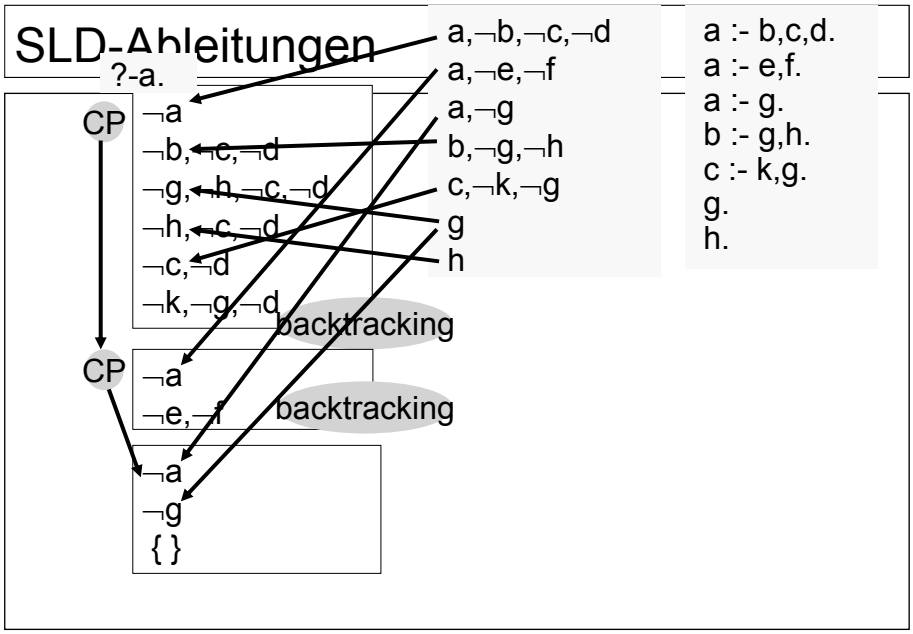
SLD-Ableitungen



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.

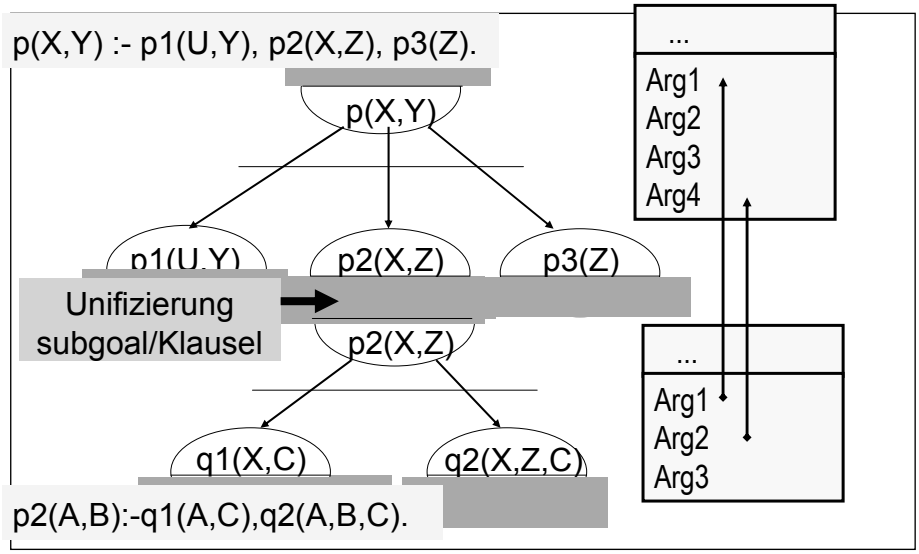
```
erreichbar(X,Y)
    :-nachbar(X,Z),erreichbar(Z,Y).
```

```
Arg1
Arg2
Arg3
```

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

```
Arg1
Arg2
```

Bindungen



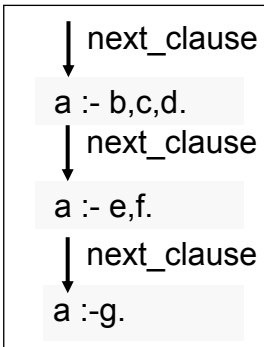
Environment

- 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

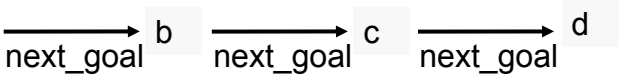
a :- b,c,d.
a :- e,f.
a :- g.



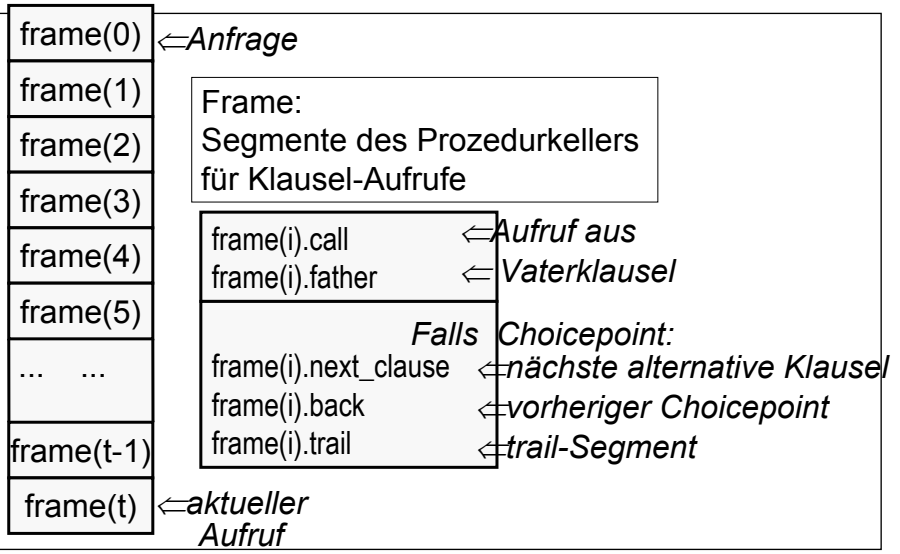
Prozedur als verkettete Liste der Klauseln

Klausel als Liste von goals

a :- b,c,d.



Frame: Prozedurkeller (local stack)



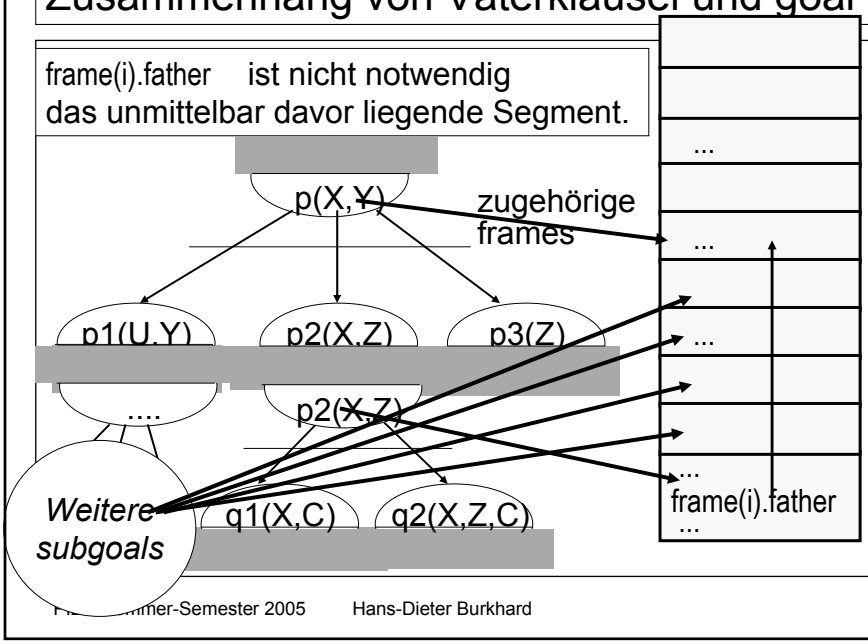
Frame

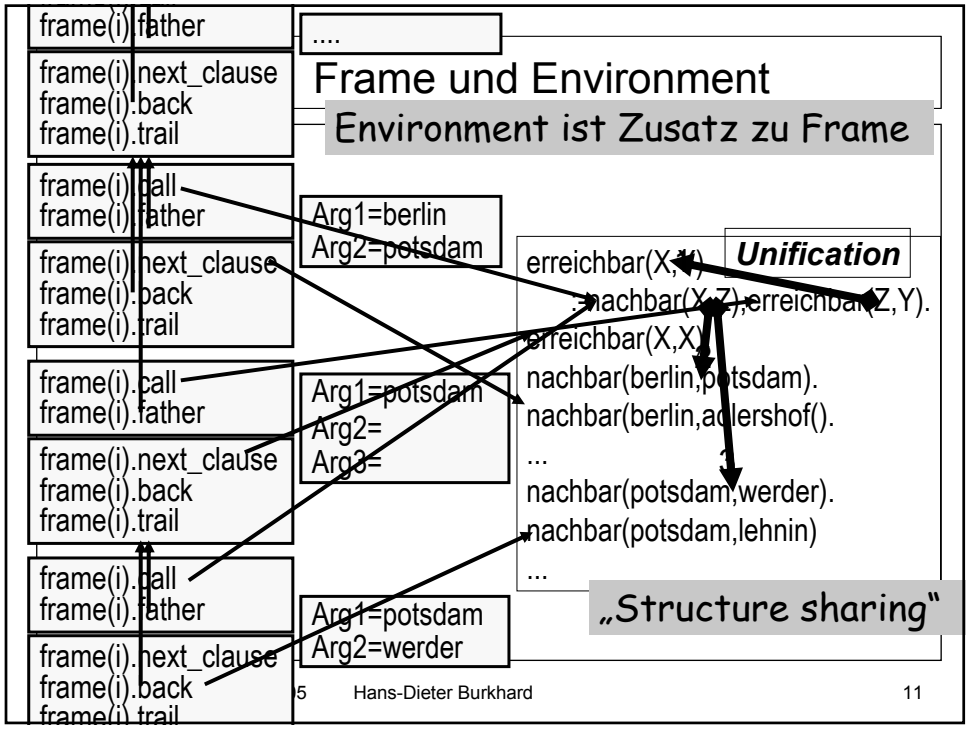
- Aufbau bei Klauselaufruf
  während Unifikation („matching“)
- Streichen beim Backtracking
  (alle Segmente oberhalb des
  jüngsten Choice point
  werden gestrichen)

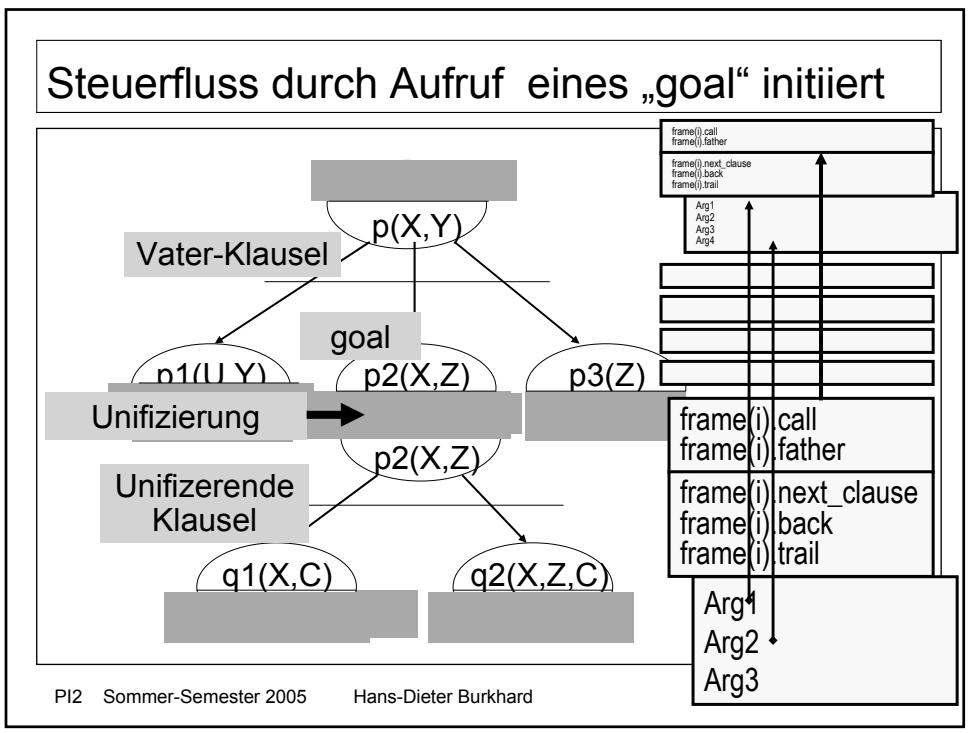
frame(t).call
frame(t).father
---
frame(t).next_clause
frame(t).back
frame(t).trail

Zusammenhang von Vaterklausel und goal

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

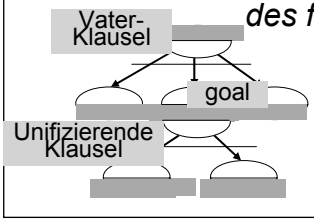



frame(i).father
frame(i).next_clause
frame(i).back
frame(i).trail
frame(i).call
frame(i).father
frame(i).next_clause
frame(i).back
frame(i).trail
frame(i).call
frame(i).father
frame(i).next_clause
frame(i).back
frame(i).trail
frame(i).call
frame(i).father
frame(i).next_clause
frame(i).back
frame(i).trail
....
Frame und Environment
Environment ist Zusatz zu Frame
Arg1=berlin
Arg2=potsdam
Arg1=potsdam
Arg2=
Arg3=
Arg1=potsdam
Arg2=werder
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“

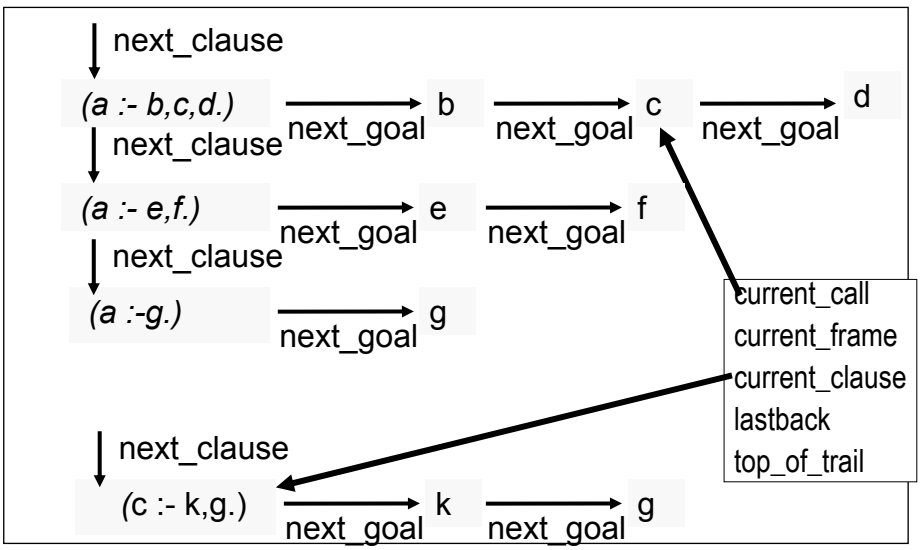

Steuerfluss durch Aufruf eines „goal“ initiiert
p(X,Y)
Vater-Klausel
goal
p1(U,Y)
p2(X,Z)
p3(Z)
Unifizierung
p2(X,Z)
Unifizierende Klausel
q1(X,C)
q2(X,Z,C)
frame(i).call
frame(i).father
frame(i).next_clause
frame(i).back
frame(i).trail
Arg1
Arg2
Arg3
Arg4
frame(i).call
frame(i).father
frame(i).next_clause
frame(i).back
frame(i).trail
Arg1
Arg2
Arg3

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:

current_clause := Referenz auf diese Klausel

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)

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

Bindungen gemäß trail lösen und trail zurücksetzen bis lastback.top_of_trail

top_of_trail := lastback.top_of_trail

Zurücksetzen des Prodezurkellers:

- Stellt frühere Aufrufsituation her.
- Löscht Bindungen in jüngeren environments.

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

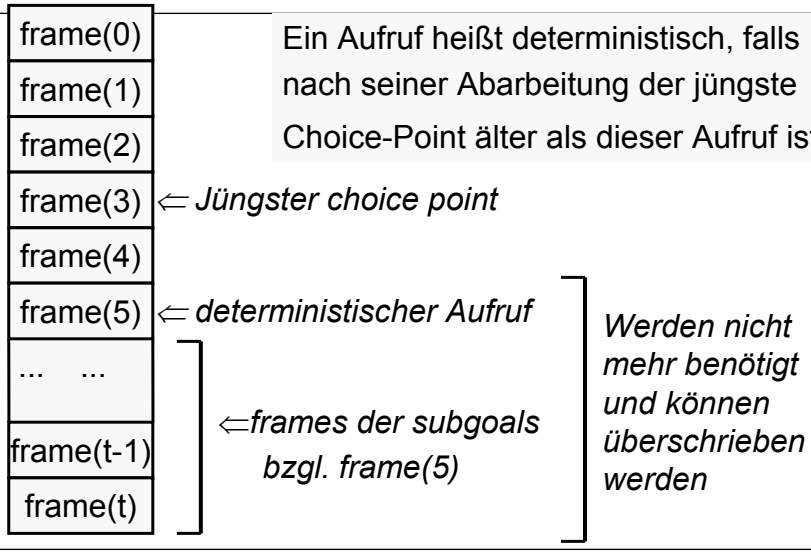
Optimierung deterministischer Aufrufe

Idee: Frames einsparen, falls nicht mehr benötigt

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

1. Variablenbindungen zwischen Subgoal und Vaterklausel
   - Einsparung möglich, wenn Variablenbindungen an environments älterer Klauseln (am Ende der Derefenzierungskette) erfolgen
2. Information zum Backtracking (alternative Klauseln)
   - Einsparung möglich, wenn kein Backtracking mehr erfolgt: „Deterministischer 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“)

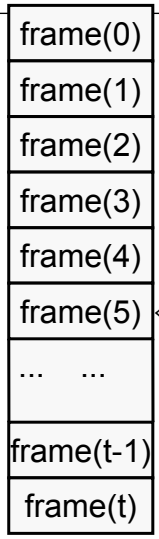
IF lastback älter als current_frame

THEN frame-Keller freigeben

bis einschließlich current_frame

current_frame:=current_frame.father

Deterministischer Aufruf



⇐ *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

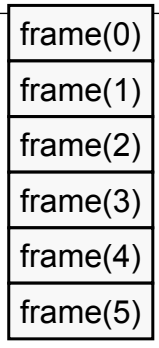
Durch geschickten Einsatz
von cut
und geeignete Wahl des ersten Arguments
kann Programm DCO unterstützen

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.

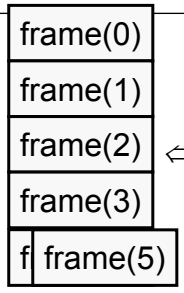
⇐ *Jüngster choice point*

⇐*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