Tabellen und Anfragen (1)

Prolog-Programm:

- % vater(X, Y) bedeute: Y ist Vater von X.
  vater(anke, christoph).
  vater(bernd, christoph).
  vater(christoph, emil).
  vater(dora, gerd).
- % mutter(X, Y) bedeute: Y ist Mutter von X.
  mutter(anke, dora).
  mutter(bernd, dora).
  mutter(christoph, frida).
  mutter(dora, hilde).

Prolog Syntax (1):

- Faktum ::= Prädikat "(" Konstante, ... ")" ".".
- Prädikat ::= Atom.
  Keine Deklaration nötig. Prolog hat kein eingebautes Typsystem.
- Konstante ::= Atom | Zahl.
- Atom ::= Kleinbuchstabe (Buchstabe|Ziffer)*.
- Kommentar ::= "%" ... Zeilenende.
  Viele Prolog-Systeme erlauben alternativ "/*" Kommentar "*/".

Tabellen und Anfragen (2)

Benutzung eines Prolog-Interpreters:

- Jede Prolog-Eingabe ist mit "." zu beenden!
- Prolog-Programm laden: ['eltern.pl'].

  Man versuche auch [eltern], consult('eltern.pl').

  Interaktive Eingabe mit [user], Ende mit Control-D (Dateiende).
- Prolog verlassen: halt.

Prolog-Syntax (2):

- Variable ::= Großbuchstabe (Buchstabe|Ziffer)*.

  "_" zählt als Großbuchstabe.
- Term ::= Variable | Konstante.
- Literal ::= Prädikat "(" Term "," ... ")".

Anfragen:

- vater(anke, christoph). ⟶ yes
- vater(anke, emil). ⟶ no
- vater(anke, X). ⟶ X=christoph
- vater(X, christoph). ⟶ X=anke

  ; ⟶ X=bernd
- vater(X, Y). ⟶ X=anke, Y=christoph

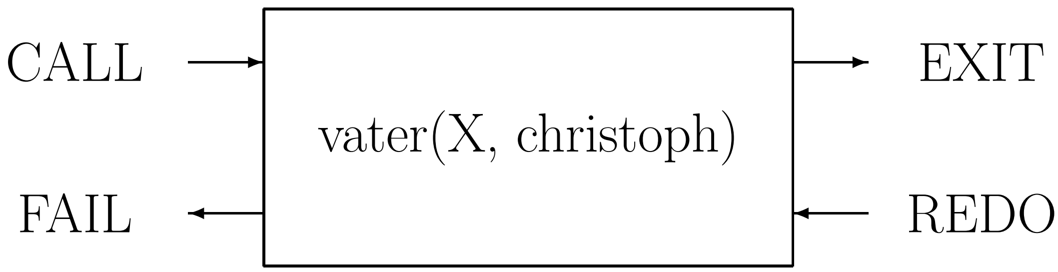
Tabellen und Anfragen (3)

Prädikate:

- Aussagen mit Platzhaltern, Relationen, Tabellen.
- “Prozeduren” eines Prolog-Programms.
- Ein-/Ausgabeparameter nicht festgelegt.
- Nur Eingabeparameter ≅ bool-wertige Funktion.
- Eindeutig bestimmter Ausgabewert ≅ Funktion.
- Mehrere Ausgabewerte: Es wird einer geliefert, bei Bedarf später Alternativen (Backtracking).

Das Vierport-Modell:

- CALL p: Erster Aufruf von p.
- REDO p: Eine weitere Lösung für p wird gesucht.
- EXIT p: Lösung für p gefunden.
- FAIL p: Es gibt keine (weitere) Lösung für p.
- Graphische Darstellung:



Tabellen und Anfragen (4)

Konjunktive Anfragen:

- Logisches "und" wird "," geschrieben.
- vater(X, christoph), X \= anke.
  Für welche X gilt vater(X, christoph) und X$\neq$anke?

Debugger-Ausgabe (Sepia-Prolog):

	(1)	0	CALL	vater(X, christoph)
	(1)	0	*EXIT	vater(anke, christoph)
B	(2)	0	CALL	anke \= anke
B	(2)	0	FAIL	anke \= anke
	(1)	0	REDO	vater(X, christoph)
	(1)	0	EXIT	vater(bernd, christoph)
B	(3)	0	CALL	bernd \= anke
B	(3)	0	EXIT	bernd \= anke

Erläuterungen:

- Erste Spalte: "B" markiert Systemprädikate (built-in predicate).
- Zweite Spalte: Nummer der Prozeduraktivierung.
- Dritte Spalte: Aufruftiefe.
- Vierte Spalte: Port (*EXIT: möglicherweise weitere Lösungen).
- Fünfte Spalte: Prozeduraufruf.

Tabellen und Anfragen (5)

Anonyme Variable:

- Wenn der Ergebniswert nicht benötigt wird, kann man "_" anstelle einer Variable schreiben.
- Jedes Vorkommen von "_" entspricht einer neuen Variablen.
- vater(_, X) ⟶ X=christoph

  Wer ist Vater? (das Kind wird nicht ausgegeben)

Aufgabe:

- Legen Sie eine kleine Geburtstags-Datenbank an:

```
geburtstag(stefan, 9, 8).
geburtstag(peter, 28, 11).
...
```

Formulieren Sie folgende Anfragen in Prolog:

- Wann hat Stefan Geburtstag?
- Wer hat im August Geburtstag?
- Wer hat vom 9.-15. August Geburtstag?

  Es gibt Systemprädikate <, >, =<, >=.
- Welche Personen haben zusammen Geburtstag?

Regeln (1)

Beispiel:

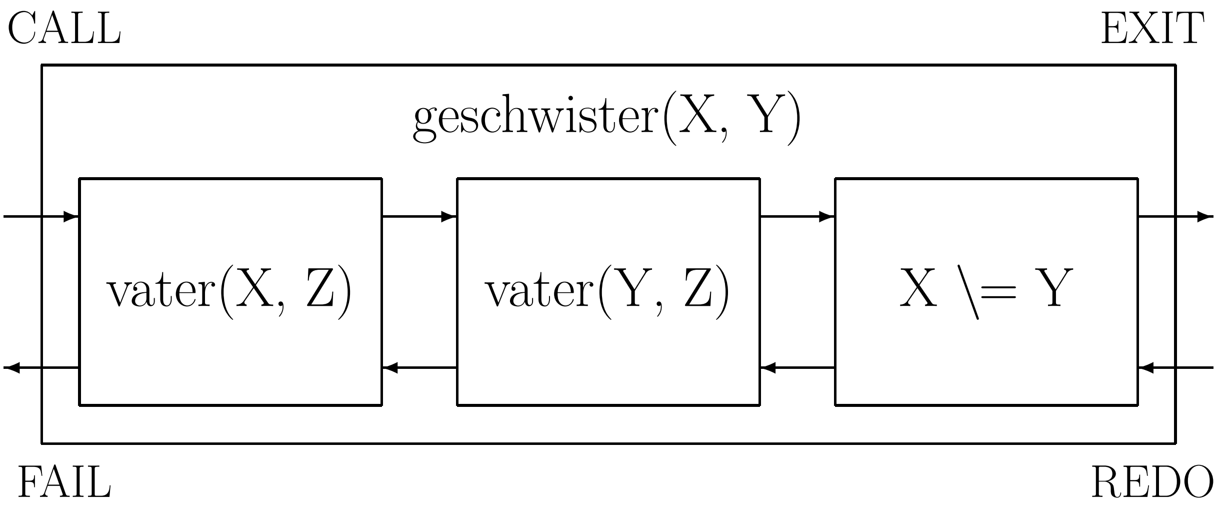
- geschwister(X, Y) :-
  vater(X, Z), vater(Y, Z), X \= Y.
- Regel definiert neues Prädikat durch Anfrage.
  Entsprechend Sichten in relationalen DBen.
- ":-" hat die Bedeutung von "wenn".
- Variablen, die nur im Rumpf auftauchen (hier Z), werden aus dem Anfrageergebnis ausgeblendet.
- X und Y sind Geschwister, wenn es ein Z gibt, das Vater von X und Y ist, und $X \neq Y$ gilt.
- Für alle X, Y, Z gilt: Wenn die Bedingung im Rumpf erfüllt ist, so sind X und Y Geschwister.

Prolog-Syntax (3):

- Regel ::= Kopf ":-" Rumpf ".".
- Kopf ::= Literal.
- Rumpf ::= Anfrage.
- Anfrage ::= Literal "," ....
- Variablennamen lokal in jeder Regel "deklariert".
  D.h. "X" in verschiedenen Regeln sind verschiedene Variablen.

Regeln (2)

Box-Modell:



Abarbeitung von geschwister(anke, Y):

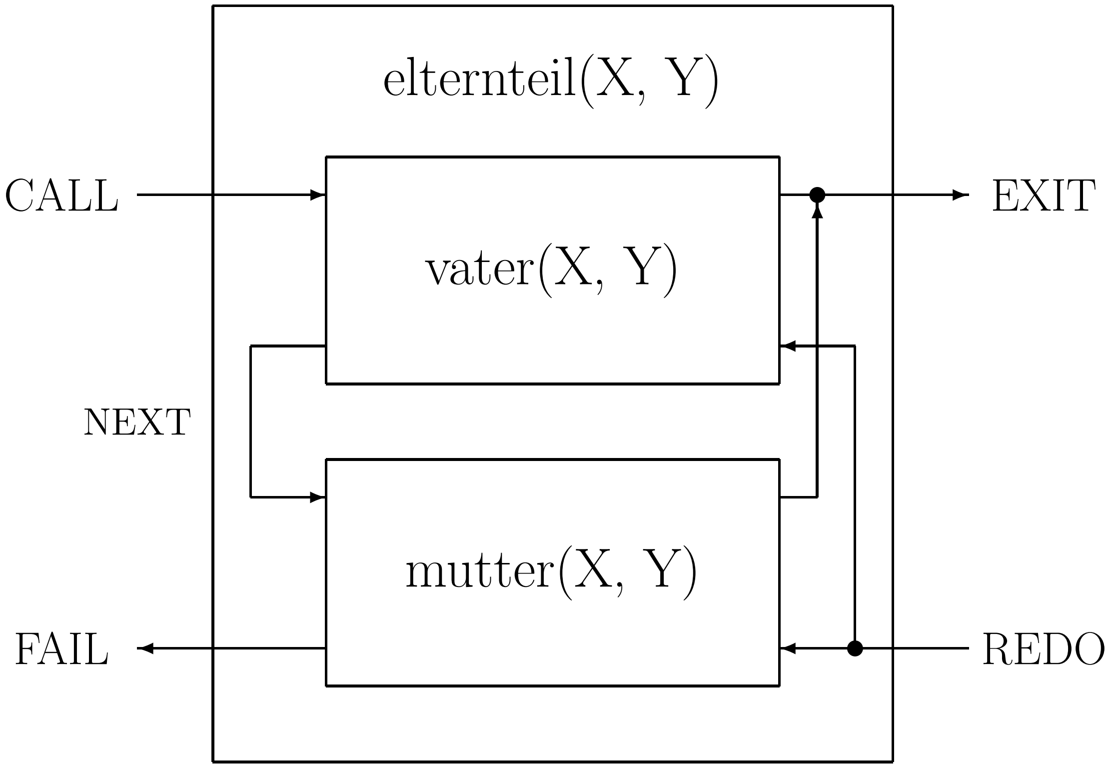
	(1)	0	CALL	geschwister(anke, Y)
	(2)	1	CALL	vater(anke, Z)
	(2)	1	EXIT	vater(anke, christoph)
	(3)	1	CALL	vater(Y, christoph)
	(3)	1	*EXIT	vater(anke, christoph)
B	(4)	1	CALL	anke \= anke
B	(4)	1	FAIL	anke \= anke
	(3)	1	REDO	vater(Y, christoph)
	(3)	1	EXIT	vater(bernd, christoph)
B	(5)	1	CALL	anke \= bernd
B	(5)	1	EXIT	anke \= bernd
	(1)	0	EXIT	geschwister(anke, bernd)

Regeln (3)

Mehrere Regeln über ein Prädikat:

- elternteil(X, Y) :- vater(X, Y).
  elternteil(X, Y) :- mutter(X, Y).
- Antworten beider Regelrümpfe werden vereinigt, entspricht disjunktiver Verknüpfung (oder).
- Für alle X, Y: vater(X, Y) $\Longrightarrow$ elternteil(X, Y).
  Für alle X, Y: mutter(X, Y) $\Longrightarrow$ elternteil(X, Y).

Box-Modell:



Bei REDO: die innere Box, die zuletzt per EXIT verlassen.

Regeln (4)

Beispiel:

- Definierte Prädikate können auch selbst wieder in Regeln verwendet werden.
- grossvater(X, Z) :- elternteil(X, Y), vater(Y, Z).

Rekursive Regel:

- vorfahr(X, Y) :- elternteil(X, Y).
  vorfahr(X, Z) :- elternteil(X, Y), vorfahr(Y, Z).
- Rekursion endet, weil elternteil azyklisch.

Aufgabe:

Es seien noch folgende Prädikate gegeben:

mann(Name).
frau(Name).
ehepaar(Mann, Frau).

Definieren Sie folgende Prädikate:

- Verheiratet (symmetrische Version von ehepaar).
- Tante.
- Fehler (Mütter, die männlich sind, etc.)

Rechenoperationen

Beispiel:

- quadrat(X, Q) :- Q is X * X.
- Variablen auf der rechten Seite von "is" müssen gebunden sein, d.h. quadrat(X, 4) ist unzulässig.
- Beliebter Fehler: "=" anstelle von "is".

  "=" dient zum Patternmatching, "is" zum Rechnen.
- Variablen können nur ein einziges Mal einen Wert zugewiesen bekommen.

  Danach wird die Variable automatisch durch ihren Wert ersetzt.

Schleife:

```
quadrate_bis(N) :-
    N = 0.
quadrate_bis(N) :-
    N > 0,
    M is N - 1,
    quadrate_bis(M),
    quadrat(N, Q),
    write(N), write(' : '), write(Q), nl.
```

Datenstrukturen (1)

Prolog-Syntax (4):

- Einzige Datenstruktur: Variante Records.
- Term ::= Variable | Konstante | Struktur.
- Struktur ::= Funktor "(" Term "," … ")".
- Funktor ::= Atom.

Beispiel:

```
fläche(G, F) :-
    G = quadrat(X),
    F is X * X.
fläche(G, F) :-
    rechteck(X, Y) = G,
    F is X * Y.
```

Alternativ (effizienter):

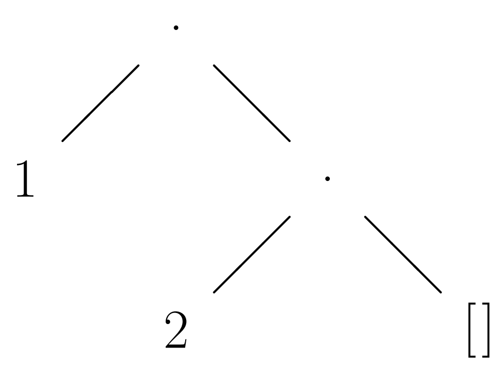
```
fläche(quadrat(X), F) :-
    F is X * X.
fläche(rechteck(X, Y), F) :-
    F is X * Y.
```

Datenstrukturen (2)

Listen:

- Rekursiv aufgebaute Records.
- Konstante "[]": leere Liste.
- Funktor "." (Erstes Element, Rest-Liste).
- D.h. .(1, .(2, [])) ist die Liste "1, 2".

Listendarstellung:



Prolog-Syntax (5):

Prolog versteht folgende Abkürzungen:

- [1,2] $\longrightarrow$.(1, .(2, []))
- [Kopf|Rest] $\longrightarrow$.(Kopf, Rest)
- [Elem1,Elem2|Rest] $\longrightarrow$.(Elem1, .(Elem2, Rest))

Datenstrukturen (3)

Listen-Länge:

```
length([], 0).
length([Elem|Rest], N1) :-
    length(Rest, N),
    N1 is N+1.
```

Element-Beziehung:

```
member(Elem, [Elem|_]).
member(Elem, [_|Rest]) :-
    member(Elem, Rest).
```

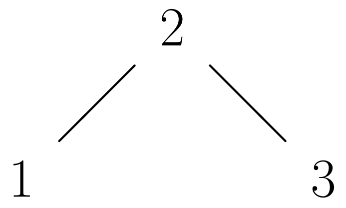
Aufgabe:

Definieren Sie folgende Prädikate:

- append(X, Y, Z): Z ist Konkatenation $X \circ Y$.
  Kann man das Prädikat auch zum Aufspalten verwenden?
- subset(X, Y): Jedes Element von X ist auch in Y enthalten.

Datenstrukturen (4)

Suchbaum:



Darstellung als Term:

- sb(Wert, linker Teilbaum, rechter Teilbaum)
- sb(2, sb(1, nil, nil), sb(3, nil, nil))

Suche in einem Baum:

- ```
  in(X, sb(Wert, Links, Rechts)) :-
          X = Wert.
  in(X, sb(Wert, Links, Rechts)) :-
          X < Wert,
          in(X, Links).
  in(X, sb(Wert, Links, Rechts)) :-
          X > Wert,
          in(X, Rechts).
  ```
- Anfrage: in(2, B), in(1, B), in(3, B).

  Liefert B = sb(2, sb(1, X1, X2), sb(3, X3, X4)).
  ```