

Einführung in die KI

Prof. Dr. sc. Hans-Dieter Burkhard
Vorlesung Winter-Semester 2004/05

Wissensrepräsentation 2:
Strukturierung
Regelsysteme
Frames, Skripts
Beschreibungssysteme

Strukturen für die Wissensrepräsentation

- Regelsysteme:
„Wenn - dann“ - Zusammenhänge
- Terminologische Repräsentation
Begriffshierarchien, -netze
- Semantische Netze, Frames,
– Beschreibungssysteme
- Skripts
Ereignisse beschreiben

H.D.Burkhard, HU Berlin
Winter-Semester 2004/05

Vorlesung Einführung in die KI
Wissensrepräsentation (Regeln)

2

3.1 Regelsysteme („Produktionssysteme“)

Beschreibung von

„Wenn - dann“ - Aktionen

IF Bedingung(en) THEN Aktion(en)

linke Seite

rechte Seite

left hand side (LHS)

right hand side (RHS)

IF *Ball weit weg* THEN *Home-Position einnehmen*
IF *Bein gebrochen* THEN *ruhig stellen*
IF *neuer Kunde* THEN *Statistik aktualisieren*

H.D.Burkhard, HU Berlin
Winter-Semester 2004/05

Vorlesung Einführung in die KI
Wissensrepräsentation (Regeln)

3

Regelanwendung IF Bedingung(en) THEN Aktion(en)

1. Matchen:

Auslösebedingung(en) erfüllen Bedingung(en) der linken Seite,
durch Variablen-Bindungen entsteht *Regelinstanz*

2. Abarbeiten („Feuern der Regel“)

Aktionen der rechten Seite der *Regelinstanz* ausführen

IF *Spieler X frei* THEN *Pass zu Spieler X*

IF *Bestellung X von Kunde K UND X im Lager*
THEN *Lieferauftrag X an K*

IF *Bestellung X von Kunde K UND NICHT X im Lager*
THEN *Absage X an K UND Nachbestellung X*

H.D.Burkhard, HU Berlin
Winter-Semester 2004/05

Vorlesung Einführung in die KI
Wissensrepräsentation (Regeln)

4

Einfaches („1-schichtiges“) Regelsystem

Tabelle von Regeln

Ablauf:

Triggerbedingungen kommen „von außen“,
Mit passender Regel: Antwort „nach außen“

R1: IF *Bedingung(en)-1* THEN *Aktion(en)-1*
R2: IF *Bedingung(en)-2* THEN *Aktion(en)-2*
R3: IF *Bedingung(en)-3* THEN *Aktion(en)-3*
R4: IF *Bedingung(en)-4* THEN *Aktion(en)-4*
R5: IF *Bedingung(en)-5* THEN *Aktion(en)-5*
R6: IF *Bedingung(en)-6* THEN *Aktion(en)-6*
.....
Rk: IF *Bedingung(en)-k* THEN *Aktion(en)-k*

Bedingung(en)

Bedingung(en) matchen

Bedingung(en) matchen

Aktion(en)

H.D.Burkhard, HU Berlin
Winter-Semester 2004/05

Vorlesung Einführung in die KI
Wissensrepräsentation (Regeln)

5

Mehrschichtige Regelsysteme

Beschreibung komplexer Prozessen mit Regeln.
Abarbeitung einer Regel aktiviert weitere Regeln.

IF *Anfrage X von Kunde K*
THEN *Überprüfung Kunde K UND Analyse Anfrage X*

IF *Überprüfung Kunde K UND NOT K in Kundendatei*
THEN *Eintrag K in Kundendatei*

IF *Analyse Anfrage X UND X = STORNO*
THEN *STORNO-Meldung . . .*

Benötigt internen Arbeitsspeicher
und Interpreter (komplexe Inferenz-Maschine)

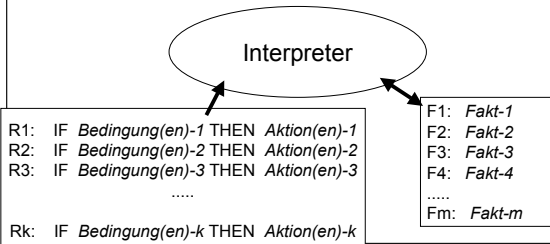
H.D.Burkhard, HU Berlin
Winter-Semester 2004/05

Vorlesung Einführung in die KI
Wissensrepräsentation (Regeln)

6

Komponenten eines Regelsystems

- Regelmenge (Programm)
- Arbeitsspeicher (Wissens-Zustand: Fakten-Menge)
- Interpreter/Ablaufsteuerung (Inferenz-Maschine)



H.D.Burkhard, HU Berlin
Winter-Semester 2004/05

Vorlesung Einführung in die KI
Wissensrepräsentation (Regeln)

7

Arbeitsspeicher

Datenbasis (Menge von Fakten) :
Arbeitsspeicher, „Working-Memory“
Aktueller interner Wissens-Zustand

F1: Fakt-1
F2: Fakt-2
F3: Fakt-3
F4: Fakt-4
.....
Fm: Fakt-m

Fakten: „Working-Memory-Elements“ (WME)

Form: Datenbankeinträge, Prädikate, Zeitstempel.

Veränderung durch Aktionen bei der Regelabarbeitung,
z.B.

REMOVE (Fakt)
ADD (Fakt)
CHANGE (Fakt1, Fakt2)

H.D.Burkhard, HU Berlin
Winter-Semester 2004/05

Vorlesung Einführung in die KI
Wissensrepräsentation (Regeln)

8

Regelanwendung IF Bedingung(en) THEN Aktion(en)

1. Matchen: Regel(-instanz) ist „feuerbar“.

Falls LHS durch Fakten im Arbeitsspeicher erfüllbar:
Regelinstanz R (= Regel + Belegung der Variablen)

2. Abarbeiten („Feuern“):

Aktionen der Regelinstanz R ausführen:

- Änderungen im Arbeitsspeicher
- Seiteneffekte (E/A-Operationen usw.)

Mehrere Regeln können matchen.

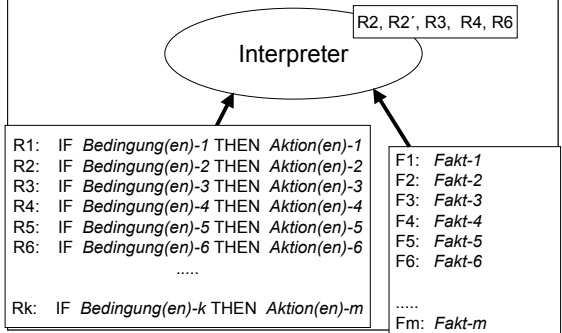
Zu einer Regel können mehrere Instanzen existieren.

H.D.Burkhard, HU Berlin
Winter-Semester 2004/05

Vorlesung Einführung in die KI
Wissensrepräsentation (Regeln)

9

Interpreter: Matchen



H.D.Burkhard, HU Berlin
Winter-Semester 2004/05

Vorlesung Einführung in die KI
Wissensrepräsentation (Regeln)

10

Interpreter: Recognize-Act-Zyklus

Recognize-Act-Zyklus in 3 Phasen

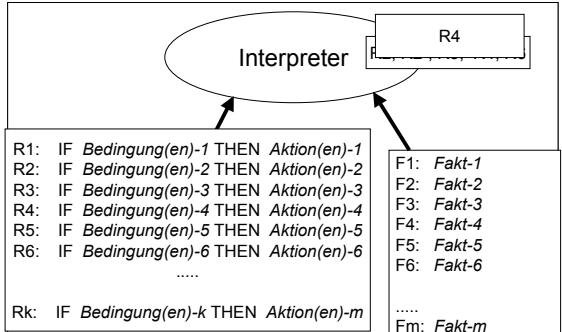
1. MATCHING:
Feuerbare Regelinstanzen $R_1, \dots, R_n$ bestimmen
2. CONFLICT RESOLUTION
Auswahl einer Regelinstanz R aus $R_1, \dots, R_n$
3. ACTION:
Abarbeiten der ausgewählten Regelinstanz R

H.D.Burkhard, HU Berlin
Winter-Semester 2004/05

Vorlesung Einführung in die KI
Wissensrepräsentation (Regeln)

11

Interpreter: Conflict Resolution

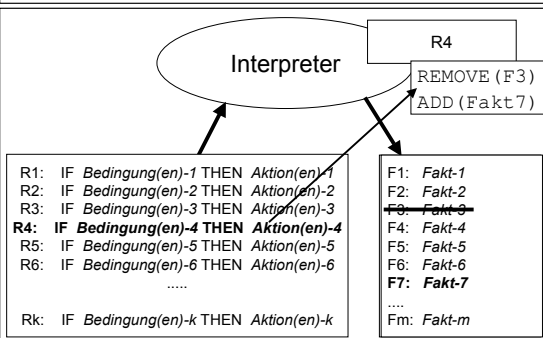


H.D.Burkhard, HU Berlin
Winter-Semester 2004/05

Vorlesung Einführung in die KI
Wissensrepräsentation (Regeln)

12

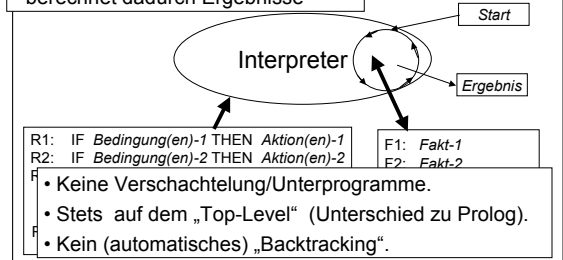
Interpreter: Action



Arbeit des Regelsystems

Zyklische Arbeit des Interpreters:

- manipuliert Arbeitsspeicher und
- berechnet dadurch Ergebnisse



Arbeit des Interpreters als Suchstrategie

- Zustände: Fakten-Mengen (Arbeitsspeicher-Zustände)
 - Anfangszustand: initiale Fakten-Menge
 - Zielzustände: durch Resultate bestimmt
- Operatoren: Regeln, Anwendbarkeit abhängig vom Zustand

Vorwärts- oder Rückwärts-Strategien möglich
(Forward/backward chaining)

Suchstrategie:

- Verfolgen eines einzelnen (!) Pfades
- Auswahl gemäß „Conflict-Resolution“ im Interpreter
- kein Verfolgen mehrerer Pfade
(fehlt: Liste OPEN bzw. Backtracking)
- evtl. durch Regel-Programm simulieren

Conflict-Resolution

Mögliche Prioritäts-Kriterien:

- Regel/Regelinstanz nicht wiederholt ausführen

⇒ Schleifen vermeiden

- Jüngere Regelinstanzen bevorzugen

⇒ Tiefen-Suche.

Alternativ: Ältere Regelinstanzen bevorzugen

⇒ Breiten-Suche.

*Alter einer Regelinstanz berechnen
aus dem Alter der Fakten (WME),
die den Bedingungsteil der Instanz matchen*

Conflict-Resolution

Mögliche Prioritäts-Kriterien:

- Regeln mit vielen Bedingungen bevorzugen

⇒ Ausnahme-Behandlung.

zusätzliche Ausnahmebedingungen

- Metaregeln bevorzugen

⇒ Strukturierung von Regel-Programmen

Kontexte für Regeln einführen (als Fakten)

Ausnahmen von allgemeinen Regeln

- Allgemeine Regel („Default“):
IF in_Bibliothek(x) THEN arbeitet(x)
- Ausnahme-Regeln
IF in_Bibliothek(x) AND muede(x) THEN schlaeft(x)
- Ausnahme von der Ausnahme
IF in_Bibliothek(x) AND muede(x) AND vorKlausur(x)
THEN arbeitet(x)

Ausnahme-Behandlung (Regeln mit vielen Bedingungen)
hat Vorrang

Einfaches Verfahren für nichtmonotones Schließen!

Strukturierung von Regel-Programmen

Problem:

Regeln gleichberechtigt - Ablauf schwer durchschaubar.

Strukturierung durch Gruppierung mittels *Kontexten*:

```
IF context1 AND ... THEN ...  
IF context1 AND ... THEN ...
```

```
IF context2 AND ... THEN ...  
IF context2 AND ... THEN ...
```

Metaregeln zum Setzen von Kontexten:

```
IF ... THEN ADD context1  
IF ... THEN REMOVE context2
```

Rücksetzregeln für Kontexte nutzbar für Backtracking

Grundidee Regelbasierter Systeme

- Regeln:
 - mitteilbares Wissen oft in Regelform
 - wegen leichter Anwendbarkeit von hoher Bedeutung
- Faktenmenge = aktueller Wissensstand,
- Matching mit Datenbasis steuert Ablauf

Eignung für Probleme mit

- vielen, unabhängigen Einzelaktionen/Fakten
- Separierbarkeit von Wissen und Bearbeitung

Eigenschaften Regelbasierter Systeme

- Einheitlichkeit
- Erklärbarkeit
- Erweiterbarkeit,
- Unabhängigkeit
- wenig Strukturierung

Nachteile bei großen Systeme

- Abarbeitungsreihenfolge schwer überschaubar,
 - Steuerung implizit
 - Komplexität
- 1000 Regeln, 10 Tests pro Regel, 500 WME.
Theoretisch: 5 Mio. Tests pro Zyklus.

OPS-5

„Official Production System“

entwickelt zur Konfiguration von VAX-Rechnern:
Expertensysteme R1/XCON Firma DEC

Effizienz durch Compilation/Abarbeitung mit Rete-Algorithmus:

- Programm (Bedingungen) in netzartiger Struktur
- Markierung von Bedingungen gemäß Arbeitsspeicher
- Markierungfeuerbarer Regelinstanzen

Programm wird in *Netzartige Struktur* überführt,
Matching nicht durch Vergleich aller Regeln mit
Arbeitsspeicher, sondern durch Management der
Änderungen bzgl. feuerbarer Regelinstanzen.

Datenstrukturen („Klassen“) für WME

Der Arbeitsspeicher - **Working memory** –
besteht aus **WME's** mit „Zeitstempeln“
(fortlaufende Numerierung entsprechend der Erzeugung).

Der Datenbestand kann mit der Funktion
wm sichtbar gemacht werden.

Ein **WME** besteht aus

- einem (Klassen-)Namen
- einer Menge von *Attribut-Wert-Paaren*
- einem *Zeitstempel*

(block ^name A ^size 10 ^place heap) 987

Speichertechnisch:
Attribute durch Stellung spezifiziert,
nur Werte abspeichern

Datenstrukturen („Klassen“) für WME

Deklaration einer Klasse mit Hilfe von literalize:

- Angabe des Klassen-Namens und
- einer (beliebigen) Anzahl von Attributen.

literalize block name size place

Erzeugung eines WME mit make.

(make block ^name A ^size 10 ^place heap)

erzeugt zur Zeit 987 das WME

(block ^name A ^size 10 ^place heap) 987

Vereinfachte Syntax

```
CE = condition element
AV-pair = attribut-value-pair

<regel> ::= ( p <regelname> <LHS> --> <RHS> )
<LHS>   ::= <CE> { [-] <CE> }
<RHS>   ::= {<action>}
<CE>    ::= ( <class name> { <AV-pair> } )
<AV-pair> ::= ~ <attribut> <value>

<action> ::= (make <class name> {<AV-pair>} )
           | (remove <number> )
           | (modify <number> {<AV-pair> } )
           | (call <function name> )
           | (write ... )
           | (halt)
           | (build ... - f"ur Regeln - )
           | ...
```

Variable

- treten nur in Regeln auf,
- ohne Typvereinbarung (es gibt aber z.B. arithm. Ausdr.),
- haben eine auf die Regelabarbeitung beschränkte Lebensdauer und
- sind, falls einmal belegt, in der ganzen Regelinstanz so belegt (Konsistenz).
- Bei negierten Bedingungen (-<cond_element>) erfolgt keine Variablenbindung. Negierte Bedingungen sind erfüllt, falls kein WME matcht.

„value“ kann eine Variable oder ein arithm./logischer Ausdruck sein:

{ > 5 < 10 } UND-Verknüpfung: x > 5 und x < 10

<< < 5 > 10 >> ODER-Verknüpfung: x < 5 oder x > 10

Matching WME und CE

Der Bezug auf die Attribut-Werte-Paare erfolgt in den CE durch die Attribut-Namen, dabei müssen nicht alle Paare aus der angegebenen Klasse vorkommen, auch die Reihenfolge kann verändert werden.

Matching WME und CE

Das Matching eines CE mit einem WME ist erfolgreich,

- falls die Klassen-Namen übereinstimmen und
- die im CE angegebenen Attr.-Werte-Paare mit den entsprechenden Werten des WME matchen, d.h.:
 - die Werte stimmen überein oder
 - im CE steht eine ungebundene Variable <x> :
x erhält den entspr. Wert aus dem WMEoder
- im CE steht eine Bedingung,
die der Wert aus dem WME erfüllt
oder ... (weitere Bedingungen siehe Literatur)

Aktionen der rechten Seite:

- make erzeugt ein entsprechendes WME (mit fortlaufendem Zeitstempel).
- remove i entfernt das WME, das der i-ten Bedingung der LHS entspricht.
- modify i ... ersetzt ein WME:
das WME, das der i-ten Bedingung entspricht, wird aus dem Working Memory gestrichen, ein WME mit neuem Zeitstempel wird aufgenommen, das sich von dem alten in den angegebenen Attributen unterscheidet.
- (weitere: siehe Lit.)

Abarbeitung

Feuerbare Regelinstanzen:

Jeweils Regel mit einem „Satz“ matchender WMEs aus dem aktuellen Speicher.

Bestimmung der feuerbaren Regelinstanzen: Alle Bedingungen (<CE>) der LHS einer Regel müssen mit geeigneten WME's „matchen“ (s.o.).

Für die Konflikt-Lösung sind die Zeitstempel maßgeblich, dafür gibt es zwei Strategien (LEX und MEA), die am Programmanfang gesetzt werden (s.u.).

Konfliktlösungsstrategien

- Die Konfliktlösung erfolgt in mehreren Schritten bis nur eine Instanz übrig bleibt.
- In jedem Schritt wird i.a. eine Halbordnung über den Instanzen erzeugt, alle nicht dominierenden Instanzen werden gestrichen.
- Sobald in einem (Teil-)Schritt nur eine Regelinstanz übrig ist, wird diese genommen.

Instanz =
Regel
+ Variablenbelegung
+ Menge der Zeitstempel der „matchenden“ WME

Strategie LEX

(„lexikographische“ Ordnung, s. Schritt 2.)

1. Schritt : Refraction

- Regelinstanzen, die gefeuert haben, streichen.
- Falls danach die Regelinstanzenmenge leer ist: STOP.

Bemerkung:
Gleichartige Instanzen mit unterschiedlichen
Zeitstempeln sind verschieden.

Die Regel (p loop (start) --> (make start)) wäre
immer wieder feuerebar als neue Instanz.

Strategie LEX

2. Schritt: Recency und Recency Specificity

- Sei n_1 der größte (jüngste) Zeitstempel, der bei den Regelinstanzen der Konfliktmenge vorkommt: Regelinstanzen, bei denen n_1 nicht vorkommt, streichen.
- Sei n_2 der zweitgrößte Zeitstempel in der aktuellen Konfliktmenge. Regelinstanzen, bei denen n_2 nicht vorkommt, streichen. (Einschließlich derjenigen, mit nur einem CE: „Specify“.)
- usw. solange mehr als 1 Instanz in der Konfliktmenge, bzw. wenn keine Einschränkung mehr möglich: weiter bei 3.

Strategie LEX

3. Schritt Test-Specificity

Instanzen mit den meisten Tests in den Bedingungen auswählen. (Variablenbindungen gelten nicht als Test.)

4. Schritt zufällige Auswahl.

Strategie MEA

(means-ends-analysis)

Das erste CE jeder Regel hat besonderes Gewicht.

Abarbeitung wie bei LEX, außer bei:

2. unterteilt sich in

- (a) Maßgeblich ist der maximale Zeitstempel t_1 der ersten Bedingung einer jeden Regelinstanz: Alle Instanzen streichen, bei denen die erste Bedingung mit einem älteren WME (mit Zeitstempel $t_1' < t_1$) matcht
- (b) weiter wie bei LEX

Durch die MEA-Strategie kann der
Ablauf eines Programms besser bestimmt werden.
(Setzen von Kontexten - siehe im Beispiel: (goal ...))

Beispielprogramm
nimmt Blöcke der
Größe nach von
einer Halde und
plaziert sie der
Größe
entsprechend.

```
(strategy mea)

(literalize goal task index)
(literalize block name size place)
(literalize start)

(p begin
 (start)
 -->
 (make block ^ name A ^ size 10 ^place heap)
 (make block ^ name B ^ size 20 ^place heap)
 (make block ^ name C ^ size 30 ^place heap)
 (make block ^ name D ^ size 40 ^place heap)
 (remove 1)
 (make goal ^ task add ^ index 1)
)

(p pick-up
 (goal ^ task add)
 (block ^ size <size> ^ place heap)
 -(block ^ size { > <size> } ^ place heap)
 -->
 (modify 2 ^ place hand)
)
```

Beispielprogramm
nimmt Blöcke der
Größe nach von
einer Halde und
plaziert sie der
Größe
entsprechend.

```
(p holding
  (goal ^task add)
  (block ^ place hand)
  ==>
  (modify 1 ^ task put-down)
)

(p put-down
  (goal ^ task put-down ^ index <rank>)
  (block ^ place hand)
  ==>
  (modify 2 ^ place <rank>)
  (bind <a> (compute <rank> + 1 ) )
  ; Berechnung und Bindung der Variablen a
  (modify 1 ^ task add ^ index <a> )
)

(p stop
  (goal ^ task add)
  -(block ^ place heap)
  ==>
  (remove 1)
  ( vm ) ; druckt Arbeitsspeicher aus
  halt) ; beendet Abarbeitung
)

(defun rerun nil
  ( remove *) ; löscht gesamten Arbeitsspeicher
  (make start)
  (run) ; startet Abarbeitung
)
```