# Programming Paradigms

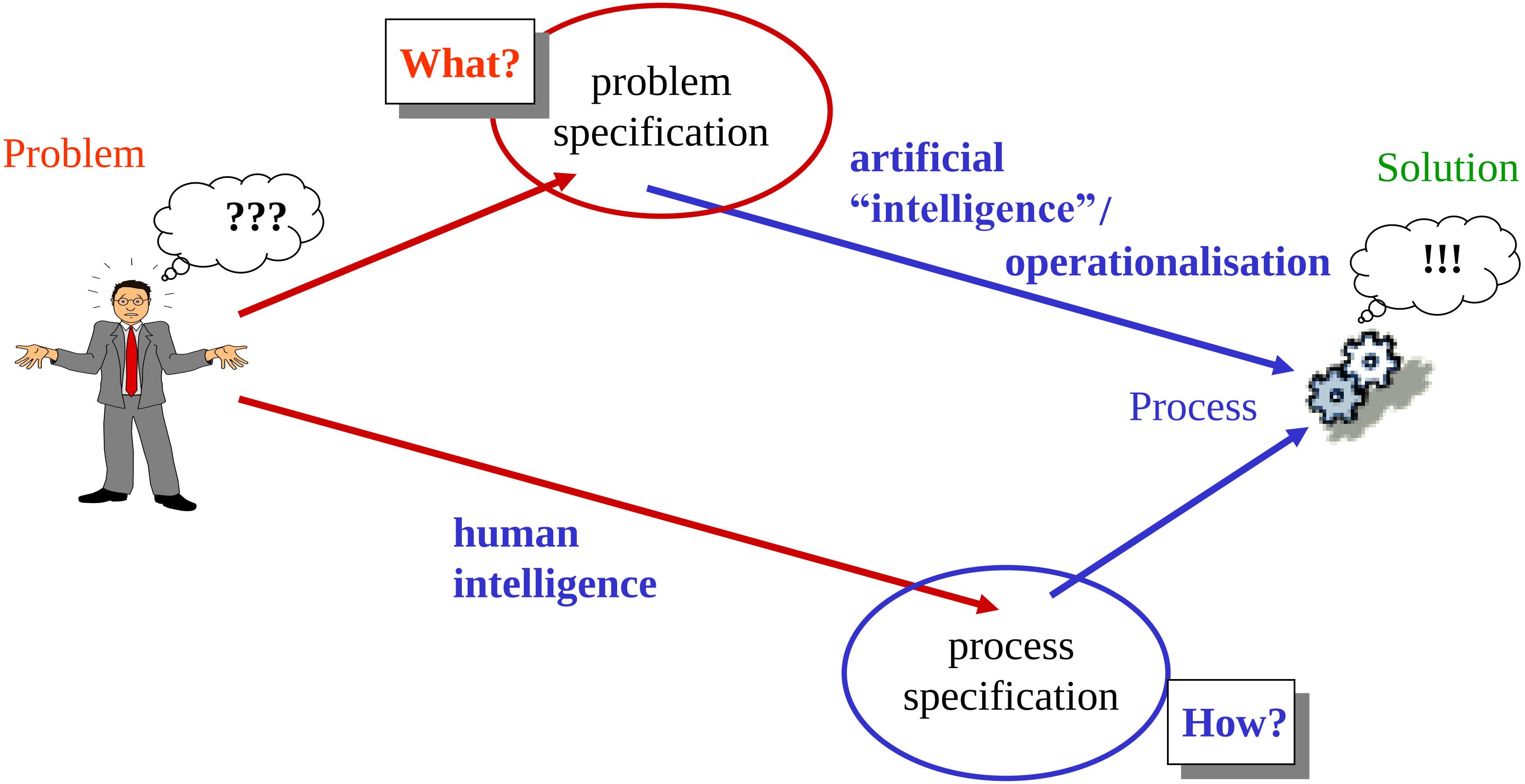
## Summer Term 2017

10th Lecture

**Prof. Janis Voigtländer**
**University of Duisburg-Essen**

# Recall: Ideal (and to some extent, history) of declarative programming

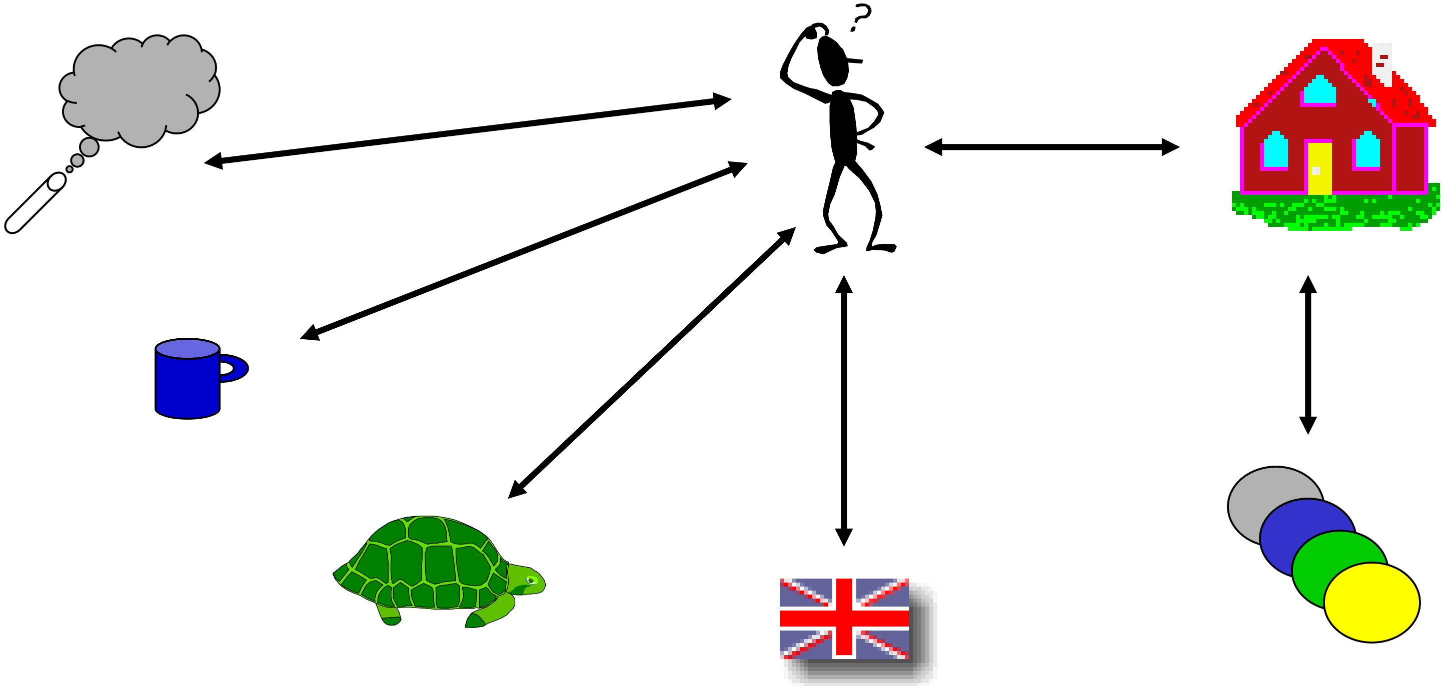
Freeing the programmer from the necessity to explicitly plan and specify the computation process that leads to a problem solution: **“What instead of How”**



# A famous logical puzzle as a declaratively specified problem

"There are five houses, each of a different color and inhabitated by a man of a different nationality with a different pet, drink and brand of smokes …"

("Einstein's Riddle", see http://en.wikipedia.org/wiki/Zebra_Puzzle)



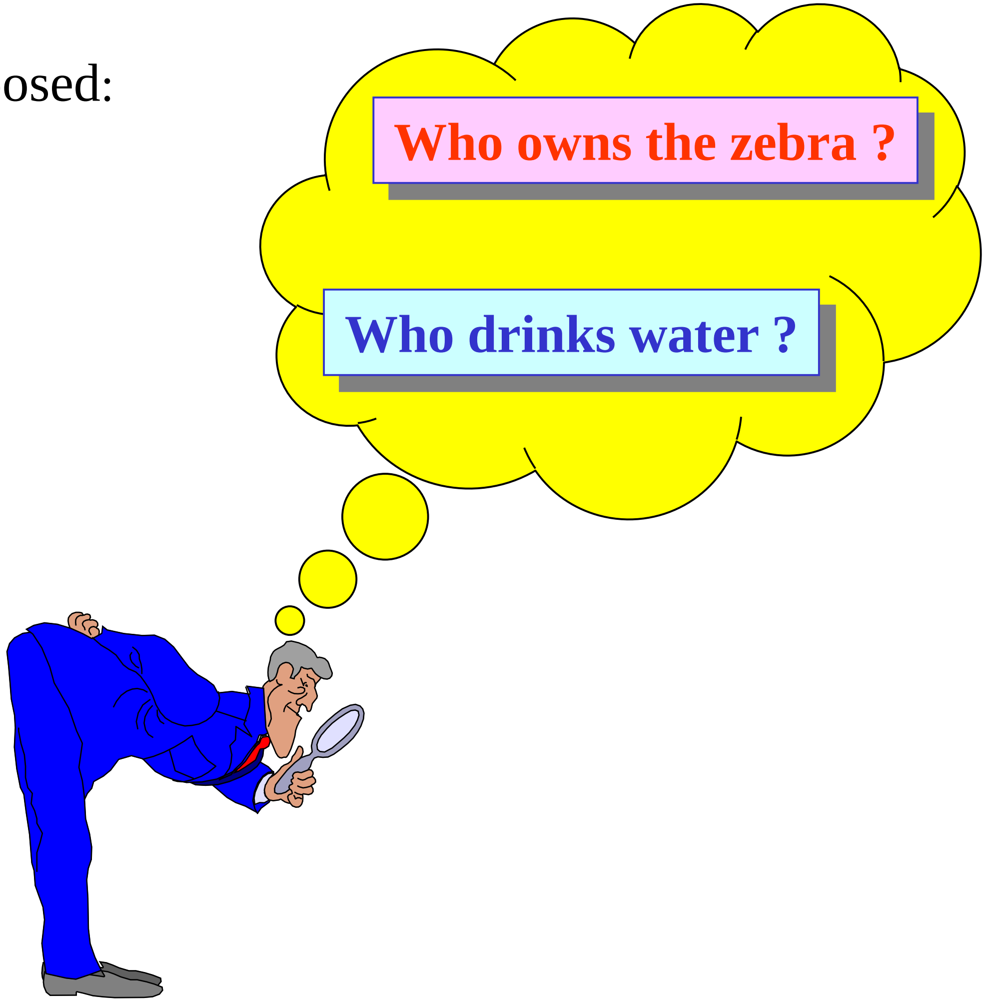
# Puzzle (1)

Overall there are 14 clues that define the “world” of the puzzle:

1. The Englishman lives in the red house.
2. The Spaniard owns the dog.
3. Coffee is drunk in the green house.
4. The Ukrainian drinks tea.
5. The green house is immediately to the right of the ivory house.
6. The Winston smoker owns snails.
7. Kools are smoked in the yellow house.
8. Milk is drunk in the middle house.
9. The Norwegian lives in the leftmost house.
10. The man who smokes Chesterfield lives in the house next to the man with the fox.
11. Kools are smoked in the house next to the house where the horse is kept.
12. The Lucky Strike smoker drinks orange juice.
13. The Japanese smokes Parliaments.
14. The Norwegian lives next to the blue house.

# Puzzle (2)

The problem posed:

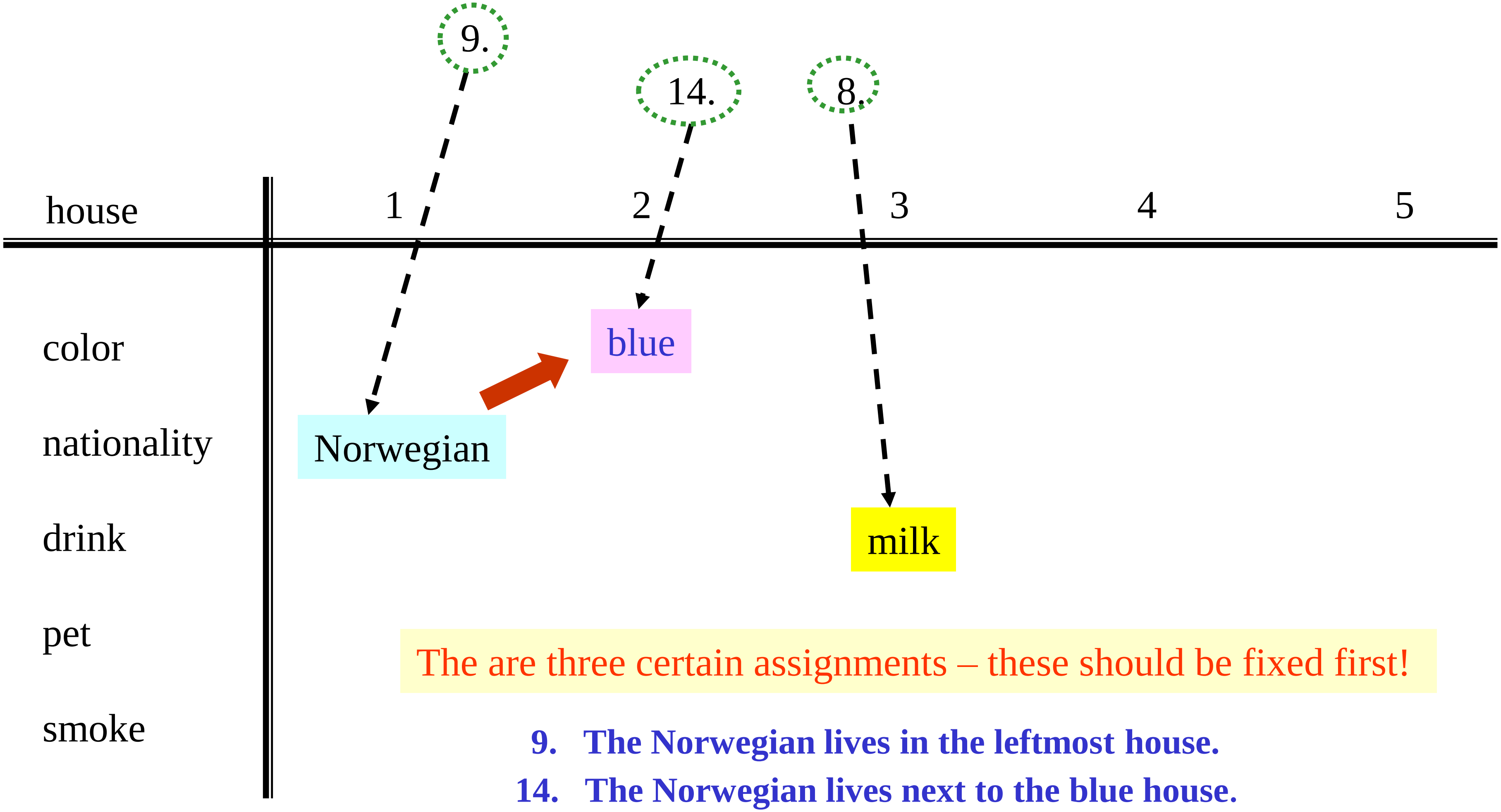


**Who owns the zebra ?**

**Who drinks water ?**

# Puzzle (3)

Systematic construction of the solution (by a human):



| house | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| color | | blue | | | |
| nationality | Norwegian | | | | |
| drink | | | milk | | |
| pet | | | | | |
| smoke | | | | | |

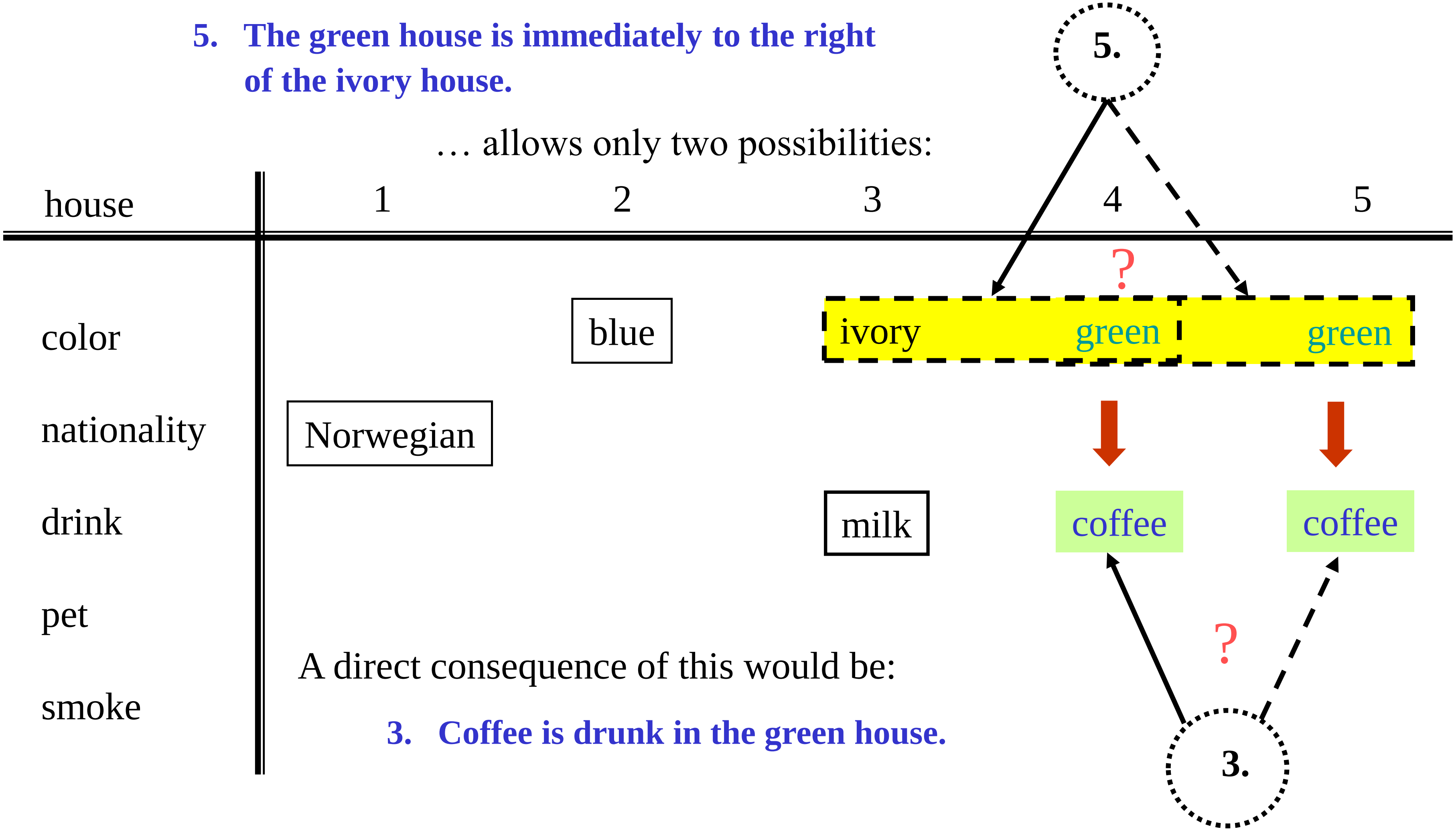
The are three certain assignments – these should be fixed first!

**9. The Norwegian lives in the leftmost house.**
**14. The Norwegian lives next to the blue house.**
**8. Milk is drunk in the middle house.**

# Puzzle (4)

Condition 5:

5. **The green house is immediately to the right of the ivory house.**



… allows only two possibilities:

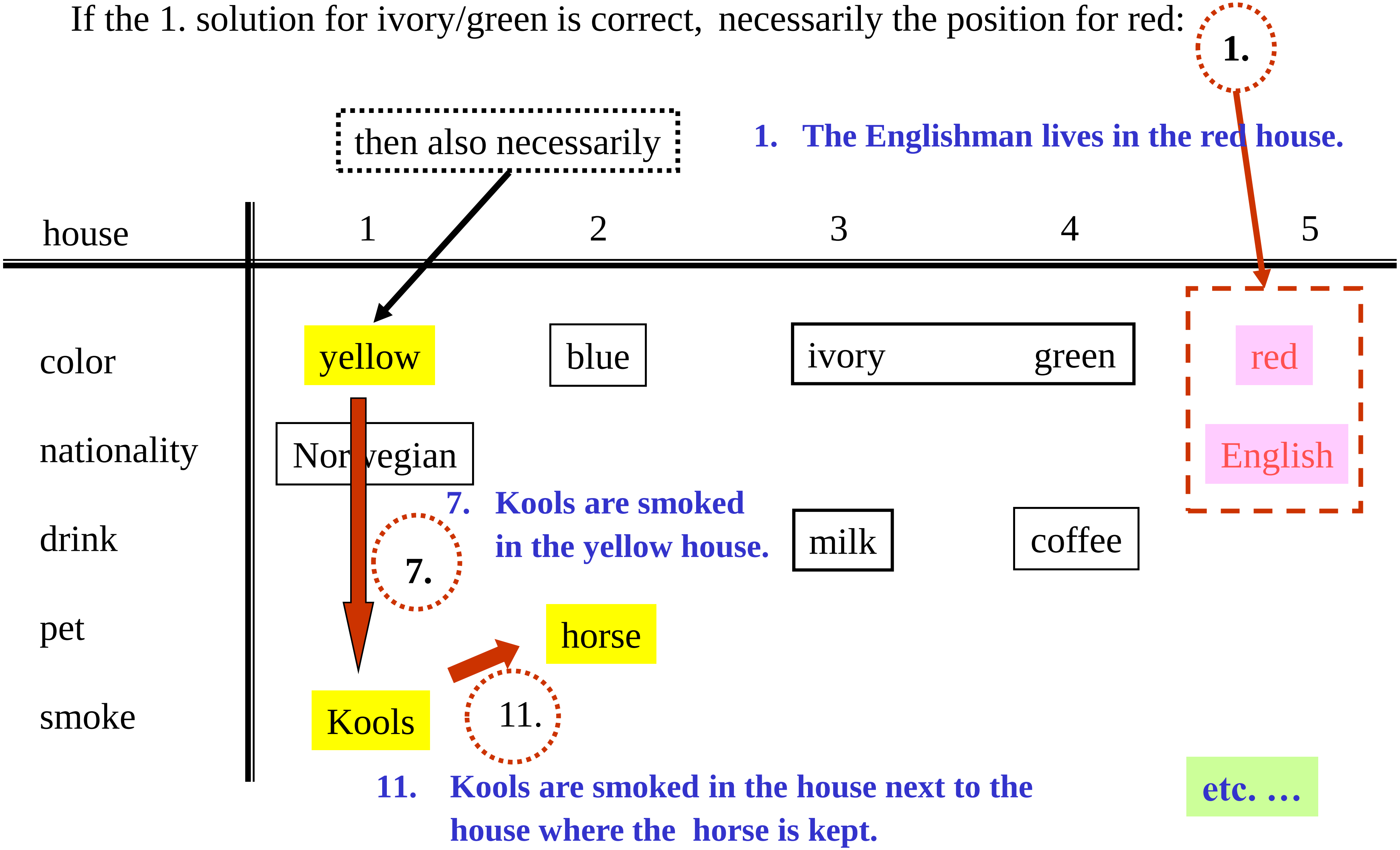
| house | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| color | | blue | ivory | green | green |
| nationality | Norwegian | | | | |
| drink | | | milk | coffee | coffee |
| pet | | | | | |
| smoke | | | | | |

A direct consequence of this would be:

3. **Coffee is drunk in the green house.**

# Puzzle (5)

If the 1. solution for ivory/green is correct, necessarily the position for red:



then also necessarily

**1. The Englishman lives in the red house.**

| house | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| color | yellow | blue | ivory | green | red |
| nationality | Norwegian | | | | English |
| drink | | | milk | coffee | |
| pet | | horse | | | |
| smoke | Kools | | | | |

**7. Kools are smoked in the yellow house.**

**11. Kools are smoked in the house next to the house where the horse is kept.**

**etc. …**

# Puzzle (6)

Unique solution of the puzzle (to be found via several backtracking steps):

| house | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| color | yellow | blue | red | ivory | green |
| nationality | Norwegian | Ukrainian | English | Spanish | Japanese |
| drink | **water** | tea | milk | juice | coffee |
| pet | fox | horse | snails | dog | **zebra** |
| smoke | Kools | Chesterfield | Winston | Lucky Strike | Parliaments |

# Algorithm = Logic + Control

> *„An algorithm can be regarded as consisting of a logic component, which specifies the knowledge to be used in solving problems, and a control component, which determines the problem-solving strategies by means of which that knowledge is used. The logic component determines the meaning of the algorithm whereas the control component only affects its efficiency.*
>
> *The efficiency of an algorithm can often be improved by improving the control component without changing the logic of the algorithm. We argue that computer programs would be more often correct and more easily improved and modified if their logic and control aspects were identified and separated in the program text.“*
>
> Robert Kowalski, 1979

# Puzzle: one possible specification in Prolog

```
right_of(R, L, [ L | [ R | _ ] ]).
right_of(R, L, [ _ | Rest ]) :- right_of(R, L, Rest).
next_to(X, Y, List) :- right_of(X, Y, List).
next_to(X, Y, List) :- right_of(Y, X, List).
zebra(Zebra_Owner) :-
  8. ∧ 9.   Houses = [ [ _, norwegian, _, _, _ ], _, [ _, _, milk, _, _ ], _, _ ],
     1.     member([ red, englishman, _, _, _ ], Houses),
     2.     member([ _, spaniard, _, dog, _ ], Houses),
     3.     member([ green, _, coffee, _, _ ], Houses),
     4.     member([ _, ukrainian, tea, _, _ ], Houses),
     5.     right_of([ green, _, _, _, _ ], [ ivory, _, _, _, _ ], Houses),
     6.     member([ _, _, _, snails, winston ], Houses),
     7.     member([ yellow, _, _, _, kools ], Houses),
    10.     next_to([ _, _, _, _, chesterfield ], [ _, _, _, fox, _ ], Houses),
    11.     next_to([ _, _, _, _, kools ], [ _, _, _, horse, _ ], Houses),
    12.     member([ _, _, juice, _, lucky ], Houses),
    13.     member([ _, japanese, _, _, parliaments ], Houses),
    14.     next_to([ _, norwegian, _, _, _ ], [ blue, _, _, _, _ ], Houses),
     ?      member([ _, Zebra_Owner, _, zebra, _ ], Houses),
     ?      member([ _, _, water, _, _ ], Houses).
```

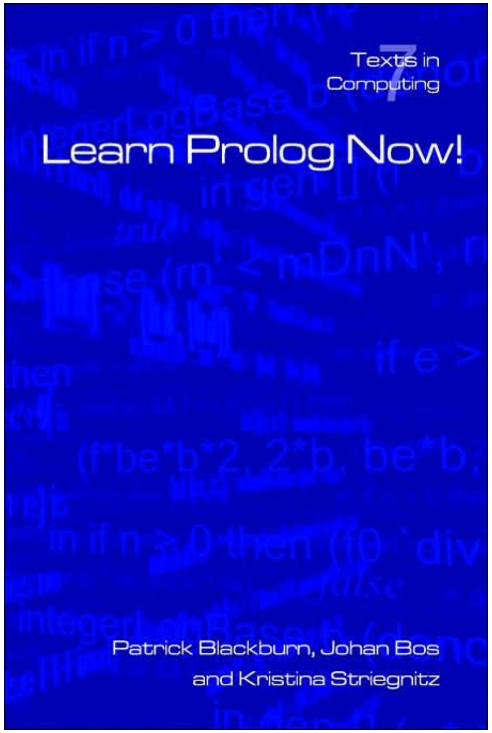
# History of Prolog

- Prolog as name is abbreviated from “Programming with logic”.
- It is the most common logic programming language.
- Some history on Prolog:

| | |
|---|---|
| 1965: | John Alan Robinson provides theoretical foundations for theorem provers using the resolution calculus. |
| 1972: | Alain Colmerauer (Marseilles) and his group develop Prolog. |
| in the '70s: | David D.H. Warren builds the first Prolog compiler. |
| 1981–92: | 5th Generation Computer Project in Japan (made Prolog “popular”) |

# Literature on Prolog

- A lot of books and tutorials exist.
- The slides use a lot of examples from this book:



Patrick Blackburn, Johan Bos,
Kristina Striegnitz:
„Learn Prolog Now!“
College Publications, 2006

# Programming Paradigms

## Prolog Basics/Syntax

# Prolog in simplest case: facts and queries

- A kind of data base with a number of facts:

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
```

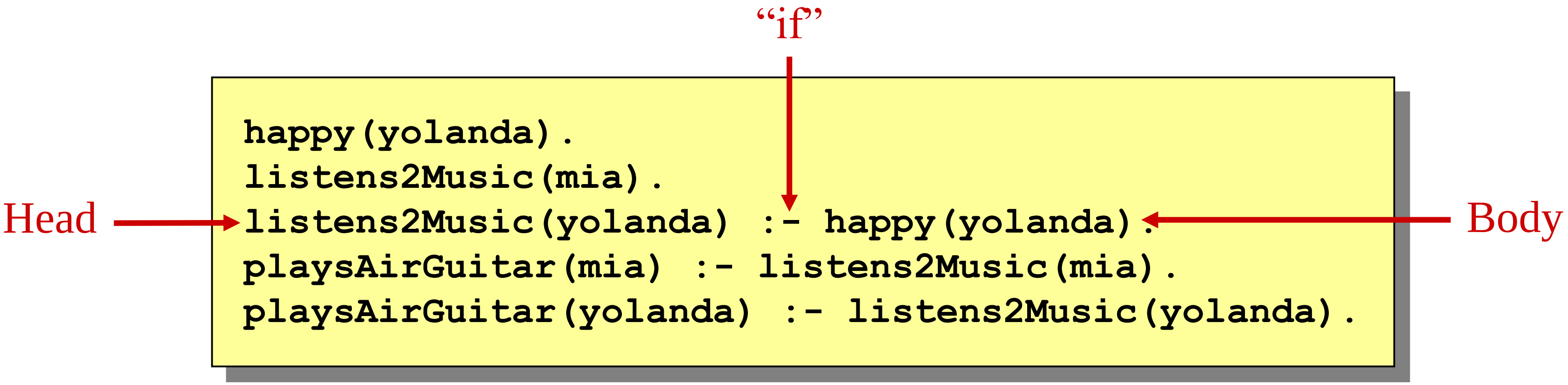
- Queries:

```
?- woman(mia).
true.

?- playsAirGuitar(jody).
true.

?- playsAirGuitar(mia).
false.

?- playsAirGuitar(vincent).
false.

?- playsPiano(jody).
false.
```

The dot is essential!

or an error message

# Facts + simple implications



“if”

Head

Body

```
happy(yolanda).
listens2Music(mia).
listens2Music(yolanda) :- happy(yolanda).
playsAirGuitar(mia) :- listens2Music(mia).
playsAirGuitar(yolanda) :- listens2Music(yolanda).
```

- Queries:

```
?- playsAirGuitar(mia).
true.

?- playsAirGuitar(yolanda).
true.
```

because of:

```
happy(yolanda)
  ⇒ listens2Music(yolanda)
  ⇒ playsAirGuitar(yolanda)
```

# More complex rules

“and”

```
happy(vincent).
listens2Music(butch).
playsAirGuitar(vincent) :- listens2Music(vincent),
                           happy(vincent).
playsAirGuitar(butch) :- happy(butch).
playsAirGuitar(butch) :- listens2Music(butch).
```

Alternatives

- Queries:

```
?- playsAirGuitar(vincent).
false.

?- playsAirGuitar(butch).
true.
```

- Alternative notation:

“or”

```
...
playsAirGuitar(butch) :- happy(butch);
                         listens2Music(butch).
```

# Relations, and more complex queries

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent,mia).
loves(marsellus,mia).
loves(mia,vincent).
loves(vincent,vincent).
```

multi-ary (concretely, binary) predicate

- Queries:

```
?- woman(X).
X = mia ;
X = jody ;
X = yolanda.

?- loves(vincent,X).
X = mia ;
X = vincent.

?- loves(vincent,X), woman(X).
X = mia ;
false.
```

semicolon entered by user

# Variables in rules (not just in queries)

```
loves(vincent,mia).
loves(marsellus,mia).
loves(mia,vincent).

jealous(X,Y) :- loves(X,Z), loves(Y,Z).
```

- Queries:

```
?- jealous(marsellus,X).
X = vincent ;
X = marsellus ;
false.

?- jealous(X,_).
X = vincent ;
X = vincent ;
X = marsellus ;
X = marsellus ;
X = mia.
```

anonymous variable

# Variables in rules (not just in queries)

```
loves(vincent,mia).
loves(marsellus,mia).
loves(mia,vincent).

jealous(X,Y) :- loves(X,Z), loves(Y,Z), X \= Y.
```

important that at end

- Queries:

```
?- jealous(marsellus,X).
X = vincent ;
false.

?- jealous(X,_).
X = vincent ;
X = marsellus ;
false.

?- jealous(X,Y).
X = vincent,
Y = marsellus ;
X = marsellus,
Y = vincent ;
false.
```

# Some observations on variables

```
loves(vincent,mia).
loves(marsellus,mia).
loves(mia,vincent).

jealous(X,Y) :- loves(X,Z), loves(Y,Z), X \= Y.
```

- Variables in rules and in queries are independent from each other.

```
?- jealous(marsellus,X).
X = vincent ;
false.
```

- Within a rule or a query, the same variables represent the same objects.
- But different variables do not necessarily represent different objects.
- It is possible to have several occurrences of the same variable in a rule's head!
- In a rule's body there can be variables that do not occur in its head!

# Intuition on “free” variables

```
loves(vincent,mia).
loves(marsellus,mia).
loves(mia,vincent).

jealous(X,Y) :- loves(X,Z), loves(Y,Z), X \= Y.
```

- What is the “logical” interpretation of **`Z`** above? (or of the whole rule?)

- Possibly, for arbitrary (but fixed) **`X`** , **`Y`**:
  if for every choice of **`Z`** holds: **`loves(X,Z)`**, and **`loves(Y,Z)`**, and **`X \= Y`**,
  then also holds: **`jealous(X,Y)`**

- Or, for arbitrary (but fixed) **`X`** , **`Y`**:
  for every choice of **`Z`** holds: if **`loves(X,Z)`**, and **`loves(Y,Z)`**, and **`X \= Y`**,
  then also holds: **`jealous(X,Y)`**

**???**

# Intuition on “free” variables

```
loves(vincent,mia).
loves(marsellus,mia).
loves(mia,vincent).

jealous(X,Y) :- loves(X,Z), loves(Y,Z), X \= Y.
```

- What is the “logical” interpretation of `Z` above? (or of the whole rule?)

- Possibly, for arbitrary (but fixed) `X` , `Y`:
  if for every choice of `Z` holds: `loves(X,Z)`, and `loves(Y,Z)`, and `X \= Y`,
  then also holds: `jealous(X,Y)`

- Or, for arbitrary (but fixed) `X` , `Y`:
  for every choice of `Z` holds: if `loves(X,Z)`, and `loves(Y,Z)`, and `X \= Y`,
  then also holds: `jealous(X,Y)`

# Intuition on “free” variables

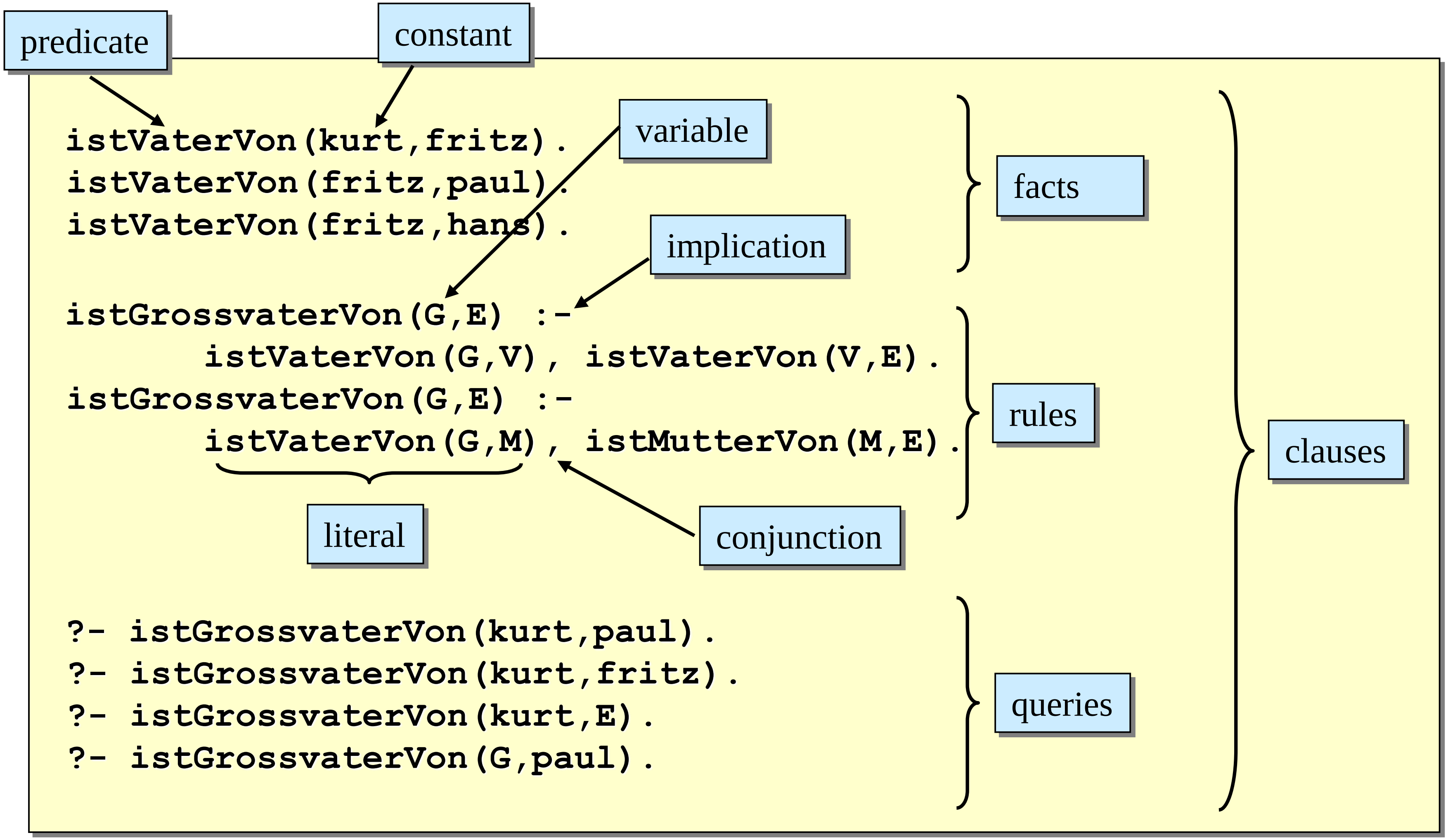
```
loves(vincent,mia).
loves(marsellus,mia).
loves(mia,vincent).

jealous(X,Y) :- loves(X,Z), loves(Y,Z), X \= Y.
```

- What is the “logical” interpretation of **`Z`** above? (or of the whole rule?)

- Or, for arbitrary (but fixed) **`X`** , **`Y`**:
  for every choice of **`Z`** holds: if **`loves(X,Z)`**, and **`loves(Y,Z)`**, and **`X \= Y`**,
  then also holds: **`jealous(X,Y)`**

- Logically equivalent, for arbitrary (but fixed) **`X`** , **`Y`**:
  if for any choice of **`Z`** holds: **`loves(X,Z)`**, and **`loves(Y,Z)`**, and **`X \= Y`**,
  then also holds: **`jealous(X,Y)`**

# Syntax / notions in Prolog



predicate

constant

variable

implication

facts

rules

clauses

literal

conjunction

queries

```
istVaterVon(kurt,fritz).
istVaterVon(fritz,paul).
istVaterVon(fritz,hans).

istGrossvaterVon(G,E)  :-
        istVaterVon(G,V), istVaterVon(V,E).
istGrossvaterVon(G,E)  :-
        istVaterVon(G,M), istMutterVon(M,E).


?- istGrossvaterVon(kurt,paul).
?- istGrossvaterVon(kurt,fritz).
?- istGrossvaterVon(kurt,E).
?- istGrossvaterVon(G,paul).
```

# Language objects in Prolog

- To build clauses, Prolog uses different kinds of objects:

| | |
|---|---|
| - constants | ( numbers, strings, …) |
| - variables | ( X,Y, ThisThing, …) |
| - operator terms | ( … 1 + 3 * 4 …) |
| - structures | ( date(27,11,2007), person(fritz, mueller), …<br>composite, recursive, “infinite”, …) |

- Note: Prolog has no type system!

# Language objects in Prolog

## Constants in Prolog

- Numbers

```
-17   -2.67e+021   0   1   99.9   512
```

- Atoms, i.e. strings that satisfy one of these rules:

1. The string starts with a lower case letter, followed by arbitrarily many lower or upper case letters, numbers and underscores '_'.

2. The string starts and ends with an apostrophe ( ' ). In between, there can be arbitrary characters. If an apostrophe should appear in the string, it has to be denoted twice.

3. The string consists only of symbols.

| | | | | | |
|---|---|---|---|---|---|
| correct: | **`fritz`** | **`new_york`** | **`:-`** | **`-->`** | **`'I don''t know!'`** |
| wrong: | **`Fritz`** | **`new-york`** | **`_xyz`** | **`123`** | |

# Language objects in Prolog

## Variables in Prolog

- Variables:
  - Name starts with an upper case letter or an underscore '_'.
  - Examples: `Land   Jahr   M   V   _45   _G107   _europa`

internal format for variables

- Anonymous variables (simply '_', even if several anonymous variables):
  - if the object is not of interest:

```
?- istVaterVon(_,fritz).
```