Logik und Inferenz

Einführung

- wesentliches Merkmal wissensbas. Systeme:
  Problemwissen u. *Wissensverarbeitung* getrennt

- Wissensmanagement z.T. konventionell (DB)

- herausragende Rolle von Logik und Inferenz
  Logik zur Wissensrepräsentation
  Inferenz zur Ableitung nicht explizit vorhandenen Wissens

- Repräsentationmethode u. Inferenz bilden Einheit

- übersetze nichtlogische Formalismen in Logik
  z.B. Frames, semantische Netze $\rightsquigarrow$ Präd.logik

Verarbeitungsmodelle für Regelsysteme

- **vorwärtsverkettend** (*datenorientiert*)
  suche anwendb. Wissen ausgehend v. d. Daten

- **rückwärtsverkettend** (*zielorientiert*)
  suche anwend. Wissen ausgehend v. Gesamtziel
  zerlege Gesamtziel in einfachere Unterziele

Beispiele

- **vorwärtsverkettend**
  OPS-5 (regelorientierte Programmiersprache)
  SOAR, ACT (simulieren kogn. Verhaltens)
  R1/XCON (Konfiguration von VAX-Rechnern)

- **rückwärtsverkettend**
  MYCIN (diagnostiziert Infektionskrankheiten)
  PROLOG (logische Programmiersprache)

Verarbeitungsmodelle für Regelsysteme

- **vorwärtsverkettend** (*datenorientiert*)
  suche anwendb. Wissen ausgehend v. d. Daten

- **rückwärtsverkettend** (*zielorientiert*)
  suche anwend. Wissen ausgehend v. Gesamtziel
  zerlege Gesamtziel in einfachere Unterziele

Beispiele

- **vorwärtsverkettend**
  OPS-5 (regelorientierte Programmiersprache)
  SOAR, ACT (simulieren kogn. Verhaltens)
  R1/XCON (Konfiguration von VAX-Rechnern)

- **rückwärtsverkettend**
  MYCIN (diagnostiziert Infektionskrankheiten)
  PROLOG (logische Programmiersprache)

Elemente eines Produktionensystems (1)

- **Regel-/Produktionensystem (PS):**
  1. Arbeitsspeicher (*working memory (WM)*)
  2. Regelspeicher (*rule memory (RM)*)

- *WM*: Menge von Typen + deren Instanzen
  Typen $\widehat{=}$ structure Deklarationen
  WM Elemente (WME) $\widehat{=}$ Instanzen

Beispiel WME

```
(Person
        ↑name     Jonas
        ↑alter    5
        ↑straße   Ahorngasse 15
        ↑ort      Nirgendwo)
```

- WME `Person` mit den Attributen `name`, ...
- Attribut `name` hat Wert `Jonas`, ...

Operationen auf WMEs:

`make`: Erstellen eines WME
`modify`: Ändern eines WMEs
`remove`: Löschen eines WMEs

Elemente eines Produktionensystems (2)

Aufbau von Regeln mit Beispiel

```
(p   Regelname
     Bedingung 1
     :
     Bedingung n
→
     Aktionen)
```

```
(p   finde-bunten-block
     (goal ↑typ finde-block ↑farbe ⟨c⟩)
-    (goal ↑typ emergency-stop)
     (block ↑farbe ⟨c⟩ ↑selektiert n)
→
     (modify 2 ↑selektiert j))
```

- Bedingungsteil = LHS (left hand side)
- Aktionsteil = RHS (right hand side)

⟨c⟩ – Variable
– – Negation

Anwenden von Regeln

- Regel anwendbar, wenn *alle* Vorbed. durch aktuellen Zustand des WM erfüllt sind

- Bsp: WMEs `goal` und `block` gleicher `farbe`
  der `typ` des `goal` ist `finde-block`
  der `block` ist nicht selektiert
  es gibt *kein* `goal` v. `typ emergency-stop`

- Regelanwendung: führe Aktionen in RHS aus

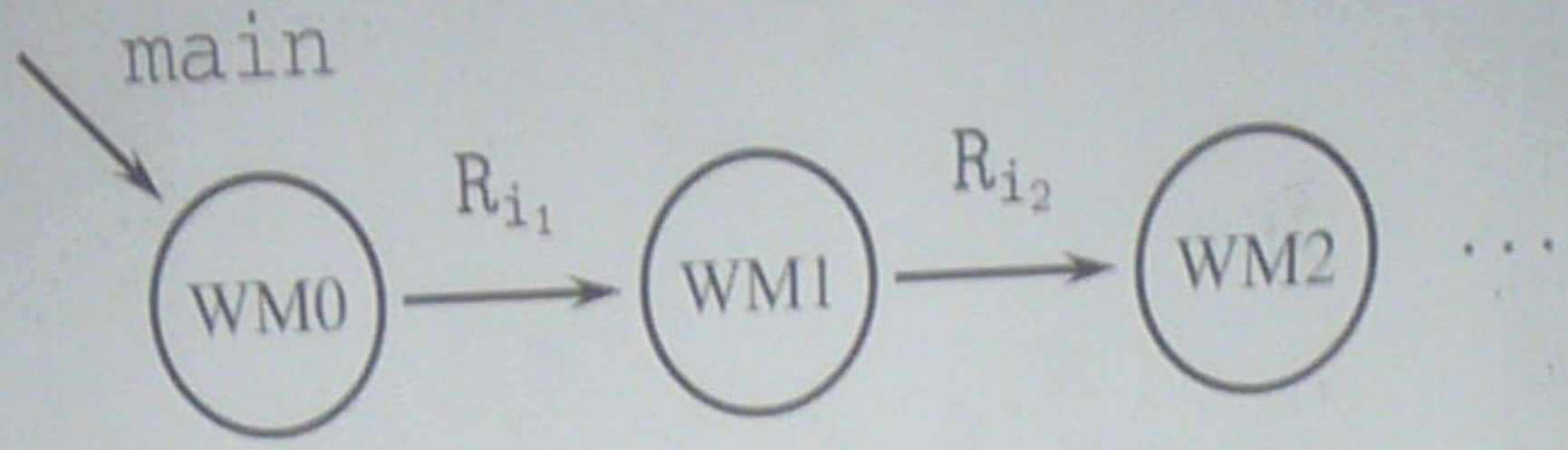
- Bsp: `(modify 2 ↑selektiert j)`
  modifiziere das an 2ter Stelle gebundene WME

- Zählweise: von oben *ohne* negative WMEs
  $\Longrightarrow$ `selektiert` in `block` auf 'j'

Dynamisches Abarbeitungsmodell

Prinzipielle Idee von forward-chaining:

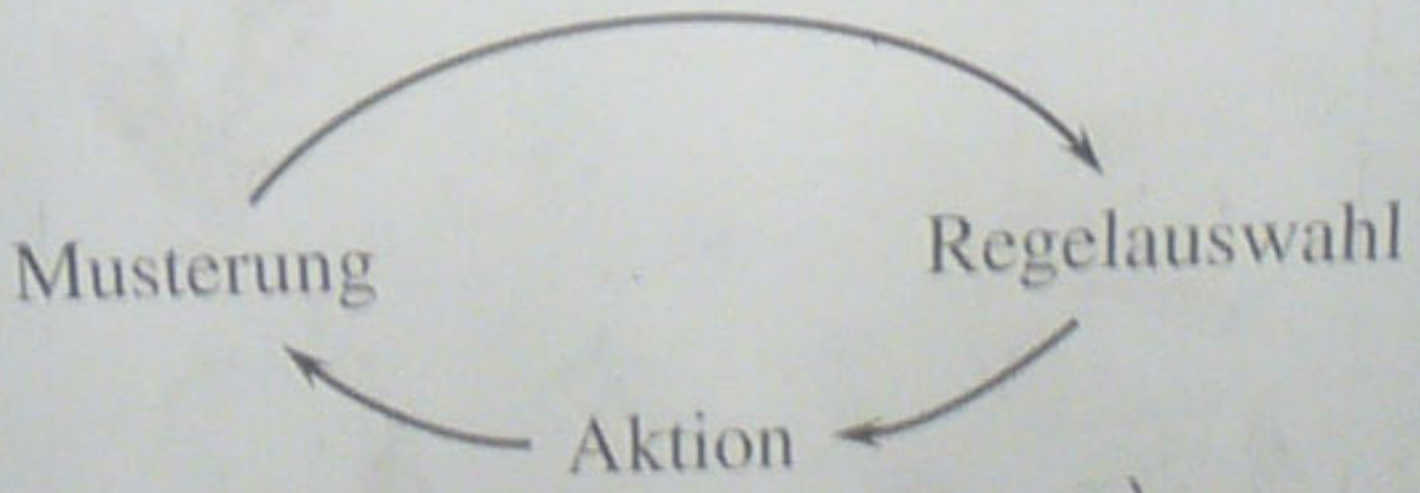
Wende Regel auf Startzustand an und generiere Zustand, der als neuer Startzustand betrachtet wird.



(WMi: WM nach der i-ten Regelanwendung)

- Initialisierung: Regel „`main`" (ohne LHS)
  erzeugt alle Initial-„Fakten"
  Resultat: WM0

- starte *Recognize-Act-Cycle* (RAC)



Musterung (Matching)

- Überprüfe Relevanz des in Regelform vorliegenden Wissens auf aktuelle Situation (WM)
- Gegeben: aktueller Zustand WMj
- Gesucht: alle „auf" WMj anwendbaren Regeln
- Durchsuche für jede Regel WMj, ob für
  1. pos Bedingungen: WME vorhanden
  2. neg Bedingungen: *kein* WME vorhanden
- Liefert **conflict set (CS)** = Menge aller Regelinstanzen

  Regelinstanz: $\langle R, (\mathrm{WME}_1, \ldots, \mathrm{WME}_n) \rangle$

  R: anwendbare Regel

  WME_i: pos Bedingungen als geordnete Liste

Regelauswahl

Ausgang: Conflict Set (CS)

- bewerte das im CS bereitgestellte Wissen heuristisch

- Ziel: Auswahl *einer* Regel aus dem CS

- Wahl einer Standard-Selektionsstrategie oder eigene Selektionsstrategie

Kriterien für die Regelauswahl:

- zeitliche Kriterien

  z.B. bevorzuge „neuere" Information

- syntaktische Kriterien

  z.B. bevorzuge spezifischere Information

- Meta-Wissen

  z.B. berücksichtige explizite Prioritäten

- zufällige Auswahl

Aktion

- Anwendung des in der gegebenen Situation „besten" Wissens
- Ausführung der Aktionen der gewählten Regel

Abschließende Bemerkungen

- RAC terminiert, wenn CS leer oder `halt` Befehl in RHS ausgeführt wird
- gute Musterungs-Methode: *RETE*-Algorithmus (kompiliere die Regeln in Diskriminantennetz)
- unterschiedliche Parallelisierungsmethoden
- Probleme:
  - Termination
  - Semantik
  - nicht „zielsensitiv"

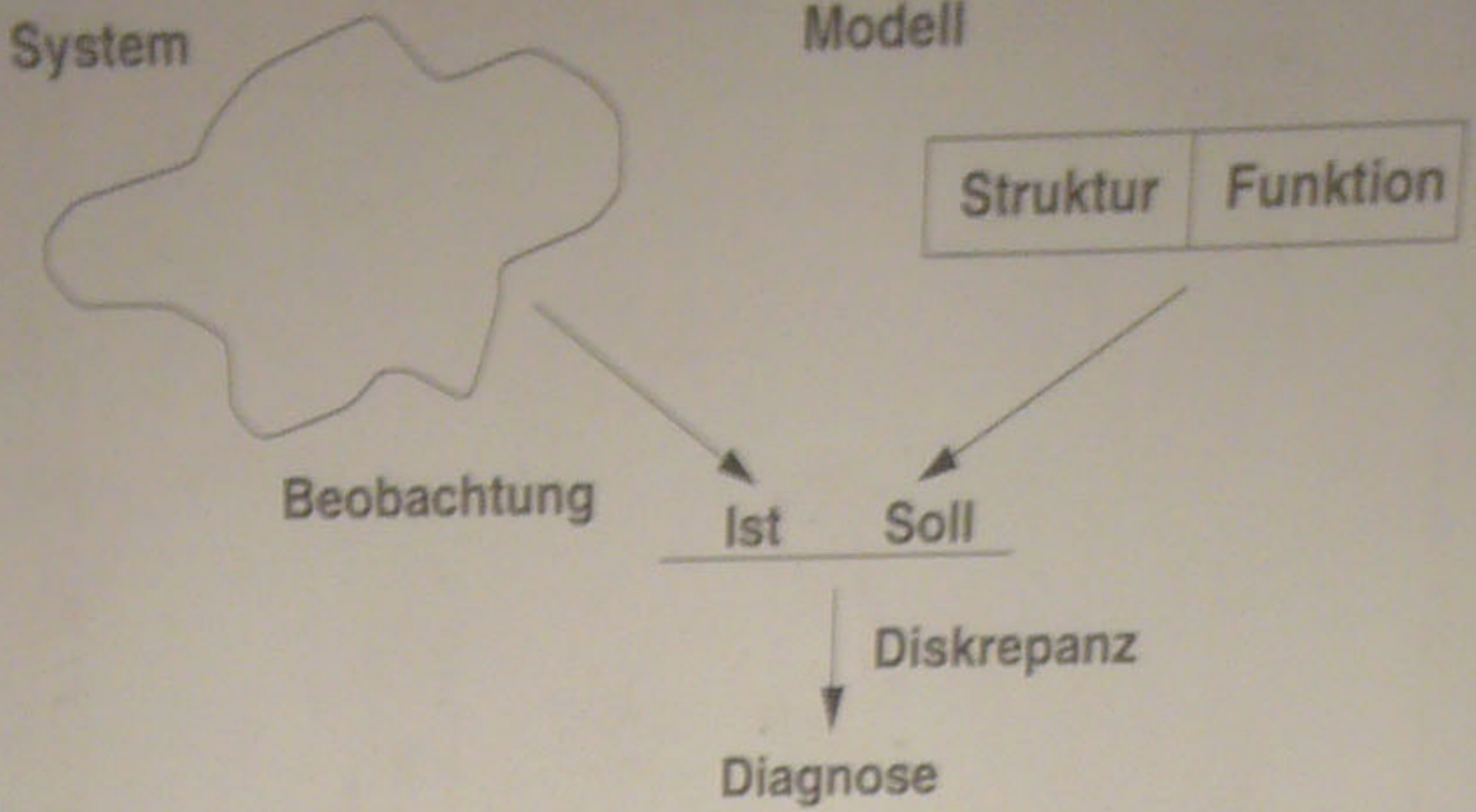
Diagnose

Aufgabe: Feststellen fehlerhafter Komponenten.

Lösungsansätze:

- regelbasierte Ansätze:
  - schwer wart- bzw. erweiterbar etc.
- **modellbasierte Ansätze:**
  - Systemverhalten wird formal beschrieben
    $\Longrightarrow$ Fehler durch Vergleich zwischen vorhergesagtem und beobachtetem Verhalten erkennen.
  - Neue Komponenten können leicht eingebunden werden.
  - Spezifikation der Komponenten oft schon während der Entwicklung vorhanden.

Modellbasierte Diagnose



Diagnosemodell: Beschreibung des Systems.

Domain Theory + Fakten: Bauteile, Beobachtungen

Korrektheitsannahmen der Komponenten

Diagnosealgorithmus: Ermittle fehlerhafte Komponenten. Prinzip:

- Beobachtungen (Ist) machen das Systemmodell (Soll) **inkonsistent**.
- Mache Modell durch **Korrektheitsannahmen** der Komponenten **konsistent**.
- **Diagnose** = Menge als fehlerhaft angenommenen Bauteile.

Meßpunktselektion: Auswahl von optimalen Meßpunkten (Beobachtungswerte).

Diagnoseproblem

Ein **Diagnoseproblem** $(SD, OBS, COMP)$ besteht aus:

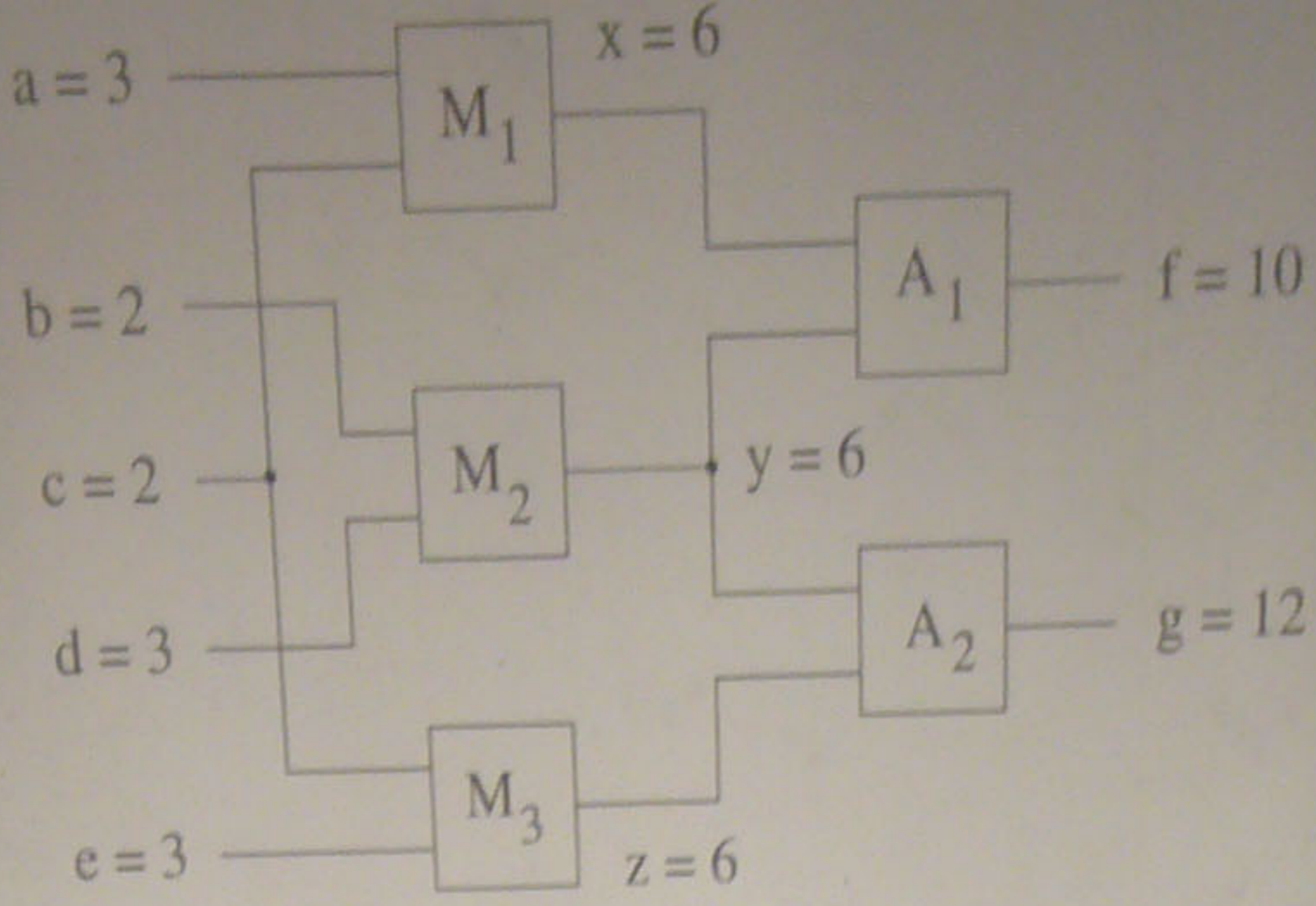
- einer logischen Theorie SD (*Systembeschreibung*);
- einer Menge von Fakten OBS (*Beobachtungen*);
- einer Menge von Konstantensymbolen $COMP$ (die *Bauteile*).

Eine **Diagnose** für $(SD, OBS, COMP)$ ist eine bzgl. $\subseteq$ minimale Menge $\Delta \subseteq COMP$ Komponenten, sodaß

$$SD \cup OBS \cup \{\neg ok(c) \mid c \in \Delta\} \cup \{ok(c) \mid c \in COMP \setminus \Delta\}$$

erfüllbar ist. ($ok(x)$ .... Komponente x ist korrekt)

Beispiel: Schaltkreis



M_i: Multiplizierer, A_i: Addierer

SD:

– Eigenschaften der Bauteile: z.B.

$$(type(m, multiplier) \wedge ok(m) \wedge val(in_1(m), v_1) \wedge val(in_2(m), v_2) \wedge v_3 = v_1 \cdot v_2)) \rightarrow val(out(m), v_3).$$

↖ built-in Multiplikation

– Eigenschaften der Verbindungsstellen: z.B.

$$(val(P, v_1) \wedge val(P, v_2)) \wedge v_1 \neq v_2) \rightarrow \bot$$

$$(conn(p_1, p_2) \wedge val(p_1, v)) \rightarrow val(p_2, v).$$

– Bauteile und Verbindungen:

$type(M_1, multiplier)$; $conn(out(M_1), in_1(A_1))$, ...

OBS: Meßwerte, z.B. $val(a, 3)$; $val(x, 6)$.

COMP: $\{M_1, M_2, M_3, A_1, A_2\}$

$$(type(m, multiplier) \wedge ok(m) \wedge val(in_1(m), v_1) \wedge val(in_2(m), v_2) \wedge v_3 = v_1 \cdot v_2)) \rightarrow val(out(m), v_3).$$

$$(type(m, multiplier) \wedge ok(m) \wedge val(out(m), v_3) \wedge (in_2(m), v_2) \wedge v_1 = v_3 \div v_2)) \rightarrow val(in_1(m), v_1).$$

$$(type(m, multiplier) \wedge ok(m) \wedge val(out(m), v_3) \wedge val(in_1(m), v_1) \wedge v_2 = v_3 \div v_1)) \rightarrow val(in_2(m), v_2).$$

$$(type(a, adder) \wedge ok(a) \wedge val(in_1(a), v_1) \wedge val(in_2(a), v_2) \wedge v_3 = v_1 + v_2)) \rightarrow val(out(a), v_3).$$

$$(type(a, adder) \wedge ok(a) \wedge val(out(a), v_3) \wedge val(in_2(a), v_2) \wedge v_1 = v_3 - v_2)) \rightarrow val(in_1(a), v_1).$$

$$(type(a, adder) \wedge ok(a) \wedge val(out(a), v_3) \wedge val(in_1(a), v_1) \wedge v_2 = v_3 - v_1)) \rightarrow val(in_2(a), v_2).$$

$$(val(P, v_1) \wedge val(P, v_2)) \wedge v_1 \neq v_2) \rightarrow \bot.$$

$$(conn(p_1, p_2) \wedge val(p_1, v)) \rightarrow val(p_2, v).$$
$$(conn(p_1, p_2) \wedge val(p_2, v)) \rightarrow val(p_1, v).$$

$type(M_1, multiplier)$; $type(M_2, multiplier)$; $type(M_3, multiplier)$.
$type(A_1, adder)$; $type(A_2, adder)$;

$conn(out(M_1), in_1(A_1))$; $conn(in_2(M_1), in_1(M_3))$;
$conn(out(M_2), in_2(A_1))$; $conn(out(M_2), in_1(A_2))$;
$conn(out(M_3), in_2(A_2))$.

$conn(a, in_1(M_1))$; $conn(x, out(M_1))$;
$conn(b, in_1(M_2))$; $conn(y, out(M_2))$;
$conn(c, in_1(M_3))$; $conn(z, out(M_3))$;
$conn(d, in_2(M_2))$; $conn(f, out(A_1))$;
$conn(e, in_2(M_3))$; $conn(g, out(A_2))$.

$val(a, 3)$; $val(b, 2)$; $val(c, 2)$; $val(d, 3)$; $val(e, 3)$;
$val(f, 10)$; $val(g, 12)$;
$val(x, 6)$; $val(y, 6)$; $val(z, 6)$.

Diagnosevorgang

(1) Messwerte für a bis e: Menge von Fakten, konsistent mit SD u. Annahme daß kein Bauteil defekt $\Longrightarrow$ Diagnose $\{\}$.

(2) Messe $f = 10$. $\Longrightarrow$ Widerspruch mit Systembeschreibung: $f = 12$.

Mögliche Diagnosen: $\{M_1\}$, $\{M_2\}$, und $\{A_1\}$.

(3) Messe $g = 12$. $\Longrightarrow$ Schließe $\{M_2\}$ aus.

Neue Diagnosen: $\{M_1\}$, $\{A_1\}$, $\{M_2, M_3\}$, $\{M_2, A_2\}$. ← Maskierung!

(4) Messung $x = 6$: Endgültige Diagnose $\{A_1\}$

Noch Diagnosen: $\{M_2, M_3\}$, $\{M_2, A_2\}$

Bemerkungen:

- SD kann auch **Verhaltensmodelle** von fehlerhaften Komponenten enthalten.
- **Systemwartung** einfacher durch[illegible] von Diagnosewissen und Verhaltens[illegible]ng als bei heuristischen Verfahren
- Mehrfachfehler

Diagnosevorgang

(1) Messwerte für a bis e: Menge von Fakten, konsistent mit SD u. Annahme daß kein Bauteil defekt $\Longrightarrow$ Diagnose $\{\}$.

(2) Messe $f = 10$. $\Longrightarrow$ Widerspruch mit Systembeschreibung: $f = 12$.

Mögliche Diagnosen: $\{M_1\}$, $\{M_2\}$, und $\{A_1\}$.

(3) Messe $g = 12$. $\Longrightarrow$ Schließe $\{M_2\}$ aus.

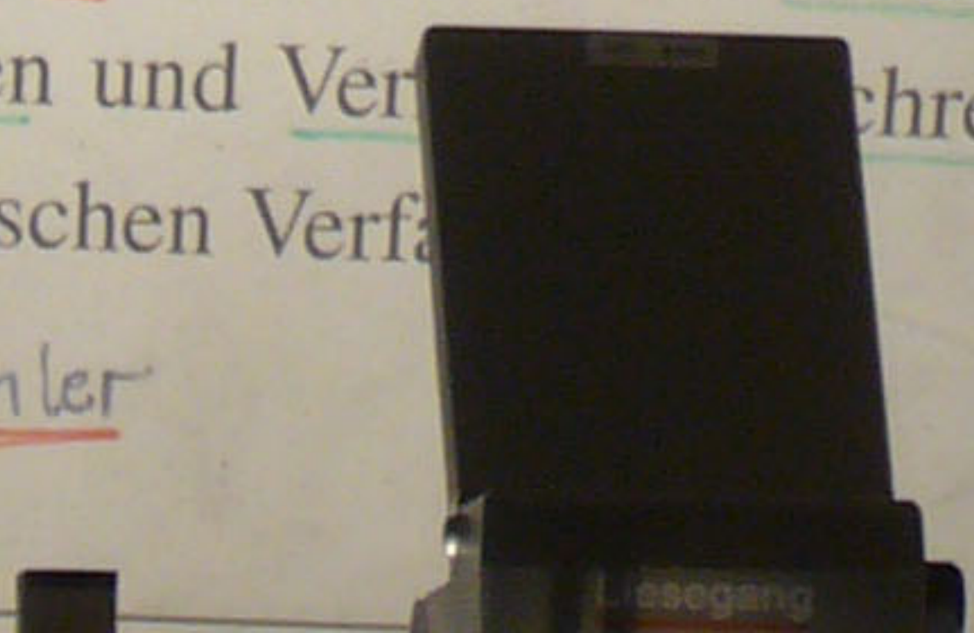
Neue Diagnosen: $\{M_1\}$, $\{A_1\}$, $\{M_2, M_3\}$, $\{M_2, A_2\}$. ← Maskierung!

(4) Messung $x = 6$: Endgültige Diagnose $\{A_1\}$

Noch Diagnosen: $\{M_2, M_3\}$, $\{M_2, A_2\}$

Bemerkungen:

- SD kann auch **Verhaltensmodelle** von fehlerhaften Komponenten enthalten.
- **Systemwartung** einfacher durch Trennung von Diagnosewissen und Ver[illegible]chreibung als bei heuristischen Verfa[illegible]
- Mehrfachfehler



Suchraummodelle

Planungsproblem: Erreiche Ziel durch Aktionen

Plan: Ablauf der Aktionen

Für formales Planungssystem erforderlich:

- Modell der „Welt" ;
- Modell für Aktionen und deren Effekte.

Zustandsraum Graph

Suchgraph $G = (V, E)$

Knoten $v_i \in V$: möglicher Zustand (*possible world*)

Kante $v_1 \rightarrow v_2 \in E$ markiert mit Aktion α:
Ausführung von α im Zustand v_1 resultiert im Zustand v_2.

Einfaches Planen: Suche Pfad im Graphen G von Startknoten s zu Zielknoten (Goal) g

Voraussetzungen und Eigenschaften

- Alle möglichen Zustände müssen repräsentiert werden.
- *Exakte* Modellierung der Aktionen
- Aktionen müssen angegebene Effekte haben.
  $\Rightarrow$ keine *Unsicherheit* (Erreiche Effekt mit bestimmter Wahrscheinlichkeit)
  $\Rightarrow$ kein *Nichtdeterminismus* (α führt s in s_1 oder s_2 über)
- Annahme: *keine äußeren Einflüsse*, *keine* Änderungen im System während der Planungsphase

Probleme:

- Suchgraph oft extrem groß
- Graph wird nicht explizit generiert: Suchbaum (Knoten mehrfach)

Situationskalkül

Planungssystem: verwendet Präd.logik 1. Stufe

- Formalisiere Zustände und Aktionen durch *Objekte*.
- Repräsentiere Wissen über Zustände, Übergänge etc. durch *Prädikate und Formeln*.
- Gewinne Plan aus Antwort zu Anfrage A an Deduktionssystem

  A: „Kann ein Zielzustand s erreicht werden?"

  Antwort: s, das nach Folge von Aktionsausführungen resultiert.

Zustände S: *Objektkonstanten* $S_0, S_1, \ldots$

Fluents: Prädikate für Aussagen über Systemzustände, z.B. $On(A, B)$.

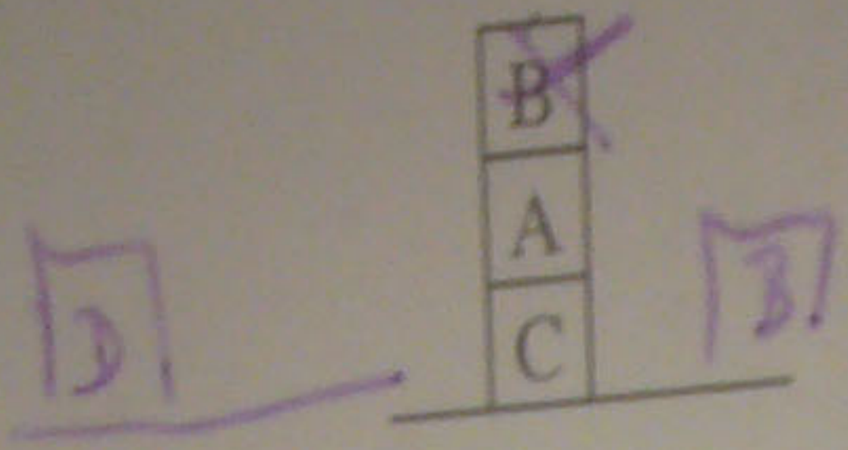
Aktionen α: Funktionen, z.B. $move(A, B, Fl)$

„*do*" **Funktion:** $do : \mathsf{Actions} \times \mathsf{States} \rightarrow \mathsf{States}$

$(\alpha, S) \mapsto do(\alpha, S)\ (= S')$.

Intuitiv: Erhalte S' durch Ausführung von α aus S.

Beispiel: „Blocks-World" Szenario



On(B, A, S_0)
On(A, C, S_0)
On(C, Fl, S_0)
Clear(B, S_0)
Clear(Fl, S_0)

Objekte: Zustände $S_0, S_1, \ldots$ Blöcke A, B, C
Boden Fl (Unique Names Assumption)

Fluents: $clear(x, s)$: Objekt x ist frei im Zustand s.
$on(x, y, s)$: x liegt im Zustand s auf y

Aktion: $move(x, y, z)$: bewege x von y auf z hin

Wirkung von Aktionen: **Effektaxiome**, z.B.: (+/−)

$$\forall x, y, z, s[on(x, y, s) \wedge clear(x, s) \wedge clear(z, s) \wedge (x \neq z) \rightarrow on(x, z, do(move(x, y, z), s))] \quad (+)$$

Nötig: **Frameaxiome** (was bleibt gleich), z.B.: (+/−)

$$\forall x, y, z, x', y', s[(on(x, y, s) \wedge (x \neq x')) \rightarrow on(x, y, do(move(x', y', z), s))] \quad (+)$$

x bleibt auf y, wenn ein von x verschiedenes Objekt x' bewegt wird

Hintergrundwissen

$$\forall s(clear(Fl, s))$$

$$\forall x \forall y \forall s(on(x, y, s) \wedge (y \neq Fl) \rightarrow \neg clear(y, s))$$

$$\forall x \forall s(\neg on(x, x, s))$$

Effektaxiome (Paare *on*, *move* und *clear*, *move*)

\+ $\forall x, y, z, s[on(x, y, s) \wedge clear(x, s) \wedge$
$clear(z, s) \wedge (x \neq z) \rightarrow on(x, z, do(move(x, y, z), s))]$

\- $\forall x, y, z, s[on(x, y, s) \wedge clear(x, s) \wedge clear(z, s) \wedge$
$(x \neq z) \wedge (y \neq z) \rightarrow \neg on(x, y, do(move(x, y, z), s))]$

\+ $\forall x, y, z, s[on(x, y, s) \wedge clear(x, s) \wedge (y \neq z)$
$clear(z, s) \wedge (x \neq z)) \rightarrow clear(y, do(move(x, y, z), s))]$

\- $\forall x, y, z, s[on(x, y, s) \wedge clear(x, s) \wedge clear(z, s) \wedge$
$(x \neq z) \wedge (z \neq Fl) \rightarrow \neg clear(z, do(move(x, y, z), s))]$

Frameaxiome (Paare *on*, *move* und *clear*, *move*)

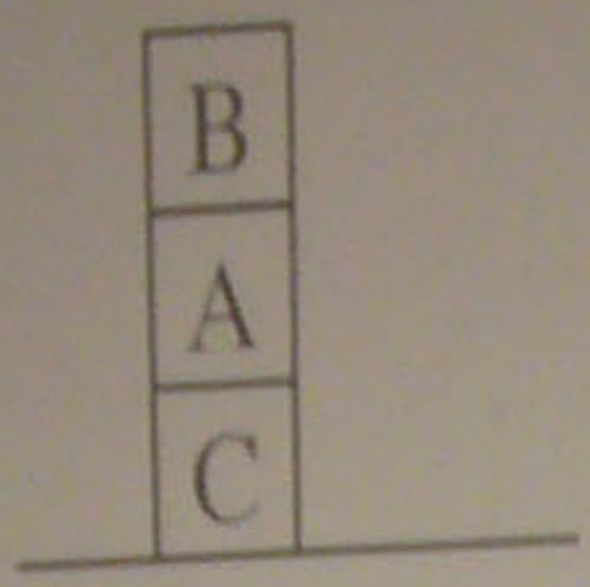
\+ $\forall x, y, z, x', y', s[(on(x, y, s) \wedge (x \neq x')) \rightarrow$
$on(x, y, do(move(x', y', z), s))]$

\- $\forall x, y, z, x', y', s[(\neg on(x, y, s) \wedge \neg(x = x' \wedge y = z)) \rightarrow$
$\neg on(x, y, do(move(x', y', z), s))]$

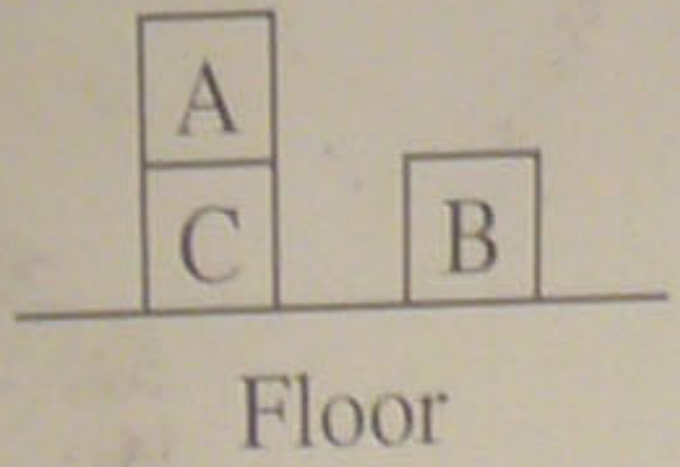
\+ $\forall x, y, z, x', s[(clear(x', s) \wedge (x' \neq z)) \rightarrow$
$clear(x', do(move(x, y, z), s))]$

\- $\forall x, y, z, x' s[(\neg clear(x', s) \wedge (x' \neq y)) \rightarrow$
$\ldots, do(move(x, y, z), s))]$

Aktionsausführung



On(B,A,S0)
On(A,C,S0)
On(C,Fl,S0)
Clear(B,S0)
Clear(Fl,S0)



Inferred using effect axioms:

On(B,Fl,do(move(B,A,Fl),S0))
¬On(B,A,do(move(B,A,Fl),S0))
Clear(A,do(move(B,A,Fl),S0))

Inferred using frame axioms:

On(A,C,do(move(B,A,Fl),S0))
On(C,Fl,do(move(B,A,Fl),S0))
Clear(B,do(move(B,A,Fl),S0))

True in all states:

(∀s)Clear(Fl,s)

Planen im Situationskalkül:

1. Formuliere „Goal" $g(s)$ (Formel, s ist Variable), das den Zielzustand beschreibt.

2. Beweise $\exists s.g(s)$, z.B. durch Resolution

   Antwort-Substitution θ für s:

$$do(\alpha_n, (do(\alpha_{n-1}, (\cdots (do(\alpha_1, S_0)) \cdots))))$$

   (S_0 = Anfangszustand) ergibt Plan:

$$P = \alpha_1, \alpha_2, \ldots, \alpha_n.$$

Probleme des Ansatzes

- Mangelnde Effizienz: großer Beweisaufwand, auch bei kleiner Beschreibung
- „Representational Frame Problem":
  Die Zahl der Frameaxiome proliferiert mit der Zahl der Aktionen/Fluents
- Qualifikationsproblem, Ramifikationsproblem