INF FAKULTÄT FÜR INFORMATIK

Intelligente Systeme

Regelbasierte Systeme

Prof. Dr. R. Kruse **C. Braune**

{kruse,cbraune}@iws.cs.uni-magdeburg.de

Institut für Wissens- und Sprachverarbeitung

Fakultät für Informatik

Otto-von-Guericke Universität Magdeburg

Was sind Regeln?

Regeln sind formalisierte Konditionalsätze der Form

Wenn A **dann** B.

mit der Bedeutung:

Wenn A wahr (erfüllt, bewiesen) ist,
dann schließe, dass auch B wahr ist.

A und B sind dabei Aussagen

A (Wenn-Teil): Prämisse, Antezedenz

B (Dann-Teil): Konklusion, Konsequenz

Begriffliches

Es gibt weitere Arten von Regeln, wie z.B. Produktionsregeln:

Wenn der Abstand zu klein ist, dann verringere die Geschwindigkeit.

Ist die Prämisse erfüllt, so wird die Aktion durchgeführt - "die Regel feuert"

Regel werden zum Beispiel bei der Warenkorb-Analyse in Form von Assoziationsregeln genutzt.

Häufige Schreibweise einer Regel $A \rightarrow B$

Beispiel: Assoziationsregel (Warenkorbanalyse) *Bier* $\rightarrow$ *Chips*

Regel beinhaltet Expertenwissen (aus Analyse)

Liefert wertvolle Hinweise (z.B. zur Gestaltung des Ladenaufbaus)

Praktischer Nutzwert

Sehr guter Kompromiss zw. Verständlichkeit der Wissensdarstellung und formalen Ansprüchen

Babylonische Tafeln regelten Alltagsleben der Menschen und benutzten *Wenn-dann*-Konstrukte

Menschen sind intuitiv mit Regeln vertraut

Expertenwissen lässt sich häufig sehr gut mit Regeln modellieren, da Regeln dem menschlichem Denken nahekommen

Festlegungen

Wünschenswert ist eine möglichst einfache syntaktische Form der Regeln (Effizienz der Bearbeitung, Übersichtlichkeit)

Häufig werden zwei Bedingunen gefordert:

- Verknüpfung $\vee$ (*oder*) darf nicht in Prämisse auftreten
- Konklusion soll nur aus *einem* Literal (positives oder negiertes Atom) bestehen

Falls Regeln beiden Bedingungen nicht genügen, dann u.U. Vereinfachung mittels logischer Äquivalenzen

Beispiel

Regel: *Wenn es morgen regnet oder schneit, gehen wir ins Kino oder bleiben zu Hause.*
Beide Bedingungen werden verletzt, deswegen werden aus der Regel folgende vier Regeln gewonnen:

Wenn es morgen regnet und wir nicht ins Kino gehen, dann bleiben wir zu Hause.

Wenn es morgen regnet und wir nicht zu Hause bleiben, dann gehen wir ins Kino.

Wenn es morgen schneit und wir nicht ins Kino gehen, dann bleiben wir zu Hause.

Wenn es morgen schneit und wir nicht zu Hause bleiben, dann gehen wir ins Kino.

Einer komplexen Regel entsprechen hier *4* vereinfachte Regeln.

Regelumformungen

In der klassischen Logik gilt:

$$A \rightarrow B \equiv \neg A \vee B$$

Durch Distributivgesetze und de Morgan'schen Regeln lässt sich folgende Darstellung erreichen:

Prämisse A ist Disjunktion von Konjunktionen K_i von Literalen

Konklusion B ist Konjunktion von Disjunktionen D_j von Literalen

Regelumformungen

Transformation von komplexeren Regeln in syntaktisch einfachere durch (ggf. wiederholte) Anwendung der folgenden Schritte:

Ersetze Regel

$$\textbf{if } K_1 \vee \ldots \vee K_n \textbf{ then } D_1 \wedge \ldots \wedge D_m$$

durch $n \cdot m$ Regeln

$$\textbf{if } K_i \textbf{ then } D_j,\ i \in \{1, \ldots, n\},\ j \in \{1, \ldots, m\}$$

Ersetze Regel

$$\textbf{if } K \textbf{ then } L_1 \vee \ldots \vee L_p$$

(wobei K Konjunktion von Literalen ist) durch p Regeln

$$\textbf{if } K \wedge (\bigwedge_{k \neq k_0} \neg L_k) \textbf{ then } L_{k_0}, k_0 \in \{1, \ldots, p\}$$

Beispiel: Geldautomat

R_1 :	**if**	Karte	=	gültig	**and**
		PIN	=	richtig	**and**
		Versuche	$\neq$	überschritten	**and**
		Betrag	$\leq$	Maximalbetrag	**and**
		Kontostand	=	ausreichend	
	then	Auszahlung	=	soll erfolgen	
R_2:	**if**	Versuche	=	überschritten	
	then	Kartenrückgabe	=	nein	

Äquivalenzregeln werden allerdings verletzt:

Auszahlung *genau dann, wenn* keine Voraussetzung verletzt

Karte einbehalten *genau dann, wenn* Anzahl der zulässigen Versuche überschritten

Beispiel: Geldautomat

Die Regelbasis muss um Gegenstücke zu R1 und R2 erweitert werden.

$\Rightarrow$ Erweiterung der Regelbasis um Gegenstücke zu R_1 und R_2:

R_1':	**if**	Auszahlung	$=$	soll erfolgen	
	then	Karte	$=$	gültig	**and**
		PIN	$=$	richtig	**and**
		Versuche	$\neq$	überschritten	$=$ **and**
		Betrag	$\leq$	Maximalbetrag	**and**
		Kontostand	$=$	ausreichend	
R_2':	**if**	Kartenrückgabe	$=$	nein	
	then	Versuche	$=$	überschritten	

Beispiel: Geldautomat

Führt zu logisch äquivalenten Regeln:

R_1'':	**if**	Karte	$=$	ungültig	**or**
		PIN	$=$	falsch	**or**
		Versuche	$=$	überschritten	**or**
		Betrag	$>$	Maximalbetrag	**or**
		Kontostand	$\neq$	nicht ausreichend	
	then	Auszahlung	$\neq$	soll erfolgen	
R_2'':	**if**	Versuche	$\neq$	überschritten	
	then	Kartenrückgabe	$=$	ja	

Wissensbasis eines regelbasierten Systems

Besteht aus *Objekten* und deren Beschreibungen durch endliche Mengen diskreter *Werte*

Regeln repräsentieren Zusammenhänge zwischen *Objekten* oder *Mengen von Objekten*

Objekte und Regeln bilden *abstraktes Wissen* der Wissensbasis

Bei Anwendung auf einen konkreten Fall kommt *fallspezifisches Wissen* hinzu:

- unmittelbare Beobachtungen
- abgeleitetes Wissen

Dieses wird auch *Evidenz* genannt um zu betonen, dass für ein Faktum Anhaltspunkte oder Beweise vorliegen

Beispiel: Geldautomat – abstraktes Wissen

Parameter	**mögliche Werte**
Karte	{gültig, ungültig}
PIN	{richtig, falsch}
Versuche	{überschritten, nicht überschritten}
Kontostand	{ausreichend, nicht ausreichend}
Betrag	{$\leq$ Maximalbetrag, $>$Maximalbetrag}
Auszahlung	{soll erfolgen, soll nicht erfolgen}
Kartenrückgabe	{ja, nein}

Beispiel: Geldautomat – fallspezifisches Wissen

Kunde tritt an Automaten heran und möchte Geld abheben
Er erfüllt alle Bedingungen, allerdings wünscht er eine Auszahlung, die nicht durch seinen Kontostand gedeckt ist
Fallspezifisches Wissen:

Karte	$=$	gültig
PIN	$=$	richtig
Versuche	$\neq$	überschritten
Betrag	$\leq$	Maximalbetrag
Kontostand	$=$	nicht ausreichend

Mit Hilfe der Wissensbasis kann man zeigen, dass die Auszahlung soll nicht erfolgen darf.

Inferenz im regelbasierten System

Grundlegende Inferenzregel: *modus ponens*

if A **then** B	(Regel)
A wahr	(Faktum)
B wahr	(Schlussfolgerung)

Im Geldautomatenbeispiel:

if Kontostand = nicht ausreichend **then** Auszahlung = soll nicht erfolgen
Kontostand = nicht ausreichend wahr
Auszahlung = soll nicht erfolgen wahr

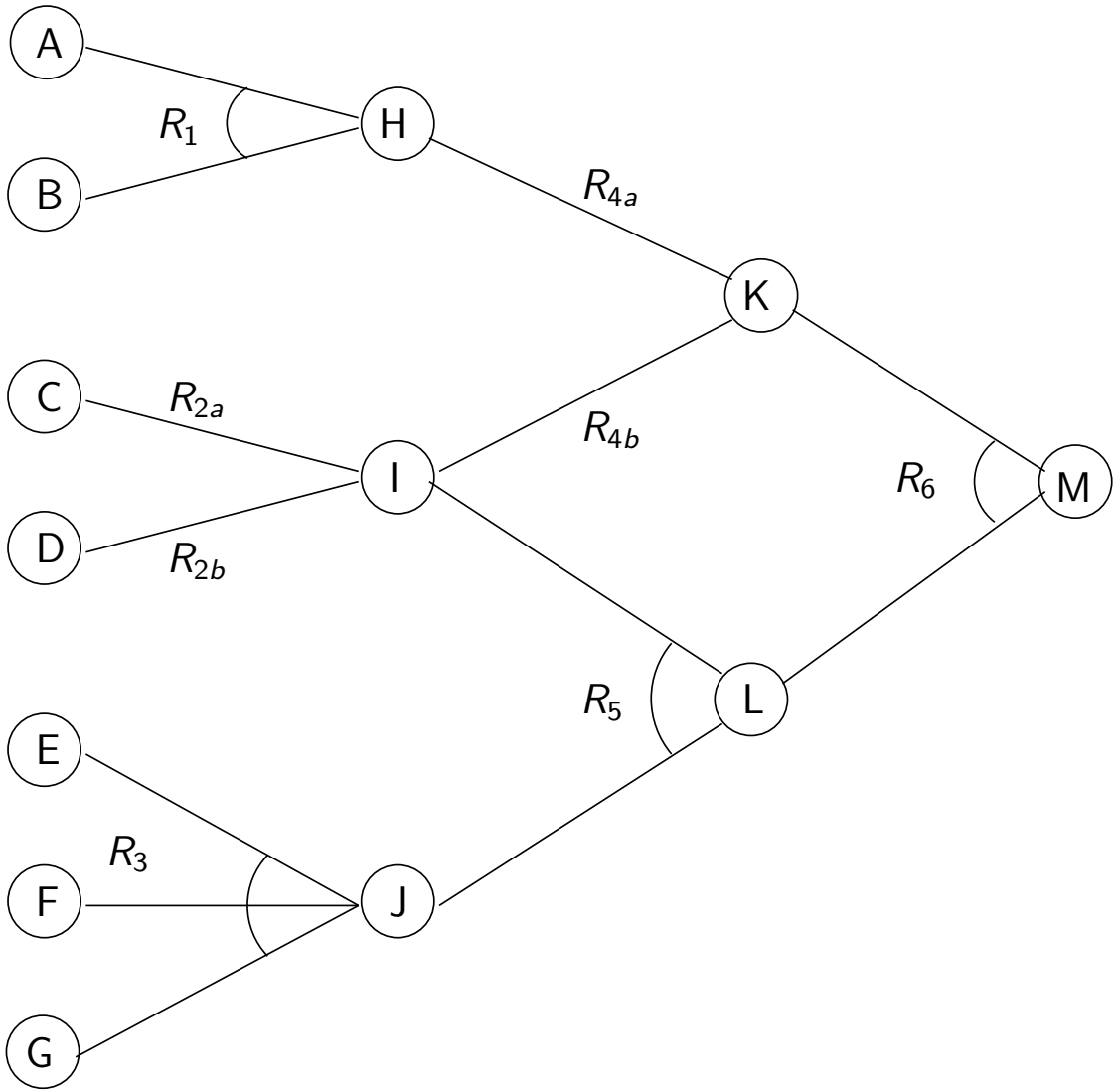
Wissensbasis: Regelnetzwerk

Objekte: A, B, C, D, E, F, G, H, I, J, K, L, M

Regeln:

$$
\begin{array}{lllll}
R_1: & \text{if} & A \wedge B & \text{then} & H \\
R_2: & \text{if} & C \vee D & \text{then} & I \\
R_{2a}: & \text{if} & C & \text{then} & I \\
R_{2b}: & \text{if} & D & \text{then} & I \\
R_3: & \text{if} & E \wedge F \wedge G & \text{then} & J \\
R_4: & \text{if} & H \vee I & \text{then} & K \\
R_{4a}: & \text{if} & H & \text{then} & K \\
R_{4b}: & \text{if} & I & \text{then} & K \\
R_5: & \text{if} & I \wedge J & \text{then} & L \\
R_6: & \text{if} & K \wedge L & \text{then} & M
\end{array}
$$

Wissensbasis: Regelnetzwerk



Datengetriebene Inferenz

Alternative Bezeichnung: Vorwärtsverkettung

Fallspezifisches Wissen als Ausgangspunkt für Inferenzprozess

Aus erfüllten Prämissen wird auf Wahrheit der Konklusion geschlossen

Abgeleitete Fakten gehen erneut in Wissensbasis ein

Das Verfahren endet, wenn keine neuen Fakten mehr generiert werden können

Datengetriebene Inferenz: ForwardChain

Eingabe: Wissensbasis RB (Objekte, Regeln), Menge $\mathcal{F}$ von Fakten
Ausgabe: Menge der gefolgerten Fakten

1: Sei $\mathcal{F}$ Menge der gegebenen (evidentiellen) Fakten
2: **for each** Regel **if** A **then** B aus RB $\{$
3: Ist A erfüllt, so schließe auf B
4: $\mathcal{F} := \mathcal{F} \cup \{B\}$
5: $\}$
6: Gehe zu Schritt 2, bis $\mathcal{F}$ nicht mehr vergrößert werden kann

Datengetriebene Inferenz: Regelnetzwerk

Beispiel:

Gegeben seien Fakten $\mathcal{F} = \{H, C, E, F, G\}$

Damit feuern Regeln R_{2a}, R_{4a} und R_3

Somit vergrößert sich $\mathcal{F}$ zu $\mathcal{F} := \{H, C, E, F, G\} \cup \{I, J, K\}$

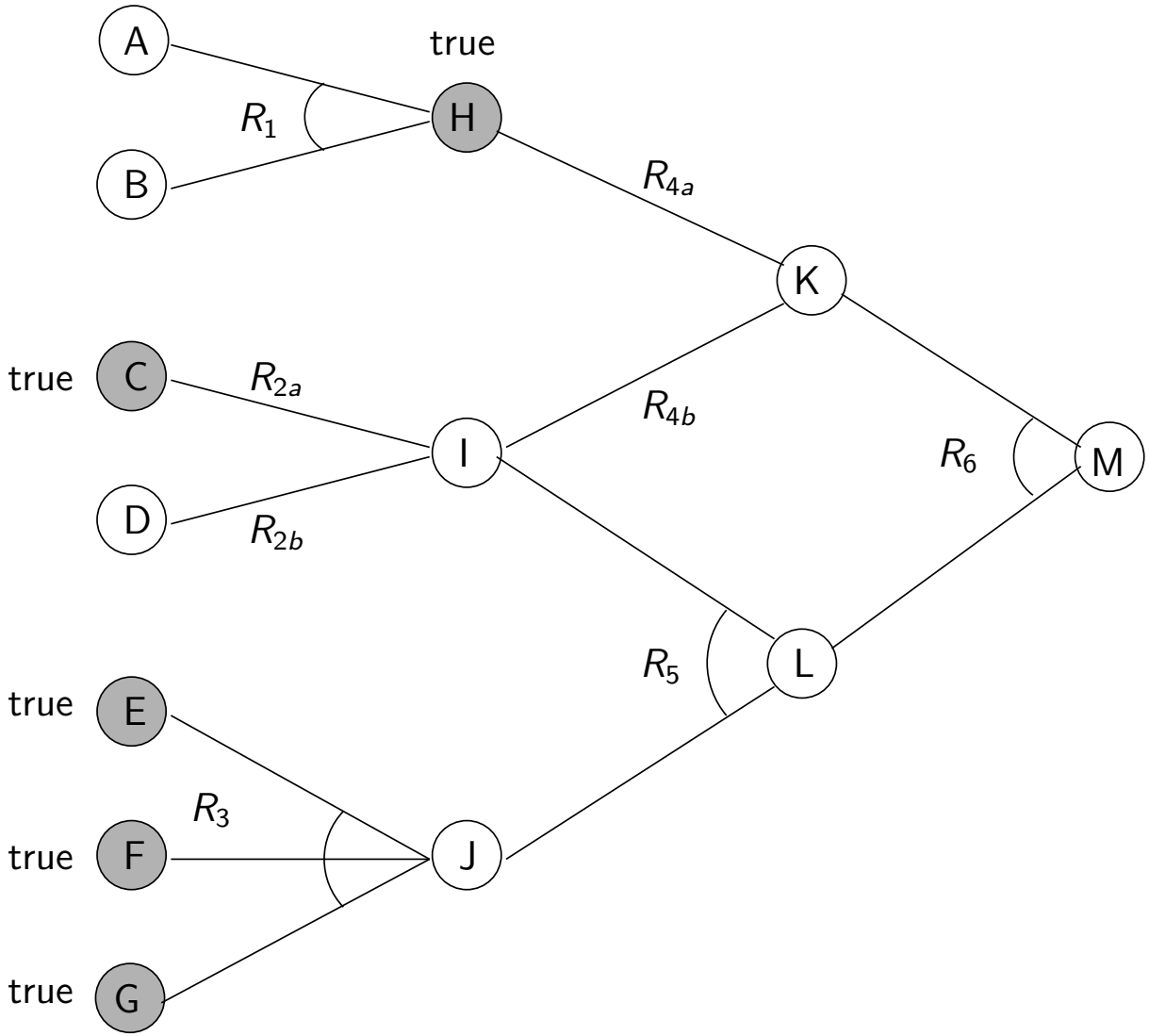
In weiterem Durchlauf feuern nun R_{4b} und R_5

$\mathcal{F} := \{H, C, E, F, GI, J, K\} \cup \{L\}$

Somit feuert nun R_6

Endgültige Faktenmenge $\mathcal{F} := \{H, C, E, F, G, IJ, K, L\} \cup \{M\}$

Datengetriebene Inferenz: Regelnetzwerk



Zielorientierte Inferenz

Alternative Bezeichnung: Rückwärtsverkettung

Zielobjekt Z als Ausgangspunkt

Durchsuchen der Regelbasis nach Regeln, die Z in Konklusion enthalten

Objekte der Prämissen werden zu Zwischenzielen

Zielorientierte Inferenz: BackChain

Eingabe: Regelbasis RB, (evidentielle) Faktenmenge $\mathcal{F}$, Liste von Zielen (atomaren Anfragen) $[q_1, \ldots, q_n]$

Ausgabe: *yes*, falls alle q_i ableitbar sind, sonst *no*

```
 1: if n = 0 {
 2:     return yes
 3: }
 4: if q_1 ∈ F {
 5:     BackChain([q_2, ..., q_n])
 6: } else {
 7:     for each Regel p_1 ∧ ... ∧ p_m → q aus RB mit q_1 = q {
 8:         if BackChain([p_1, ..., p_m, q_2, ..., q_n]) = yes {
 9:             return yes
10:         }
11:     }
12: }
13: return no
```

Zielorientierte Inferenz

Wissensbasis enthalte (zweiwertigen) Objekte O_1, O_2, O_3 und Regeln:

- Regel 1: **if** O_1 **then** O_2
- Regel 2: **if** O_2 **then** O_3

Frage: Zustand des Zielobjektes O_3

- O_3 ist wahr, wenn O_2 wahr ist
- O_2 ist wahr, wenn O_1 wahr ist
- Informationen über O_1 benötigt

Das Problem der Widersprüchlichkeit

Falls Negation in Fakten oder Konklusion von Regeln erlaubt:
Regelbasis kann zu widersprüchlichen Ableitungen führen

In der Praxis häufig auftretendes Problem

Experten benutzen zur Formulierung der Regelbasis häufig:

- Implizite, unausgesprochene Annahmen,
- Oder übersehen, dass Ausnahmen zu Regeln auftreten können

Beispiel-Grammatik: **if** V **then** F, **if** $V \wedge P$ **then** $\neg F$

Instantiiere V und P mit *wahr* $\Rightarrow$ Schlüsse F und $\neg F$

Veranschaulichung: V – Vögel, P – Pinguine, F – Fliegen können

Problem der Widersprüchlichkeit

Regelbasen können auf zwei Arten widersprüchlich sein::

- es gibt keine Belegung der Objekte mit Werten, so dass alle Regeln erfüllt sind
- Regelbasis führt zu widersprüchlichen Ableitungen

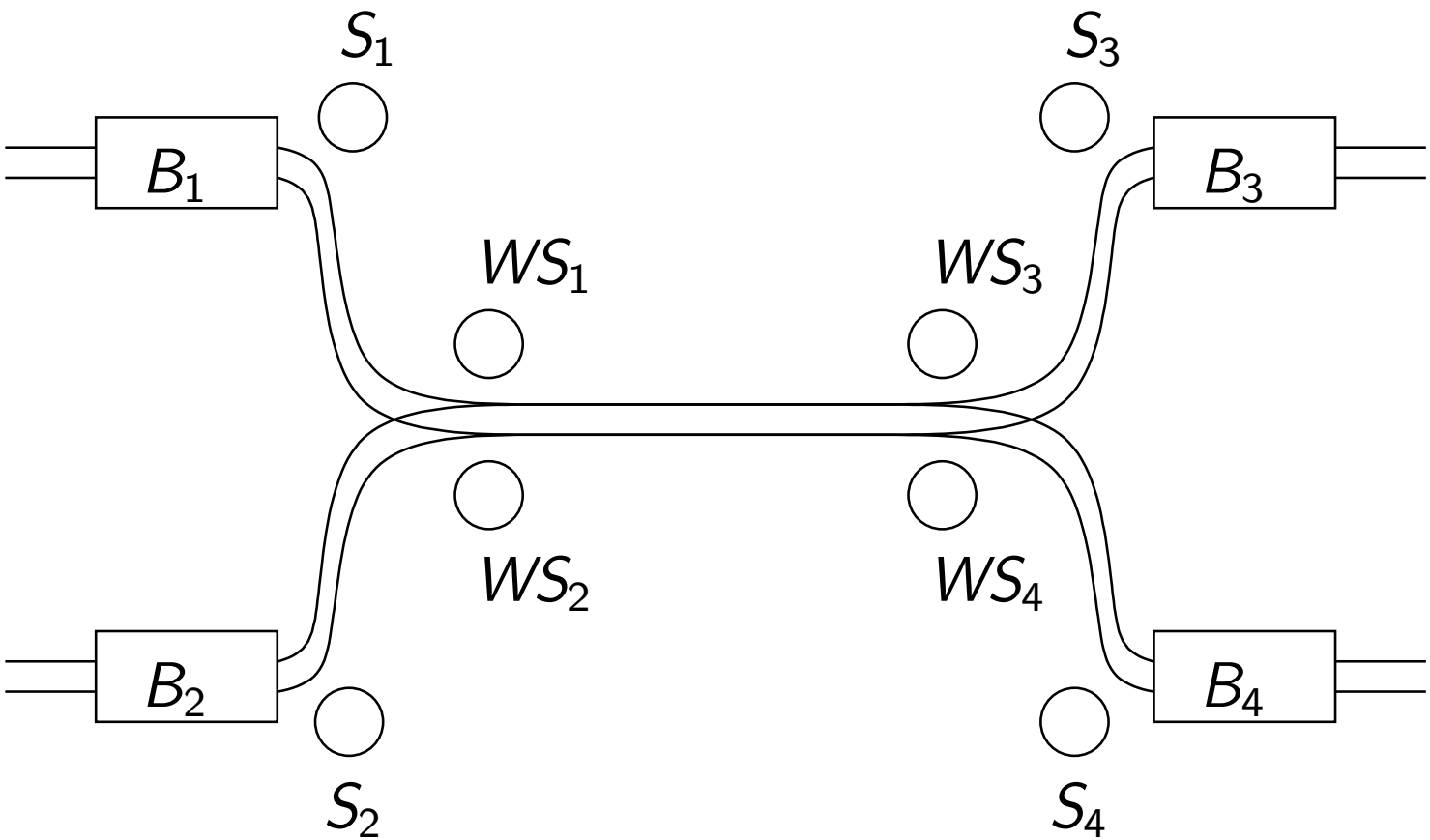
Beispiel:

- $\{\textbf{if } V \textbf{ then } F, \textbf{if } V \wedge P \textbf{ then } \neg F\}$ nicht inkonsistent
- $\{\textbf{if } V \textbf{ then } F, \textbf{if } V \wedge P \textbf{ then } \neg F,\ V \wedge P\}$ ist inkonsistent

Konsistenzüberprüfung: wichtige Warnfunktion

Dient der Verbesserung der Regelbasis

Beispiel: Signalsteuerung im Eisenbahnverkehr



Beispiel: Signalsteuerung im Eisenbahnverkehr

Objekte	mögliche Werte
S_1, S_2, S_3, S_4	{rot,grün}
WS_1, WS_2, WS_3, WS_4	{rot,grün}
B_1, B_2, B_3, B_4	{frei, belegt}

S_i – Bahnhofssignale

WS_i – Weichensignale

B_i – Bahnhöfe

Alle gezeigten Strecken sind eingleisig

Gefahrloser Bahnverkehr soll durch regelbasiertes System gewährleistet werden

Beispiel: Signalsteuerung im Eisenbahnverkehr

Vermeidung von Zusammenstößen auf Strecke — immer nur höchstens 1 der S_i auf grün, Rest rot:

R1:	**if**	S_1	=	grün	**then**	S_2	=	rot
R2:	**if**	S_1	=	grün	**then**	S_3	=	rot
R3:	**if**	S_1	=	grün	**then**	S_4	=	rot
R4:	**if**	S_2	=	grün	**then**	S_1	=	rot
R5:	**if**	S_2	=	grün	**then**	S_3	=	rot
R6:	**if**	S_2	=	grün	**then**	S_4	=	rot
R7:	**if**	S_3	=	grün	**then**	S_1	=	rot
R8:	**if**	S_3	=	grün	**then**	S_2	=	rot
R9:	**if**	S_3	=	grün	**then**	S_4	=	rot
R10:	**if**	S_4	=	grün	**then**	S_1	=	rot
R11:	**if**	S_4	=	grün	**then**	S_2	=	rot
R12:	**if**	S_4	=	grün	**then**	S_3	=	rot

Beispiel: Signalsteuerung im Eisenbahnverkehr

Kein Zug in einem belegten Bahnhof:

R13:	**if**	B_1	$=$	belegt	**then**	WS_1	$=$	rot
R14:	**if**	B_2	$=$	belegt	**then**	WS_2	$=$	rot
R15:	**if**	B_3	$=$	belegt	**then**	WS_3	$=$	rot
R16:	**if**	B_4	$=$	belegt	**then**	WS_4	$=$	rot

Mittlerer Teil der Strecke darf nicht durch wartende Züge blockiert werden:

R17:	**if**	$WS_1 = rot$	$\wedge$	$WS_2 = rot$	**then**	S_3	$=$	rot
R18:	**if**	$WS_1 = rot$	$\wedge$	$WS_2 = rot$	**then**	S_4	$=$	rot
R19:	**if**	$WS_3 = rot$	$\wedge$	$WS_4 = rot$	**then**	S_1	$=$	rot
R20:	**if**	$WS_3 = rot$	$\wedge$	$WS_4 = rot$	**then**	S_2	$=$	rot

Beispiel: Signalsteuerung im Eisenbahnverkehr

WS_1/WS_2 bzw. WS_3/WS_4 sind Weichensignale, d.h. es kann höchstens eine der beiden zugehörigen Strecken freigegeben werden:

R21:	**if**	WS_1	=	grün	**then**	WS_2	=	rot
R22:	**if**	WS_2	=	grün	**then**	WS_1	=	rot
R23:	**if**	WS_3	=	grün	**then**	WS_4	=	rot
R24:	**if**	WS_4	=	grün	**then**	WS_3	=	rot

Beispiel: Signalsteuerung im Eisenbahnverkehr

Beispielsituation

B_1 und B_3 seien belegt:

$\mathcal{F}_1$:	B_1	$=$	belegt
	B_3	$=$	belegt

Ableitbare Aussagen:

$\mathcal{C}_1$:	WS_1	$=$	rot	(R13)
	WS_3	$=$	rot	(R15)

Beispiel: Signalsteuerung im Eisenbahnverkehr

Zug fährt in Bahnhof B_2 ein:

$\mathcal{F}_2$:	B_1	$=$	belegt
	B_3	$=$	belegt
	B_2	$=$	belegt

Ableitbare Aussagen:

$\mathcal{C}_1$:	WS_1	$=$	rot	(R13)
	WS_3	$=$	rot	(R15)
	WS_2	$=$	rot	(R14)
	S_3	$=$	rot	(R17)
	S_4	$=$	rot	(R18)

Beispiel: Signalsteuerung im Eisenbahnverkehr

Fahrt für Zug in Bahnhof B_1 wird freigegeben, d.h. S_1 wird auf grün gestellt:

$\mathcal{F}_2$:	B_1	$=$	belegt
	B_3	$=$	belegt
	B_2	$=$	belegt
	S_1	$=$	grün

Ableitbare Aussagen:

$\mathcal{C}_1$:	WS_1	$=$	rot	(R13)
	WS_3	$=$	rot	(R15)
	WS_2	$=$	rot	(R14)
	S_3	$=$	rot	(R17 und R2)
	S_4	$=$	rot	(R18 und R3)
	S_2	$=$	rot	(R1)

Nicht-Monotones Schließen

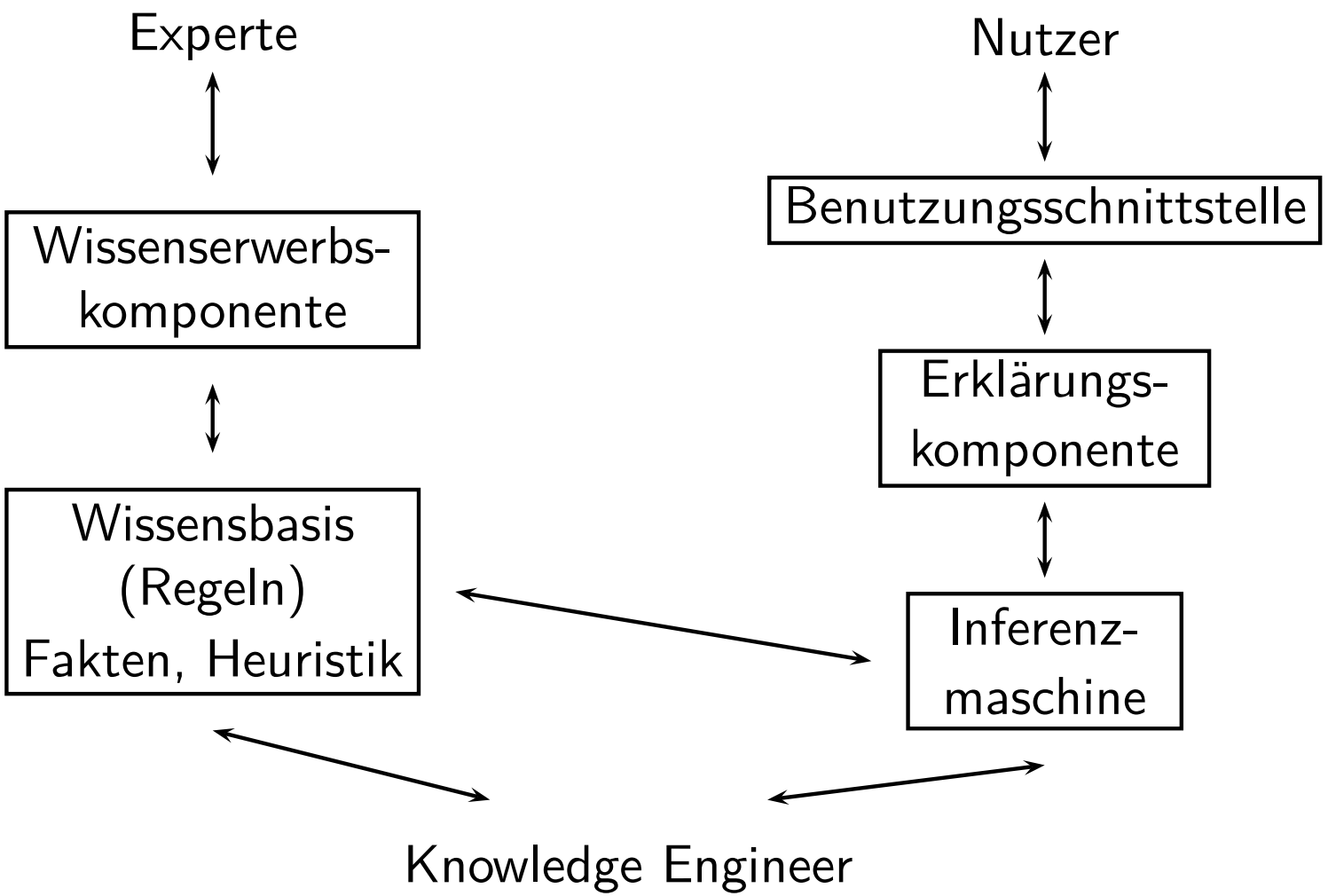
Die Menge der Schlussfolgerungen wächst *monoton* mit der Größe der Faktenmenge

Allgemeingültigkeit der Regeln kann nicht immer garantiert werden

Es kann also vorkommen, dass bereits gezogene Schlussfolgerungen revidiert werden müssen

Dies führt zu *nichtmonotonem* Ableitungsverhalten

Regelbasierte Systeme in realen Anwendungen



Beispiel: „Darlehensvergabe“

OK	Darlehen sollte bewilligt werden.
COLLAT	Sicherheit für Darlehen ist genügend.
PYMT	Antragsteller kann Darlehenszahlungen tilgen.
REP	Antragsteller hat gute finanzielle Reputation.
APP	Beurteilung der Sicherheit ist ausreichend größer als Größe des Darlehens.
RATING	Antragsteller hat gute Bonität.
INC	Einkommen des Antragstellers ist größer als seine Ausgaben.
BAL	Antragsteller hat ausgezeichnete Bilanz.

Beispiel: „Darlehensvergabe“

Regeln zur Entscheidungsfindung:

$\text{COLLAT} \wedge \text{PYMT} \wedge \text{REP} \Rightarrow \text{OK}$

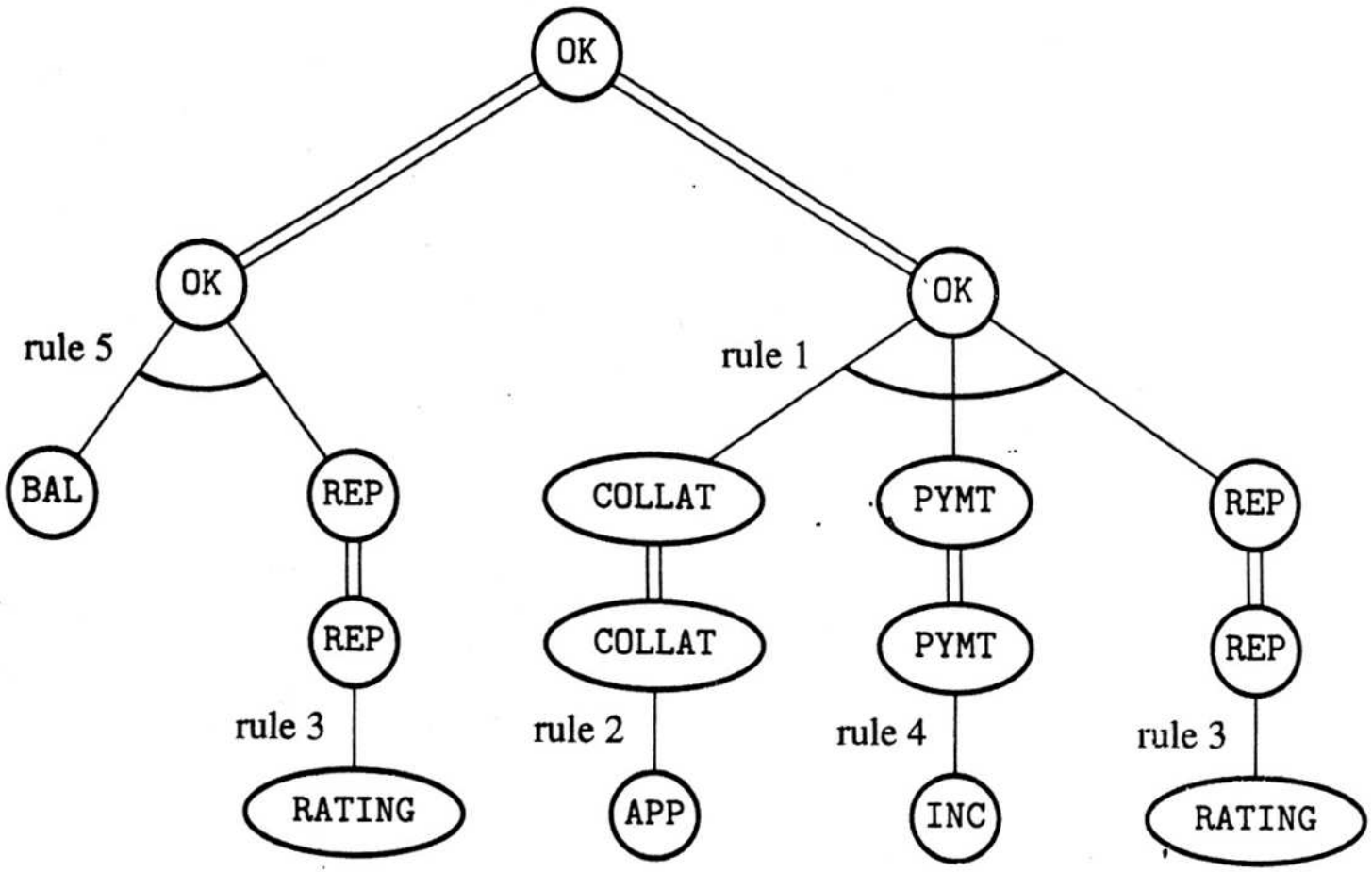
$\text{APP} \Rightarrow \text{COLLAT}$

$\text{RATING} \Rightarrow \text{REP}$

$\text{INC} \Rightarrow \text{PYMT}$

$\text{BAL} \wedge \text{REP} \Rightarrow \text{OK}$

Entscheidungsbaum (Und/Oder-Baum)



Modi der Verwendung

Anfragemodus:

Fakten werden aus Datenbank geholt oder Benutzer wird gefragt

Erklärungsmodus:

Das System beantwortet Fragen, wie die Entscheidung gefällt wurde

Beispiele:

Nutzer: „Warum glaubst du, dass *das Darlehen bewilligt werden soll*?“

System: „Weil *die Sicherheit für das Darlehen genügend ist*, und *der Antragsteller die Darlehenszahlungen tilgen kann*, und *der Antragsteller eine gute finanzielle Reputation hat.*“

Anfrage- und Erklärungsmodus

Nutzer: „Warum glaubst du, dass *der Antragsteller eine gute finanzielle Reputation hat*?“

System: „Weil die Schufa gesagt hat, dass *der Antragsteller eine gute Bonität hat*.“

Nutzer: „Warum hast du mich gefragt, ob *das Einkommen des Antragstellers größer ist als seine Ausgaben* oder nicht?“

System: „Weil ich versucht habe, herauszufinden ob *der Antragsteller die Darlehenszahlungen tilgen kann* oder nicht.“

Nutzer: „Warum versuchst du herauszufinden, ob *der Antragsteller die Darlehenszahlungen tilgen kann* oder nicht?“

System: „Weil es eines der Kriterien zum Herausfinden ob *das Darlehen bewilligt werden sollte* oder nicht.“

Nutzer: „Warum versuchst du herauszufinden, ob *das Darlehen*

XCON/R1

Beispiel für ein Konfigurationssystem

XCON/R1: Werkzeug für Konfiguration von *DEC Vax-Computern*

1980 entwickelt und löst folgende Aufgaben:

- Identifikation fehlender Komponenten,
- Platzierung der Komponenten bezüglich Bussen, Schnittstellen und Gehäusen

Implementiert in *OPS5* (vorwärtsverkettetes regelbas. System)

Eine der ersten erfolgreichen kommerziellen Anwendungen von regelbasierten Expertensystemen

Hatte großen Einfluss auf industrielles Interesse an Expertensystemen

Wurde ständig weiterentwickelt: heute ca. 10.000 Regeln

Löst routinemäßig Aufgaben mit 100–200 Komponenten

Beispiel einer XCON/R1 Regel

IF

the most current active context is distributing massbus devices, and there is a single-port disk drive that has not been assigned to a massbus, and there are no unassigned dual-port disk drives, and the number of devices that each massbus should support is known, and there is a massbus that has been assigned at least one disk drive and that should support additional disk drives, and the type of cable needed to connect the disk drive to the previous device on the massbus is known

THEN

assign the disk drive to the massbus.

Schlußfolgerungsverfahren

Modus Ponens

$$\frac{A, \quad A \rightarrow B}{B}$$

Resolution:

$$\frac{A \vee B, \quad \neg B \vee C}{A \vee C}$$

Die abgeleitete Klausel $A \vee C$ nennt man *Resolvente*

Verallgemeinerte Resolution

Mit Literalen $A_1, \ldots, A_m, B, C_1, \ldots, C_n$ erhält man

$$\frac{(A_1 \vee \ldots \vee A_m \vee B), \quad (\neg B \vee C_1 \vee \ldots \vee C_n)}{(A_1 \vee \ldots \vee A_m \vee C_1 \ldots \vee \vee C_n)}$$

Die Literale B und $\neg B$ heißen *komplementär*

Widerspruchsbeweis

Zu zeigen: Aus Wissensbasis KB folgt Q

Lösung: Durch Widerspruch von $KB \wedge \neg Q$

Ziel: Erzeugen zweier komplementärer Klauseln, die zur leeren Klauselmenge als Resolvente führen

Klausel in KNF enthält positive und negierte Literale

Repräsentation als

$$\neg A_1 \vee \ldots \vee \neg A_m \vee B_1 \vee \ldots \vee B_n$$

mit Variablen $A_1, \ldots, A_m$ und $B_1, \ldots, B_n$

Transformation in

$$A_1 \wedge \ldots \wedge A_m \rightarrow B_1 \vee \ldots \vee B_n$$

Hornklauseln

Hornklauseln = Klauseln mit maximal einem positiven Literal

$$\neg A_1 \vee \ldots \vee \neg A_m \vee B$$

bzw.

$$A_1 \wedge \ldots \wedge A_m \rightarrow B$$

Klausel mit einem positiven Literal ist ein **Fakt**

In Klauseln mit negativem und einem positivem Literal heißt das positive literal **Kopf**

Beispiel: Ski fahren

$$(\text{schönes Wetter})_1$$
$$(\text{Schneefall})_2$$
$$(\text{Schneefall} \rightarrow \text{Schnee})_3$$
$$(\text{schönes Wetter} \wedge \text{Schnee} \rightarrow \text{Ski fahren})_4$$

Wenn wir wissen wollen, ob Ski fahren gilt, dann Modus ponens nutzen:

$$\frac{A_1 \wedge \ldots \wedge A_m, \quad A_1 \wedge \ldots \wedge A_m \rightarrow B}{B}$$

$MP(I_1, \ldots, I_k)$ sei Anwendung des Modus ponens auf Klauseln I_1 bis I_k

Beispiel: Ski fahren

$$MP(2,3) : (\text{Schnee})_5$$
$$MP(1,5,4) : (\text{Ski fahren})_6$$

Mit Modus ponens: komplette Berechnung für Formeln, die aus Hornklauseln bestehen

Für große Wissensbasen für MP zu vielen unnötigen Formeln (wenn mit falscher Formel begonnen wird)

Darum: Berechnung nutzen, die mit Anfrage anfängt und rückwärts arbeitet bis Fakten erreicht sind

Backward chaining

Backward Chaining für Hornklauseln

SLD-Resolution wird genutzt

SLD: selektive regelgetriebene lineare Resolution für definite Klauseln

Beispiel:

$$(\text{Schönes Wetter})_1$$
$$(\text{Schneefall})_2$$
$$(\text{Schneefall} \rightarrow \text{Schnee})_3$$
$$(\text{Schönes Wetter} \wedge \text{Schnee} \rightarrow \text{Ski fahren})_4$$
$$(\text{Ski fahren} \rightarrow f)_5$$

Beispiel: Ski fahren

$$(\text{Schönes Wetter})_1$$
$$(\text{Schneefall})_2$$
$$(\text{Schneefall} \rightarrow \text{Schnee})_3$$
$$(\text{Schönes Wetter} \wedge \text{Schnee} \rightarrow \text{Ski fahren})_4$$
$$(\text{Ski fahren} \rightarrow f)_5$$

$$\text{Res}(5,4) : (\text{Schönes Wetter} \wedge \text{Schnee} \rightarrow f)_6$$
$$\text{Res}(6,1) : (\text{Schnee} \rightarrow f)_7$$
$$\text{Res}(7,3) : (\text{Schneefall} \rightarrow f)_8$$
$$\text{Res}(8,2) : ()_{10}$$

„lineare Resolution“: große Reduktion des Suchraums

Fest Abarbeitungsreihenfolge

SLD spielt wichtige Rolle in PROLOG

Beispiel: Implementierung über Prolog

„Roboterarm bewegt sich, wenn Batterie i.O. ist und Gewicht nicht zu schwer ist“

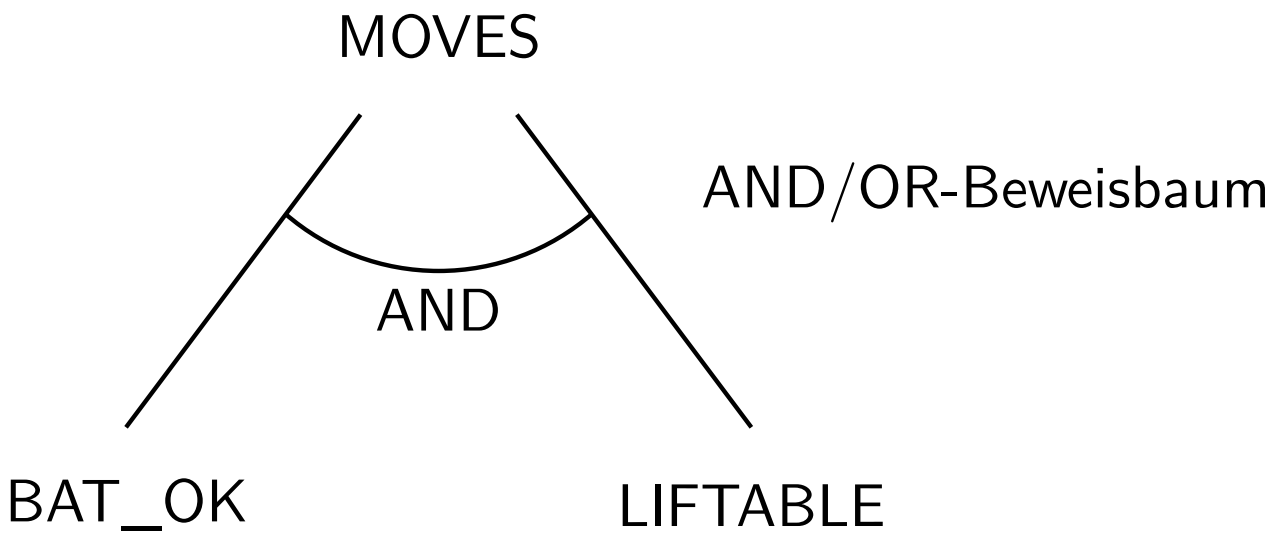
Programm entspricht der Aussage:

$$(\text{BAT\_OK} \wedge \text{LIFTABLE} \wedge (\text{BAT\_OK} \wedge \text{LIFTABLE} \Rightarrow \text{MOVES})) \Rightarrow \text{MOVES}$$

```
:- MOVES                          Ziel (Anfrage)
BAT_OK :-                         Fakten
LIFTABLE :-
MOVES :- BAT_OK, LIFTABLE         Regeln
```

Prolog beweist Aussage mit „depth-first“ und „backward chaining“

Beispiel: Schlussfolgern mit Hornklauseln



Schlussfolgern mit Hornklauseln

Ist Bauklotz A über C, wenn A auf B und B auf C?
Prolog:

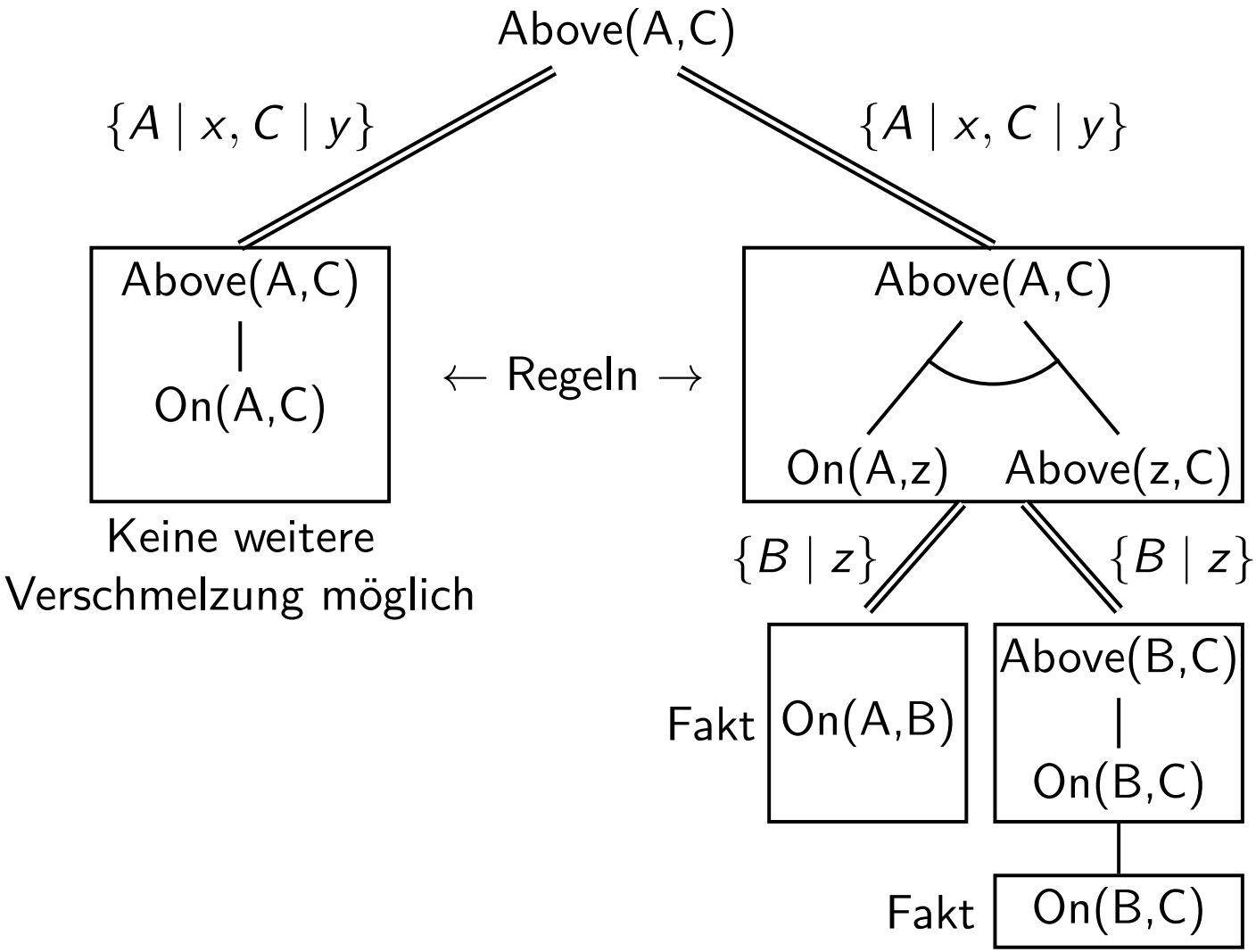
```
:- Above(A,C)
On(A,B) :-
On(B,C) :-
Above(x,y) :- On(x,y)
Above(x,y) :- On(x,z), Above(z,y)
```

PK1:

$$(\forall x, y, z)\ \mathrm{On}(x, y) \Rightarrow \mathrm{Above}(x, y)$$

$$(\forall x, y)\ (\exists z)\ \mathrm{On}(x, z) \wedge \mathrm{Above}(z, y) \Rightarrow \mathrm{Above}(x, y)$$

Schlussfolgern mit Hornklauseln



Vergleich zur klassischen Programmierung

Eigenschaft	klass. Programmierung	Expertensystem
gesteuert von...	Anordnung der Aussagen	Inferenzmaschine
Regelung und Daten	implizierte Integration	explizite Separation
Lösung durch...	Algorithmus	Regeln
Eingabe	als richtig angenommen	unvollständig, falsch
unerwartete Eingabe	schwer händelbar	sehr zugänglich
Ausgabe	immer richtig	variiert mit dem Problem
Erklärung	generell sequentiell	opportunistische Regeln
Programmentwurf	strukturiert	kaum oder keine Struktur
Modifizierbarkeit	schwer	möglich
Erweiterung	in großen Sprüngen	inkrementell

Vergleich zur klassischen Programmierung

Algorithmen + Datenstrukturen = Programme

Wissen + Inferenz = wissensbasiertes System

Pyramide des Wissens