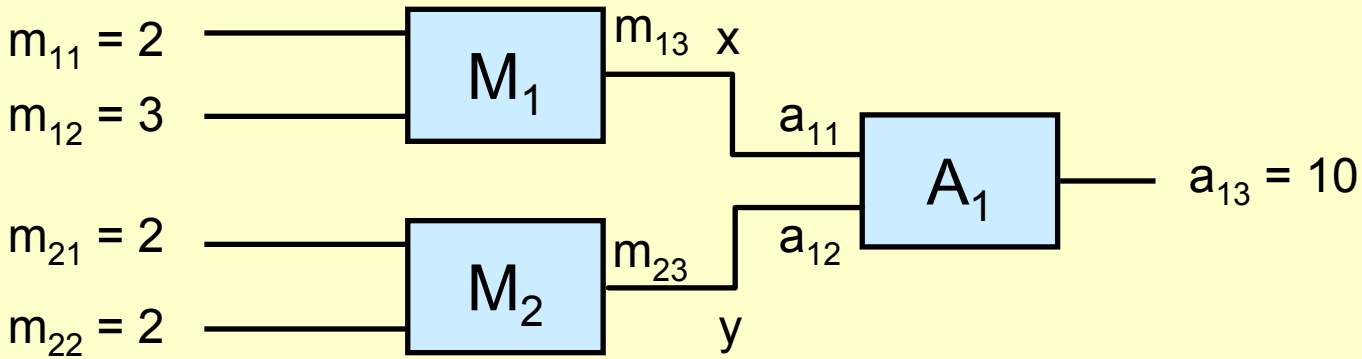
Wissensbasierte Systeme

Sebastian Iwanowski
FH Wedel

Kap. 2:
Logische Grundlagen der KI

Ein vereinfachtes Diagnose-Beispiel

Systemstruktur



Komponentenmodelle

- Multiplizierer M_i: $y_i = 1 \Rightarrow m_{i3} = m_{i1} * m_{i2}$; $y_i = 1$ v $y_i = 0$
- Addierer A_i : $x_i = 1 \Rightarrow a_{i3} = a_{i1} + a_{i2}$; $x_i = 1$ v $x_i = 0$

Frage: **In welcher Form braucht man die Constraints für das GDE-Prinzip ?**

KI-Antwort: **Beschreibe das Diagnoseproblem mit einer aussagenlogischen Formel !**

Wdh.: Aussagenlogik

Was ist eine Aussage ?

- **Eine Aussage ist ein beliebiges Objekt.**
- **In der Aussagenlogik sind Aussagen unteilbar.**
  - Wegen der Unteilbarkeit heißen Aussagen auch *Atome*
  - Da sie wegen der Unteilbarkeit sinnvollerweise mit Buchstaben abgekürzt werden, nennt man Aussagen auch *Literale*

Was ist ein Wahrheitswert ?

- **Ein Wahrheitswert ist ein Element aus einer zweielementigen Menge (z.B. dargestellt als {`w`, `f`}).**

Was macht die Aussagenlogik ?

- **Die Aussagenlogik beschäftigt sich mit Funktionen, die jeder Aussage einen Wahrheitswert zuordnen.**
  - Solche Funktionen heißen *binäre Funktionen*

Wdh.: Aussagenlogische Verknüpfungen

Einstelliger Operator:

- **Negation (¬)**

Zweistellige Operatoren:

- **Konjunktion (∧), Disjunktion (∨), Implikation (→), Äquivalenz (↔)**

p	q	¬p	p ∧ q	p ∨ q	p → q	p ↔ q
w	w	f	w	w	w	w
w	f	f	f	w	f	f
f	w	w	f	w	w	f
f	f	w	f	f	w	w

Syntaktische Ersetzungsregeln:

$p \rightarrow q \Leftrightarrow \neg p \vee q$

Ersetzen der Implikation durch ¬ und ∨

$p \leftrightarrow q \Leftrightarrow (p \rightarrow q) \wedge (q \rightarrow p)$

Ersetzen der Äquivalenz durch Implikationen

$\neg(p \wedge q) \Leftrightarrow \neg p \vee \neg q$

$\neg(p \vee q) \Leftrightarrow \neg p \wedge \neg q$

deMorgansche Regeln

$\neg\neg p \Leftrightarrow p$

Doppelte Negation

Folgerung aus den Ersetzungsregeln:

- Für die Darstellung jeder logischen Verknüpfung reichen die Operatoren ¬, ∧ und ∨ aus !
- Man kann sogar auf einen dieser 3 Operatoren verzichten !

  (auf welchen ?)

Wdh.: Aussagenlogische Formeln

- **Eine aussagenlogische Formel ist eine Verknüpfung von endlich vielen Literalen mit aussagenlogischen Operatoren.**
  - Die Literale sind als Variable aufzufassen, die genau einen von 2 Werten annehmen können
- **Eine Belegung einer Formel ist eine Zuweisung von Wahrheitswerten an die Literale derart, dass dieselben Literale immer denselben Wahrheitswert erhalten.**
- **Eine Formel heißt erfüllbar, wenn es eine Belegung gibt derart, dass die Formel wahr ist.**
  - Das Erfüllbarkeitsproblem ist in der Aussagenlogik immer lösbar, da man alle (endlich vielen) Belegungsmöglichkeiten der Variablen nur nacheinander auszuprobieren braucht.
  - Leider dauert das Lösen durch Ausprobieren sehr lange (exponentiell in der Anzahl der Variablen). Es ist bis heute kein effizienterer Algorithmus bekannt.

Das Problem ist NP-vollständig !

Aussagenlogik als Beschreibungssprache

Nach dieser Wiederholung wissen wir, wie das Diagnoseproblem als aussagenlogisches Problem formuliert werden kann:

Warum sollten wir das Diagnoseproblem als aussagenlogisches Problem formulieren ?

- ➔ Wegen der allgemeingültigen und international einheitlichen Ausdrucksmöglichkeit
- ➔ Um das Diagnoseproblem mit einem allgemeinen aussagenlogischen Inferenzmechanismus zu lösen

Das ist der klassische KI-Ansatz zur Realisierung wissensbasierter Systeme !

Logische Programmiersprachen

Es gibt allgemeine Algorithmen, die <u>jedes</u> aussagenlogisch formulierte Problem lösen.

➔ **Logisches Programmieren reduziert sich auf das Formulieren des Problems in einer logischen Beschreibungssprache.**

- Damit steht ein allgemeingültiges Problemlösungsverfahren zur Verfügung.

- Die Formulierung der Eingabe ist nur mit Übung zu bewältigen.
- Die Lösung kann sehr ineffizient sein.

Hilfstechnik zur Beschleunigung des Verfahrens:

➔ **Resolution**

$$(p \vee q) \wedge (r \vee \neg q) \Rightarrow (p \vee r)$$

Resolventenbildung

- Im ungünstigen Fall bringt das gegenüber Ausprobieren keine Verbesserung.
- Es gibt viele Probleme, die sich aussagenlogisch gar nicht formulieren lassen.

Bsp. für die mangelnde Ausdruckskraft der Aussagenlogik

In der Aussagenlogik haben alle Formeln nur zwei mögliche Werte für die Variablen.

Formeln mit Variablen, die einen unendlichen Wertebereich haben, sind also nicht enthalten.

- Anm.: Für solche Formeln versagt die Methode des Ausprobierens aller Möglichkeiten.

Bsp.: Finde $x,y \in \mathbf{R}$ mit folgenden Eigenschaften:

$$(2 < x < 4) \wedge (0 < y < 6) \wedge (x + y > 7) \wedge (x \cdot y < 10)$$

Kommen solche Probleme bei wissensbasierten Systemen überhaupt vor ?

Ja, sehr oft !

Wdh.: Prädikatenlogik

Die Prädikatenlogik (1. Stufe) erweitert die Aussagenlogik um folgende Elemente:

- **Prädikate**
  - Aussagen, die von Variablen abhängen
    (wenn es von k Variablen abhängt,
    dann heißt das Prädikat k-stellig)
- **Variable**
  - entsprechen den Literalen der Aussagenlogik,
    können aber beliebig viele Werte annehmen
- **Funktionen**
  - eindeutige Zuordnungen, die von Variablen abhängen
    (wenn sie von k Variablen abhängt,
    dann heißt die Funktion k-stellig)
  - 0-stellige Funktionen sind Konstante
- **Quantoren**
  - Existenzquantor (∃) und Allquantor (∀)
  - Quantoren werden nur auf Variablen angewendet (sonst nicht 1. Stufe)

Wdh.: Prädikatenlogische Formeln

- **Eine prädikatenlogische Formel ist eine Verknüpfung von endlich vielen Variablen, Funktionen und Prädikaten mit aussagenlogischen Operatoren oder Quantoren.**
  Die Quantoren dürfen sich bei Formeln 1. Stufe nur auf Variable beziehen.

Bsp.: Formel $\varphi = \forall x\,(\,R(\mathbf{y}, \mathbf{z}) \wedge \exists y\,(\neg P(\mathbf{y}, \mathbf{x}) \vee R(\mathbf{y}, \mathbf{z}))\,)$

Grüne Vorkommen von *y* und *z* sind **frei**.
Rote Vorkommen von *x, y* und *z* sind **gebunden**.

Geschlossene Formeln:

Formeln, die keine freien Variablen enthalten.

Offene Formeln:

Formeln, die keine gebundenen Variablen enthalten.

Wdh.: Prädikatenlogische Formeln

- **Eine Belegung einer Formel ist eine Zuweisung von *Werten aus festgelegten Definitionsbereichen an die freien Variablen* derart, dass dieselben Variablen immer denselben Wert erhalten.**

- **Eine Formel heißt erfüllbar, wenn es eine Belegung gibt derart, dass die Formel wahr ist.**

  - Das Erfüllbarkeitsproblem ist in der Prädikatenlogik nicht entscheidbar, d.h. kein Algorithmus kann jemals in der Lage sein, von jeder Formel zu entscheiden, ob sie erfüllbar ist oder nicht.

Das allgemeine Problem ist unlösbar !

Gibt es dennoch einen Ausweg ?

Ja, löse ein spezielleres Problem !

Widerspruchsfindung mittels Resolution

Idealziel: *geht nicht !*

Versuch, alle Folgerungen aus einer Menge prädikatenlogischer Formeln automatisch zu gewinnen

Realistischeres Ziel:

Versuch, möglichen Widerspruch aus einer Menge prädikatenlogischer Formeln aufzudecken

Resolutionsprinzip:

Generierung einer neuen Formel als Folgerung aus 2 gegebenen Formeln

Prinzip: Finde Literal c, der in den Formeln a ∨ c und b ∨ ¬c vorkommt.

Dann kann c **eleminiert** werden: (a ∨ c) ∧ (b ∨ ¬c) → (a ∨ b)

Die neue Formel heißt **Resolvente** der alten Formeln.

Durch eine solche Eliminierung können einzelne Literale isoliert werden:

Bsp.: (a ∨ c) ∧ ¬c → a Interpretation: a muss in der Formelsammlung gelten.

Wenn auf diese Weise auch die Negation isoliert wird, ergibt sich ein Widerspruch: ***Widerspruch!***

Bsp.: (¬a ∨ d) ∧ ¬d → ¬a Interpretation: ¬a muss in der Formelsammlung gelten.

Reduktion der Termvielfalt mittels Unifikation

Beispiel:

$\Phi = \{\neg P(x, f(y)), P(z, f(g(z)))\}$

Frage: Wieso ist diese Formelmenge widersprüchlich?

Antwort: Weil die beiden Atome $P(x, f(y))$ und $P(z, f(g(z)))$
durch geschickte Wahl von z und g identifiziert werden können.

Eine logische Programmiersprache braucht daher ***Unifikation***:

Ersetzung der Variablen durch Terme, so dass beide Atome gleich werden.

Substitution:

Die Ersetzung [*x*/*t*] angewendet auf φ bezeichnet diejenige Formel, die aus φ entsteht, wenn alle ***freien*** Vorkommen in φ von *x* durch Term *t* ersetzt werden.
Analog wird die Ersetzung [x1/t1, x2/t2, ..., xn/tn] gebildet.

Bezeichnung: σ = [x1/t1, x2/t2, ..., xn/tn] heißt ***Substitution***.
$\sigma\,\varphi$ *ist* die ***Anwendung von*** σ auf φ

Beispiel: Formel: φ = P (f (x), y)
Substitution: σ = [x/z, y/f (z)]
Anwendung: $\sigma\,\varphi$ = P (f (z), f (z))

Reduktion der Termvielfalt mittels Unifikation

Definition:

Eine Substitution σ heißt ***Unifikator*** für die Formeln α_1 und α_2, wenn gilt: $\sigma\alpha_1 = \sigma\alpha_2$.

Beispiel:

Unifikation der Atome $Q(f(x), v, b)$ und $Q(f(a), g(u), y)$
durch die Substitution $\sigma = [x/a, v/g(u), y/b]$

Satz (Existenz):

Für je zwei Ausdrücke gibt es, bis auf Variablenumbenennung, entweder einen eindeutigen allgemeinsten Unifikator oder die beiden Ausdrücke sind nicht unifizierbar.

Satz (Berechenbarkeit):

Es gibt einen Algorithmus, der für zwei beliebige Ausdrücke entweder die Nichtunifizierbarkeit beweist oder den allgemeinsten Unifikator berechnet.

Reduktion der Termvielfalt mittels Unifikation

Übungsbeispiele für Unifikation:

$P(x)$ und $Q(y)$

$P(x, y)$ und $P(z)$

$P(x, y)$ und $P(a, f(a))$

$P(x, y)$ und $P(f(z), g(z))$

$P(x, f(x, x), z, f(z, z))$ und $P(f(a, a), y, f(y, y), u)$

$P(x, f(y))$ und $P(z, f(g(z))$

$P(x, x)$ und $P(f(y), f(g(z))$

$P(x, f(x))$ und $P(y, y)$

$P(x, a)$ und $P(b, x)$

Logisches Programmieren

Das Prinzip der Programmiersprache PROLOG:

PROLOG versucht, mittels wiederholter und verschachtelter Anwendung von **Resolution** und **Unifikation** zu einer gegebenen Formelmenge einen Widerspruch zu finden.

Satz (Widerspruchsvollständigkeit):

Falls die Formelmenge widersprüchlich ist, kann man den Widerspruch immer finden.

Was fehlt ?

Satz (*Folgerung* der Folgerbarkeit aus der Widerspruchsaufdeckung):

Wer zu jeder Formelmenge jeden Widerspruch aufdecken kann,
kann zu jeder Formelmenge und zu jeder neuen ***daraus folgenden*** Formel beweisen,
dass die neue Formel aus der alten Formelmenge folgt.

Beweis ?

Was kann also Prolog auch noch ?

Logisches Programmieren

Wie macht man aus PROLOG eine <u>vollständige</u> Programmiersprache ?

Durch Beschränkung der Eingabe !

PROLOG akzeptiert nur Mengen von Formeln der Form:

$$p \wedge q \wedge \ldots \wedge r \rightarrow x$$

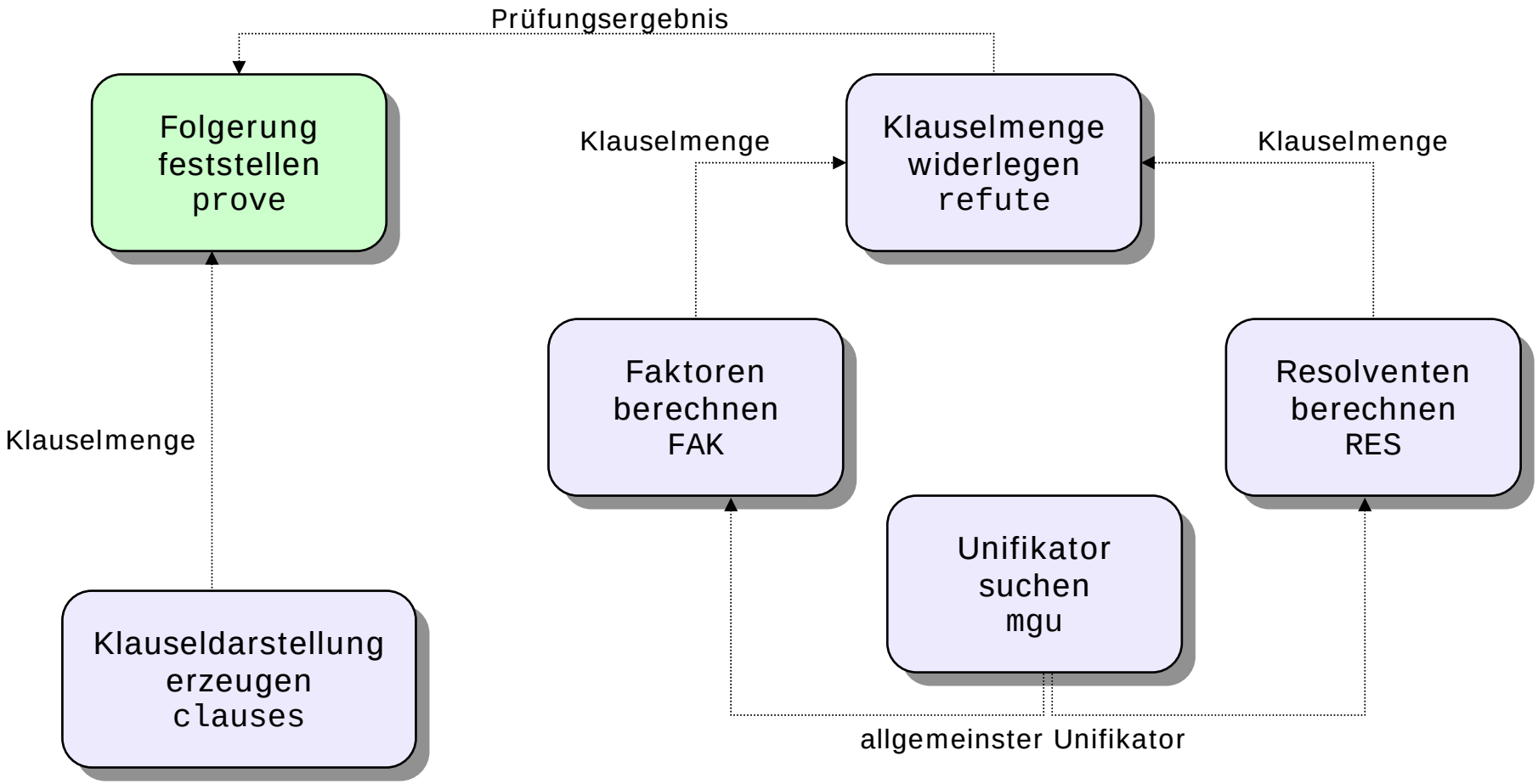
Regeln (Hornklauseln)

<u>Satz (Vollständigkeit der Resolution auf Hornklauseln):</u>

Für jede Menge von Hornklauseln und eine neue Hornklausel kann Prolog nach endlicher Zeit entscheiden, ob die neue Hornklausel aus der alten Menge folgt ***oder nicht***

Anmerkung: „Endliche Zeit“ kann „sehr lange“ heißen !

Implementierung eines Resolutionsbeweisers



Details in Seminarvortrag und Ausarbeitung von Daniel Dittmann, SS 2005, Nr. 8, (http://www.fh-wedel.de/~iw/Lehrveranstaltungen/SS2005/SeminarKI.html)