Einführung in die KI

Prof. Dr. sc. Hans-Dieter Burkhard
Vorlesung Winter-Semester 2005/06

Wissensrepräsentation:
Logische Beschreibung
Resolution (im PK1)

Einige von denen, die nicht glauben, dass Gott die Welt in sieben Tagen geschaffen hat sind keine Fundamentalisten, aber einige Fundamentalisten glauben, dass Gott die Welt in sieben Tagen geschaffen hat – also ist keiner, der nicht glaubt, dass Gott die Welt in sieben Tagen geschaffen hat, ein Fundamentalist.

Aus U.Eco: das Foucaultsche Pendel

2. Resolution

Vorbild für „Formalismus“:

– exakt, präzise, (theoretisch) beherrscht

Aufbau:

–Zeichen

–Ausdrücke (rekursive Definition)

–Sätze („Theorie“) **Th**

•syntaktisch bestimmt:

Th = Abl(**Ax**)

Nach speziellen formalen Regeln erzeugbare Formeln

•semantisch bestimmt:

Th = allgemeingültige Sätze einer Struktur

Beweise

„Irreflexive, transitive Relationen sind asymmetrisch“

Inhaltlicher Beweis:

– Argumentation

Formaler (syntaktischer) Beweis:

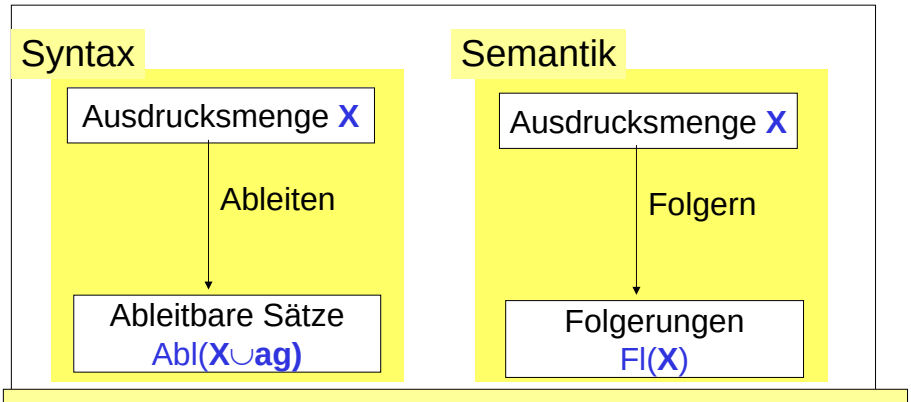
– Umformung von Ausdrücken („Kalkül“)

Theorembeweiser:

– (Syntaktisches) Verfahren zur Entscheidung, ob ein Ausdruck zu einer Satzmenge (Theorie) gehört:

$H \in Th$?

Axiomatische Behandlung der Logik



Korrektheit von Abl : $Abl(\mathbf{X} \cup \mathbf{ag}) \subseteq Fl(\mathbf{X})$

Vollständigkeit von Abl : $Abl(\mathbf{X} \cup \mathbf{ag}) \supseteq Fl(\mathbf{X})$

Äquivalenz von Abl : $Abl(\mathbf{X} \cup \mathbf{ag}) = Fl(\mathbf{X})$

Formales Ableiten (Resolutionsregel)

Für Klauseln („Disjunktion von Literalen“)

Voraussetzung:

K1 und K2 unifizierbar mittels Unifikator σ

K =Res(K1,K2,σ) entsteht durch

- „Vereinigung“ von σ(K1) und σ(K2)
- Streichen komplementärer Literale

Formales Ableiten (Resolution)

KA1: $\neg R(x,x)$

KA2: $\neg R(u,y) \vee \neg R(y,z) \vee R(u,z))$

K1: $R(c,d)$

K2: $R(d,c)$

K3 =Res(KA1,KA2,σ): $\neg R(w,y) \vee \neg R(y,w)$

mit $\sigma(u)= \sigma(z)= \sigma(x)= w$,$\sigma(y)= y$

K4 = Res(K1,K3,σ): $\neg R(d,c)$

mit $\sigma(w)=c$ $\sigma(y)=d$

K5 = Res(K2,K4,σ):

Enscheidbarkeit in der Logik

(rein logisch) allgemeingültige Sätze:
ag = Abl(**axp**) = Fl(∅)

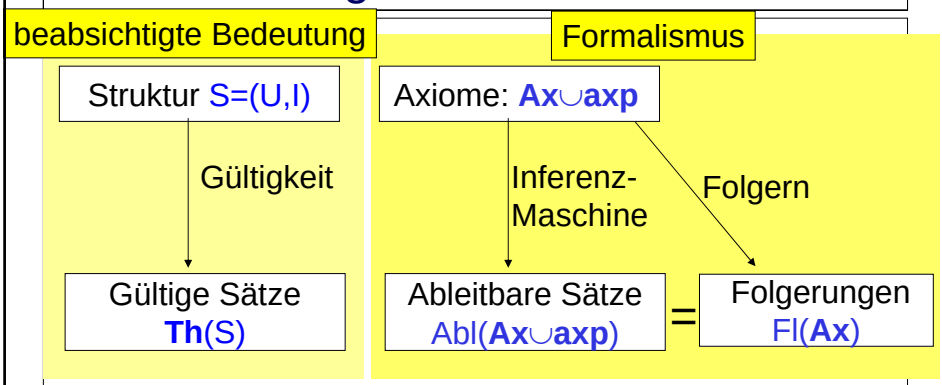
axp : Axiomensystem des PK1

H ∈ **ag** ?

- Entscheidbar im AK
- Unentscheidbar im PK1

  (aber aufzählbar, da axiomatisierbar)

Formalisierung einer Domäne

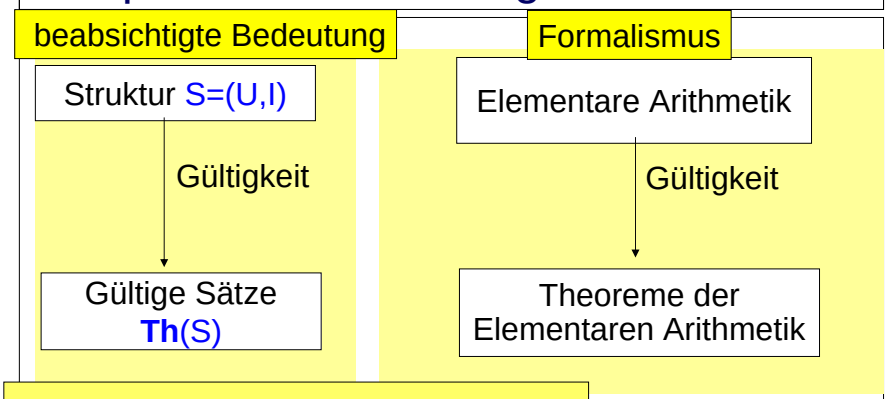


Korrektheit der Formalisierung : $Abl(\mathbf{Ax} \cup \mathbf{axp}) \subseteq \mathbf{Th}(S)$

Vollständigkeit der Formalisierung: $Abl(\mathbf{Ax} \cup \mathbf{axp}) \supseteq \mathbf{Th}(S)$

Äquivalenz der Formalisierung : $Abl(\mathbf{Ax} \cup \mathbf{axp}) = \mathbf{Th}(S)$

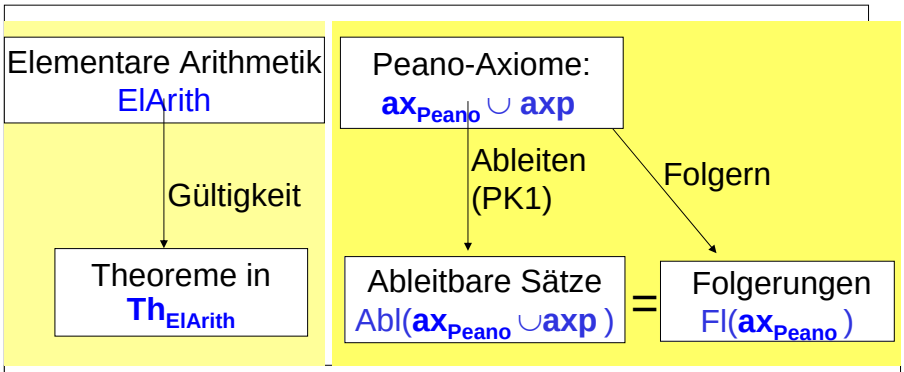
Beispiel: Formalisierung der Arithmetik



$Th(S) = \{ H \mid Wert_S(H,\beta) = 1 \text{ für alle } \beta \text{ über } U \}$

β : Belegung der Variablen
mit Individuen aus dem Universum U

Beispiel: Formalisierung der Arithmetik



Korrektheit der Formalisierung :

$$Abl(\mathbf{Ax_{Peano}} \cup \mathbf{axp}) \subseteq \mathbf{Th_{ElArith}}$$

Unvollständigkeit der Formalisierung:

$$Abl(\mathbf{Ax_{Peano}} \cup \mathbf{axp}) \neq \mathbf{Th_{ElArith}}$$

Adäquatheit der Formalisierung

Adäquatheit:

Die wesentlichen Aspekte werden in der Struktur S bzw. den Axiomen **Ax** korrekt erfaßt.

–Parallelen-Axiom der Geometrie

–Stetigkeitsdefinition in der Analysis

–Modellierung eines Staubsaugers

Modellierung eines Staubsaugers

Alternativen:

$$(S1) \quad \forall staubi : Staubsauger(staubi) \Rightarrow \exists l, m, s : Lampe(l) \wedge HatAlsTeil(staubi, l) \wedge Motor(m) \wedge HatAlsTeil(staubi, m) \wedge Stecker(s) \wedge HatAlsTeil(staubi, s)$$

$$(S2) \quad \forall staubi : Staubsauger(staubi) \Leftrightarrow \exists l, m, s : Lampe(l) \wedge HatAlsTeil(staubi, l) \wedge Motor(m) \wedge HatAlsTeil(staubi, m) \wedge Stecker(s) \wedge HatAlsTeil(staubi, s)$$

Modellierung eines Staubsaugers

Was folgt für *staubi*$_1$ aus (S1) bzw. (S2) zusammen mit einer der folgenden Aussagen:

$$\exists l, m, s : Lampe(l) \wedge HatAlsTeil(staubi_1, l) \wedge Motor(m) \wedge HatAlsTeil(staubi_1, m) \wedge Stecker(s) \wedge HatAlsTeil(staubi_1, s)$$

$$\neg\exists l, m, s : Lampe(l) \wedge HatAlsTeil(staubi_1, l) \wedge Motor(m) \wedge HatAlsTeil(staubi_1, m) \wedge Stecker(s) \wedge HatAlsTeil(staubi_1, s)$$

Modellierung - Adäquatheit

Mehr Axiome – weniger Modelle - mehr Folgerungen

Mehr Axiome – komplexere Verarbeitung

Adäquatheit hinsichtlich
- Anwendungsproblematik
- Komplexität

Vereinfachung, wenn klar ist,
dass es (nur) um Staubsauger geht

Modellierung eines Staubsaugers

Funktionsbeschreibung: Verhalten

$(M1) \quad \forall m : Brummt(m) \Leftarrow Motor(m) \wedge [\exists staubi, s, sd : Staubsauger(staubi) \wedge HatAlsTeil(staubi, m) \wedge Stecker(s) \wedge HatAlsTeil(staubi, s) \wedge Steckdose(sd) \wedge StecktIn(s, sd)]$

Angeschlossen(m)

Diagnose: Fehlverhalten ¬brummt(meinMotor)

$(M2) \quad \forall m : Brummt(m) \Leftarrow Motor(m) \wedge Angeschlossen(m)$

$(M3) \quad \forall m : Brummt(m) \Leftarrow Motor(m) \wedge Angeschlossen(m) \wedge Ok(m)$

Syntax des PK1

Terme: Individuen (Konstante, Variable, Funktionen)

Prädikate (atomare Formeln):

Relationen $R(x_1,...,x_n)$ (wahr/falsch)

Ausdrücke:

logische Beziehungen zwischen Prädikaten mittels

- Aussagenlogischen Operatoren $\neg \wedge \vee \leftrightarrow \rightarrow$
- Quantifikation von Variablen $\forall \exists$

z.B. $\forall x \exists y\ R(x, x_1,...,x_n) \wedge \neg R(y, x_1,...,x_n)$

Positives Literal: nicht negierte atomare Formel

Negatives Literal: negierte atomare Formel

Syntax des PK1: Ableiten

Ableiten: Ausdrücke umformen mit Ableitungsregeln

z.B. Abtrennungsregel (modus ponens)

$$\frac{H_1 \ ,\ H_1 \rightarrow H_2}{H_2}$$

H *ableitbar aus* **X ,**

falls Ableitungsfolge für H aus **X** existiert

X |- H oder: H $\in$ **X**|- oder: H $\in$ Abl(**X**)

Allgemeingültigkeit, Erfüllbarkeit

Belegung β erfüllt den Ausdruck H in der Struktur S = [U, I] ,

falls $Wert_S$ (H, β) = W.

(Rein logische) Erfüllbarkeit eines Ausdrucks H:

H heißt erfüllbar, falls β, U,I existieren mit $Wert_{[U,I]}$ (H, β) = W.

ef : Menge aller erfüllbaren Ausdrücke

(Rein logische) Allgemeingültigkeit eines Ausdrucks H:

H heißt allgemeingültig, falls $Wert_{[U,I]}$ (H, β) = W für alle β, U,I .

ag : Menge aller allgemeingültigen Ausdrücke

Freie Variable werden bei Allgemeingültigkeit wie generalisierte Variable behandelt.

Allgemeingültigkeit, Erfüllbarkeit

H ist allgemeingültig gdw. ¬H nicht erfüllbar ist.

H ist erfüllbar gdw. ¬H nicht allgemeingültig ist.

ag axiomatisierbar:

Es gibt aufzählbares Axiomensystem **axp** mit **ag** = Abl(**axp**)

Axiomatisierbar = aufzählbar = partiell entscheidbar
M entscheidbar **gdw**. M und Komplement von M aufzählbar

Allgemeingültigkeit, Erfüllbarkeit

Satz von Church:

Erfüllbarkeit/Allgemeingültigkeit im PK1 sind unentscheidbar.

Menge der allgemeingültigen Ausdrücke: axiomatisierbar.

Menge der nicht erfüllbaren Ausdrücke: axiomatisierbar.

Menge der nicht allgemeingültigen Ausdrücke: nicht axiomat.

Menge der erfüllbaren Ausdrücke: nicht axiomatisierbar.

H ist allgemeingültig gdw. ¬H nicht erfüllbar ist.

H ist erfüllbar gdw. ¬H nicht allgemeingültig ist.

Folgern im PK1

Eine Struktur S = [U, I] und eine Belegung β
sind ein *Modell* für eine Menge **X** von Ausdrücken,
wenn für alle H ∈ **X** gilt:
β erfüllt H in der Struktur S = [U, I] , d.h. $Wert_S(H, \beta) = W$.

Es sei **X** eine Menge von Ausdrücken, H ein Ausdruck.

H *folgt aus* **X** , falls gilt:

Jedes Modell von **X** ist ein Modell von H .

X |= H oder: H ∈ **X**|= oder: H ∈ Fl(**X**)

Folgern im PK1

Sei *staubi1* Modell für

(S1) $\forall staubi : Staubsauger(staubi) \Rightarrow$
$\exists l, m, s : Lampe(l) \wedge HatAlsTeil(staubi, l) \wedge$
$Motor(m) \wedge HatAlsTeil(staubi, m) \wedge$
$Stecker(s) \wedge HatAlsTeil(staubi, s)$

(*) $\neg\exists l, m, s : Lampe(l) \wedge HatAlsTeil(staubi_1, l) \wedge$
$Motor(m) \wedge HatAlsTeil(staubi_1, m) \wedge$
$Stecker(s) \wedge HatAlsTeil(staubi_1, s)$

staubi1 ist auch Modell von : $\neg$*Staubsauger(staubi1)*

Gilt bei allen Modellen,
d.h. $\neg$*Staubsauger(staubi1)* ist Folgerung von (S1) und (*)

Folgern und Ableiten im PK1

Fl ist syntaktisch beschreibbar mittels Abl

Fl = Abl „modulo **ag**“

ag = Fl(∅) = Abl(**axp**)

axp : Axiomensystem des PK1

Fl und Abl sind monoton:

X ⊆ **Y** ⇒ Fl (**X**) ⊆ Fl (**Y**)

X ⊆ **Y** ⇒ Abl (**X**) ⊆ Abl (**Y**)

(H → G) ∈ Fl (**X**) ⇔ G ∈ Fl (**X** ∪{H})

H ∈ Fl (**X**) ⇔ (∧**X** → H) ∈ **ag**

Formale Theorien im PK1

Als *Theorie*
wird eine bezüglich Folgern (Ableiten)
abgeschlossene Menge **Th**
von Ausdrücken bezeichnet:

Th = Fl(**Th**) = Abl(**Th**) „modulo **ag**“

Formale Theorien im PK1

Theorie mit *semantisch bestimmter Satzmenge*:

Gegeben ist eine Struktur S = [U, I] mit

Th = { F | F allgemeingültig in S}

= { F | $Wert_S$ (F, β) = W für alle β über U }

Arithmetik
Staubsauger

Theorie mit *syntaktisch bestimmter Satzmenge*:

Gegeben ist ein Ausdruckmenge **Ax** („Axiome“) mit

Th = Abl(**Ax** ∪**axp**) (= Fl(**Ax**))

Peano-Arithmetik
Staubsauger-Axiomatik

Formalisierung mittels PK1

Ziel: Axiome zur Beschreibung von Sachverhalten finden

Analoges Ziel: **Computerverarbeitung ermöglichen**

1. semantisch definierte Theorie **Th** bestimmen
   (Universum, Relationen/Funktionen, Interpretation)

2. axiomatische Beschreibung von **Th** :
   Axiome **Ax** mit **Th** = Abl(**Ax** $\cup$ a**xp**)

   (z.B. PROLOG-Programm)

Anwendung PK1

Formalisierung: Axiome **X**

Problem durch Ausdruck H beschreiben

Entscheiden, ob

$H \in Fl(\mathbf{X})$

d.h. ob $H \in Abl(\mathbf{X} \cup \mathbf{axp})$

Programme dafür: Theorembeweiser

Beweise

Positiver Kalkül: Allgemeingültigkeit entscheiden

$H \in \mathbf{Th}$?

Negativer Kalkül: Unerfüllbarkeit untersuchen

$\{\neg H\} \cup \mathbf{Th}$ widersprüchlich ?

Deduktiver Kalkül:

Erweitern der Axiome um H zu finden (forward chaining)

Testkalkül:

Reduktion von H auf Axiome (backward chaining)

Beispiel:

Resolution (PROLOG) ist *Negativer Testkalkül*

Beweise

$H \in Fl(\mathbf{X})$

gilt gdw. $H \in Abl(\mathbf{X} \cup \mathbf{axp})$ → Positiver Kalkül

$H \in Fl(\mathbf{X})$

gilt gdw. $(\wedge\mathbf{X} \rightarrow H) \in \mathbf{ag}$

gilt gdw. $\neg(\wedge\mathbf{X} \rightarrow H) \notin \mathbf{ef}$

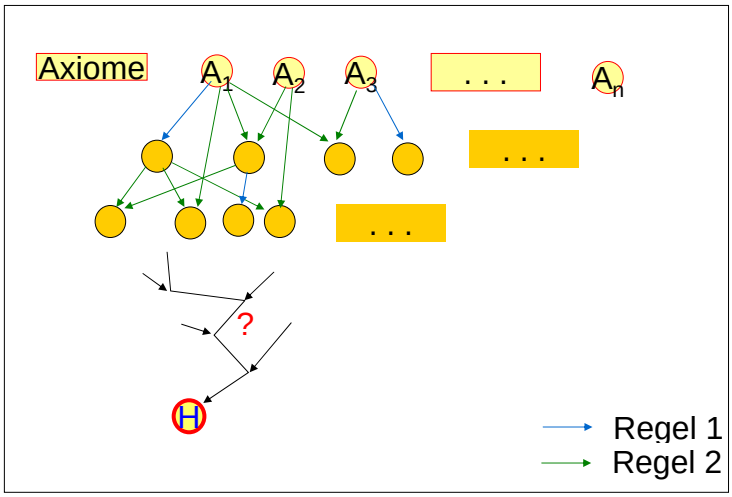
gilt gdw. Skolemform von $(\wedge\mathbf{X} \wedge \neg H)$ nicht erfüllbar

gilt gdw. Klauselform von $(\wedge\mathbf{X} \wedge \neg H)$ nicht erfüllbar

gilt gdw. Leere Klausel mittels Resolution ableitbar

→ Negativer Kalkül

Deduktiver Kalkül



Suchraum bei deduktivem Kalkül

Klassischer AK : 15 Axiome für **ag**, 2 Regeln

sogar entscheidbar, allerdings NP

Klassischer PK1:

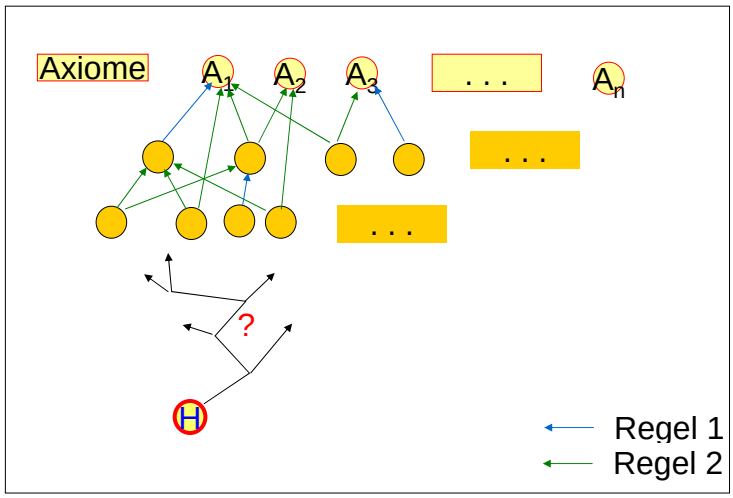
- abzählbar viele Axiome für **ag** + weitere für **Th** ,
  7 Regeln

- Alle allgemeingültigen Ausdrücke im Suchraum **Th** :
  ag = Fl(∅) ⊆ Fl(**Th**) = **Th**

- Mit H viele weitere Ausdrücke im Suchraum **Th** :
  z.B. H∨G für beliebiges G

Vollständigkeit als Nachteil

Test-Kalkül



Normalformen (Definitionen)

- ein Ausdruck H heißt *bereinigt*, falls gilt:
  - es gibt keine Variable x, die in H sowohl frei als auch gebunden vorkommt,
  - die hinter den Quantoren in H vorkommenden Variablen sind alle verschieden.

- ein Ausdruck H heißt *pränex*, falls gilt:
  - H hat die Form $Q_1\, x_1 \ldots Q_n\, x_n\, H'$,
    wobei $Q_1, \ldots, Q_n$ Quantoren sind
    und H' keine Quantoren enthält.

Normalformen (Definitionen)

- ein Ausdruck H heißt *pränexe Normalform*, falls gilt:
  - H hat pränexe Form $Q_1 x_1 \ldots Q_n x_n H'$,
  - H' ist eine konjunktive Normalform (KNF)

    $H' = \wedge \{ \{ \vee L_{ij} \mid i=1...n \} \mid j=1...m_n \}$

    wobei die L_{ij} Literale sind.

Literal:
atomare Formel (Prädikat, Relation) r(X1,...,Xn)
oder
negierte atomare Formel ¬r(X1,...,Xn)

Normalformen (Sätze)

1. Zu jedem Ausdruck H_1 existiert ein Ausdruck H_2 in bereinigter Form, so dass H_1 und H_2 semantisch äquivalent sind.
2. Zu jedem Ausdruck H_2 in bereinigter Form existiert ein Ausdruck H_3 in bereinigter pränexer Form, so dass H_2 und H_3 semantisch äquivalent sind.
3. Zu jedem Ausdruck H_3 in bereinigter pränexer Form existiert ein Ausdruck H_4 in bereinigter pränexer Normalform, so dass H_3 und H_4 semantisch äquivalent sind.

Zu jedem Ausdruck H
existiert ein semantisch äquivalenter Ausdruck G
in bereinigter pränexer Normalform.

Skolem-Normalform

Eine Skolem-Normalform ist eine pränexe Normalform, deren Variable sämtlich generalisiert sind.

Umformung einer bereinigten pränexen Normalform G in eine Skolem-Normalform F:

1. ∃-Quantoren einführen für alle freien Variablen in G. (erfüllbarkeits-äquivalente Umformung !!)
2. Elimination aller ∃-Quantoren mit Hilfe von Skolem-Funktionen.

Die entstehende Formel F ist erfüllbarkeits-äquivalent zu G.

Klauselform

Eine Klausel ist eine Menge von Literalen $K = \{L_1, ..., L_n\}$.

Ein Literal ist eine negierte oder nicht-negierte Atomformel.

Vorteil der Klauselnotation:

Mengenschreibweise beseitigt rein formale Unterschiede äquivalenter Ausdrücke bzgl. Kommutativität/Idempotenz.

Klauselform

Umformung einer Skolem-Normalform F in Klauselform:

1. Verteilung der Allquantoren auf die Alternativen.
   (Semantisch äquivalente Umformung.)
2. Gebundenene Umbenennungen derart, dass sich insgesamt alle Quantoren auf unterschiedliche Variablen beziehen. (Semantisch äquivalente Umformung.)

   $\mathbf{F} = \wedge_i \vee_j L_{ij}$
3. Darstellung der Alternativen als Klauseln:

   d.h. Menge von Literalen $\{ L_{ij} \mid j = 1, ...m_i \}$.

   Darstellung von F als Menge von Klauseln **Kl** :

   $\mathbf{Kl} = \{ \{ L_{ij} \mid j = 1, ...m_i \} \mid i = 1, ..., n \}$

Substitution (Definition)

Eine *Substitution* σ ist eine Abbildung der Individuenvariablen (eines Ausdrucks) in die Menge der Terme.

– Grundsubstitution: Abbildung in die Grundterme.

– Variablenumbenennung:

Ersetzung von Variablen durch Variable.

$\sigma(L)$: Ergebnis der Substitution σ für ein Literal L
(rekursiv definiert).

Hintereinanderausführung von Substitutionen:

$\sigma_1 * \sigma_2 (x) = \sigma_2(\sigma_1(x))$

$\sigma(L)$ ist jeweils „Spezialfall“ von L .

L ist die allgemeinste Form bzgl. aller $\sigma(L)$ für beliebige σ.

Substitution, Unifikation (Definition)

Eine Substitution σ heißt *Unifikator*

für eine Menge $\{L_1, ..., L_n\}$ von Literalen, falls gilt:

$\sigma(L_1) = \sigma(L_2) = ... = \sigma(L_n)$ (syntaktische Gleichheit).

Ein Unifikator σ heißt *allgemeinster Unifikator*

(m.g.u. = most general unifier),

falls für jeden Unifikator σ_0 eine Substitution σ_{00} existiert mit

$\sigma_0 = \sigma * \sigma_{00}$ (σ_0 ist spezieller als σ).

Der allgemeinste Unifikator ist eindeutig bis auf Umbenennungen.

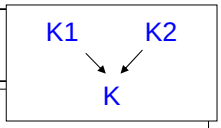
Substitution, Unifikation

- Der allgemeinste Unifikator σ einer Menge $\{ L_1,...,L_n \}$ beschreibt die Menge der gemeinsamen Spezialisierungen (Beispiele).
- Werden die Variablen der Literale zuvor separiert, so ergibt sich ein allgemeinerer Unifikator.

Satz

- Zu jeder unifizierbaren Menge von Literalen existiert ein allgemeinster Unifikator.
- Es ist entscheidbar, ob ein Unifikator existiert.
- Ein allgemeinster Unifikator kann ggf. konstruiert werden.

Resolutionsregel



Voraussetzungen:

- K_1 und K_2 Klauseln ohne gemeinsame Variablen
  (sonst: Separation durch Variablenumbenennungen)
- Positive Literale $L'_1,\ldots, L'_{n'} \in K_1$ und
  negative Literale $\neg M'_1,\ldots, \neg M'_{m'} \in K_2$,
  wobei $L'_1,\ldots, L'_{n'}, M'_1,\ldots, M'_{m'}$ unifizierbar sind mittels Unifikator σ.

Durch Resolution entsteht die *Resolvente*

$$K = \sigma ((K_1 - \{L'_1, \ldots, L'_{n'}\}) \cup (K_2 - \{\neg M'_1, \ldots, \neg M'_{m'}\}))$$

$$\frac{\{L_1,..,L_n,L'_1,..,L'_{n'}\}, \{M_1,.., M_m,\neg M'_1,..,\neg M'_{m'}\}, \sigma(L'_1)=..=\sigma(L'_n)=\sigma(M'_1)=..=\sigma(M'_m)}{\sigma (\{ L_1,..,L_n , M_1,..,M_m \})}$$

Resolutionsregel

Zerlegung in zwei Regeln.

Einfache Resolutionsregel:

$$\frac{\{L_1,..,L_n,L'\}, \{M_1,.., M_m, \neg M'\}, \sigma(L')=\sigma(M')}{\sigma (\{ L_1,..,L_n , M_1,..,M_m \})}$$

Faktorisierungsregel:

$$\frac{\{L_1,..,L_n,L_1',L'_2\}, \sigma(L_1')=\sigma(L_2')}{\sigma (\{ L_1,..,L_n ,L_1' \})}$$

Spezialfälle

Schnittregel:

$$\frac{A \vee B,\ \neg B' \vee C\ ,\ \sigma(B)=\sigma(B')}{\sigma\ (A \vee C)}$$

Modus ponens:

$$\frac{B,\ B \rightarrow C}{C}$$

Indirekter Beweis:

$$\frac{B,\ \neg\ B}{\ }$$

Unerfüllbarkeitsnachweis

Res(**K**) = Menge aller mit Resolution in endlich vielen Schritten aus einer Klauselmenge **K** ableitbare Klauseln

Klauseln(H) = Klauseldarstellung eines Ausdrucks H
(nicht eindeutig)

Satz:

$H \notin$ **ef** gdw. $\in$ Res(**Klauseln**(H))

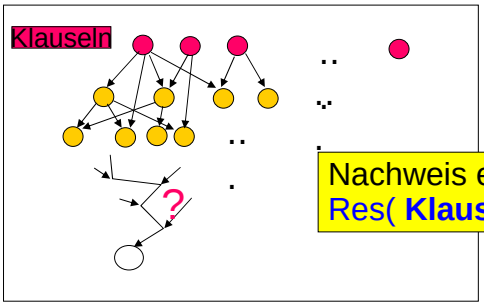
Beweisidee: Herbrand-Modelle

- alle durch Grundsubstitutionen (Konstante, Funktionen) erzeugbaren aussagenlogischen Formeln untersuchen
- Resolution für abzählbar viele Formeln:
  Unifikation als verzögerte Substitution (lazy evaluation)

Unerfüllbarkeitsnachweis

Nachweis der Unerfüllbarkeit eines Ausdrucks H ($H \notin$ **ef**):

Suche in Res(**Klauseln**(H)) nach



Nachweis erfolgreich, wenn in Res(**Klauseln**(H)) gefunden.

Res(**Klauseln**(H)) i.a. unendlicher Suchraum.

Vollständigkeit des Suchverfahrens?

Resolutionsverfahren

Nachweis von $H \in Fl$ (**X**)

Konjunktive Normalform von ($\wedge$**X** $\wedge \neg H$) bilden.

Darstellung als (erfüllbarkeitsäquivalente) Klauselmenge

$$\mathbf{Kl} = \{ \{ L_{ij} \mid j = 1, ...m_i \} \mid i = 1, ..., n \}$$

Suchverfahren:

- Wiederholtes Erweitern der Klauselmenge durch neue Resolventen.
- Abbruch, wenn leere Klausel abgeleitet wurde: EXIT($H \in Fl$ (**X**)).
- Wenn keine neuen Klauseln ableitbar: EXIT($H \notin Fl$ (**X**)). (i.a. aber unendlicher Suchraum)

Resolutionsverfahren

Nachweis von $H \in Fl(\mathbf{X})$

Resolutionsverfahren liefert Lösung im Fall der Unerfüllbarkeit in endlich vielen Schritten z.B. bei Breite-Zuerst-Verfahren.

Für erfüllbare Formeln dagegen evtl. kein Resultat (unendliches Resolvieren ohne Erreichen von).

Resolutionsverfahren

- ist Negativer Testkalkül. Verwendet
  - spezielle Normalformen (Klauseln)
  - Resolutionsregel (Schnittregel, Unifikation)
- ist nicht vollständig (bzgl. Folgerungen)
- aber widerlegungsvollständig
  - wenn H unerfüllbar, so mit Resolution ableitbar
- und widerlegungskorrekt
  - wenn mit Resolution ableitbar, so H unerfüllbarist

H unerfüllbar gdw. mit Resolution ableitbar

Beispiel: Formales Ableiten

Voraussetzungen (Axiome):

A1: $\forall x (\neg R(x,x))$ (Irreflexivität)

A2: $\forall x \forall y \forall z (R(x,y) \wedge R(y,z) \rightarrow R(x,z))$ (Transitivität)

Behauptung:

H: $\forall x \forall y (R(x,y) \rightarrow \neg R(y,x))$ (Asymmetrie)

Zu zeigen:

$A1 \wedge A2 \rightarrow H$ ist allgemeingültig

bzw.

$\neg (A1 \wedge A2 \rightarrow H)$ ist unerfüllbar

d.h.

$(A1 \wedge A2 \wedge \neg H)$ ist unerfüllbar

Beispiel: Formales Ableiten (Klauselform)

$(A1 \wedge A2 \wedge \neg H)=$

$\forall x(\neg R(x,x)) \wedge \forall x \forall y \forall z(R(x,y) \wedge R(y,z) \rightarrow R(x,z)) \wedge \neg \forall x \forall y(R(x,y) \rightarrow \neg R(y,x))$

ist semantisch äquivalent zu

$\forall x(\neg R(x,x)) \wedge \forall x \forall y \forall z(R(x,y) \wedge R(y,z) \rightarrow R(x,z)) \wedge \exists x \exists y \neg (R(x,y) \rightarrow \neg R(y,x))$

ist semantisch äquivalent zur pränexen Form

$\forall x \forall u \forall y \forall z \exists v \exists w(\neg R(x,x) \wedge (R(u,y) \wedge R(y,z) \rightarrow R(u,z)) \wedge \neg (R(v,w) \rightarrow \neg R(w,v)))$

ist semantisch äquivalent zur pränexen KNF

$\forall x \forall u \forall y \forall z \exists v \exists w(\neg R(x,x) \wedge (\neg R(u,y) \vee \neg R(y,z) \vee R(u,z)) \wedge R(v,w) \wedge R(w,v))$

ist erfüllbarkeitsäquivalent zur Skolemform

$\forall x \forall u \forall y \forall z(\neg R(x,x) \wedge (\neg R(u,y) \vee \neg R(y,z) \vee R(u,z)) \wedge R(f1(x,u,y,z),f2(x,u,y,z)) \wedge R(f2(x,u,y,z),f1(x,u,y,z)))$

ist erfüllbarkeitsäquivalent zur Klauselform

$\{ \{\neg R(x,x)\},\{\neg R(u,y),\neg R(y,z),R(u,z) \}, \{R(f1(x,u,y,z),f2(x,u,y,z))\}, \{R(f2(x,u,y,z),f1(x,u,y,z)\} \}$

Beispiel: Formales Ableiten (Klauselform)

Ausgehend von der Umstellung

$(A1 \wedge A2 \wedge \neg H) = (\neg H \wedge A1 \wedge A2) =$

ergibt sich einfachere Form:

ist semantisch äquivalent zur pränexen KNF

$\exists v \exists w(\forall x \forall u \forall y \forall z\ (R(v,w) \wedge R(w,v) \wedge \neg R(x,x) \wedge (\neg R(u,y) \vee \neg R(y,z) \vee R(u,z)))$

ist erfüllbarkeitsäquivalent zur Skolemform

$\forall x \forall u \forall y \forall z (R(c,d) \wedge (R(d,c) \wedge \neg R(x,x) \wedge (\neg R(u,y) \vee \neg R(y,z) \vee R(u,z)))$

ist erfüllbarkeitsäquivalent zur Klauselform

$\{ \{\neg R(x,x)\}, \{\neg R(u,y), \neg R(y,z), R(u,z)\}, \{R(c,d)\}, \{R(d,c)\} \}$

Beispiel:Formales Ableiten (Resolution)

KA1: $\neg R(x,x)$

KA2: $\neg R(u,y) \vee \neg R(y,z) \vee R(u,z))$

K1: $R(c,d)$

K2: $R(d,c)$

K3 =Res(KA1,KA2,σ): $\neg R(w,y) \vee \neg R(y,w)$
mit $\sigma(u) = \sigma(z) = \sigma(x) = w$, $\sigma(y) = y$

K4 = Res(K1,K3,σ): $\neg R(d,c)$
mit $\sigma(w)=c$ $\sigma(y)=d$

K5 = Res(K2,K4,σ):

Spezielle Resolutionsstrategien

Resolutionsmethode ist Grundlage für

Deduktionssysteme, Theorembeweiser, ...

Problem: Kombinatorische Explosion des Suchraums

Effizienzverbesserungen durch

1) Streichen überflüssiger Klauseln, z.B. :
   - tautologische Klauseln K , d.h. $\{A, \neg A\} \subseteq K$
   - subsumierte Klauseln: K_1 wird von K_2 subsumiert, falls $\sigma(K2) \subseteq K1$ bei einer geeigneten Substitution σ.

2) Heuristiken für Suchstrategie, Klauselgraphen

Problem: Widerspruchsvollständigkeit sichern.

Spezielle Resolutionsstrategien

Definition:

- Eine Atomformel ist ein *positives Literal*, eine negierte Atomformel ist ein *negatives Literal*.
- Eine *Klausel heißt negativ*, wenn sie nur negative Literale enthält.
- Eine *Klausel heißt definit*, falls sie genau ein positives Literal (und evtl. noch negative Literale) enthält.
- Eine Klausel heißt *HORN-Klausel*, wenn sie höchstens ein positives Literal (und evtl. noch negative Literale) enthält.

Hornklausel: definit (Prolog-Klauseln)
oder negativ (Prolog-Anfrage)

Spezielle Resolutionsstrategien

P-Resolution:
Eine der resolvierenden Klauseln enthält nur positive Literale.

N-Resolution:
Eine der resolvierenden Klauseln enthält nur negative Literale.

Lineare Resolution:
Resolution benutzt jeweils die im vorigen Schritt erzeugte Resolvente.

Input-Resolution:
Resolution benutzt jeweils die im vorigen Schritt erzeugte Resolvente und eine Klausel der Ausgangsmenge.
(Spezialfall der linearen Resolution).

Einheitsresolution:
Resolution benutzt jeweils mindestens eine ein-elementige Klausel.

Spezielle Resolutionsstrategien

Stützmengenresolution:
Resolution benutzt jeweils eine Klausel aus der Stützmenge **T**.
Als Stützmenge wird eine Teilmenge **T** der Ausgangsmenge **KI** verwendet mit **KI - T** erfüllbar.

Satz
P-Resolution, N-Resolution, lineare Resolution und Stützmengen-Resolution sind widerlegungsvollständig.
Input-Resolution und Einheitsresolution sind widerlegungsvollständig für HORN-Klauseln (aber nicht im allgemeinen Fall).

SLD-Resolution für HORN-Klauseln

S = „Selection Function“ (s.u.)
L = lineare Resolution
D = definite Klauseln

Definition

- Ein definites Programm **P** ist eine Menge definiter Klauseln.
- Eine Zielklausel ist eine negative Klausel.
- Ein SLD-Widerlegungsbeweis für eine (positive) Klausel G aus einem Programm **P** ist SLD-Ableitung von aus $\mathbf{P} \cup \{\neg G\}$.

SLD-Resolution für HORN-Klauseln:

Start mit negativer Klausel $\neg G$.

In jedem Schritt wird eine negative mit einer definiten Klausel aus **P** resolviert. Alle Resolventen sind negativ.

Spezialfall der linearen Resolution, Input-Resolution, N-Resolution.

SLD-Resolution für HORN-Klauseln

Sei σ_i die im i-ten Schritt der SLD-Resolution verwendete Substitution (Unifikator).

Dann ist $\sigma = \sigma_1 * ... * \sigma_n$ die beim Erfolg nach n Schritten erzeugte Antwortsubstitution.

Satz

Für jede Antwortsubstitution σ
eines SLD-Widerlegungsbeweises für G aus **P** gilt

$$\mathbf{P} \models \sigma(G) .$$

Falls $\mathbf{P} \models \sigma(G)$, so existiert ein SLD-Widerlegungsbeweis für G aus **P** mit einer Antwortsubstitution σ' , die allgemeiner als σ ist.

SLD-Resolution für HORN-Klauseln

Jeder Resolutionsschritt der SLD-Resolution benutzt eine negative und eine definite Klausel.

Es sind zwei Entscheidungen zu treffen:

1. Auswahl eines Literals $\neg L$ der negativen Klausel mittels „selection function“ (im Prinzip beliebig: *Und-Verzweigung*).
2. Auswahl einer Klausel mit einem positiven Literal M, das mit L unifizierbar ist (*Oder-Verzweigung*).

Suche im Und-Oder-Baum,

- z.B. Breite-Zuerst (Findet ggf. eine Lösung.)
- oder Tiefe-Zuerst.

PROLOG

Logische Programmiersprache.

Algorithmus = Logik + Steuerung

HORN-Klauseln durch programmiersprachliche Konzepte ergänzt:

- E/A-Funktionen
- meta-logische Funktionen
  - Analyse und Synthese von Prädikaten
  - Analyse von Bindungen
  - Programm-Modifikation
  - ...
- Eingriff in Suchprozedur (Cut)

PROLOG

Algorithmus = Logik + Steuerung

Steuerung durch Interpreter mit Auswahlstrategie:

1. links vor rechts
2. oben vor unten

und Suchstrategie:

Tiefe-Zuerst.

Implementation

- der Abarbeitung: Siehe Und-Oder-Bäume
- der Variablenbindungen als Ergänzung der goal-stacks
- Prolog-Compiler (WAM=Warren Abstract Machine)

Spezielle Speicherorganisation für Effizienz:
- Prozedurkeller (aufgerufene Teilziele/Klauseln)
- Backtrack-Punkte (Choice points)
- Variablenbindungen

PROLOG

Problem: Behandlung der Negation

$\neg L$ mit SLD-Resolution aus HORN-Klauseln nicht beweisbar

Closed world assumption (CWA)

Postulat:
$\neg L$ gilt, wenn L nicht bewiesen werden kann.

Beweisverfahren „**Negation by failure“**:

$\neg L$ ist bewiesen, wenn L nicht beweisbar ist.

Problem: Nicht-Beweisbarkeit ist nicht aufzählbar.

PROLOG: Im Rahmen der verwendeten Beweis-Strategie ist nur **Negation by finite failure** bzgl. Tiefe-Zuerst-Suche realisiert.

PROLOG-Semantiken

- Deklarative Semantik
  Ableitbarkeit/Folgerungen aus Programm
- Prozedurale Semantik
  Verhalten des Prolog-Interpreters

Unterschiede:
- Suchverfahren (Tiefe-Zuerst)
- Seiteneffekte (z.B. Eingabe/Ausgabe)
- Eingriffe in Beweisprozedur (z.B. Cut)
- „Meta-logische“ Prädikate
- Negation
- Occur-Check
- . . .

PROLOG

Spezielles Verhalten wegen Negation by finite failure:

```
maennlich(X)  :-  not  weiblich  (X).
weiblich(anna).
weiblich(frieda).
?  -  maennlich(fritz).      -  yes
?  -  maennlich(anna).       -  no
?  -  maennlich(heidi).      -  yes
?  -  maennlich(X).          -  no
```

Seiteneffekte (z.B. E/A) abhängig von Klauselreihenfolge:

```
...,  X = a,  write(X),  ...
...,  write(X),  X  =  a,  ...
```

Reihenfolge mit Einfluss auf Erfolg:

```
...,  X = a,  var(X),  ...
...,  var(X),  X = a,  ...
```

PROLOG

Problematik Occur Check: x enthalten in σ(x) ?
Beispiel: r(x) und r(f (x)) unifizieren, σ(x) = f (x) ?
Unterschiedliche Varianten:

- als unendlichen bzw. zyklischen Term behandeln: r(...f (f (f (x))) ...)
- ignorieren.
  PROLOG-Systeme liefern unterschiedliche Antworten:
  Test mit Klausel p(X,f (X)) .

```
? - p(X,X).                      yes   ?
? - p(X,X),write(X).             X =   ?
? - p(X,X),p(Y,Y ),X = Y .       yes   ?
```

- verbieten: aufwendiger Test!

In vielen PROLOG-Systemen: Occur-Check wahlweise
(Programmtest mit Occur-Check, laufendes Programm ohne).

Prolog-Interpreter

Umsetzung des SLD-Verfahrens als Zustandsraumsuche
(Tiefe-Zuerst)

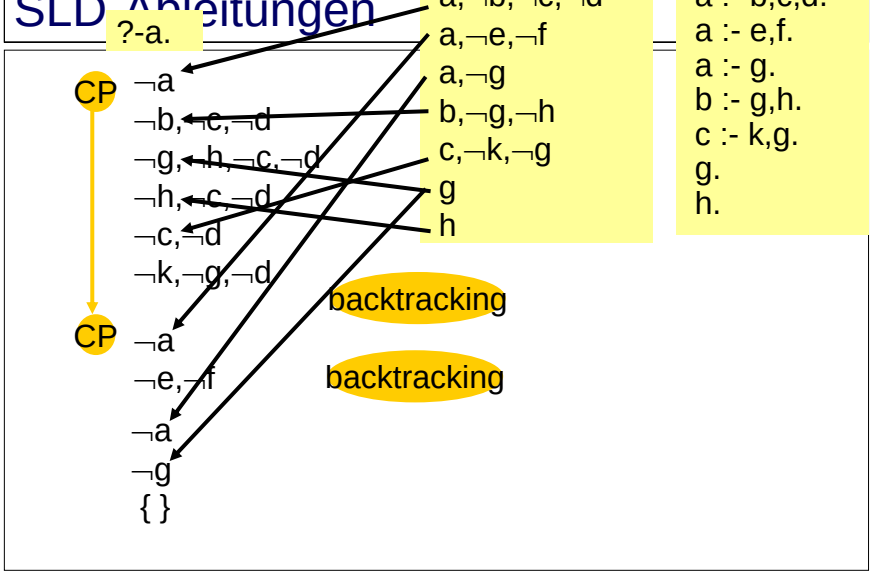
Prozedurkeller (**Frames**) für Klauselaufrufe

Verwendung des Backtracking-Prinzips
(Organisiert durch Rücksetzen des Prozedurkellers)

Environments für Variablenbindungen (durch Unifikation)

Optimierungen

SLD Ableitungen



Structure Sharing vs. Structure Copying

Structure Copying

Für jeden Aufruf einer Klausel wird eine Kopie mit Variablenbindungen angelegt

```
erreichbar(X,Y)
    :-nachbar(X,Z),erreichbar(Z,Y).
erreichbar(X,X).
nachbar(berlin,potsdam).
nachbar(berlin,adlershof).
...
nachbar(potsdam,werder).
nachbar(potsdam,lehnin)
...
```

```
erreichbar(berlin,Y) :-nachbar(berlin,Z),erreichbar(Z,Y).
nachbar(berlin,potsdam).
erreichbar(potsdam,Y) :-nachbar(potsdam,Z),erreichbar(Z,Y).
nachbar(potsdam,werder).
erreichbar(werder,Y) :-nachbar(werder,Z),erreichbar(Z,Y).
```

Structure Sharing vs. Structure Copying

Structure Sharing

Für die Aufrufe einer Klausel werden Referenzen auf die Klauselstruktur im Programm angelegt.

```
erreichbar(X,Y)
    :-nachbar(X,Z),erreichbar(Z,Y).
erreichbar(X,X).
nachbar(berlin,potsdam).
nachbar(berlin,adlershof).
...
nachbar(potsdam,werder).
nachbar(potsdam,lehnin)
...
```

Alle Aufrufe der Klausel
- benutzen die Strukturen des Programms.
- besitzen spezielles Segment für Variablenbindungen.

Implementation: Frames

Prozedurkeller: Für jeden Klauselaufruf wird im Laufzeitsystem ein Speicherbereich reserviert: „Frame“

Er enthält:
- Klausel einschließlich der Argumente.
  Effiziente Variante: „structure sharing“, in Frame nur
  - Verweis auf template der Klausel
  - Argumente der Klausel („environment“).
- Verweis auf nächstes subgoal der aktuellen Klausel.
- Verweis auf Vater-Klausel der aktuellen Klausel.
- Ggf. Verweis auf jüngsten Backtrack-Punkt.

Implementation: Frames

Frame enthält zusätzlich für Backtrack-Punkte :

- Nächste Klausel beim Backtracking.
- Davor liegender Backtrack-Punkt.
- Trail-Information
  (Protokoll der zu löschenden Variablenbindungen)

Dafür Reorganisation bei jedem Backtracking:
Streichen von Frames (Rücksetzen des Prozedurkellers)

Environment

für Argumente einer Klausel wird ein Speicherbereich angelegt: Environment.

Bindung erfolgt bei Unifikation durch Verweise
an Argumente von (im allgemeinen) älteren Klauseln
bzw. an Konstante.

```
erreichbar(X,Y)
    :-nachbar(X,Z),erreichbar(Z,Y).
```

Arg1
Arg2
Arg3

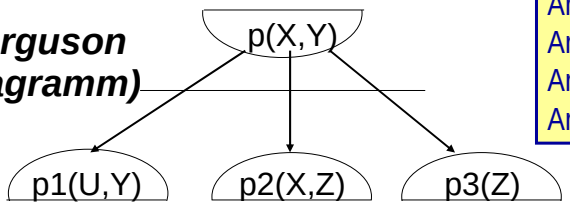
```
erreichbar(X,Y).
```

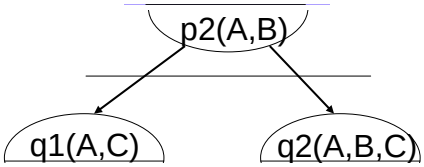
Arg1
Arg2

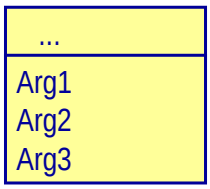
Bindungen

p(X,Y) :- p1(U,Y), p2(X,Z), p3(Z).

(Ferguson -Diagramm)



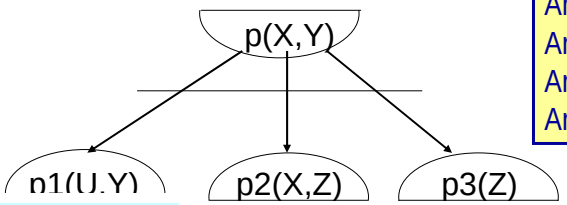


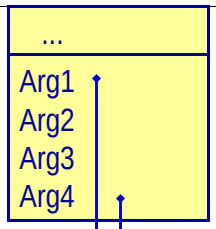


p2(A,B):-q1(A,C),q2(A,B,C).

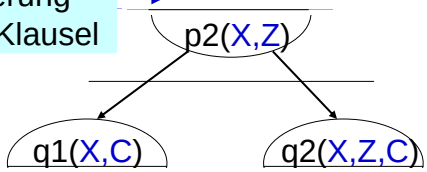
Bindungen

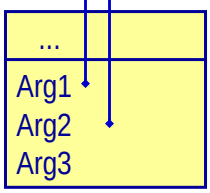
p(X,Y) :- p1(U,Y), p2(X,Z), p3(Z).





Unifizierung subgoal/Klausel





p2(A,B):-q1(A,C),q2(A,B,C).

Environment

Bindungen führen (im allg.) in ältere Teile des Kellers

Unifikation durch Ausführen des „matches“ zwischen Aufruf einer Klausel (als subgoal) und Kopf einer Klausel

- Dereferenzieren eines Arguments Arg_i entlang der Bindungen führt zu $Deref(Arg_i)$
  (kann Variable, Atom oder Struktur sein)
- Unifikation entsprechend Unifikationsregeln für die dereferenzierten Argumente

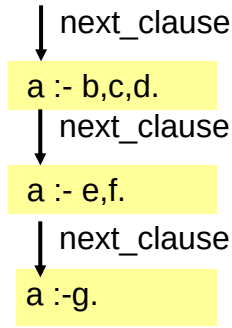
Beim Backtracking entfallen viele Bindungen durch streichen der Keller-Segmente (Frames)

Bindungen, die nicht dadurch entfallen, werden im trail protokolliert und beim Backtracking explizit aufgelöst.

Organisation der Prozeduren

Prozedur a

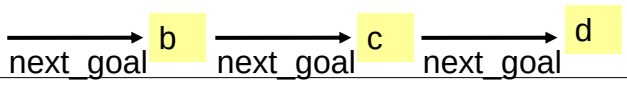
a :- b,c,d.
a :- e,f.
a :- g.



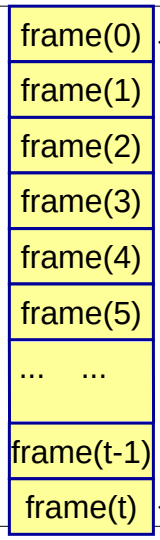
Prozedur als verkettete Liste der Klauseln

Klausel als Liste von goals

a :- b,c,d.



Frame: Prozedurkeller (local stack)



⇐*Anfrage*

⇐*aktueller Aufruf*

Frame:
Segmente des Prozedurkellers
für Klausel-Aufrufe

frame(i).call ⇐*Aufrufendes goal aus*
frame(i).father ⇐*Vaterklausel*

Falls Choicepoint:
frame(i).next_clause ⇐*nächste alternative Klausel*
frame(i).back ⇐*vorheriger Choicepoint*
frame(i).trail ⇐*trail-Segment*

Frame

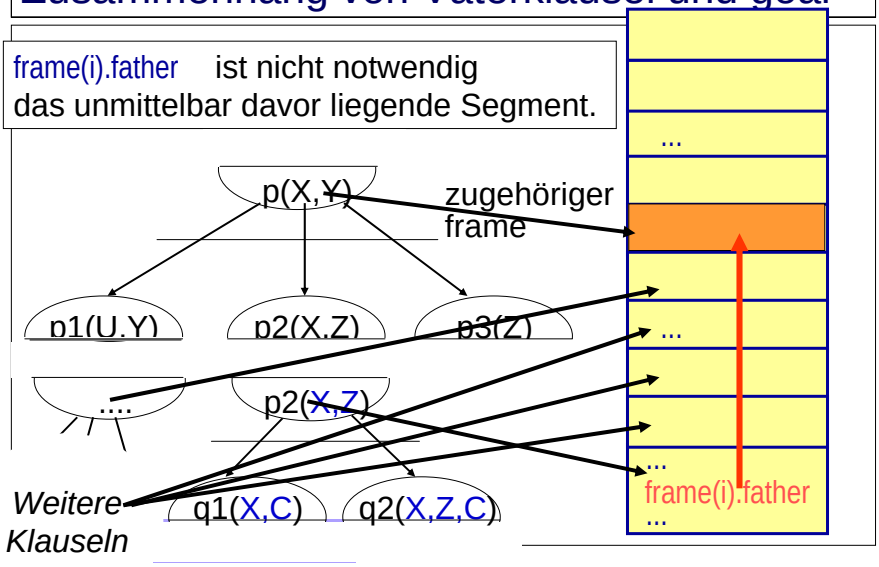
Aufbau bei Klauselaufruf
während Unifikation („matching“)

Streichen beim Backtracking
(alle Segmente oberhalb des
jüngsten Choice point
werden gestrichen)

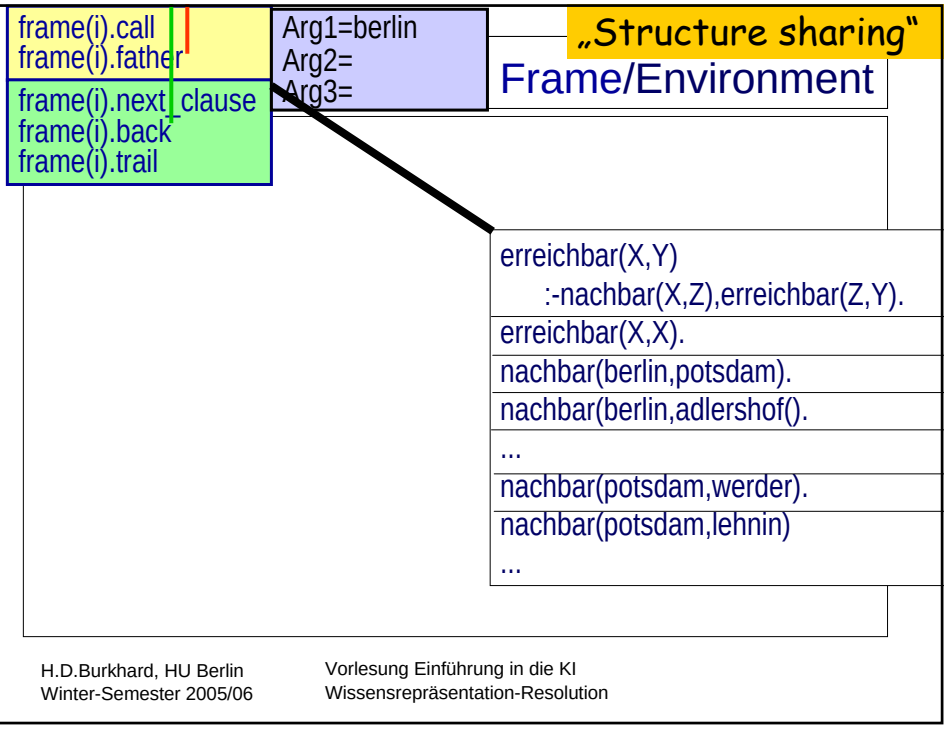
frame(t).call
frame(t).father
---
frame(t).next_clause
frame(t).back
frame(t).trail

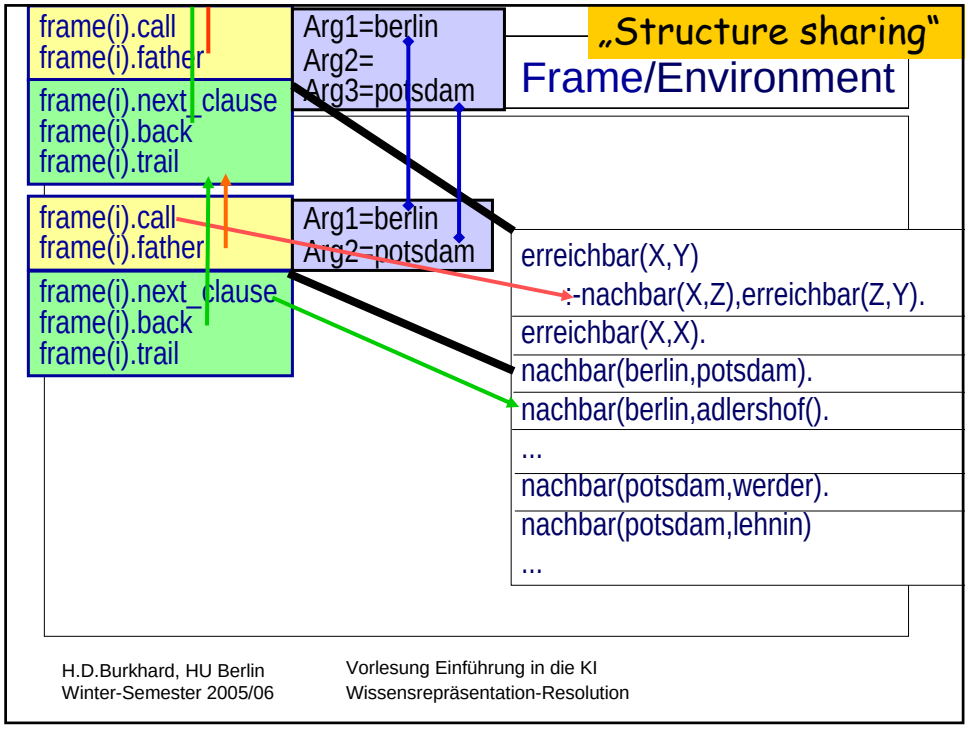
Zusammenhang von Vaterklausel und goal

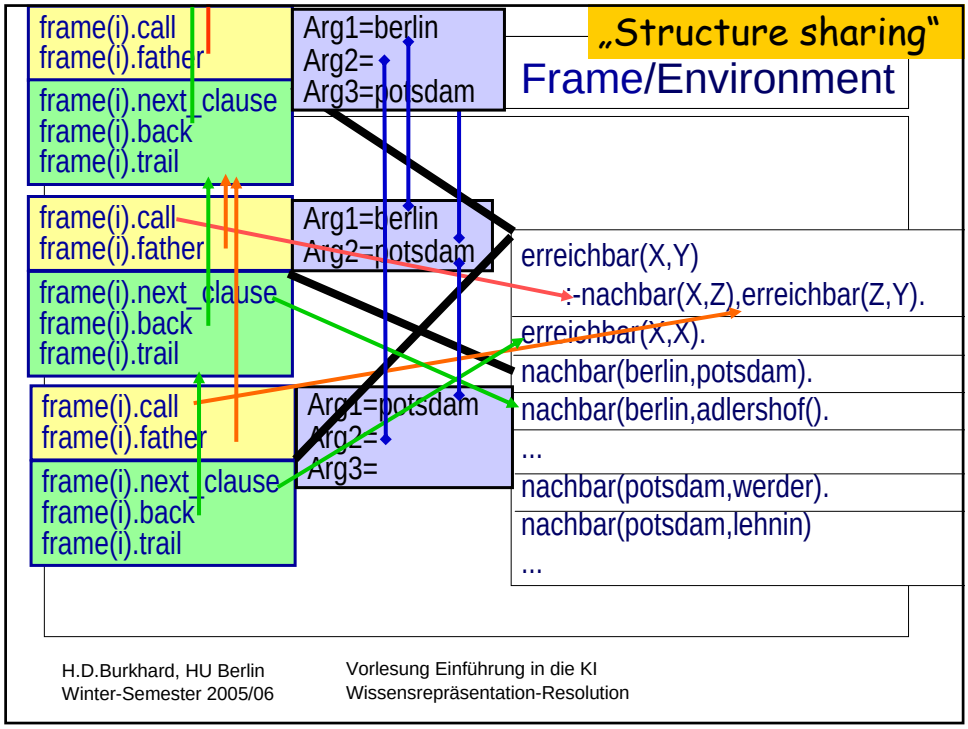
frame(i).father ist nicht notwendig das unmittelbar davor liegende Segment.

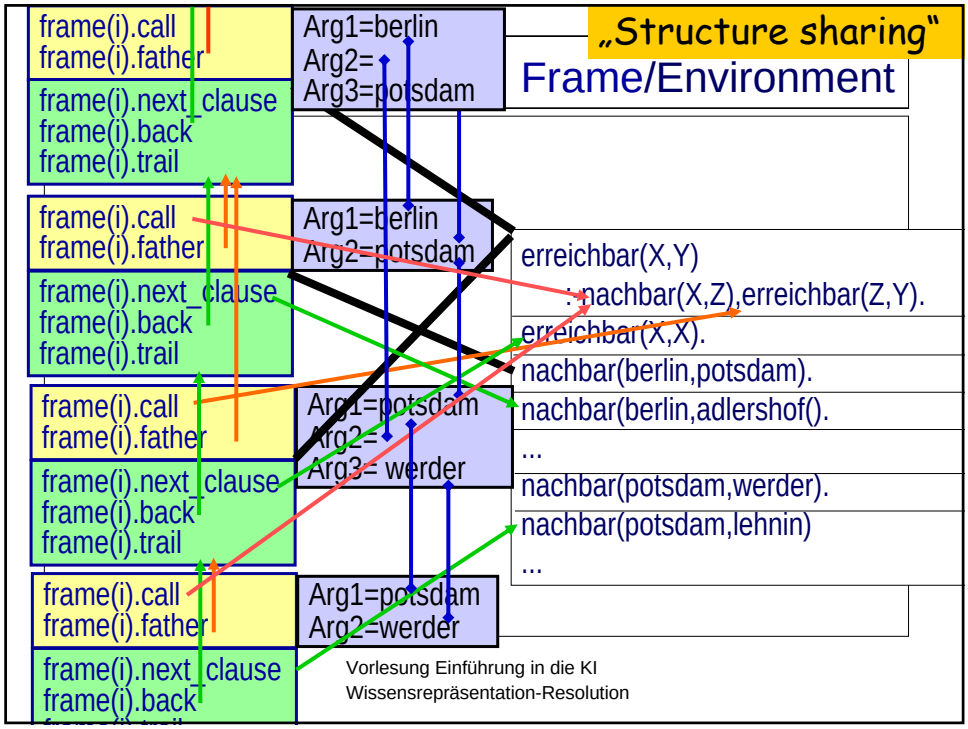


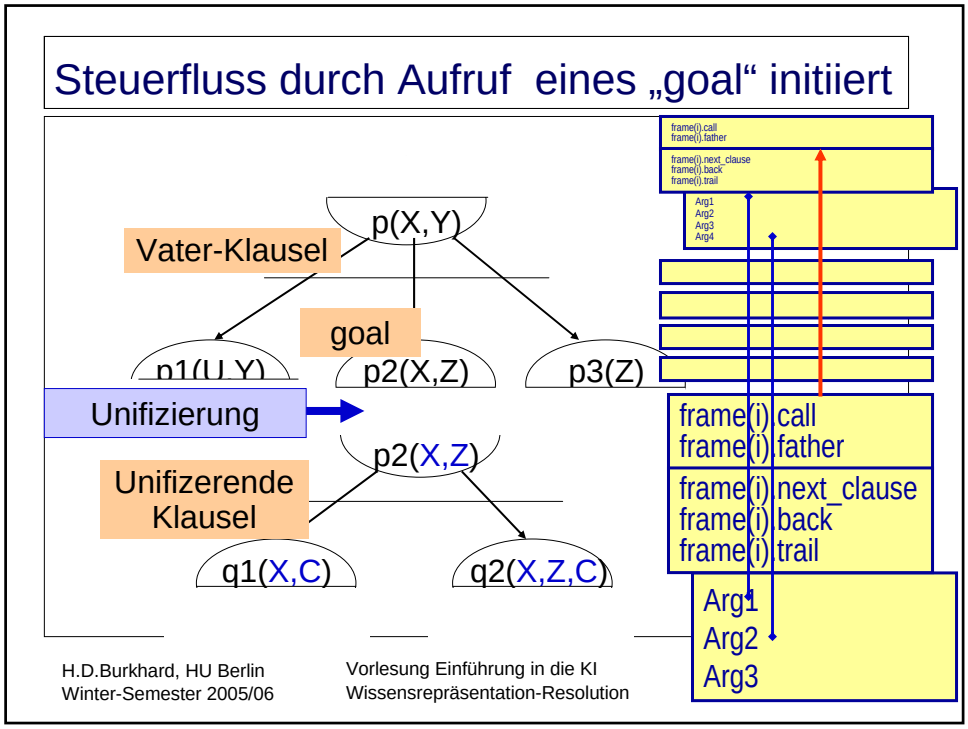
„Structure sharing"




„Structure sharing"
Frame/Environment
frame(i).call
frame(i).father
frame(i).next_clause
frame(i).back
frame(i).trail
Arg1=berlin
Arg2=
Arg3=potsdam
frame(i).call
frame(i).father
frame(i).next_clause
frame(i).back
frame(i).trail
Arg1=berlin
Arg2=potsdam
erreichbar(X,Y)
:-nachbar(X,Z),erreichbar(Z,Y).
erreichbar(X,X).
nachbar(berlin,potsdam).
nachbar(berlin,adlershof().
...
nachbar(potsdam,werder).
nachbar(potsdam,lehnin)
...
H.D.Burkhard, HU Berlin
Winter-Semester 2005/06
Vorlesung Einführung in die KI
Wissensrepräsentation-Resolution


„Structure sharing"
Frame/Environment
frame(i).call
frame(i).father
frame(i).next_clause
frame(i).back
frame(i).trail
Arg1=berlin
Arg2=
Arg3=potsdam
frame(i).call
frame(i).father
frame(i).next_clause
frame(i).back
frame(i).trail
Arg1=berlin
Arg2=potsdam
frame(i).call
frame(i).father
frame(i).next_clause
frame(i).back
frame(i).trail
Arg1=potsdam
Arg2=
Arg3=
erreichbar(X,Y)
:-nachbar(X,Z),erreichbar(Z,Y).
erreichbar(X,X).
nachbar(berlin,potsdam).
nachbar(berlin,adlershof().
...
nachbar(potsdam,werder).
nachbar(potsdam,lehnin)
...
H.D.Burkhard, HU Berlin
Winter-Semester 2005/06
Vorlesung Einführung in die KI
Wissensrepräsentation-Resolution


„Structure sharing“
Frame/Environment
frame(i).call
frame(i).father
frame(i).next_clause
frame(i).back
frame(i).trail
Arg1=berlin
Arg2=
Arg3=potsdam
Arg1=berlin
Arg2=potsdam
Arg1=potsdam
Arg2=
Arg3= werder
Arg1=potsdam
Arg2=werder
erreichbar(X,Y)
:-nachbar(X,Z),erreichbar(Z,Y).
erreichbar(X,X).
nachbar(berlin,potsdam).
nachbar(berlin,adlershof().
...
nachbar(potsdam,werder).
nachbar(potsdam,lehnin)
...
Vorlesung Einführung in die KI
Wissensrepräsentation-Resolution


Steuerfluss durch Aufruf eines „goal“ initiiert
p(X,Y)
Vater-Klausel
goal
p1(U,Y)
p2(X,Z)
p3(Z)
Unifizierung
p2(X,Z)
Unifizierende Klausel
q1(X,C)
q2(X,Z,C)
frame(i).call
frame(i).father
frame(i).next_clause
frame(i).back
frame(i).trail
Arg1
Arg2
Arg3
H.D.Burkhard, HU Berlin
Winter-Semester 2005/06
Vorlesung Einführung in die KI
Wissensrepräsentation-Resolution

Zustand während Abarbeitung

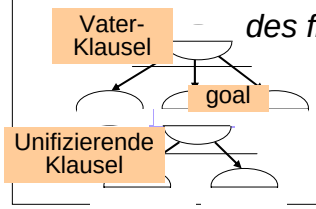
current_call ⇐*aktuelles (aufrufendes) goal*

current_frame ⇐*frame der Vaterklausel von goal*

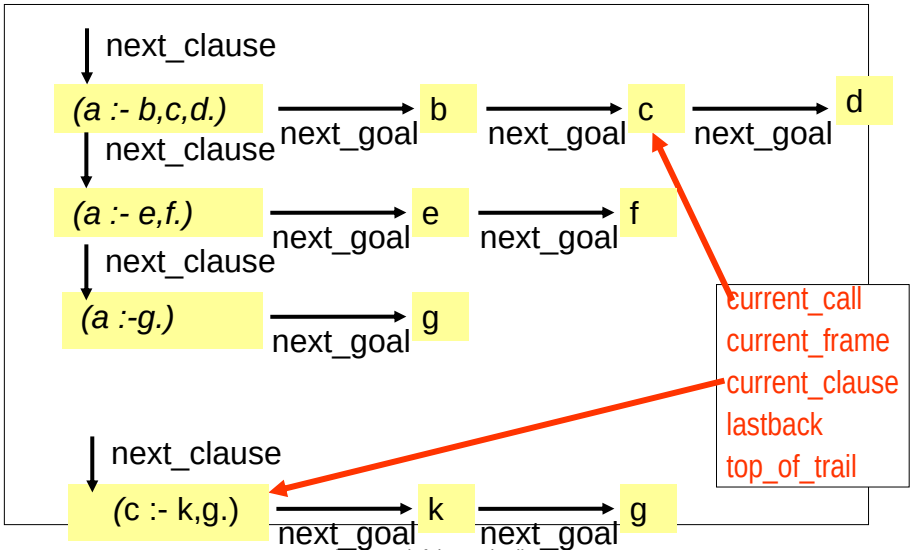
current_clause ⇐*mit goal unifizierende Klausel*

lastback ⇐*frame des jüngsten choicepoint*

top_of_trail ⇐*top des trail-stacks (protokolliert Bindungen, die nicht beim Backtracking durch Kontraktion des frame-stacks gelöst werden können)*



Referenzen auf Programm zur Laufzeit



0.Schritt (Aufruf eines goals)

Zustand:
current_call
current_frame
current_clause
lastback
top_of_trail

Aufrufendes goal referenziert durch current_call

Kandidat zur Unifizierung dieses goals ist erste Klausel der Prozedur bzgl. goal-funktor (sofern sie existiert):

current_clause := Referenz auf diese Klausel

Falls keine entsprechende Klausel existiert:
Weiter mit Schritt 3b.

current_call
current_frame
current_clause
lastback
top_of_trail

1.Schritt: Frame anlegen

current_call
current_frame
current_clause
lastback
top_of_trail

Frame für Klausel anlegen

frame(t+1).call := current_call

frame(t+1).father := current_frame

current_frame := Referenz auf Frame für Klausel

current_call
current_frame
current_clause
lastback
top_of_trail

1.Schritt: Erweiterung bei Choice-point

Falls (alternative) Klausel existiert
d.h. von aktueller Klausel ausgehende
Referenz next_clause ≠ NIL :
Referenzen für Choice-Point anlegen

current_call
current_frame
current_clause
lastback
top_of_trail

frame(t+1).back:=last_back

frame(t+1).next_clause:=next_clause

frame(t+1).trail:=top_of_trail

last_back := current_frame
(Referenz auf Frame für aktuelle Klausel)

current_call
current_frame
current_clause
lastback
top_of_trail

2. Schitt: Unifikationsversuch

Match des aufrufenden goal (current_call)
mit aktueller Klausel (current_clause)

Bindungen von Variablen erfolgen in
frame(t+1).father (environment im Frame für
Vaterklausel des aufrufenden goals)
current_frame (dazu neu angelegtes environment
im Frame für aktuelle Klausel)

Für Bindungen, die bei einem Backtracking durch
Kellerrücksetzen nicht gelöst werden können:
In trail protokollieren, top_of_trail weitersetzen

Bei komplexen Argumenten müssen auch die
entsprechenden Strukturen angelegt werden.

current_call
current_frame
current_clause
lastback
top_of_trail

Schritt 3a: Falls Unifikation erfolgreich

Nächstes goal bestimmen (für Bearbeitung in frame(t+2))

current_call :=first_call in body of current_clause

(evtl. NIL falls Fakt)

falls current_call ≠ NIL

weiter in Schritt 0 (Anlegen von frame(t+2)

current_call
current_frame
current_clause
lastback
top_of_trail

Schritt 3a (erfolgreiche Unifikation, Fortsetzung)

falls current_call = NIL (d.h. current_clause war ein Fakt):
Nächstes goal ergibt sich aus offenen subgoals in früheren Klauseln

WHILE current_call = NIL DO

IF current_frame = „top_of_frame“ THEN weiter Schritt 4a:ERFOLG

ELSE current_call := next_goal in current_frame.call („rechter Bruder“)

current_frame:=current_frame.father

Fakt bzw. später: Ende der Klausel

weiter Schritt 0

current_call
current_frame
current_clause
lastback
top_of_trail

Schritt 3b: Unifikation nicht erfolgreich

Backtracking:

Alternative Klauseln im jüngsten choicepoint anwenden

Falls lastback=NIL : Weiter bei Schritt 4b (FAILURE)

Falls lastback ≠ NIL :

current_call := lastback.call
current_frame = lastback.father
current_clause:= lastback.clause
lastback:=lastback.back

Zurücksetzen des Prodezurkellers:
- Stellt frühere Aufrufsituation her.
- Löscht Bindungen in jüngeren environments.

Bindungen gemäß trail lösen und trail zurücksetzen bis lastback.top_of_trail

top_of_trail := lastback.top_of_trail

current_call
current_frame
current_clause
lastback
top_of_trail

Schritt 4a: ERFOLG

current_frame = „top_of_frame“
(Segment des Aufrufs)

d.h. es gibt keine weiteren unerfüllten subgoals.

Ausgabe:

Bindungen der Variablen in „top_of_frame“

bzw. „yes“, falls Anfrage ohne Variable

Prozedurkeller enthält i.a. weitere frames (mit choice points) für offene alternative Beweisversuche.
Mit Eingabe „ ; “ werden diese aktiviert: weiter bei Schritt 3b

Schritt 4b: MISSERFOLG

lastback=NIL

Es gibt keine alternativen Beweismöglichkeiten für die noch offenen subgoals:

Der Beweisversuch ist fehlgeschlagen.

Ausgabe:

„no“

Interpreter setzt folgende Strategien um

SLD-Resolution

Structure sharing

Backtrack-Konzept für Tiefe-Zuerst-Suche

Implementierung des Cut

`!/0` gelingt stets und löscht Choice-Points für
- –aktuelle Klausel
- –subgoals im Klauselkörper, die vor dem Cut stehen
- –subgoals dieser subgoals usw.

Gefundene Lösung wird ,,eingefroren“
Alternativen für Backtracking entfallen

current_call
current_frame
current_clause
lastback
top_of_trail

⇐*frame des jüngsten choicepoint*

Muss korrigiert werden

Implementierung des Cut

Reduktion der Abarbeitungsschritte 0-3:
- Kein frame für goal „cut“ anlegen
- Keine Unifizierung
- Falls choicepoint bei Vaterklausel:
  lastback:=current_frame.last_back
  Sonst: lastback:= frame.last_back für jüngsten davorliegenden frame mit Choicepoint
- current_call weitersetzen
- weiter bei Schritt 3a

current_call
current_frame
current_clause
lastback
top_of_trail

⇐*frame des jüngsten choicepoint*

Optimierung des Laufzeitkellers

Bei Beendigung deterministischer Aufrufe
(DCO = deterministic call optimization)

Bei Aufruf des letztes Goals einer Klausel
(LCO = last call optimization)

- Speziell für Rekursion an letzter Stelle
  erreichbar(X,Y):-nachbar(X,Z),erreichbar(Z,Y).
- Voraussetzung: deterministische Aufrufe

Optimierung deterministischer Aufrufe

Idee: Frames einsparen, falls nicht mehr benötigt

Prolog-Laufzeitkeller enthält frames für alle Klauseln im aktuellen Beweisbaum für

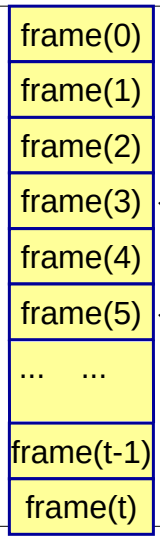
Variablenbindungen zwischen Subgoal und Vaterklausel

- Einsparung möglich, wenn Variablenbindungen an environments älterer Klauseln (am Ende der Derefenzierungskette) erfolgen

Information zum Backtracking (alternative Klauseln)

- Einsparung möglich, wenn kein Backtracking mehr erfolgt: „Deterministischer Aufruf"

Deterministischer Aufruf



⇐ *Jüngster choice point*

⇐ *deterministischer Aufruf*

⇐*frames der subgoals bzgl. frame(5)*

Werden nicht mehr benötigt und können überschrieben werden

Ein Aufruf heißt deterministisch, falls nach seiner Abarbeitung der jüngste Choice-Point älter als dieser Aufruf ist.

Ergänzung von Schritt 3a für DCO

WHILE current_call = NIL DO

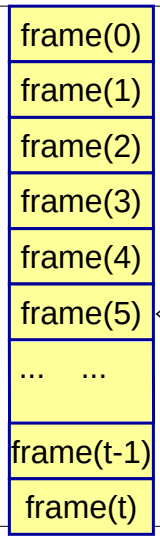
IF current_frame = „top_of_frame“ THEN weiter Schritt 4a:ERFOLG

ELSE current_call := next_goal in current_frame.call („rechter Bruder“)

IF lastback älter als current_frame
THEN frame-Keller freigeben
bis einschließlich current_frame

current_frame:=current_frame.father

Deterministischer Aufruf



⇐ *Jüngster choice point*

⇐*Deterministischer Aufruf*

⇐ *frames der subgoals*

Ein Aufruf heißt deterministisch, falls nach seiner Abarbeitung der jüngste Choice-Point älter als dieser Aufruf ist.

Müssen ebenfalls deterministisch sein
(DCO in tieferen Schichten bereits erfolgt)

Unterstützung von DCO

Cut löscht choice-points und macht dadurch Aufrufe deterministisch

Indexierung der Klauseln nach:

- Funktor
- erstes Argument

Interpreter kann das ausnutzen:
Alternativen (choicepoints)
nur bei Unifizierbarkeit
bzgl. erstem Argument

Durch geschickten Einsatz
von cut
und geeignete Wahl des ersten Arguments
kann Programm DCO unterstützen

Optimierung des letzten Subgoal-Aufrufs

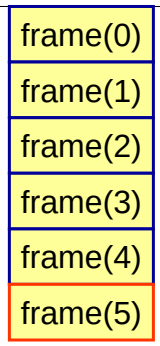
Idee für LCO:

Nach Bildung von frame und environment des letzten Subgoals einer Vater-Klausel werden frame und environment der Vater-Klausel nicht mehr benötigt

Voraussetzungen:

- Geeignete Form der Variablenbindungen (wie DCO)
- Aufruf der Vater-Klausel ist deterministisch (jüngster choice point älter als Vaterklausel)

Implementierung von LCO



Gemeinsam mit DCO implementieren.

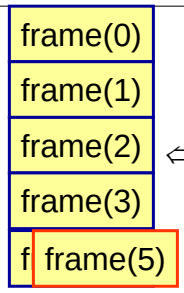
⇐ *Jüngster choice point*

⇐*Deterministischer Aufruf*

⇐*Aufruf des letzten subgoals*

⇐ *frames der weiteren subgoals bereits mit DCO gelöscht*

Implementierung von LCO



Gemeinsam mit DCO implementieren.

⇐ *Jüngster choice point*

⇐*Frame des letzten subgoals überschreibt frame der Vaterklausel*

Tail-Optimierung

Reduzierung des Speicherbedarfs mittels DCO/LCO:

Rekursiver Aufruf als letztes Teilziel

```
erreichbar(X,Y) :- nachbar(X,Z), erreichbar(Z,Y).
```

Aufrufe deterministisch

- Alternativen ggf. davor

```
erreichbar(X,X).
erreichbar(X,Y) :- nachbar(X,Z), erreichbar(Z,Y).
```

- evtl. cut geeignet einsetzen

Fazit: Leistungsfähigkeit PK1

Gut ausgebauter Kalkül, Klarheit von Begriffen und Verfahren.
Ersatz inhaltlicher Begriffe (Gültigkeit, Folgern)
durch formale Begriffe und Verfahren (Ableitbarkeit).

Nicht ausgedrückt werden können:
Quantifizierung von Relationen und Funktionen:
- für alle Funktionen mit ... gilt ...
- manchmal Ersatz durch Schema (vgl. Induktionsaxiom)

Logiken höherer Stufe
– erlauben Quantifizierung von Relationen und Funktionen
– sind nicht axiomatisierbar

Leistungsfähigkeit PK1

Erweiterungen/Modifikationen notwendig für
„Defaultwissen“:
- für gewöhnlich gilt ... *(Nicht-Monotonie des Folgerns!)*

Modale, temporale Operatoren:
- es ist möglich, dass ...
- ich nehme an, dass ...
- ich weiß, dass du weißt, dass ich weiß . . .
- irgendwann gilt ...
  (evtl. Umschreibung: „es existiert Zeitpunkt t mit ...“)

nicht-extensionale Operatoren
nicht austauschbar z.B. „Mann im Mond“ / „quadratischer Kreis“

differenziertere Wahrheitswerte (mehrwertige Logik)
- vgl. Fuzzy-Logik