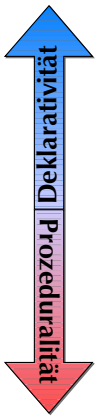
Daten- und Kontrollfluss

Übersicht

- Prozedurale vs. deklarative Semantik
- Datenfluss durch Variablen und Unifikation
- Kontrollfluss
  - Abstraktion, Sequenz, Alternation
- Elimination der Disjunktion
- Spezielle Lenkung des Kontrollflusses
  - Programmiertes Scheitern: fail/0
  - Nicht-Beweisbarkeit: \+/1
  - Suchbäume stutzen: !/0
  - Aufruf: call/1

Prozedurale und deklarative Semantik



Deklarativ: Das Prädikat p ist …

- wahr, falls q und r wahr ist.
- beweisbar, falls q und r beweisbar sind.

```
p :-
   q,
   r.
```

Prozedural: Um das Ziel/die Prozedur p …

- zu erfüllen (*satisfy*), erfülle zuerst q und dann r.
- abzuarbeiten, rufe (*call*) zuerst q und dann r auf.

Prozedurale Interpretation

- hat relevante Reihenfolge!
- ist das, was Prolog-Interpreter kennt/berechnet.

Datenfluss und Modus

Durch textuelle Variablen fliessen Ein- und Rückgabe-werte zwischen Prozeduren hin und her.

- Eingabe-Wert (*Input*): Beim Aufruf instantiierte Variablen
  - Modusdeklaration: **+**
- Rückgabe-Werte (*Output*): Nach Aufruf instantiierte Variablen
  - Modusdeklaration: **–**
- Wenn sowohl Ein- wie Rückgabe möglich ist:
  - Modusdeklaration: **?**

```
% max(+Num,+Num,?Num)
max(X, Y, X) :-
   X >= Y.
max(X, Y, Y) :-
   X < Y.
```

```
?- max(2, 4, Max).
Max = 4

?- max(2, 4, 2).
no
```


Datenfluss und Unifikation

Doppelfunktion von Unifikation beim Beweisen

- **Filter**: Unifikation macht Fallunterscheidungen!
  - Nennt man auch *Pattern Matching* (Mustervergleich)
- **Konstruktor**:Unifikation macht automatischen Aufbau von Datenstrukturen!

Instantiierung beim Aufruf entscheidet, ob Argument Filter oder Konstruktor ist!

```
belegt(ida, vorlesung(ecl1)).
belegt(ida, uebung(pcl1)).
belegt(ida, vorlesung(pcl1)).
belegt(udo, vorlesung(ecl1)).
```

```
?- belegt(ida, vorlesung(X)).
X = ecl1 ;
X = pcl1 ;
no
```

```
?- belegt(X, vorlesung(ecl1)).
X = ida ;
X = udo ;
no
```

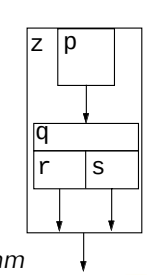
Strukturierung des Kontrollflusses

Prädikatsdefinition, konjugierte Prädikate und **mehrfache Klauseln** stellen die elementaren, prozeduralen Kontrollen zur Verfügung:

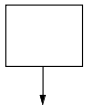
```
z :-
    p,
    q.

q :- r.
q :- s.

p.

r.

s.
```

Abstraktes Programm

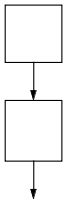
```
?- z.
```



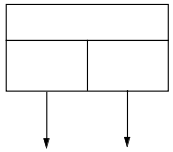
Abstraktion
(Kapselung)



Sequenz
(Abfolge)



Alternation
(Fallunterscheidung)



Disjunktion

```
grossvater(Opa, Enkel) :-
    vater(Opa, Person),
    ( mutter(Person, Enkel)
    ; vater(Person, Enkel)
    ).
```

```
grossvater(Opa, Enkel) :-
    vater(Opa, Person),
    elter(Person, Enkel).

elter(Person, Kind) :-
    mutter(Person, Kind).
elter(Person, Kind) :-
    vater(Person, Kind).
```

Elimination der Disjunktion

- Disjunktionen können immer ersetzt werden, indem die disjunktiv verknüpften Terme zu Klauseln eines neuen Prädikats werden
- Disjunktionen sind wie Klauseldefinitionen: Es findet Backtracking statt inklusive Rückgängigmachen von Variablenbindungen!

Disjunktion und Backtracking

```
person(hans).
person(klara).
person(gabi).
person(kevin).
```

Programm

```
?- person(X).
X = hans    ;
X = klara   ;
X = gabi    ;
X = kevin   ;
no
```

Anfrage

Wie gehen wir vor, um alle Lösungen eines Ziels zu erhalten?

- bereits bekannt: Manuelle Eingabe eines Semikolons
- Anstatt der manuellen Nachfrage möchte man jedoch einen **programmierbaren Mechanismus** haben.

Programmiertes Backtracking: fail/0

Das eingebaute Prädikat fail/0 kann nie erfüllt werden!

- Backtracking kann damit programmiert werden
- Wichtig für Programmiertechnik *Failure-Driven-Loop*
  - Durch Scheitern lassen sich alle möglichen Lösungen ausgeben!

```
person(hans).
person(klara).
person(gabi).
person(kevin).

alle :-
  person(X),
  write(X), nl,
  fail.
```

Programm

```
?- alle.
no
```

Anfrage

```
hans
klara
gabi
kevin
```

Ausgabe

Failure-Driven Loop

Das Fehlschlagen der Anfrage: Ein behebarer Makel

- Eine zusätzliche, bedingungslos erfüllbare Klausel, die erst dann zum Zug kommt, wenn es keine weiteren Lösungen mehr gibt.

```
person(hans).
person(klara).
person(gabi).
person(kevin).
alle :-
  person(X),
  write(X), nl,
  fail.
alle.
```

Programm

```
?- alle.
yes
```

Anfrage

```
hans
klara
gabi
kevin
```

Ausgabe

Nicht-Beweisbarkeit: \+

Bisher konnten wir nur fragen, ob Prolog etwas beweisen kann:

```
person(hans).
person(klara).
```

- Ist Klara eine Person?

```
?- person(klara).
```

Der Präfix-Operator \+ fragt, ob der nachfolgende Term *nicht* bewiesen werden kann:

- Ist es nicht der Fall, dass Gerda eine Person ist?

```
?- \+ person(gerda).
no
```

- Gibt es niemanden, der eine Person ist?

```
?- \+ person(Jemand).
no
```


Nicht-Beweisbarkeit und Negation

```
weiblich(X) :-
  \+ maennlich(X).
maennlich(hans).
```

```
?- weiblich(hans).
no
```

```
?- weiblich(gabi).
yes
```

Beachte: \+ ist keine logische Negation!

- es gibt keine positive Information, dass Gabi weiblich ist
- Gabi ist genauso weiblich wie Hermann nach diesem Programm
- je vollständiger ein Prädikat definiert ist, umso mehr nähert sich die Nicht-Beweisbarkeit der Negation an (*closed world assumption*)

```
?- weiblich(Wer).
no
```

```
?- weiblich(hermann).
yes
```


Zwecklose Alternativen

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

```
?- max(3, 2, Max).
Max = 3
```

Was geschieht bei der Anfrage?

- Beweisversuch mit erster Klausel, der auch gelingt.
- Prolog merkt sich als Entscheidungspunkt die zweite Klausel. Es weiss nicht, dass die beiden Klausel nie gleichzeitig erfüllbar sind.. D.H. dass die beiden Klauseln *deterministisch* sind!

Cut !/0: Exklusive Fallunterscheidung

```
max2(X, Y, X) :-
  X >= Y,
  !.
max2(X, Y, Y) :-
  X < Y,
  !.
```

Mit dem eingebauten Prädikat !\0 lassen sich Klauseln als exklusive Fallunterscheidungen markieren.

- *Der letzte Cut ist eigentlich unnötig!*

Wirkung des Cut

A. Wegschneiden aller alternativen Prädikats-Klauseln *unterhalb* jener, die den Cut enthält

B. Wegschneiden aller alternativen Lösungen für Ziele, die in derselben Klausel *links* vom Cut stehen.

Wirkung des Cut: Ausgangsprogramm

Abstraktes Beispielprogramm

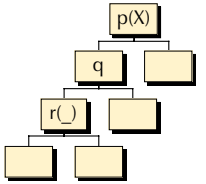
- Die Anfrage ?- p(X). hat 4 Lösungen.

```
p(1) :- q.
p(2).

q :- r(_).
q.

r(1).
r(2).

s.
```

Programm

```
?- p(X).
X = 1 ? ;
X = 1 ? ;
X = 1 ? ;
X = 2 ? ;
no
```

Anfrage

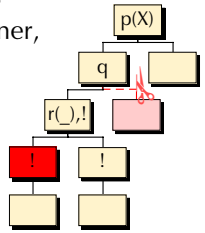


Such-/Beweisbaum

Wirkung des Cut (A)

cut/0 gelingt immer, aber als Nebeneffekt wird Suchbaum gestutzt.

A. Wegschneiden aller alternativen Prädikats-Klauseln *unterhalb* jener, die den Cut enthält.



(A) im Such-/Beweisbaum

```
p(1) :- q.
p(2).

q :- r(_), !.
q.

r(1).
r(2).

s.
```

(A) im Programm

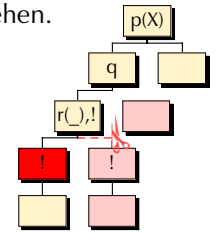
Wirkung des Cut (B)

cut/0 gelingt immer, aber als Nebeneffekt wird Suchbaum gestutzt.

B. Wegschneiden aller alternativen Lösungen für Ziele, die in derselben Klausel *links* vom Cut stehen.

```
?- p(X).
X = 1 ? ;
X = 2 ? ;
no
```

Anfrage



(B) im Such-/Beweisbaum

```
p(1) :- q.
p(2).

q :- r(_), !.
q.

r(1).
r(2).

s.
```

(B) im Programm

Grüne vs. rote Cuts

Der Cut kann nur prozedural verstanden werden!

Grüne Cuts

- schneiden Suchäste ohne Lösungen weg.
- machen Programme effizienter (bei gleicher Lösungsmenge).
- zeigen oft Determinismus an (Kommentar `% green cut`).

Rote Cuts

- schneiden auch Suchäste mit unerwünschten Lösungen weg.
- machen Programme effizienter (bei veränderter Lösungsmenge).
- können u.a. Determinismus erzwingen.
- sind oft schlecht verständlich und heikel in der Verwendung! (Kommentar `% red cut`)

Datenstrukturen zu Aufrufen: call/1

Das eingebaute Prädikat call/1 gelingt, falls sein Argument bewiesen werden kann.

- Daten und Prozeduren sind keine strikt getrennten Welten.
- In der Anfrage "?- call(person(hans))." wäre call/1 redundant.
- call/1 ist sinnvoll, wo das Argument variabel ist:

Zum Beispiel once/1:
Ein Prädikat, das sein Argument aufruft, aber höchstens eine Lösung erzeugt.

```
once(Goal) :-
   call(Goal),
   !.
```


Definition von \+

Unter Verwendung von call/1, !/0 und fail/0 lässt sich Nicht-Beweisbarkeit definieren.

```
:- op(900, fy, \+).
\+ C :- call(C), !, fail.

\+ C.
```

- Damit wird das Antwortverhalten bei weiblich/1 erklärbar:

```
weiblich(X) :-
  \+ maennlich(X).
maennlich(hans).
```

```
?- weiblich(Wer).
no
?- weiblich(hermann).
yes
```