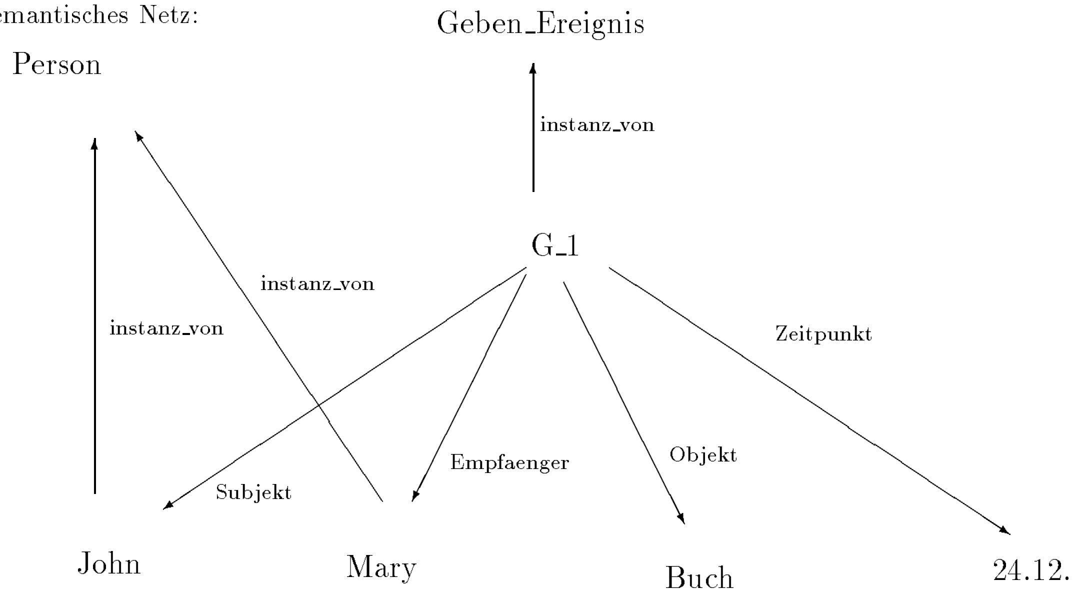
iv) Viele Systeme stellen ein allgemeinstes Klassenkonzept, etwa "Objekt", zur Verfügung. In diesem Fall sind genau diejenigen Konzeptknoten Individualknoten, von denen eine instanz_von - Kante ausgeht, und genau diejenigen Konzeptknoten sind Klassenknoten (außer "Objekt"), von denen eine is_a - Kante ausgeht.

v) Prinzipiell kann man jedes semantische Netz wieder durch logische Formeln darstellen, nämlich die Knoten durch Terme und die Kanten durch zweistellige Prädikate. Umgekehrt kann man (wenn auch mühevoll, vgl. 3.4.2 in [Rei91]) jede Formel aus PL1 auch durch ein semantisches Netz darstellen. Beide Repräsentationsformate sind also gleichmächtig. Man wählt zwischen ihnen nach pragmatischen Gesichtspunkten.

Beispiel 4.5 Mehrstellige Beziehungen zwischen Objekten, die sich in PL1 durch mehrstellige Prädikate ausdrücken lassen, sind auf geeignete zweistellige Beziehungen zurückzuführen, etwa "John gab Mary das Buch am 24.12" in

PL1: geben(John,Mary,Buch,24.12)

als semantisches Netz:



4.2 Vererbungsmechanismus

Der bei jeder Darstellung von Konzepten benötigte Mechanismus zur Vererbung von Eigenschaften wird nun formalisiert. Er stellt einen Teil der Inferenzkomponente für Systeme mit semantischen Netzen dar.

Definition 4.6 Es sei (K, σ, T, τ) ein semantisches Netz. Eine Kante $(a, b) \in \sigma$ heißt *Eigenschaftskante*, wenn a ein Konzeptknoten und b ein Eigenschaftsknoten ist, sie heißt *Beziehungskante*, wenn a und b Konzeptknoten sind.
Die den Kanten jeweils zugordneten Typen heißen entsprechend *Eigenschaftstypen* bzw. *Beziehungstypen*.
Es ist etwa in Bem. 4.4 i) *instanz_von* ein Beziehungstyp und *bluetenfarbe* ein Eigenschaftstyp.

Definition 4.7 Es sei (K, σ, T, τ) ein semantisches Netz. *Ererbte Beziehungen* und *Eigenschaften* werden nicht explizit repräsentiert, sondern werden nach folgenden Regeln (1) bis (6) abgeleitet. Für alle Konzeptknoten k, k', l, l', i alle Eigenschaftsknoten w, alle Eigenschaftstypen e und alle Beziehungstypen $b \notin$ {is_a, instanz_von, may_be_a}.

1) $\text{klasse}(k) \wedge \text{ekante}(e, k, w) \wedge (\text{is\_a}(k', k) \vee k' = k) \rightarrow \text{hat\_eigenschaft}(e, k', w)$

$$\begin{tikzcd} k \arrow[r, "\text{e}"] \arrow[d, dash, "\text{is\_a}"] & w \\ k' & \end{tikzcd} \quad \Rightarrow \quad \begin{tikzcd} k \arrow[r, "\text{e}"] \arrow[d, dash, "\text{is\_a}"] & w \\ k' \arrow[ur, "\text{hat\_eigenschaft}"'] & \end{tikzcd}$$

2) $\text{klasse}(k) \wedge \text{ekante}(e, k, w) \wedge \text{instanz\_von}(i, k) \rightarrow \text{hat\_eigenschaft}(e, i, w)$

$$\begin{tikzcd} k \arrow[r, "\text{e}"] & w \\ k' \arrow[u, "\text{instanz\_von}"'] & \end{tikzcd} \quad \Rightarrow \quad \begin{tikzcd} k \arrow[r, "\text{e}"] & w \\ k' \arrow[u, "\text{instanz\_von}"'] \arrow[ur, "\text{hat\_eigenschaft}"'] & \end{tikzcd}$$

3) $\text{klasse}(k) \wedge \text{bkante}(b, k, l) \wedge \text{klasse}(l) \rightarrow \text{beziehung}(b, k, l)$

$$k \xrightarrow{\text{b}} l \quad \Rightarrow \quad k \xrightarrow{\text{beziehung}} l$$

4) klasse(k) $\wedge$ bkante(b, k, l) $\wedge$ klasse(l)$\wedge$ is_a(k', k)
$\rightarrow \exists l'$(is_a(l', l)$\vee$ instanz_von(l', l) $\vee$ $l' = l$)$\wedge$ beziehung(b, k', l')

$$
\begin{tikzcd}
k \arrow[r, "\text{b}"] & l \\
k' \arrow[u, "\text{is\_a}"'] &
\end{tikzcd}
\quad \Rightarrow \quad
\begin{tikzcd}
k \arrow[r, "\text{b}"] & l \\
k' \arrow[u, "\text{is\_a}"'] \arrow[r, "\text{beziehung}"] & l' \arrow[u, "\text{is\_a } \textit{oder} \text{ instanz\_von}"']
\end{tikzcd}
$$

5) klasse(k) $\wedge$ bkante(b, k, l) $\wedge$ klasse(l)$\wedge$ instanz_von(k', k)
$\rightarrow \exists l'$(instanz_von(l', l)$\wedge$ beziehung(b, k', l'))

$$
\begin{tikzcd}
k \arrow[r, "\text{b}"] & l \\
k' \arrow[u, "\text{instanz\_von}"'] &
\end{tikzcd}
\quad Rightarrow \quad
\begin{tikzcd}
k \arrow[r, "\text{b}"] & l \\
k' \arrow[u, "\text{instanz\_von}"'] \arrow[r, "\text{beziehung}"] & l' \arrow[u, "\text{is\_a } \textit{oder} \text{ instanz\_von}"']
\end{tikzcd}
$$

6) klasse(k) $\wedge$ bkante(b, k, i) $\wedge$ individuum(i)$\wedge$ (is_a(k', k)$\vee$ instanz_von(k', k)$\vee k = k'$) $\rightarrow$ beziehung(b, k', i)

$$
\begin{tikzcd}
k \arrow[r, "\text{b}"] & i \\
k' \arrow[u, "\text{is\_a } \textit{oder} \text{ instanz\_von}"'] &
\end{tikzcd}
\quad \Rightarrow \quad
\begin{tikzcd}
k \arrow[r, "\text{e}"] & i \\
k' \arrow[u] \arrow[ur, "\text{beziehung}"'] &
\end{tikzcd}
$$

Die Prädikate *klasse* und *individuum* stellen fest, ob es sich bei einem Knoten um ein Klassenkonzept oder ein Individualkonzept handelt. Analog stellen *ekante* und *bkante* fest, ob die Kanten Eigenschafts- oder Beziehungskanten sind. Die Prädikate *hat_eigenschaft* und *beziehung* stellen das explizit oder implizit dargestellte Wissen zur Beantwortung von Anfragen zur Verfügung.
In (4) und (5) sind jeweils Knoten l' explizit zu bestimmen, notfalls durch Rückfrage beim Benutzer, und eine Beziehungskante vom Typ b zu setzen.

Definition 4.8 Zwei Klassenkonzepte k und k' stehen in der *may_be_a-Relation*, in Zeichen *may_be_a*(k, k'), wenn jede Instanz von k auch als Instanz von k' auftreten *kann*, formal

$\forall k, k', i$ (instanz_von(i, k)$\wedge$ may_be_a(k, k') $\Rightarrow \diamond$ instanz_von(i, k')).

Beispiel 4.9 Für die Klassen Student und wiss. Hilfskraft bestehen die Relationen:

$$
\begin{tikzcd}
\text{STUDENT} \arrow[d, shift left=3, "\text{may\_be\_a}"] \\
\text{wiss. HILFSKRAFT} \arrow[u, shift left=3, "\text{is\_a}"]
\end{tikzcd}
$$

Dagegen besteht für die Klassen Säugetier und Katze nur die Relation:

$$
\begin{tikzcd}
\text{Katze} \arrow[r, "\text{is\_a}"] & \text{Säugetier}
\end{tikzcd}
$$

Besteht die Beziehung

$$
\begin{tikzcd}
k \arrow[r, shift left, "\text{is\_a}"] & k' \arrow[l, shift left, "\text{may\_be\_a}"]
\end{tikzcd}
$$

so nennt man k' auch die Repräsentation einer *Rolle*.

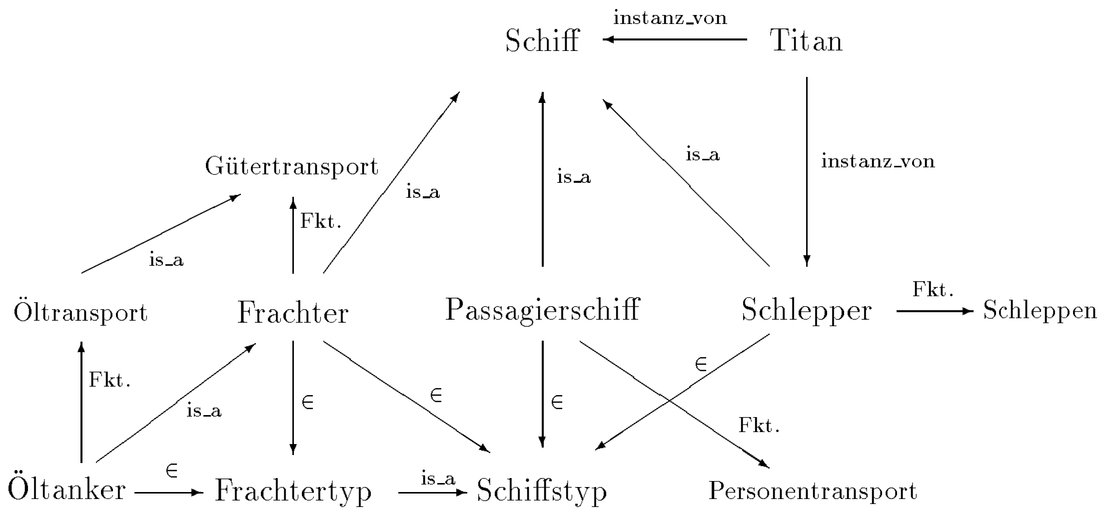
Definition 4.10 Eine *Gruppenklasse* g ist ein Klassenkonzept, in dem verschiedene Klassekonzepte $k_1, \ldots, k_n$, die alle in einer is_a - Relation mit einer gemeinsamen (Ober-)Klasse k stehen, nach einem *Gruppierungskriterium* zusammengefaßt werden. Ist das Gruppierungskriterium eine Eigenschaft e, dann müssen $k_1, \ldots, k_n$ jeweils eine Eigenschaft vom Typ e besitzen, ist es eine Beziehung b, dann müssen $k_1, \ldots, k_n$ jeweils eine Beziehung vom Typ b zu beliebigen anderen Konzepten besitzen. Schreibweise:

$$
\begin{tikzcd}
 & & k & & \\
{[k_1']}\ w_1 & k_1 \arrow[l, "\text{e}"', "{[\text{b}]}"] \arrow[ur] \arrow[dr, "\in"'] & \cdots & k_n \arrow[ul, "\text{is\_a}"'] \arrow[dl, "\in\ (\text{gruppenelement})"] \arrow[r, "\text{e}", "{[\text{b}]}"'] & w_n\ {[k_n']} \\
 & & g & &
\end{tikzcd}
$$

Die Vererbung von Gruppierungsbeziehungen erfolgt nach der Regel

(7) gruppenelement$(k, g) \wedge$ is_a$(k', k) \Rightarrow$ gruppenelement(k', g)

Beispiel 4.11



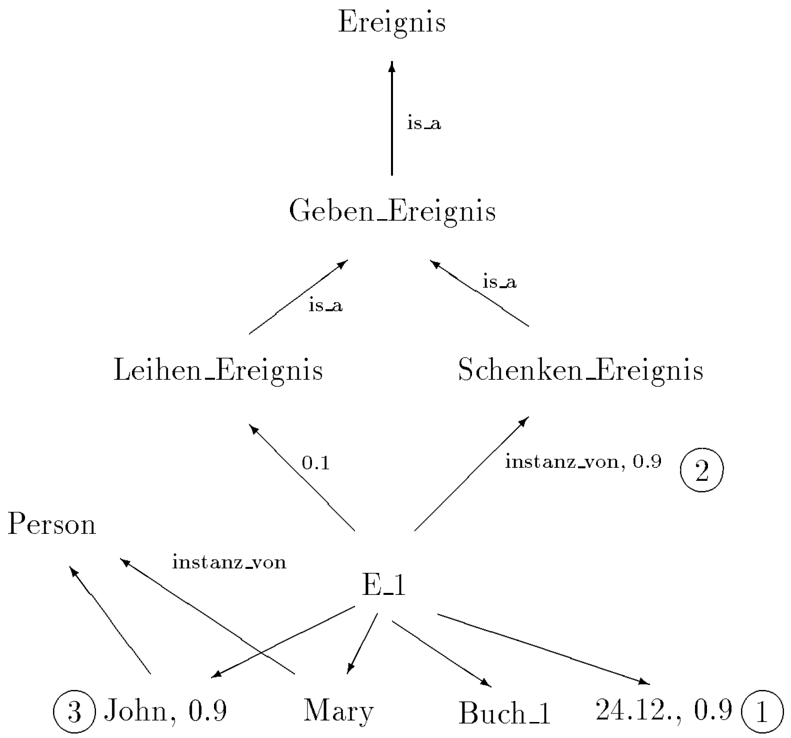
vgl. Unterschied: Dampfschiffe, Segelschiffe, Galeeren, nach Antriebsart.

4.3 Darstellung verschiedener Wissensarten

Eine sehr ausführliche Diskussion dieses Themas findet sich in 3.2. von [Rei91].

Semantische Netze sind (wegen ihrer Herkunft) hervorragend zur Darstellung *begrifflichen Wissens* geeignet, einschließlich Ereignissen und Handlungen.
Unsicheres Wissen läßt sich ebenfalls leicht darstellen, indem Sicherheitsfaktoren an Knoten und/oder Kanten gesetzt werden:

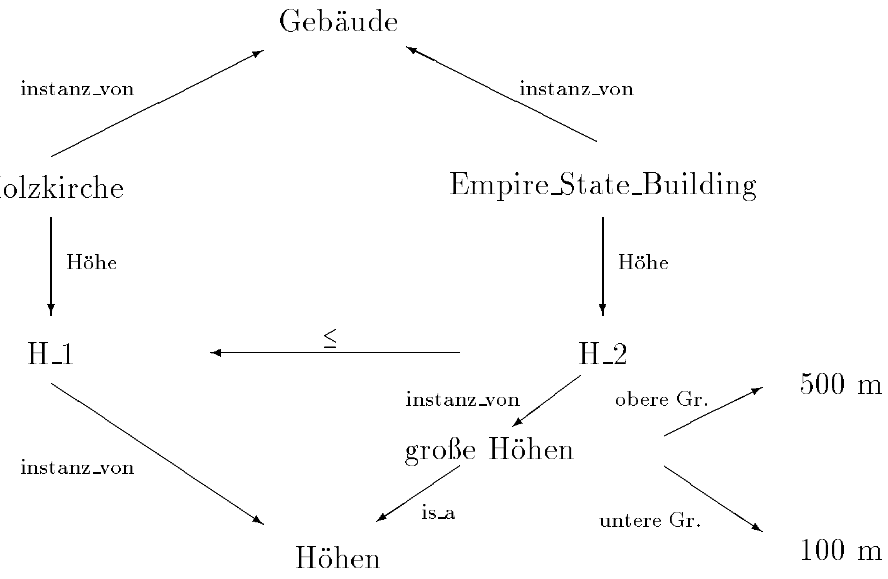
Beispiel 4.12
Unsicheres Wissen läßt sich z. B. wie folgt darstellen:



(1) John schenkte Mary das Buch wahrscheinlich *am* 24.12.
(2) Wahrscheinlich *schenkte* John Mary das Buch am 24.12.
(3) Wahrscheinlich schenkte *John* Mary das Buch am 24.12.

Auch *unvollständiges Wissen* und *ungenaues Wissen* läßt sich darstellen.

Beispiel 4.13

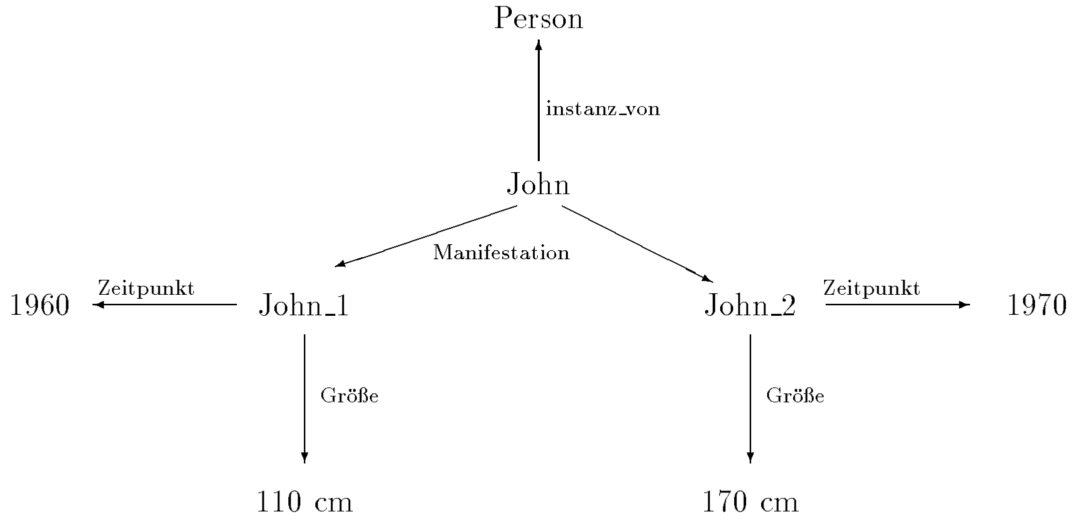


Auf diese Weise lassen sich auch *einschränkende Bedingungen* (für Attributwerte) recht gut darstellen.

Regelhafte Zusammenhänge, die über Vererbungen hinausgehen, lassen sich nur mühevoll darstellen(vgl. 3.2.9 in [Rei91])
Zur Darstellung zeitabhängigen Wissens führt man noch folgenden Begriff ein.

Definition 4.14 Repräsenationen eines Individualkonzeptes, die sich auf unterschiedliche Zeitpunkte beziehen, heißen *Manifestationen* oder *Inkarnationen*.

Beispiel 4.15 Ein Beispiel für eine Manifestitation lautet:



4.4 Operationen auf semantischen Netzen

Natürlich sind für die Arbeit mit semantischen Netzen elementare Operationen, wie Definitionen neuer Knoten und Kanten bzw. Kantentypen nötig. Hier sollen jedoch komplexere Operationen betrachtet werden, die für das Ziehen von Schlüssen zur Beantwortung von Anfragen benutzt werden können. Bei Schlußfolgerungen mittels der Vererbungsmechanismen unterscheidet man zwischen zwei Strategien:

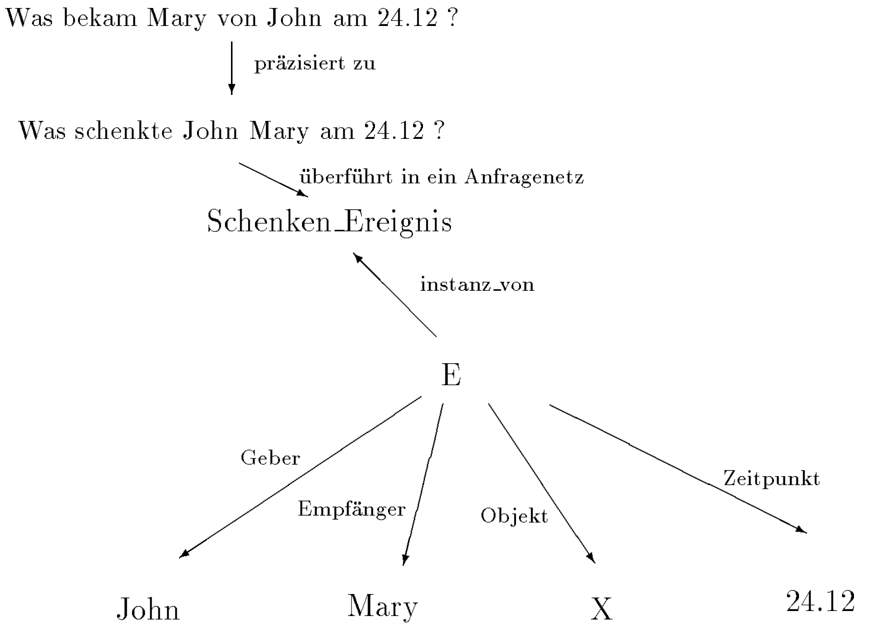
i) *Vererbung zur Anfragezeit* (vgl. Def. 4.7)
ii) *Vererbung zur Änderungszeit*

Zur *Bestimmung aller Eigenschaften* eines Konzeptes k ist bei i) für alle Oberbegriffe $k', k'', \ldots$ von k, sowie für k selbst jede Eigenschaftskante am jeweils *aktivierten* Konzeptknoten zu betrachten und aus ihrem Typ, sowie den anhängenden Knoten die Ergebnismenge zu konstruieren.
Bei ii) sind dagegen nur entsprechend die Eigenschaftskanten von k selbst zu betrachten.

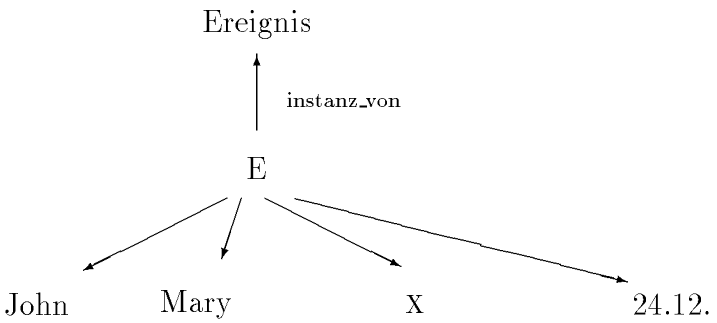
Bei der *Ergänzung einer Eigenschaft* w vom Typ e zum Knoten k ist im Fall i) nur eine Kante vom Typ e zum Knoten w zu legen, während im Falle ii) dieses auch für alle Unterbegriffe von k zu geschehen hat.
Strategie i) spart Speicherplatz, ist aber für Anfragen zeitkritisch. Bei ii) verhält es sich umgekehrt: mehr Speicherplatzbedarf, Änderungen zeitaufwendig, Anfragen schnell auswertbar.

Zur Beantwortung von Benutzerfragen an ein semantisches Netz muß die Anfrage zunächst auf ein *Anfragenetz* abgebildet werden.

Beispiel 4.16 *Anfragenetz*
An das Netz aus Beispiel 4.12 wird die Anfrage gestellt:



Anfragenetze werden wie folgt ausgewertet:
Ist das Anfragenetz ein Teilnetz des gegebenen Netzes, wird die Frage mit "Ja" beantwortet. Treten im Anfragenetz variable Knoten auf, so wird versucht, diese Knotenvariablen derart zu instanziieren, daß das Anfragenetz zum Teilnetz wird (*Abgleichen* oder *matching*). Bei diesem Abgleich dürfen auch alle für die Repräsentation definierten Inferenzregeln (z.B. Vererbungsmechanismen) benutzt werden, etwa bei der Anfrage:



Dadurch weisen Algorithmen zum Netzabgleich eine hohe Komplexität auf.

Kommen im Anfragenetz auch variable Kanten vor, also etwa $a \xrightarrow{?} b$, $a \xrightarrow{?} ?$, $? \xrightarrow{?} b$, so wird das Verfahren der *Aktivierungsausbreitung* benutzt.

Zur Beantworung von $a \xrightarrow{?} b$ werden im ersten Schritt alle mit a und b direkt verbundenen Knoten aktiviert, im nächsten Schritt dann alle Knoten, die mit einem aktivierten Knoten direkt verbunden sind, usw. Die Anfrage wird mit "Ja" beantwortet, wenn ein Knoten von zwei verschiedenen Nachbarknoten aus aktiviert wird, von denen einer von a aus und der andere von b aus aktiviert wurde. Dann ist auch eine entsprechende Knotenfolge gefunden, die eine Beziehung zwischen a und b darstellt. Ist neben der Kante auch ein Knoten variabel, dann breitet sich die Aktivierung nur von einem Zentrum aus.

Beispiel 4.17

Was verbindet John und Mary ?

$\Big\downarrow$ überführt in ein Anfragenetz

$\text{John} \xleftrightarrow{?} \text{Mary}$

Zunächst werden die Knoten John und Mary aktiviert, aber es besteht keine direkte Kante zwischen ihnen. Daher werden im nächsten Schritt die Knoten Person und E_1 aktiviert und damit zwei Beziehungen gefunden:

$\text{John} \xrightarrow{\text{instanz\_von}} \text{Person} \xleftarrow{\text{instanz\_von}} \text{Mary}$

"John und Mary sind Personen !"

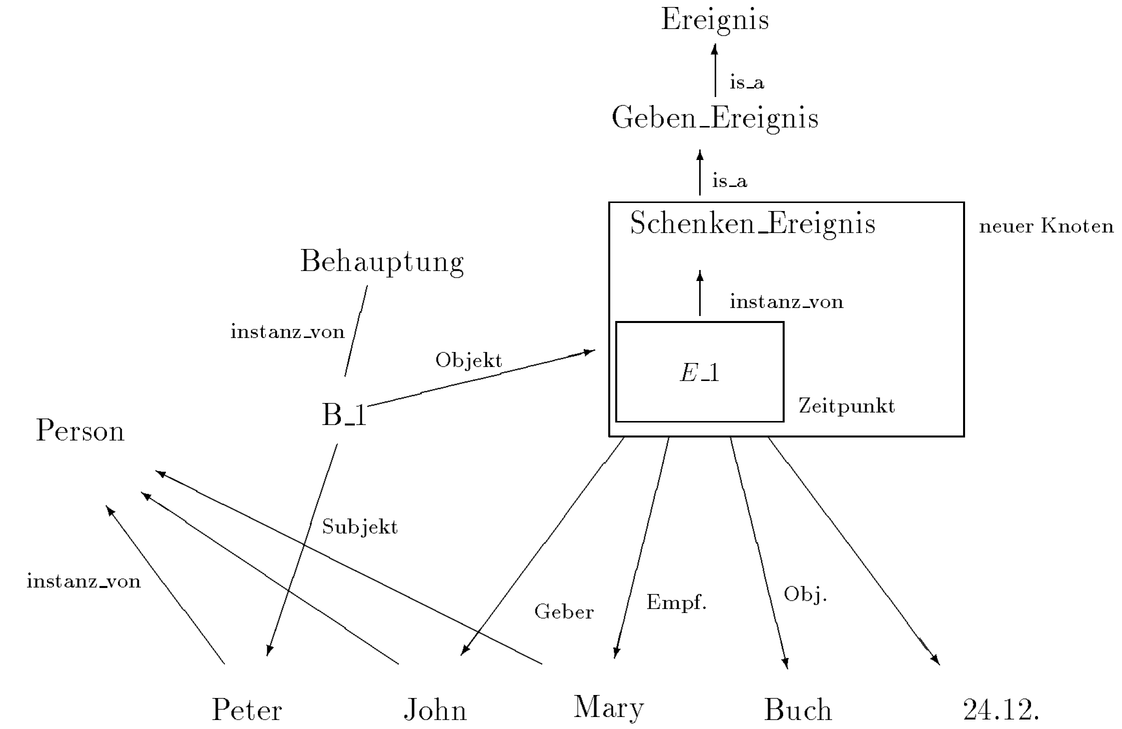
$\text{John} \xleftarrow{\text{Geber}} \text{E\_1} \xrightarrow{\text{Nehmer}} \text{Mary}$

"Es gibt ein Ereignis, an dem John als Geber und Mary als Empfänger beteiligt war !" $\rightarrow$ "John gab Mary etwas !"

Die Ausdruckskraft semantischer Netze kann durch *Partitionierung* gesteigert werden. Dabei werden Teilnetze zu Knoten zusammengefaßt (vgl. 3.4 in [Rei91]).

Beispiel 4.18

"Peter behauptet, daß John Mary ein Buch gab !"



Dabei sind Knoten und Kanten, deren Beschriftungen innerhalb des neuen Knotens stehen, von außen nicht sichtbar, also etwa, daß E_1 am 24.12. stattfand oder oder ein *Schenken_Ereignis* ist.

Kapitel 5

Frame-artige Repräsentationsstrukturen

5.1 Einführung

Diese Formate gehen auf den Schema-Begriff in der Kognitionspsychologie zurück (vgl. semantische Netze) und wurden erstmals 1975 von Marvin Minsky zur Wissensdarstellung vorgeschlagen. Ein *Schema* ist ein Modell für eine Gedächtnisstruktur, die ein *stereotypisches Erinnerungsmuster* darstellt.
Die Aktivierung eines derartigen Musters bedingt eine bestimmte *Erwartungshaltung*, die eine kontextabhängige Interpretation von Objekten und Ereignissen darstellt.

Beispiel 5.1 Die Aktivierung des Schemas für das Konzept 'Universität' bewirkt die Vorstellung einer typischen Universität mit Instituten, Mensa, Professoren, Studenten, etc. Charakteristisch für solch ein Muster ist seine Unvollständigkeit und die höhere Gewichtung besonders typischer Merkmale. Man vergleiche etwa mit dem Konzept 'Fern-Universität'.

Experimentelle Untersuchungen legen nahe, daß Klassenkonzepte im menschlichen Gedächtnis generell durch ihren typischsten Repräsentanten, den *Prototypen*, dargestellt werden. Dieser ist dadurch bestimmt, daß er möglichst viele Eigenschaften gemeinsam mit anderen Klassenmitgliedern hat und möglichst wenige Eigenschaften gemeinsam mit Mitgliedern anderer Klassen. Dieser Prototyp muß real gar nicht existieren!
Die formale Repräsentation solcher Schemata geschieht nun durch sogenannte Frame-Strukturen (frame = Rahmen). Ein Frame besteht dabei aus mehreren Slots (slot = Schlitz), die jeweils ein Merkmal des zu repräsentierenden Konzeptes darstellen. Jeder Slot besitzt Einträge, die den entsprechenden Merkmalswert beschreiben. Ist das Merkmal eine Beziehung zu einem anderen Konzept, so kann der Eintrag selbst wieder ein Frame sein.

5.2 Frame-Strukturen

Definition 5.2 Ein *Frame* ist ein Tripel (N, SN, ST) bestehend aus
- dem *Frame-Namen* N
- einer Menge $SN = \{sn_1, \ldots, sn_k\}, k \geq 0$, *nicht-terminaler Slots* und
- einer Menge $ST = \{st_1, \ldots, st_m\}, m \geq 0$, *terminaler Slots.*

Jeder nicht-terminale Slot sn_i ist wiederum ein Tripel $(N_{sn_i}, SB_{sn_i}, SE_{sn_i})$, wobei N_{sn_i} der *Slot-Name*, SB_{sn_i} die *Slot-Beschreibung* und SE_{sn_i} eine Menge von Frame-Namen, den *Slot-Einträgen* ist.
Jeder terminale Slot st_i ist ebenfalls ein Tripel $(N_{st_i}, SB_{st_i}, SE_{st_i})$ aus *Slot-Name* N_{st_i}, *Slot-Beschreibung* SB_{st_i} und einer Menge von *Slot-Einträgen* SE_{st_i}, die jetzt aber elementare Daten sein müssen.

Beispiel 5.3 Ein (noch etwas unvollständiger) Frame zum Beispiel 5.1 ist etwa:

('Universität',{ ('Institute',∅ [1],{'Mathematik','Informatik',...}),
('Zentrale Einrichtungen',∅ [1], {'Mensa','Rechenzentrum',...})},
{('Studentenzahl','genau ein Wert aus $I\!N$',{10000} [2])})

Die Slot-Beschreibung von 'Studentenzahl' läßt sich gemäß Definition 5.5 formaler schreiben als $(I\!N, (1,1), \{10000\})$.

[1] Es fehlt noch die exakte Beschreibung
[2] Default - Wert.

Bemerkung 5.4

i) Diese Definition ist nur als Grundmuster zu verstehen, das in konkreten Frame-Sprachen stark verändert sein kann.

ii) Slot-Beschreibungen geben u.a. die Mengen zulässiger Slot-Einträge an, legen fest, wieviele Einträge ein Slot mindestens oder höchstens haben kann, oder spezifizieren, welche Einträge Default-Werte darstellen (vgl. Def. 5.5).

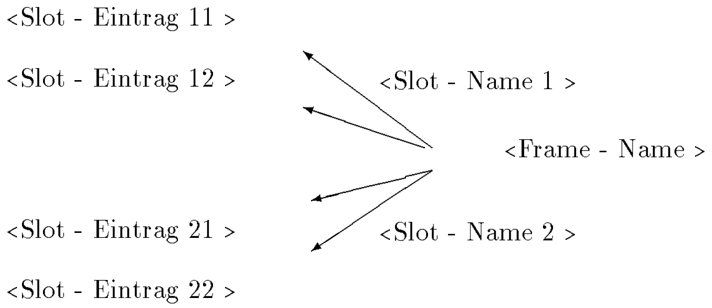
iii) Elementare Daten sind etwa Zahlen oder Strings.

iv) Das folgende Schema zeigt, daß die Ausdrucksstärke von Frames genauso groß ist wie die semantischer Netze und damit von PL1.

Frame:

<Frame - Name >	<Slot - Name1 >	<Slot - Name2 >	...
	<Slot - Eintrag11>	<Slot - Eintrag21>	
	<Slot - Eintrag12>	...	
	...		

Netz:



Definition 5.5 Für jeden Slot (N, SB, SE) ist die Slot-Beschreibung von der Form $SB = (EE, KARD, DEF)$. Dabei ist EE eine (extensional oder intensional definierte, eventuell unendliche) Menge erlaubter Einträge und es muß $SE \subseteq EE$ gelten. Die Einschränkung der Anzahl erlaubter Einträge ist von der Form $KARD = (MIN, MAX)$ und es muß $\mid SE \mid = 0 \vee MIN \leq \mid SE \mid \leq MAX$ gelten. Für die Menge DEF der *Default-Einträge* muß ebenfalls $DEF \subseteq EE$ gelten.

Bemerkung 5.6

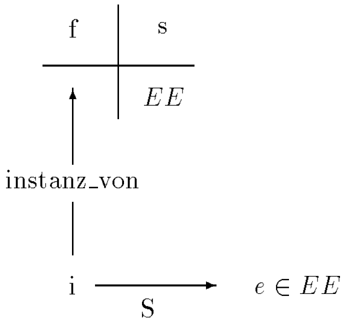
i) Im Fall terminaler Slots ist EE also eine Menge elementarer Daten in geeigneter Darstellung, also etwa durch Aufzählung oder mittels geeigneter Testprädikate. Im Falle nicht-terminaler Slots erfolgt die Angabe erlaubter Frame-Namen i.a. als Menge $\{x \mid F(x)\}$ mit einer geeigneten prädikatenlogischen Formel F und einer freien Variablen x.

ii) Manchmal wird für *jeden terminalen* Slot $MIN = MAX = 1$ verlangt. Man spricht auch von *einwertigen* und *mehrwertigen Slots*. Der Fall $\mid SE \mid = 0$, also $SE = \emptyset$ ist so zu interpretieren, daß der Slot - Eintrag noch unbekannt ist, aber prinzipiell existiert (vgl. (3) in Def. 5.7).

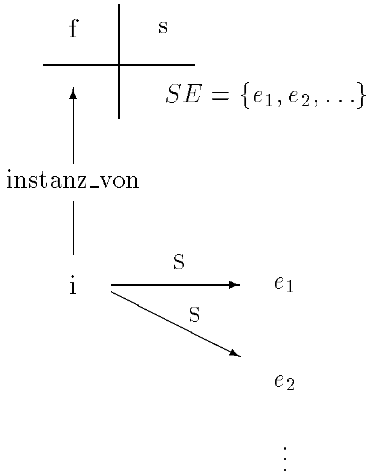
Man kann die Semantik von Frame-Strukturen genau wie die der semantischen Netze wieder durch Angabe von prädikatenlogischen Formeln formalisieren, etwa für terminale Slots:

Definition 5.7 Für alle Frame-Namen f und alle Slot-Namen s gelten:

1) $klasse(f) \wedge hat\_terminalslot(f,s) \rightarrow (\forall i : instanz\_von(i,f) \rightarrow$
$\exists e \in erlaubte\_eintr\ddot{a}ge(f,s) : hat\_eigenschaft(i,s,e))$



2) $klasse(f) \wedge hat\_terminalslot(f,s) \rightarrow (\forall i : instanz\_von(i,f) \rightarrow$
$\forall e \in eintr\ddot{a}ge(f,s) : hat\_eigenschaft(i,s,e))$



3) $individuum(f) \wedge hat\_terminalslot(f,s) \rightarrow (\exists e : eintr\ddot{a}ge(f,s) = \{e\} \vee$
$(eintr\ddot{a}ge(f,s) = \emptyset \wedge e \in erlaubte\_eintr\ddot{a}ge(f,s) \wedge hat\_eigenschaft(f,s,e)))$

Dabei stellt das Prädikat $hat\_terminalslot(f,s)$ fest, ob s ein terminaler Slot von f ist. Die Funktion $erlaubte\_eintr\ddot{a}ge(f,s)$ bzw. $eintr\ddot{a}ge(f,s)$ liefern die entsprechenden Mengen EE bzw. SE von s. Alle anderen Prädikate haben dieselbe Bedeutung wie bei semantischen Netzen.

Man beachte, daß bei der Zuordnung von semantischen Netzen und Frames gemäß Bem. 5.4 iv) die terminalen Slots gerade für Eigenschaftskanten und ihre Einträge für Eigenschaftsknoten stehen.
Bei den nicht-terminalen Slots kann man verschiedene Typen unterscheiden. Zunächst die System-Slots:

Definition 5.8 Ein Slot $s = (N, SB, SE)$ mit $N \in \{is\_a, instanz\_von, may\_be\_a\}$ heißt *System-Slot*. Für alle System-Slots s und alle Frames f gilt:

$$(4) \quad \left\{ \begin{array}{c} klasse(f) \\ individuum(f) \\ klasse(f) \end{array} \right\} \wedge hat\_slot(f,s) \wedge N = \left\{ \begin{array}{c} is\_a \\ instanz\_von \\ may\_be\_a \end{array} \right\} \rightarrow$$

$$(erlaubte\_eintr\ddot{a}ge(f,s) = \{x \mid klasse(x)\} \wedge \forall e \in eintr\ddot{a}ge(f,s) : \left\{ \begin{array}{c} is\_a(f,e) \\ instanz\_von(f,e) \\ may\_be\_a(f,e) \end{array} \right\}$$

Beispiel 5.9 In der Notation von Bem. 5.4 iv).

Säugetier	Lebensraum	...
is_a	$EE = \{Wasser, Land\}$	

Elefant	is_a	may_be_a	Lebensraum	Herkunft
	{Säugetier}	{Zootier,Zirkustier}	{Land}	$EE = \{Indien, Afrika\}$

Zootier	...

Zirkustier	...

Bonnie	instanz_von	Lebensraum	Herkunft	
	{Elefant, Zirkustier}	{Land}	{Afrika}	

Clyde	instanz_von	Lebensraum	Herkunft	
	{Elefant, Zootier}	{Land}	{Indien}	

Der terminale Slot 'Lebensraum' des Frames 'Elefant' wird vom Oberbegriff 'Säugetier' geerbt. Der terminale Slot 'Herkunft' spezifiziert dagegen den Unterbegriff näher.

Bemerkung 5.10 Insgesamt kann man für Frame-Strukturen folgende Spezialisierungsmöglichkeiten unterscheiden:
1. Hinzufügen eines Slots.
2. Einschränkung der Menge EE.
3. Einschränkung von $KARD$.
4. Vergrößern von SE.
5. Spezialisierung eines Eintrages in einen nicht-terminalen Slot durch 1. bis 5.

Weitere Typen nicht-terminaler Slots unterscheidet die folgende Definition.

Definition 5.11 Es sei $s = (N, SB, SE)$ ein nicht-terminaler Slot mit $SB = (EE, KARD, DEF)$

i) s heißt *Relationen-Slot*, wenn gilt $\neg klasse(N) \wedge \neg individuum(N)$.

ii) s heißt *assoziativer Slot*, wenn gilt
$klasse(N) \wedge \forall e \in EE : is\_a(e, N) \vee instanz\_von(e, N)$.

iii) s heißt *Rollen-Slot*, wenn gilt
$klasse(N) \wedge \forall e \in EE\ \exists\ k\ may\_be\_a(k, N) \wedge (is\_a(e, k) \vee instanz\_von(e, k))$.

Beispiel 5.12

Student	is_a	may_be_a
	$EE = \{x \mid is\_a(x, Person) \vee instanz\_von(x, Person)\}$	$EE = \{Tutor, Praktikant, \ldots\}$

is_a

Tutor	is_a	Fach
	$EE = \{Student\}$	$EE = \{AnalysisI, AlgebraI, \ldots\}$

Üb.-Gruppe	Tutor	Üb.-Raum	Zeit	Teilnehmer
	$EE = \{Tutor\}$	$EE = \{Üb. - Raum\}$	$EE = \{Zeit-raum$	$EE = \{Student\}$

Üb.-Raum	Kapazität

Die Slots des Frames *Übungsgruppe* sind dann von folgendem Typ:

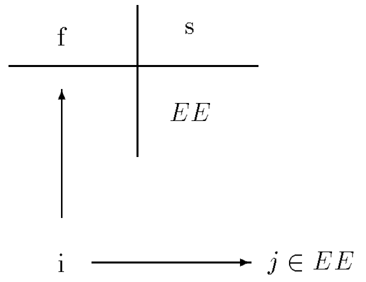
Zeit ist terminaler Slot.
Teilnehmer ist ein Relationen-Slot.
Tutor ist ein Rollen-Slot.
Übungsraum ist ein assoziativer Slot.

Ähnlich wie für terminale Slots läßt sich auch hier die Semantik formalisieren, z.B. für Relationen-Slots (für assoziative und Rollen-Slots vgl. [Rei91], 4.2.2 und 4.2.3):

Definition 5.13 Für alle Frames f und alle Slots s gelten:

(5) $klasse(f) \wedge hat\_relationenslot(f, s) \rightarrow (\forall\, i : instanz\_von(i, f) \rightarrow$

$\exists\, j : individuum(j) \wedge j \in erlaubte\_eintr\ddot{a}ge(f, s) \wedge beziehung(s, i, j))$



(6) $klasse(f) \wedge hat\_relationenslot(f, s) \rightarrow (\forall\, e \ \in eintr\ddot{a}ge(f, s) \forall\, i : instanz\_von(i, f) \rightarrow$

$\exists\, j : (klasse(e) \wedge instanz\_von(j, e)) \vee (individuum(e) \wedge j = e)) \wedge beziehung(s, i, j)))$

(7) $individuum(f) \wedge hat\_relationenslot(f, s) \rightarrow$

$\exists\, j : individuum(j) \wedge j \in erlaubte\_eintr\ddot{a}ge(f, s) \wedge beziehung(s, f, j)$

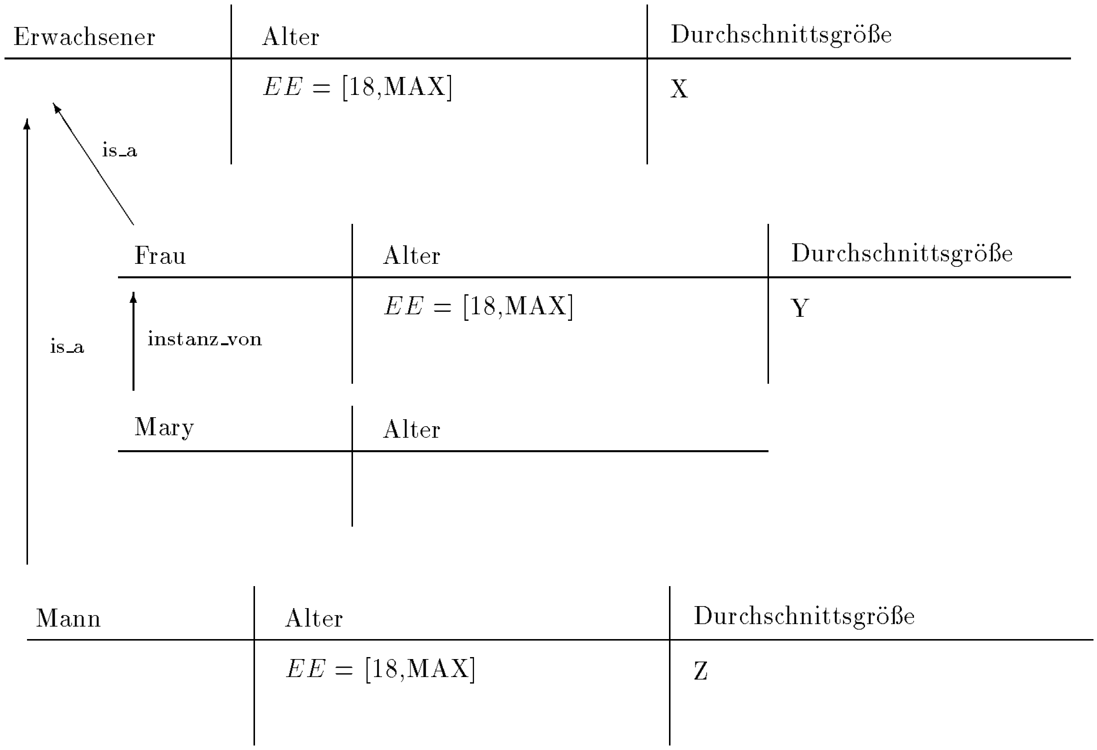
(8) $individuum(f) \wedge hat\_relationenslot(f, s) \rightarrow$

$(\forall\, e \ \in eintr\ddot{a}ge(f, s) \exists\, j : ((klasse(e) \wedge instanz\_von(j, e)) \vee$

$(individuum(e) \wedge j = e)) \wedge beziehung(s, f, j))$

Bemerkung 5.14

i) Durch die Axiome (1) bis (8) und die analogen Axiome für assoziative und Rollen-Slots wird der Vererbungsmechanismus für Frames festgelegt. Zusätzlich werden natürlich die System-Slots in derselben Weise vererbt, wie die entsprechenden Relationen in semantischen Netzen.
Manchmal werden für Klassenkonzepte noch spezielle Klassenslots definiert, die Eigenschaften der Klasse selbst und nicht ihrer Elemente beschreiben. Diese Klassenslots werden natürlich nicht auf die Instanzen der Klasse vererbt. Dagegen werden sie sehr wohl auf Unterklassen vererbt, wobei sich die Einträge aber ändern können.



i.a. $Y < X < Z$

ii) In vielen Frame-Sprachen werden die Vererbungsmechanismen dadurch realisiert, daß den Slots Prozeduren (*Methoden*) zugeordnet werden. Ordnet man jedem Slot, der einen Eintrag erben soll, eine Prozedur zu, die immer dann, wenn dieser Eintrag abgefragt wird, bei allen Oberbegriffen nachschaut und dort eventuelle Einträge ermittelt, realisiert man Vererbung zur Anfragezeit. Ordnet man dagegen jedem Slot, der einen Eintrag vererben soll eine Prozedur zu, die immer dann, wenn dieser Eintrag verändert wird (*aktive Werte*), einen entsprechenden Eintrag bei allen Unterbegriffen vornimmt, dann realisiert man Vererbung zur Änderungszeit. Läßt man beliebige benutzerdefinierte Prozeduren sogar als Einträge für Slots zu, um etwa Inferenzmechanismen zu realisieren, so gibt man damit jede Kontrolle über die Integrität des dargestellten Wissens auf.

5.3 Darstellung verschiedener Wissensarten durch Frames

Wegen der engen Verwandtschaft von Frame-Strukturen und semantischen Netzen (vgl. Bem. 5.4 (iv)) gelten hier im wesentlichen dieselben Aussagen wie für semantische Netze. In 5.1 wurde aber erwähnt, daß Frames speziell zur Darstellung von Default-Aussagen und prototypischem Wissen entwickelt wurden.

Definition 5.15 Es sei $s = (N, SB, SE)$ mit $SB = (EE, KARD, DEF)$ Slot eines Frames f, also hat_slot(f,s) wahr. Jeder Wert aus DEF heißt *zulässiger Default-Wert*, jeder Wert aus $DEF \cap SE$ *Default-Wert* oder *default-basierter-Eintrag*. Jeder Wert aus $SE \setminus DEF$ heißt *gesicherter Wert*.

Beispiel 5.16

Elefant	Farbe
	$EE = \{grau, weiss\}$ $DEF = \{grau\}$

is_a

grauer Elefant	Farbe
	$EE = \{grau, weiss\}$ $DEF = \{\ \}$ $SE = \{grau\}$

Albino-Elefant	Farbe
	$EE = \{grau, weiss\}$ $DEF = \{\ \}$ $SE = \{weiss\}$

(Hier ist "grau" ein definitorischer Wert, gehört also notwendig zu "grauer Elefant".)
Ist nun Bonnie ein Elefant, von dem nichts weiter bekannt ist, und Clyde ein Albino-Elefant, dann ergeben sich folgende Frames:

Bonnie	Farbe
	$DEF = \{grau\}$ $SE = \{grau\}$

Clyde	Farbe
	$DEF = \{\ \}$ $SE = \{weiss\}$

default - orientierter Eintrag

gesicherter Wert

Die Semantik von Default - Aussagen läßt sich mit Hilfe des Modaloperators M ("es ist konsistent anzunehmen, daß") formalisieren (vgl. [Ric89]).

Bemerkung 5.17

i) Durch die Unterscheidung default-basierter Einträge und gesicherter Einträge schaltet man eine Möglichkeit des Entstehens von Widersprüchen aus. Wären nämlich alle von einem Oberbegriff geerbten Einträge als Default-Werte beliebig überschreibbar, so könnte man Unterbegriffe mit Werten definieren, die denjenigen des Oberbegriffes widersprächen (vgl. Albino in Bsp 5.16).

ii) Immer wenn im Verlauf von Inferenzprozessen als wahr angenommene Einträge (Default-Einträge) durch zusätzliches Wissen überschrieben werden müssen, bedeutet dies, daß man ein außergewöhnliches Exemplar oder eine Klasse solcher Exemplare gefunden hat.

Unsicheres Wissen läßt sich wieder wie üblich durch Sicherheitsfaktoren für Slots bzw. deren Einträge repräsentieren. Ebenso leicht lassen sich unvollständige Konzeptbeschreibungen darstellen, in denen Eigenschaften noch unbekannt oder erst teilweise bestimmt sind.
Regelhafte Zusammenhänge und einschränkende Bedingungen werden in Frame-Sprachen durch den Slots angeheftete Prozeduren oder Formeln repräsentiert, die eine entsprechende Verbindung zwischen Slots herstellen bzw. unerlaubte Einträge zurückweisen (etwa Zusammenhang zwischen Bafög - Satz eines Studenten und Einkommen seiner Eltern). Streng genommen handelt es sich dann jedoch nicht mehr um ein reines Frame-Format, sondern um ein hybrides Format.
Zur leichteren Darstellung von Handlungen und Ereignissen werden Frames oft zu sogenannten Scripts erweitert.

Definition 5.18 *Script*
Ein *Script* ist ein 6-Tupel (N, EB, AB, TE, SN, ST), wobei N, SN, ST dieselbe Bedeutung wie bei Frames haben (vgl. Def. 5.2).
EB und AB sind logische Formeln, die *Eingangsbedingung* EB bzw. *Ergebniszustand* AB *des Scripts* angeben.
TE ist ein n-Tupel $(n \geq 0)$ von *Ereignisslots*. Jeder Ereignis-Slot ist ein nicht-terminaler Slot (N, SB, SE), wobei SE ein m-Tupel $(m \geq 0)$ von Scriptnamen ist.

Beispiel 5.19

Buchausleihe	EB	AB	TE	Zeitpunkt	Ausleiher	Buch

(Bibliothek betreten, Katalogeinsicht,
Buchentnahme, Verbuchung, Bibliothek verlassen)

EB beschreibt etwa, daß zum aktuellen Zeitpunkt der Buchausleihe die Bibliothek geöffnet ist und der Ausleiher zur Ausleihe berechtigt sein muß.

AB beschreibt etwa bei der Verbuchung, daß das entliehene Buch zum gegebenen Zeitpunkt ausgeliehen wurde.

Für weitere Einzelheiten zu Scripts vgl. [Rei91] 4.2.10.
Zu Anfragen an Frame-Strukturen und dafür benötigte Operationen vgl. man [Rei91] 4.3.

Kapitel 6

Heuristische Suchverfahren

Weitere Informationen zum Thema *Heuristische Suchverfahren* findet man im Literaturverzeichnis unter [Nil82], [Bun83], [Ban90], [Kai89], [KK88], [Len90], [Pea85] und [Rei89].

6.1 Das Modell des Suchraumes

Definition 6.1 Eine *Heuristik* ist eine problemspezifische Information, die zur Steigerung der Effizienz von allgemeinen Suchverfahren bei der Lösung spezieller Probleme verwendet werden kann.

Definition 6.2

i) Ein *Suchraum* oder *Problemraum* ist ein Paar (S, T), bestehend aus einer Menge $S \neq \emptyset$ von *Situationen* oder *Zuständen* und einer Menge $T \neq \emptyset$ von *partiellen Transformationen* von S, d.h. $t : D_t \to S$ mit $D_t \subseteq S$.

ii) Ein Problem ist ein 4 - Tupel (S, T, s_0, Z), wobei (S, T) ein *Problemraum*, $s_0 \in S$ ein *Startzustand* und $\emptyset \neq Z \subseteq S$ eine *Menge von Zielzuständen* ist.

iii) Eine *Lösung* eines Problems (S, T, s_0, Z) ist eine Folge $t_1 t_2 \ldots t_n \in T^*$ von Transformationen mit $s_0 t_1 \ldots t_{\nu-1} \in D_{t_\nu}$ für $\nu = 1, \ldots, n$ und $s_0 t_1 \ldots t_n \in Z$.

Beispiel 6.3 Das 8-er Puzzle besteht aus 8 quadratischen Marken, die mit den Ziffern von 1 bis 8 beschriftet sind und in einem quadratischen 3×3 - Gitter angeordnet sind. Ein Gitterplatz bleibt also leer. S besteht aus den möglichen Belegungen des Gitters, also etwa:

$$(1) \quad \begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 8 & & 4 \\ \hline 7 & 6 & 5 \\ \hline \end{array}$$

Es ist $T = \{o, u, l, r\}$ die Menge der Verschiebungen des leeren Gitterplatzes nach oben, unten, links, rechts:

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 8 & & 4 \\ \hline 7 & 6 & 5 \\ \hline \end{array} \quad \underset{l}{\overset{r}{\rightleftarrows}} \quad \begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 8 & 4 & \\ \hline 7 & 6 & 5 \\ \hline \end{array}$$

Als Startzustand s_0 wird eine beliebige Belegung des Gitters vorgegeben, Z besteht genau aus dem durch (1) beschriebenen Zustand.
Das 8 - er Puzzle läßt sich zum $(n^2 - 1)$ - er Puzzle auf einem $n \times n$ - Gitter verallgemeinern.
Der Zustandsraum S *explodiert kombinatorisch* mit der Dimension n des Problems gemäß $\mid S \mid = (n^2)!$

n	$\mid S \mid$
2	$4! = 24$
3	$9! = 362880$
4	$16! = 2.1 * 10^{13}$
5	$25! = 1.5 * 10^{25}$
6	$36! = 3.7 * 10^{41}$
7	$49! = 6.1 * 10^{62}$
8	$64! = 1.27 * 10^{89}$

Weitere Beipiele für Suchprobleme liefern Produktionsregelsysteme mit der anfänglichen Wissensbasis als Startzustand, der Endebedingung als Beschreibung der Endzustände und den Produktionsregeln als Transformationen der möglichen Zustände der dynamischen Wissensbasis.
Auch beim automatischen Theorembeweisen treten Suchprobleme auf:
Zustände sind endliche Mengen von Formeln aus PL1, Startzustand ist das gegebene Programm. Zielzustand ist jeder Zustand, der das zu beweisende Theorem enthält. Die Transformationen werden durch die Schlußregeln des Theorembeweisers beschrieben. (Man mache sich die Modifikation für einen PROLOG - Interpreter klar!)

Definition 6.4 Es sei (S,T) einSuchraum. Der zugehörige *Zustandgraph* ist der gerichtete Graph (S,K), wobei die Menge der Kanten $K \subseteq S \times S$ durch
(2) $(s,s') \in K \Leftrightarrow$ ex. $t \in T$ mit $s \in D_t$ und $s' = st$
für alle $s, s' \in S$ definiert sei.

Bemerkung 6.5

i) Man beachte, daß es in (2) durchaus mehrere Transformationen $t_1, \ldots, t_k$ geben kann, die s in s' überführen. Will man eine umkehrbar eindeutige Zuordnung zwischen Suchraum und zugehörigem Zustandsgraph erreichen, dann muß man die Kante (s,s') mit allen diesen Transformationen markieren. Wir werden in Zukunft ein Problem immer einfach durch einen Zustandsgraphen (S,K), einen *Startknoten* $s_0 \in S$ und eine Menge $Z \subseteq S$ von *Zielknoten* repräsentieren.

ii) In der Praxis ist oft die Transformation eines Zustandes s in seinen Folgezustand $s' = st$ mit einem gewissen Aufwand (Kosten,Zeit,etc.) verbunden. Um dies zu berücksichtigen, betrachtet man Zustandsgraphen meist als bewertete Graphen (S,K,w), wobei $w: K \rightarrow I\!R_+$ für jede Kante $(s,s') \in K$ einen Wert $w(s,s')$ angibt. Ist $(s,s') \notin K$, so setzt man $w(s,s') = \infty$ und hat damit sogar eine Wertefunktion $w: S \times S \rightarrow I\!R_+ \cup \{\infty\}$.

iii) Für einen bewerteten Zustandsgraphen (S,K,w) kann durchaus $\mid S \mid = \infty$ sein. Man verlangt jedoch, daß der Graph *lokal endlich* ist, d.h. daß für $s \in S$ stets nur endlich viele $s' \in S$ mit $(s,s') \in K$ existieren. Außerdem wird gefordert, daß ein $\epsilon > 0$ mit $w(s,s') \geq \epsilon$ für alle $(s,s') \in K$ existiert.

6.2 Der GRAPHSEARCH-Algorithmus

Definition 6.6

i) Ein *gerichteter bewerteter Graph* (S,K,w) besteht aus einer nichtleeren Menge von *Knoten*, einer *Kantenrelation* $K \subseteq S \times S$ und einer *Bewertungsfunktion* $w: S \times S \rightarrow I\!R_+ \cup \{\infty\}$ mit
1)$w(s,s') = \infty \Leftrightarrow (s,s') \notin K$
2)$w(s,s') \geq \epsilon$ für ein $\epsilon > 0$ und alle $s, s' \in S$

ii) Ein *Pfad p in* (S,K,w) der *Länge* n ist eine endliche Folge von Knoten $p = (s_0, \ldots, s_n)$ mit $(s_{\nu-1}, s_\nu) \in K$ für $\nu = 1, \ldots, n$. Der *Wert* von $p = (s_0, \ldots, s_n)$ ist $w(p) = w(s_0,s_1) + \ldots + w(s_{n-1}, s_n)$.

Bemerkung 6.7

i) Wir fordern zusätzlich, daß (S,K) lokal endlich ist. Dann gibt es zu jedem $s_0 \in S$ und jedem $n \in I\!N$ nur endlich viele Pfade $p = (s_0, \ldots, s_n)$.

ii) Es ist $(I\!R_+ \cup \{\infty\}, min, +)$ ein Halbring, wenn man $min\{a,b\}$ als Addition und $a+b$ als Multiplikation auffaßt. Dabei ist ∞ dann das Nullelement des Halbringes. Für manche Anwendungen bewerteter Graphen ist es sinnvoll, Bewertungsfunktionen $w: S \times S \rightarrow H$ für beliebige Halbringe $(H, +, \cdot)$ zu betrachten.
Statt 1) hat man dann:
1′) $w(s,s') = 0 \Leftrightarrow (s,s') \notin K$
und auf 2) verzichtet man ganz.
Die Bewertung eines Pfades $p = (s_0, \ldots, s_n)$ ergibt sich dann als $w(p) = w(s_0,s_1) \cdot \ldots \cdot w(s_{n-1}, s_n)$.

iii) Jedem *endlichen* gerichteten bewerteten Graphen (S,K,w) mit $\mid S \mid = n$, also etwa $S = \{1, \ldots, n\}$, kann man umkehrbar eindeutig eine $n \times n$ - Matrix als *Adjazenzmatrix* zuordnen gemäß $A[i,j] = w(i,j)$ für alle $i, j \in S$. Auf diese Weise kann man viele Probleme für derartige Graphen in algebraische Probleme überführen und mit algebraischen Methoden lösen.

Satz 6.8 Sei (S, K, w) lokal endlicher bewerteter Zustandsgraph des Problems (S, T, s_0, Z) und $p = (s_0, \ldots, s_n)$ ein Pfad mit $s_n \in Z$. Dann existiert ein Pfad $p' = (s_0, \ldots, s_m)$ mit $s_m \in Z$, für den $w(p')$ *minimal* unter allen derartigen Pfaden ist.
Beweis:
Sei $c = w(p)$. Für jeden Pfad $\tilde{p} = (s_0, \ldots, s_l)$, $s_l \in Z$ mit $l > \frac{c}{\epsilon}$ gilt $w(\tilde{p}) \geq l * \epsilon > c$, also ist $w(\tilde{p})$ nicht minimal. Es gibt aber nur endlich viele Pfade $p = (s_0, \ldots, s_m)$ einer Länge $m \leq \frac{c}{\epsilon}$. Daher gibt es einen minimalen Pfad der behaupteten Art.

Bemerkung 6.9 Zu jedem lösbaren Problem existiert daher eine *minimale Lösung*!

In der folgenden Definition wird ein Prä - Algorithmus GRAPHSEARCH beschrieben, den wir zu verschiedenen Algorithmen zur Suche nach solch minimalen Lösungen ausbauen werden.

Definition 6.10 Sei (S, K, w) lokal endlicher bewerteter Zustandsgraph des Problems (S, T, s_0, Z). Weiterhin sei OPEN eine Liste der noch zu untersuchenden Knoten und CLOSED eine Liste schon untersuchter Knoten.

G1 Setze OPEN := (s_0) und CLOSED := ( ).

G2 Ist OPEN := ( ), dann existiert keine Lösung des Problems.

G3 Wähle einen Knoten s beliebig aus OPEN aus, wenn möglich $s \in Z$, entferne s aus OPEN und hänge ihn an CLOSED an.

G4 Falls $s \in Z$, ist eine Lösung gefunden.

G5 *Expandiere* s, indem alle *direkten Nachfolger* s' von s in einer festen Reihenfolge erzeugt werden, d.h. alle $s' \in S$ mit $(s, s') \in K$.
Für jeden direkten Nachfolger s' von s führe G6 und G7 durch.

G6 Ist s' weder in OPEN noch in CLOSED, hänge ihn an OPEN an.

G7 Ist s' in OPEN, lasse ihn dort. Ist s' in CLOSED, lösche ihn dort und hänge ihn an OPEN an.

G8 Goto G2.

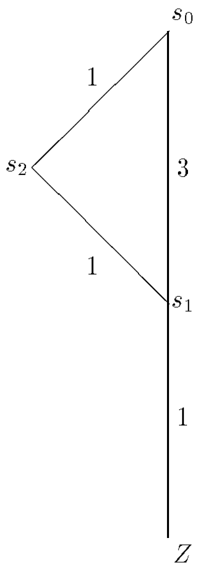
Bemerkung 6.11

i) In G3 kann man die Auswahl von s aus OPEN "geeignet" spezifizieren. Ebenso, wo genau s an CLOSED angehängt wird.

ii) In G4 kann man einen Lösungspfad angeben, wenn in G5 jeder direkte Nachfolger von s durch einen entsprechenden Zeiger auf s verweist.

iii) In G6 kann man genauer angeben, wo s' an OPEN angehängt wird. Dasselbe gilt für G7.

iv) Das Verlagern des schon untersuchten Knotens s' von CLOSED nach OPEN in G7 ist nur notwendig, wenn man bei der Suche alle Pfade (der jeweiligen Länge) berücksichtigen muß, etwa zur Bestimmung minimaler Lösungen:



Ist der Zustandsgraph ein Baum, kann man die Verwaltung von CLOSED sparen und G6 und G7 zusammenfassen.

v) Neben den unter ii) erwähnten rückwärtigen Zeigern kann es für die weitere Suche sinnvoll sein, neben dem Knoten s noch zusätzliche Informationen für die weitere Suche abzuspeichern, etwa die Länge des Pfades von s_0 nach s, die Kosten dieses Pfades, die schon erzeugten Nachfolger, etc. Wir gehen davon aus, daß diese Informationen gegebenenfalls in der Beschreibung von s enthalten sind.

Wir verfeinern jetzt GRAPHSEARCH:

Definition 6.12 Unter den Voraussetzungen von Def. 6.10 entsteht der Algorithmus zur *BREITENSUCHE* aus G1 bis G8 durch folgende Änderungen:

G3 Sei s der *erste Knoten* aus OPEN. Entferne s aus OPEN und hänge ihn an CLOSED an.

G6 Ist s' weder in OPEN noch in CLOSED, hänge ihn *hinten* an OPEN an.

G7 Ist s' in OPEN, lasse ihn dort. Ist s' in CLOSED, lösche ihn dort und hänge ihn *hinten* an OPEN an.

Bemerkung 6.13

i) Bei der BREITENSUCHE wird für jedes $n \in I\!N$ jeder Pfad $p = (s_0, \ldots, s_n)$ der Länge n erzeugt und zwar *vor jedem* Pfad der Länge $n + 1$.

ii) Existiert ein Lösung, dann findet man in G4 also einen kürzesten Lösungspfad.

iii) Merkt man sich für jeden erzeugten Knoten s' die Länge des Pfades von s_0 nach s', so kann man mit Hilfe des Beweises von Satz 6.8 die Länge eines minimalen Lösungspfades nach oben abschätzen und solange weitersuchen, bis alle Pfade dieser Länge untersucht sind. Man hat dann sogar *alle minimalen* Lösungen gefunden.

iv) BREITENSUCHE ist sehr speicheraufwendig und daher nur für kleine Suchräume praktikabel.

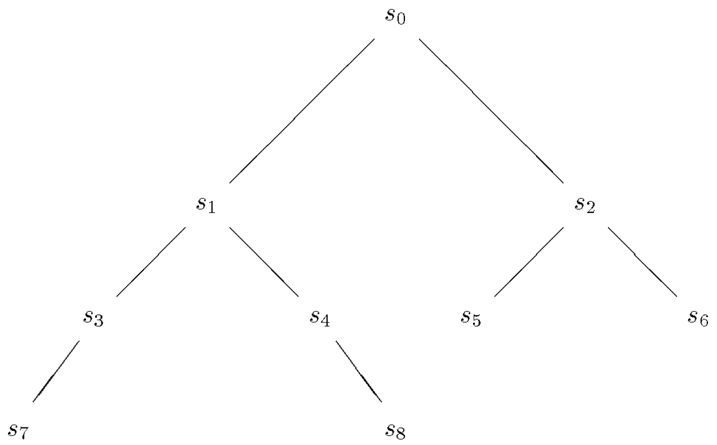
Definition 6.14 Unter der Voraussetzung von Def. 6.10 entsteht der Algorithmus TIEFENSUCHE aus G1 bis G8 durch folgende Änderungen:

G3 Sei s der *erste Knoten* aus OPEN. Entferne s aus OPEN und hänge ihn an CLOSED an.

G6 Ist s' weder in OPEN noch CLOSED, hänge ihn *vorn* an OPEN an.

G7 Ist s' in OPEN, lasse ihn dort. Ist s' in CLOSED, lösche ihn dort und hänge ihn *vorn* an OPEN an.

Beispiel 6.15 Der Zustandsgraph des Problems sei der folgende Baum, alle Kanten seien mit 1 bewertet, einziger Zielknoten sei s_8:



BREITENSUCHE:

OPEN	CLOSED
s_0	–
–	s_0
$s_1 s_2$	s_0
s_2	$s_0 s_1$
$s_2 s_3 s_4$	$s_0 s_1$
$s_3 s_4$	$s_0 s_1 s_2$
$s_3 s_4 s_5 s_6$	$s_0 s_1 s_2$
$s_4 s_5 s_6$	$s_0 s_1 s_2 s_3$
$s_4 s_5 s_6 s_7$	$s_0 s_1 s_2 s_3$
$s_5 s_6 s_7$	$s_0 s_1 s_2 s_3 s_4$
$s_5 s_6 s_7 s_8$	

TIEFENSUCHE:

OPEN	CLOSED
s_0	–
–	s_0
$s_1 s_2$	$s_0 s_1$
s_2	$s_0 s_1$
$s_3 s_4 s_2$	$s_0 s_1 s_3$
$s_4 s_2$	$s_0 s_1 s_3$
$s_7 s_4 s_2$	$s_0 s_1 s_3 s_7$
$s_4 s_2$	$s_0 s_1 s_3 s_7$
s_2	$s_0 s_1 s_3 s_7 s_4$
$s_8 s_2$	

Bemerkung 6.16 Bei reiner TIEFENSUCHE wird für $\mid S \mid = \infty$ i.a. keine Lösung gefunden, etwa wenn in Bsp. 6.15 an s_7 noch ein unendlicher Teilbaum hängt. Man modifiziert den Algorithmus daher weiter:

Der BACKTRACKING-Algorithmus entsteht aus TIEFENSUCHE dadurch, daß man in G6 und G7 nur den ersten noch nicht erzeugten Nachfolger von s behandelt und s in G3 noch nicht aus OPEN entfernt, sondern nur einen Verweis auf den nächsten Nachfolger legt. Man entfernt s erst dann aus OPEN, wenn keine weiteren Nachfolger mehr erzeugt werden können. Man speichert in OPEN dann nur Knoten, die jeweils auf einem Pfad liegen. Der Algorithmus ist also extrem speichergünstig, hat aber noch denselben Nachteil wie reine TIEFENSUCHE.

Beispiel 6.17

Für den Zustandsgraphen aus Bsp. 6.15 ergibt BACKTRACKING :

OPEN	Verweise
s_0	–
$s_1 s_0$	$s_0 \rightarrow s_2$
$s_3 s_1 s_0$	$s_0 \rightarrow s_2, s_1 \rightarrow s_4$
$s_7 s_3 s_1 s_0$	$s_0 \rightarrow s_2, s_1 \rightarrow s_4$
$s_3 s_1 s_0$	$s_0 \rightarrow s_2, s_1 \rightarrow s_4$
$s_1 s_0$	$s_0 \rightarrow s_2, s_1 \rightarrow s_4$
$s_4 s_1 s_0$	$s_0 \rightarrow s_2$
$s_8 s_4 s_1 s_0$	

Man modifiziert daher weiter:

Definition 6.18 Unter den Voraussetzungen von Def. 6.10 entsteht der Algorithmus zur *beschränkten TIEFENSUCHE* aus TIEFENSUCHE in Def. 6.14 dadurch, daß in G5 die direkten Nachfolger s' von s nur dann erzeugt werden, wenn $p = (s_0, \ldots, s, s')$ bzgl. seiner Länge (oder seines Wertes $w(p)$) noch unterhalb einer vorgegebenen Schranke liegt. Dieselbe Modifikation wendet man auf BACKTRACKING an.

Beispiel 6.19 Für den Zustandsgraphen aus Bsp. 6.15 ergibt sich bei beschränktem BACKTRACKING und einer Schranke von $w(p) \leq 2$:

OPEN	Verweise
s_0	–
$s_1 s_0$	$s_0 \rightarrow s_2$
$s_3 s_1 s_0$	$s_0 \rightarrow s_2, s_1 \rightarrow s_4$
$s_1 s_0$	$s_0 \rightarrow s_2, s_1 \rightarrow s_4$
$s_4 s_1 s_0$	$s_0 \rightarrow s_2$
$s_1 s_0$	$s_0 \rightarrow s_2$
s_0	$s_0 \rightarrow s_2$
$s_2 s_0$	–
$\vdots$	

Bemerkung 6.20 Setzt man die Schranke a priori "hoch genug", so findet beschränkte TIEFENSUCHE jedenfalls wie BREITENSUCHE eine Lösung und ist bei Einsatz von BACKTRACKING sehr speichergünstig. Man investiert aber einen hohen Zeitaufwand, wenn man die Schranke viel zu hoch ansetzt. Die Krönung der uniformierten Suche liefert daher:

Definition 6.21 Unter den Voraussetzungen von Def. 6.10 erhält man den Algorithmus zur *iterativen beschränkten TIEFENSUCHE* dadurch, daß man in einer Schleifenvariablen die Schranke für Tiefensuche iterativ erhöht und innerhalb dieser Schleife beschränkte Tiefensuche durchführt, solange bis eine Lösung gefunden wurde (oder OPEN leer wurde).

6.3 Heuristische Funktionen

Wir versuchen jetzt, Informationen über die Zustände für den Suchprozeß nutzbar zu machen (vgl. Def 6.1).

Definition 6.22 Sei (S, K, w) lokal endlich bewerteter Zustandsgraph des Problems (S, T, s_0, Z) und $P = \{p = (s_0, \ldots, s_n) \mid s_\nu \in S\}$ die Menge aller Pfade in (S, K) von s_0 zu einem beliebigen Knoten $s_n \in S$. Eine *(Knoten-)Bewertung* ist eine Funktion $f : P \to I\!R_+$.

Bemerkung 6.23 In der Literatur wird eine Knotenbewertung oft in der Form $f : S \to I\!R_+$ definiert. Dieses ist i.a. jedoch nicht korrekt, da für verschiedene Pfade $p = (s_0, \ldots, s_n) \neq (s_0, \ldots, s_m)$ mit $s_n = s_m$ durchaus $[f(s_n) =]f(p)\neg = f(p')[= f(s_m)]$ gelten kann. Der Wert eines Knotens kann sich nämlich im Verlauf der Suche ändern. Ist der Zustandsgraph jedoch ein Baum, dann entsprechen sich Pfad und Endknoten umkehrbar eindeutig und man kann f als Funktion auf S auffassen. In jedem Fall sei $f(s)$ stets gemeinsam mit s abgespeichert.

Definition 6.24 Unter den Voraussetzungen von Def. 6.22 entsteht der BEST-FIRST-Algorithmus aus G1 bis G8 durch folgende Änderungen:

G3 Wähle einen Knoten s aus OPEN mit *minimaler Bewertung* $f(s)$ aus, wenn möglich $s \in Z$, entferne s aus OPEN und hänge ihn an CLOSED an.

G5 Expandiere s und berechne für alle direkten Nachfolger s' von s jeweils $f(s') = f(s_0, \ldots, s, s')$ und führe G6 und G7 durch.

G6 Ist s' weder in OPEN noch in CLOSED, so hänge ihn mit seiner Bewertung $f(s')$ an OPEN an (evtl. sortiert).

G7 Ist s' in OPEN, so lasse ihn dort, setze aber die neue Bewertung $f(s')$ ein. Ist s' in CLOSED, lösche ihn dort und hänge ihn mit der neuen Bewertung $f(s')$ an OPEN an (evtl. sortieren).

Bemerkung 6.25

i) Führt man die Sortierungen stets durch, dann kann man wie bei BREITENSUCHE in G3 immer den ersten Knoten wählen.

ii) Setzt man $f(s_0) = 0$ und $f(s') = f(s) + 1$ in G5, dann liefert BEST-FIRST dasselbe wie BREITENSUCHE. Man hat durch f nämlich keinerlei Probleminformation eingebracht.

Bessere Bewertungsfunktionen liefern die folgenden Überlegungen:

Definition 6.26 Sei (S, K, w) lokal endlicher bewerteter Zustandsgraph des Problems (S, T, s_0, Z) und $s_n \in S$. Setze $P(s_n) := \{p = (s_0, \ldots, s_n)\}$ und $P'(s_n) := \{p' = (s_n, \ldots, s_m) \mid s_m \in Z\}$ für die Menge aller Pfade von s_0 nach s_n bzw. von s_n zu einem Zielknoten. Weiterhin seien $f^*, g^*, h^* : S \to I\!R_+ \cup \{\infty\}$ durch

$f^*(s_n) := g^*(s_n) + h^*(s_n)$,

$g^*(s_0) := 0$,

$$g^*(s_n) := \begin{cases} \infty, \text{ falls} P(s_n) = \emptyset \\ min\{w(p) \mid p \in P(s_n)\}, \text{ sonst,} \end{cases}$$

$h^*(s_n) := 0$ für $s_n \in Z$,

$$h^*(s_n) := \begin{cases} \infty, \text{ falls} P'(s_n) = \emptyset \\ min\{w(p') \mid p' \in P'(s_n)\}, \text{ sonst,} \end{cases}$$

definiert.

Bemerkung 6.27

i) Wegen Satz 6.8. sind g^*, h^* und damit f^* wohldefiniert.

ii) Es liefert $g^*(s_n)$ die minimalen Werte für alle Pfade von s_0 nach s_n, $h^*(s_n)$ die minimalen Werte für alle Pfade von s_n zu einem Zielknoten und daher $f^*(s_n)$ die minimalen Werte für alle Pfade von s_0 zu einem Zielknoten, die über s_n laufen. Liegt insbesondere s_n auf einem optimalen Pfad, so gilt $f^*(s_n) = f^*(s_0) = h^*(s_0)$.

iii) Benutzt man f^* als Knotenbewertung für BEST-FIRST, dann findet dieser Algorithmus *direkt* einen minimal bewerteten Pfad. In der Praxis sind aber weder g^* noch h^* bekannt. Man muß sie und damit f^* daher "möglichst gut" schätzen, um mit BEST-FIRST schnell solche Lösungen zu finden.

iv) Man definiert $g : D_g \to I\!R_+, D_g \subseteq S$ als partielle Funktion während der Suche für jeden erzeugten Knoten s' durch $g(s') = min\{w(p) \mid p \in P(s'), p \text{ untersucht}\}$. Es folgt $g(s') \geq g^*(s')$ und $g(s_0) = 0$.

Definition 6.28 Jede Funktion $h : S \to I\!R_+ \cup \{\infty\}$ mit $h(s) = 0$ für alle $s \in Z$ heißt *heuristische Funktion.* Sie heißt *zulässig* oder *optimistisch*, wenn $h(s) \leq h^*(s)$ für alle $s \in S$ gilt, sie heißt *pessimistisch*, wenn $h^*(s) \leq h(s)$ für alle $s \in S$ gilt; sie *erfüllt die monotone Restriktion*, wenn für alle $s \in S$ und alle direkten Nachfolger s' von s bereits $h(s) - h(s') \leq w(s, s')$ gilt.

Satz 6.29 Ist $h : S \to I\!R_+ \cup \{\infty\}$ heuristische Funktion, die die monotone Restriktion erfüllt, dann ist h optimistisch. *Beweis:*
Sei $s_n \in S$ und $p' = (s_n, \ldots, s_m)$ ein beliebiger Pfad. Dann gilt $h(s_{\mu-1}) \leq w(s_{\mu-1}, s_\mu) + h(s_\mu)$ für $\mu = n+1, \ldots, m$. Es folgt $h(s_n) \leq w(s_n, s_{n+1}) + h(s_{n+1} \leq \ldots \leq w(p') + h(s_m)$. Für $s_m \in Z$ gilt also $h(s_n) \leq w(p')$. Mit Def. 6.26 folgt $h(s_n) \leq h^*(s_n)$.

Definition 6.30 Unter den Voraussetzungen von Def. 6.22 sei $h : S \to I\!R_+ \cup \{\infty\}$ eine heuristische Funktion, $g : D_g \to I\!R_+$ wie in Bem. 6.27 iv) und $f = g + h$. Der *A^* - Algorithmus* (mit Parameter h) ist wie folgt definiert:

A1 Setze OPEN $:= (s_0)$, CLOSED $:= (), g(s_0) := 0, f(s_0) := h(s_0)$.

A2 Ist OPEN $= ()$, dann existiert keine Lösung.

A3 Wähle einen Knoten s aus OPEN mit minimalem $f(s)$, wenn möglich $s \in Z$, entferne s aus OPEN und hänge ihn an CLOSED an.

A4 Falls $s \in Z$, ist $(s, g(s))$ das Ergebnis.

A5 Expandiere s und berechne für jeden direkten Nachfolger s' von s den Wert $g_{s'} := g(s) + w(s, s')$. Führe A6 bzw. A7 durch.

A6 Ist s' nicht in OPEN oder in CLOSED, setze $g(s') := g_{s'}$ und $f(s') := g_{s'} + h(s')$ und hänge s' an OPEN an.

A7 Ist s' in OPEN oder in CLOSED und $g(s') > g_{s'}$, dann setze $g(s') := g_{s'}$ und $f(s') := g_{s'} + h(s')$. Ist s' in OPEN, lasse ihn dort. Ist s' in CLOSED, entferne ihn dort und hänge ihn an OPEN an (evtl. einsortieren).

A8 Goto A2.

Beispiel 6.31 Für das 8er-Puzzle sind etwa folgende heuristischen Funktionen möglich.
1) $h(s) = 0 (\Rightarrow BREITENSUCHE)$,
2) $h(s) =$ Anzahl der falsch plazierten Zahlen,
3) $h(s) =$ Summe der "Manhattendistanzen" über alle Zahlen.
Dabei wird die Manhattandistanz einer Zahl zu ihrem Ziel als die Anzahl der mindestens notwendigen Züge von ihrem augenblicklichen Platz an ihr Ziel definiert. Dies ist eine Metrik!

3	5	7
8	1	2
6	4	

1	2	3
8		4
7	6	5

Für die 1 ergibt sich diese Distanz zu2, für die 5 zu 3, für die 8 zu 0 etc.

Satz 6.32 Ist das Problem (S, T, s_0, Z) lösbar, dann terminiert der A^* - Algorithmus für beliebiges h in A4 mit einem $s \in Z$. Ist h optimistisch, dann ist die gefundene Lösung $(s_0, \ldots, s)$ sogar optimal und es gilt $g(s) = h^*(s_0)$. *Beweis:*
Nach Satz 6.8 existiert ein Lösungspfad $p = (s_0, \ldots, s_m), s_m \in Z$ mit minimaler Bewertung. Dann existiert zu jedem Zeitpunkt, wenn A2 durchlaufen wird, ein Knoten s_μ aus p in OPEN. Dieses folgt durch Induktion. Zunächst gilt die Behauptung für s_0. Gilt sie später für einen Knoten s_μ, der in A3 aus OPEN entfernt wird, dann ist $s_{\mu+1}$ aus p direkter Nachfolger von s_μ und gelangt in A6 oder A7 nach OPEN, bevor der A^*-Algorithmus über A8 wieder nach A2 gelangt. Der Algorithmus terminiert also niemals in A2.

Annahme: *A^* terminiert auch nicht in A4, also niemals.*
Dann wird A3 unendlich oft durchlaufen, also unendlich viele Knoten aus OPEN ausgewählt. Dabei können nicht alle Knoten aus p gewählt werden, da A^* sonst in A4 mit $s = s_m$ terminiert. Sei s_μ der erste Knoten aus p, der in OPEN liegt und niemals gewählt wird. Setze $d := \frac{f(s_\mu)}{\epsilon}$.
Für jeden Knoten $s \in S$, der von s_0 aus nur durch einen Pfad einer Länge $l > d$ zu erreichen ist, gilt $f(s) = g(s) + h(s) \geq g(s) \geq l * \epsilon > d * \epsilon = f(s_\mu)$. Falls s also in OPEN liegt, wird s niemals vor s_μ ausgewählt. Es gibt aber nur endlich viele Knoten, die von s_0 aus durch einen Pfad einer Länge $l \leq d$ zu erreichen sind. Also können in A3 auch nur endlich viele Knoten vor s_μ ausgewählt werden, d.h. s_μ wird doch einmal gewählt. (**Widerspruch !!**) Also terminiert A^* in A4.

Sei nun h optimistisch und s_μ Knoten in OPEN aus dem optimalen Lösungspfad p. Dann gilt $g(s_\mu) = g^*(s_\mu)$ und daher $f(s_\mu) = g(s_\mu) + h(s_\mu) \leq g^*(s_\mu) + h^*(s_\mu) = f^*(s_\mu) = f^*(s_0)$. Es gibt also stets Knoten s_μ in OPEN mit $f(s_\mu) \leq f^*(s_0)$.

Annahme: Der in A4 gefundene Pfad $p' = (s_0, \ldots, s)$ ist nicht optimal. Dann gilt $f(s) = g(s) + h(s) = g(s) > g^*(s) = f^*(s) \geq f^*(s_0)$.
Also wurde s in A3 nicht vor den Knoten s_μ aus p aus OPEN gewählt, insbesondere nicht vor s_m. Dann terminiert aber A^* in A4 mit $p = (s_0, \ldots, s_m) \neq p'$. (**Widerspruch !!**)
Es gilt also einerseits $f(s_m) = g(s_m) \leq f^*(s_0) = h^*(s_0)$ und andererseits $f(s_m) = g(s_m) \geq g^*(s_m) \geq f^*(s_0) = h^*(s_0)$, d.h. $h^*(s_0) = g(s)$ für $s0 s_m \in Z$.

Bemerkung 6.33 Der Beweis von Satz 6.32 zeigt noch:

a) Für jeden in A3 ausgewählten Knoten s gilt $f(s) \leq f^*(s_0) = h^*(s_0)$.

b) Jeder Knoten s aus OPEN mit $f(s) < h^*(s_0)$ wird tatsächlich expandiert.

c) Kein Knoten s mit $f(s) > h^*(s_0)$ wird jemals expandiert.

d) Ein Knoten $s_n \in S$ wird expandiert, wenn es einen Pfad $p = (s_0, \ldots, s_n)$ gibt, so daß für jeden Knoten s_ν aus p schon $f(s_\nu) < h^*(s_0)$ gilt.

e) Damit ein Knoten $s_n \in S$ expandiert werden kann, muß es einen Pfad $p = (s_0, \ldots, s_n)$ geben, für dessen Knoten s_ν auch $f(s_\nu) \leq h^*(s_0)$ gilt.

Definition 6.34 Seien $h_1, h_2 : S \to I\!R_+ \cup \{\infty\}$ optimistische heuristische Funktionen. Dann heißt h_2 *besser informiert als* h_1, wenn $h_1(s) < h_2(s)$ für alle $s \notin Z$ gilt.
Bezeichnen A_1^* bzw. A_2^* jeweils den A^*-Algorithmus mit h_1 bzw. h_2 als Parameter, dann heißt A_2^* *besser informiert* als A_1^*.

Satz 6.35 Sei (S, T, s_0, Z) lösbar. Ist A_2^* *besser informiert* als A_1^*, dann wurde jeder Knoten, der von A_2^* expandiert wurde, auch von A_1^* expandiert, nachdem beide Algorithmen terminierten.
Beweis:
Beweise die Behauptung für alle von A_2^* expandierten Knoten $s = s_n$ durch Induktion über die Länge der von A_2^* untersuchten Pfade $p = (s_0, \ldots, s_n)$.
$n = 0 : s_n = s_0$; falls $s_0 \in Z$ ist, terminieren beide Algorithmen sofort, ohne einen Knoten zu expandieren, falls $s_0 \notin Z$ ist, expandieren beide Algorithmen s_0.
$n \geq 1$: Nach Induktionsvoraussetzung wurde jeder Knoten $s_0, \ldots, s_{n-1}$ sowohl von A_2^* als auch von A_1^* expandiert. Also ist $g_1(s_n) \leq g_2(s_n)$.
Nimmt man nun an, s_n wurde zwar von A_2^*, aber nicht von A_1^* expandiert, so gilt $s_n \in OPEN$ für A_1^*. Da A_1^* terminierte, gilt nach Bem. 6.33 b): $g_1(s_n) + h_1(s_n) = f_1(s_n) \geq h^*(s_0)$, woraus $h_1(s_n) \geq h^*(s_0) - g_1(s_n) \geq h^*(s_0) - g_2(s_n)$ folgt.
Aus Bem. 6.33 a) ergibt sich $g_2(s_n) + h_2(s_n) = f_2(s_n) \leq h^*(s_0)$, also der Widerspruch $h_2(s_n) \leq h^*(s_0) - g_2(s_n) \leq h_1(s_n)$.

Satz 6.36 Sei (S, T, s_0, Z) lösbar und die heuristische Funktion h erfülle die monotone Restriktion. Für den A^*-Algorithmus mit Parameter h gelten dann:

i) $g(s) = g^*(s)$ für alle $s \in CLOSED$.

ii) $f(s) \leq f(s')$, falls s' nach s expandiert wurde.

iii) $g^*(s) + h(s) \leq h^*(s_0)$, falls s expandiert wurde.

iv) Jeder Knoten $s \in S$ mit $g^*(s) + h(s) < h^*(s_0)$ wird schließlich expandiert.

Beweis:

i) Sei $p = (s_0, \ldots, s_n)$ mit $s_n = s \in CLOSED$ ein optimaler Pfad. Gilt $g(s_n) > g^*(s_n)$, dann muß OPEN mindestens einen weiteren Knoten aus p enthalten. Sei s_μ der erste davon. Es gilt also $g(s_\mu) = g^*(s_\mu)$ und daher
$f(s_\mu) = g^*(s_\mu) + h(s_\mu) \leq g^*(s_\mu) + w(s_\mu, s_{\mu+1}) + \ldots + w(s_{n-1}, s_n) + h(s_n) = g^*(s_n) + h(s_n) < g(s_n) + h(s_n) = f(s_n)$.
Dann wurde aber s_n noch nicht expandiert und kann sich nicht in CLOSED befinden, womit man einen Widerspruch erhalten hat. ($\Rightarrow$ Behauptung)

ii) Es reicht, den Fall zu betrachten, daß s' direkt nach s expandiert wurde. Der Rest folgt dann durch Induktion. Befanden sich beide in OPEN als s expandiert wurde, ist die Behauptung klar. Sonst ist s' direkter Nachfolger von s. Es folgt
$f(s) = g(s) + h(s) \leq g(s) + w(s, s') + h(s') = g(s') + h(s') = f(s')$.

iii) Dies folgt aus Bem. 6.33 a) und i): $f(s) = g(s) + h(s) = g^*(s) + h(s) \leq h^*(s_0)$.

iv) Sei wiederum $p = (s_0, \ldots, s_n)$ mit $s = s_n$ ein optimaler Pfad. Es ist $g^*(s_n) + h(s_n) = g^*(s_{n-1}) + w(s_{n-1}, s_n) + h(s_n) \geq g^*(s_{n-1}) + h(s_{n-1})$, also auch $g^*(s_{n-1}) + h(s_{n-1}) < h^*(s_0)$. Durch Induktion folgt $g^*(s_\nu) + h(s_\nu) < h^*(s_0)$ für alle s_ν aus p. Sei s_ν der erste dieser Knoten, der nach Terminierung von A^* noch auf OPEN liegt. Da auch $p' = (s_0, \ldots, s_\nu)$ optimal ist, gilt $g(s_\nu) = g^*(s_\nu)$, also $f(s_\nu) = g(s_\nu) + h(s_\nu) < h^*(s_0)$.
Nach Bem. 6.33 b) wurde aber s_ν expandiert. Dieses führt wiederum zum Widerspruch. ($\Rightarrow$ Auch s_n wurde expandiert).

In [KK88], S. 200 ff werden einige Varianten des A^*-Algorithmus vorgestellt und ihre Eigenschaften verglichen. So ist die Zeitkomplexität des A^*-Algorithmus im ungünstigsten Fall von der Größenordnung $O(2^{|S|})$, während die beiden gleich angegebenen Varianten C und D nur eine quadratische Zeitkomplexität haben.

Definition 6.37 Unter der Voraussetzung von Def. 6.30 seien die Algorithmen C und D wie folgt definiert:

$C1$ Setze OPEN := (s_0); CLOSED := $()$; $g(s_0) := 0$; $f(s_0) := h(s_0)$; $F := 0$; (F ist eine globale Variable, die das Maximum der f - Werte bisher expandierter Knoten enthält $\Rightarrow F \leq h^*(s_0)$).

$C2$ $= A2$

$C3$ Falls ein Knoten s aus OPEN mit $f(s) \leq F$ existiert, dann wähle unter allen solchen Knoten einen aus, dessen Wert $g(s)$ minimal ist.
Sonst wähle unter allen Knoten aus OPEN mit minimalem $f(s)$ einen aus, dessen Wert $g(s)$ minimal ist und setze $F := f(s)$. Bevorzuge bei der Auswahl aber stets Knoten $s \in Z$. Entferne s aus OPEN und hänge ihn an CLOSED an.

C_n $= A_n, n = 4, \ldots, 8$.

$D1$ $= A1$

$D2$ $= A2$

$D3$ Wähle unter den Knoten s aus OPEN mit minimalem $f(s)$ einen aus, dessen Wert $g(s)$ minimal ist, wenn möglich $s \in Z$. Entferne s aus OPEN und hänge ihn an CLOSED an.

$D4$ $= A4$

$D5$ Expandiere s. Wenn s keine direkten Nachfolger besitzt, setze $h(s) = \infty$ und gehe nach $D2$.
Für jeden direkten Nachfolger s' von s berechne $g_{s'} := g(s) + w(s, s')$. Berechne $m := \min_{s'}\{w(s, s') + h(s')\}$. Falls $h(s) < m$ setze $h(s) := m$, sonst setze für jedes s' mit $h(s') < h(s) - w(s, s')$ bereits $h(s') := h(s) - w(s, s')$.

D_n $= A_n, n = 6, 7, 8$.

Bemerkung 6.38

i) In C wird in der Bewertung durch $f = g + h$ der g -Anteil bevorzugt. Man beachte, daß nach Bem. 6.33 b) jeder Knoten s aus OPEN mit $f(s) < F \leq h^*(s_0)$ sowieso *einmal* expandiert werden muß. Fraglich ist wie oft!

ii) In D5 wird die Schätzung von $h^*(s)$ durch $h(s)$ verbessert, denn es gibt
$h(s) \leq h^*(s), h(s') \leq h^*(s') \Rightarrow m \leq h^*(s)$.
Durch die Neuberechnung von $h(s')$ wird versucht, die *monotone Restriktion* "lokal" zu erfüllen!

6.4 UND/ODER - Graphen

In manchen Problemräumen (S,T) gibt es *Zerlegungstransformationen* $t \in T$, die ein gestelltes Problem $s \in S$ nicht in ein anderes $s' \in S$ transformieren, sondern die s in eine endliche Menge $\{s'_1, \ldots, s'_k\}$ von (einfacheren) Teilproblemen zerlegen. Bei der Suche nach einer Lösung muß dann nicht nur ein Lösungspfad, sondern mehrere Lösungspfade parallel gefunden werden.

Beispiel 6.39 Die Menge S bestehe aus integrierbaren Funktionen, deren unbestimmtes Integral zu berechnen sei. In T seien die bekannten Integrationsregeln beschrieben, etwa $\int c \cdot f\ dx = c \cdot \int f\ dx$, $\int (f+g)\ dx = \int f\ dx + \int g\ dx$, Substitution, part. Integration etc. Die Menge Z bestehe aus den Integralen der "einfachen" Funktionen, z.B. in einer Integraltafel gesammelt. Gesucht sei das Integral:

$$s_0 = \int \frac{sin^4(x)}{cos^4(x)}\ dx$$

Dann kann sich etwa folgender Suchbaum ergeben:

$$\begin{tikzcd}
 & & s_0 \arrow[d, "y = tan(x)"] & & \\
 & & s_1 = \int \frac{y^4}{1+y^2}\ dy \arrow[d, "\text{Partialbruchzerlegung}"] & & \\
 & & s_2 = \int (-1 + y^2 + \frac{1}{1+y^2})\ dy \arrow[dl] \arrow[d] \arrow[dr] & & \\
 s_6 = \int 1\ dy & s_3 = \int -1\ dy \arrow[l] & s_4 = \int y^2\ dy & s_5 = \int \frac{1}{1+y^2}\ dy \arrow[r, "y = tan(z)"] & s_7 = \int 1\ dz
\end{tikzcd}$$

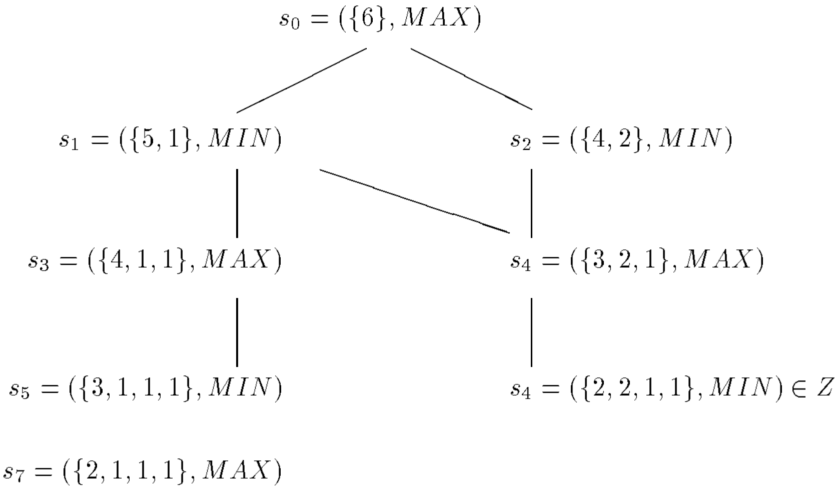
Die jeweils letzten Integrale liegen in Z und die Lösung ergibt sich aus allen drei Pfaden.

Um solche Strukturen behandeln zu können, führt man die Datenstruktur des UND/ODER - Graphen ein. Dazu noch einige Vorbereitungen:

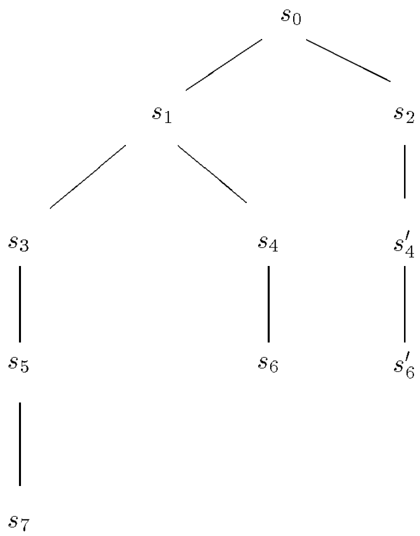
Definition 6.40 Es sei (S,K) ein gerichteter Graph. Ein Knoten $s \in S$ heißt *terminaler Knoten*, wenn er keinen direkten Nachfolger hat, andernfalls *nichtterminaler Knoten*. (S,K) heißt *azyklisch*, wenn kein Knoten sein eigener Nachfolger ist.
Ein *Baum* ist ein azyklischer gerichteter Graph (S,K), in dem jeder Knoten bis auf einen ausgezeichneten Knoten $s_0 \in S$, der *Wurzel* des Baumes, genau einen direkten Vorgänger hat. Die terminalen Knoten eines Baumes heißen auch *Blätter*.

Bemerkung 6.41 Jeder azyklische gerichtete Graph (S,K) kann durch *Entfaltung*, d.h. Vervielfachung von Knoten, die mehrere direkte Vorgänger haben, in einen äquivalenten Baum $entf(S,K)$ überführt werden.

Beispiel 6.42 Zwei Personen MAX und MIN spielen *"Grundy's Game"*:
Am Anfang liegt vor ihnen ein Stapel Spielmarken. Jeder Spieler wählt nun abwechselnd einen der vorhandenen Stapel aus und teilt ihn in zwei Stapel mit *ungleicher* Anzahl von Spielmarken. Ist einem Spieler dies nicht mehr möglich, so hat er verloren.



Entfaltung:



Definition 6.43

i) Ein Problemraum (S,T) heißt *UND/ODER-Problemraum*, wenn für jedes $s \in S$ die folgende Alternative gilt.

   Entweder (1) kein $t \in T$ mit $s \in D_t$ ist eine Zerlegungstransformation
   oder (2) es gibt genau eine Zerlegungstransformation $t \in T$ mit $s \in D_t$.

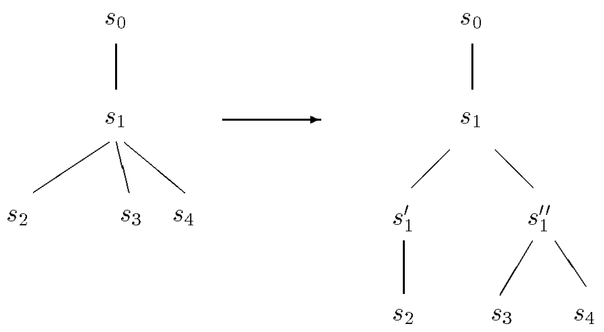
ii) Für einen UND/ODER-Problemraum ist die Kantenrelation K im zugehörigen Zustandsgraphen (S,K) definiert durch
   $(s,s') \in K \Leftrightarrow$ ex. $t \in T$ mit $s' = st$ oder $s' \in st = \{s_1', \ldots, s_k'\}$.

Ein nichtterminaler Knoten $s \in S$ mit (1) heißt ODER-Knoten, einer mit (2) heißt UND-Knoten.

Bemerkung 6.44

i) Man betrachtet allgemeiner beliebige gerichtete Graphen als UND/ODER-Graphen, wenn die (nichtterminalen) Knoten mit Marken "UND" bzw. "ODER" markiert werden können. Diese lassen sich (noch abstrakter) als gerichtete Hypergraphen (S, H) auffassen, wobei die *Hyperkantenrelation* H als Teilmenge von $S \times (\mathcal{P}(S) \setminus \{\emptyset\})$ definiert ist, also $(s, S') \in H$ für gewisse $s \in S$ und $\emptyset \neq S' \subseteq S$.

ii) Gibt es in einem Problemraum (S, T) einen Zustand s_1, auf den sowohl eine Zerlegungstransformation t_1 als auch eine übliche Transformation t_2 anwendbar ist, dann führt man Hilfszustände s_1', s_1'' ein, um (1) bzw. (2) zu erreichen:



Entsprechend verfährt man für je zwei (oder mehrere) Zerlegungstransformationen.

Beispiel 6.45

i) In Bsp. 6.39 sind s_0, s_1, s_3, s_5 ODER-Knoten und s_2 ist ein UND-Knoten.

ii) Im Spielbaum aus Bsp. 6.42 kann man die nichtterminalen "MAX-Knoten" als ODER-Knoten und die "MIN-Knoten" als UND-Koten auffassen. MAX kann nämlich unter seinen Zügen einen auswählen, muß aber *jeden* Gegenzug von MIN berücksichtigen.

Wir setzen ab jetzt voraus, daß (S, T) für das zu betrachtende Problem (S, T, s_0, Z) ein UND/ODER-Problemraum ist und der zugehörige UND/ODER-Zustandsgraph (S, K) azyklisch ist. Dann können wir ihn nach Bem. 6.41 als UND/ODER-Baum auffassen. Dabei dürfen wir s_0 als Wurzel und jedes $s \in Z$ als Blatt annehmen.

Definition 6.46 Es sei (S, K) UND/ODER-Baum zum Problem (S, T, s_0, Z). Eine *Lösung* von (S, T, s_0, Z) ist ein endlicher Teilbaum S' von S mit der Wurzel s_0 und folgenden Eigenschaften:

(3) Jeder terminale Knoten $s \in S'$ liegt in Z.

(4) Für jeden nichtterminalen UND-Knoten $s \in S' \subseteq S$ liegen sämtliche direkten Nachfolger s' von s in S bereits in S'.

(5) Für jeden nichtterminalen ODER-Knoten $s \in S' \subseteq S$ liegt genau einer der direkten Nachfolger s' von s in S auch in S'.

Um wieder optimale Lösungen finden zu können, muß eine Bewertung der Lösungen eingeführt werden. Dieses geschieht rekursiv:

Definition 6.47 Es sei (S, K) UND/ODER-Baum zum Problem (S, T, s_0, Z) und $c : T \rightarrow \mathbb{R}_+$, eine *Bewertung der Transformationen*. Für jeden Knoten s einer Lösung $S' \subseteq S$ von (S, T, s_0, Z) ist die *(Knoten-)Bewertung* $k(s)$ wie folgt definiert:

i) Ist $s \in Z$, dann ist $k(s) = 0$.

ii) Ist $s \in S'$ ein UND-Knoten und $t \in T$ die Zerlegungstransformation mit $st \subseteq S'$, dann ist

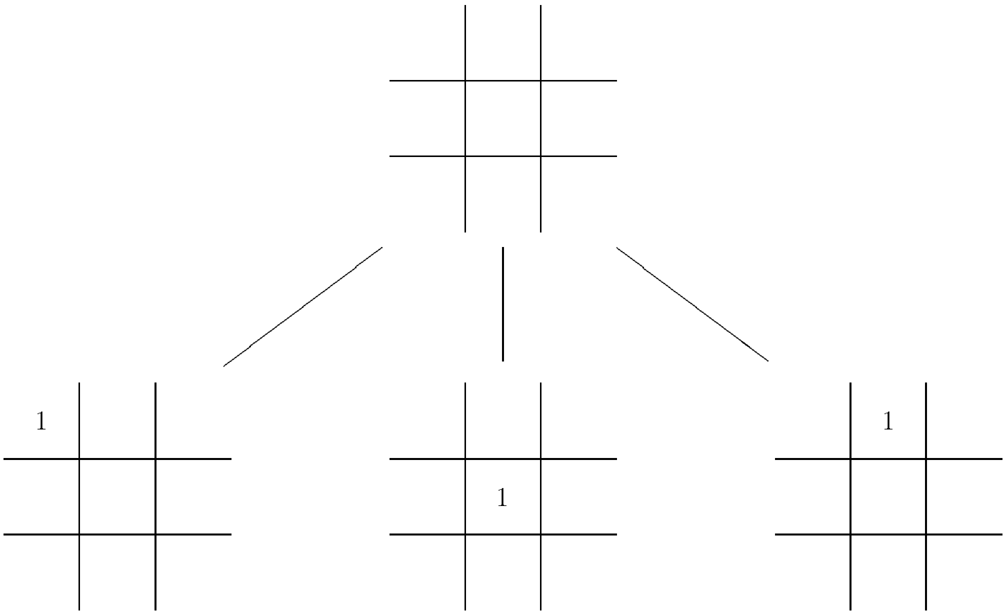
$$k(s) := c(t) + \sum_{s' \in st} k(s').$$

iii) Ist $s \in S'$ ein ODER-Knoten und $t \in T$ die Transformation mit $st = s \in S'$, dann ist $k(s) := c(t) + k(s')$.

Weiterhin sei $k(s') := k(s_0)$ die *Bewertung der Lösung* S'.

Bemerkung 6.48 In [Nil82] und auch [KK88] wird ein sogenannter AO^*-Algorithmus zur heuristischen Suche in UND/ODER-Graphen angegeben, der für ein gemäß Def. 6.47 bewertetes lösbares Problem stets eine optimale Lösung S' findet, d.h. eine, für die $k(S')$ minimal wird. Dazu ist u.a. auch der Begriff der heuristschen Funktion geeignet anzupassen.
Hier sollen statt dessen zwei ähnliche Techniken besprochen werden, die zur Suche nach optimalen Zügen in Zwei-Personen-Spielen mit vollständiger Information dienen. Als Demonstrationsobjekt nehmen wir ein hinreichend komplexes Spiel mit einfachen Regeln.

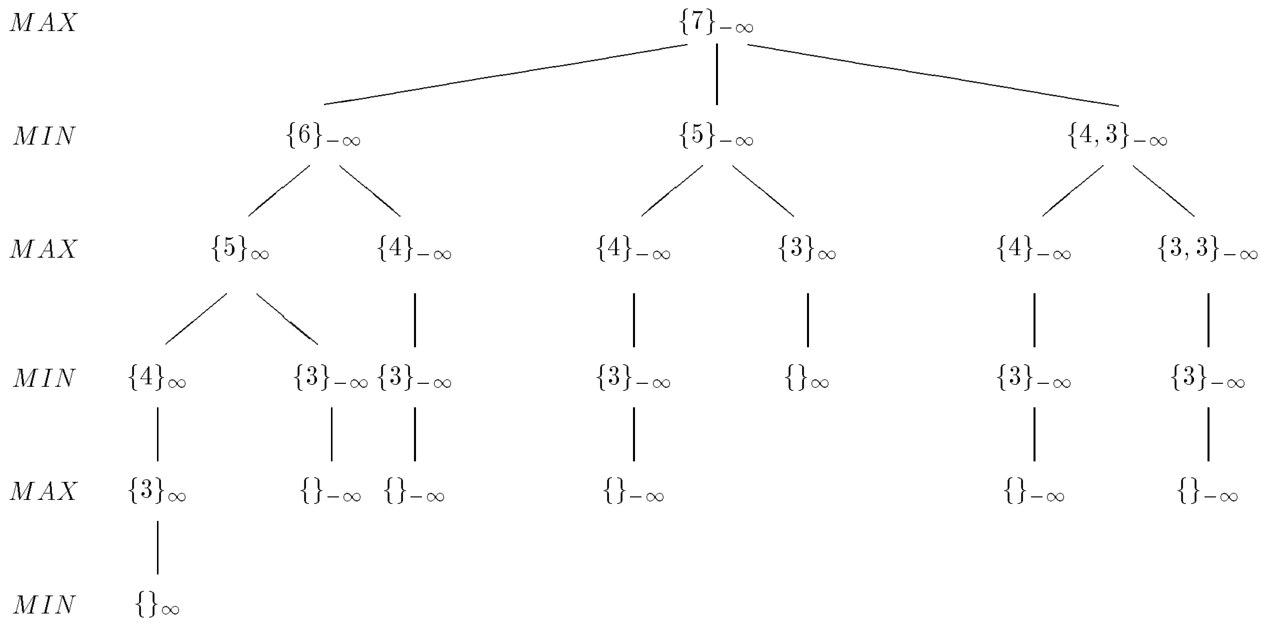
Beispiel 6.49 Tic-Tac-Toe (für Mathematiker/Informatiker).
Zwei Personen MAX und MIN nehmen abwechselnd Einträge in die noch freien Stellen einer 3×3 - Matrix vor. MAX trägt jeweils eine 1 ein, MIN eine 0. Gewonnen hat derjenige Spieler, der als erster eine Zeile, eine Spalte oder eine der beiden Diagonalen mit seiner Ziffer gefüllt hat. Wie immer beginnt MAX. Berücksichtigt man die Symmetrie der Situationen, dann hat MAX für den ersten Zug nicht 9, sondern nur 3 Möglichkeiten.



Die Komplexität des Suchbaumes kann man z.B. durch die Anzahl der möglichen Blätter abschätzen. Diese ist (ohne Symmetriebetrachtungen) 9!, also von der gleichen Größe wie beim 8er-Puzzle. Auch hier ist leicht eine Verallgemeinerung auf $n \times n$ - Matrizen und sogar auf Felder in mehr als zwei Dimensionen möglich. (Bei einer Variante dieses Spiels für Physiker wird die Schwerkraft berücksichtigt, d.h. ein Eintrag darf nur jeweils an tiefsten freien Stellen in jeder Spalte erfolgen. Für Informatiker: die Spalten sind Stacks!)

Definition 6.50 Die Bewertung eines *vollständigen* Spielbaumes S erfolgt ausgehend von seinen Blättern s rekursiv. Stellt s eine Gewinnsituation für MAX dar, wird es mit $c(s) = \infty$ bewertet, stellt es eine Gewinnsituation für MIN dar, mit $c(s) = -\infty$. Alle Remis-Situationen erhalten den Wert 0. Dann wird jeder MIN-Knoten mit dem Minimum der Werte seiner direkten Nachfolger bewertet und jeder MAX-Knoten mit dem Maximum. Bei $c(s_0) = \infty$ kann MAX das Spiel immer gewinnen, bei $c(s_0) = -\infty$ dagegen MIN. Bei $c(s_0) = 0$ kann MAX bestenfalls ein Remis erreichen, wenn auch MIN optimal spielt.

Beispiel 6.51 Grundy 's Game aus Bsp. 6.42.



Bei großen Spielbäumen ist eine vollständige rekursive Bewertung nicht möglich. Man nimmt daher eine *geeignete (statische)* Bewertung $c : S \rightarrow I\!R \cup \{-\infty, \infty\}$ der Knoten vor, die aber für Blätter mit der Bewertung aus Def. 6.50 übereinstimmt.

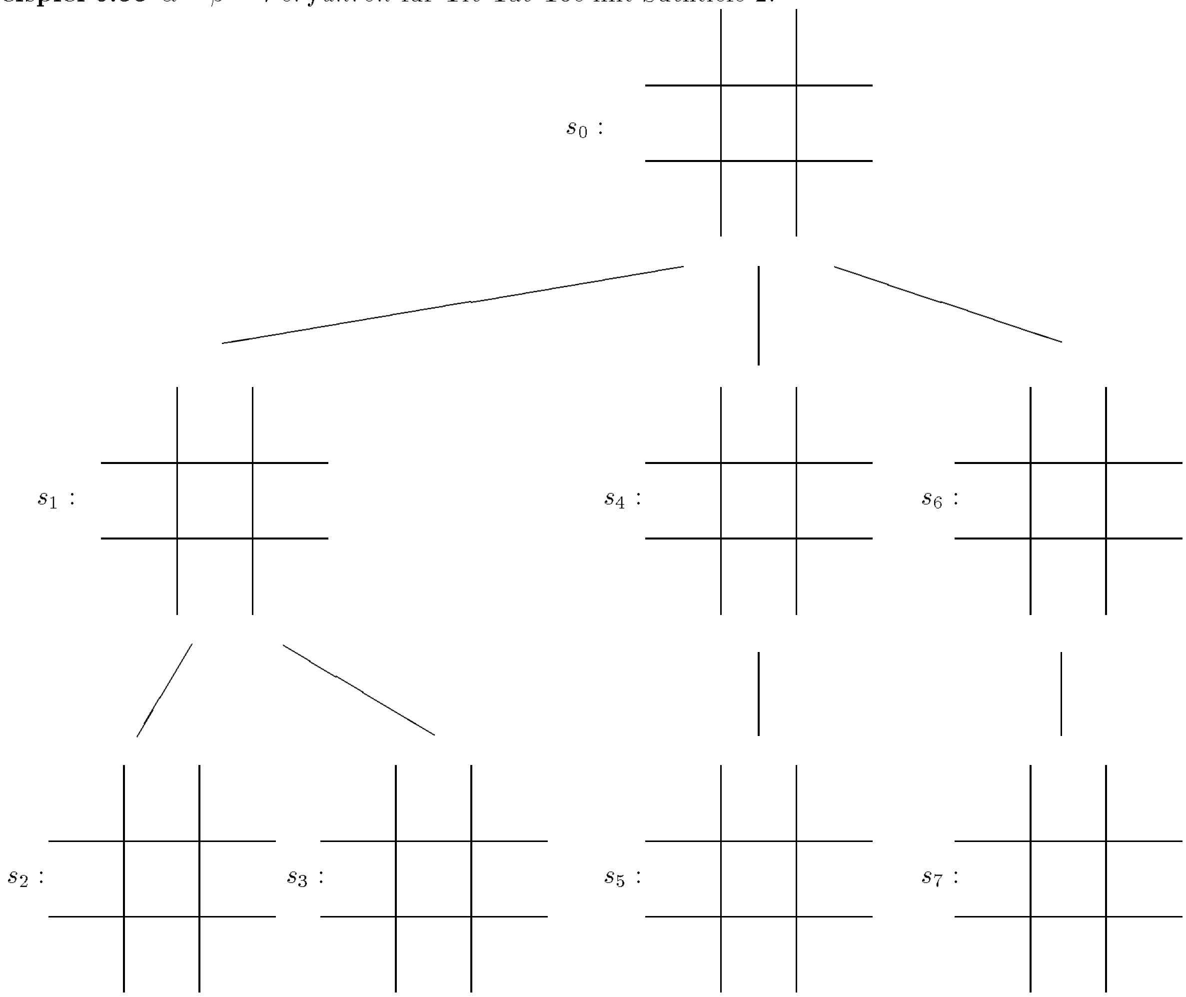
Beispiel 6.52 Für Situationen s des Tic-Tac-Toe-Spiels ist eine geeignete Bewertung $c : S \rightarrow I\!R \cup \{-\infty, \infty\}$ gegeben durch $c(s) = -\infty$, falls s Gewinnsituation für MIN, $c(s) = \infty$, falls s Gewinnsituation für MAX ist. Für jede andere Situation s ist $c(s) =$ *Anzahl der vollständigen Zeilen, Spalten und Diagonalen, die für MAX noch möglich sind - Anzahl der vollständigen Zeilen, Spalten und Diagonalen, die für MIN noch möglich sind.* Bei optimaler Strategie wählt dann MAX einen Zug $s \rightarrow s'$, für den $c(s')$ maximal wird. (In beliebig bewerteten Graphen ist dieses Verfahren unter dem Namen HILL-CLIMBING bekannt).

Definition 6.53 Beim MINIMAX-Verfahren wird der jeweilige Zug $s \rightarrow s'$ von MAX wie folgt bestimmt. Zunächst wird ein partieller Spielbaum S' mit der Wurzel s durch *beschränkte BREITENSUCHE* bis zu einer vorgegebenen Tiefe (Horizont) ermittelt. Die Blätter von S' werden mit der ebenfalls gegebenen Bewertungsfunktion $c : S \rightarrow I\!R \cup \{-\infty, \infty\}$ bewertet. Anschließend erfolgt die rekursive Bewertung aller anderen Knoten von S' wie in Def. 6.50. MAX wählt dann einen Zug $s \rightarrow s'$, für den $c(s')$ maximal wird.

Definition 6.54 Beim *$\alpha - \beta$-Verfahren* wird der jeweilige Zug $s \rightarrow s'$ von MAX wie folgt bestimmt. Der partielle Spielbaum S' mit der Wurzel s wird mit *beschränkter TIEFENSUCHE* durchsucht. Jedem Blatt s' von S' wird der Wert $c(s')$ zugeordnet und diese Werte werden wie in Def. 6.50 an ihre direkten Vorgänger weitergegeben. Jedem nichtterminalen MAX-Knoten s' wird ein *α-Wert* $\alpha(s')$ zugeordnet, jedem nichtterminalen MIN-Knoten ein *β-Wert* $\beta(s')$. Dabei ist $\alpha(s')$ eine untere Schranke für die noch möglichen c-Werte $c(s')$ und $\beta(s')$ eine obere Schranke. Also ist $\alpha(s') = -\infty$ bzw. $\beta(s') = \infty$ bei erstmaliger Expansion von s'. Der augenblickliche α-Wert eines MAX-Knotens ist gleich dem Maximum der c-Werte seiner bereits erzeugten direkten Nachfolger, der β-Wert eines MIN-Knotens entsprechend gleich dem Mininmum. (Daher können die α-Werte im Verlauf der Suche nur wachsen, die β-Werte nur fallen.) Jeder MIN-Knoten s', dessen

β-Wert kleiner oder gleich dem α-Wert eines seiner Vorgänger ist, wird nicht weiter expandiert und erhält den c-Wert $c(s') = \beta(s')$. Jeder MAX-Knoten s', dessen α-Wert größer oder gleich dem β-Wert eines seiner Vorgänger ist, wird nicht weiter expandiert und erhält den c-Wert $c(s') = \alpha(s')$.
Ist schließlich $c(s)$ ermittelt, so wählt MAX den Zug $s \rightarrow s'$, wobei s' der erste Nachfolger von s mit $c(s) = c(s')$ ist.

Beispiel 6.55 $\alpha - \beta - Verfahren$ für Tic-Tac-Toe mit Suchtiefe 2.



Schrittweise Berechnung der α, β, c-Werte:
$\alpha(s_0) = -\infty$
$\beta(s_1) = \infty$
$c(s_2) = 1,\ \beta(s_1) = 1$
$c(s_3) = 2,\ c(s_1) = 1,\ \alpha(s_0) = 1$
$\beta(s_4) = \infty$
$c(s_5) = -1,\ \beta(s_4) = -1 \Rightarrow$ keine weiteren Nachfolger von s_4, $c(s_4) = -1$
$\beta(s_6) = \infty$
$c(s_7) = 1, \beta(s_6) = 1 \Rightarrow$ keine weiteren Nachfolger von s_6, $c(s_6) = 1,\ c(s_0) = 1 \Rightarrow s_0 \rightarrow s_1$.

Kapitel 7

Die Behandlung unsicheren Wissens

Unsicherheiten, Unschärfen und Vagheiten treten in vielen Ausprägungen in fast allen realen Anwendungen auf. Einzelne Ansätze zu ihrer Behandlung berücksichtigen immer nur bestimmte Aspekte, und es ist stets wichtig festzustellen, welcher Mechanismus zu einer konkreten Problematik überhaupt paßt. Schon seit längerem bekannt sind wahrscheinlichkeitstheoretische Verfahren und die Verwendung von Sicherheitsfaktoren in Produktionsregelsystemen. Hier sollen nun einige weitere Ansätze behandelt werden (vgl. [Ric89], §14).

7.1 Die Methode der approximierten Prädikate

Betrachtet werden einstellige Prädikate $P(x)$ auf einem Universum U. Dabei wird das Prädikat mit der Menge $P = \{x \in U \mid P(x) \text{ ist wahr}\}$ identifiziert.

Definition 7.1 Eine *Ununterscheidbarkeitsrelation* $\approx$ ist eine reflexive und symmetrische Relation auf U. Für jede Teilmenge $P \subseteq U$ sei

$$P_u = \{x \in U \mid \text{für alle } \; y \in U \text{mit} x \approx y \; \text{gilt} y \in P\}$$

die *untere Approximation von* P und

$$P_o = \{x \in U \mid \text{es gibt ein } \; y \in P \text{ mit} x \approx y\}$$

die *obere Approximation von* P. Die *Unsicherheit* von P sei dann $\Delta P = P_o \setminus P_u$

Beispiel 7.2 i) Für Objekte $u_1, \ldots, u_k$ und Attribute $m_1, \ldots, m_n$ sei eine $k \times n$-Matrix $(w_{i,\nu})$ gegeben, wobei der Wert $w_{i,\nu}$ angibt, daß für das Objekt u_i das Attribut m_ν gerade den Wert $w_{i,\nu}$ annimmt. Auf der Menge U dieser Objekte wird dann durch

$$u_i \approx u_j \Leftrightarrow w_{i,\nu} = w_{j,\nu} \text{ für} \nu = 1, \ldots, n$$

eine Ununterscheidbarkeitsrelation $\approx$ definiert, die sogar transitiv, also eine Äquivalenzrelation ist.

ii) In einem Kriminalfall seien für die folgenden Verdachtspersonen die Merkmale "Haarfarbe" und "Blutgruppe" bekannt:

Verdachtspersonen	Haarfarbe	Blutgruppe
Max	blond	A
Ernst	blond	AB
Karl	braun	B
Fritz	blond	A
Heinz	braun	0
Emil	blond	A
Ludwig	braun	B
Kurt	braun	0

Die Äquivalenzrelation $\approx$ besteht dann aus den Klassen $\{Max, Fritz, Emil\}$, $\{Ernst\}$, $\{Karl, Ludwig\}$, $\{Heinz, Kurt\}$. Besteht die Menge P der akut Verdächtigen" aus Max,$Karl$ und $Ludwig$, dann sind ihre Approximationen

$$P_u = \{Karl, Ludwig\} \quad \text{und} \quad P_o = \{Max, Fritz, Emil, Karl, Ludwig\}.$$

Unsicherheit besteht also über die Personengruppe $\Delta = P_o \setminus P_u$.

Beim folgenden Satz beachte man, daß die logischen Junktoren $\neg, \vee, \wedge$ für Prädikate genau den mengentheoretischen Operationen Komplement, Vereinigung und Durchschnitt entsprechen.

Satz 7.3 Für beliebige Mengen P, Q, P_i von U gelten folgende Aussagen:

1) $P_u \subseteq P \subseteq P_o$,
2) $(P_u)_u \subseteq P_u$, $P_o \subseteq (P_o)_o$,
3) $P \subseteq Q \Rightarrow P_u \subseteq Q_u, P_o \subseteq Q_o$,
4) $\overline{P_u} = (\overline{P})_o, \overline{P_o} = (\overline{P})_u$,
5) $(\bigcap P_i)_u = \bigcap (P_i)_u$,
6) $(\bigcup P_i)_o = \bigcup (P_i)_o$,
7) $P_u \cup Q_u \subseteq (P \cup Q)_u$,
8) $(P \cap Q)_o \subseteq P_o \cap Q_o$.

In 2) gilt genau dann immer schon Gleichheit, wenn $\approx$ transitiv ist.
Beweis: Übungsaufgabe.

Bemerkung 7.4

i) Da die Implikation $P \to Q$ gleichwertig zu $Q \vee \neg P$ ist, kann man die Unsicherheiten für solche "Regeln" aus denen von Prämisssen und Konklusion nach Satz 7.3 wie folgt abschätzen:

$$P_o \to Q_u \subseteq (P \to Q)_u \subseteq (P \to Q)_o = P_u \to Q_o.$$

ii) In Anwendungen ist $\approx$ oft nicht transitiv. Daher gilt in 2) i.a. $(P_u)_u \subset P_u$ und $P_o \subset (P_o)_o$. Unsicherheiten könen bei längeren Iterationen daher stark anwachsen und daher sollten längere Schlußketten vermieden werden.

7.2 Evidenztheorie von Dempster und Shafer

Es sei U stets eine nichtleere endliche Menge, das *Universum*, und $I\!I = [0, 1]$ das Einheitsintervall.

Definition 7.5 Ein *Wahrscheinlichkeitsmaß* auf U ist eine Funktion $p : \mathbf{P}(u) \to I\!I$ mit

1) $p(\emptyset) = 0$,
2) $p(U) = 1$,
3) $p(A \cup B) = p(A) + p(B)$ für $A, B \in \mathbf{P}(U)$ mit $A \cap B = \emptyset$.

Bemerkung 7.6 Durch vollständige Induktion folgt aus 3) sofort $p(A_1 \cup \ldots \cup A_n) = \sum_{\nu=1}^{n} p(A_\nu)$ für endlich viele paarweise disjunkte $A_i \in \mathbf{P}(U)$. Ein Wahrscheinlichkeitsmaß ist daher eindeutig durch seine Werte $p(\{x\}) = p_x \in I\!I$ mit $\sum_{x \in U} p_x = 1$ bestimmt.

Definition 7.7

i) Eine *Massenverteilung* auf U ist eine Funktion $m : \mathbf{P}(U) \to I\!I$ mit

   4) $m(\emptyset) = 0$,
   5) $\sum_{A \subseteq U} m(A) = 1$.

   Die Mengen $A \subseteq U$ mit $m(A) > 0$ heißen *fokal.* Die Massenverteilung m heißt *gleichlaufend*, wenn ihre fokalen Mengen eine Kette $A_1 \subseteq A_2 \subseteq \ldots \subseteq A_n$ bilden.

ii) Die zu einer Massenverteilung m gehörende *B-Funktion* (von "believe") $Bel_m : \mathbf{P}(U) \to I\!I$ ist gegeben durch $Bel_m(B) = \sum_{A \subseteq B} m(A)$, die zu m gehörende *P-Funktion* (von plausibel") $Pl_m : \mathbf{P}(U) \to I\!I$ durch $Pl_m(B) = \sum_{A \cap B \neq \emptyset} m(A)$. Ist m gleichlaufend, so heißt auch Bel_m gleichlaufend.

Folgerung 7.1 Eine B-Funktion Bel_m ist genau dann ein Wahrscheinlichkeitsmaß auf U, wenn $m(\{x\}) = p_x \in I\!I$ und $m(A) = 0$ für $\mid A \mid \neq 1$ gilt.

Bemerkung 7.8

i) Es gibt stets die *nichtssagende Massenverteilung* m_0 mit $m_0(U) = 1$ und $m_0(A) = 0$ sonst. Sie stellt die leere Information dar und besitzt die B-Funkion $Bel_{m_o} = m_0$.

ii) Die fokalen Teilmengen sind diejenigen, für die eine gewisse Information vorhanden ist (eine Evidenz besteht), daß ein bestimmtes Element in ihnen zu finden ist. Dies kann etwa die Lösung eines Problemes sein, wenn U sämtliche alternative mögliche Lösungen beschreibt.

iii) Der Wert von $m(U)$ drückt aus, welchen Anteil an Evidenz man noch auf ganz U schieben muß, da man keine genauere Information besitzt. Je mehr Information man hat, desto kleiner wird $m(U)$ sein.

iv) Existiert nur ein fokales Element von m, also etwa $m(A) = 1$, dann gilt $Bel_m(B) = \begin{cases} 1 & A \subseteq B, \\ 0 & \text{sonst.} \end{cases}$

und $Pl_m(B) = \begin{cases} 1 & A \cap B \neq 0, \\ 0 & \text{sonst.} \end{cases}$

Definition 7.9 Ein *Möglichkeitsmaß* auf U ist eine Funktion $Poss : \mathbf{P}(U) \to I\!I$ mit

6) $Poss(\emptyset) = 0$,

7) $Poss(U) = 1$,

8) $Poss(A \cup B) = max\{Poss(A), Poss(B)\}$ für alle $A, B \in \mathbf{P}(U)$.

Die zugehörigen *Konturfunktion* ist $\mu_{Poss} : U \to I\!I$ mit $\mu_{Poss}(x) = Poss(\{x\})$, das zugehörige *Notwendigkeitsmaß* ist $Nec : \mathbf{P}(U) \to I\!I$ mit $Nec(A) = 1 - Poss(\overline{A})$ für alle $A \in \mathbf{P}(U)$.

Folgerung 7.2

i) Für alle $A \in \mathbf{P}(U)$ gelten $Poss(A) = max\{\mu_{Poss}(x) \mid x \in A\}$ und $Nec(A) = min\{1 - \mu_{Poss}(x) \mid x \notin A\}$.

ii) Für alle $A, B \in \mathbf{P}(U)$ hat man

 9) $A \subseteq B \Rightarrow Poss(A) \leq Poss(B)$ und $Nec(A) \leq Nec(B)$,
 10) $Nec(A) \leq Poss(A)$,
 11) $Nec(A \cap B) = min(Nec(A), Nec(B))$,
 12) $Nec(A) > 0 \Rightarrow Poss(A) = 1$,
 13) $Poss(A) < 1 \Rightarrow Nec(A) = 0$,
 14) $min(Nec(A), Nec(\overline{A})) = 0$,
 15) $max(Poss(A), Poss(\overline{A})) = 1$.

Beweis: Übungsaufgabe

Definition 7.10 Seien m_1 und m_2 Massenverteilungen auf U. Die Größe $K = \sum_{A \cap B = \emptyset} m_1(A)m_2(B)$ heißt *Konflikt* zwischen m_1 und m_2. Für $K = 0$ heißen m_1 und m_2 *konfliktfrei*, für $K = 1$ *unvereinbar*. Das Gewicht des Konfliktes zwischen m_1 und m_2 ist $-log(1 - K)$.

Bemerkung 7.11 Offensichtlich ist stets $0 \leq K \leq 1$. Sind m_1 und m_2 unvereinbar, so bedeutet dies, daß es keine Alternative gibt, für die sowohl m_1 als auch m_2 eine positive Evidenz angeben. In allen anderen Fällen kann man die durch m_1 und m_2 ausgedrückten Informationen wie folgt "akkumulieren":

Definition 7.12 Seien m_1 und m_2 Massenverteilungen auf U mit $K \neq 1$. Dann wird die *direkte Summe* $m_1 \oplus m_2 : \mathbf{P}(U) \to I\!I$ (nach Dempsters Regel) definiert durch $m_1 \oplus m_2(\emptyset) = 0$ und

$$m_1 \oplus m_2(C) = \frac{1}{1-K} \sum_{A \cap B = C} m_1(A)m_2(B) \text{ für} C \neq \emptyset.$$

Satz 7.13 Sind m_1 und m_2 Massenverteilungen auf U mit $K \neq 1$, dann auch $m_1 \oplus m_2$. Weiterhin gilt $m_1 \oplus m_2(U) \leq m_i(U)$ für $i = 1, 2$.
Beweis: Übungsaufabe

Bemerkung 7.14 $m_1 \oplus m_2$ "verschiebt" im Vergleich zu m_1 und m_2 Evidenz auf kleinere Mengen. Durch Akkumulation von Informationen wird so die Alternative, die eine Problemlösung darstellt, immer weiter eingekreist.

Beispiel 7.15 Auf $U = \{x_1, x_2, x_3\}$ sei die Massenverteilung m_1 durch $m_1(\{x_1, x_2\}) = \frac{1}{3}$ und $m_1(\{x_1, x_3\}) = \frac{2}{3}$ gegeben. Mit $m_2 = m_1$ gilt $K = 0$ und $m_1 \oplus m_1(\{x_1\}) = \frac{4}{9}$, $m_1 \oplus m_1(\{x_1, x_2\}) = \frac{1}{9}$, sowie $m_1 \oplus m_1(\{x_1, x_3\}) = \frac{4}{9}$. Dies zeigt, daß die Operation $\oplus$ (zwar stets kommutativ und assoziativ!), aber auch im konfliktfreien Fall nicht idempotent ist.

Sind m_1 und m_2 Massenverteilungen auf U_1 bzw. U_2, dann kann man jede von ihnen zu einer Massenverteilung auf $U_1 \times U_2$ fortsetzen. Dazu bezeichne π_i jeweils die Projektion von $U_1 \times U_2$ auf die i-te Komponente. Durch $m_1'(A) = m_1(\pi_1(A))$ bzw. $m_2'(B) = m_2(\pi_1(B))$ werden nun Funktionen $m_i' : \mathbf{P}(U_1 \times U_2) \to I\!R_+$ definiert, die $\sum_{A \subseteq U_1 \times U_2} m_1'(A) = c_1 > 0$ und $\sum_{B \subseteq U_1 \times U_2} m_2'(B) = c_2 > 0$ erfüllen. Dann sind aber $\widetilde{m_1} = \frac{1}{c_1} m_1'$ und $\widetilde{m_2} = \frac{1}{c_2} m_2'$ Massenverteilungen auf $U_1 \times U_2$, die *natürliche Fortsetzungen* von m_1 bzw. m_2 genannt werden.

7.3 Fuzzy-Logik

Zur Betrachtung von Wahrheitswerten aus dem Intervall $I\!I = [0, 1]$, anstelle der klassischen Wahrheitswerte aus $I\!B = \{0, 1\}$, ist es notwendig, die logischen Junktoren $\neg, \vee, \wedge$ entsprechend zu verallgemeinern.

Definition 7.16 Eine *Negation* N ist eine Funktion $N : I\!I \to I\!I$ mit

1) $N(0) = 1$,
2) $N(1) = 0$,
3) $N(a) \leq N(b)$ für alle $a, b \in I\!I$ mit $b \leq a$, (*Antimonotonie*).

Beispiel 7.17 Die am häufigsten verwendete Negation ist gegeben durch $N(a) = 1 - a$ für alle $a \in I\!I$.

Definition 7.18

i) Eine *T-Norm* (triangular norm") K ist eine Funktion $K : I\!I^2 \to I\!I$ mit
   1) $K(a, b) = K(b, a)$, (Kommutativität)
   2) $K(a, K(b, c)) = K(K(a, b), c)$, (Assoziativität)
   3) $K(a, b) \leq K(c, d)$, falls $a \leq c$ und $b \leq d$, (Monotonie)
   4) $K(a, 1) = a$,
   5) $K(0, 0) = 0$,

   für alle $a, b, c, d \in I\!I$.

ii) Eine *T-Conorm* D ist ein Funktion $D : I\!I^2 \to I\!I$ mit
   1') $D(a, b) = D(b, a)$,
   2') $D(a, D(b, c)) = D(D(a, b), c)$,
   3') $D(a, b) \leq D(c, d)$, falls $a \leq c$ und $b \leq d$,
   4') $D(0, b) = b$,
   5') $D(1, 1) = 1$,

   für alle $a, b, c, d \in I\!I$.

Bemerkung 7.19

i) Die Einschränkungen jeder Negation, T-Norm bzw. T-Conorm auf $\mathbb{B}$ ergibt jeweils die klassischen logischen Junktoren.

ii) Verwendet man die Negation N aus Beispiel 7.19, so nennt man ein T-Norm K und eine T-Conorm D *N-dual* zueinander, wenn sie die De Morganschen Regeln für alle $a, b \in \mathbb{I}$ erfüllen:

$$K(a,b) = N(D(N(a), N(b))) \text{ und } D(a,b) = N(K(N(a), N(b)))$$

Diese sind offensichtlich gleichwertig.

Beispiel 7.20

i) Die *triviale T-Norm* K_0 bzw. die *triviale T-Conorm* D_0 sind gegeben durch

$$K_0(a,b) = \begin{cases} a & \text{für} \quad b = 1, \\ b & \text{für} \quad a = 1, \\ 0 & \text{sonst,} \end{cases} \text{ bzw. } D_0(a,b) = \begin{cases} a & \text{für} \quad b = 0, \\ b & \text{für} \quad a = 0, \\ 1 & \text{sonst.} \end{cases}$$

ii) Oft benutzte Paare von T-Normen und T-Conormen sind die folgenden (K_1, D_1),(K_2, D_2),(K_3, D_3) mit

$$\begin{array}{ll} K_1(a,b) = max(0, a+b-1), & D_1(a,b) = min(1, a+b), \\ K_2(a,b) = ab, & D_2(a,b) = a+b-ab, \\ K_3(a,b) = min(a,b), & D_3(a,b) = max(a,b). \end{array}$$

Folgerung 7.3

i) Für $i = 0, 1, 2, 3$ sind K_i und D_i N-dual zueinander.

ii) Für jede T-Norm K und jede T-Conorm D gilt $K_0 \leq K \leq K_3$ und $D_3 \leq D_2 \leq D_1$.

iii) Außerdem hat man $K_1 \leq K_2$ und $D_2 \leq D_1$

Beweis: Übungsaufgabe

Bemerkung 7.21 Vergleicht man K_1 bis K_3 miteinander, so sieht man, daß in K_3 versucht wird, ein Argument durch das andere zu subsumieren ("best case"), in K_2 werden beide Argumente als unabhängig voneinander betrachtet (klassische Wahrscheinlichkeiten), während in K_1 sich beide Argumente so weit wie möglich ausschließen ("worst case"). Dies kann eine Entscheidungshilfe für eines der drei Paare aus Bsp. 7.22 ii) bei konkreten Anwendungen sein.

Definition 7.22

i) Eine *Fuzzy-Teilmenge* f einer Menge U wird durch ihre *charakteristische Funktion* $\mu_f : U \to \mathbb{I} = [0,1]$ gegeben. Sie heißt *normalisiert*, wenn es ein $x \in U$ mit $\mu_f(x) = 1$ gibt. Eine weitere Fuzzy-Teilmenge f' von U heißt *Teilmenge* von f, wenn $\mu_{f'} \leq \mu_f$, also $\mu_{f'}(x) \leq \mu_f(x)$ für alle $x \in U$ gilt.

ii) Die mengentheoretischen Operationen sind für die Fuzzy-Teilmengen erklärt durch:

1) $\mu_{f \cap g}(x) := (\mu_f \wedge \mu_g)(x) := min\{\mu_f(x), \mu_g(x)\}$,
2) $\mu_{f \cup g}(x) := (\mu_f \vee \mu_g)(x) := max\{\mu_f(x), \mu_g(x)\}$,
3) $\mu_{\overline{f}}(x) := \overline{\mu_f}(x) := 1 - \mu_f(x)$.

Bemerkung 7.23

i) Jede Teilmenge $f \subseteq U$ im üblichen Sinne ist eine Fuzzy-Teilmenge, deren charakteristische Funktion μ_f nur die Werte aus $\mathbb{B}$ annimmt. Hierbei ist als einzige die leere Menge $\emptyset$ mit $\mu_\emptyset = 0$ für alle $x \in U$ nicht normalisiert. Für derartige Teilmengen gehen die Operationen unter ii) in die üblichen mengentheoretischen Operationen über.

ii) Man kann in 1) - 3) auch andere T-Normen, T-Conormen oder Negationen verwenden, wenn es für eine Anwendung zweckmäßig sein sollte.

iii) Die Fuzzy-Teilmengen einer Menge U bilden bzgl. der Operationen $\vee$ und $\wedge$ einen beschränkten distributiven Verband. Dieser ist jedoch keine Boolesche Algebra, da z.B. für die Fuzzy-Teilmenge f mit $\mu_f(x) = 0.5$ für alle $x \in U$ gilt $\overline{\mu_f} = \mu_f$ und $\mu_f \wedge \overline{\mu_f} \neq \mu_\emptyset$.

iv) Fuzzy-Teilmengen auf U können zur Beschreibung "unscharfer" einstelliger Prädikate $P(x)$ auf U verwendet werden.

v) Für manche Anwendungen ist es ausreichend, nur endlich viele Werte $\{0, w_1, \ldots, w_{n-2}, 1\} \subseteq \mathit{II}$ für die charakteristischen Funktionen μ_f von Fuzzy-Teilmengen zur Verfügung zu stellen. Meistens wählt man $w_i = \frac{i}{n-1}$, $i = 1, \ldots, n-2$ für einen geeigneten Wert von n. Man spricht dann von *n-wertiger Logik*. Diese Werte haben dann gewisse inhaltliche Bedeutungen etwa: unmöglich, fast unmöglich, möglich, ziemlich sicher, absolut sicher.

vi) Ist $U = U_1 \times U_2$, dann wird eine Fuzzy-Teilmenge r von U auch als *Fuzzy-Relation* bezeichnet. Entsprechend verfährt man bei mehreren Faktoren U_i. Die Projektionen $\pi_i(r)$ auf U_i bezeichnet man auch als *marginale Fuzzy-Teilmengen* zu r, also etwa

$$\mu_{\pi_i(r)}(x_i) := sup\{\mu_r(x_1, \ldots, x_n) \mid (x_1, \ldots, x_n) \in U\}.$$

Entsprechend kann man jede Fuzzy-Teilmenge f_i von U_i zu einer Fuzzy-Teilmenge $\pi_i^{-1}(f_i)$ erweitern gemäß

$$\mu_{\pi_i^{-1}(f_i)}(x_1, \ldots, x_n) := \mu_{f_i}(x_i).$$

vii) Ist $\varphi : U \longrightarrow U'$ eine Abbildung und f Fuzzy-Teilmenge von U, dann definiert man die Fuzzy-Teilmenge $\varphi(f)$ von U' gemäß

$$\mu_{\varphi(f)}(x') = sup\{\mu_f(x) \mid x \in U \text{mit } \varphi(x) = x'\}$$

Definition 7.24 Ist f Fuzzy-Teilmenge von U, dann heißt $f_\alpha = \{x \in U \mid \mu_f(x) \geq \alpha\}$ für jedes $\alpha \in \mathit{II}$ der *α-Schnitt* von f.

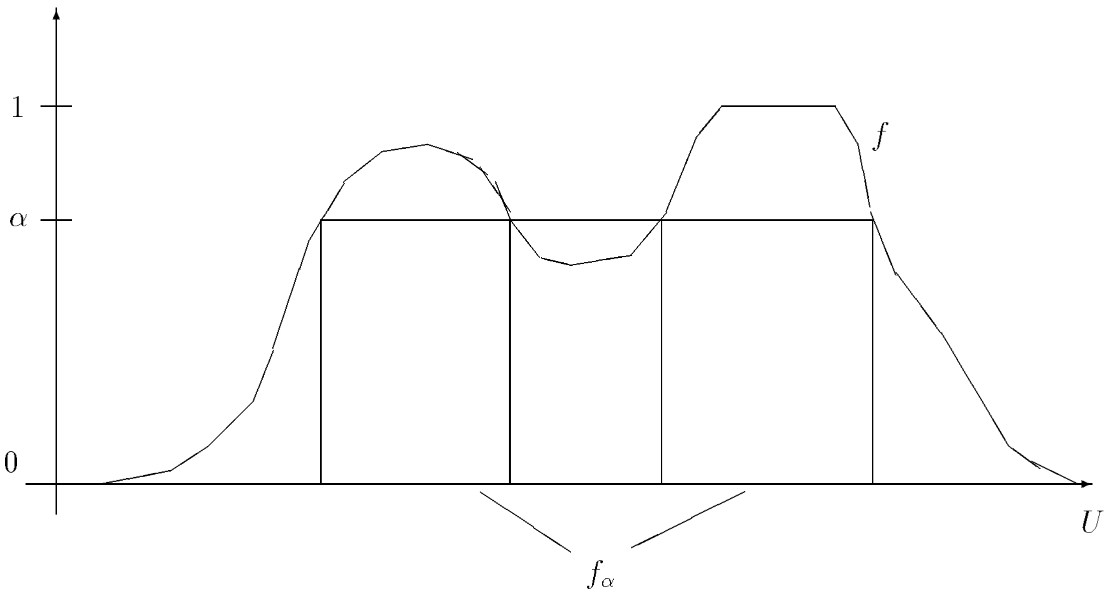
Folgerung 7.4 Die α-Schnitte einer Fuzzy-Teilmenge f von U erfüllen:

i) $\alpha > \alpha' \Rightarrow f_\alpha \subseteq f_{\alpha'}$,

ii) $f_0 = U$,

iii) $f_\alpha = \bigcap_{\beta < \alpha} f_\beta$.

Ist umgekehrt eine Familie $(f_\alpha)_{\alpha \in \mathit{II}}$ von Teilmengen von U gegeben, dann wird durch $\mu_f(x) = sup\{\alpha \in \mathit{II} \mid x \in f_\alpha\}$ eine Fuzzy-Teilmenge f von U beschrieben.



Definition 7.25 Ist f Fuzzy-Teilmenge von U, dann heißt $Poss_f : \mathbf{P}(U) \rightarrow \Pi$ gemäß $Poss_f(A) := sup\{\alpha \mid A \cap f_\alpha \neq \emptyset\} = sup\{\mu_f(x) \mid x \in A\}$ das *zu f gehörende Möglichkeitsmaß* und $Nec_f : \mathbf{P}(U) \rightarrow \Pi$ gemäß $Nec_f(A) = 1 - Poss_f(\overline{A})$ das *zu f gehörende Notwendigkeitsmaß.*

Die Anwendung von Fuzzy-Teilmengen in Produktionsregelsystemen kann etwa wie folgt aussehen:
Seien x_i Variablen, die Werte in den Mengen U_i annehmen können. Die Fakten in der Wissensbasis seien von der Form:

x_1 liegt in f', wobei f' eine Fuzzy-Teilmenge von U_1 ist.

Also etwa "Temperatur des Patienten ist sehr niedrig", wobei "sehr niedrig" eine Fuzzy-Teilmenge der Temperaturskala ist.
Die Regeln seien von der Form

WENN x_1 in f, DANN x_2 in g, wobei g Fuzzy-Teilmenge von U_2 ist.

Also etwa "WENN Temperatur niedrig ist, DANN ist Grippeinfektion eher unwahrscheinlich".
Jedem Faktum, das mit der Regel gematcht werden soll wird nun das Möglichkeitsmaß

$$Poss_{Fakt}(x_1, x_2) := \mu_{f'}(x_1)$$

zugeordnet.
Der Regel entspricht z.B. das Möglichkeitsmaß

$$Poss_{Regel}(x_1, x_2) := \begin{cases} 1, & \text{falls } \mu_f(x_1) \leq \mu_g(x_2) \\ \mu_g(x_2) & \text{sonst.} \end{cases}$$

Die Regelanwendung erfolgt nun nach dem "Prinzip der kleinsten Spezifität

$$Poss_{Produkt}(x_1, x_2) := min\{Poss_{Regel}(x_1, x_2), Poss_{Fakt}(x_1, x_2)\}.$$

Dem abgeleiteten neuen Faktum "x_2 in g' "entspricht schließlich

$$Poss_{Resultat}(x_2) := sup\{Poss_{Produkt}(x_1, x_2) \mid x_1 \in U_1\}.$$

Für die Möglichkeitsmaße $Poss_{Regel}$ und $Poss_{Produkt}$ sind auch andere Kombinationen von T-Normen und T-Conormen vorgeschlagen worden (vgl. [KSH91], 10.3).

Anhang A

Aufgabensammlung

A.1 Aufgaben

A.1.1 Wissen und seine Repräsentation

Aufgabe A.1
Der Kontext (G, M, κ) sei durch $M = \{m_1, m_2, m_3\}$ und $G = \{g_0 = \emptyset, g_1 = \{m_1\}, g_2 = \{m_2\}, g_3 = \{m_3\}, g_4 = \{m_1, m_3\}, g_5 = \{m_2, m_3\}\}$ sowie $g\kappa m \Leftrightarrow m \in g$ für $g \in G$ und $m \in M$ gegeben. Bestimmen Sie $B(G, M, \kappa)$.

A.1.2 Prädikatenlogik

Aufgabe A.2
Geben Sie alle Terme und Literale zum Beispiel 2.5 an und kennzeichnen Sie davon alle Grundterme und alle Grundformeln.

Aufgabe A.3
Drücken Sie die folgenden Sätze in Horn-Klauseln aus, wobei Sie Prädikate Ihrer Wahl benutzen dürfen.

1) Drachen, die im Zoo leben, sind glücklich.
2) Tiere, die freundlichen Menschen begegnen, sind glücklich.
3) Zoobesucher sind freundlich.
4) Tiere, die im Zoo leben, begegnen Zoobesuchern.

Welche beiden zusätzlichen Annahmen sind notwendig, um die Folgerung zu rechtfertigen, daß kein Drache in einem Zoo lebt ?

Aufgabe A.4
Ein Experte beschreibt sein Wissen über seinen Wissensbereich wie folgt:

Drei Prieks gibt es, nämlich den Bopriek, den Kipriek und den Blapriek, und jeder Priek gnaselt genau eine Lapümpfe. Folglich gibt es drei verschiedene Lapümpfen: Die Drauslapümpfe, die Glesslapümpfe und die Haplapümpfe. Jede Lapümpfe wiederum knaudert, und zwar knaudert die eine Lapümpfe an, die zweite ab und die dritte knaudert über. Wenn nun die überknaudernde Lapümpfe entweder vom Kipriek oder vom Bopriek gegnaselt wird, dann knaudert die Drauslapümpfe an. Nun wird die Haplapümpfe nicht vom Kipriek gegnaselt und sie überknaudert auch nicht. Außerdem gnaselt der Bopriek nicht die Drauslapümpfe, und die von ihm gegnaselte Lapümpfe knaudert nicht ab. Und wenn die Drauslapümpfe vom Kipriek gegnaselt wird, dann knaudert die Glesslapümpfe nicht über.

Stellen Sie dieses Wissen durch Horn-Klauseln dar, wenn möglich in PROLOG-Notation.

Aufgabe A.5
Die Sprache $\mathcal{L}$ bestehe aus den prädikatenlogischen Formeln, die iterativ gemäß (a) - (c) gebildet werden:

(a) Jedes nullstellige Prädikatssymbol liegt in $\mathcal{L}$.

(b) Ist F aus $\mathcal{L}$, dann auch $(\neg F)$.

(c) Sind F und G aus $\mathcal{L}$, dann auch $(F \rightarrow G)$.

Es ist also $\mathcal{L}$ eine Teilmenge der aussagenlogischen Formeln. Weiterhin seien die folgenden drei nullstelligen Ableitungsregeln (die auch **Axiomenschemata** genannt werden) gegeben.

(A1) $(A \rightarrow (B \rightarrow A))$,

(A2) $((A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$,

(A3) $((\neg(\neg A)) \rightarrow A)$,

sowie die zweistellige Ableitungsregel des **Modus ponens (MP)** gemäß

$$\frac{A \qquad A \rightarrow B}{B}$$

für beliebige $A, B, C \in \mathcal{L}$ definiert.

a) Zeigen Sie, daß es sich bei jeder gemäß (A1),(A2) oder (A3) gebildeten Formel aus $\mathcal{L}$ um eine Tautologie handelt. (Dies besagt gerade die Korrektheit der entsprechenden Ableitungsregel!)

b) Zeigen Sie, daß auch (MP) korrekt ist.

c) Es sei $\mathcal{R} = \{(A1), (A2), (A3), (MP)\}$. Zeigen Sie $\emptyset \vdash_{\mathcal{R}} (F \rightarrow F)$, $\{F\} \vdash_{\mathcal{R}} (G \rightarrow F)$ und $\{(F \rightarrow G), (G \rightarrow H)\} \vdash_{\mathcal{R}} (F \rightarrow H)$ für beliebige Formeln $F, G, H \in \mathcal{L}$.

Aufgabe A.6
Gegeben sind die folgenden Aussagen:

1. Wer bedrüpt ist, makst nicht.
2. Wer knaselt, hat eine Mause.
3. Jeder, der nicht deuken kann, makst.
4. Alle Drumser nehmen Pfuff.
5. Wer kein Pli ist, matupelt nicht.
6. Kein betuxter Klep kann deuken.
7. Wer Pfuff nimmt, ist bedrüpt.
8. Nur betuxte Kleps knaseln nicht.
9. Jeder Pli ist ein Drumser.

Diese Fakten sind durch Wahl geeigneter Prädikate als PROLOG-Regeln abzulegen, so daß nach Ablage des Faktums ,,matupelt(elf)”, wobei ,,elf” der Name einer extraterrestrischen Lebensform ist, entschieden werden kann, ob elf eine Mause hat oder nicht.

Aufgabe A.7
In Analogie zur vorgestellten Lösung der Aufgabe 4 ist ein PROLOG-Programm zu schreiben, das die folgende Denksportaufgabe löst.

,,Ich finde es äußerst verwirrend”, stöhnt der Innungsmeister in Kuhdorf, ,,unsere drei Handwerker, der Schuster, der Schneider und der Metzger heißen Schmidt, Schmied und Schmitt, und ich bringe immer durcheinander, wer was ist.”
,,Das ist sehr einfach”, beruhigt ihn sein Sekretär: ,,Herr Schmidt ist nicht der Schneider. Herr Schmitt ist nicht der Schuster. Herr Schmidt ist der Schuster. Und Herr Schmitt ist nicht der Schneider.”
Aber so einfach ist das eben nicht, denn nur eine der vier Aussagen des Sekretärs ist wahr. Wer ist was?

Aufgabe A.8
Bezüglich eines Roboters Robbie seien die beiden Prädikate $s(X, T)$ mit der Bedeutung ,,Robbie sieht das Objekt X zum Zeitpunkt T” und $g(X, T)$ mit der Bedeutung ,,Robbie ergreift das Objekt X zum Zeitpunkt T” definiert. Außerdem werde $T1 < T2$ als ,,der Zeitpunkt T1 liegt früher als der Zeitpunkt T2” interpretiert. Formulieren Sie die folgenden Aussagen damit in PL1.

1. Manchmal sieht Robbie nichts.
2. Jedes Objekt sieht Robbie irgendwann einmal.
3. Wenn er etwas sieht, ergreift Robbie es sofort.
4. Ehe er etwas ergreift, hat Robbie es gesehen.
5. Was er immer sieht, ergreift Robbie nie.
6. Robbie ergreift alles, was er nicht immer sieht.
7. Wenn Robbie einmal zwei Dinge zugleich gesehen hat, sieht er sie danach auch immer.
8. Wenn Robbie etwas gleichzeitig gesehen und ergriffen hat, macht er dies später nacheinander oder gar nicht.

A.1.3 Produktionsregelsysteme

Aufgabe A.9
Es seien das folgende Produktionsregelsystem (WB, R, S) mit der Endebedingung $Z = \{z(a)\}$ und der Wissenbasis $WB = \{v(a)\}$ gegeben. Die Produktionsregeln seien:

R1: WENN $v(X)$ DANN ergaenze_faktum($u(X)$).
R2: WENN $u(X)$ DANN ergaenze_faktum($s(X)$).
R3: WENN $v(X)$ DANN ergaenze_faktum($r(X)$).
R4: WENN $r(X)$ DANN ergaenze_faktum($w(X)$).
R5: WENN $r(X)$ DANN ergaenze_faktum($z(X)$).

und die Kontrollstrategie S:

Eine Regel ist gegenüber einer anderen zu bevorzugen, wenn sie noch nicht angewandt wurde, die andere dagegen schon. Sollten danach noch R_i und R_j mit $i < j$ in der Konfliktmenge sein, so ist R_i anzuwenden.

a) Geben Sie die Änderungen der dynamischen Wissensbasis bei Vorwärtsverkettung an.

b) Geben Sie die Änderungen des Zieles bei Rückwärtsverkettung an.

Aufgabe A.10
In einem Produktionsregelsystem werden die Beziehungen zwischen den Begriffen a, b, c, d, e, f, g, h durch die Formeln:

$$is\_a(a, c), is\_a(c, f), is\_a(c, g), is\_a(b, d), is\_a(b, c), is\_a(d, g), is\_a(e, h)$$

dargestellt. Die einzige Regel sei:

R1: $is\_a(X, Y), is\_a(Y, Z) \rightarrow is\_a(X, Z)$

Das Ziel sei die Bestimmung der gemeinsamen Oberbegriffe von a und b.

a) Geben Sie die Änderung der dynamischen Wissenbasis bei Vorwärtsverkettung an.

b) Beschreiben Sie dasselbe Wissen durch ein semantisches Netz.

c) Beschreiben Sie die Schritte (durch Angabe der Liste der jeweils aktivierten Knoten), die ein System bei diesem Repräsentationsformat zum Erreichen desselben Zieles durchführen muß.

A.1.4 Frames

Aufgabe A.11
Gegeben seien die Namen $F1, \ldots, F5$ und $S1, S2, S3$ für Klassenkonzepte sowie die Mengen:
$EE1 = \{x \mid is\_a(x, S1) \vee instanz\_von(x, S1)\}$ und
$EE2 = \{x \mid is\_a(x, S2) \vee instanz\_von(x, S2)\}$.

Damit werden die Frames $F1, \ldots, F5$ beschrieben:

1) $F1$ habe den einen einwertigen Slot $S1$, dessen erlaubte Einträge aus $EE1$ sind.

2) $F2$ habe den einen Slot $S2$, dessen erlaubte Einträge ebenfalls aus $EE1$ sind.

3) $F3$ habe den einen einwertgien Slot $S2$, dessen erlaubte Einträge aus $EE2$ sind. Der aktuelle Eintrag sei $SE = \{S1\}$.

4) $F4$ habe den einen Slot $S2$, dessen erlaubte Einträge aus $EE2$ sind. Der aktuelle Eintrag sei $SE = \{S1\}$.

5) $F5$ habe den einen Slot $S1$, dessen erlaubte Einträge aus $EE1$ sind.

Weiterhin bestehen die Relationen $is\_a(S1, S2)$ und $is\_a(S3, S2)$.

a) Beschreiben Sie den Typ von $S1$ und $S2$ in $F1, \ldots, F5$ (gem. Definition 5.11) genauer.

b) Welche is_a-Relationen bestehen zwischen den Klassen $F1, \ldots, F5$?

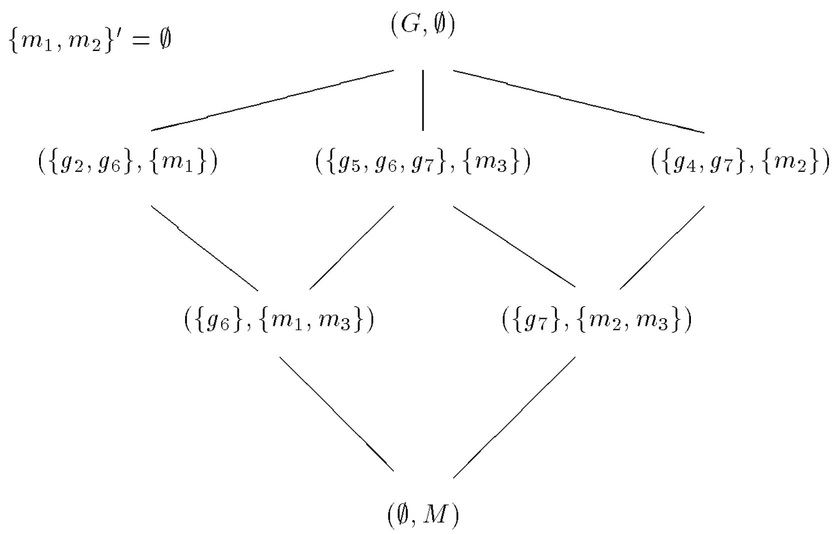
c) Beschreiben Sie die Beziehungen zwischen den Klassen durch ein semantisches Netz.

A.2 Lösungen

A.2.1 Wissen und seine Repräsentation

Lösung A.1

κ	m_1	m_2	m_3
$g_1 = \emptyset$			
$g_2 = \{m_1\}$	x		
$g_4 = \{m_2\}$		x	
$g_5 = \{m_3\}$			x
$g_6 = \{m_1, m_3\}$	x		x
$g_7 = \{m_2, m_3\}$		x	x



A.2.2 Prädikatenlogik

Lösung A.2
$\mathcal{F} = \{$c,f,g$\}$, $\mathcal{P} = \{$q,p$\}$, $\mathcal{V} = \{$x,y$\}$
Terme (geordnet nach der Anzahl der Symbole):

1: x,y,$\underline{\text{c}}$

2: f(x),f(y),$\underline{\text{f(c)}}$

3: f(f(x)),f(f(y)),$\underline{\text{f(f(c))}}$,g(x,x),g(x,c),...,$\underline{\text{g(c,c)}}$,...

4: f(f(f(x))),...,$\underline{\text{f(f(f(c)))}}$,f(g(x,x)),...,$\underline{\text{f(g(c,c))}}$,g(f(x),x),...,$\underline{\text{g(c,f(c))}}$,...

$\vdots$

Literale:

$\underline{q}, \neg q$
$p(x), \neg p(x), p(y), \neg p(y), \underline{p(c)}, \neg p(c)$
$p(f(x)), \ldots, \underline{p(f(c))}, \ldots$

Diese Mengen spielen eine wichtige Rolle bei der Definition spezieller Interpretationen, der sog. *Herbrand-Interpretationen* bzw. *Herbrand-Modelle*
Herbrand-Universum U ist die Menge aller Grundterme.
Herbrand-Basis B ist die Menge aller Grundformeln.
Interpretation: Jede Funktion wird durch "sich selbst" interpretiert, also $I(f)(t_1, \ldots, t_n) = f(t_1, \ldots, t_n) \in U$.
Die zu den Prädikatssymbolen gehörenden Relationen auf U werden durch Auswahl einer Menge $A \subseteq B$ bestimmt:
$p(t_1, \ldots, t_n)$ gilt genau dann, wenn $p(t_1, \ldots, t_n) \in A$.

Lösung A.3

1) ist_drache(X) ∧ lebt_im_Zoo(X) → ¬ist_glücklich(X).

2) ist_tier(X) ∧ ist_mensch(Y) ∧ ist_freundlich(Y) ∧ begegnet(X,Y) → ist_glücklich(X).

3) besucht_zoo(X) → ist_freundlich(X).

Zwei nicht ganz korrekte Formalisierungen von 4) sind:

4′) ist_tier(X) ∧ lebt_im_zoo(X) ∧ besucht_zoo(Y) → begegnet(X,Y).
"Tiere, die im Zoo leben, begegnen den (= allen) Zoobesuchern."

4″) ist_tier(X) ∧ lebt_im_zoo(X) ∧ begegnet(X,Y) → besucht_zoo(Y).
"Tiere, die im Zoo leben, begegnen nur Zoobesuchern."

Besser und korrekter geht es mittels Skolemisierung:

4a) ist_tier(X) ∧ lebt_im_zoo(X) → begegnet(X,zoobesucher(X)).

4b) ist_tier(X) ∧ lebt_im_zoo(X) → besucht_zoo(zoobesucher(X)).

5) ist_drache(X) → ist_tier(X).

6) besucht_zoo(X) → ist_mensch(X).

Theorem: $\forall X$: ¬(ist_drache(X) ∧ lebt_im_zoo(X))
⇒ 7) ($\exists X$) ist_drache(X) ∧ lebt_im_zoo(X)
oder 7.1) ist_drache(X)
7.2) lebt_im_zoo(X)

Somit folgt also:

7),1)	⇒ 8) ¬ ist_glücklich(X).
5),7.1)	⇒ 9) ist_tier(X).
9),4.1)	⇒ 10) lebt_im_zoo(X) → begegnet(X,zoobesucher(X)).
10),7.2)	⇒ 11) begegnet(X,zoobesucher(X)).
9),4.2)	⇒ 12) lebt_im_zoo(X) → besucht_zoo(zoobesucher(X)).
12),7.2)	⇒ 13) besucht_zoo(zoobesucher(X)).
13),6)	⇒ 14) ist_mensch(zoobesucher(X)).
13),3)	⇒ 15) ist_freundlich(zoobesucher(X)).
2),11)	⇒ 16) ist_tier(X) ∧ ist_mensch(zoobesucher(X)) ∧ ist_freundlich(X)) → ist_glücklich(X).
16),14)	⇒ 17) ist_tier(X) ∧ ist_freundlich(zoobesucher(X)) → ist_glücklich(X).
17),15)	⇒ 18) ist_tier(X) → ist_glücklich(X).
18),9)	⇒ 19) ist_glücklich(X).
8),19)	⇒ □

Lösung A.4

```
priek(bopriek) ...
lapümpfe(draus) ...
drei_prieks(X,Y,Z) :-
        priek(X),
        priek(Y), X /= Y,
        priek(Z), X /= Z, Y /= Z.
drei_lapümpfe(X,Y,Z) :- ...
drei_knauderarten(X,Y,Z) :- ...
loesung(A,B,C,X,Y,Z,U,V,W) :-
        drei_lapümpfe(A,B,C),
        drei_prieks(X,Y,Z),
        drei_knauderarten(U,V,W),
        retractall(wahr(_,_,_)),
        asserta(wahr(A,X,U)),
        asserta(wahr(B,Y,V)),
        asserta(wahr(C,Z,W)),
        konsistent.
konsistent :- bed1,bed2,bed3,bed4.
bed4 :- not wahr(draus,kipriek,_);
        not wahr(gless,_,ueber).
bed3 :- not wahr(draus,bopriek,_);
        not wahr(_,bopriek,ab).
bed2 :- not wahr(hap,kipriek,_);
        not wahr(hap,_,ueber).
bed1 :- not wahr(draus,_,an);
        (not wahr(_,bopriek,ueber),not wahr(_,kipriek,ueber)).
```

Lösung A.5

$\mathcal{L}$ im Prinzip alle aussagenlogischen Formeln:
$(A \vee B) :\Leftrightarrow ((\neg A) \rightarrow B)$
$(A \wedge B) :\Leftrightarrow (\neg(A \rightarrow (\neg B)))$
$(A \leftrightarrow B) :\Leftrightarrow ((A \rightarrow B) \wedge (B \rightarrow A)) \Leftrightarrow (\neg((A \rightarrow B) \rightarrow (\neg(B \rightarrow A)))$

a) Seien (I, U) eine Interpretation von $\mathcal{L}$ und A, B, C beliebige Formeln aus $\mathcal{L}$. Der Beweis erfolgt jeweils durch vollständige Fallunterscheidung über die Wahrheitswerte von A, B, C, also etwa auch durch Wahrheitstafeln.

   (A1) Man beachte: $A \rightarrow (B \rightarrow A)$ ist äquivalent zu $\neg A \vee (B \rightarrow A)$.
   Entweder ist A wahr bzgl. (I, U), dann gilt $B \rightarrow A$ und damit (A1), oder $\neg A$ ist wahr bzgl. (I, U), also ebenfalls (A1).
   (A2) und (A3) werden analog bewiesen.

b) Zu zeigen ist: $\{A, A \rightarrow B\} \models B$
   Sei (I, U) ein Modell für A und $\neg A \vee B$. Dann sind also A und $\neg A \vee B$ wahr bzgl. (I, U). Also ist $\neg A$ falsch bzgl. (I, U), folglich B wahr bzgl. (I, U). Also ist (I, U) auch ein Modell für B!

c)

i) $\emptyset \vdash_{\mathcal{R}} (F \to F)$
Seien $A := F, B := (F \to F), C := F$ aus $\mathcal{L}$

(A1): (1) $(F \to ((F \to F) \to F))$
(A2): (2) $((F \to ((F \to F) \to F)) \to ((F \to (F \to F)) \to (F \to F)))$
(MP): (3) $((F \to (F \to F)) \to (F \to F))$
(A1): (4) $((F \to (F \to F))$ mit $A := F, B := F$
(MP): (5) $(F \to F)$

ii) $\{F\} \vdash_{\mathcal{R}} (G \to F)$
(1) (F)
(A1): (2) $(F \to (G \to F))$
(MP): (3) $(G \to F)$

iii) $\{(F \to G), (G \to H)\} \vdash_{\mathcal{R}} (F \to H)$
(1) $(F \to G)$
(2) $(G \to H)$
Nach ii) gilt
(3) $F \to (G \to H)$
(A2): (4) $((F \to (G \to H) \to ((F \to G) \to (G \to H))$
(MP): (5) $((F \to G) \to (F \to H))$
(MP): (6) $(F \to H)$

Lösung A.6
PROLOG-Regeln:
1. nicht_makst(X) :- bedruept(X).
2. hat_mause(X) :- knaselt(X).
3. kann_deuken(X) :- nicht_makst(X).
4. nimmt_pfuff(X) :- drumser(X).
5. ist_pli(X) :- matupelt(X).
6. ist_kein_betuxter_klep(X) :- kann_deuken(X).
7. bedruept(X) :- nimmt_pfuff(X).
8. knaselt(X) :- ist_kein_betuxter_klep(X).
9. drumser(X) :- ist_pli(X).

PROLOG-Fakt:
matupelt(elf).
?-hat_mause(elf) → Antwort: YES.

Lösung A.7
```
beruf(schuster).
beruf(schneider).
beruf(metzger).


drei_berufe(X,Y,Z) :-
        beruf(X),
        beruf(Y), X /= Y,
        beruf(Z), X /= Z, Y /= Z.

bed1 :- not wahr(schmidt,schneider).
bed2 :- not wahr(schmitt,schuster).
bed3 :- wahr(schmidt,schuster).
bed4 :- not wahr(schmitt,schneider).
```

```
konsistent :-
        (bed1,not bed2, not bed3, not bed4);
        (not bed1,bed2, not bed3, not bed4);
        (not bed1,not bed2, bed3, not bed4);
        (not bed1,not bed2, not bed3, bed4);

loesung(Schmidt,Schmied,Schmitt) :-
                        drei_berufe(Schmidt,Schmied,Schmitt),
                        retractall(wahr(_,_,_)),
                        assert(wahr(schmidt,Schmidt)),
                        assert(wahr(schmied,Schmied)),
                        assert(wahr(schmitt,Schmitt)),
                        konsistent.

loesung(_,_,_) :- retractall(wahr(_,_,_)).
retractall(X) :- retract(X), fail.
retractall(X).
```

Lösung A.8

1. $\exists T \neg \exists X : s(X,T)$ oder $\exists T \forall X : \neg s(X,T)$.
2. $\forall X \exists T : s(X,T)$.
3. $\forall X \forall T : s(X,T) \rightarrow g(X,T)$.
4. $\forall X \forall T : g(X,T) \rightarrow \exists T1 : (T1 < T) \wedge s(X,T1)$.
5. $\forall X (\forall T : s(X,T)) \rightarrow \neg \exists T1 : g(X,T1)$.
6. $\forall X (\neg \forall T : s(X,T)) \rightarrow \exists T1 : g(X,T1)$.
7. $\forall X \forall Y \forall T : (s(X,T) \wedge s(Y,T)) \rightarrow \forall T1 : ((T1 < T) \rightarrow (s(X,T1) \rightarrow s(Y,T1))$.
8. $\forall X \forall T : (s(X,T) \wedge g(X,T)) \rightarrow \forall T1 : ((T < T1) \rightarrow \neg(s(X,T1) \wedge g(X,T1)))$.

A.2.3 Produktionsregelsysteme

Lösung A.9

a) $\{v(a)\} \xrightarrow{R1} \{v(a), u(a)\} \xrightarrow{R2} \{v(a), u(a), s(a)\} \xrightarrow{R3} \{v(a), u(a), s(a), r(a)\} \xrightarrow{R4} \{v(a), u(a), s(a), r(a), w(a)\} \xrightarrow{R5} \{v(a), u(a), s(a), r(a), w(a), z(a)\}$

b) $\{z(a)\} \xrightarrow{R5} \{r(a)\} \xrightarrow{R3} \{v(a)\} = WB$

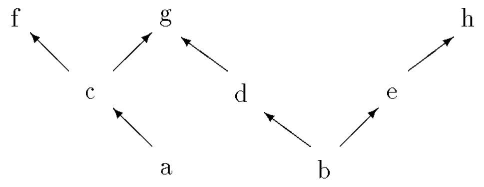
Lösung A.10

Realisierung der Transitivität der *is_a-Relation* in PROLOG:

```
is_a1(a,c).
⋮
is_a(X,Y) :- is_a1(X,Y).
is_a(X,Z) :- is_a1(X,Y),is_a(Y,Z).
```

a) WB = {is_a(a,c),is_a(c,f),is_a(c,g),is_a(b,d),is_a(b,e),is_a(d,g),is_a(e,h)}
∪{is_a(a,f)}∪{is_a(a,g)}∪{is_a(b,g)}∪{is_a(b,h)}

b)

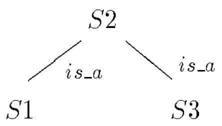


Alle Relationen sind is_a-Relationen

c) (a), (b)
(a,c), (b,d,e)
(a,c,f,$\underline{g}$),(b,d,e,$\underline{g}$,h)

A.2.4 Frames

Lösung A.11
Beachte:



$\Rightarrow EE1 \subseteq EE2$

$F1$	$S1$
	$EE = EE1$
	$\mid SE \mid = 1$

$F2$	$S2$
	$EE = EE1$

$F3$	$S2$
	$EE = EE2$
	$\mid SE \mid = 1$
	$SE = \{S1\}$

$F4$	$S2$
	$EE = EE2$
	$SE = \{S1\}$

$F5$	$S1$
	$EE = EE1$

Es handelt sich stets um nichtterminale Slots, da als Einträge wieder Frames erlaubt sind.

a) Prüfe Def. 5.11 i) bis iii).
Für die Slot-Namen $S1$ und $S2$ gilt $klasse(S1)$ und $klasse(S2)$.
Also sind es keine Relationen-Slots.
Es handelt sich auch um keine Rollen-Slots, da niemals $may\_be\_a(k, S1)$ bzw. $may\_be\_a(k, S2)$ erfüllt ist.

$S1$ in $F1$: $\forall x \in EE1$ gilt $is\_a(x, S1) \vee instanz\_von(x, S1)$.
Also ist $S1$ in $F1$ ein assoziativer Slot.

Ebenso $S1$ in $F5$.
Analog folgt: Auch $S2$ in $F3$ und $S2$ in $F4$ sind assoziative Slots.

$S2$ in $F2$: Für $x \in EE1$ gilt wegen $is\_a(S1, S2)$ auch $x \in EE2$, d.h. $is\_a(x, S2) \vee instanz\_von(x, S2)$. Also ist auch $S2$ in $F2$ ein assoziativer Slot.

b) Wegen 2) und 5) beschreiben $F2$ und $F5$ dieselben Klassen, nur mit verschiedenen Namen für das einzige Attribut.
Wegen 1) und 3) beschreiben $F1$ und $F3$ dieselben Klassen:
Für jedes Element von $F1$ kommt als Wert für das einzige Attribut nur genau ein Wert aus $EE1$ in Frage.
Für jedes Element von $F3$ kommt als Wert für das einzige Attribut nur genau ein Wert aus $EE1$ in Frage, diesmal wegen des aktuellen Eintrages $SE = \{S1\}$.
Aus 1) und 2) folgt sofort $is\_a(F1, F2)$, aber nicht umgekehrt.
Wegen 2) sind für jedes Element von $F2$ als Wert für das einzige Attribut beliebige Elemente aus $EE1$ erlaubt.
Wegen 4) sind für jedes Element von $F4$ als Wert für das einzige Attribut beliebige Elemente aus $EE2$ erlaubt, jedoch muß wegen $SE = \{S1\}$ mindestens eines aus $EE1$ sein. Daher gilt $is\_a(F2, F4)$, aber nicht umgekehrt.
Die restlichen Beziehungen ergeben sich aus der Transitivität von is_a.

c)

$$\begin{tikzcd} F4 \arrow[d, dash, "is\_a"'] & \\ F2 \arrow[r, leftrightarrow, "koref"] \arrow[d, dash, "is\_a"'] & F5 \\ F1 \arrow[r, leftrightarrow, "koref"] & F3 \end{tikzcd}$$

Literaturverzeichnis

[Ban90] R. R. Banerji. *Formal Techniques in AI.* North Holland, 1990.

[BB87] K. H. Bläsius and H. J. Bürckert. *Deduktionssysteme.* Oldenbourg, 1987.

[Bun83] A. Bundy. *The Computer Modelling of Mathematical Reasoning.* Academic Press, 1983.

[CM81] W. F. Clocksin and C.S. Mellish. *Programming in PROLOG.* Springer, 1981.

[CM85] E. Charniak and D. McDermott. *Introduction to Artifical Intelligence.* Addison-Wesley, 1985.

[GN89] M. R. Genesereth and N. J. Nilsson. *Logische Grundlagen der künstlichen Intelligenz.* Vieweg, 1989.

[Hel91] G. Helm. *Symbolische und konnektionistische Modelle der menschlichen Informationsverarbeitung.* IFB 288. Springer, 1991.

[HK89] D. Hofbauer and R.-D. Kutsche. *Grundlagen des maschinellen Beweisens.* Vieweg, 1989.

[Kai89] H. Kaindl. *Problemlösen durch heuristische Suche in der Artificial Intelligence.* Springer, 1989.

[KBS86] H. Kleine-Büning and S. Schmitgen. *PROLOG.* Teubner, 1986.

[KK88] L. Kanal and V. Kumar. *Search in Artificial Intelligence.* Springer, 1988.

[KLN87] D. Klahr, P. Langley, and R. Neche. *Production Systems, Models of Learning and Development.* MIT Press, 1987.

[Kow79] R. Kowalski. *Logic for Problem Solving.* North Holland, 1979.

[Kre91] H.-J. Kreowski. *Logische Grundlagen der Informatik.* Oldenbourg, 1991.

[Kro87] F. Kroeger. *Temporal Logic of Programs.* Springer, 1987.

[KSH91] R. Kruse, E. Schwecke, and J. Heinsohn. *Uncertainly and Vagueness in Knowledge Based Systems.* Springer, 1991.

[Len90] D. B. Lenat. *The nature of Heuristics.* Artificial Intelligence, 19:189 - 249, 1990.

[Llo84] J. W. Lloyd. *Foundations of Logic Programming.* Springer, 1984.

[Nil82] N. J. Nilsson. *Principles of Artificial Intelligence.* Springer, 1982.

[Pea85] J. Pearl. *Heuristics.* Addison-Wesley, 1985.

[Rei89] A. Reinefeld. *Spielbaum-Suchverfahren.* Springer, 1989.

[Rei91] U. Reimer. *Einführung in die Wissensrepräsentation.* Teubner, 1991.

[Ric89] M. M. Richter. *Prinzipien der Künstlichen Intelligenz.* Teubner, 1989.

[Sie90] D. Siefkes. *Formalisieren und Beweisen.* Vieweg, 1990.

[Sme88] Ph. Smets. *Non-Standard Logics for Automated Reasoning.* Academic Press, 1988.

[Som92] L. Sombe. *Schließen bei unsicherem Wissen.* Vieweg, 1992.

[Wil87] R. Wille. Bedeutungen von Begriffsverbänden. In B. Ganter, R. Wille and K. E. Wolff, editors, *Beiträge zur Begriffsanalyse*,Seiten 161 - 211. BI Wissenschaftsverlag, 1987.

[Win89] P. H. Winston. *Künstliche Intelligenz.* Addison-Wesley, 1989.

Stichwortverzeichnis