4. Regelbasierte Systeme

- Eine weitverbreitete Form der Wissensrepräsentation ist die Formulierung von Regeln.
- Regeln sind formalisierte Konditionalsätze der Form:

  Wenn (`if`) F dann (`then`) G.

- Bedeutung:

Wenn	F wahr ist,
dann	schließe, daß auch G wahr ist.

- Dabei sind in einer logischen Interpretation F und G Aussagen oder PL1-Formeln.

Beispiele für Regeln:

Wenn der aktuelle Buchstabe eines Wortes *q* ist und das Wort keine Abkürzung ist,
dann ist der nächste Buchstabe ein *u*.

Wenn die Motortemperatur zu hoch ist,
dann prüfe die Kühlflüssigkeit.

- Die Formel im “Wenn”-Teil einer Regel wird als *Prämisse* oder *Antendenz* bezeichnet.
- Bezeichnung innerhalb regelbasiertes Systeme: left-hand side (LHS)
- Die Formel im “Dann”-Teil heißt *Konklusion* oder *Konsequenz*.
- Bezeichnung innerhalb regelbasiertes Systeme: right-hand side (RHS)
- Ist die Prämisse einer Regel erfüllt, so sagt man, daß *die Regel feuert*.

Die beiden Regeln des vorangegangenen Beispiels weisen Unterschiede auf:

- Die erste Regel entspricht einer logischen Implikation.
- Die Konklusion der zweiten Regel ist dagegen eine Aktion, die ausgeführt wird, wenn die Prämisse erfüllt ist.

Regeln der zweiten Art werden auch *Produktionsregeln* oder *Aktionsregeln* genannt.

Solche Regeln werden gerne in Produktionssystemen zur Steuerung eingesetzt.

Vorteile der Wissensdarstellung in Form von Regeln:

- Regeln stellen einen guten Kompromiß zwischen Verständlichkeit der Wissensdarstellung und formalen Ansprüchen dar.
- Konditionalsätze werden von Menschen seit langer Zeit benutzt, um Handlungsanweisungen oder Prognosen auszudrücken.
- Regeln sind dem Benutzer daher hinreichend vertraut.
- Ein großer Teil von Expertenwissen läßt sich üblicherweise in Regelform ausdrücken.

☞ Regeln haben eine große Nähe zum menschlichen Denken.

Syntaktischer Aufbau und Umformung von Regeln

- Man legt i.d.R. Wert auf eine möglichst einfache syntaktische Form der Regeln.
- Dies ermöglicht eine effizientere Abarbeitung der Regeln und
- es verbessert die Übersichtlichkeit der Wirkung einzelner Regeln.

Üblicherweise stellt man folgende Bedingungen an die Form der Regeln:

$\vee$ (oder) darf nicht in der Prämisse einer Regel auftreten.
Die Konklusion einer Regel sollte nur aus einem Literal bestehen.

- Regeln, die diesen Bedingungen nicht genügen, müssen umgeformt werden.
- Hierbei kommt die klassische Logik mittels der Äquivalenz zum Einsatz.

Beispiel 4.1.

> Wenn es morgen regnet oder schneit, gehen wir ins Kino oder bleiben zu Hause.

Diese Regel läßt sich äquivalent durch die folgenden vier Regeln ausdücken:

1. Wenn es morgen regnet und wir nicht ins Kino gehen, dann bleiben wir zu Hause.
2. Wenn es morgen regnet und wir nicht zu Hause bleiben, dann gehen wir ins Kino.
3. Wenn es morgen schneit und wir nicht ins Kino gehen, dann bleiben wir zu Hause.
4. Wenn es morgen schneit und wir nicht zu Hause bleiben, dann gehen wir ins Kino.

Nun erfüllt jede Regel die gestellten Anforderungen. Die Zahl der Regeln hat sich allerdings erhöht.

Durch folgende Umformungen können wir jede Regel in eine Menge von Regeln der gewünschten Form umwandeln:

1. Bringe die Prämisse in DNF und die Konklusion in KNF.
2. Ersetze die Regel

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

   durch $n \cdot m$ Regeln: $\texttt{if } K_i \texttt{ then } D_j$
3. Ersetze die Regel

$$\texttt{if } K \texttt{ then } L_1 \vee \ldots \vee L_k$$

   durch k Regeln

$$\texttt{if} K \wedge (\bigwedge_{i \neq i_0} \neg L_i) \texttt{ then } L_{i_0}$$

Bemerkungen zur Negation (I):

- Fakten stellen Wissen über konkrete Sachverhalte dar. Die Fakten werden in den Prämissen der Regeln verwendet.
- Prinzipiell ist es möglich, sowohl positive als auch negative Fakten darzustellen.
- Wie negative Fakten zu interpretieren sind, hängt von der Form der verwendeten Negation ab.
- Geht man davon aus, daß das gesamte Wissen über den Anwendungsbereich in der Datenbasis enthalten ist, kann man ein Faktum als falsch ansehen, wenn es nicht in der Datenbasis enthalten ist.
- Man nennt diese grundlegende Annahme *closed-world-assumption*.

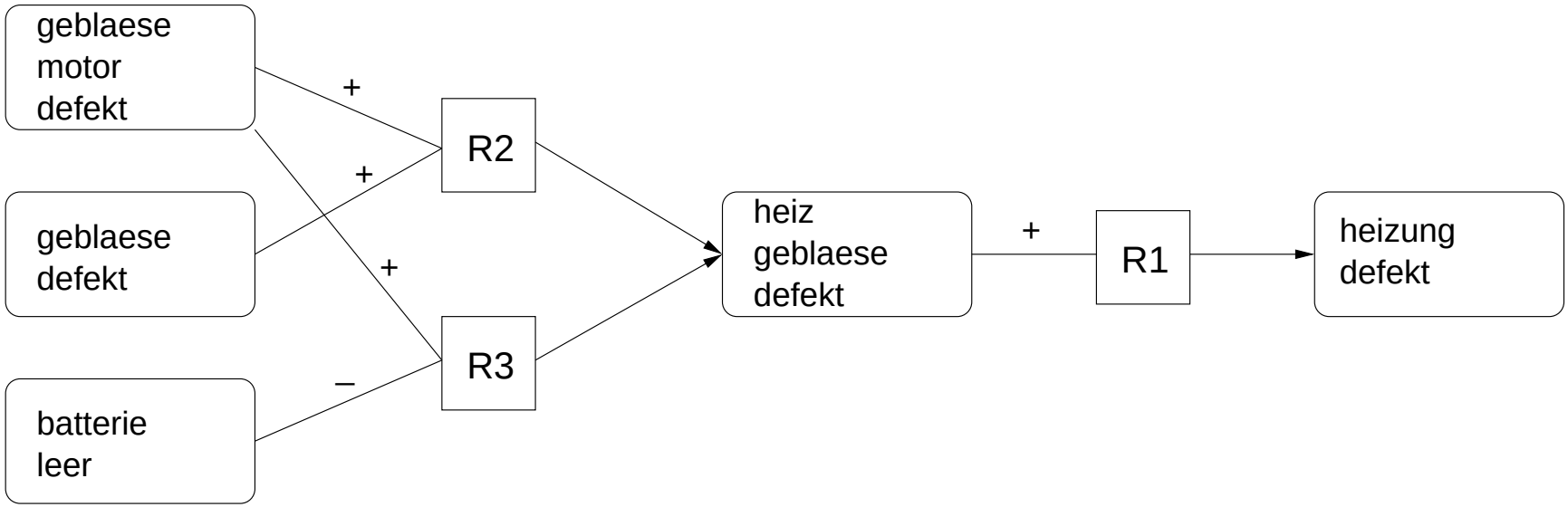
- Ein negatives Faktum f wird in der Form $-f$ geschrieben,
- ein positives in der Form $+f$.
- Die Fakten können noch von weiteren Argumenten abhängen.
- Die Regeln werden benannt und in der Form

$$R : b_1 \wedge \ldots \wedge b_n \rightarrow h$$

geschrieben. Beispiel:

$$\begin{aligned} R_1 &: +rot(ampel) \rightarrow stop \\ R_2 &: +gelb(ampel) \rightarrow stop \\ R_3 &: +gruen(ampel) \rightarrow stop \end{aligned}$$

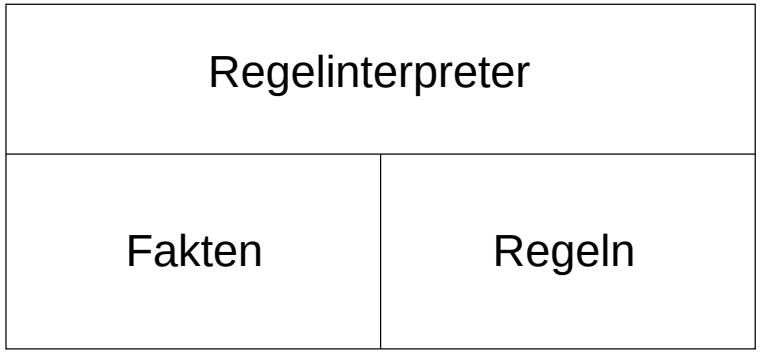
Regeln können auch in graphischer Form als *Regelnetze* repräsentiert werden.

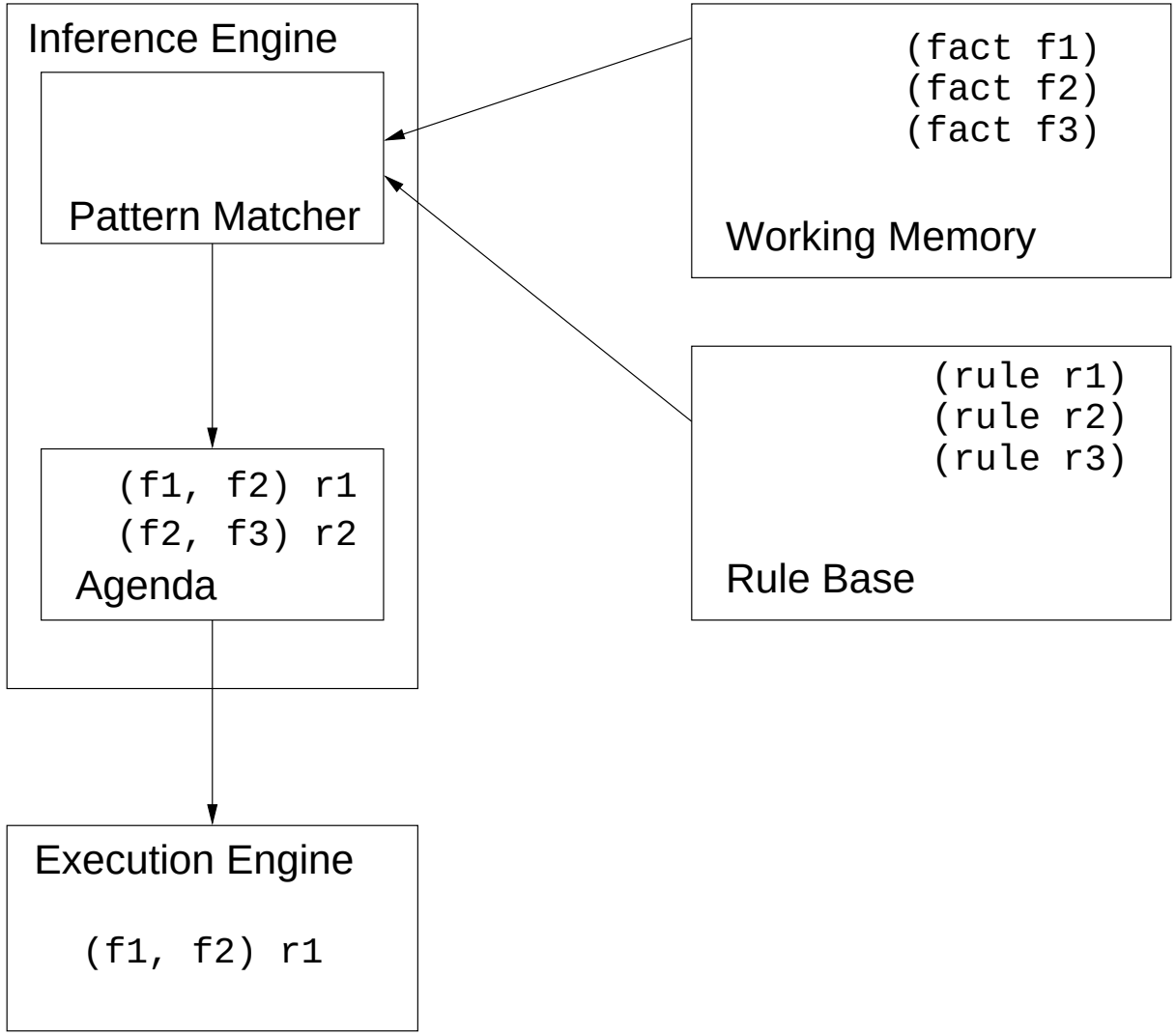


$$R1 : +heizgeblaesedefekt \rightarrow heizungdefekt$$
$$R2 : +geblaesemotordefekt \wedge geblaesedefekt \rightarrow heizgeblaesedefekt$$
$$R3 : +geblaesemotordefekt \wedge -batterieleer \rightarrow heizgeblaesedefekt$$

Architektur regelbasierter Systeme

- Ein *regelbasiertes System (rule-based system, rule engine)* besteht aus Fakten, Regeln und einem Regelinterpretierer (inference engine).
- Die Fakten und Regeln bilden zusammen die *Wissensbasis*.
- Der Teil der Wissensbasis, der die Fakten enthält, wird als *Faktenbasis* bezeichnet.
- Ein sehr einfaches regelbasiertes System ist Prolog.




Inference Engine
Pattern Matcher
(f1, f2) r1
(f2, f3) r2
Agenda
(fact f1)
(fact f2)
(fact f3)
Working Memory
(rule r1)
(rule r2)
(rule r3)
Rule Base
Execution Engine
(f1, f2) r1

Inference Engine

Die Inference Engine ist die wichtigste Komponente eines regelbasierten Systems.

Sie kontrolliert die Anwendung der Regeln auf die Fakten.

1. Die Inference Engine bestimmt mit Hilfe des Pattern Matchers, welche Regeln anwendbar sind.
   Die Menge der anwendbaren Regeln heißt *Konfliktmenge (conflict set)*.
2. Die Konfliktmenge wird geordnet, die entstehende geordnete Liste heißt *Agenda*.
   Die Bestimmung der Ordnung und damit die Auswahl einer Regeln nennen wir *Konfliktlösung*.
3. Die erste Regel der Agenda wird ausgeführt (feuert). Danach wiederholt sich der Prozeß.

Rule Base

- Die *Regelbasis (rule base)* enthält die Regeln des Systems.
- Die Definition der Regeln erfolgt mit Hilfe einer Regelsprache.
- Üblicherweise wird ein *Regel-Compiler* eingesetzt, um die Regeln in eine kompaktere Form zu überführen.
- Durch die Compilierung erfolgt der Aufbau eines Rete-Netzwerks, hierzu später mehr.

Working Memory

- Das sogenannte *working memory* enthält die aktuell gültigen Fakten.
- Üblicherweise werden Indexe eingesetzt, um schnell ermitteln zu können, welche Fakten für welche Regelprämissen relevant sind.
- Das *working memory* ist i.d.R. ähnlich einer relationalen Datenbank strukturiert.
- Schnittstellen zur Repräsentation von Fakten durch Objekte (z.B. in Java) sind üblich (z.B. Jess, ILOG JRules)

Pattern Matcher

- Zweck des pattern matchers ist es, die Menge der anwendbaren Regeln, die Konfliktmenge, zu ermitteln.
- Hierzu müssen Kombinationen von Fakten und Regeln betrachtet werden.
- Hierbei hilft das Rete-Netzwerk.
- Nach einer Änderung des Working Memory wird die Konfliktmenge nicht neu berechnet.
- Stattdessen wird ermittelt, welche Auswirkungen die Änderungen auf die Konfliktmenge haben.

Agenda

- Nach der Ermittlung der Konfliktmenge muß entschieden werden, welche der anwendbaren Regeln ausgeführt werden soll.
- Hier gibt es eine Reihe von Strategien, um Konflikte aufzulösen.
- Prioritäten
- dynamisches Ein- und Ausschalten von Regelmodulen
- spezifischere Regeln zuerst
- zuletzt aktivierte Regeln zuerst

Execution Engine

- Ausführung der ausgewählten Regel
- Änderungen am Working Memory möglich
- Programmierung von Seiteneffekte
- Integration mit einer Programmiersprache/-umgebung

Inferenz in Regelsystemen

Die grundlegende Inferenzregel in einem regelbasierten System ist der Modus Ponens:

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

- Der Modus Ponens benutzt jeweils eine Regel zur Inferenz.
- Die eigentliche Leistung eines regelbasierten Systems zeigt sich aber in seiner Fähigkeit, komplexe Informationen durch die Verkettung von Regeln zu verarbeiten.
- Dabei spielt die Richtung des Inferenzprozesses eine große Rolle.
- Man unterscheidet dabei zwischen *Vorwärtsverkettung (forward chaining)* bzw. *datengetriebene Inferenz (data driven inference)* und
- *Rückwärtsverkettung (backward chaining)* bzw. *zielorientierte Inferenz (goal-oriented inference)*.

Beispiel 4.2. Die Faktenbasis enthalte die Objekte

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

jeweils mit einem zugehörigen Wahrheitswert.

Die Regelbasis enthalte die folgenden Regeln:

R1 : **if** $A \wedge B$ **then** H
R2 : **if** $C \vee D$ **then** I
R3 : **if** $E \wedge F \wedge G$ **then** J
R4 : **if** $H \vee I$ **then** K
R5 : **if** $I \wedge J$ **then** L
R6 : **if** $K \wedge L$ **then** M

Wir werden dieses Beispiel später für die Veranschaulichung von Vorwärts- und Rückwärtsverkettung benutzen.

Es gibt einen wesentlichen Unterschied zwischen regelbasierten Systemen und der klassischen Logik bei der Inferenz:

☞ Regeln sind *gerichtet*, d.h. sie können nur dann angewandt werden (feuern), wenn ihre Prämisse erfüllt ist.

Die beiden Regeln

$$\textbf{if } A \textbf{ then } B \qquad \text{und} \qquad \textbf{if } \neg B \textbf{ then } \neg A$$

sind zwar logisch äquivalent, als Regeln in einer Regelbasis können Sie aber zu unterschiedlichen Ableitungen führen.

Wird nur der Modus Ponens als Inferenzregel verwendet, feuert bei Vorliegen des Faktums A nur die erste Regel, bei Nichtvorliegen des Faktums B nur die zweite Regel.

Legt man auf die Erhaltung der logischen Äquivalenz Wert, gibt es die folgenden Möglichkeiten:

- Man nimmt beide Regeln in die Wissensbasis auf oder
- man implementiert zusätzlich den Modus Tollens.

if A **then** B	(Regel)
B falsch	(Faktum)
A falsch	(Schlußfolgerung)

Rückwärtsverkettung

- Bei der *Rückwärtsverkettung* geht man von einem Zielobjekt aus, über dessen Zustand der Benutzer Informationen wünscht.
- Das System durchsucht dann die Regelbasis nach geeigneten Regeln, die das Zielobjekt in der Konklusion enthalten.
- Die Objekte der Prämissen werden zu Zwischenzielen.
- Die Regeln werden also vom Folgerungsteil zum Bedingungsteil hin ausgewertet.
- Dies entspricht der Vorgehensweise eines Prolog-Interpreters.

Beispiel 4.3. Wir greifen Beispiel 4.2 auf. Gegeben sei die Faktenmenge $\mathcal{F}$ durch:

$$\mathrm{H} = \mathrm{true}, \mathrm{C} = \mathrm{true}, \mathrm{E} = \mathrm{true}, \mathrm{F} = \mathrm{true}, \mathrm{G} = \mathrm{true}$$

Das Zielobjekt sei M.

Durch die Rückwärtsverkettung werden in dieser Reihenfolge die Werte

$$\mathrm{K} = \mathrm{true}, \mathrm{I} = \mathrm{true}, \mathrm{J} = \mathrm{true}, \mathrm{L} = \mathrm{true}, \mathrm{M} = \mathrm{true}$$

abgeleitet.

Steuerung der Regelinterpretation

Die Reihenfolge, in der die Regeln bei der Rückwärtsverkettung verarbeitet werden, hat Einfluß auf

- die Performanz des Systems und
- die Anzahl der Fragen, die an den Benutzer gestellt werden.

Bisher wird die Reihenfolge durch die Reihenfolge der Regeln in der Wissensbasis bestimmt.

Dieses Schema ist im allgemeinen zu unflexibel.

- Wir gehen davon aus, daß gewisse Regeln mit größerer Wahrscheinlichkeit als andere Regeln zutreffen.
- In diesem Fall sollte man die Regeln mit höherer Wahrscheinlichkeit zuerst abarbeiten.
- Gründe für unterschiedliche Wahrscheinlichkeiten:
  - Es werden Fakten abgeleitet, die andere Hypothesen weniger wahrscheinlich machen.
    Beispiel: Ist das Geschlecht einer Person bekannt, so können gewisse nicht zutreffende geschlechtsspezifische Krankheiten als sehr unwahrscheinlich angesehen werden.
  - Im Verlauf längerer Erfahrungen mit einem Regelsystem kann sich herausstellen, daß bestimmte Regeln generell häufiger erfüllt sind als andere.

Steuerungswissen, Metawissen

- Wissen dieser Form heißt *Steuerungswissen* oder *strategisches Wissen*. Es dient dazu, die Inferenz zu steuern. Steuerungswissen ist eine Form von *Metawissen*.
- Es unterscheidet sich damit von dem *Objektwissen*, welches das Wissen über die Objekte des Anwendungsbereichs darstellt.
- Wir gehen davon aus, daß die Auswahl einer Regel unabhängig von der Reihenfolge der Regeln in der Regelbasis getroffen werden kann.
- Eine Situation, in der mehrere Regeln anwendbar sind, nennt man *Regelkonflikt*.
- Die Menge der anwendbaren Regeln ist die *Konfliktmenge*.
- Die Auswahl einer Regeln nennt man *Konfliktlösung*.

Konfliktlösung mit statischen Prioritäten

Es kann sehr schwierig sein, die “richtige” Regel auszuwählen und von dieser Regeln kann viel abhängen.

☞ Die Konflikte sind der natürliche Preis der Annehmlichkeiten des deklarativen Programmierens.

- Jede Regel erhält eine Priorität.
- Aus der Konfliktmenge wird die Regel mit der höchsten Priorität ausgewählt. Dies entspricht einer partiellen Ordnung der Regelmenge.
- Ist die Regel nicht eindeutig bestimmt, wird der Konflikt wiederum anders aufgelöst (z.B. durch die Reihenfolge).

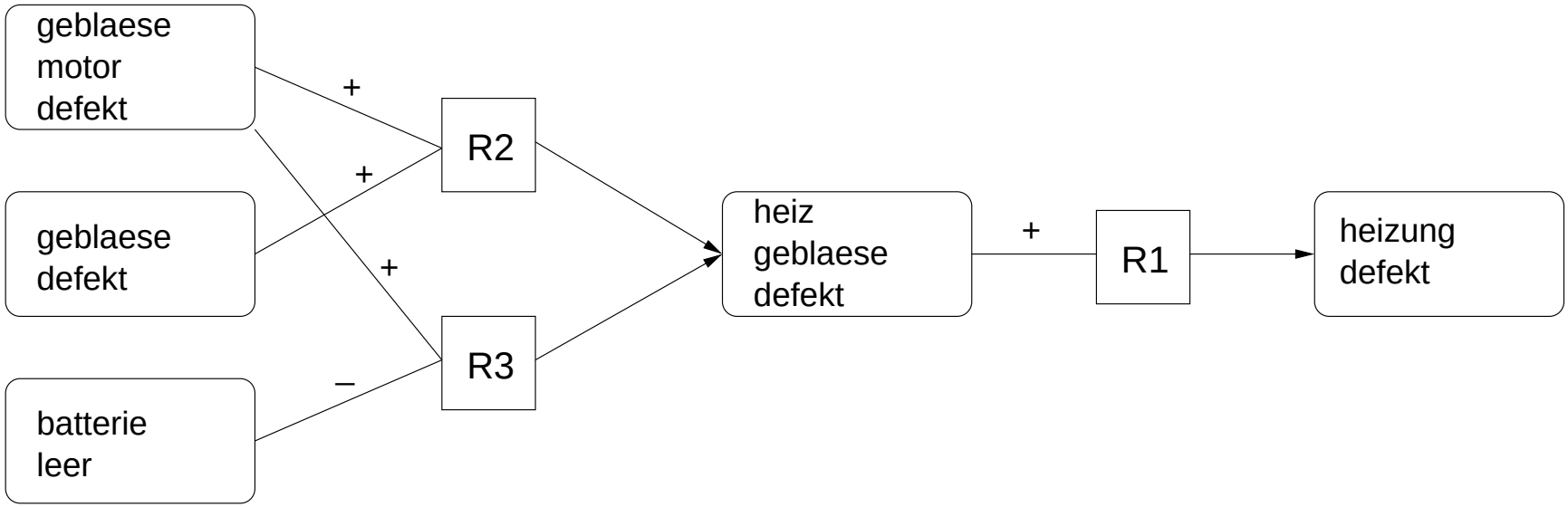
Konfliktlösung mit dynamischen Prioritäten

- Wir führen zusätzlich *heuristische Regeln* ein. Heuristische Regeln repräsentieren das Steuerungswissen.
- Die Prämisse einer heuristischen Regel hat die gleiche Form wie die einer normalen Regel.
- Die Konklusion einer heuristischen Regel ist eine Aktion, die eine Änderung einer Regelpriorität bewirkt.
- Schreibweise:

$$\mathfrak{h} : \mathfrak{a}_1 \wedge \ldots \wedge \mathfrak{a}_n \rightarrow \mathfrak{add}(R, \delta)$$

- Semantik solch einer heuristischen Regel: Ist die Prämisse $\mathfrak{a}_1 \wedge \ldots \wedge \mathfrak{a}_n$ erfüllt, so addiere δ zu der Priorität der Regel R.

Regelnetzwerk



$R1 : +heizgeblaesedefekt \rightarrow heizungdefekt$

$R2 : +geblaesemotordefekt \wedge geblaesedefekt \rightarrow heizgeblaesedefekt$

$R3 : +geblaesemotordefekt \wedge -batterieleer \rightarrow heizgeblaesedefekt$

Beispiel 4.4. Wir betrachten diese Regeln. Die Faktenbasis sei leer. Eine mögliche Ableitung ohne Prioritäten lautet:

Hypothesen	Regeln	Faktenbasis	Eingabe
Heizung defekt?	R_1	-	
Heizgebläse defekt?	R_2, R_3	-	
Gebläsemotor defekt?	-	-	Ja
Gebläse defekt?	-	Gebläsemotor defekt	Nein
Heizgebläse defekt?	R_3	Gebläsemotor defekt	
Gebläsemotor defekt?	-	Gebläsemotor defekt	
Batterie leer?	-	Gebläsemotor defekt	Nein

Wir nehmen nun eine heuristische Regel hinzu:

$$h_1 : +licht \rightarrow add(R_3, 5)$$

Verlauf Tafel ✎

Vorwärtsverkettung

Rückwärtsverkettung ist nur unter den folgenden Bedingungen sinnvoll:

- Es muß möglich sein, realistische Hypothesen zielgerichtet zu generieren.
- Bei der Rückwärtsverkettung wird der Benutzer aufgefordert, Beobachtungen zu machen, Tests durchzuführen oder Werte einzugeben. Sind im voraus alle Fakten bekannt, ist diese Vorgehensweise weniger sinnvoll.

- Kernstück der *Vorwärtsverkettung* ist die wiederholte Auswahl einer Regel, deren Prämisse vollständig erfüllt ist.
- Der Aktionsteil dieser Regel wird ausgeführt, die Faktenbasis wird entsprechend angepaßt.
- Anschließend wird eine neue Regel ausgeführt.

Terminierung:

☞ Die Rückwärtsverkettung endet, wenn die Hypothese bewiesen wurde oder alle Beweismöglichkeiten ausgeschöpft sind.

☞ Die Vorwärtsverkettung endet, wenn keine neuen Fakten mehr abgeleitet werden können oder ein Zielfaktum generiert wurde.

- Man kann sich ein vorwärtsverkettetes System wie ein Spiel vorstellen.
- Die Regeln entsprechen den Spielregeln.
- Die Fakten beschreiben die aktuelle Spielstellung.
- Die Regeln geben nur einen formalen Rahmen für den Ablauf vor. Sie geben keine Auskunft darüber, wie das Spiel zu gewinnen ist.
- Nichtsdestotrotz können die Regeln natürlich gewisse Gewinnstrategien implementieren.

Beispiel 4.5. Das Spiel besteht aus zwei Regeln:

R1 : Wenn sich im Korb zwei schwarze Chips befinden, dann nimm sie heraus und lege einen Weißen hinein.

R2: Wenn sich im Korb ein schwarzer und ein weißer Chip befinden, dann nimm den Weißen heraus.

Das Spiel ist beendet, wenn kein Zug mehr möglich ist.

Ziel ist es, am Ende so wenig Chips wie möglich im Korb zu behalten.

Eine Spielstellung beschreiben wir durch ein Faktum $\mathtt{status}(s, w)$, wobei s die Anzahl der schwarzen und w die Anzahl der weißen Chips angibt.

Die Regeln lauten für das Chipspiel formal:

$$R_1 \quad : \quad +status(s, w) \wedge s \geq 2 \rightarrow status(s - 2, w + 1)$$

$$R_2 \quad : \quad +status(s, w) \wedge s \geq 1 \wedge w \geq 1 \rightarrow status(s, w - 1)$$

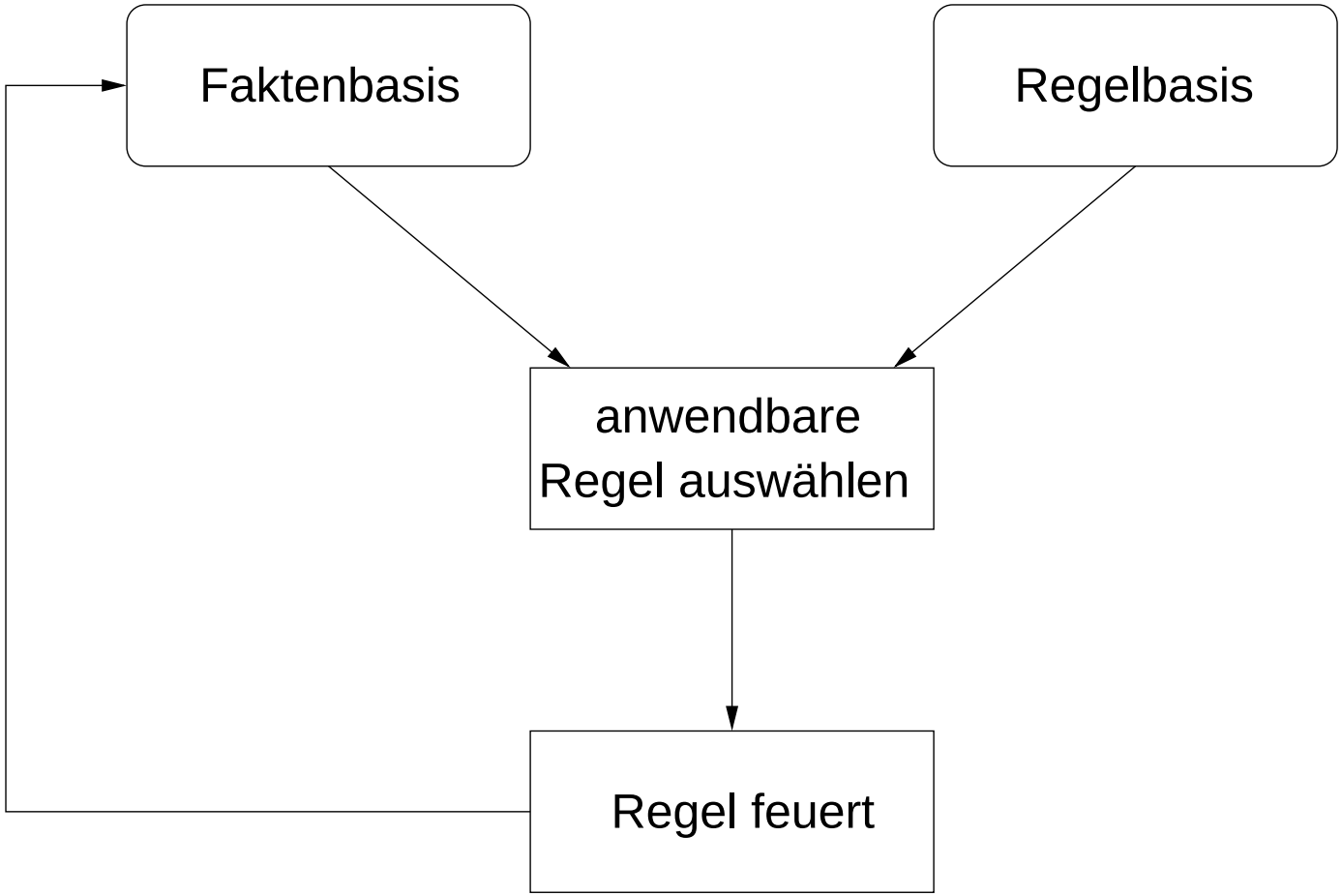
Als Anfangsstellung könnte beispielsweise

$$\{status(2, 2)\}$$

vorliegen.

- Die Regelinterpretation verläuft dann in einer Schleife.
- In jedem Zyklus wird eine anwendbare Regel ausgewählt.

Einfaches Schema für die Vorwärtsverkettung:



Definition 4.1.

- Es sei b eine variablenfreie Prämisse.
  Ist $b = +a$, so ist b *erfüllt* gdw. a in der Faktenbasis ist.
  Ist $b = -a$, so ist b *erfüllt* gdw. a nicht in der Faktenbasis enthalten ist.
  Ist b eine prozedurale Prämisse, so ist b *erfüllt* gdw. b zu `true` ausgewertet wird.
- Sei $r = b_1 \wedge \ldots \wedge b_n \rightarrow h$ eine Regel und σ eine Substitution. Wir definieren

$$\sigma(r) := \sigma(b_1) \wedge \ldots \wedge \sigma(b_n) \rightarrow \sigma(h)$$

  $\sigma(r)$ heißt eine *Instanz* von r.
- Eine Regelinstanz $\sigma(r)$ heißt *anwendbar* in einer Faktenbasis gdw. alle Prämissen von $\sigma(r)$ in der Faktenbasis erfüllt sind.

Bemerkungen:

- Ist eine anwendbare Regelinstanz ausgewählt, so kann diese Regel feuern, d.h. die Konklusion wird als neues Faktum in die Faktenbasis eingetragen.

- Im Verlauf der Interpretation werden so neue Fakten erzeugt, die alten werden aber nicht gelöscht.
- Würde man die Fakten löschen, könnte man keine Alternativen im Suchbaum ausprobieren.

Schreibweise: Eine Regelinstanz $\sigma(r)$ schreiben wir auch in der Form $r(x_1, \ldots, x_n)$, wobei die x_1 bis x_n die in der Regel verwendeten Variablenwerte sind.

Beispiel 4.6. Ein möglicher Verlauf des Chipsspiel:

Iteration	Konfliktmenge	neues Faktum
1	$R_1(2,2)*, R_2(2,2)$	$status(0,3)$
2	$R_1(2,2), R_2(2,2)*$	$status(2,1)$
3	$R_1(2,2), R_2(2,2), R_1(2,1), R_2(2,1)*$	$status(2,0)$
4	$R_1(2,2), R_2(2,2), R_1(2,1), R_2(2,1), R_1(2,0)*$	$status(0,1)$

Die jeweils ausgewählte Regel ist mit * markiert.