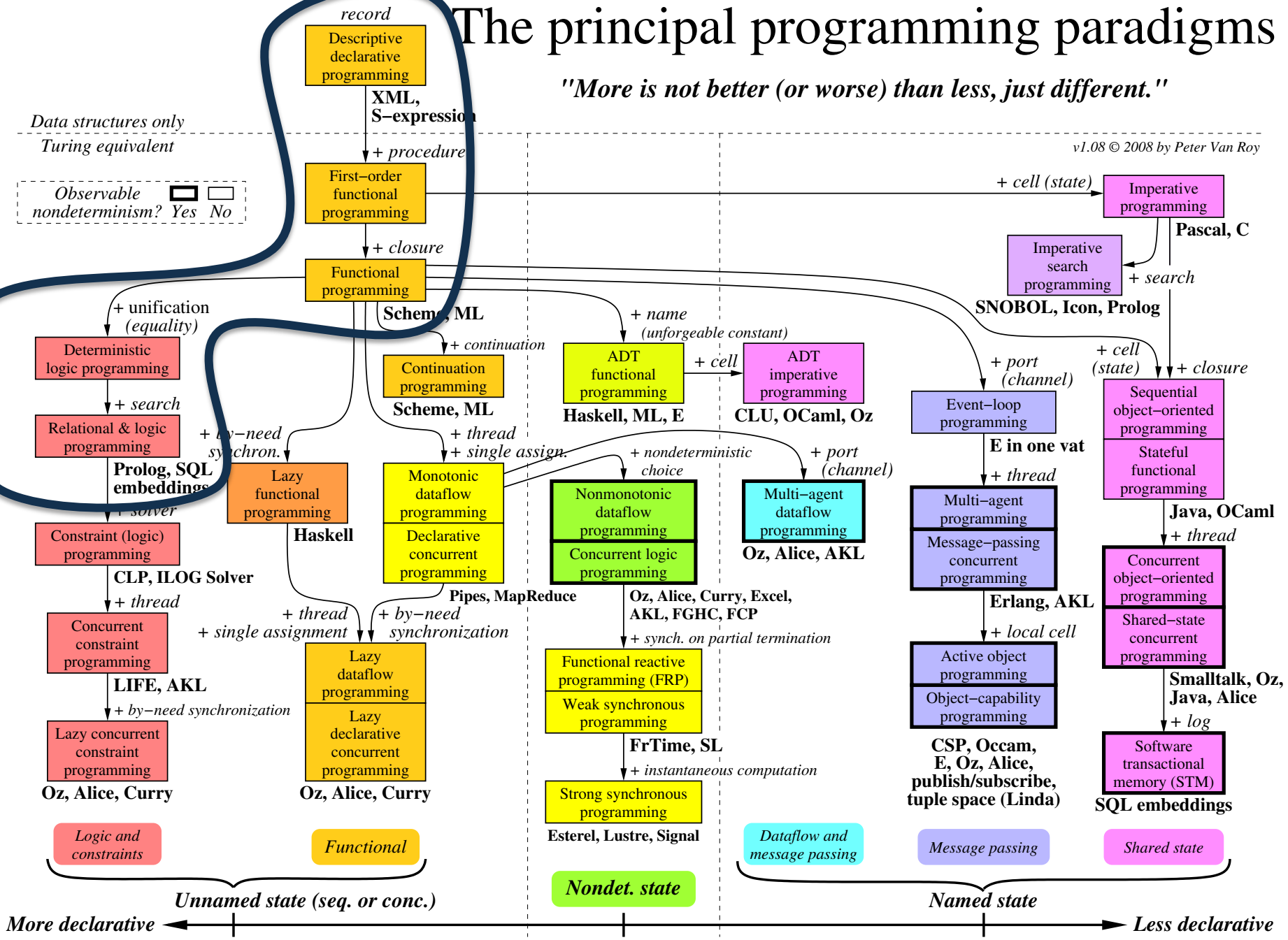
# Prolog

Prof. dr. Kris Luyten
Jo Vermeulen

"Geavanceerde Programmeertechnologie"
Academiejaar 2011-2012

# The principal programming paradigms

*"More is not better (or worse) than less, just different."*



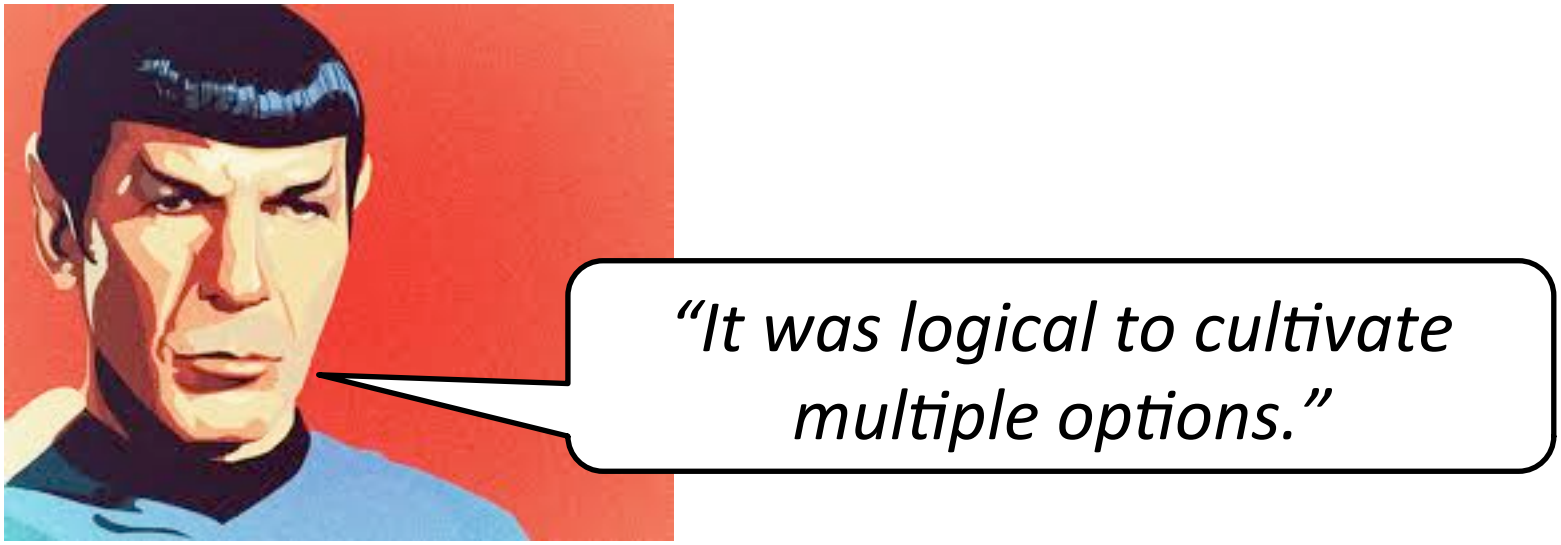

Source: http://www.info.ucl.ac.be/~pvr/paradigms.html

# Prolog

# **Pro**gramma**t**ion en **Log**ique

# **Pro**gramma**t**ion en **Log**ique

°1972




# **Pro**gramma**tion** en **Log**ique

°2230

# **Pro**gramma**t**ion en **Log**ique

Horn Clauses

# **Pro**gramma**tion** en **Log**ique

Horn Clauses

$(a \wedge b \wedge c \wedge \cdots \wedge s \wedge t) \rightarrow u$

# **Pro**gramma**t**ion en **Log**ique

u :- a, b, c, ⋯, s, t

Om te tonen wanneer ***u*** waar is, zoeken we alle toekenningen die **a,b,c,…,s** en **t** waar maken

u :- a, b, c, ⋯, s, t

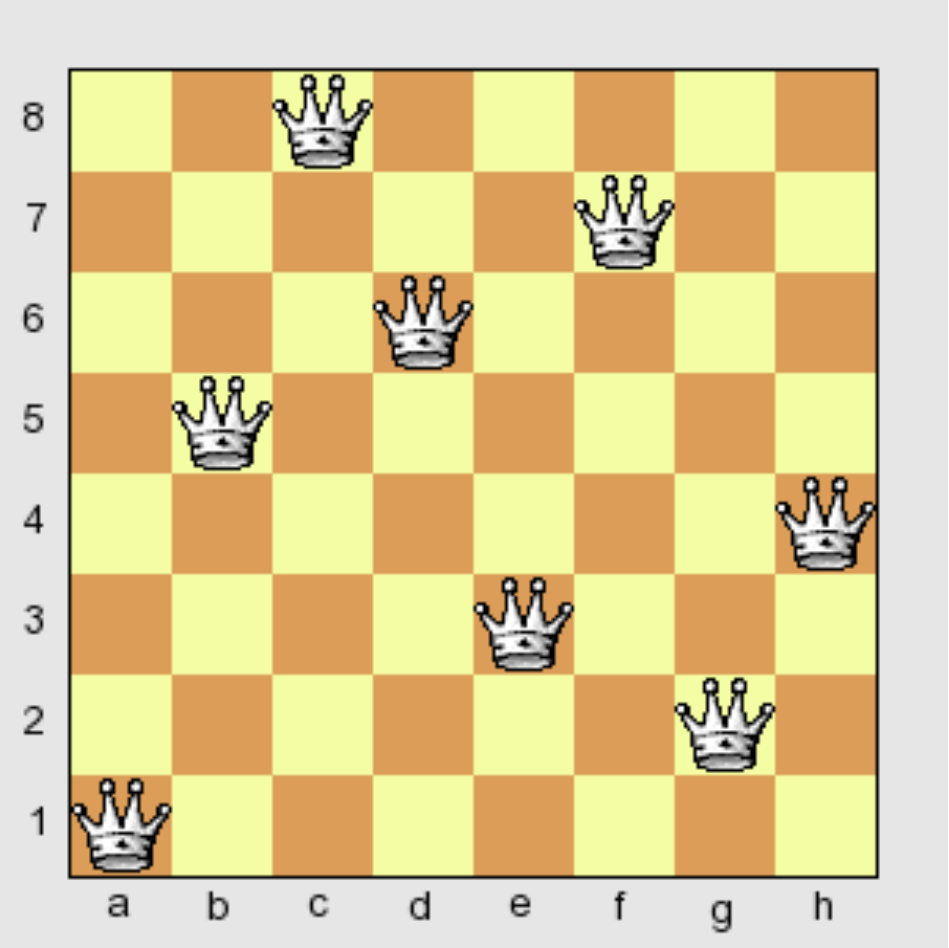
Om te tonen wanneer ***u*** waar is, zoeken we alle toekenningen die **a,b,c,…,s** en **t** waar maken

Automatische bewijsvoering

# Schaken

8 koninginnen op een 8x8 schaakbord

http://en.wikipedia.org/wiki/Eight_queens_puzzle




8 koninginnen die elkaar niet bedreigen

==

Enige pion op horizontal, verticale en de diagonale rijen

http://www.javaist.com/blog/2008/11/06/eight-queens-problem-in-proglog/

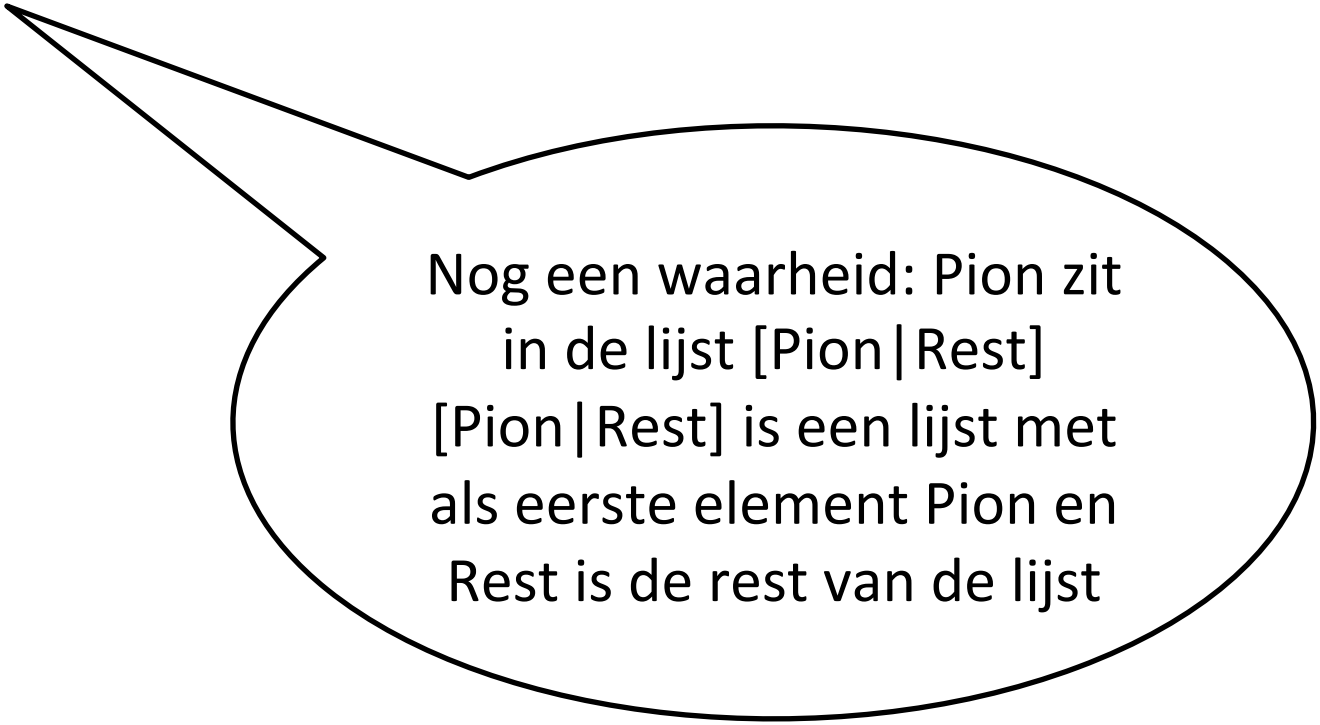
```
oplossing ([]).
```

http://www.javaist.com/blog/2008/11/06/eight-queens-problem-in-proglog/

```
oplossing ([]).
```

```
oplossing ([]).

member(Pion,[Pion|Rest]).
```

```
oplossing ([]).

member(Pion,[Pion|Rest]).
```



```
oplossing ([]).


member(Pion,[Pion|Rest]).
member(Pion,[AnderePion|Rest]) :-
   member(Pion, Rest).
```


```
sing ([]).

member(Pion,[Pion|Rest]).
member(Pion,[AnderePion|Rest]) :-
   member(Pion, Rest).
```

member.pl

```
member(Pion,[Pion|Rest]).
member(Pion,[AnderePion|Rest]) :-
   member(Pion, Rest).
```

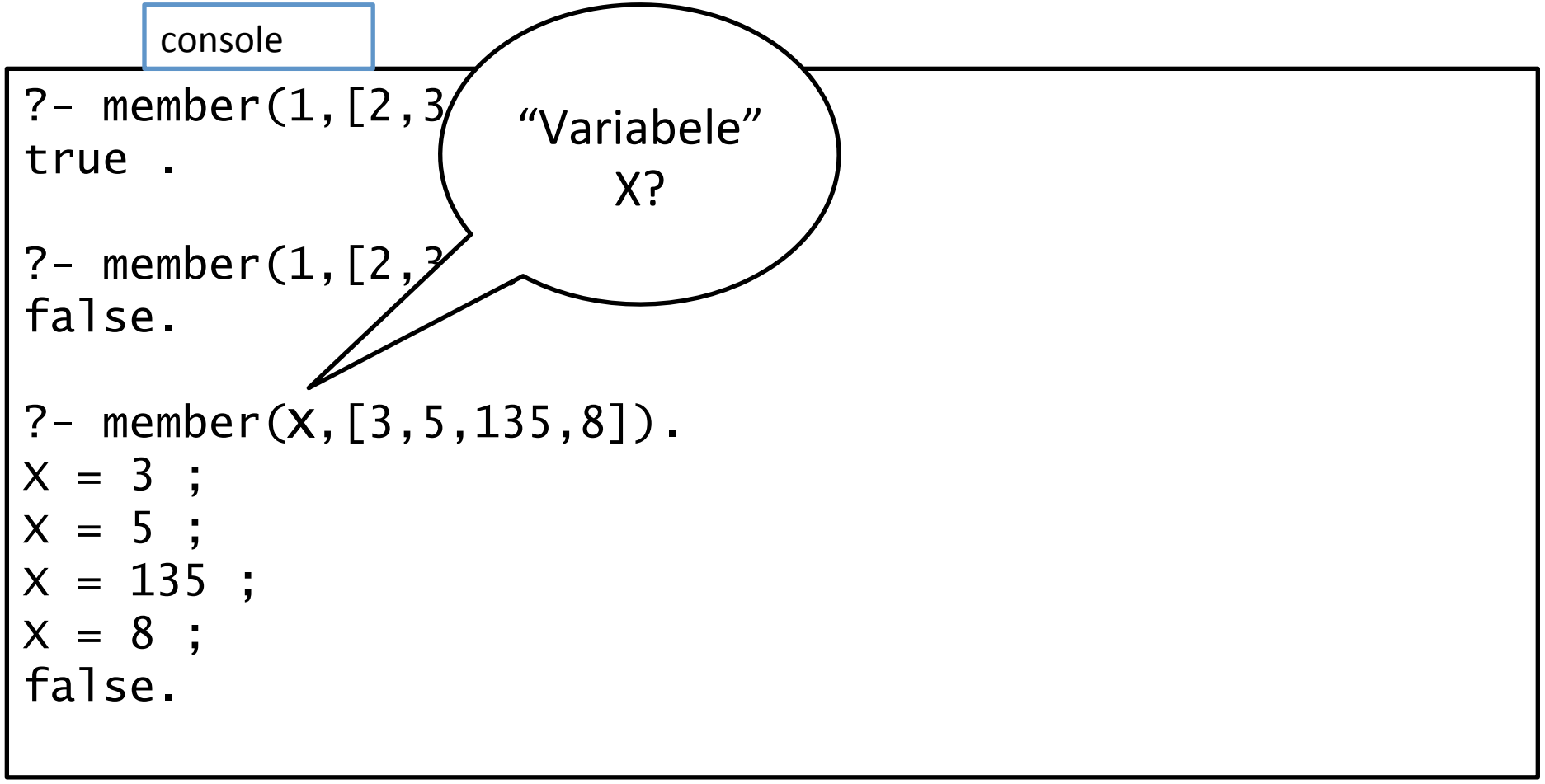
Installeer SWI-Prolog (www.swi-prolog.org)

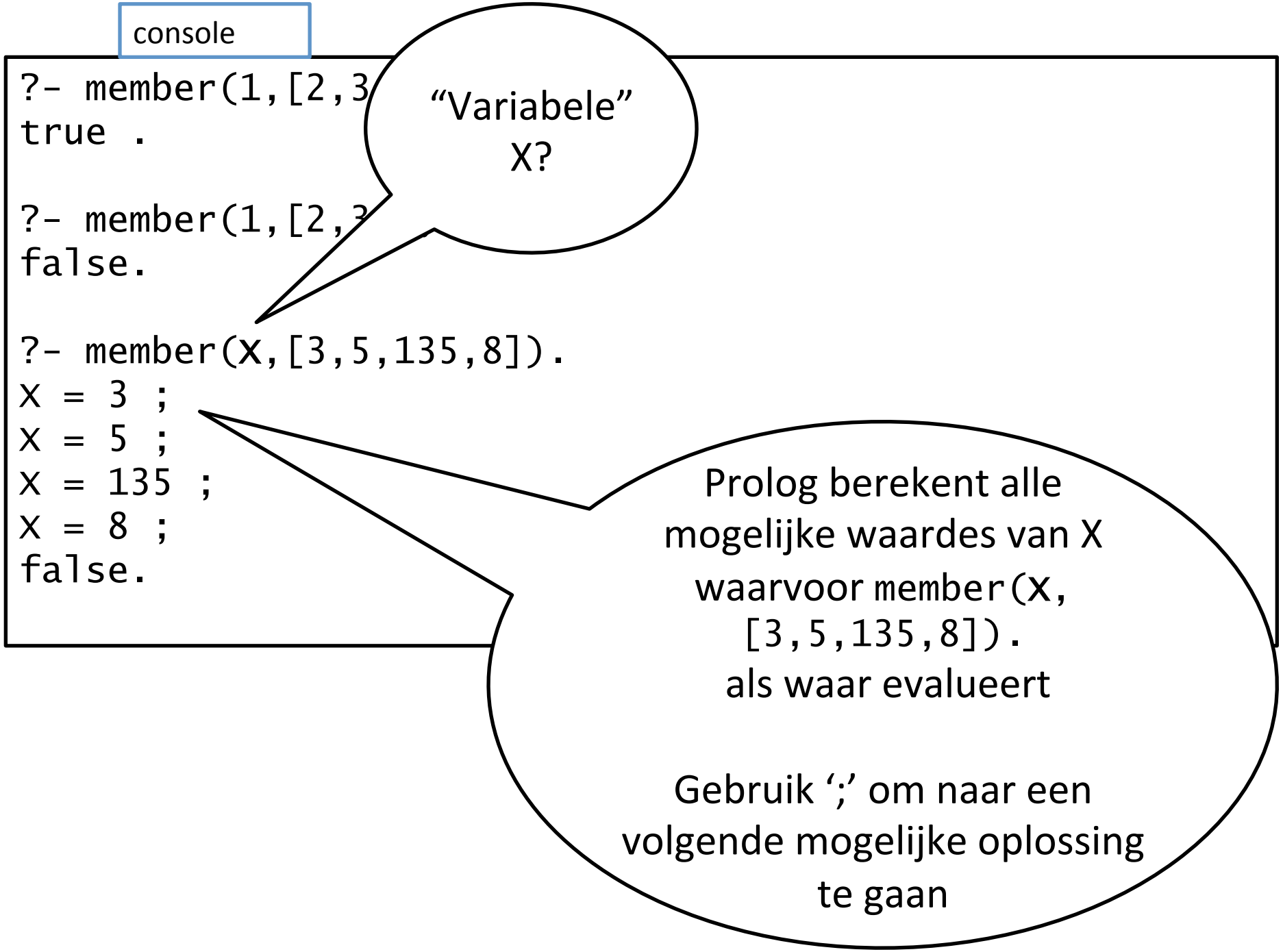
console

```
prompt$ swipl
….

For help, use ?- help(Topic). or ?- apropos(Word).

?- [member].
```

console

```
prompt$ swipl
….

For help, use ?- help(Topic). or ?- apropos(Word).

?- [member].
% member compiled 0.00 sec, 1,024 bytes
true.

?- member(1,[2,3,4,5,6,1,7]).
true .

?- member(1,[2,3,4,5,6,7]).
false.

?- member(1,[1]).
true .

?- member(1,[]).
false.
```


console
?- member(1,[2,3
true .
?- member(1,[2,3
false.
?- member(X,[3,5,135,8]).
X = 3 ;
X = 5 ;
X = 135 ;
X = 8 ;
false.
“Variabele”
X?


console
?- member(1,[2,3
true .
?- member(1,[2,3
false.
?- member(X,[3,5,135,8]).
X = 3 ;
X = 5 ;
X = 135 ;
X = 8 ;
false.
“Variabele”
X?
Prolog berekent alle mogelijke waardes van X waarvoor member(X, [3,5,135,8]). als waar evalueert
Gebruik ‘;’ om naar een volgende mogelijke oplossing te gaan

console

```
?- member(1,[2,3,4,5,6,1,7]).
true .

?- member(1,[2,3,4,5,6,7]).
false.

?- member(X,[3,5,135,8]).
X = 3 ;
X = 5 ;
X = 135 ;
X = 8 ;
false.
```

Voor alle andere mogelijkheden, evalueert Prolog de statement als `false`

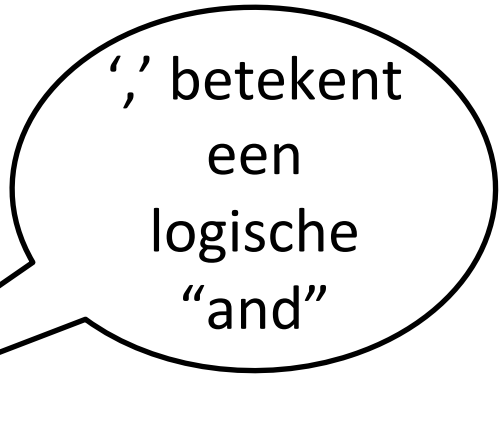
console

```
?- member(1,[2,3,4,5,6,1,7]).
true .

?- member(1,[2,3,4,5,6,7]).
false.

?- member(X,[3,5,135,8]).
X = 3 ;
X = 5 ;
X = 135 ;
X = 8 ;
false.
```

*"Once you have eliminated the impossible, whatever remains, however improbable, must be the truth."*

```
oplossing ([]).

oplossing([X/Y|Rest]) :-
  oplossing(Rest),
  member(Y, [1,2,3,4,5,6,7,8]),
  veilig(X/Y, Rest)

member(Pion,[Pion|Rest]).
member(Pion,[AnderePion|Rest]) :-
  member(Pion, Rest).
```

```
oplossing ([]).

oplossing([X/Y|Rest])
  oplossing(Rest),
  member(Y, [1,2,3,4,5,6,7,8]),
  veilig(X/Y, Rest).

member(Pion,[Pion|Rest]).
member(Pion,[AnderePion|Rest]) :-
  member(Pion, Rest).
```

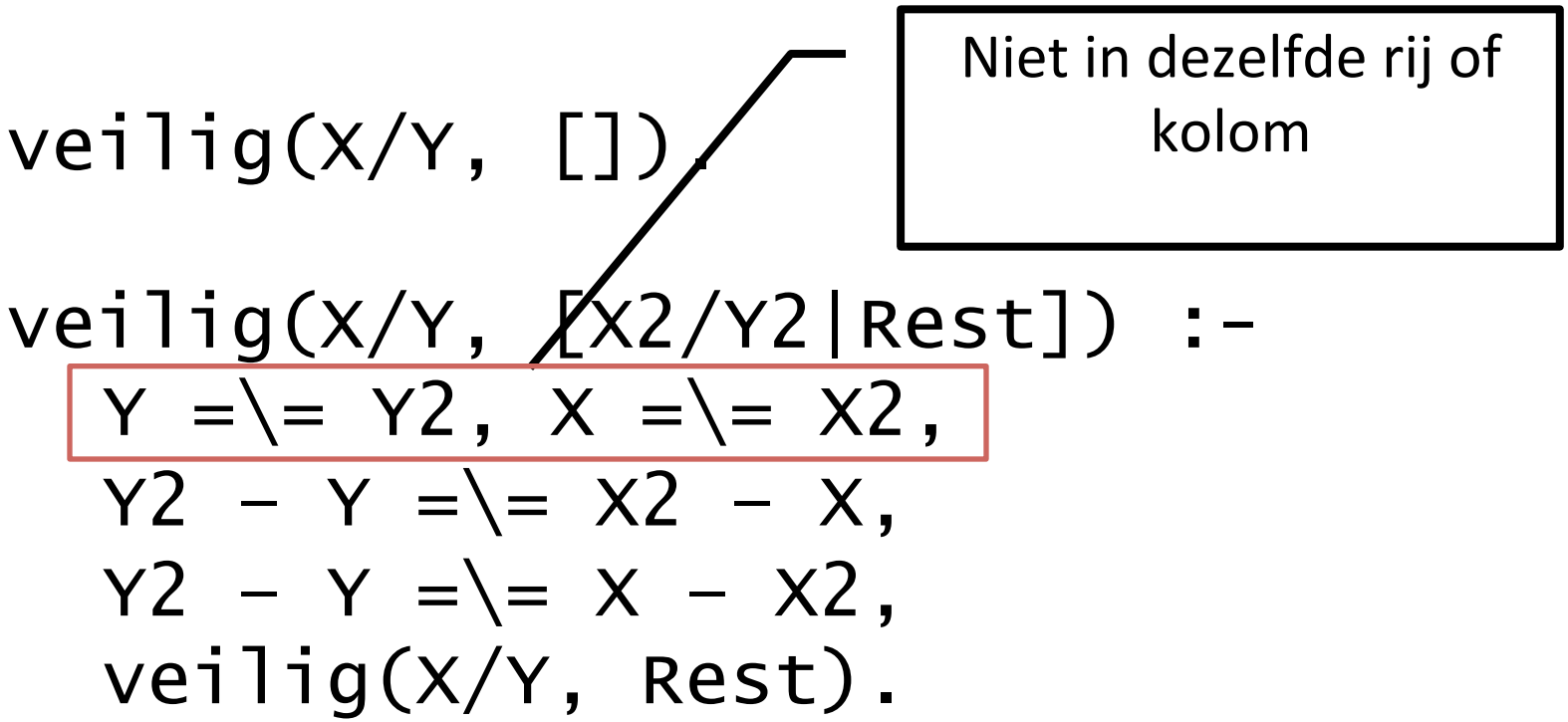


```
oplossing ([]).

oplossing([X/Y|Rest]) :-
   oplossing(Rest),
   member(Y, [1,2,3,4,5,6,7,8]),
   veilig(X/Y, Rest).

member(Pion,[
member(Pion,
   member(Pi
```

Dus opdat
“oplossing([X/Y|Rest])“ waar is moet
“oplossing(Rest)“ waar zijn,
“member(Y.[1,…,8])” waar zijn en
“veilig(X/Y, Rest)” waar zijn

```
oplossing ([]).
oplossing([X/Y|Rest]) :-
  oplossing(Rest),
  member(Y, [1,2,3,4,5,6,7,8]),
  veilig(X/Y, Rest).

member(Pion,[Pion|Rest]).
member(Pion,[AnderePion|Rest]) :-
  member(Pion, Rest).

veilig(X/Y, []).
```

```
veilig(X/Y, []).

veilig(X/Y, [X2/Y2|Rest]) :-
   Y =\= Y2, X =\= X2,
   Y2 - Y =\= X2 - X,
   Y2 - Y =\= X - X2,
   veilig(X/Y, Rest).
```


Niet in dezelfde rij of kolom
veilig(X/Y, []).
veilig(X/Y, [X2/Y2|Rest]) :-
Y =\= Y2, X =\= X2,
Y2 - Y =\= X2 - X,
Y2 - Y =\= X - X2,
veilig(X/Y, Rest).

```
veilig(X/Y, []).

veilig(X/Y, [X2/Y2|R
   Y =\= Y2, X =\= X2,
   Y2 - Y =\= X2 - X,
   Y2 - Y =\= X - X2,
   veilig(X/Y, Rest).
```

Niet in dezelfde diagonale rijen
(tel maar uit)

```
veilig(X/Y, []).

veilig(X/Y, [X2/Y2|Rest]) :-
   Y =\= Y2, X =\= X2,
   Y2 - Y =\= X2 - X,
   Y2 - Y =\= X - X2,
   veilig(X/Y, Rest).
```

Check recursief met de andere elementen in Rest

queens.pl

```
oplossing ([]).
oplossing([X/Y|Rest]) :-
   oplossing(Rest),
   member(Y, [1,2,3,4,5,6,7,8]),
   veilig(X/Y, Rest).

veilig(X/Y, []).
veilig(X/Y, [X2/Y2|Rest]) :-
   Y =\= Y2, X =\= X2,
   Y2 - Y =\= X2 - X,
   Y2 - Y =\= X - X2,
   veilig(X/Y, Rest).


member(Pion,[Pion|Rest]).
member(Pion,[AnderePion|Rest]) :-
   member(Pion, Rest).
```

queens.pl

```
oplossing ([]).
oplossing([X/Y|Rest]) :-
   oplossing(Rest),
   member(Y, [1,2,3,4,5,6,7,8]),
   veilig(X/Y, Rest).

veilig(_, []).
veilig(X/Y, [X2/Y2|Rest]) :-
   Y =\= Y2, X =\= X2,
   Y2 - Y =\= X2 - X,
   Y2 - Y =\= X - X2,
   veilig(X/Y, Rest).


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

queens.pl

```
oplossing ([]).
oplossing([X/Y|Rest]) :-
   oplossing(Rest),
   member(Y, [1,2,3,4,5,6,7,8]),
   veilig(X/Y, Rest).

veilig(_, []).
veilig(X1/Y1, [X2/Y2|Rest]) :-
   Y1 =\= Y2,
   Y2 - Y1 =\= X2 - X1,
   Y2 - Y1 =\= X1 - X2,
   veilig(X1/Y1, Rest).


member(Pion,[Pion|_]).
member(Pion,[AnderePion|Rest]) :-
   member(Pion, Rest).

template([1/Y1,2/Y2,3/Y3,4/Y4,5/Y5,6/Y6,7/Y7,8/Y8]).
```

console

```
prompt$ swipl
….

For help, use ?- help(Topic). or ?- apropos(Word).

?- [queens].
% queens compiled 0.00 sec, 4,208 bytes
true.

?- template(S),oplossing(S).
S = [1/4, 2/2, 3/7, 4/3, 5/6, 6/8, 7/5, 8/1] ;
S = [1/5, 2/2, 3/4, 4/7, 5/3, 6/8, 7/6, 8/1] ;
S = [1/3, 2/5, 3/2, 4/8, 5/6, 6/4, 7/7, 8/1] ;
S = [1/3, 2/6, 3/4, 4/2, 5/8, 6/5, 7/7, 8/1] ;
S = [1/5, 2/7, 3/1, 4/3, 5/8, 6/6, 7/4, 8/2] ;
S = [1/4, 2/6, 3/8, 4/3, 5/1, 6/7, 7/5, 8/2] ;
S = [1/3, 2/6, 3/8, 4/1, 5/4, 6/7, 7/5, 8/2] ;
S = [1/5, 2/3, 3/8, 4/4, 5/7, 6/1, 7/6, 8/2]
```

# *“Insufficient facts always invite danger”*

Star Trek Original Series

# Terminology

# Atoms en variabelen

Atoms beginnen met een kleine letter.

pion, driewerf, p2001

Variabelen beginnen met een grote letter.

X, Pion, Dt

# Facts

naam(kris).
zoonvan(brent,kris).
zoonvan(jarne,kris)
woonplaats(borgloon).
werk(uhasselt).
vervoer(uhasselt,auto).
vervoer(bakker,fiets).
vervoer(opvang,tevoet).

# Query

?- zoonvan(X,kris).
X=brent ;
X=jarne;
false.

# Rule

head :- body

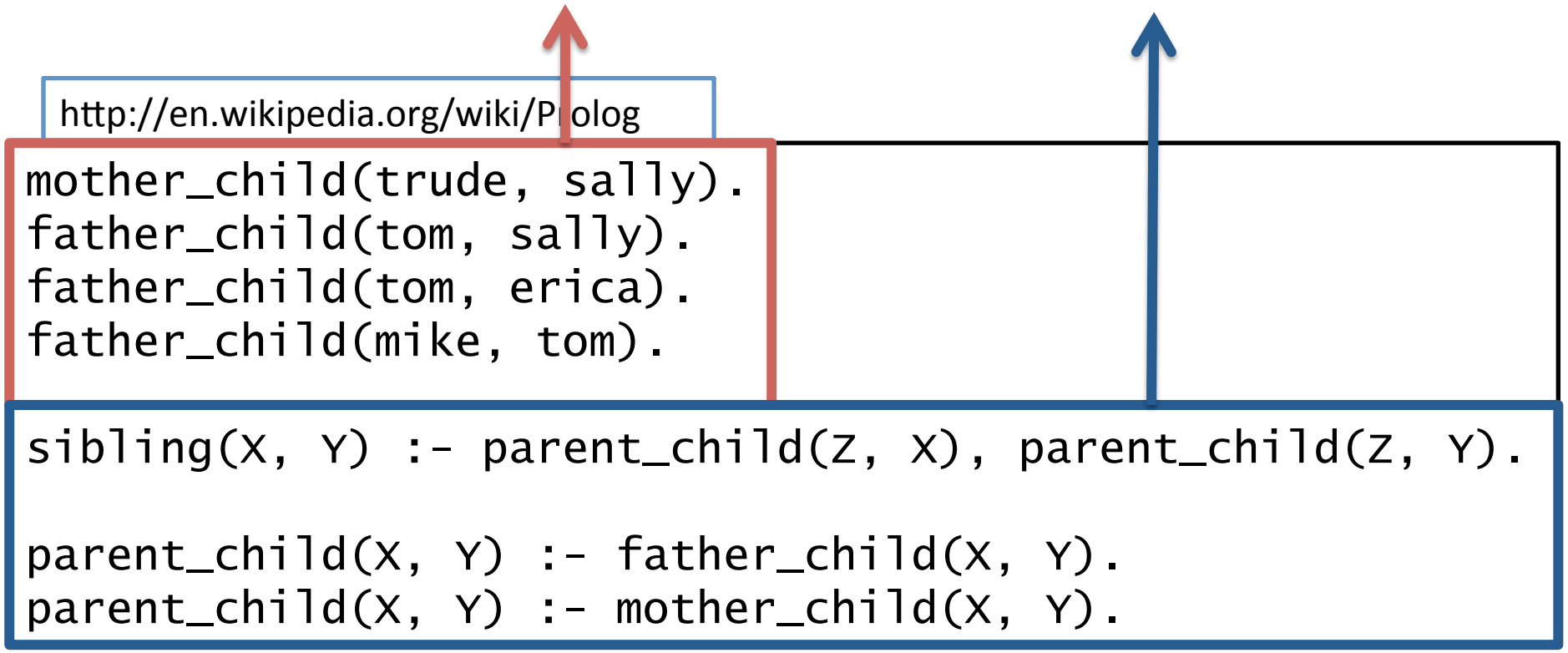
kleinzoon(X,Y) :-
zoonvan(X,Z),
zoonvan(Z,Y).

# Kennisdatabank + Regels

http://en.wikipedia.org/wiki/Prolog

```
mother_child(trude, sally).
father_child(tom, sally).
father_child(tom, erica).
father_child(mike, tom).

sibling(X, Y) :- parent_child(Z, X), parent_child(Z, Y).

parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).
```

# Kennisdatabank + Regels



# Kennisdatabank + Regels

http://en.wikipedia.org/wiki/Prolog

```
mother_child(trude, sally).
father_child(tom, sally).
father_child(tom, erica).
father_child(mike, tom).

sibling(X, Y) :- parent_child(Z, X), parent_child(Z, Y).

parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).
```

De **goals** van deze regel.

# Regels combineren

http://en.wikipedia.org/wiki/Prolog

```
mother_child(trude, sally).
father_child(tom, sally).
father_child(tom, erica).
father_child(mike, tom).

sibling(X, Y) :- parent_child(Z, X), parent_child(Z, Y).

parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).
```

Met logisch en ‘,’

Zolang goal (query) niet true evalueert, probeer de volgende clause

# Regels combineren

http://en.wikipedia.org/wiki/Prolog

```
mother_child(trude, sally).
father_child(tom, sally).
father_child(tom, erica).
father_child(mike, tom).

sibling(X, Y) :- parent_child(Z, X), parent_child(Z, Y).

parent_child(X, Y) :- father_child(X, Y) ;
                      mother_child(X, Y).
```

Met logisch en ‘,’

Met logische of ‘;’

# Lijsten

[1,2,3,6,1314,29]

[hond,kat,giraf,walvis]

[Head|Tail]

[1,2,3,6,1314,29]

[1,2,3,6|Tail] met Tail = [1314,29]

[1,2,3|Tail] met Tail = [6,1314,29]

[1,2|Tail] met Tail = [3,6,1314,29]

[1|Tail] met Tail = [2,3,6,1314,29]

# Append

```
%  append/3   “functie definitie”
%  append met 3 parameters

append([],L,L).
append(L,[],L).
append([Head|Tail],L,[Head|Result]) :-
     append(Tail,L,Result).
```

```
quick_sort([],[]).
quick_sort([S|T],Sorted):-
  splitL(S,T,L1,L2),
  quick_sort(L1,List1),
  quick_sort(L2,List2),
  append(List1,[S|List2],Sorted).

splitL(H,[],[],[]).
splitL(H,[X|T],[X|L],G):-
  X=<H,splitL(H,T,L,G).
splitL(H,[X|T],L,[X|G]):-
  X>H,splitL(H,T,L,G).
```

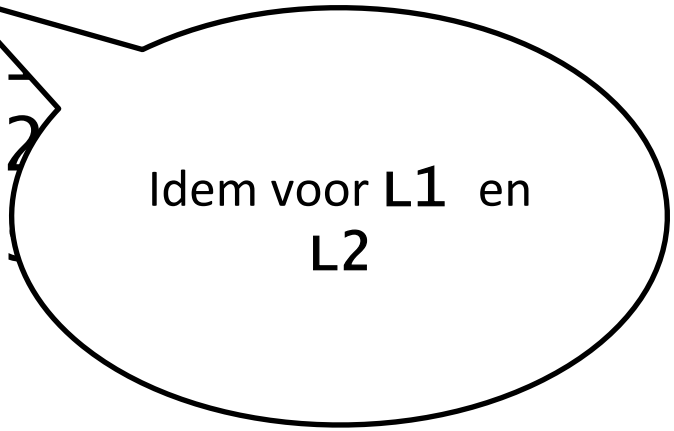
Let op, dit werkt omgekeerd dan wat je gewoon bent!
Prolog tracht variabelen te binden met waarden zodat alle goals waar zijn.
Deze quicksort implementatie zoekt dus naar een lijst waarvoor alle condities waar zijn.

```
quick_sort([],[]).
quick_sort([S|T],Sorte   :-
   splitL(S,T,L1,L2),
   quick_sort(L1,List1),
   quick_sort(L2,List2),
   append(List1,[S|List2],Sorted).

splitL(H,[],[],[]).
splitL(H,[X|T],[X|L],G):-
   X=<H,splitL(H,T,L,G).
splitL(H,[X|T],L,[X|G]):-
   X>H,splitL(H,T,L,G).
```

```
quick_sort([],[]).
quick_sort([S|T],Sorted):-
  splitL(S,T,L1,L2),
  quick_sort(L1,List1),
  quick_sort(L2,List2),
  append(List1,[S|List2],Sorted).

splitL(H,[],[],[]).
splitL(H,[X|T],[X|L],G):-
  X=<H,splitL(H,T,L,G).
splitL(H,[X|T],L,[X|G]):-
  X>H,splitL(H,T,L,G).
```

**Sorted** is de variabele waar de resulterende lijst aan gebonden zal worden

```
quick_sort([],[]).
quick_sort([S|T],Sorted):-
   splitL(S,T,L1,L2),
   quick_sort(L1,List1),
   quick_sort(L2,List2),
   append(List1,[S|List2],Sorted).

splitL(H,[],[],[]).
splitL(H,[X|T],[X|L],G):-
   X=<H,splitL(H,T,L,G).
splitL(H,[X|T],L,[X|G]):-
   X>H,splitL(H,T,L,G).
```



```
quick_sort([],[]).
quick_sort([S|T],Sorted):-
  splitL(S,T,L1,L2),
  quick_sort(L1,List1),
  quick_sort(L2,List2),
  append(List1,[S|List2],Sorted).

splitL(H,[],[],[]).
splitL(H,[X|T],[X|L],G):-
  X=<H,splitL(H,T,L,G).
splitL(H,[X|T],L,[X|G]):-
  X>H,splitL(H,T,L,G).
```

Als X kleiner of gelijk aan de spil is, zit deze in de eerste lijst

```
quick_sort([],[]).
quick_sort([S|T],Sorted):-
  splitL(S,T,L1,L2),
  quick_sort(L1,List1),
  quick_sort(L2,List2),
  append(List1,[S|List
```

```
splitL(H,[],[],[]).
splitL(H,[X|T],[X|L],G)
  X=<H,splitL(H,T,L,G).
splitL(H,[X|T],L,[X|G]):-
  X>H,splitL(H,T,L,G).
```

```
quick_sort([],[]).
quick_sort([S|T],Sorted):-
  splitL(S,T,L1,L2),
  quick_sort(L1,List1),
  quick_sort(L2,List2),
  append(List1,[S|List2],Sorted).

splitL(H,[],[],[]).
splitL(H,[X|T],[X|L],G):-
  X=<H,splitL(H,T,L,G).
splitL(H,[X|T],L,[X|G]):-
  X>H,splitL(H,T,L,G).
```

console

```
?- [quicksort].
% quicksort compiled 0.00 sec, 2,840 bytes
true.

?- quick_sort([1,3,1,1,4,28,8,9,3,2,1,0],S).
S = [0, 1, 1, 1, 1, 2, 3, 3, 4|...] ;
false.

?- quick_sort([1,4,28,8,9,3,2,1,0],S).
S = [0, 1, 1, 2, 3, 4, 8, 9, 28] ;
false.
```

# Oefeningen (let op de formulering)

- Je kan de (wiskundige) operatoren die je nodig hebt opzoeken in het online boek http://www.learnprolognow.org/
  - Let op verschil tussen “=“ en “is”
- Schrijf een predicaat lastElement/2 die gegeven een lijst en het laatste element van die lijst als waar evalueert.

```
?- lastElement([1,2,3,4,100],Last).
Last = 100
```

- Schrijf een predicaat vervang/4 die als waar evalueert als al de elementen X in Lijst1 vervangen zijn door Y in Lijst2.

```
?- vervang(3,4,[3,4,5,4,3,5,3,6],Lijst2).
Lijst2 = [4,4,5,4,4,5,4,6]
```

# Oefeningen (let op de formulering)

- Schrijf een predicaat pak_in/2 die als waar evalueert als een lijst de opeenvolgende identieke waardes van een andere lijst inpakt in sublijsten.

```
?- pak_in([1,1,1,2,2,8,8,8,8,8,8],L).
L = [[1,1,1],[2,2],[8,8,8,8,8,8]]
```

- Schrijf een predicaat rlencode/2 die een run length encoding doet van een lijst. (gebruik of implementeer het predicaat length/2 die de lengte van een lijst teruggeeft en gebruik pak_in/2)

```
?- rlEncode([1,1,1,2,2,8,8,8,8,8,8],CodedList).
CodedList = [[3,1],[2,2],[6,8]]
```

# Nog wat oefenen?

http://sites.google.com/site/prologsite/prolog-problems/1

http://sites.google.com/site/prologsite/prolog-problems/2