Universität Bielefeld

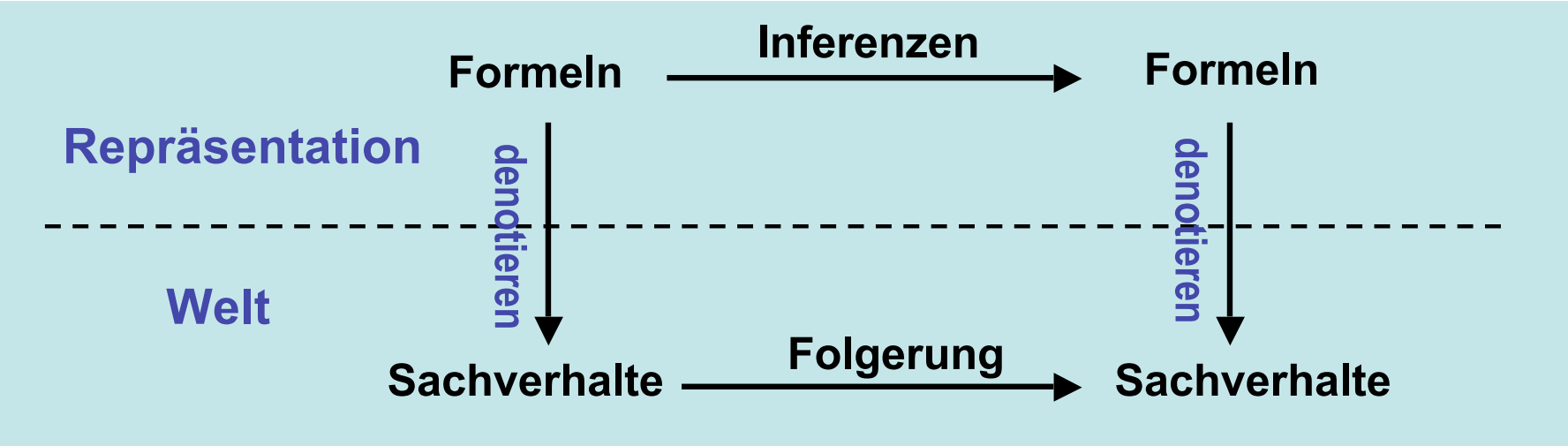
3 Logik und Inferenz

9. Vorlesung: Indexing;
Inferenz durch Graphsuche

Methoden der Künstlichen Intelligenz

Ipke Wachsmuth WS 2008/2009

Inferieren und Folgern



Eigenschaften von Inferenzverfahren

- **Korrektheit**: Denotieren die inferierten Formeln Sachverhalte der betrachteten Welt?
- **Vollständigkeit**: Können alle Sachverhalte der betrachteten Welt inferiert werden?

Inferenzregeln Übersicht

(I) Modus Ponens

Aus p und (if p q) inferiere q

(II) Universelle Einsetzung

Aus (forall (*-vars-*) p) inferiere p mit allen Vorkommen jeder Variable durch den gleichen Term eingesetzt

(III) Unifikations-Inferenz (Modus Ponens verallg.)

Aus p' und (if p q) inferiere q'

wobei p mit p' unifizieren muss: $p'\theta = p\theta$

und die resultierende Substitution θ auf q angewandt wird,

wodurch man q' erhält als: $q' = q\theta$

(IV) Subsumtions-Inferenz

Aus p subsumiert q schließe

q folgt aus p

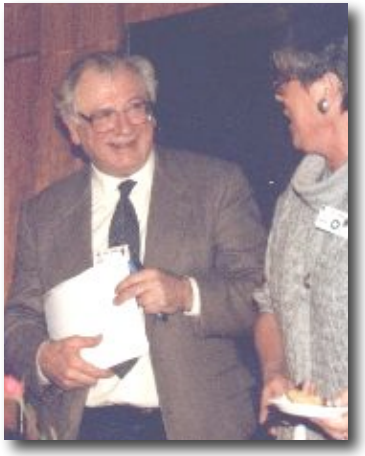
mit unifizierender Substitution analog zu (III)

(V) Allgemeine Resolutionsregel

Aus (or n_1 (not m')) und (or m n_2)

inferiere (or n_1' n_2')

Theorembeweiser: Bemerkungen



J.A. Robinson, 1996

Algorithmen zum Theorembeweisen sind oft unvollständig (existierende Beweise werden nicht immer gefunden)!

Versuche, einen Theorembeweiser vollständig(er) zu machen, werden i.a. teuer mit Effizienzverlusten bezahlt.

Es gibt etliche Varianten des Resolutionsverfahrens und speziellen Strategien dafür, die im Gebiet "Automatisches Beweisen" untersucht werden.

Sie alle gehen auf das allgemeine Resolutionsverfahren ("complete resolution") von ROBINSON (1965) zurück.

(Wdh.) Zu unterscheidende Ebenen

– siehe 7. Vorlesung –

Alternative Optionen:

(1) Logikbasierte Repräsentation (z.B. Prädikatenlogik)

vs.

(2) objektbasierte Repräsentation (z.B. semantische Netzwerke)

Bislang zunächst Fall (1) betrachtet.

- **Fakten- oder Wissensebene**

  bezieht sich auf darzustellende Sachverhalte (eines betrachteten Weltausschnitts)

- **Ebene der abstrakten Repräsentation**

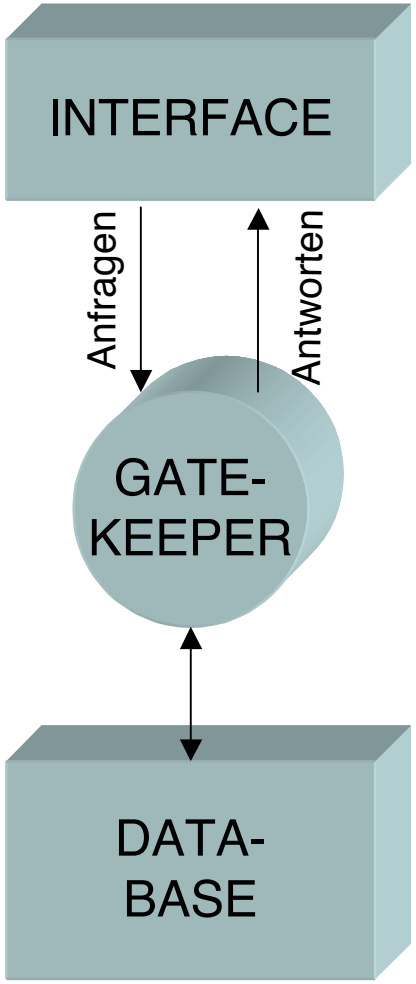
  Darstellung von Sachverhalten in symbolischen Ausdrücken **Fokus: *"Inhalt"***

- **Ebene der konkreten Repräsentation**

  maschinengeeignete und -verarbeitbare Darstellung (Implementierung) von Ausdrücken als Datenstrukturen

  Fokus: *"Index"*

Indexing (verbesserter Zugriff)



Frage bisher:

– Wie drückt man Sachverhalte über eine Domäne aus, so dass sich formale Schluss-weisen darauf anwenden lassen?

Frage jetzt:

– Wie organisiert man geschickt den Zugriff auf Einträge in DATABASE?

Zwei Basisansätze:

1. **Indexieren von PL- Formeln** in DATABASE
2. **semantische Netzwerke** in DATABASE und Inferenz durch Graphsuche

Indexieren von PL-Formeln

Fetch-Pattern („Hol-Muster“)

- Ausgangspunkt:

  DATABASE hält (skolemisierte) Fakten, die von einem Theorembeweiser manipuliert werden.

- Aufgabe jetzt:

  Schnelles Finden aller Fakten, die mit einer (als *Fetch-Pattern*) gegebenen Formel unifizieren.

- Diese Manipulationen müssen (vom Rechenaufwand her) also billig sein!

Fetch beim Backward Chaining

Gegeben goal *G*

dann muss GATEKEEPER alle Formeln der Form

(i) *G'*

oder (ii) (<– *G' S*) – hier jetzt für (`if` *S G'*) geschrieben –

auffinden, für die gilt: *G'* unifiziert mit *G*.

(i) das Fetch-Pattern ist *G* selbst

(ii) das Fetch-Pattern ist (<– *G ?v*)

(*?v* darf in *G* nicht vorkommen)

Indexieren von PL-Formeln: Beispiel

Das Fetch-Pattern `(contains ?x a)`

"matcht" (unifiziert) mit: `(contains new-york a)`

oder mit: `(contains universe ?v)`

NICHT mit: `(contains new-york empire-state-bldg)`

> **Eine passende Formel in DATABASE darf keine anderen Konstanten als die an den jeweiligen Positionen des Fetch-Pattern gegebenen haben.**

> **Die Formeln in DATABASE müssen so abgelegt sein, dass es einfach ist, alle "matchenden" aufzufinden (für gegebene Konstante und Position).**

> **Dazu erhalten sie einen Index: --> *indexing***

Indexierte DATABASE

Jeder Konstante ist eine Liste der Assertionen, die diese Konstante enthalten, zugeordnet, organisiert nach der <u>Position</u> der Konstante:

((*pos N -formulas-*)
(*pos N -formulas-*) ...)

pos Position, repräsentiert als Tupel natürlicher Zahlen:

< > ganze Formel

<1 2> erstes Element des zweiten Elements ...

N Zahl der Formeln mit dieser Konstante an dieser Position

var für enthaltene Variablen

```
contains:
    ((<1> 4 (contains new-york empire-state-bldg)
            (contains us new-york)
            (contains universe ?y)
            (contains us california))
     (<1 2> 1 (← (contains ?x ?y)
                 (and (contains ?x ?z)
                      (contains ?z ?y)))))

new-york:
    ((<2> 1 (contains new-york empire-state-bldg))
     (<3> 1 (contains us new-york))
     . . .)

*var*:
    ((<3> 1 (contains universe ?y))
     (<2 2> 2 (← (mortal ?x) (inst ?x human))
              (← (contains ?x ?y)
                 (and (contains ?x ?z)
                      (contains ?z ?y))))
     (<3 2> 1 (← (contains ?x ?y)
                 (and (contains ?x ?z)
                      (contains ?z ?y))))
     . . .)
```

Wie wird *fetch* durchgeführt?

- leite Schlüssel *(keys)* aus den Konstanten des Fetch-Pattern ab (also an welcher *pos* zu suchen)
- finde anhand der Schlüssel die derart indexierten Formeln als Formellisten *(sog. buckets)* in DATABASE auf

Algorithmen dafür sind bei Interesse in Charniak/McDermott, S. 398 zu finden.

Bemerkungen zum Indexing

Es werden nicht nur Formeln ausgeworfen, die das Fetch-Pattern *garantiert* matchen (d.h. Formeln werden geholt, für die Unifikation fehlschlägt).

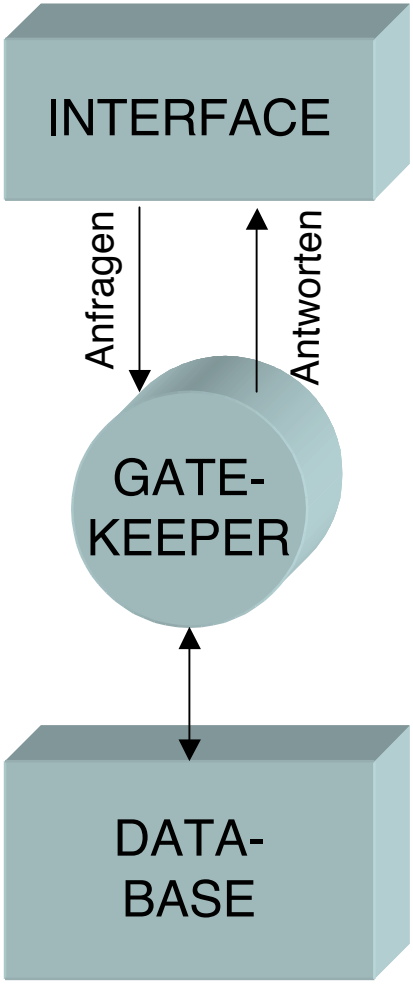
Trotz selektiver Verarbeitung kann der Rechenaufwand für Unifikation immer noch sehr teuer sein.

Der Algorithmus erfordert teure Operationen der Durchschnittbildung, für die es zwar Verbesserungen gibt `(mark-and-test)`, aber keinen grundsätzlichen "Durchbruch" (bei großen buckets).

Es gibt verbesserte Indexing-Methoden, die z.B. die Prädikat-Position spezifisch behandeln (in der Prädikatenlogik erster Stufe kommen keine Prädikatvariablen vor).

Alternativer Ansatz:
Semantische Netzwerke

Inferenz durch Graphsuche



vorher:

indexierte Assertionen in DATABASE

Theorembeweiser im GATEKEEPER

jetzt:

Semantisches Netz in der DATABASE

Graph-Suchalgorithmen (in der Regel mehrere spezielle) im GATEKEEPER

Semantische Netzwerke

– siehe 2. Vorlesung –

(Aufgabe wieder: Angabe von Datenstrukturen in DATABASE, die Fakten repräsentieren und auf die sich geschickt zugreifen lässt.)

Semantische Netzwerke **(auch "assoziative Netzwerke"):**
Allgemeiner Begriff für Repräsentationssysteme auf der Basis von Graphen, deren Kanten ("links") mit Prädikaten korrespondieren.

Auf Ebene der konkreten Repräsentation als Pointer-Strukturen implementiert, die eine Struktur von Relationen in der modellierten Domäne ausdrücken.

Basis für Inferenzen: *Graph-Suchalgorithmen*

Graph-Suchalgorithmen *vs.* Theorembeweiser

- Pointer verfolgen vom Rechenaufwand billiger als deduktive Prozesse mit Unifikation durchzuführen

- Ein Graph-Suchalgorithmus kann den Graph bei der Suche ohne weiteres markieren und z.B. Zirkularitäten sofort feststellen

- Bereichsabhängige Heuristiken können genutzt werden, z.B. erst "grobe", dann "feine" Suche (Stadtplanbeispiel: Erst auf der Ebene von Stadtteilen, dann auf Straßenebene suchen)

- *schneller, aber unflexibler:*

  spezifische Information

  z.B. `(connected-to a b)`

  kann leicht, allgemeine Information

  z.B.
  ```
  (forall (x) (if (connected-to x a)
                  (connected-to x b)
  ```

  in der Regel nicht dargestellt/genutzt werden

In vielen Fällen wird diese Flexibilität der Effizienz geopfert.

"speed outweighs flexibility"

„links“ als Datenstruktur

Beispiel: Auszudrücken sei

`(contains a b), (contains a c), (contains d a)` etc.

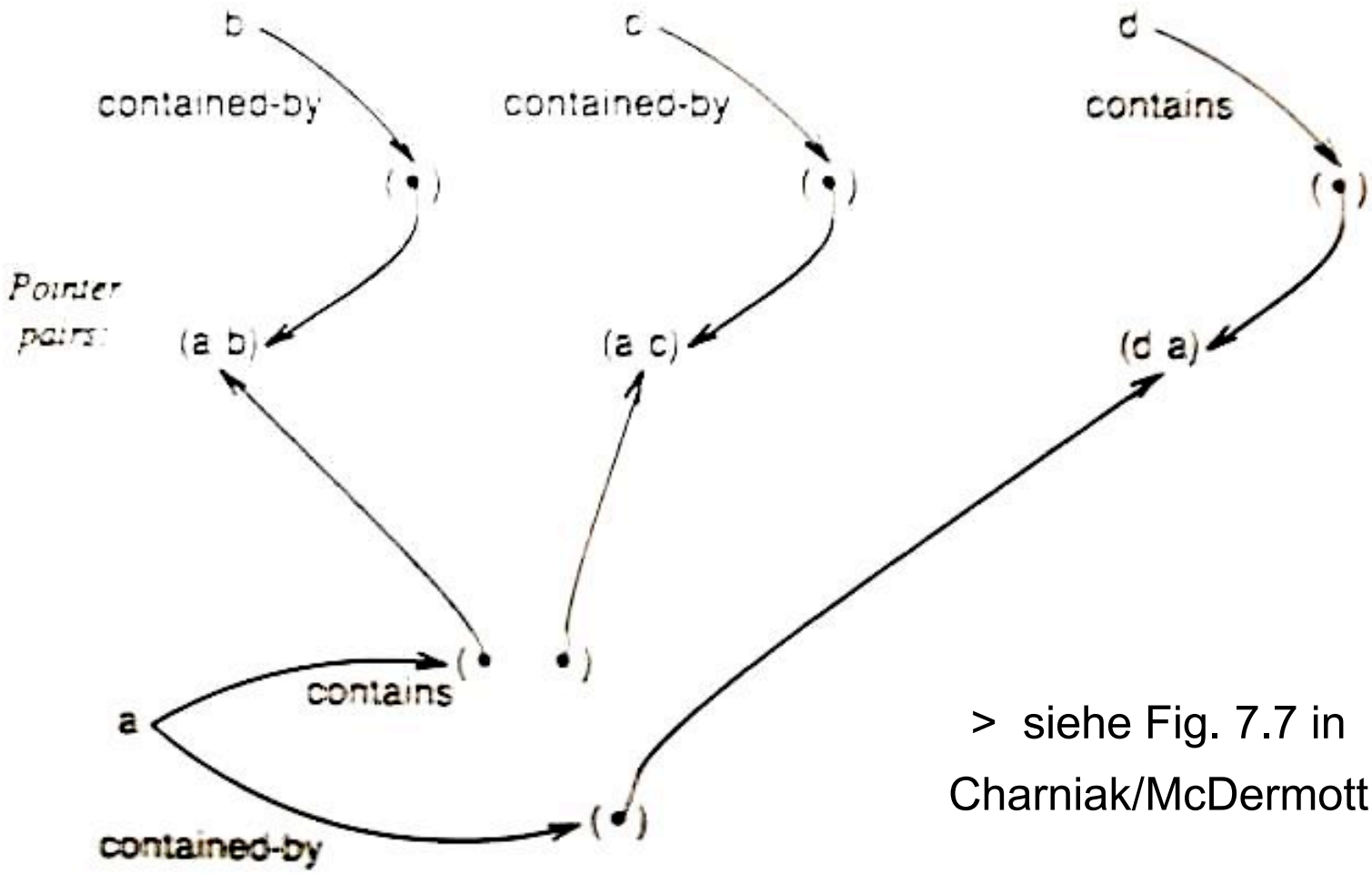
- **Es soll möglich sein, alles, was in `a` enthalten ist, von `a` aus leicht zu finden**
  --> *forward pointer* (für die Relation `contains`)
- **Es soll möglich sein, alles, worin `a` enthalten ist, von `a` aus leicht zu finden**
  --> *backward pointer* (für die Relation `contained-by`)

Pointer-Paar **(interne Darstellung eines links):**

Zweielementige Liste, deren Elemente über entsprechend gelabelte pointer auf die Liste zeigen.

`(a,b)` : links steht erstes, rechts zweites Argument einer `contains` -Relation

Implementation von links



> siehe Fig. 7.7 in Charniak/McDermott

Vergleich mit indexierten Assertionen

```
new-york:
   ((<2> 1 (contains new-york empire-state-bldg))
    (<3 > 1 (contains us new-york))
    ...)
```

ausgedrückt durch `contains` -Pointer (forward pointer)

ausgedrückt durch `contained-by` -Pointer (back pointer)

Frage: Welche Dinge `x` enthält `a`?

- mit backward chaining

  `(Show: (contains a ?x))`

  dann unifizierende Pattern suchen und ggfs. rückwärts verkettete Regeln ausnutzen, um subgoals zu bekommen.

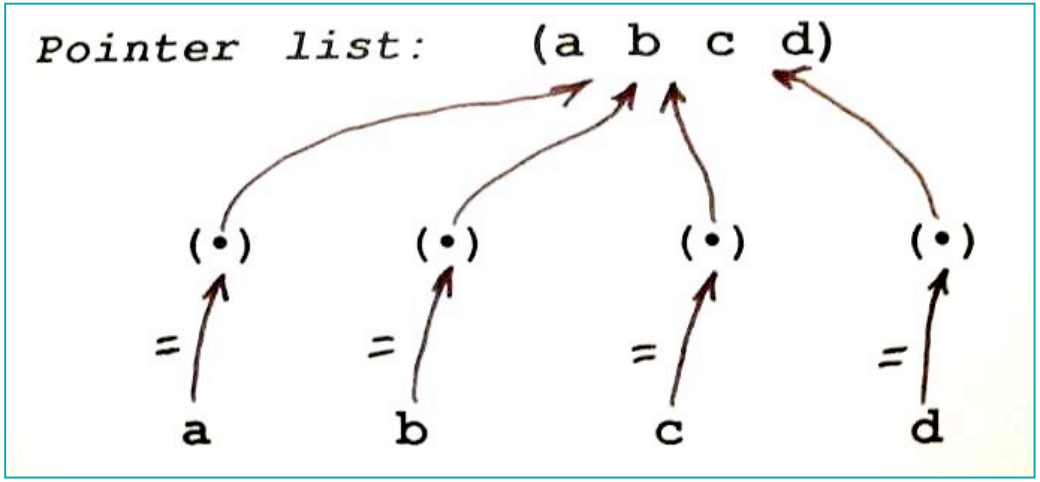
- im Netzwerk schneller (nämlich unmittelbar zugreifbar):
  - Betrachte die `contains` -links von `a` (siehe Fig. 7.7)
  - werfe jeweils die zweiten Elemente des Pointer-Paars aus

Vermeidung von backward chaining und Unifikation als Hauptvorteil für Netz-Inferenzen!

Weitere Vorteile

für symmetrische, transitive Relationen

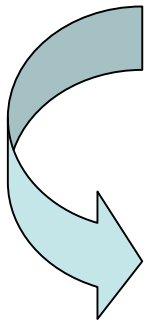
(= a b) (= b c) (= a d)



Dann (= c a) genauso leicht zu bestätigen wie (= a c) usw.

Scheinbarer Nachteil

- **Beschränkung auf binäre Relationen (für Assertions nicht der Fall)**
- **jedoch kann jede Assertion mit mehrstelligen Prädikaten in eine Menge von "binären" Assertionen überführt werden:**



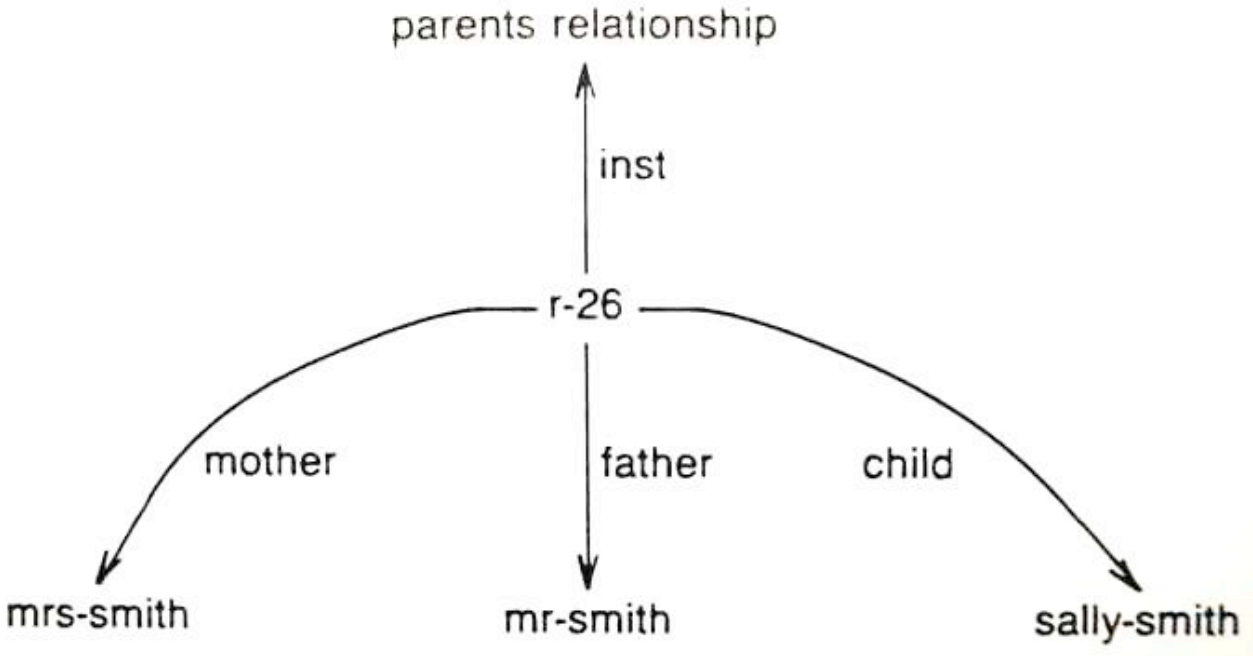
```
(p   a1   a2   . . .   an )
```

neue Konstante!

```
(inst    r   p-Relation)
(arg-1   r   a1 )
(arg-2   r   a2 )
   ...
(arg-n   r   an )
```

Dies ist Slot-Assertion-Notation! **(siehe 2. Vorlesung)**

Ein Beispiel



D.h. es werden ein expliziter Relationen-Knoten (hier: parents relationship) eingeführt und markierte Kanten für die Binär-Relationen (Pointer), die eine Instanz der Relation (hier: r-26) mit ihren Argumenten verbinden.

Die drei betrachteten link-Typen

- ***gerichtete Kanten*** **(für binäre Relationen)**
- ***Äquivalenzklassen*** **(für symmetrische, transitive Relationen)**
- ***explizite Knoten*** **(für Relationen mit > 2 Argum.)**

erfordern jeweils spezielle Graph-Suchalgorithmen ("link traverser"), die aber die Effizienz erheblich steigern.

Indexing – Bemerkungen

- **Indexing-Verfahren (oder Speicherorganisation allgemein) findet man in neueren KI-Büchern wenig berücksichtigt.**
- **Ein Grund dafür mag sein, dass die Bequemlichkeit direkter Logik-Repräsentation lauffähige Programme ohne explizite Betrachtung der Speicherorganisation erlaubt.**
- **Geschickt eingeschränkter Wissenszugriff ist jedoch wichtig: Anwendungswissensbasen können Hunderte oder Tausende von Axiomen oder noch viel mehr enthalten.**

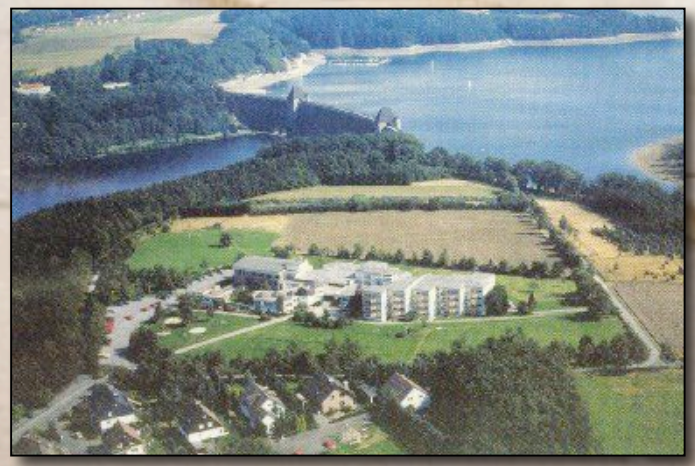
Lesehinweis heute:

- Charniak & McDermott, Kapitel 7, Seite 396-405

Rückblick auf Teil 3

- Ausgangspunkt: Ausstattung eines Agenten mit Schluss-folgerungsfähigkeiten
- Basis-Aufbau mit DATABASE und GATEKEEPER sowie Basisoperationen `assert`, `retract`, `query`
- Verschiedene deduktive Inferenzverfahren, insbesondere für variablenhaltige Formeln der PL
- Vorbereitung der Formeln für maschinelle Verarbeitung (insbesondere Skolemform) durch Theorembeweiser
- Indexing-Verfahren; Inferenz durch Graphsuche in assoziativen Netzen

Interdisziplinäres Kolleg IK2009
Günne/Möhnesee 6. - 13. März
http://www.ik2009.de



Student Stipends

Participants lacking financial support may apply for a stipend of up to 150 EUR to support their board & lodging. Applications must be made by filling in this web form and must be submitted by December 31st, 2008.