h_da

HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Wissensbasierte Diagnostik

Kap. 4.3: Wissensbasierte Systeme: Aussagenlogik, Prädikatenlogik & logische Programmierung

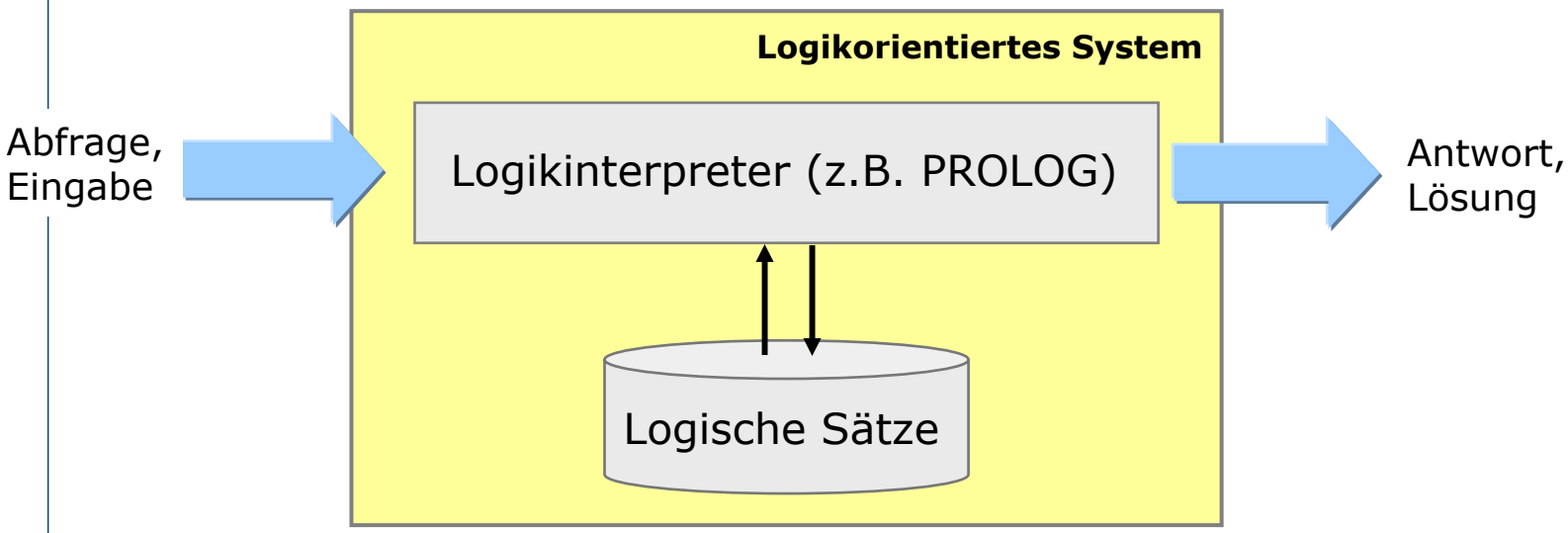
Dr. N. Waleschkowski
Fachbereich Informatik
c/o Semantis Information Builders GmbH
www.semantis-ib.de

Vorlesung Master-Studiengang
Wintersemester 2009/10

M.C. Escher: Andere Welt

Logik und Logikverarbeitung – Das Prinzip

- In einem logikbasierten System wird das Wissen in Form von logischen Ausdrücken repräsentiert und von einem Logikprozessor verarbeitet.



Aussagenlogik (1)

- Wir betrachten zunächst die Aussagenlogik, bei der es nur konstante Argumente von sog. Prädikaten gibt. Ein Prädikat ist dabei ein Ausdruck, der nur die Werte WAHR und FALSCH annehmen kann. Die allgemeine Prädikatenlogik erlaubt auch Variable als Argumente und führt die sog. Quantoren ein.

AUSSAGENLOGIK

- Die einfache Aussagenlogik (engl. propositional calculus) kennt nur Aussagen (proposition), d.h. Prädikate mit konstanten Argumenten. Hier einige Beispiele:
  - IST_RENNWAGEN (FERRARI_1)
  - ¬ IST_RENNWAGEN (WERKSTATT_2)
  - IST_IN (FERRARI_1, WERKSTATT_2)
  - ¬ IST_EINGESCHALTET (DIAGNOSECOMPUTER_5)

FERRARI_1 ist ein Rennwagen.

WERKSTATT_2 ist kein Rennwagen.

FERRARI_1 ist in WERKSTATT_2.

DIAGNOSECOMPUTER_5 ist nicht eingeschaltet.

- Jedes Prädikat hat also ein oder mehrere Objekte als Parameter.
- Prädikate können negiert werden.
- Durch jedes Prädikat wird eine Aussage getroffen, ob eine Behauptung über ein oder mehrere Objekte wahr oder falsch ist.

Aussagenlogik (2)

FERRARI_1 ist ein Rennwagen und steht in WERKSTATT_2.

WERKSTATT_2 ist weder ein Rennwagen noch ein Telefon.

DIAGNOSECOMPUTER_5 ist nicht eingeschaltet oder FERRARI_1 ist defekt.

Wenn DIAGNOSECOMPUTER_5 defekt ist, dann ist WEKSTATT_2 nicht betriebsbereit.

- Aussagen (elementare aussagenlogische Formeln) können zu komplexeren Aussagen verknüpft werden. Hier einige Beispiele:
  - IST_RENNWAGEN (FERRARI_1) $\wedge$ IST_IN (FERRARI_1, WERKSTATT_2)
  - $\neg$ IST_RENNWAGEN(WERKSTATT_2) $\wedge \neg$ IST_TELEFON (WERKSTATT_2)
  - $\neg$ IST_EINGESCHALTET (DIAGNOSECOMPUTER _5) $\vee$ IST_DEFEKT (FERRARI_1)
  - DEFEKT (DIAGNOSECOMPUTER_5) $\Rightarrow \neg$ BETRIEBSBEREIT (WERKSTATT_2)
- In diesen Beispielen kommen logische Verknüpfungen elementarer Aussagen vor. Die Negation sowie die wichtigsten zweistelligen Verknüpfungen – es gibt 16 – sind unten in einer Wahrheitswertetafel mit ihren Wahrheitswerten aufgeführt. Dabei sind die Symbole A und B Stellvertreter für konkrete Aussagen.

A	$\neg$ A
W	F
F	W

A	B	A $\wedge$ B	A $\vee$ B	A $\Rightarrow$ B	A XOR B	A $\Leftrightarrow$ B
W	W	W	W	W	F	W
W	F	F	W	F	W	F
F	W	F	W	W	W	F
F	F	F	F	W	F	W

Aussagenlogik (3)

- Insbesondere die Zeile 3 der WENN-DANN-Verknüpfung ($\Rightarrow$), auch Implikation genannt, bereitet manchmal einige Verständnis-probleme. Wieso sollte aus etwas Falschem etwas Wahres folgen können? Und wieso sollte aus etwas Falschem gleichzeitig etwas Falsches folgen können?
- Aus etwas Falschem kann man offensichtlich alles, also etwas Wahres oder etwas Falsches schließen. Das ist aber tatsächlich wahr.
- Wie die folgende Wahrheitswertetafel zeigt, ist $A \Rightarrow B$ logisch äquivalent zu $\neg A \vee B$.

A	B	$A \Rightarrow B$	$\neg A \vee B$
W	W	W	W
W	F	F	F
F	W	W	W
F	F	W	W

- Diese Äquivalenz wird beim sog. Resolutionsbeweis in der logischen Programmierung benötigt.

Aussagenlogik (4)

- Weitere logische Äquivalenzen

Gesetz der doppelten Negation:

$\neg \neg A \Leftrightarrow A$

De Morgan's Gesetze (Verteilen der Negation):

$\neg (A \wedge B) \Leftrightarrow \neg A \vee \neg B$

$\neg (A \vee B) \Leftrightarrow \neg A \wedge \neg B$

Distributivgesetze:

$A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$

$A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$

Äquivalenzen mit Implikation:

$A \Rightarrow B \Leftrightarrow \neg A \vee B$

$\neg (A \Rightarrow B) \Leftrightarrow A \wedge \neg B$

$A \Rightarrow B \Leftrightarrow \neg B \Rightarrow \neg A$ (Kontraposition)

Wenn keine Klammerung etwas anderes vorgibt, gelten folgende Präferenzen bei der Anwendung logischer Operatoren:

$(\text{Negation } \neg) \prec (\text{Und } \wedge) \prec (\text{Oder } \vee) \prec (\text{Implikation } \Rightarrow) \prec (\text{Äquivalenz } \Leftrightarrow)$

$\prec$ bedeutet „hat Präferenz vor"

Aussagenlogik (5)

- Logische Ableitungen

Die Konjunktion:	A, B	→	$A \wedge B$
Die Disjunktion:	A, B	→	$A \vee B$
Modus Ponens:	A, $A \Rightarrow B$	→	B

Dabei bedeutet → „daraus läßt sich ableiten, dass"

- Einige Beispiele für logische Schlüsse:

Es gelten folgende Aussagen:

1. IST_RENNWAGEN (FERRARI_1)
2. IST_IN (FERRARI_1, WERKSTATT_2)

Dann gelten auch folgende Aussagen:

- IST_RENNWAGEN (FERRARI_1) $\wedge$ IST_IN (FERRARI_1, WERKSTATT_2)
- IST_RENNWAGEN (FERRARI_1) $\vee$ IST_IN (FERRARI_1, WERKSTATT_2)

Es gelten folgende Aussagen:

1. DEFEKT (DIAGNOSEOMPUTER_1)
2. DEFEKT (DIAGNOSEOMPUTER _1) $\Rightarrow \neg$ BETRIEBSBEREIT (WERKSTATT_2)

Dann gilt gemäß Modus Ponens:

- $\neg$ BETRIEBSBEREIT (WERKSTATT_2)

Prädikatenlogik (1)

- Die Prädikatenlogik (engl. predicate calculus) ist eine Erweiterung der Aussagenlogik. Zunächst eine wichtige Begriffsdefinitionen:

Ein Prädikat ist ein Ausdruck, der (für konkrete Objekte) nur die Werte WAHR und FALSCH annehmen kann.

Eine Konstante ist ein ganz bestimmtes Objekt, z.B. ein konstantes Argument eines Prädikats.

Eine Variable ist Stellvertreter eines beliebigen Objekts, z.B. ein variables Argument eines Prädikats.

Logische Operatoren verknüpfen Aussagen/Prädikate mit Aussagen/Prädikaten.

- NICHT (Negation, $\neg$)
- UND (Konjunktion, $\wedge$)
- ODER (Disjunktion, $\vee$)
- WENN-DANN (Implikation, $\Rightarrow$)

Eine Funktion ist ein Ausdruck mit einem oder mehreren Objekten als Argument und liefert ein neues Objekt zurück.

Quantoren sind Konstrukte, die es erlauben, Aussagen über die Existenz bestimmter Objekte zu treffen bzw. Aussagen über Sätze zu treffen, die für alle Objekte mit bestimmten Eigenschaften gelten.

Prädikatenlogik (2)

Quantoren in der Prädikatenlogik

Arten von Quantoren:

Der Allquantor ∀ (engl. universal quantifier) gestattet logische Sätze, die sich auf alle Objekte mit bestimmten Eigenschaften beziehen.

(∀ x) P(x) „Für alle Objekte x ist P(x) erfüllt (wahr).“

Der Existenzquantor ∃ (engl. existential quantifier) gestattet Aussagen über die Existenz bestimmter Objekte.

(∃ x) P(x) „Es gibt ein Objekt x, für das P(x) erfüllt (wahr) ist.“

Einige Äquivalenzen mit Quantoren:

De Morgan's Gesetze (Verteilen der Negation)

¬ (∀ x) P(x) ⇔ (∃ x) ¬ P(x) „Nicht für alle x gilt P(x).“

¬ (∃ x) P(x) ⇔ (∀ x) ¬ P(x) „Es gibt kein x mit P(x) bzw. für alle x gilt P(x) nicht.“

Distributivgesetze:

(∀ x) (P(x) ∧ Q(x)) ⇔ (∀ x) P(x) ∧ (∀ x) Q(x)

(∃ x) (P(x) ∨ Q(x)) ⇔ (∃ x) P(x) ∨ (∃ x) Q(x)

Umbenennung von quantifizierten Variablen:

(∀ x) P(x) ⇔ (∀ y) P(y) sowie (∃ x) P(x) ⇔ (∃ y) P(y)

Prädikatenlogik (3)

- Hier einige Begriffe aus der Wissensrepräsentation in der Sprache der Prädikatenlogik:

Terme (engl. terms) dienen als Argumente von Prädikaten. Terme sind also konstante oder variable Objekte bzw. Funktionsergebnisse, also von Funktionen gelieferte Objekte.

Ein Atom (engl. atomic formula) ist ein Element, das nicht mehr in kleinere Einheiten unterteilt werden kann, also etwa eine Aussage oder ein einzelnes Prädikat mit seinen Argumenten.

Ein Literal (engl. Literal) ist ein Atom oder ein negiertes Atom.

Wohlgeformte Ausdrücke (engl. well formed formulas) sind einzelne Literale oder durch logische Operatoren verknüpfte Literale, die ihrerseits wiederum wohlgeformte Ausdrücke enthalten können. Dabei kann es sich auch um quantifizierte Ausdrücke handeln.

Sätze (engl. sentences) sind wohlgeformte Ausdrücke ohne freie Variablen, in denen also alle Variable gebunden sind. Eine Variable heißt gebunden, wenn ihr ein ganz bestimmtes Objekt zugewiesen wurde, oder wenn sie an einen Quantor gekoppelt ist.

Klauseln (engl. clauses) sind Sätze, welche nur aus Literalen bestehen, die durch ODER verknüpft sind.

Prädikatenlogik (4)

Prädikatenlogiken 0., 1. und höherer Ordnung bzw. Stufe

- Wir beschäftigen uns im folgenden mit der Prädikatenlogik erster Ordnung (engl. first order predicate calculus). Diese Logik arbeitet mit den bisher eingeführten Begriffen und ist dadurch gekennzeichnet, dass Variablen nur Objekte bezeichnen dürfen.
- In Prädikatenlogiken höherer Ordnung können Variablen auch Prädikate bezeichnen oder sogar Quantoren.
- Die Aussagenlogik heißt auch Prädikatenlogik 0. Ordnung. In dieser Logik gibt es keine Variablen.

Prädikatenlogik (5)

Wissensrepräsentation mittels Prädikatenlogik

- Im folgenden soll die Mächtigkeit der Ausdrucksfähigkeit der Prädikatenlogik erster Ordnung an einigen Beispielen demonstriert werden.

1. $(\exists x)$ [IST_RENNWAGEN(x)]
2. $(\exists x)$ [IST_ DIAGNOSECOMPUTER(x) $\wedge$ IST_IN (x, WERKSTATT_2)]
3. $(\exists x)$ [IST_IN (x, WERKSTATT_2) $\wedge \neg$ IST_RENNWAGEN(x)]
4. $(\exists x)$ [$\neg$ EINGESCHALTET(x)]
5. $(\exists x)$ [IST_ DIAGNOSECOMPUTER(x) $\wedge \neg$ IST_DEFEKT(x)]
6. $(\forall x)$ [IST_DIAGNOSECOMPUTER(x) $\Rightarrow \neg$ EINGESCHALTET(x)]
7. $(\forall x)$ [IST_ DIAGNOSECOMPUTER(x) $\wedge$ IST_IN (x, WERKSTATT_2) $\Rightarrow$ $\neg$ EINGESCHALTET(x)]
8. $(\forall x)$ [IST_DIAGNOSECOMPUTER(x) $\Rightarrow (\exists y)$ [IST_IN(x,y)]]
9. $(\exists x)$ $(\exists y)$ [IST_RENNWAGEN(x) $\wedge$ IST_RENNWAGEN(y) $\wedge$ UNGLEICH(x,y)]
10. $(\forall x)$ $(\forall y)$ [(IST_ DIAGNOSECOMPUTER (x) $\vee$ IST_HEBEBUEHNE (x)) $\wedge$ ($\neg$ EINGESCHALTET (x) $\vee$ DEFEKT (x)) $\wedge$ IST_WERKSTATT(y) $\wedge$ IST_IN (x,y) $\Rightarrow \neg$BETRIEBSBEREIT (y)]

1. Es gibt ein Objekt x, das ein Rennwagen ist.
2. Es gibt mind. einen Diagnosecomputer in WERKSTATT_2.
3. Es gibt etwas in WERKSTATT_2,was kein Rennwagen ist.
4. Irgendetwas ist nicht eingeschaltet.
5. Irgendein Diagnosecomputer ist nicht defekt.
6. Alle Diagnosecomputer sind ausgeschaltet.
7. Alle Diagnosecomputer in WERKSTATT_2 sind nicht eingeschaltet.
8. Für alle Diagnosecomputer gibt es etwas (z.B. eine Halle), worin sich diese befinden.
9. Es gibt mindestens zwei verschiedene Rennwagen.
10. Wenn ein Diagnosecomputer oder eine Hebebühne in einer Werkstatt nicht eingeschaltet oder defekt sind, dann ist diese Werkstatt nicht betriebsbereit.

Das Resolutionsverfahren (1)

Logisches Beweisen mit Robinson's Resolutionsverfahren

Modus Ponens

$$\left.\begin{array}{l} A \\ A \Rightarrow B \end{array}\right\} \rightarrow B$$

- Das Resolutionsverfahren basiert auf der Ableitungsregel Modus Ponens $A, A \Rightarrow B \rightarrow B$. Wie vorher gezeigt, ist $A \Rightarrow B$ logisch äquivalent zu $\neg A \vee B$. Der Modus Ponens lautet also in anderer Schreibweise

$$A, \neg A \vee B \rightarrow B$$

- Nun verallgemeinern wir diese Ableitungsregel zum Schließen durch Resolution.

Schließen durch Resolution:

1. Voraussetzung: $A \vee B_1 \vee B_2 \vee \ldots \vee B_n$ ist wahr.
2. Voraussetzung: $\neg A \vee C_1 \vee C_2 \vee \ldots \vee C_m$ ist wahr.

Dann gilt die Schlußfolgerung: $\underbrace{B_1 \vee B_2 \vee \ldots \vee B_n} \vee \underbrace{C_1 \vee C_2 \vee \ldots \vee C_m}$

- Beweis: Aus den Voraussetzungen entsteht der Schluß quasi durch „Kürzen des Prädikats A".

  Angenommen, A ist wahr. Dann ist auch $A \vee B_1 \vee B_2 \vee \ldots \vee B_n$ wahr.
  Wenn A wahr ist, dann ist $\neg A$ falsch. Dann muss aber der Rest $C_1 \vee C_2 \vee \ldots \vee C_m$ der 2. Voraussetzung wahr sein.
  Dann ist aber wegen der reinen ODER-Verknüpfung auch $B_1 \vee B_2 \vee \ldots \vee B_n \vee C_1 \vee C_2 \vee \ldots \vee C_m$ wahr.
  Für die Annahme, A sei falsch, verläuft der Beweis ganz ähnlich.

Das Resolutionsverfahren (2)

Logisches Beweisen mit dem Resolutionsverfahren

- Ein Sonderfall liegt vor, wenn alle Prädikate B_i und C_j weggelassen werden. Dann reduziert sich das Resolutionsverfahren zu

  $A, \neg A \rightarrow \varnothing$ ($\varnothing$ ist die Leerklausel.)

  Aus dem offensichtlichen Widerspruch A und $\neg$ A allein läßt sich natürlich nichts schließen.

- Ergibt sich umgekehrt als Resolutionsergebnis die Leerklausel, so muss ein Widerspruch in den Voraussetzungen vorgelegen haben.

- Darauf basiert der sog. Widerspruchsbeweis. Dies ist ein grundlegendes Prinzip in der logischen Programmierung. Betrachte hierzu das folgende Beispiel:

  Wir nehmen die Aussagen A1 –A3 als wahr an.

  A1 $\neg$ DEFEKT (DIAGNOSECOMPUTER_1) $\wedge \neg$ DEFEKT (HEBEBUEHNE_7) $\wedge$ EINGESCHALTET (HEBEBUEHNE_7)

  A2 $\neg$ DEFEKT (HEBEBUEHNE_7) $\wedge$ EINGESCHALTET (HEBEBUEHNE_7) $\Rightarrow$ AKTIVIERT (HEBEBUEHNE_7)

  A3 $\neg$ DEFEKT(DIAGNOSECOMPUTER_1) $\wedge$ AKTIVIERT (HEBEBUEHNE_7)] $\Rightarrow$ BETRIEBSBEREIT (WERKSTATT_2)

Das Resolutionsverfahren (3)

Aus A1 werden drei Klauseln:

K1: ¬ DEFEKT (DIAGNOSECOMPUTER_1)

K2: ¬ DEFEKT (HEBEBUEHNE_7)

K3: EINGESCHALTET (HEBEBUEHNE_7)

Aus A2 wird die Klausel:

K4: DEFEKT (HEBEBUEHNE_7) ∨ ¬ EINGESCHALTET(HEBEBUEHNE_7)
∨ AKTIVIERT (HEBEBUEHNE_7)

Aus A3 wird die Klausel:

K5: DEFEKT(DIAGNOSECOMPUTER_1) ∨ ¬ AKTIVIERT (HEBEBUEHNE_7)
∨ BETRIEBSBEREIT (WERKSTATT_2)

Zu beweisen sei der Satz S1

S1: BETRIEBSBEREIT (WERKSTATT_2)

Daher fügen wir die Klausel K6 als Negation von S1 hinzu.

K6: ¬ BETRIEBSBEREIT (WERKSTATT_2)

Aus K2 und K4 folgt:

¬ EINGESCHALTET(HEBEBUEHNE_7) ∨ AKTIVIERT (HEBEBUEHNE_7)

Daraus folgt mit K3:

AKTIVIERT (HEBEBUEHNE_7)

Aus K1 und K5 folgt:

¬ AKTIVIERT (HEBEBUEHNE_7) ∨ BETRIEBSBEREIT (WERKSTATT_2)

Insgesamt folgt dann:

BETRIEBSBEREIT (WERKSTATT_2)

Das ist ein Widerspruch zu K6, also gilt die Leerklausel. Damit ist Satz S1 bewiesen, die WERKSTATT_2 ist also betriebsbereit.

Der Trick beim Widerspruchsbeweis besteht darin, daß die zu beweisende Behauptung negiert wird. Dann zeigt man, dass die negierte Aussage falsch ist. Also muss die ursprüngliche Behauptung wahr sein.

Das Resolutionsverfahren (4)

- Schema des Widerspruchsbeweises mit dem Resolutionsverfahren

1. Stelle alle wahren Aussagen des zu untersuchenden Systems zusammen.
2. Füge die Negation des zu beweisenden Satzes hinzu.
3. Transfomiere alle Sätze in Klauselform.
4. Versuche nun, durch Anwendung des Resolutionsverfahrens die Leerklausel abzuleiten.

   Wenn das Resolutionsergebnis die Leerklausel ist, so lag ein Widerspruch vor. Also muss die Negation des zu beweisenden Satzes falsch sein. Damit muss der Satz selbst wahr gewesen sein.

Das Resolutionsverfahren (5)

- Gemäß Schritt 3. des Verfahrens müssen alle Sätze in Klauselform transformiert werden. Es stellt sich die Frage, wie das erreicht werden kann.
- Im folgenden wird das Transformationsverfahren dargestellt.

Transformation von Sätzen in Klauselform:

1. Beseitigung der Implikationen.
2. Negationen auf die einzelnen Prädikate durchdrücken.
3. Umbenennung der Variablen, so daß jeder Quantor eindeutig benannte Variablen hat.
4. Beseitigung der Existenzquantoren.
5. Herausziehen der Allquantoren.
6. Darstellung in konjunktiver Normalform.
7. Überführung der konjunktiven Normalform in eine Menge von Klauseln.
8. Umbenennung der Variablen, so dass jede Klausel eindeutig benannte Variablen erhält.
9. Weglassen der Allquantoren.

Die abgeleitete Klauselform ist logisch völlig äquivalent zur ursprünglichen Darstellung. Sie ist intuitiv allerdings nicht so gut zu erfassen.

Ihr Vorteil liegt darin, daß sie einfacher von Programmen verarbeitet werden kann, die mit der Resolutionsmethode arbeiten. Bsp.: Die Sprache PROLOG (Programming in Logic).

Das Resolutionsverfahren (6)

Skolemfunktion:

Die Einführung einer solchen Funktion dient zur eindeutigen Benennung von Objekten. Dieses Verfahren der Namensgebung wird in der Logik als Skolemfunktion bezeichnet, benannt nach dem Logiker Thoraf Skolem. Im Alltag tun wir etwas ähnliches auch sehr oft. Wenn wir den Namen einer Person, z.B. Otto, nicht kennen oder wenn dieser Name nicht eindeutig ist, weil wir mehrere Personen namens Otto kennen, dann bezeichnen wir diese Person eben als Herrn Schmidt oder den Ehemann von Fr. Schmidt oder den Leiter der Abteilung X, also durch Nennung einer Funktion.

- Die Transformation soll an einem Beispiel verdeutlicht werden.
- Das folgende Beispiel ist etwas konstruiert:

$$(\forall x)\ [\ \text{IST\_DIAGNOSECOMPUTER}\ (x) \wedge \neg\ \text{EINGESCHALTET}(x) \Rightarrow (\exists y)\ [\ \text{IST\_WERKSTATT}\ (y) \wedge \neg\ \text{BETRIEBSBEREIT}\ (y)\ ]\ ]$$

„Für alles, was ein Diagnosecomputer ist und nicht eingeschaltet ist, gibt es eine Werkstatt, die nicht betriebsbereit ist."

1. Beseitigung der Implikationen (via $A \Rightarrow B \Leftrightarrow \neg A \vee B$)

$$(\forall x)\ [\ \neg\ (\text{IST\_DIAGNOSECOMPUTER}\ (x) \wedge \neg\ \text{EINGESCHALTET}(x)) \vee (\exists y)\ (\ \text{IST\_WERKSTATT}\ (y) \wedge \neg\ \text{BETRIEBSBEREIT}\ (y)\ )\ ]$$

2. Negationen auf Prädikate durchdrücken

$$(\forall x)\ [(\neg\ \text{IST\_DIAGNOSECOMPUTER}\ (x) \vee \text{EINGESCHALTET}(x)) \vee (\exists y)\ (\ \text{IST\_WERKSTATT}\ (y) \wedge \neg\ \text{BETRIEBSBEREIT}\ (y)\ )\ ]$$

3. Umbenennung der Variablen

Zu jedem Quantor müssen eindeutige Variablen gehören.
Dies ist bereits der Fall, so dass keine Änderung erforderlich ist.

$$(\forall x)\ [(\neg\ \text{IST\_DIAGNOSECOMPUTER}\ (x) \vee \text{EINGESCHALTET}(x)) \vee (\exists y)\ (\ \text{IST\_WERKSTATT}\ (y) \wedge \neg\ \text{BETRIEBSBEREIT}\ (y)\ )\ ]$$

4. Beseitigen der Existenzquantoren

Um den Existenzquantor loszuwerden, müssen wir die Werkstatt benennen, in der sich der Diagnosecomputer befindet. Dieses soll die neu eingeführte Funktion werkstatt(x) leisten. Eine solche Funktion zur Entfernung eines Existenzquantors, bei der jede an einen Allquantor gebundene Variable zu einem ihrer Argumente wird, nennt man auch Skolemfunktion.

Das Resolutionsverfahren (7)

$$(\forall x)\ [(\neg\ \text{IST\_DIAGNOSECOMPUTER}\ (x) \vee \text{EINGESCHALTET}(x)\ ) \vee (\ \text{IST\_WERKSTATT}\ (\text{werkstatt}(x)) \wedge \neg\ \text{BETRIEBSBEREIT}(\text{werkstatt}(x))\ )]$$

5. Herausziehen der Allquantoren

Alle Allquantoren könnten nun nach vorne gezogen werden.
Dies ist bereits der Fall, so dass keine Änderung erforderlich ist.

Aber $(\forall x)\ [\ \text{IST\_RENNWAGEN}\ (x)\ ] \vee (\forall y)\ [\ \text{IST\_WERKSTATT}\ (y)\ ]$
würde übergehen in die sog. Pränexform

$$(\forall x)\ (\forall y)\ [\ \text{IST\_RENNWAGEN}\ (x) \vee \text{IST\_WERKSTATT}\ (y)\ ]$$

6. Darstellung in konjunktiver Normalform (KNF)

$$(\forall x)\ [(\neg\ \text{IST\_DIAGNOSECOMPUTER}(x) \vee \text{EINGESCHALTET}(x) \vee \text{IST\_WERKSTATT}\ (\text{werkstatt}(x))\ ) \wedge (\ \neg\ \text{IST\_DIAGNOSECOMPUTER}(x) \vee \text{EINGESCHALTET}(x) \vee \neg\text{BETRIEBSBEREIT}(\text{werkstatt}(x))\ )\ ]$$

7. Weglassen der Allquantoren

Wir vereinbaren ab jetzt, dass alle Variablen als allquantisiert gelten. Daher können die Allquantoren weggelassen werden.

8. Transformation der KNF-Darstellung in eine Menge von Klauseln

Die UND-Verknüpfungen werden aufgelöst, und es ergeben sich zwei Klauseln.

$$\neg\ \text{IST\_DIAGNOSECOMPUTER}(x) \vee \text{EINGESCHALTET}(x) \vee \text{IST\_WERKSTATT}(\text{werkstatt}(x))$$

$$\neg\ \text{IST\_DIAGNOSECOMPUTER}(x) \vee \text{EINGESCHALTET}(x) \vee \neg\text{BETRIEBSBEREIT}(\text{werkstatt}(x))$$

Das Resolutionsverfahren (8)

9. Umbenennung der Variablen, damit jede Klausel eindeutig benannte Variablen besitzt

$\neg$ IST_DIAGNOSECOMPUTER(x) $\vee$ EINGESCHALTET(x)
$\vee$ IST_WERKSTATT(werkstatt(x))

$\neg$ IST_DIAGNOSECOMPUTER(y) $\vee$ EINGESCHALTET(y)
$\vee$ $\neg$BETRIEBSBEREIT(werkstatt(y))

- Damit ist die Transformation in eine Klauselmenge abgeschlossen.
- Die erste Klausel ist stets erfüllt, da IST_WERKSTATT(werkstatt(x)) stets erfüllt ist, weil werkstatt(x) immer eine Werkstatt liefert. Die Klausel kann daher weggelassen werden.
- Dies gilt z.B. auch, wenn ein Prädikat gleichzeitig mit seiner Negation auftaucht. Solche Aussagen heißen auch Tautologien. Sie können ebenfalls weggelassen werden.
- Die Transformation von logischen Sätzen „zu Fuß" ist schon eine größere Arbeit, die Sorgfalt erfordert. Um große Mengen von Sätzen handhaben zu können, ist es daher notwendig, diesen Prozess zu automatisieren.

Das Resolutionsverfahren (9)

Die Unifikation

- Um das Prinzip der Unifikation zu erläutern, zunächst ein kleines Beispiel:

  (1) Es gelten die Klauseln K1, K2 und K3.

  K1: $\neg$ IST_COMPUTER (x) $\vee$ EINGESCHALTET(x)
  $\vee \neg$ BETRIEBSBEREIT (werkstatt(x))

  K2: IST_COMPUTER (DIAGNOSECOMPUTER_5)

  K3: $\neg$ EINGESCHALTET(DIAGNOSECOMPUTER_5)

  (2) Die Skolemfunktion werkstatt(DIAGNOSECOMPUTER_5) soll WERKSTATT_2 zurückliefern.

  (3) Wir beweisen, dass dann der folgende Satz gilt:

  S1: $\neg$ BETRIEBSBEREIT(WERKSTATT_2)

  (4) Wir fügen also zunächst die Negation von S1 als Klausel K4 hinzu:

  K4: BETRIEBSBEREIT(WERKSTATT_2)

  (5) Die Klauseln K1 und K2 lassen sich dann zusammenführen, wenn die Variable x den Wert DIAGNOSECOMPUTER_5 annimmt. Dies gilt für alle Stellen, an denen x in der Klausel auftaucht. Dann ergibt sich durch Resolution der beiden sog. Resolventen sowie der o.a. Skolemfunktion

  EINGESCHALTET(DIAGNOSECOMPUTER_5) $\vee \neg$BETRIEBSBEREIT (WERKSTATT_2)

  (6) Mittels Klausel K3 folgt dann: $\neg$BETRIEBSBEREIT (WERKSTATT_2)

  (7) Wegen Klausel K4 folgt die leere Klausel. Damit ist der Satz S1 beweisen.

Das Resolutionsverfahren (10)

- In (5) haben wir eine sog. Unifikation durchgeführt, als wir der Variablen x das Objekt DIAGNOSECOMPUTER_5 zugeordnet haben.
- Der Prozess, zwei Literale durch konsistente Ersetzung von Variablen anderer Terme gleich zu machen, heißt Unifikation.
- Das Ergebnis der Unifikation ist ein Unifikator, d.h. eine Abbildung

  U: {Menge der Variablen} → {Menge der Terme},

  die beschreibt, welche Variablen durch welche Terme ersetzt werden müssen.
- Bsp.: Betrachte die Terme

  datum(2,M,J)

  datum(Tag,Monat,2006)
- Ein Unifikator U für diese Terme lautet:

  U(Tag) = 2, U(M) = 3, U(Monat) = 3, U(J) = 2006
- Unifikatoren sind oft nicht eindeutig. Ein anderer Unifikator U‘ ist

  U‘(Tag) = 2, U‘(M) = Monat, U‘(Monat) = Monat, U‘(J) = 2006
- I.d.R. ist man an möglichst allgemeinen Unifikatoren interessiert, die möglichst viele Variablen variabel lassen. Solche Unifikatoren heißen allgemeinste Unifikatoren.

Das Resolutionsverfahren (11)

- Im allgemeinen Fall ist die Unifikation allerdings wesentlich komplizierter, worauf wir hier aber nicht näher eingehen können. Hier noch einige Erläuterungen:

- Zwei Literale können unter folg. Bedingungen unifiziert werden:
  - Beide Literale besitzen dasselbe Prädikat mit derselben Stelligkeit, d.h. mit derselben Anzahl von Argumenten, aber mit verschiedenen Vorzeichen.
  - Die Terme lassen sich unter Verwendung geeigneter Substitutionen so unifizieren , dass beide Literale bis auf das Vorzeichen identisch werden. (Die Substitutionen müssen in der gesamten Klausel vorgenommen werden.)

  Konstante – Konstante
  - unifiziert, wenn beide gleich sind.

  Konstante – Variable
  - unifiziert, wenn die Variable den Wert der Konstanten annehmen kann

  Konstante – Funktion
  - unifiziert, wenn die Funktion den Wert der Konstanten liefert.

  Variable – Variable
  - unifiziert, wenn beide Variablen den gleichen Wert angenommen haben oder eine Variable hat einen Wert, den dann auch die zweite annimmt oder beide haben keinen Wert und werden dann aneinander gebunden.

Das Resolutionsverfahren (12)

Variable – Funktion

- unifiziert, wenn die Variable bereits einen Wert hat, den auch die Funktion zurückliefert oder die Funktion eine Wert liefert, den die Variable dann annimmt (Dabei darf die Variable aber nicht in der Argumentliste der Funktion vorkommen, da ansonsten eine unendliche Rekursion entsteht.)

Funktion – Funktion

- unifiziert, wenn beide Funktionen das gleiche Objekt zurückliefern.

Liste – Liste

- Hier erfolgt die Unifikation elementweise.

Das Resolutionsverfahren (13)

Resolutionsstrategien

- **Das Reihenfolgeproblem**: In der Praxis tauchen häufig große Klauselmengen auf. Es ist offensichtlich, dass die Reihenfolge der Klauselverarbeitung einen großen Einfluss auf die Laufzeit haben kann.
- Zur Abhilfe werden verschiedene Resolutionsstrategien angeboten. Das Ziel besteht darin, möglichst schnell die leere Klausel abzuleiten.

Stützmengen- bzw. Set-of-Support-Strategie

- Nach einem Resolutionsschritt wird mindestens eine Klausel aus der Stützmenge (set-of-support) entfernt.

Unit-Preference-Strategie

- Hier werden Klauseln bevorzugt, die nur aus einem Literal bestehen,

Breadth-First-Strategie:

- Hier werden zuerst die Klauseln verarbeitet, die aus dem negierten Satz hervorgegangen sind. Durch Resolution ergibt sich dann die nächste Ebene etc. Bevor die nächsthöhere Ebene untersucht wird, werden stets zuerst alle Kombinationen von Klauseln einer Ebene sowie die der darunterliegenden Ebene untersucht.

Weitere Strategien: ...

Das Resolutionsverfahren (14)

Resolutionsstrategien

- **Das Halteproblem:** Ein anderes Problem ist, ob das Resolutionsverfahren überhaupt terminiert oder ob es ewig weiterläuft.
- Durch die Kombination von Klauseln entstehen oft kombinatorisch viele Möglichkeiten, die die Lösungsfindung schließlich abwürgen.
- Grundsätzlich kann es aber auch zu Endlosschleifen kommen, wenn die Leerklausel nicht ableitbar ist, etwa weil der zu beweisende Satz falsch ist. Parallel könnte man z.B. versuchen, das Gegenteil des Satzes zu beweisen. Das Verfahren, das zuerst fertig ist, gestattet dann eine Aussage bzgl. des Wahrheitsgehaltes.

Horn-Klausel-Logik (1)

Horn-Klauseln

- Die volle Prädikatenlogik ist ein sehr mächtiges System großer Ausdruckskraft. In dieser Allgemeinheit ist sie aber schwer zu handhaben und maschinell zu verarbeiten. Dazu muß die Menge der logischen Sätze syntaktisch vereinfacht werden.
- Es gibt eine wichtige Untermenge der Prädikatenlogik, die mittels Anwendung des Resolutionsverfahren maschinell gut zu handhaben ist. Das ist die sog. Horn-Klausel-Logik
- Horn-Klauseln haben die folgende Form:

  $A_1 \wedge A_2 \wedge ... \wedge A_n \Rightarrow B$ bzw. $\neg A_1 \vee \neg A_2 \vee ... \vee \neg A_n \vee B$

  (wg. $A \Rightarrow B \Leftrightarrow \neg A \vee B$)

  Dazu gehören auch Fakten, die als Regeln ohne Prämisse aufgefaßt werden.
- Die Horn-Klausel-Logik ist also die Teilmenge der Prädikatenlogik, welche zu Klauseln mit höchstens einem nichtnegierten Literal führt. Dies entspricht also Regeln der Form

  $A_1 \wedge A_2 \wedge ... \wedge A_n \Rightarrow B$

  bzw. Fakten der Form B.

Horn-Klausel-Logik (2)

- Die folgende Regel ist daher keine Horn-Klausel:

  $$A_1 \wedge A_2 \wedge \ldots \wedge A_n \Rightarrow B_1 \vee B_2$$

  Sie ist äquivalent zu

  $$\neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_n \vee B_1 \vee B_2 ,$$

  hat also zwei nichtnegierte Literale.

- Wir betrachten eine Regel mit ODER-verknüpften Bedingungen

  $$A_1 \vee A_2 \vee \ldots \vee A_n \Rightarrow B_1$$

  Sie ist äquivalent zu

  $$(\neg A_1 \wedge \neg A_2 \wedge \ldots \wedge \neg A_n) \vee B$$

  und führt daher zu n Horn-Klauseln

  $$\neg A_1 \vee B , \neg A_2 \vee B, ..., \neg A_n \vee B$$

- Die Sprache PROLOG (Programming in Logic) basiert auf der Horn-Klausel-Logik.