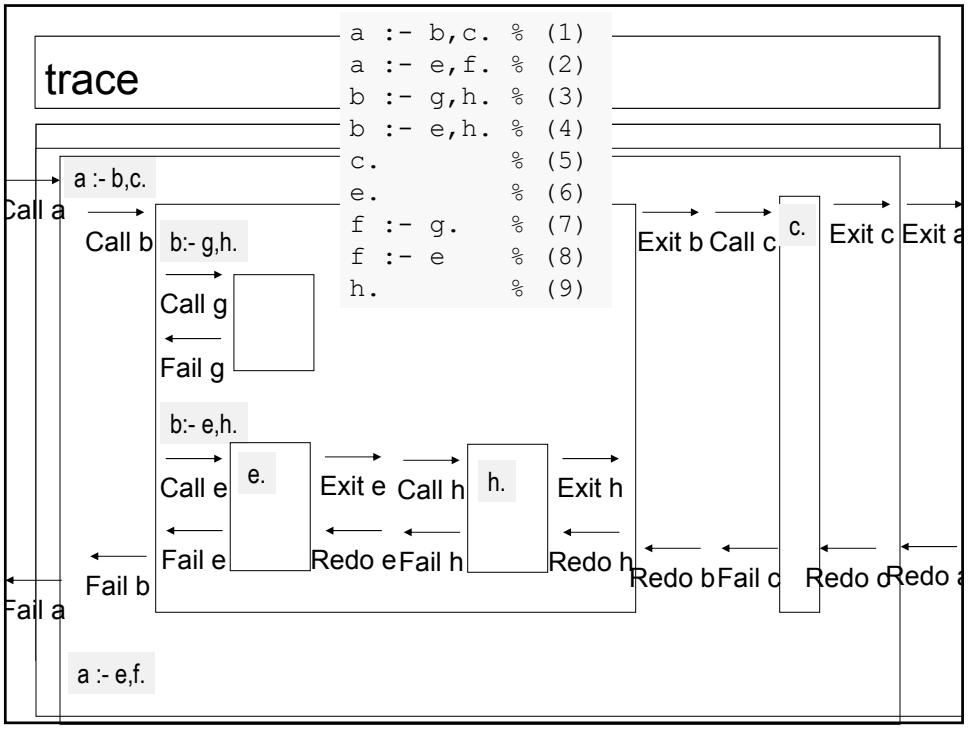
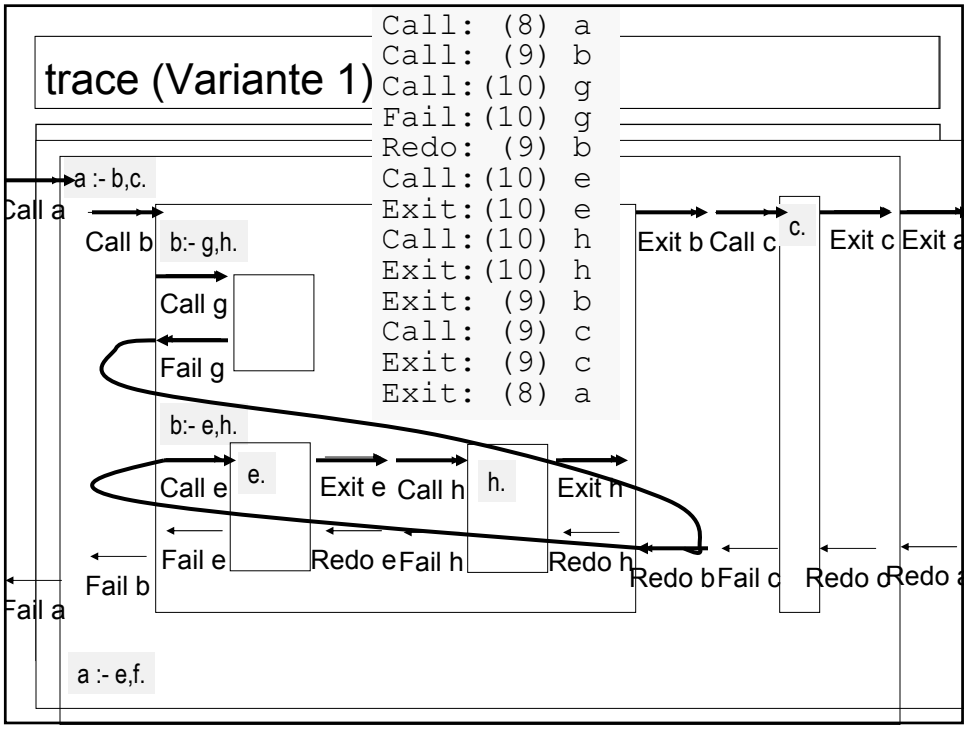
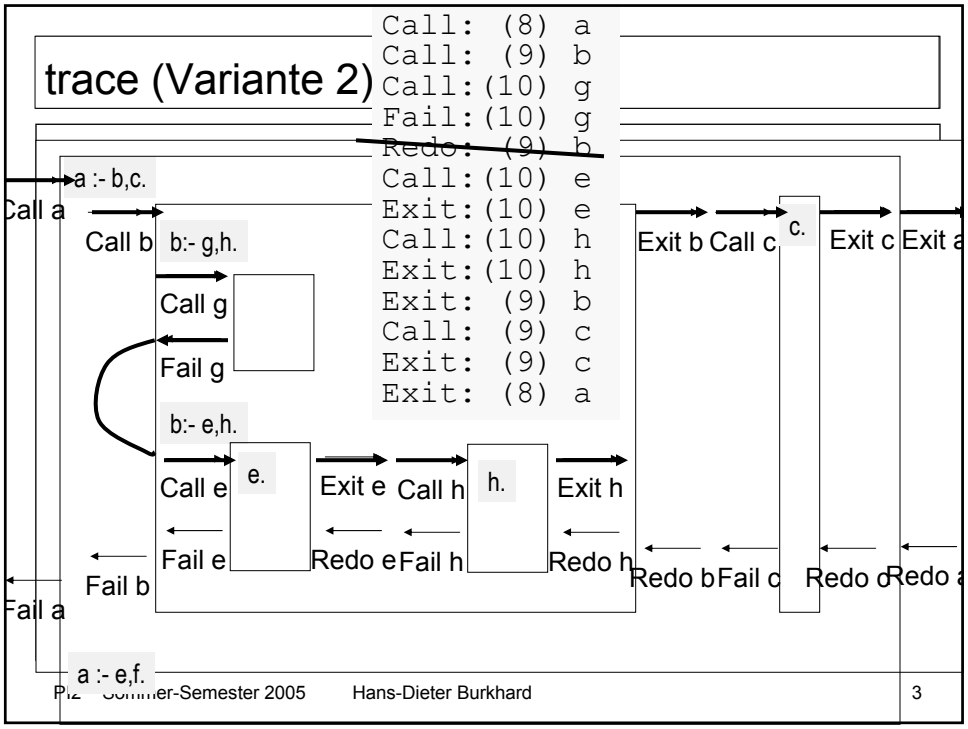
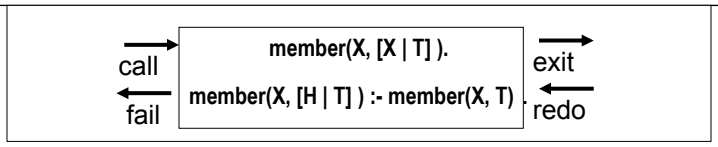

trace
a :- b,c. % (1)
a :- e,f. % (2)
b :- g,h. % (3)
b :- e,h. % (4)
c. % (5)
e. % (6)
f :- g. % (7)
f :- e % (8)
h. % (9)
a :- b,c.
Call a
Call b
b:- g,h.
Call g
Fail g
b:- e,h.
Call e
e.
Exit e
Call h
h.
Exit h
Fail e
Redo e
Fail h
Redo h
Fail b
Fail a
Exit b
Call c
c.
Exit c
Exit a
Redo b
Fail c
Redo c
Redo a
a :- e,f.


trace (Variante 1)
Call: (8) a
Call: (9) b
Call:(10) g
Fail:(10) g
Redo: (9) b
Call:(10) e
Exit:(10) e
Call:(10) h
Exit:(10) h
Exit: (9) b
Call: (9) c
Exit: (9) c
Exit: (8) a
a :- b,c.
Call a
Call b
b:- g,h.
Call g
Fail g
b:- e,h.
Call e
e.
Exit e
Call h
h.
Exit h
Fail e
Redo e
Fail h
Redo h
Fail b
Fail a
Exit b
Call c
c.
Exit c
Exit a
Redo b
Fail c
Redo c
Redo a
a :- e,f.

trace (Variante 2)



Box-Modell, trace (Variante 2)



- **Call:** Start des Beweises einer <u>Klausel</u>
- **Exit:** Erfolgreicher Beweis.
- **Redo:** Alternativer Beweisversuch (Backtracking).
  - Backtracking im Beweis der Klausel
  - aber nicht bei endgültigem Fehlschlag der Klausel
- **Fail:** Keine weiteren Beweismöglichkeiten für Klausel

- Alternative <u>Klausel</u> einer Prozedur als neuer Call

Debugger

- Mit debug/0 wird debug-Modus gestartet:
  - Anzeigen von Trace-Punkten
  - Unterbrechung an Spy-Punkten
    mit Möglichkeit zur interaktiven Arbeit
  - Setzt ggf. Optimierungen außer Kraft
  - Impliziter Aufruf durch spy bzw. trace
  - Abschalten mit nodebug/0

Debugger

- trace/2 setzt/löscht Trace-Punkte
  - für Prädikate (1.Argument)
  - an angebenen Ports (2.Argument):
    +portname bzw. –portname bzw. Liste
- spy/1 bzw. nospy/1 setzt/löscht Spy-Punkte
  - für angegebene Prädikate
- trace/0 setzt überall Trace-Punkte für folgenden Aufruf
- debugging/0 zeigt Status

Variable/Parameter in Prolog

- Imperative Programmiersprachen
  - Überschreiben von Variablen
  - Parameterübergabe bei Prozedur-Aufruf über
    - Wert (value-Parameter)
    - Referenz (reference-Parameter)
- Prolog
  - Dauerhafte Bindung (Instantiierung) von Variablen
  - Parameter-Übergabe durch Unifikation
    - Referenzierung an Variable bzw. Wert

Variable/Parameter: Verzögerte Berechnung

```
[trace]  ?- append1([a,b,c],[1,2],L).
   Call: (6) append1([a, b, c], [1, 2], _G294)
   Call: (7) append1([b, c], [1, 2], _G366)
   Call: (8) append1([c], [1, 2], _G369)
   Call: (9) append1([], [1, 2], _G372)
   Exit: (9) append1([], [1, 2], [1, 2])
   Exit: (8) append1([c], [1, 2], [c, 1, 2])
   Exit: (7) append1([b, c], [1, 2], [b, c, 1, 2])
   Exit: (6) append1([a, b, c], [1, 2], [a, b, c, 1, 2])

L = [a, b, c, 1, 2]
```

Variable/Parameter: Verzögerte Berechnung

```
[trace]  ?- naive_reverse([a,b,c],L).
   Call: (6) naive_reverse([a, b, c], _G287)
   Call: (7) naive_reverse([b, c], _G356)
   Call: (8) naive_reverse([c], _G356)
   Call: (9) naive_reverse([], _G356)
   Exit: (9) naive_reverse([], [])
   Call: (9) append1([], [c], _G360)
   Exit: (9) append1([], [c], [c])
   Exit: (8) naive_reverse([c], [c])
   Call: (8) append1([c], [b], _G363)
   Call: (9) append1([], [b], _G358)
   Exit: (9) append1([], [b], [b])
   Exit: (8) append1([c], [b], [c, b])
   Exit: (7) naive_reverse([b, c], [c, b])
   Call: (7) append1([c, b], [a], _G287)
   Call: (8) append1([b], [a], _G364)
   Call: (9) append1([], [a], _G367)
   Exit: (9) append1([], [a], [a])
   Exit: (8) append1([b], [a], [b, a])
   Exit: (7) append1([c, b], [a], [c, b, a])
   Exit: (6) naive_reverse([a, b, c], [c, b, a])
L = [c, b, a]
```

Variable/Parameter: Verzögerte Berechnung

```
[trace]  ?- reverse1([a,b,c],L).
   Call: (6) reverse1([a, b, c], _G287)
   Call: (7) reverse_acc([a, b, c], [], _G287)
   Call: (8) reverse_acc([b, c], [a], _G287)
   Call: (9) reverse_acc([c], [b, a], _G287)
   Call: (10) reverse_acc([], [c, b, a], _G287)
   Exit: (10) reverse_acc([], [c, b, a], [c, b, a])
   Exit: (9) reverse_acc([c], [b, a], [c, b, a])
   Exit: (8) reverse_acc([b, c], [a], [c, b, a])
   Exit: (7) reverse_acc([a, b, c], [], [c, b, a])
   Exit: (6) reverse1([a, b, c], [c, b, a])

L = [c, b, a]
```

Prädikate

- Prädikate (Relationen) für „gültige Sachverhalte“
- Beweise für Folgerungen/Ableitungen
- Prädikate werden vom Nutzer definiert
- Prolog-System realisiert (spezielles) Beweisverfahren
- Prolog-System besitzt eingebaute (built-in) Prädikate für
  - Programmierumgebung
  - Eingabe/Ausgabe
  - Steuerung der Abarbeitung
  - Programm-Manipulation
  - Term-Operationen
  - Häufig benutzte Prädikate
  - Arithmetik

Eingabe/Ausgabe

- Eröffnen von files:
  tell/1 see/1
- Schließen von files:
  told/0 seen/0
- Erfragen des offenen files (Terminal: user):
  telling/1 seeing/1
- Eingabe/Ausgabe von Termen:
  read/1 write/1
- Eingabe/Ausgabe von Zeichen:
  get/1 put/1

```
see(datei1),lese_von_datei(X),see(user), ...
tell(datei3),schreibe_auf_datei(Z),tell(user),...
```

und weitere ...

```
read( -Term)      ( „Eingabeparameter“)
write(+Term)      („Ausgabeparameter“)
```

Eingabe/Ausgabe

```
kubik :- write('Nächste Zahl bitte: '),  read( X ), bearbeite( X ).

bearbeite( stop ):- !.
bearbeite( N )  :- K is N*N*N,
                   write('Dritte Potenz von '), write( N ),
                   write(' ist '), write( K ), nl,
                   kubik.
```


Steuerung von Programmen

`true, fail, !, not, ;, ,, call, repeat`

- Aufruf: call/1

```
call(Ziel) :-  Ziel .
```

- Endlose Schleifen: repeat/0

```
repeat.
repeat:-repeat.
```

Als System-Prädikat repeat/0 beliebig oft backtrackbar.

```
bearbeite_datei(D) :- see(D),
                      repeat, read(Term),
                      ( Term = end_of_file ; write(Term), fail ),  ! , seen.
```

Cut !/0

Abbruch von Beweisversuchen.

Zur Wiederholung:

```
maximum1(X,Y,X) :-  X >= Y, !.
maximum1(X,Y,Y) :-  X <  Y .
```

Grüner cut

```
maximum2(X,Y,X) :- X >= Y, !.
maximum2(X,Y,Y).
```

Roter cut

Cut !/0

Abbruch von Beweisversuchen.

Nochmal Diskussion grüner/roter cut

```
maximum(X,Y,X)   :-  X >= Y.
maximum(X,Y,Y) :-  X <  Y .
```

```
maximum(X,Y,X) :-  X >= Y, !.
maximum(X,Y,Y) :-  X <  Y .
```

```
maximum(X,Y,X) :- X >= Y, !.
maximum(X,Y,Y).
```

IF X >= Y THEN Max := X
ELSE Max := Y

```
maximum(X,Y,Max) :- X >= Y, !, Max=X.
maximum(X,Y,Max) :-  Max = Y.
```

Unterschiede bei ?-maximum(5,2,2).

Programmierumgebung

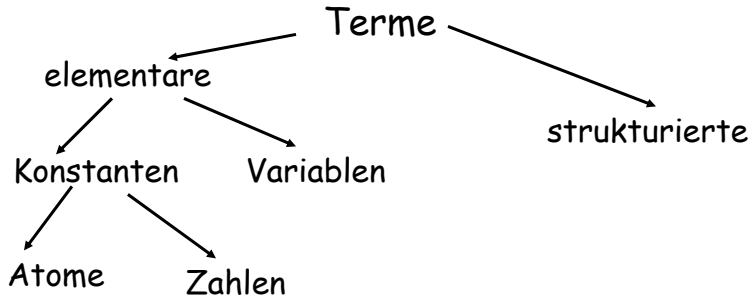
- Laden eines PROLOG-Programms:

  consult/1 oder [filename]

  consult(user) bzw. [user] : Standard-Eingabe bis CTRL D
- Beenden des PROLOG-Systems: halt/0
- Debugging: trace/0, notrace usw.
- Hilfesystem: help/0, help/1, apropos/1
- History-System: set_prolog_flag(history, N).
- Auflisten von Programm-Klauseln: listing/1, listing/0

Analyse und Konstruktion von Termen

- atom/1, integer/1, var/1, nonvar/1, ...
- compound/1, ground/1, ...



Analyse und Konstruktion von Termen

- =.. *(„univ“)* , funktor/3, arg/3 , ...

```
?- funktor_name(arg1,arg2) =.. L .
L = [funktor_name,arg1,arg2]
```

```
..., Goal=..[Funktor|ArgListe], call(Goal), ...
```

```
functor(parent(X,zeus),parent,2)
```

```
arg(2, parent(X,zeus),zeus)
```

- Termvergleich = , == , \= , \==

Programmoperationen

- Hinzufügen von Programm-Klauseln
  `assert/1`, `asserta/1`, `assertz/1`
- Löschen von Programm-Klauseln
  `retract/1`, `retractall/1` , `abolish/1` , `abolish/2`
- Suche nach Programm-Klauseln
  `clause/2`

Beschränkung auf „dynamische“ Klauseln
`dynamic/1`

- Selbstmodifikation von Prolog-Programmen
- Selbstanalyse von Prolog-Programmen

Verwendung von assert

- Doppelberechnungen vermeiden.
  z.B. Wiederverwenden von Zwischenergebnissen:

```
fibo( 0, 0 ).
fibo( 1, 1 ).
fibo( N, M ):-  N1 is N-1, N2 is N-2,
                fibo( N1, M1 ),
                fibo( N2, M2 ),
                M is M1 + M2,
                asserta( fibo( N, M )).
```

Kritik am Programm: Kein Test auf negative Zahlen!

Verwendung von assert

Vorverarbeitung, z.B. Multiplikationstabelle:

```
berechne_tabelle   :-
L=[0,1,2,3,4,5,6,7,8,9],
      member(X,L),member(Y,L),
      Z is X * Y,
      assert(produkt(X,Y,Z)),
      fail;
      true.
```

Willkürliches Abbrechen von Beweisketten

Ersatz von

```
problemloesen
  :- problemloesen_1(ARGUMENTE,Z),
     problemloesen_2(Z).
```

durch

```
problemloesen
  :- problemloesen_1(ARGUMENTE) ,
     assert(zwischenergebnis(Z)), fail.
problemloesen
  :- retract(zwischenergebnis(Z))
     problemloesen_2(Z).
```

Idee:

Der durch problemloesen_1 belegte Laufzeitspeicher wird am Ende der 1.Klausel freigegeben.

Bei Bedarf Cut in problemloesen_1 um Backtracking zu vermeiden.