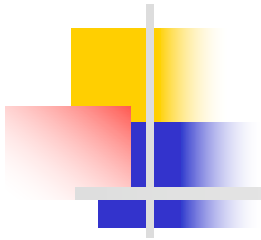
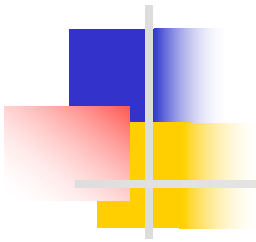
COMP3411-9814- Artificial Intelligence



Itroduciton to Prolog

2019 – Summer Term

Tatjana Zrimec



Outline

- Tools for constructing intelligent systems
- Prolog
- Procedural and Declarative Programing
- Robot's World in Prolog

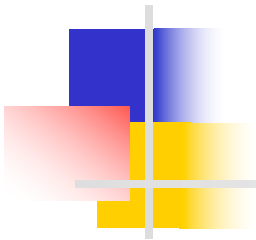
Tools for constructing intelligent systems

- Object-oriented programming languages, for example,
  - C++, Java, and CLOS.
- Traditional procedural programming languages, for example,
  - C, Pascal, and Fortran.
- Artificial-intelligence programming languages for processing words, symbols, and relations, for example,
  - Lisp and Prolog.

AI programming languages: Lisp and Prolog

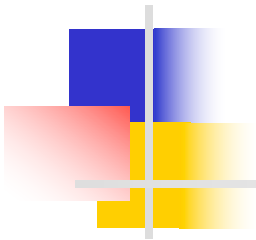
- The two main AI programming languages:
  - Lisp and
  - Prolog

- A key feature of both languages is
  - the ability to manipulate symbolic data: characters and words, as well as numerical data.
  - One of the most important structures for this manipulation is lists.



Prolog

- Prolog is suited to symbolic (rather than numerical) problems, particularly logical problems involving relationships between items.
- It is also suitable for tasks that involve data lookup and retrieval, as pattern-matching is fundamental to the functionality of the language.
- Because Prolog is so different from other languages in its underlying concepts, many newcomers find it a difficult language.
  - Whereas most languages can be learned rapidly by someone with computing experience,
  - Prolog is perhaps more easily learned by someone who has never programmed.



Prolog

- Prolog is an AI language that can be programmed declaratively.
- It is very different from Lisp, which is a procedural (or, more precisely, functional) language that can nevertheless be used to build declarative applications.
- Although Prolog can be used declaratively, programmers need to understand how Prolog uses the declarative information that they supply.

Prolog – answering questions

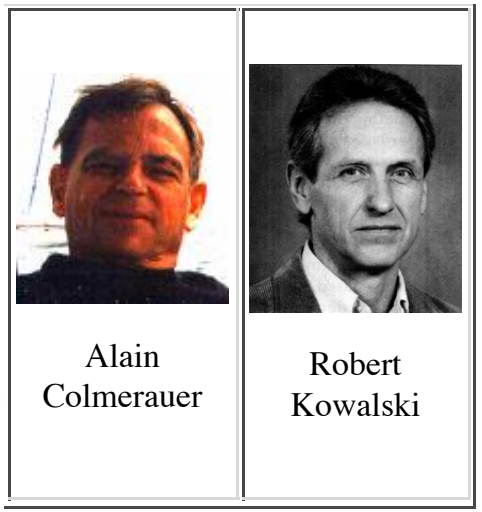
- Prolog has attempted to match the query to the relations (only one relation in our example) that it has stored.

- In order for any two terms to match, either:
  - the two terms must be identical; or
  - it must be possible to set (or instantiate) any local variables in such a way that the two terms become identical.

Prolog – Interpreter

- If Prolog is trying to match two or more clauses and comes across multiple occurrences of the same local variable name, it will always instantiate them identically.
- The only exception to this rule is the underscore character, which has a special meaning when used on its own.
- Each occurrence of the underscore character's appearing alone means: I don't care what '_' matches so long as it matches something.

What is Prolog?



- Prolog = Programmation en Logique (Programming in Logic).
- Prolog is a declarative programming language unlike most common programming languages.

- Invented early seventies by Alain Colmerauer in France and Robert Kowalski in Britain

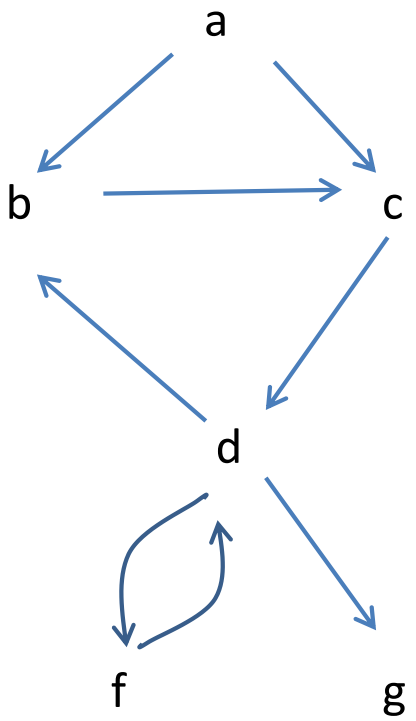
Declarative Programming

- According to one definition, a program is "declarative" if it describes *what* something is like, rather than *how* to create it.

- This is a different approach from traditional imperative programming languages such as Fortran, C, and Java, which require the programmer to specify an algorithm to be run.
  - **Imperative** programs make the **algorithm explicit** and leave the goal implicit
  - **Declarative** programs make the **goal explicit** and leave the algorithm implicit.

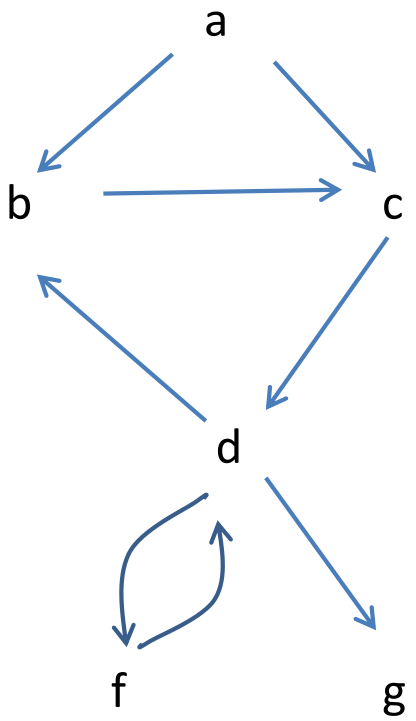
Example: Procedural And Declarative Program

Task: For a given graph, find the path from a to f which has length 3?



Example: Procedural Program

Task: For a given graph, find the path from a to f which has length 3?

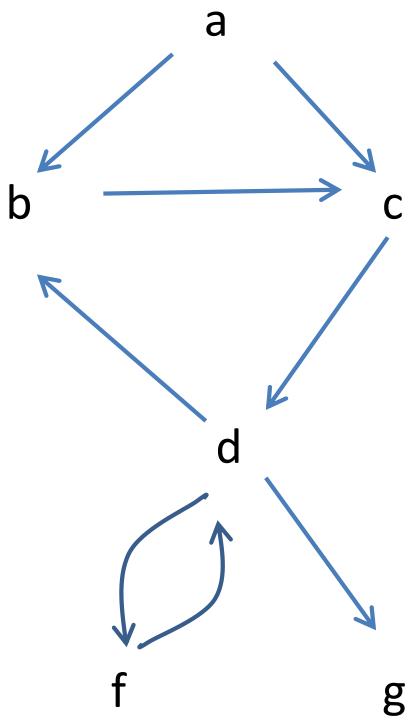


Procedural program:

start at a,
go from a to b,
go from b to c,
go from c to d, length exceeded back to c, back to b, back to a,
go from a to ...

Example: Declarative Program

Task: For a given graph, find the path from *a* to *f* which has length 3?



Declarative Program:

The search path looks like this:

a → **X** → **Y** → **f**

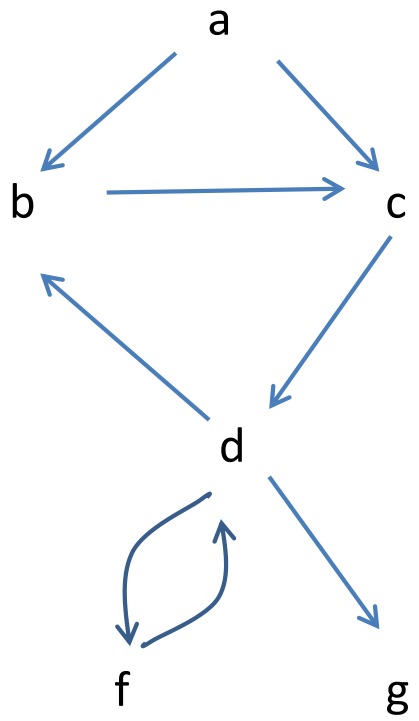
It must be:

a connected with **X**, **X** with **Y**,

Y with **f**

Find the appropriate **X** and **Y**

The graph problem in Prolog



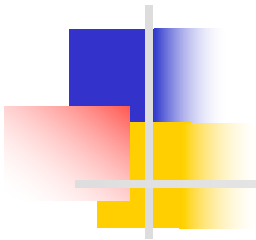
% Graph definition

link(a, b). link(a, c). link(b, c). ...
link(f, d).

% Required relations

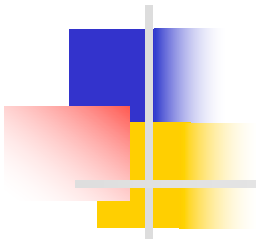
?- link(a, X), link(X, Y), link(Y, f).

X=c, Y=d



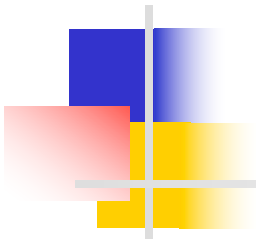
Applications of Prolog

- Some applications of Prolog are:
  - intelligent data base retrieval
  - natural language understanding
  - expert systems
  - specification language
  - machine learning
  - robot planning
  - automated reasoning
  - problem solving



So Why Learn Prolog?

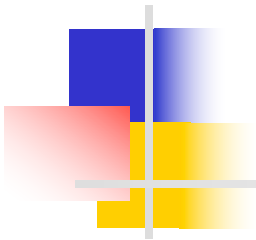
- Prolog is a declarative programming language that suits search and AI programming very well.
- Logic programming languages like Prolog have recently had a resurgence of popularity in the computing industry.
- It is an example of a non-imperative language.



So Why Learn Prolog?

- Prolog programming language is based on a small set of basic mechanisms
  - Pattern matching
  - Tree-based data structuring
  - Automatic backtracking

- Prolog has a powerful and flexible programming framework.



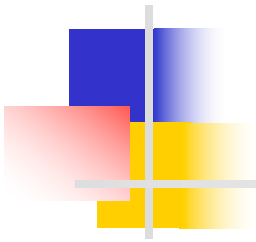
Prolog

- Prolog is a programming language for symbolic, non-numeric computations.
- It is specially suited for problems that involve *object* and *relations*.
- Relations are defined by facts.
  - For example, a family relation: The fact that Tom is a parent of Bob

```
parent( tom, bob).      % fact
```

Here ***parent*** is the name of the relation,

tom and ***bob*** are arguments.



An example of a Prolog program

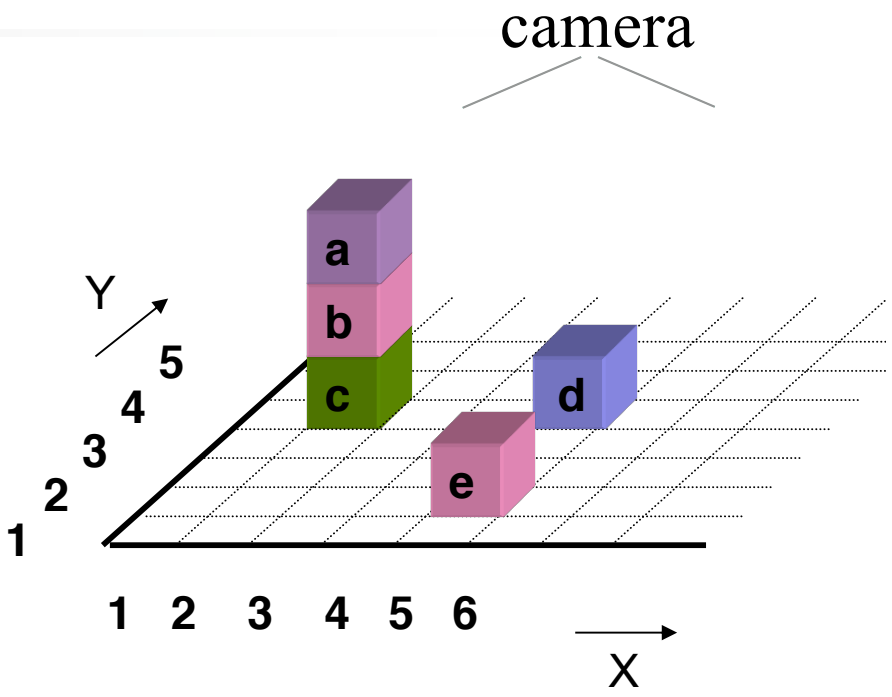
A robot that moves the cubes on the table

- The robot sees the cubes with a ceiling camera (a camera mounted on the ceiling)
- The robot is interested in the coordinates of the cube, whether the cube can be grabbed, etc.

An example of a Prolog program

A robot that can observe and moves the cubes on the table

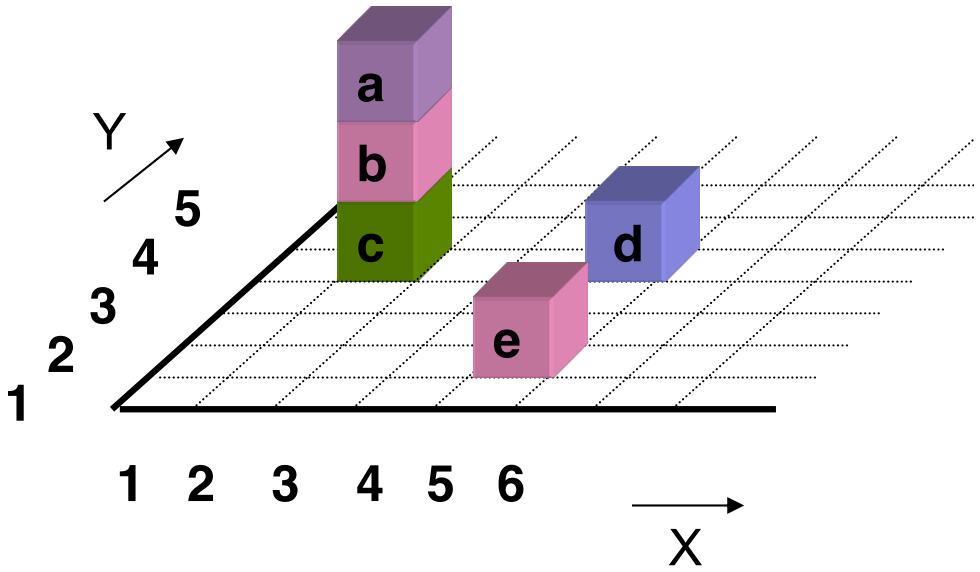
- The robot sees the cubes with a camera mounted on the ceiling.
- The robot is interested in the coordinates of the cube, whether the cube can be grabbed, etc.



A robot's world

Robot's World

```
%  see( Block, X, Y)
see( a, 2, 5).
see( d, 5, 5).
see( e, 5, 2).

% on( Block, BlockOrTable)
on( a, b).
on( b, c).
on( c, table).
on( d, table).
on( e, table).
```



Robot's World

The vision information

% see(Block, X, Y)

see(a, 2, 5).

see(d, 5, 5).

see(e, 5, 2).

Block is standing on object

% on(Block, BlockOrTable)

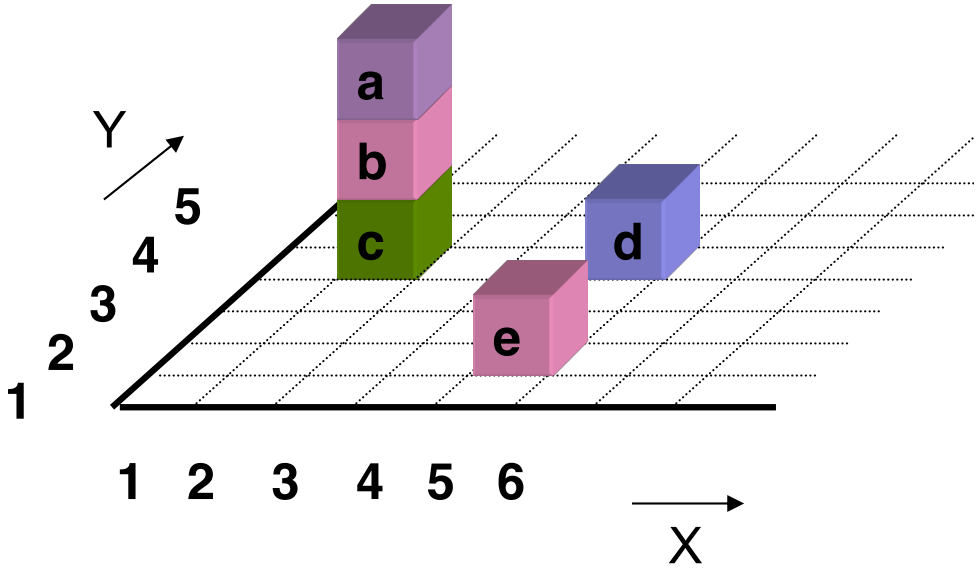
on(a, b).

on(b, c).

on(c, table).

on(d, table).

on(e, table).



Interaction With Robot Program

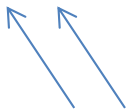
- Start Prolog interpreter

```
?- [robot].
File robot consulted          % Load file robot.pl

?- see( a, X, Y).             % Where do you see block a
X =2
Y =5

?- see( Block, _, _).         % Which block(s) do you see?
Block = a;                    % More answers?
Block = d;
Block = e;
no
```

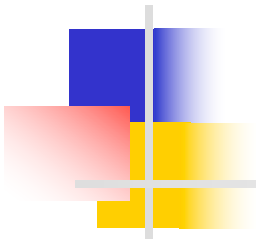
Interaction With Robot Program

- Start Prolog interpreter

```
?- [robot].
File robot consulted          % Load file robot.pl

?- see( a, X, Y).             % Where do you see block a
X =2
Y =5

?- see( Block, _, _).         % Which block(s) do you see?
Block = a;                    % More answers?
Block = d;
Block = e;
no
```

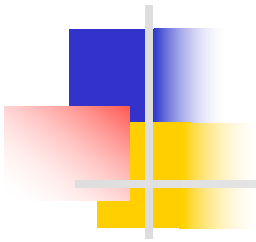


The underscore - An anonymous variable



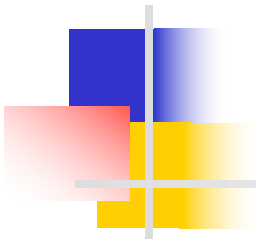
Interaction With Robot Program

```
?- see( B1, _, Y), see( B2, _, Y).    % Blocks at same Y?
```



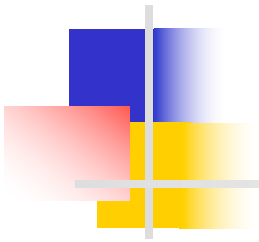
Interaction With Robot Program

```
?- see( B1, _, Y), see( B2, _, Y).    % Blocks at same Y?

% Prolog's answers may surprise!
 B1 = a,  B2 = a
 B1 = a,  B2 = d
 ...
 B1 = e,  B2 = e
```



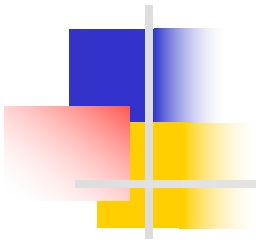
Interaction With Robot Program

```
?- see( B1, _, Y), see( B2, _, Y).    % Blocks at same Y?

% Prolog's answers may surprise!
B1 = a,  B2 = a
B1 = a,  B2 = d
...
B1 = e,  B2 = e

% Perhaps this was intended:
?- see( B1, _, Y), see( B2, _, Y), B1 \== B2.
```



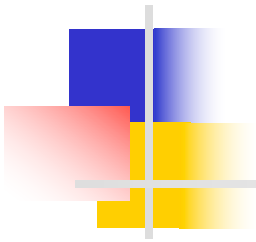
X, Y coordinates

```
% xy( Block, X, Y): X, Y coord. of Block

xy(B,X,Y) :- see( B, X, Y).


xy( B, X, Y) :-
        on( B0, B),
        xy( B0, X, Y).
```



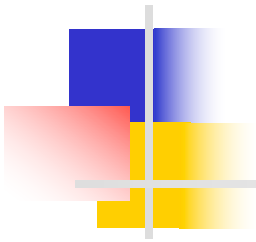
X, Y coordinates

```
% xy( Block, X, Y): X, Y coord. of Block
% Obtain xy-coord for the blocks that are visible
xy(B,X,Y) :- see( B, X, Y).


xy( B, X, Y) :-
        on( B0, B),
        xy( B0, X, Y).
```



X, Y coordinates

```
% xy( Block, X, Y): X, Y coord. of Block
% Obtain xy-coord for the blocks that are visible
xy(B,X,Y) :- see( B, X, Y).

% Blocks in stack have the same xy-coord.
xy( B, X, Y) :-
        on( B0, B),
        xy( B0, X, Y).
```



X, Y coordinates

```
% xy( Block, X, Y): X, Y coord. of Block
% Obtain xy-coord for the blocks that are visible
xy(B,X,Y) :- see( B, X, Y).

% Blocks in stack have the same xy-coord.
xy( B, X, Y) :-
        on( B0, B),
        xy( B0, X, Y).
```

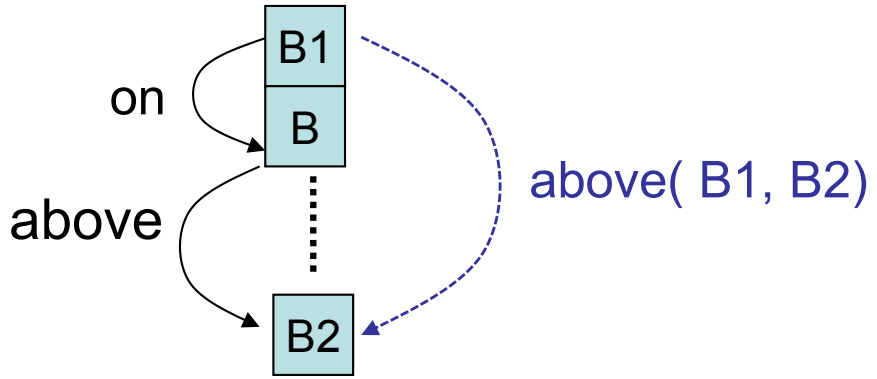
We implemented the robot's reasoning to determine object's coordinated from sensory information !

Relation above(Block1, Block2)

% above(Block1, Block2): Block1 is above Block2 in same stack

```
above( B1, B2) :-
        on( B1, B2).
```


```
above( B1, B2) :-
        on( B1, B),
        above( B, B2).
```



Declarative Meaning of this Program

GIVEN FACTS

```
on( a, b).
on( b, c).
on( c, table).
...

above( B1, B2)  :-
   on( B1, B2).

above( B1, B2)  :-
  on( B1, B),
  above( B, B2).
```

WHAT CAN WE DERIVE FROM THE FACTS?

```
on( a, b).
on( b, c).
...

above( a, b).
above( b, c).
above( a, c).
above( a, table).
...
```

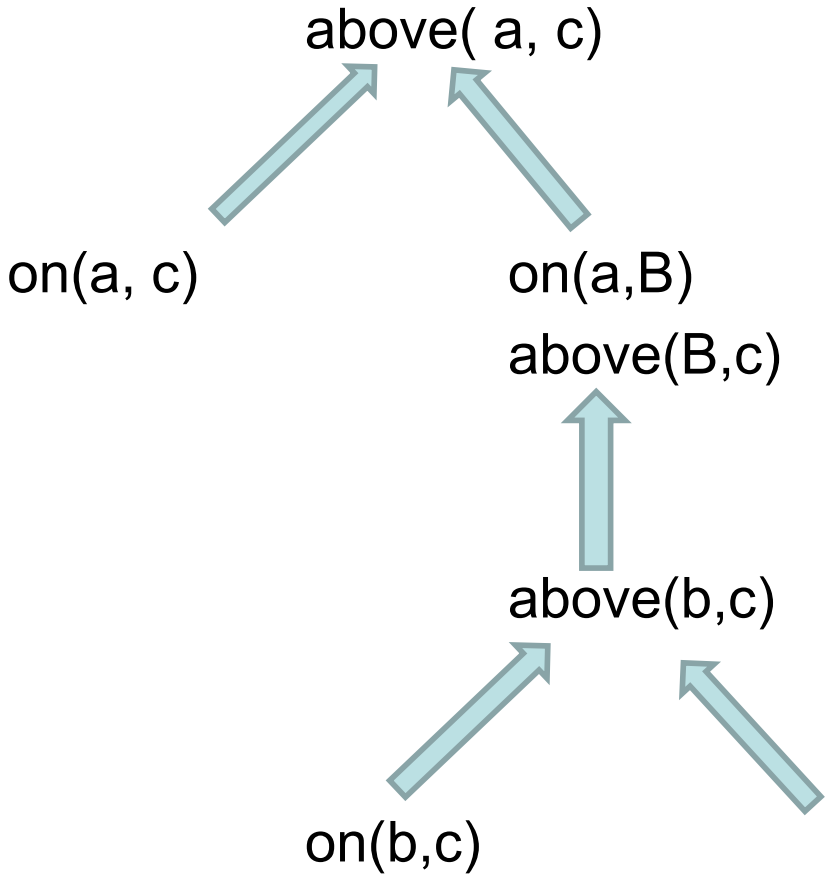
all this constitutes a *declarative meaning*

Relation „Above“

```
% above( Block1, Block2): Block1 above Block2 in the same stack

above( B1, B2) :-
        on( B1, B2).
above( B1, B2) :-
        on( B1, B),
         above( B, B2).

?- above( a, c).          % Trace proof tree for this
```

Prolog seeks evidence to satisfy goals

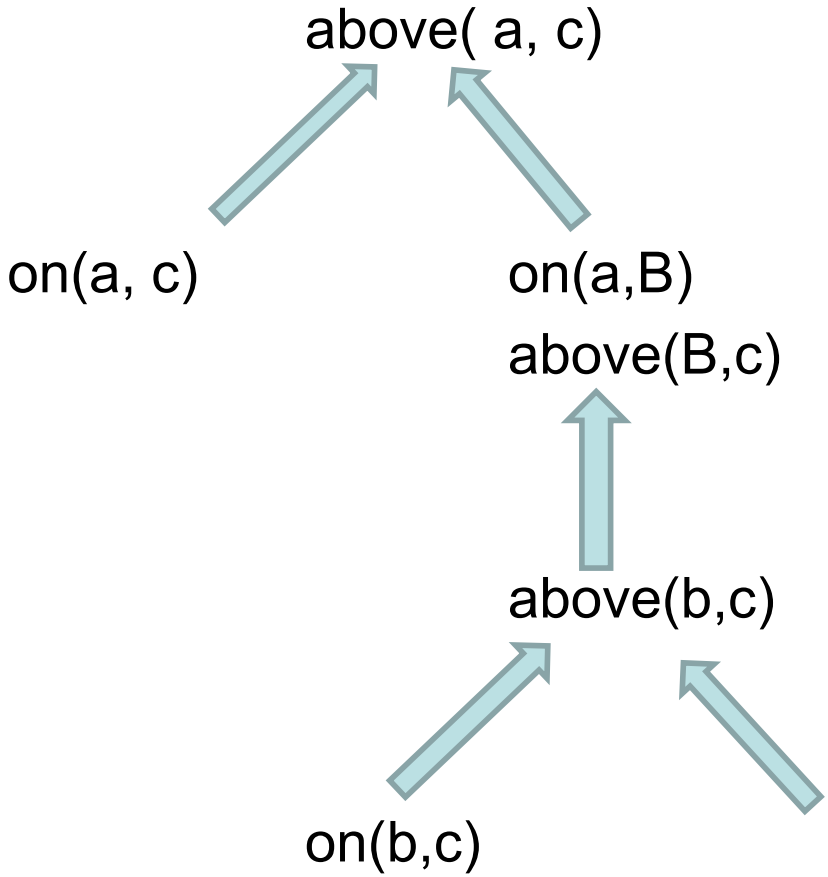
?- above(a, c).



Prolog seeks evidence to satisfy goals

?- above(a, c).

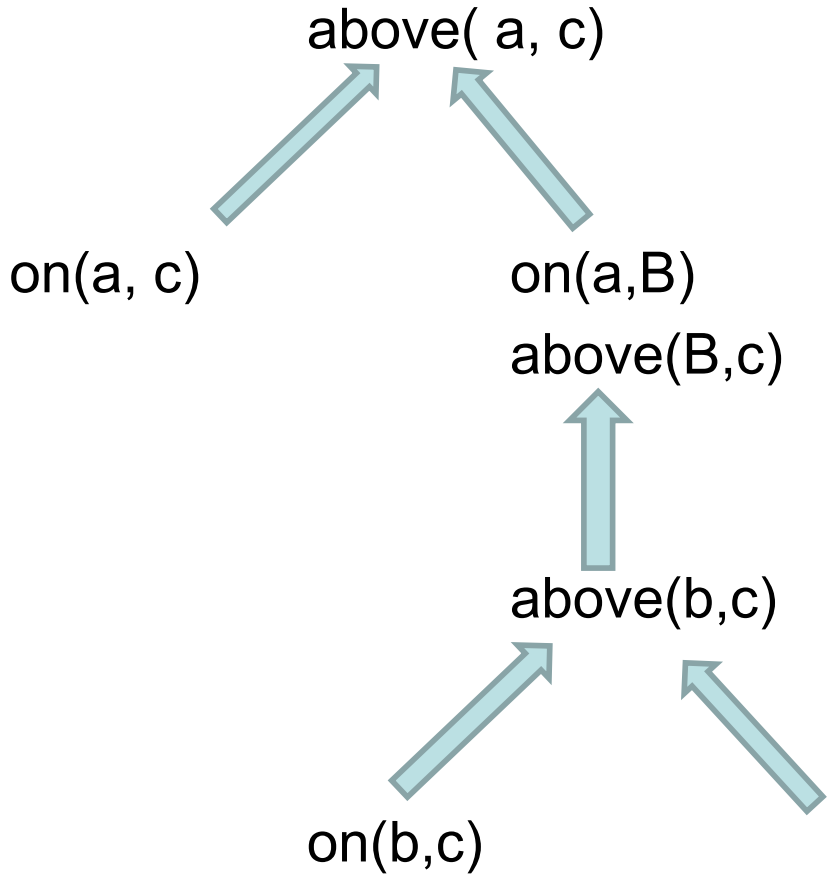
```
above( B1, B2) :-
        on( B1, B2).
above( B1, B2) :-
        on( B1, B),
         above( B, B2).
```



Prolog seeks evidence to satisfy goals

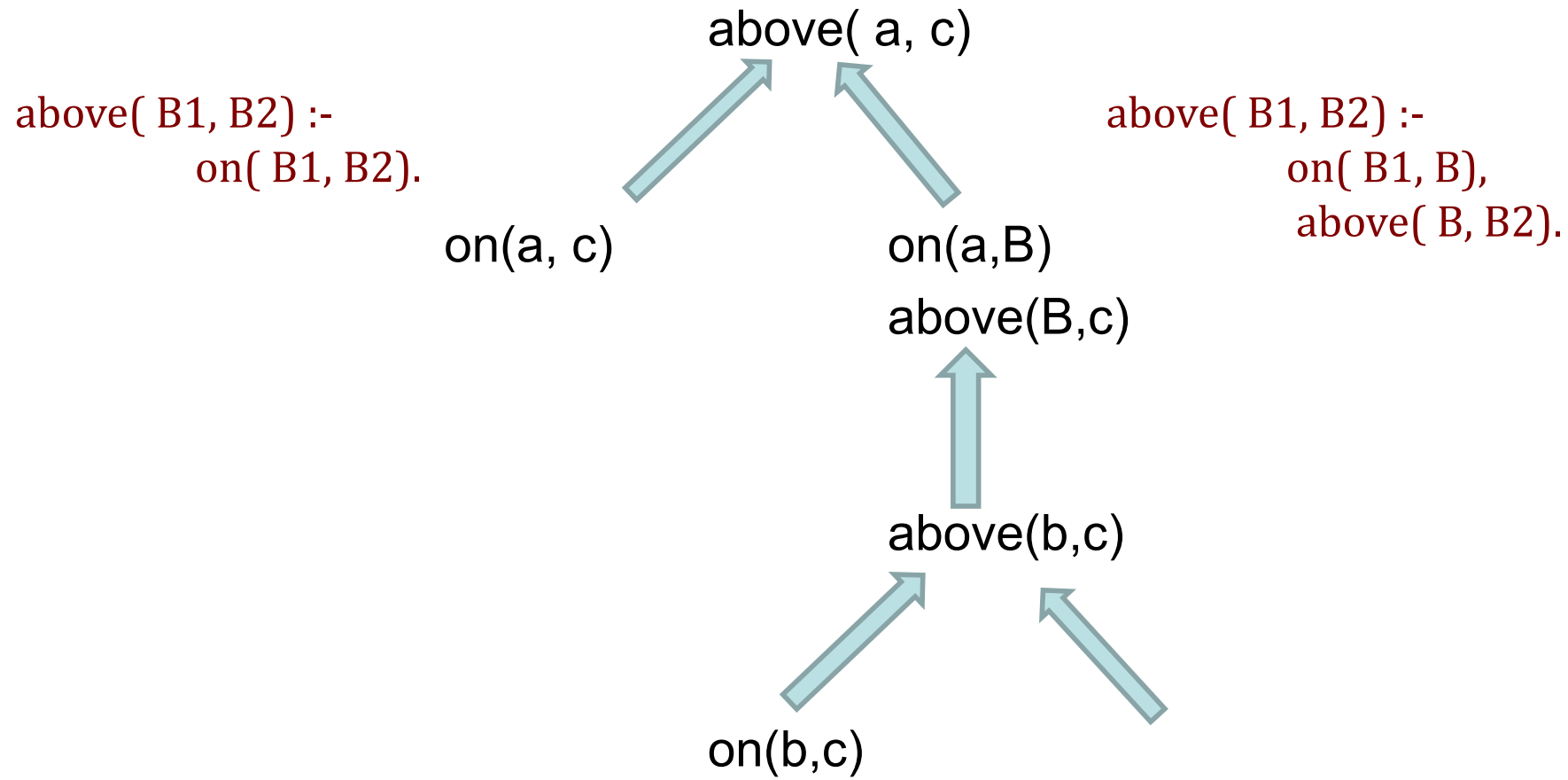
?- above(a, c).

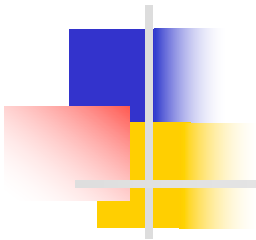
above(B1, B2) :-
on(B1, B2).



Prolog seeks evidence to satisfy goals

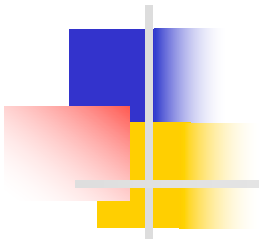
?- above(a, c).





Prolog Interpreter = Theorem prover

- It examines a proof tree - a tree of possible evidence paths
- Automatic backtracking
- Search strategy: in depth (depth first search)



Trace

% Trace proof tree of this execution

```
?- trace.
...
?- on( a, Z).
...
```