Unerfüllbarkeitsnachweis

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

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

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

ist die leere Klausel („Widerspruch“)

Unerfüllbarkeitsnachweis

Beweisidee: Herbrand-Modelle

- Satz von Herbrand: Falls Klausel erfüllbar, so schon mit Grundtermen („Herbrand-Modell“: Konstante, Funktionen, keine Variablen) erfüllbar.
- Es reicht, alle durch Grundsubstitutionen (Konstante, Funktionen) erzeugbaren aussagenlogischen Formeln untersuchen (abzählbar viele).
- Resolution für abzählbar viele Formeln durchführen. Dabei Unifikation als verzögerte Substitution (lazy evaluation).

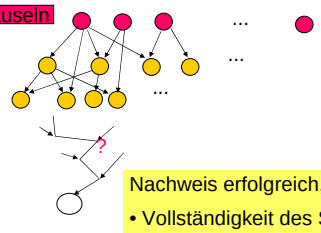
Unerfüllbarkeitsnachweis

Nachweis

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

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

Klauseln



Nachweis erfolgreich, wenn gefunden wird

• Vollständigkeit des Suchverfahrens?

• i.a. unendlicher Suchraum

Resolutionsverfahren

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

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

Darstellung als (erfüllbarkeitsäquivalente) Klauselmenge

Kl = { { L_{ij} | j = 1, ...m_i } | 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 (**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

i.a. unendlicher Suchraum:

Resolutionsverfahren liefert Lösung im Fall von H $\in$ Fl (**X**)

(bzw. Unerfüllbarkeit der entsprechenden Klauselmenge)

in endlich vielen Schritten bei geeignetem Suchverfahren

(z.B. Breite-Zuerst).

Im Fall von H $\notin$ Fl (**X**)

(bzw. Erfüllbarkeit der entsprechenden Klauselmenge)

dagegen evtl. kein Resultat

(unendliches Resolvieren ohne Erreichen von).

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 über

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

ist semantisch äquivalent zu

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

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):

Resolutionsverfahren

ist ein Negativer Testkalkül

verwendet

- spezielle Normalformen (Klauseln)
- Resolutionsregel (Unifikation, Resolventen)

widerlegungsvollständig

- wenn H unerfüllbar, so mit Resolution ableitbar

und widerlegungskorrekt

- wenn mit Resolution ableitbar, so H unerfüllbar

H unerfüllbar gdw. mit Resolution ableitbar

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 gewährleisten.

Spezielle Resolutionsstrategien

- Atomformel: *positives Literal*,
  negierte Atomformel: *negatives Literal*.
- K*lausel heißt negativ*, wenn sie nur negative Literale enthält.
- *Klausel heißt definit*, falls sie genau ein positives Literal (und evtl. noch negative Literale) enthält.
- 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 die im vorigen Schritt erzeugte Resolvente.

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

Einheitsresolution:
Resolution benutzt mindestens eine ein-elementige Klausel.

Spezielle Resolutionsstrategien

P-Resolution, N-Resolution, lineare 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“
L = lineare Resolution
D = definite Klauseln

- Programm **P** sei eine Menge definiter Klauseln.
- Ein SLD-Widerlegungsbeweis für eine (positive) Ziel-Klausel G aus dem Programm **P** ist SLD-Ableitung von aus $\mathbf{P} \cup \{\neg G\}$

- Start mit negativer Klausel $\neg G$.
- In jedem Schritt wird eine negative mit einer definiten Klausel aus dem Programm **P** resolviert.
- Alle Resolventen sind negativ (Spezialfall der **linearen** Resolution, Input-Resolution, N-Resolution).

SLD-Resolution für HORN-Klauseln

Jeder Resolutionsschritt der SLD-Resolution benutzt
- eine negative (vorherige Resolvente) und
- eine **definite** Klausel (aus Programm **P**).

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 eine existierende Lösung. -) oder
•Tiefe-Zuerst.

SLD-Resolution für HORN-Klauseln

σ_i sei 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.

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.

PROLOG

Logische Programmiersprache.

Algorithmus = Logik + Steuerung

HORN-Klauseln + programmiersprachliche Konstrukte:
- E/A-Funktionen
- meta-logische Funktionen (Programm-Modifikation)
- Eingriff in Suchprozedur (Cut)

Steuerung durch Interpreter mit Auswahlstrategie:
1. links vor rechts
2. oben vor unten

und Suchstrategie:
Tiefe-Zuerst.

Prolog-Interpreter

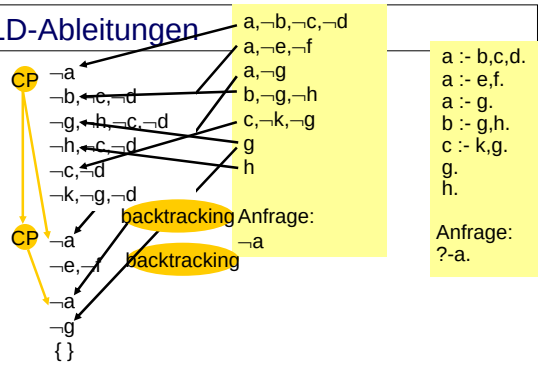
Umsetzung des SLD-Verfahrens als Zustandsraumsuche (Tiefe-Zuerst)

Verwendung des Backtracking-Prinzips (Organisiert durch Prozedurkeller)

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 Aufruf 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.

Environment

für Argumente einer Klausel werden Speicherbereiche angelegt.

Bindung erfolgt bei Unifikation durch Verweise an Argumente von (in der Regel) älteren Klauseln bzw. an Konstante.

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

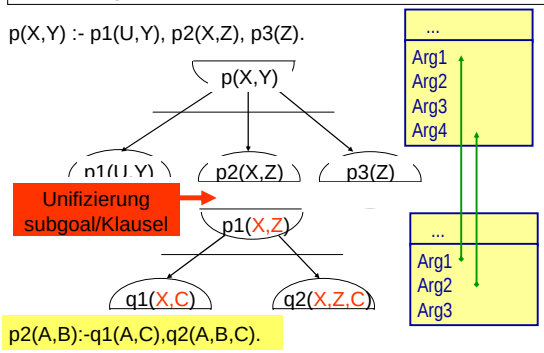
Arg1
Arg2
Arg3

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

Arg1
Arg2

Bindungen

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



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

Environment

Bindungen führen 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

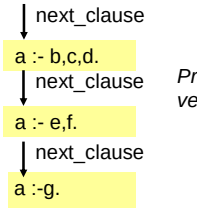
Beim Backtracking entfallen viele Bindungen durch streichen der Segmente

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

Organisation der Prozeduren

Prozedur a

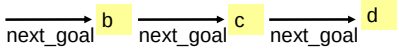
a :- b,c,d.
a :- e,f.
a :- g.



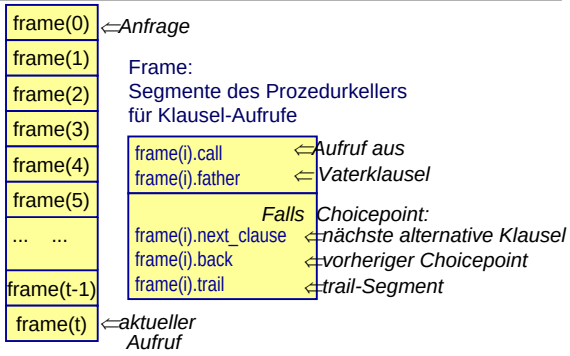
Prozedur als verkettete Liste der Klauseln

Klausel als Liste von goals

a :- b,c,d.



Frame: Prozedurkeller (local stack)



Frame

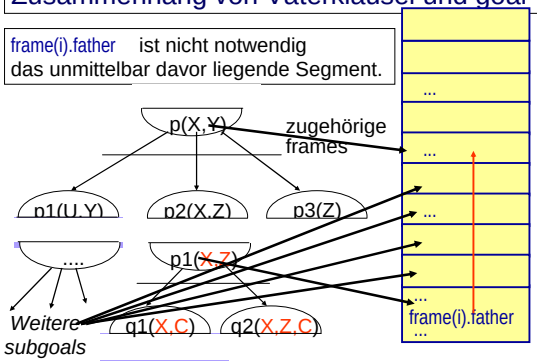
Aufbau bei Klauselaufruf
während Unifikation („matching“)

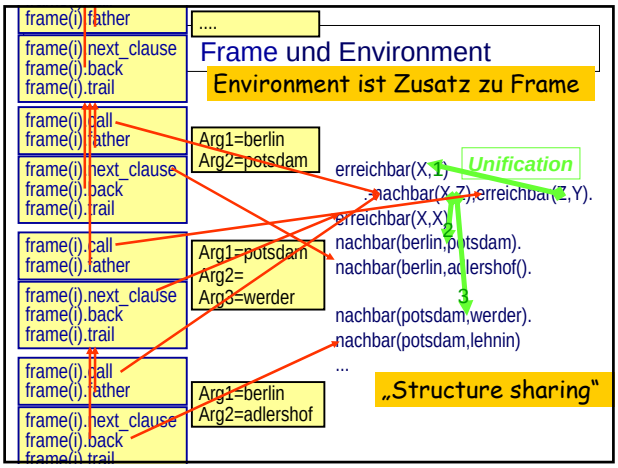
Streichen beim Backtracking
(alle Segmente oberhalb des jüngsten Choice point werden gestrichen)

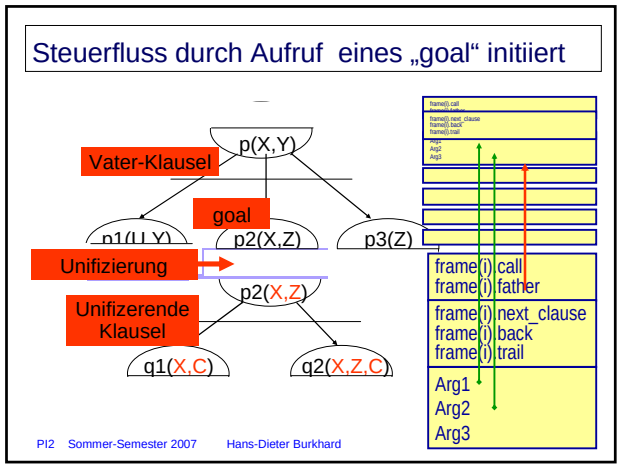
frame(t).call
frame(t).father
---
frame(t).next_clause
frame(t).back
frame(t).trail

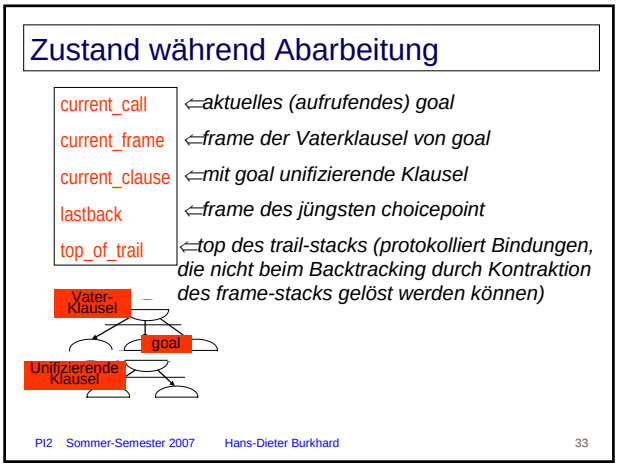
Zusammenhang von Vaterklausel und goal

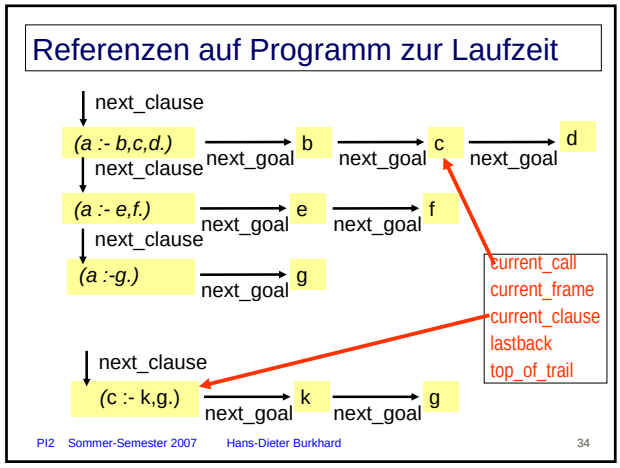
frame(i).father ist nicht notwendig das unmittelbar davor liegende Segment.



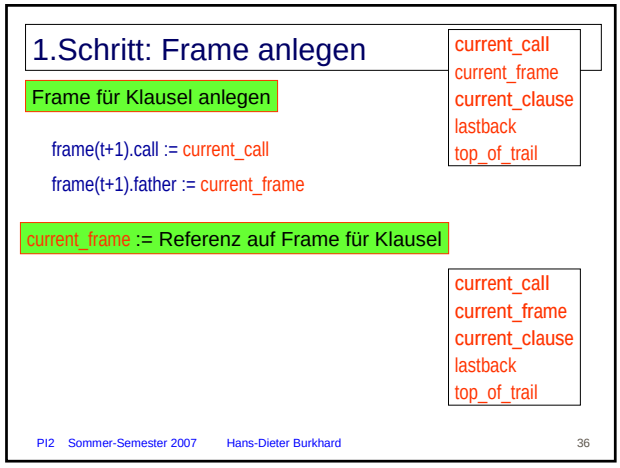

Frame und Environment
Environment ist Zusatz zu Frame
frame(i).father
frame(i).next_clause
frame(i).back
frame(i).trail
....
Arg1=berlin
Arg2=potsdam
Arg1=potsdam
Arg2=
Arg3=werder
Arg1=berlin
Arg2=adlershof
Unification
erreichbar(X,Y) :- nachbar(X,Z),erreichbar(Z,Y).
erreichbar(X,X).
nachbar(berlin,potsdam).
nachbar(berlin,adlershof().
nachbar(potsdam,werder).
nachbar(potsdam,lehnin)
...
„Structure sharing“


Steuerfluss durch Aufruf eines „goal“ initiiert
Vater-Klausel
Unifizierung
Unifizierende Klausel
goal
p(X,Y)
p1(U,Y)
p2(X,Z)
p3(Z)
p2(X,Z)
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
PI2 Sommer-Semester 2007 Hans-Dieter Burkhard


Zustand ändert sich während Abarbeitung
current_call
current_frame
current_clause
lastback
top_of_trail
⇐aktuelles (aufrufendes) goal
⇐frame der Vaterklausel von goal
⇐mit goal unifizierende Klausel
⇐frame des jüngsten choicepoint
⇐top des trail-stacks (protokolliert Bindungen, die nicht beim Backtracking durch Kontraktion des frame-stacks gelöst werden können)
Vater-Klausel
goal
Unifizierende Klausel
PI2 Sommer-Semester 2007 Hans-Dieter Burkhard 33


Referenzen auf Programm zur Laufzeit
next_clause
(a :- b,c,d.)
next_goal
b
c
d
(a :- e,f.)
e
f
(a :-g.)
g
(c :- k,g.)
k
current_call
current_frame
current_clause
lastback
top_of_trail
PI2 Sommer-Semester 2007 Hans-Dieter Burkhard 34


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:
current_clause := Referenz auf diese Klausel
PI2 Sommer-Semester 2007 Hans-Dieter Burkhard 35


1.Schritt: Frame anlegen
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
PI2 Sommer-Semester 2007 Hans-Dieter Burkhard 36

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

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

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)

Soweit Bindungen nicht „rückwärts“ angelegt 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

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“

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*

Interpreter setzt folgende Strategien um

SLD-Resolution

Structure sharing

Backtrack-Konzept für Tiefe-Zuerst-Suche

Optimierung des Prozedurkellers mittels
- – Deterministic call optimization
- – Last call optimization

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

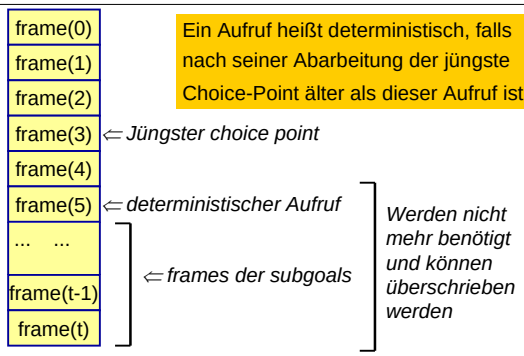
Optimierung deterministischer Aufrufe

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: „Deterministische Aufruf"

Deterministischer Aufruf



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

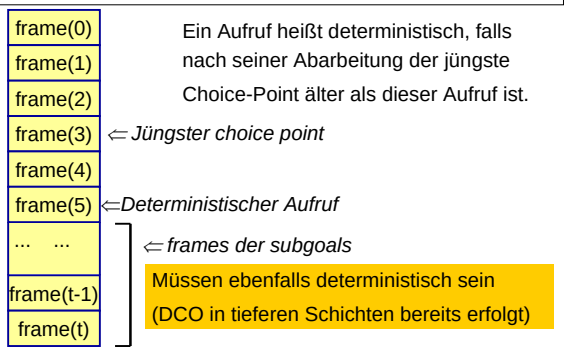
IF current_frame = „top_of_frame“ THEN weiter Schritt 4a:ERFOLG

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



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

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

Optimierung des letzten Subgoal-Aufrufs

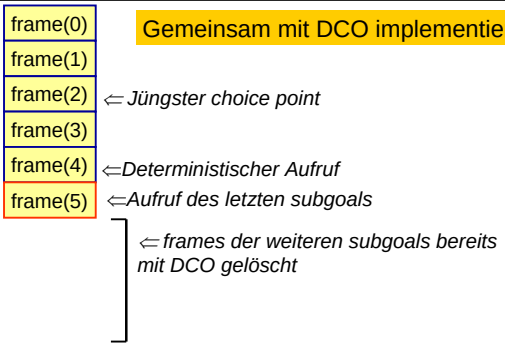
Idee für LCO:

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.

Implementierung von LCO

frame(0)
frame(1)
frame(2) ⇐ *Jüngster choice point*
frame(3)
frame(4) ⇐*Frame des letzten subgoals überschreibt frame der Vaterklausel*

Gemeinsam mit DCO implementieren.

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