Kapitel 7: Produktionsregelsysteme

Thanks to Dr. Knut Hinkelmann, FH Solothurn, Schweiz

Literatur: Jackson

Produktionsregelsysteme

- Nachbildung des menschlichen Wahrnehmungsprozesses
- Modellierung menschlichen Problemlöseverhaltens

IF *Bedingung* THEN *Aktion*

- Pattern-directed Inference: Aktionen werden ausgelöst durch Muster, die in einer Situation gefunden werden
- Natürliche Ausdrucksform zur Darstellung von Expertenwissen:
  - Erfahrungen basieren auf Problemen, die der Experte früher gelöst hat
  - Diese Erfahrungen hat der Experte in Faustregeln abstrahiert
  - Wenn ein Experte mit einem neuen Problem konfrontiert wird, wendet er die passende Faustregel an
- Durch Produktionsregelsysteme können beliebige berechenbare Verfahren nachgebildet werden ([Post 1943])

Beispiel: „Management-Informations-System

WENN Auftragsvolumen > DM 100.000
DANN Kunde ist Großkunde

WENN Kunde ist wichtig
DANN Vertriebsleiter muß verständigt werden

WENN Kunde ist Großkunde
UND Bonität des Kunden ist gut
DANN Kunde ist wichtig

WENN Kunde ist wichtig
UND Engpaßaggregat ist ausgefallen
DANN kritische Unternehmenslage ist eingetreten

Kontrollstrategien zur Regelverarbeitung

PROLOG: zielgetriebene Inferenz

nun: **datengetriebene Inferenz:**

- ausgehend von gegebenen Fakten (Daten) werden Regelkonklusionen bewiesen
- wiederholen, bis ein Zielzustand erreicht ist oder keine neuen Wissenseinheiten mehr herleitbar sind
- Ableitung mehrerer (auch unbekannter) Zielzustände möglich
- aufwendige Konfliktlösungsstrategie erforderlich (welche Regel als nächstes anwenden?)
- Verwendung zur Erzeugung von Hypothesen oder wenn man wenn keine Vorstellung vom Zielzustand hat (z.B. Konfiguration)
- dagegen Verwendung der zielgetriebenen Inferenz zum Beweisen eines bestimmten Ziels (Beweisen einer Hypothese)

Beispiel: MYCIN (1976)
[XPS zur Diagnose baktrieller Infektionskrankheiten]

IF (1) the stain of the organism is granmeg, and
(2) the morphology of the organism is rod, and
(3) the aerobic of the organism is aerobic

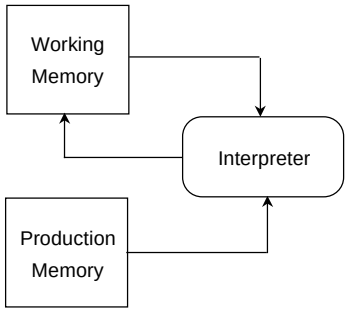
THEN there is a strongly suggestive evidence (0.8)
that the class of the organism is enterobacteriaceae

Beispiel: MYCIN (1976) , Forts.
[XPS zur Diagnose baktrieller Infektionskrankheiten]

IF (1) the infection is primary_bacteria, and
(2) the site of the culture is one of the sterile sites, and
(3) the suspected portal of entry is the gastrointestinal tract

THEN there is a suggestive evidence (0.7)
that the entity of the organism is bacteroides

Prinzipieller Aufbau von Produktionsregelsystemen



- Regeln sind die klassische, am weitesten verbreitete Repräsentationsform für Expertensysteme, z.B.
  - R1/XCON: Konfiguration von VAX Rechnern
  - MYCIN: Therapie bakterieller Infektionen
  - DENDRAL: Analyse organischer Verbindungen
- Ein Regelsystem besteht aus:
  - Working Memory (Datenbasis): Informationen, die über das aktuelle Problem bekannt sind.
    - Beispiel: In technischer Diagnose Informationen z.B. über die Maschine und bisher bekannte Symptome
  - Production Memory (Regelbasis): allgemeines Wissen über die Problemlösung in Form von Regeln. Jede Regel besteht aus einem Bedingungs- und einem Aktionsteil
    - Beispiel Diagnose: Regeln, die bei Vorliegen von Symptomen entsprechende Fehlerursachen in die Datenbasis eintragen
  - Interpreter: selektiert anwendbare Regeln und wendet sie an

Regelbasis: Verschiedene Typen von Regeln

- Enthält Menge von Regeln, die Reihenfolge der Regeln spielt keine Rolle
  IF <Bedingung> THEN <Aktion>
- **Bedingung:** Aussage über die Datenbasis, meist als Muster formuliert
  - Der *Wahrheitswert* der Bedingung muß berechnet werden können
- **Aktion:** Je nach dem, welche Aktionen erlaubt sind, kann man verschiedene Arten von Regelsystemen unterscheiden:
  - **Produktionsregeln:** als Aktionen sind (beliebige) Operationen erlaubt
    - Modifikation der Datenbasis: Hinzufügen, Löschen und Modifizieren von Elementen
    - Ausführung beliebiger externer Prozeduren, z.B. Ein-/Ausgabe, Arithmetik
    - Intendierte Bedeutung: *"Falls in der aktuellen Datenbasis die Bedingung erfüllt ist, führe die Aktion aus"*
  - **Plausibilitätsregeln:** Als Aktion ist nur Herleitung neuen Wissens erlaubt, das allerdings mit einem Konfidenzfaktor versehen ist
  - **Logische Regeln:** Die Regel wird als logische Implikation interpretiert
    <Bedingung> $\rightarrow$ <Konklusion>
    - Beispiel: Hornklauseln
    - Intendierte Bedeutung: *"Falls in der aktuellen Datenbasis die Bedingung wahr ist, dann ist auch die Konklusion wahr"*

Working Memory

- Das Working Memory enthält verschiedene Arten von Domänenwissen:
  - Allgemeines Wissen über Anwendungsgebiet, z.B. Konzept-/Klassenhierarchie
  - Wissen über die aktuelle Situation, z.B. Daten über das zu diagnostizierende Gerät
  - Informationen, die von dem System bereits abgeleitet wurden, und als Zwischenresultate in Working Memory gespeichert werden
  - Befragung des Benutzers, z.B. über fehlende Werte oder zur Beurteilung von Sachverhalten
- Je nach System sind die Einträge im Working Memory unterschiedlich repräsentiert, z.B.
  - Fakten
  - Frames
  - Semantische Netze
  - Attribut-Wert-Mengen (z.B. OPS5)
- Um Unsicherheit auszudrücken, können die Einträge mit Konfidenzfaktoren gewichtet sein

Working Memory in OPS5

```
Zeitmarke: 0  (block
                ^name       block2
                ^typ        quader
                ^laenge     150
                ^breite     100
                ^hoehe      100
                ^farbe      blau
                ^volumen    NIL)

Zeitmarke: 1  (block
                ^name       block1
                ^farbe      rot
                ^typ        quader
                ^laenge     100
                ^breite     100
                ^hoehe      100
                ^volumen    NIL)

Zeitmarke: 2  (ziel
                ^status     aktiv
                ^aufgabe    finden
                ^objekt     wuerfel
                ^farbe      rot)
```

- OPS5 ist ein bekanntes Produktionsregelsystem
- Repräsentation:
  - Objektzentrierte Attribut-Wert-Mengen (analog zu Frames):

    Element ::= (Name [^Attribut Wert]+)
  - Vektoren
  - Skalare (Zahlen und Symbole)
- Innerhalb eines Elements ist die Reihenfolge der Attribut-Wert-Paare ohne Bedeutung
- Attribute haben entweder Skalare oder Vektoren als Wert (keine Verweise auf andere Elemente)
- Jeder Eintrag ins Working Memory ist mit einer Zeitmarke versehen (wird automatisch vergeben)
- Keine zwei Elemente haben die gleiche Zeitmarke

Production Memory in OPS5: Bedingungen von Regeln

- Das Production Memory besteht aus einer ungeordneten Menge von Produktionsregeln
- Eine Produktionsregel besteht aus Name, Bedingungsteil (left-hand-side – LHS) und Aktionsteil (right-hand-side – RHS)

  Produktionsregel ::= (p Produktionsname [Bedingung]+ --> [Aktion]+)
- Der Bedingungsteil ist zu lesen als Konjunktion von positiven und negativen Bedingungen.
- Jede Bedingung ist ein Muster für ein Element des Working Memory:
  - Bedingung ::= (Name Term+) | -(Name Term+)
  - Term ::= ^Attribut Wertbeschreibung | ^Zahl Wertbeschreibung | Wertbeschreibung
  - Wertbeschreibung ::= Konstante | Variable | Literal | Disjunktion | Konjunktion
  - Variable ::= <Name>
  - Literal ::= Prädikat Konstante | Prädikat Variable
  - Prädikat ::= = | <> | <=> | <= | < | > | >=
  - Disjunktion ::= << Wertbeschreibung+ >>
  - Konjunktion ::= { Wertbeschreibung+ }
- Die Reihenfolge der Terme eines Bedingungselements muß nicht mit derjenigen im Working Memory übereinstimmen

Production Memory in OPS5: Aktionen in Regeln

- Der Aktionsteil ist eine Sequenz von Kommandos
- Folgende Aktionen sind in OPS5 definiert:
  - Aktionen, die das Working Memory ändern:
    - `make` Erzeugung eines neuen Elements
    - `remove` Löschen eines Elements
    - `modify` Modifikation eines Elements durch Überschreiben von Attributwerten (entspricht einer Sequenz von `remove` und `make`)
  - Aktion zur Modifikation des Production Memory
    - `build` Hinzufügen neuer Produktionsregeln
  - Aktionen zur Ein-/Ausgabe: `openfile, closefile, write`
  - Aktionen zur Variablenbindung: `bind, cbind`
  - Auswertung von Prozeduren: `call`
  - Terminierung des Prozesses: `halt`
- Zur Berechnung von Attributwerten stehen Funktionen zur Verfügung, z.B. zur Selektion von Attributwerten eines Elements, zur Auswertung arithmetischer Ausdrücke und zur Generierung neuer Symbole

Beispiel: Production Memory in OPS5

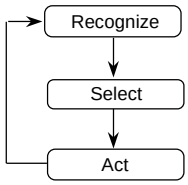
```
(p finde-farbiges-objekt
      (ziel   ^status     aktiv
              ^aufgabe    finden
              ^objekt     <x>
              ^farbe      <y>)
      (block  ^farbe      <y>
              ^name       <block>
              ^typ        <x>)
-->   (make  result  ^wert  <block>)
      (modify  1  ^status  erfuellt))

(p wuerfel-volumen
      (block  ^laenge     <x>
              ^breite     <x>
              ^hoehe      <x>)
--> (modify  1
              ^typ  wuerfel)
              ^volumen  (compute <x> * <x> * <x>))))

(p quader-volumen
      (block  ^laenge     <x>
              ^breite     <y>
              ^hoehe      <z>)
--> (modify  1 ^volumen  (compute <x> * <y> * <z>))))

(p ziel-erreicht
      (ziel   ^status     erfuellt)
-->   halt)
```

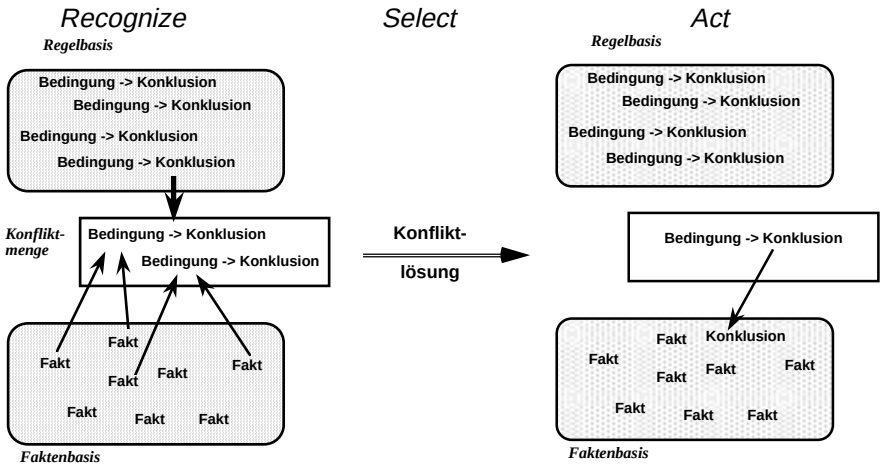

Vorwärtsverkettung: Recognize-Select-Act Zyklus




```
Prozedur Produktion
Konfliktmenge  <– {}
UNTIL WM erfüllt
      Terminierungsbedingung
BEGIN
Konfliktmenge <– Menge der Regeln,
                 deren Bedingung
                 durch Elemente des
                 WM erfüllt ist
R <– select{Konfliktmenge}
WM <– WM ∪ Aktion(R)
END
```

- Vorwärtsverkettung ist die Standardauswertung für Produktionssysteme
- Datengetriebene Verarbeitung: Wenn alle Bedingungen einer Regel durch Fakten der Wissensbasis erfüllt sind, dann führe die Aktion aus.
- Die Vorwärtsverkettung wird durch die wiederholte Ausführung von drei Aktionen realisiert: Recognize-Select-Act Zyklus
  - ***Recognize***: Finde alle Regeln, deren Bedingungsteil durch Einträge im Working Memory (WM) erfüllt sind. Ergebnis ist die Menge anwendbarer Regeln, die man als Konfliktmenge bezeichnet
  - ***Select***: Wähle aus der Konfliktmenge eine Regel R aus (Konfliktlösung).
  - ***Act***: Führe die Aktion der selektierten Regel aus.
- Terminierungsbedingungen: Die datengetriebene Auswertung terminiert, wenn
  - keine Regel mehr anwendbar ist, oder
  - wenn die Aktion "halt" ausgeführt wird

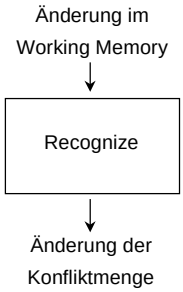
Recognize-Select-Act Zyklus



Regelauswertung in OPS5

1. **Recognize**: Werte den Bedingungsteil LHS aller Regeln aus und instantiiere die positiven Bedingungen mit Elementen des aktuellen Working Memory
   - Für positive Bedingungen werden Elemente des Working Memory gesucht, die das Muster erfüllen
   - Negative Bedingungen werden entsprechend der Closed-World Assumption (CWA) ausgewertet: eine negative Bedingung ist erfüllt, wenn kein dem Muster entsprechendes Element im Working Memory enthalten ist
   - Jede Variable wird überall mit dem gleichen Wert belegt
   - Falls eine LHS auf verschiedene Arten instantiiert werden kann, dann werden alle Möglichkeiten gefunden.
2. **Select** (Konfliktlösung): Wähle eine Produktionsregel mit erfüllter LHS. Falls keine solche Regel existiert, dann stop.
3. **Act**: Werte die Aktionen der RHS aus
   - Die Aktionen werden sequentiell ausgeführt. Bei Änderungen im Working Memory werden die Zeitmarken der davon betroffenen Elemente aktualisiert.
4. Falls die Aktion halt ausgeführt wurde, dann stop;
   sonst gehe zu 1.

Recognize: Many Pattern / Many Object Matching

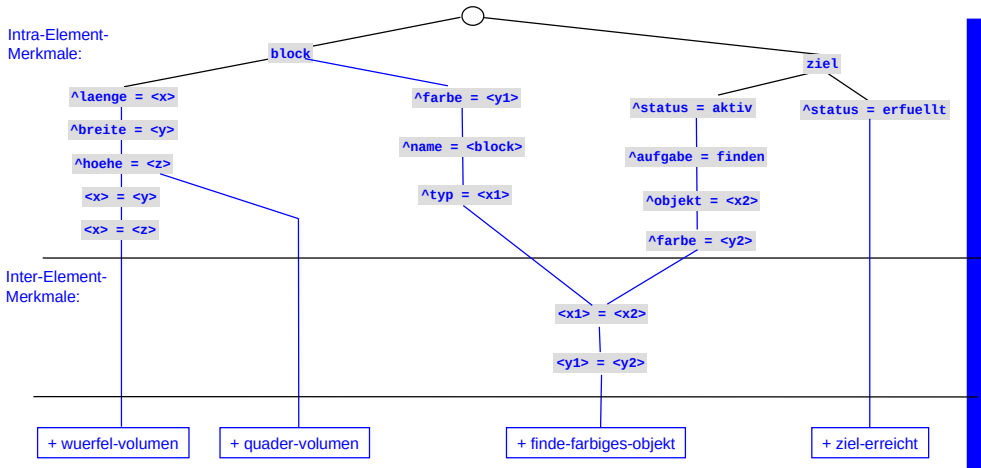


- Die Recognize Phase ist die aufwendigste Phase bei der Regelauswertung
- Naive Lösung: Vergleiche jeweils alle Regeln mit allen Elementen des Working Memory und berechne jeweils die vollständige Konfliktmenge
- Annahme für Optimierungen: Produktionssysteme sind zeitlich redundant, d.h. jede Anwendung einer Regel ändert nur einen kleinen Teil des Working Memory (so daß es auch nur wenig Änderungen in der Konfliktmenge gibt)
- Es gibt verschiedene Techniken zur Implementierung des many pattern/many object matching
  - Rete
  - TREAT
  - Lazy Matching

Rete Match Algorithmus: Idee [Forgy, 1982]

- Ziel: Vermeidung von Iterationen über das Production Memory
- Vorgehensweise: Die Bedingungen aller Regeln werden in ein Netzwerk übersetzt
  - Jeder Knoten des Netzwerks entspricht dem Test genau eines Merkmals
  - Kein Knoten und damit auch kein Test taucht mehrfach in dem Netzwerk auf ("Sharing")
  - Jede Änderung des Working Memory wird an das Netzwerk übergeben, wo sie entlang der Kanten weitergeleitet und die Tests durchgeführt werden. Als Ergebnis werden die Änderungen der Konfliktmenge ausgegeben
  - In dem Rete-Netzwerk werden zuerst die Intra-Element Merkmale getestet: Elemente die dieses Kette von Tests passieren, erfüllen einzelne Bedingungen der Regeln
- Auf der zweiten Ebene werden die Inter-Element Merkmale getestet, d.h. es wird auf konsistente Variablenbindungen zwischen je zwei Bedingungen einer Regel überprüft

Beispiel für Rete-Netzwerk: Prinzip



Konfliktlösung

- Konfliktlösungsstrategie: Sortierung der Konfiktmenge
- Kriterien für Konfliktlösung:
  - Vermeidung trivialer Scheifen: Regelinstanzen, die bereits früher ausgewertet wurden, werden aus der Konfliktmenge gelöscht
  - Bevorzugung von Regeln, die sich auf das aktuellste Datenelement beziehen. Dies wird an der Zeitmarke erkannt. Dadurch erhält man eine Art Tiefensuche, da eine einmal begonnene Teilaufgabe weitergeführt wird.
  - Bevorzugung von Regeln mit der speziellsten Bedingung (eine Bedingung ist spezieller, wenn mehr Tests notwendig sind). Da diese in weniger Fällen anwendbar sind, sind sie in den Fällen, in denen sie anwendbar sind, wohl am geeignetsten.

Was heißt „die Regel mit den neuesten Datenelementen“?

- Regel 1: 5 7 2 4 5 8 9
- Regel 2: 5 7 6 4 4 1 3
- die Strategie **MEA** bevorzugt Regel 1, weil in Reihenfolge der Notation in Regel 1 zuerst ein neueres Datenelement auftaucht.

- Regel 1: 5 7 2 4 5 8 9
- Regel 2: 5 7 6 4 4 1 3
- die Strategie **LEX** bevorzugt Regel 2, weil bei Sortierung nach Aktualität die zweite Regel zuerst ein neueres Element besitzt

Beispiel für Regelauswertung

Strategie für Konfliktlösung: Wähle Regel, die vom neuesten Element erfüllt wird. Wenn mehrere Regeln vom neuesten Element erfüllt werden, wähle die Regel mit der speziellsten Bedingung

- **Schritt 1:** **Recognize**: Anwendbare Regeln (Konfliktmenge):
  - `wuerfel-volumen` mit `<x> = 100` (Element: block1)
  - `quader-volumen` mit `<x> = 100, <y> = 100, <z> = 100` (Element: block1)
  - `quader-volumen` mit `<x> = 150, <y> = 100, <z> = 100` (Element: block2)

  Select: `wuerfel-volumen` mit `<x> = 100`
  (`block1` ist neuer, Bedingung von `wuerfel-volumen` ist spezieller als von `quader-volumen`)

  Act: Für `block1` wird `^typ wuerfel` und `^volumen 1.000.000` eingetragen (neue Zeitmarke: 3)
- **Schritt 2:** **Recognize**: Anwendbare Regeln (Konfliktmenge):
  - `wuerfel-volumen` mit `<x> = 100` (Element: block1) (bereits angewendet!)
  - `quader-volumen` mit `<x> = 100, <y> = 100, <z> = 100` (Element: block1)
  - `quader-volumen` mit `<x> = 150, <y> = 100, <z> = 100` (Element: block2)
  - `finde-farbiges-objekt` mit `<x> = wuerfel, <y> = rot` (Elemente: block1, ziel)

  Select: `finde-farbiges-objekt` mit `<x> = wuerfel, <y> = rot, <block> = block1`
  (`block1` ist das neueste Objekt, da im vorigen Schritt modifiziert)

  Act: Ein neues Objekt (`result ^wert <block>`) wird erzeugt (Zeitmarke: 4)
  Für `ziel` wird `^status erfuellt` modifiziert (neue Zeitmarke: 5)
- **Schritt 3:** **Recognize**: Anwendbare Regeln (Konfliktmenge):
  - `wuerfel-volumen` mit `<x> = 100` (Element: block1) (bereits angewendet!)
  - `quader-volumen` mit `<x> = 100, <y> = 100, <z> = 100` (Element: block1)
  - `quader-volumen` mit `<x> = 150, <y> = 100, <z> = 100` (Element: block2)
  - `ziel-erreicht` (Element: ziel)

  Select: `ziel-erreicht` (`ziel` ist das neueste Objekt, da im vorigen Schritt modifiziert)

  Act: `halt`

Aufgabe: „Arbeitslose Pendler“ :-)

```
(strategy LEX)
(literalize person pnr name vorname)
(literalize adresse pnr strasse hausnr plz stadt)
(literalize mitarbeiter pnr firma beruf)
(literalize firma firma plz stadt)

(p start (start) ->
    (make firma ^firma BASF ^stadt LU)
    (make mitarbeiter ^pnr 1 ^firma BASF)
    (make person ^pnr 1 ^name Maier)
    (make adresse ^pnr 1 ^stadt LU)
    (make mitarbeiter ^pnr 2 ^name Müller)
    ...
)
```

```
(defun myrun ()
        make start)
        (run))
```

Definiere Regeln für:

- arbeitslose Personen
- Berufspendler

Aufgabe: Fibonacci-Zahlen

```
(literalize fibo index wert)

(p begin (start) ->
    (make fibo ^index 1 ^wert 1)
    (make fibo ^index 2 ^wert 1)
    (remove 1)
)
```

```
(defun myrun ()
        make start)
        (run))
```

Regel zur Berechnung aller Fibonacci-Zahlen?

Programmierstil bei Regelsprachen ist inkrementell

- Neue Fälle werden neue Regeln
- Verfeinerung der Wissensbasis:
  - füge spezialisierte Regel hinzu
  - ersetze eine Regel durch mehrere neue (Fallunterscheidung)
- Generalisierung: Bedingungen abschwächen
- einfache Kontrollmöglichkeiten über Kontrollfakten

- Achtung: Regelprogrammierung tendiert zum **Chaos**
- Abhilfe-Ansätze:
  - übergeordnete Kontrolle z.B. durch Kontexte / Kopplung mit Frames / ...
  - Automatische Verifikation und Validierung on Regelbasen
- grundsätzlich: **mehr Wissen reinpacken!!**

Metaregeln in MYCIN zur effektiven Verwertung von Strategiewissen

IF **(1) Kultur stammt aus steriler Quelle, und**
(2) es gibt Regeln, die in ihrer Prämisse einen früheren Organismus erwähnen, der derselbe sein könnte wie der aktuelle

THEN **es ist definitiv, daß beide Regeln nicht nützlich sind (1.0)**

-> Aussortierung von Regeln mit Wissen aus dynamischer, aktueller Fallbetrachtung und Reasoning über Inhalt von Regeln in der Konfliktmenge

Metaregeln in MYCIN zur effektiven Verwertung von Strategiewissen, Forts.

IF (1) die Infektion ist ein Beckenabszeß, und
(2) es gibt Regeln, die in ihrer Prämisse enterobacteriaceae erwähnen, und
(3) es gibt Regeln, die in ihrer Prämisse gram-positive Stäbchen erwähnen

THEN es ist sinnvoll (0.4), die ersteren Regeln vor letzteren anzuwenden

-> Ordnen von Regeln mit Domänenwissen, im Gegensatz zur allgemeinen Konfliktlösungsstrategie

Metaregeln in MYCIN zur effektiven Verwertung von Strategiewissen, Forts. 2

IF (1) es gibt Regeln, die in ihrer Prämisse das aktuelle Teilziel nicht erwähnen, und
(3) es gibt Regeln, die das aktuelle Teilziel erwähnen

THEN es ist definitiv (1.0), daß letztere vor ersteren ausgewertet werden sollten

-> allgemeine Heuristik, wird zur Laufzeit ausgewertet

Etwas zur Leistungsfähigkeit von MYCIN (nach Jackson)

- 1979: Vergleich von MYCIN und Stanford Uni-Ärzten im Bereich bakterielle Anämien / Meningitis
- 10 echte Fälle
- Bewertung der vorgeschlagenen Therapie durch unabhängige Experten
- max. 80 Punkte erreichbar

MYCIN	52 Punkte
Fakultät 1	50 Punkte
Fakultät 2	48 Punkte
Inf. Experte	48 Punkte
Fakultät 3	46 Punkte
Fakultät 4	44 Punkte
Student	24 Punkte

Der Einsatz von KI-Sprachen ist empfehlenswert, wenn ...

- Suchprobleme vorliegen
  - mit logischen Zusammenhängen, oder
  - mit großen Suchräumen, oder
  - mit sehr vielen Einzelfällen und vielen Kombinationen dieser Fälle
- Probleme begriffliche Argumentationen erfordern („wenn der Auspuff qualmt, ist oft die Ölabdichung des Motors defekt“)

Der Einsatz von KI-Sprachen kann unangemessen / unnötig sein, wenn ...

- Einfache Operationen auf große Datenmengen anzuwenden sind (Buchungssystem, Bildverarbeitung)
- bereits optimale Lösungen in anderen Programmiersprachen existieren (z.B. teilweise im Operations Research)
- Probleme hardwarenah gelöst werden müssen (Betriebssystem, Schnittstellenprotokolle)
- numerische Problemstellungen vorliegen (Matrix-Operationen, Strömungsberechnung, ...)

Heutige Bedeutung von Vorwärtsregelsprachen