COMP3411-9814- Artificial Intelligence

Prolog
Terminilogy & Syntax

2019 – Summer Term

Tatjana Zrimec

Outline

- Terminology
- Syntax and semantics
- Data objects
- Structures
- Matching
- Declarative Meaning

Terminology

Terminology

- Prolog program - a set of *clauses*
- *Clauses* - facts, rules, questions
- *Fact* – things that are always, unconditionally true.

- Rules declare things that are true given condition

$$\underbrace{\texttt{rule(X,Y)}}_{\text{head}}\ \texttt{:-}\ \underbrace{\texttt{part1(X), part2(X,Y).}}_{\text{body}}$$

- Variables - X, Y, B1, X12…
- Constants – numbers or atoms (a1, tom ....)

Terminology - examples

Constants

Numbers:

1 -2 3.14

Atoms:

tigger

'100 Acre Wood'

Variables

X A_variable _

Compound terms

likes(pooh_bear,honey)

plus(4,mult(3,plus(1,9)))

AI 19T0 T.Zrimec

Terminology – examples - Rules

- Example of a rule:

$$\text{above(B1, B2)} \;\text{:-}\; \overset{\text{goal}}{\text{on(B1, B)}},\ \overset{\text{goal}}{\text{above(B, B2)}}.$$

$$\underbrace{\text{above(B1, B2)}}_{\text{head}} \;\text{:-}\; \underbrace{\text{on(B1, B), above(B, B2)}}_{\text{body}}.$$

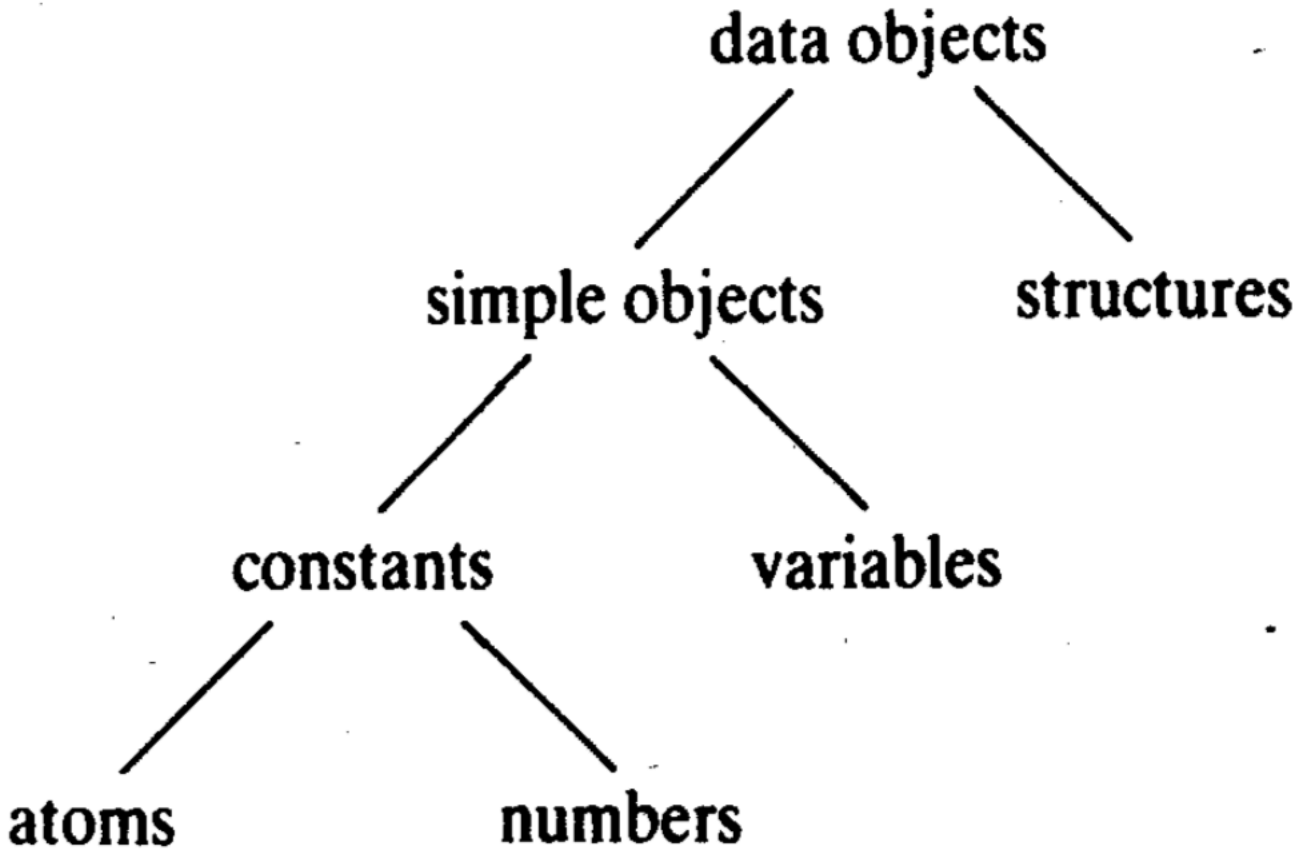
- The *head* is true if the *first goal* and the *second goal* are true.

The term *atom*

- The term atom is used to denote a fundamental data type that cannot be made up from other data types.
- For example:
  - numbers and words are atoms,
  - lists are not atoms.

Prolog – syntax and semantics

Data objects in Prolog



Object Syntax

- The type of object is always recognizable from a syntactic form

Three Syntactic Forms for Atoms

(1) Strings of letters, digits and the underscore character "-", starting with lower case letter

x x15 x_15 aBC_CBa7

alpha_beta_algorithm taxi_35

peter missJones miss_Jones2

Three Syntactic Forms for Atoms

(2) Strings of special characters

--->   <==>   <<

.   <   >   +   ++   !   ..   .::.   ::=   []

(3) Strings of characters enclosed in single quotes

'X_35'   'Peter'   'Britney Spears'

This is useful if we want an atom to start with a capital letter

Numbers

- Strings of special characters

  1 1313 0 -55

- Real numbers

  3.14 -0.0045 1.34E-21 1.34e-21

  Real numbers not much used in Prolog

Variables

- Variable are strings of letters, digits and underscore character :

  X Results Object2B Participant_list

  _x35 _335

- The lexical range of variable names is one clause.

Anonymous Variables

```
visible_block( B)  :-
 see( B, _, _).
```

It is equivalent to :

```
visible_block( B)  :-
 see( B, X, Y).
```

Anonymous Variables

```
visible_block( B)  :-
  see( B, _, _).
```

- Each occurrence of the underscore character's appearing alone means: I don't care what '_' matches so long as it matches something.

- Multiple occurrences of the character can be matched to different values.
- The '_' character is used when the value of a variable is not needed in the evaluation of a clause.

Structures

- Strukture so objekti z več komponentami
  - Npr.: datum je struktura s tremi komponentami
  - Datum 5. marec 2017:

date(5, march, 2017)

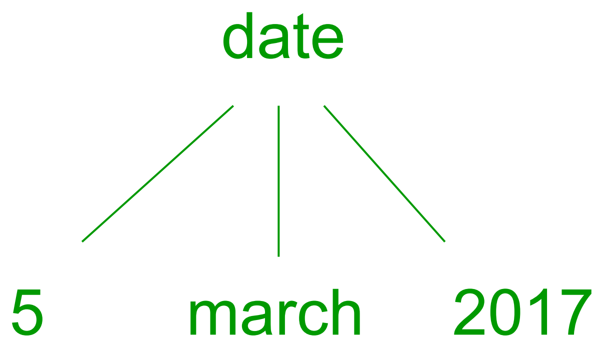
functor *arguments*

- The argument can be any object, including the structure

Tree representation of structures

- Structures are sometimes illustrated as trees:

date(5, march, 2017)

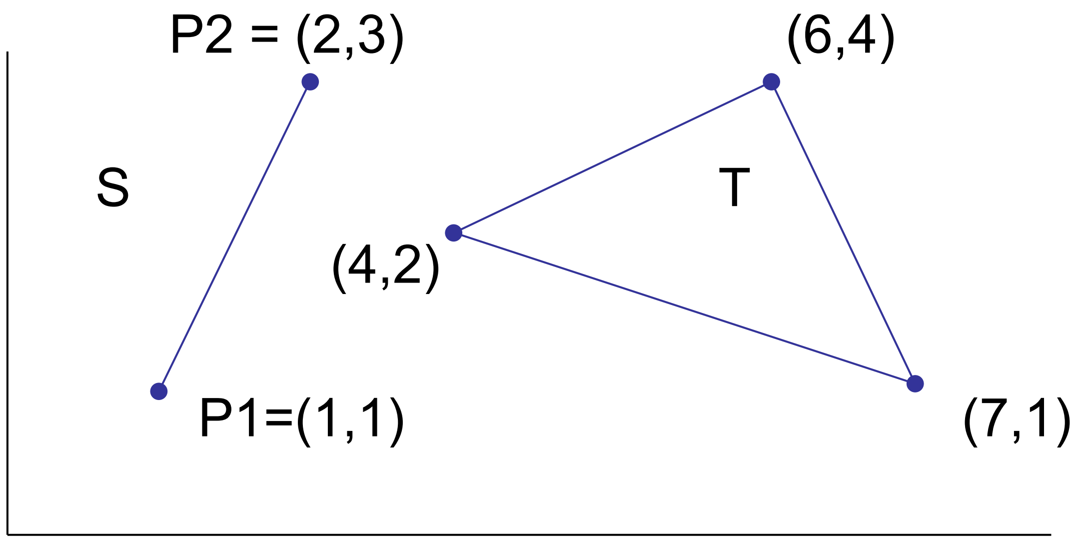


Structure

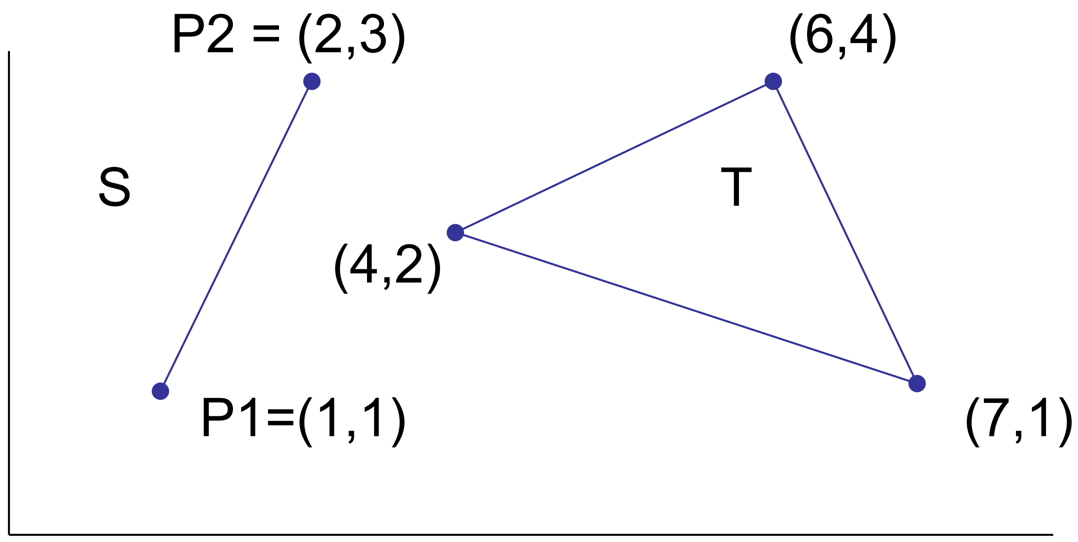
- All structured objects in the prolog can be illustrated by trees
  - This is the only way of constructing structures in a Prolog

- Syntactically all abject in Prolog are “*terms*”

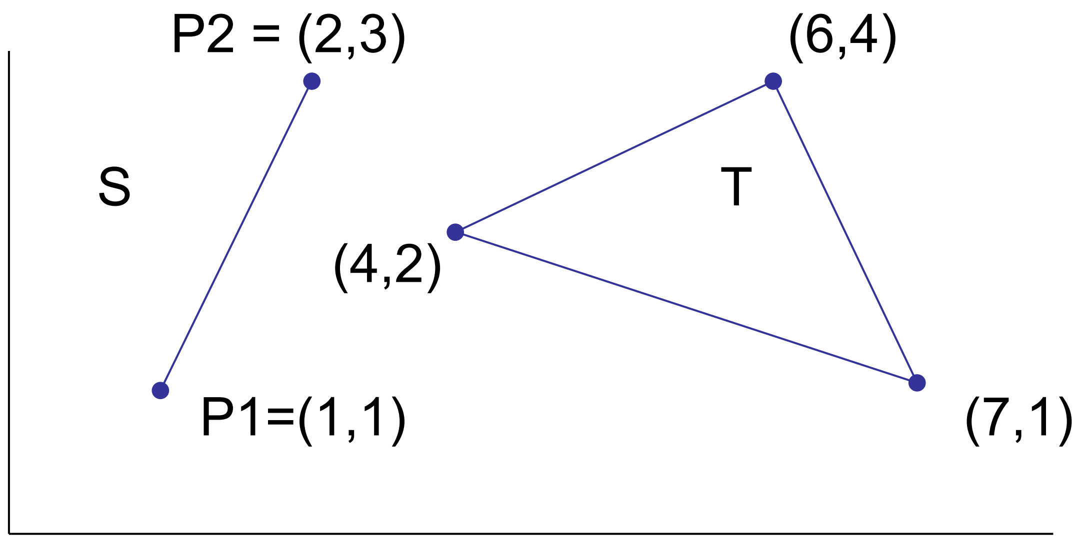
Some simple geometric objects



Some simple geometric objects

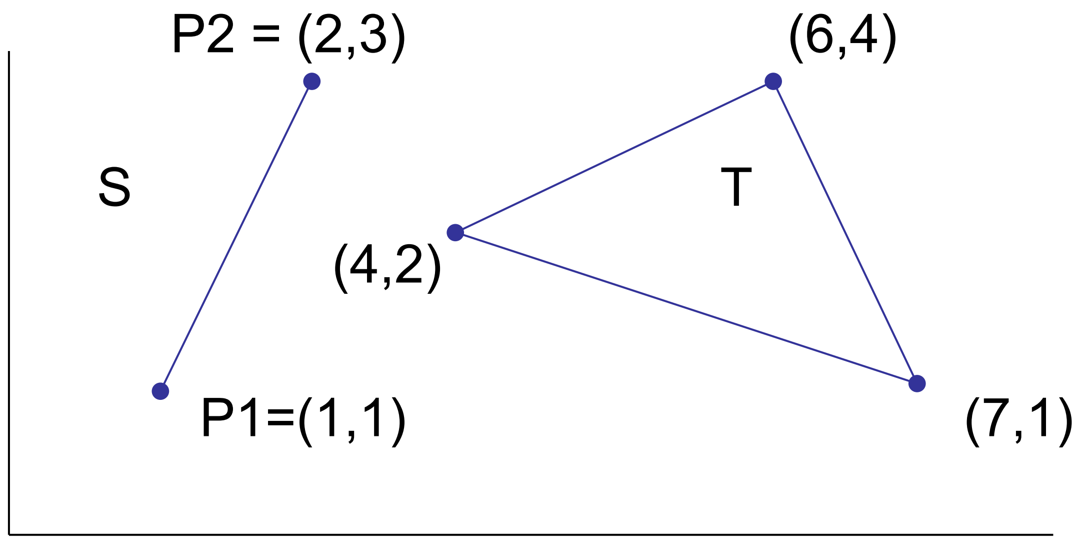


Some simple geometric objects



P1 = point(1, 1)   P2 = point(2, 3)

Some simple geometric objects

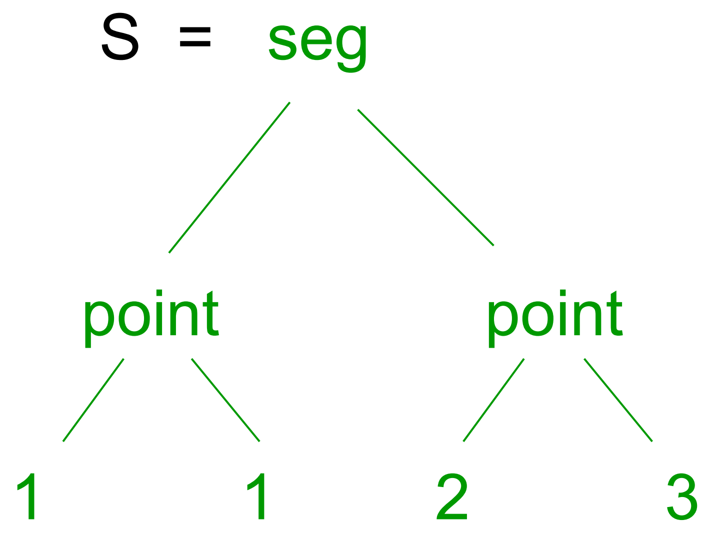


P1 = point(1, 1) P2 = point(2, 3)

S = seg(P1, P2) = seg(point(1,1), point(2,3))
T = triangle(point(4,2), point(6,4), point(7,1))

Segment

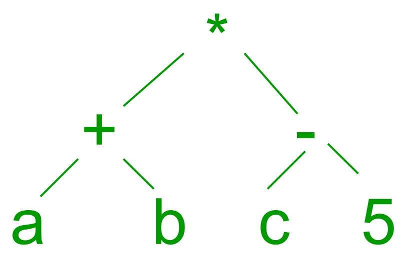
S = seg(point(1,1), point(2,3))



The Arithmetic Expressions are also Trees

- For example: (a + b) * (c - 5)
- Written as an expression with the functors:

*(+(a, b), -(c, 5))



Matching

- Matching is an operation on *terms.*
- Two s structures can be customized if:
  - (1) They are identical, or
  - (2) We can make them identical with appropriate definition of variables

definition of variable

- the variable gets the value =
- instantiation of variable

Matching

- Matching is an operation on *terms.*

Given two *terms, they march* if:

(1) They are identical, or

(2) The variable in both terms can be instantiated to objects in such a way that after the substitution of variables by these objects in the terms become identical

- substitution of variable
  - the variable gets a value = instantiation of variable

Examples of *Matching*

- *Matching of dates*:

  date(D1, M1, 2006) = date(D2, june, Y2)

- One instantiation that make both trems identical

  D1 = D2

  M1 = june

  Y2 = 2006

- This is the most general instantiation, there are others that are less general…
- For matching using the operator “=“

Matching- most general instantiation

- Prolog always returns the most general instantiation.
- With this instantiation leaves grater freedom for further instantiation if further *Matching* is required

?- date(D1, M1, 2006) = date(D2, june, Y2),
date(D1, M1, 2006) = date(17, M3, Y3).

D1 = 17, D2 = 17, M1 = june, M3 = june,
Y2 = 2006, Y3 = 2006

Matching

- *Matching* succeeds or fails.

Two terms S and T match:

1. If S and T are constants, then they match only if they are identical
2. If S is a variable, the *Matching* succeeds, S becomes equal to T.
3. If S and T are structure, then they are adjusted only if:

    a) they have the same principal functor and

    b) all their corresponding components match.

    The resulting instantiation is determined by the matching of the components

Example

- Prolog – finding answers

Prolog – finding answers

- Prolog uses depth first search to find answers !

```
a(1).
a(2).
a(3).
b(1).
b(2).
b(3).
c(A,B) :- a(A), b(B).
```

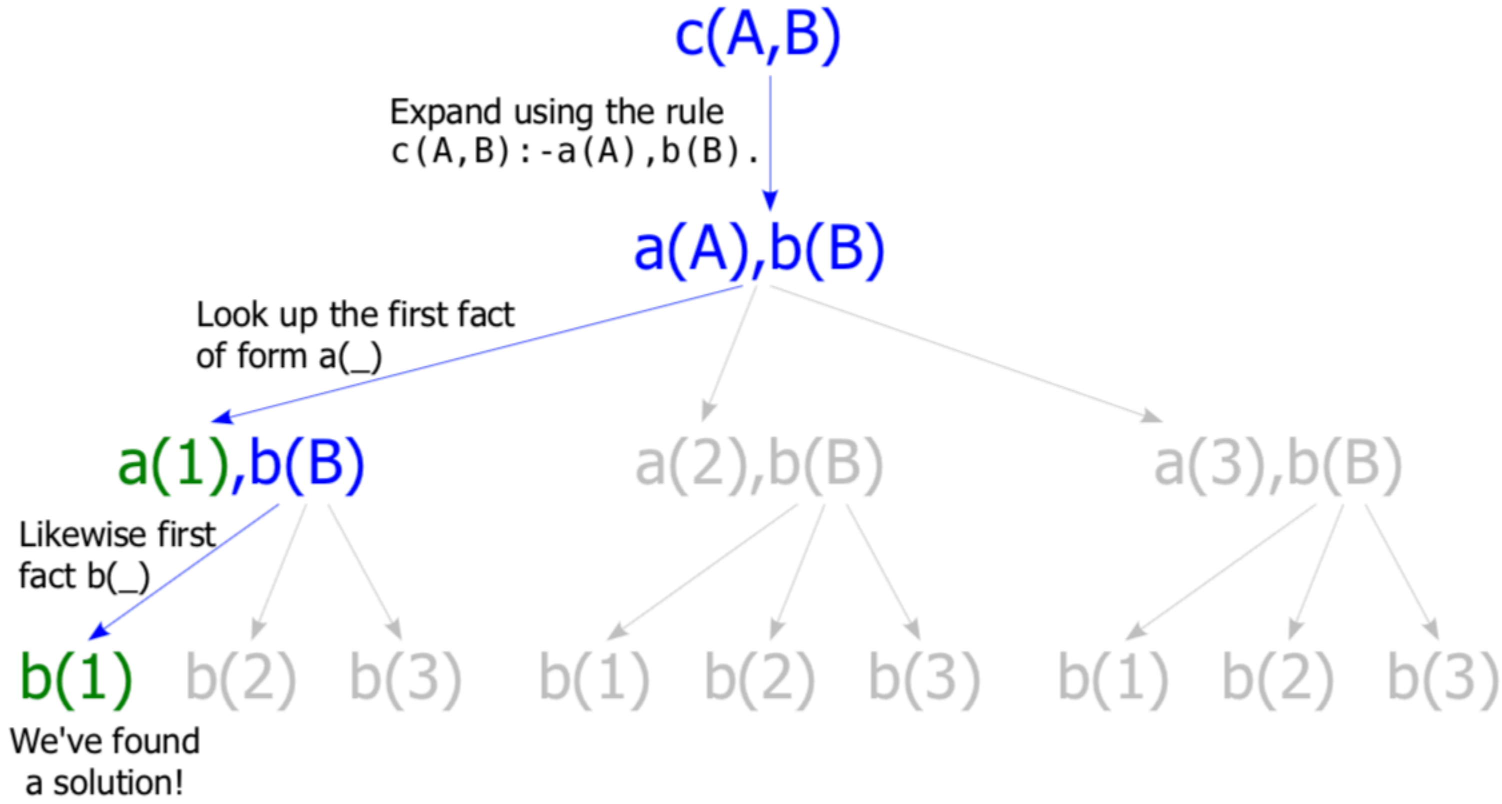
Prolog – finding answers

- Prolog uses depth first search to find answers !

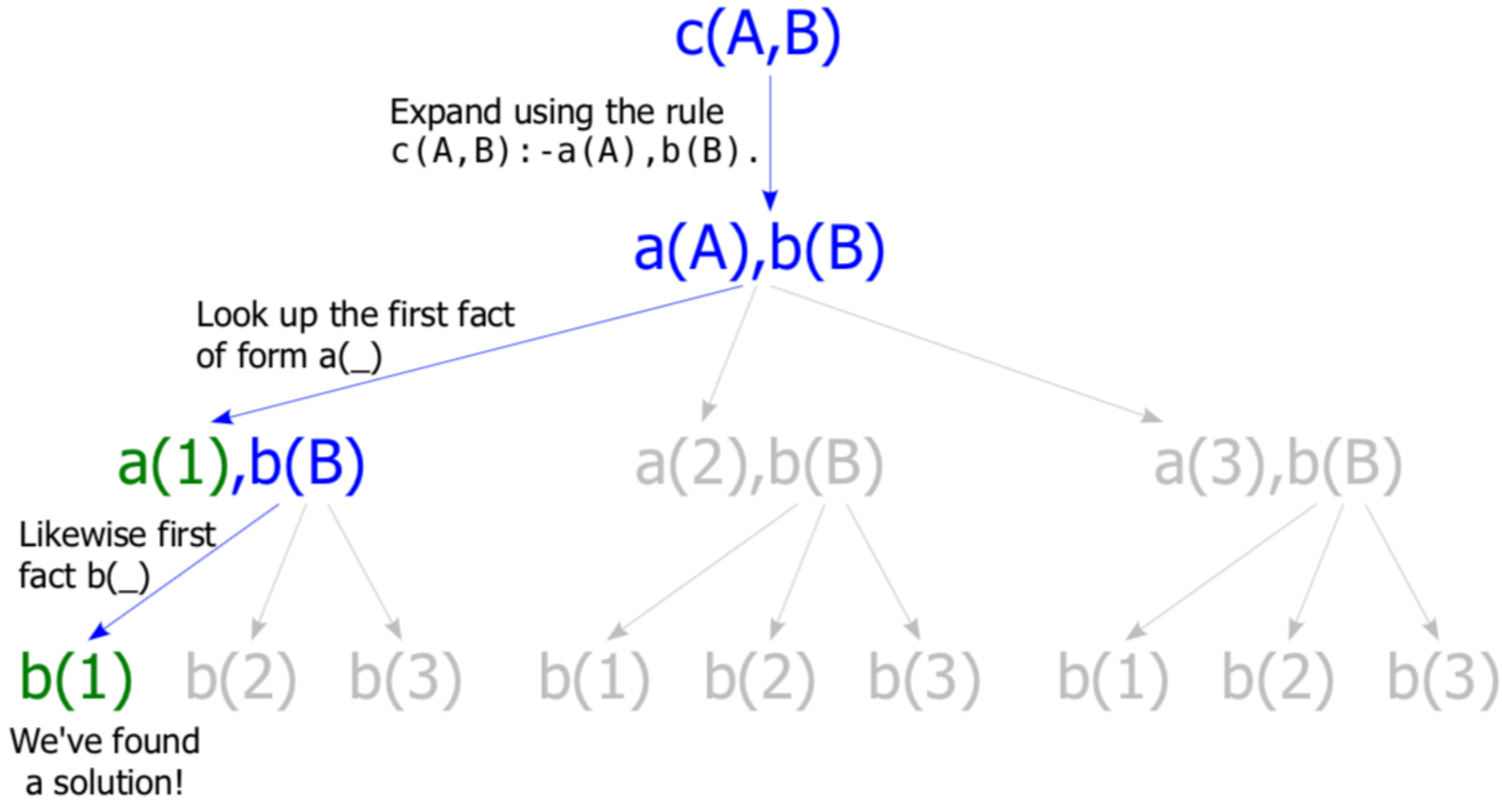
```
a(1).
a(2).
a(3).
b(1).
b(2).
b(3).
c(A,B) :- a(A), b(B).
```

What does Prolog do when given this query ? c(A,B)

Depth-first solution of query c(A,B)

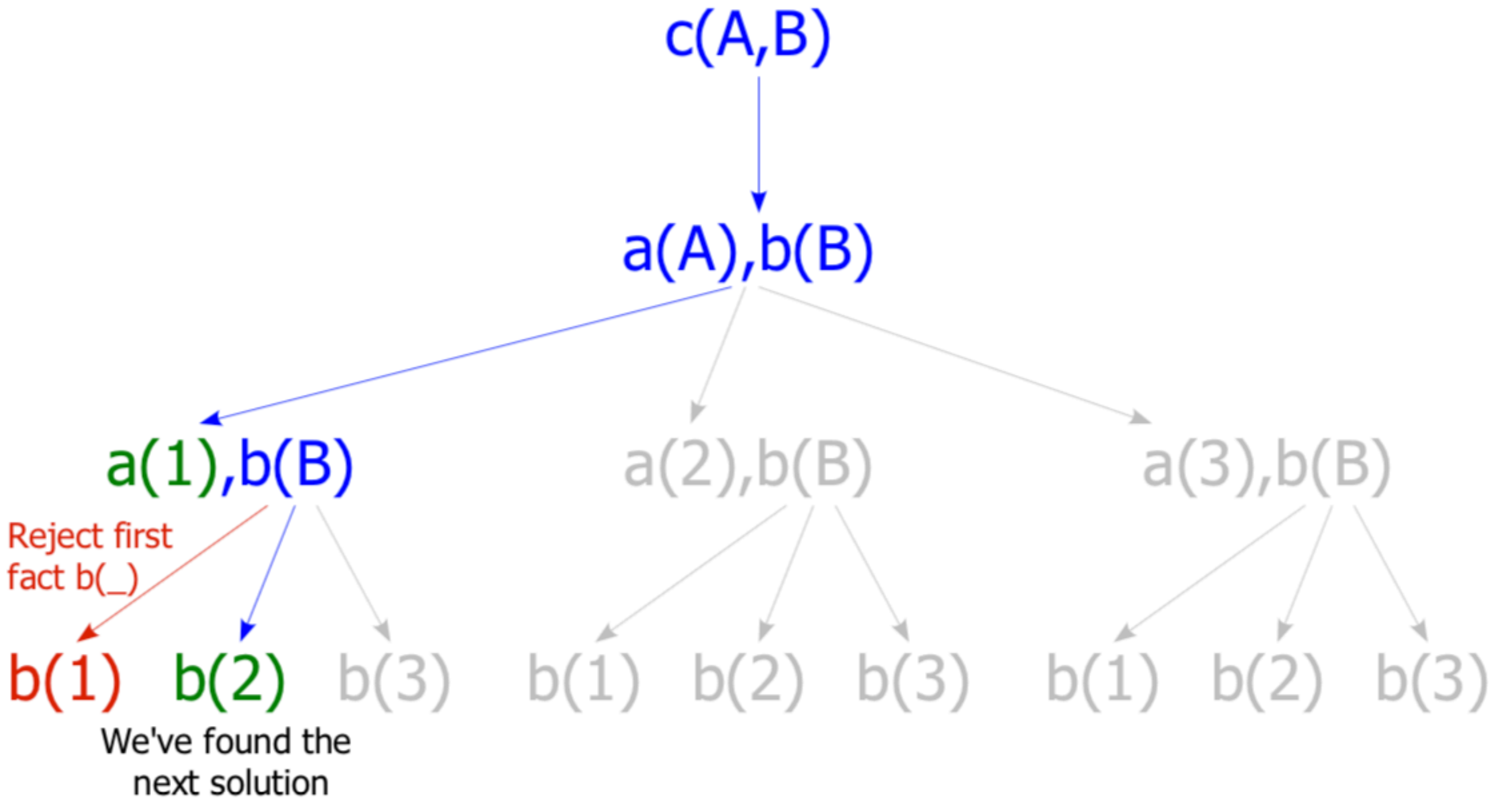


Depth-first solution of query c(A,B)

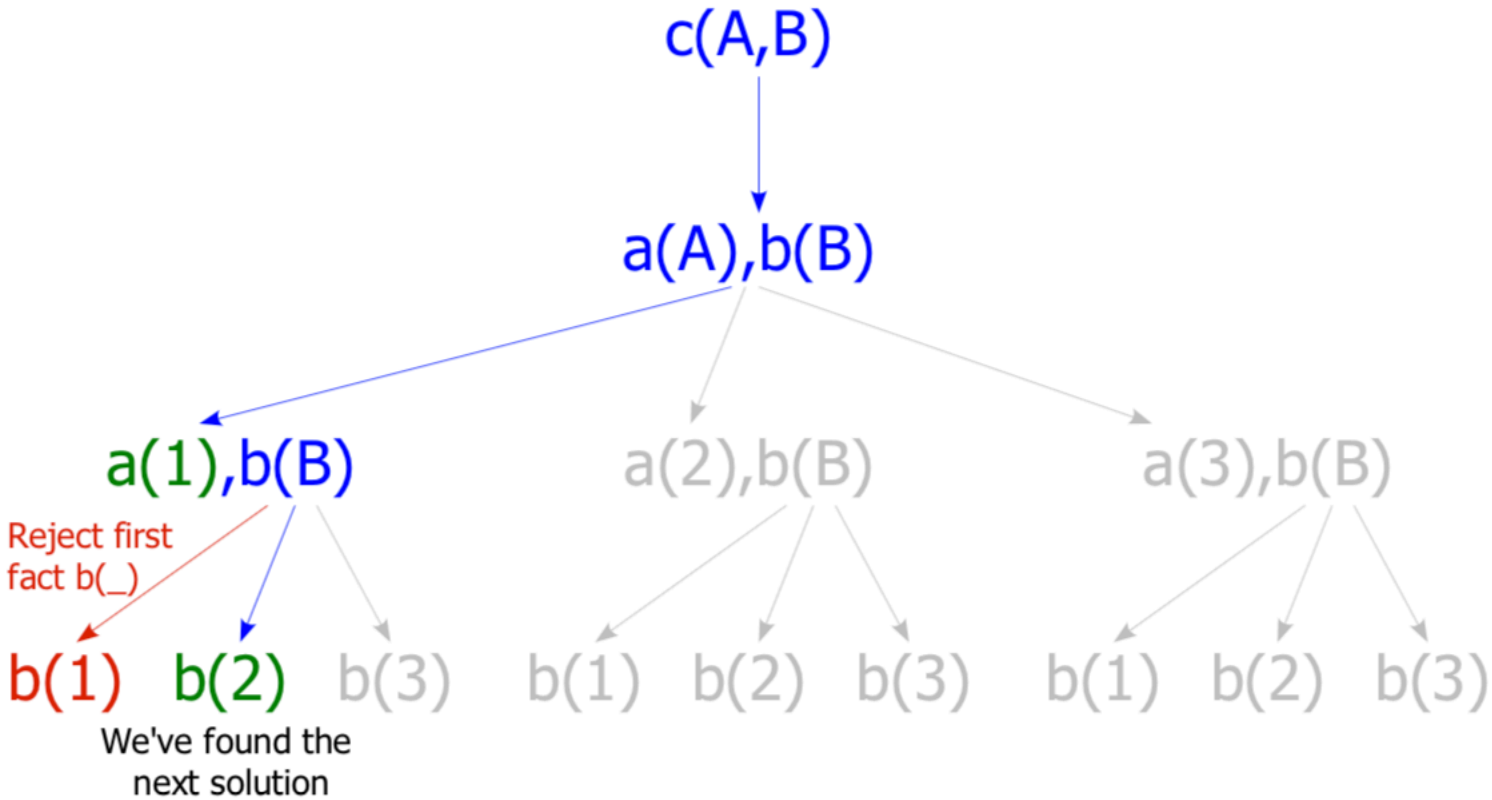


Variable bindings : A= 1, B=1

Backtrack to find another solution

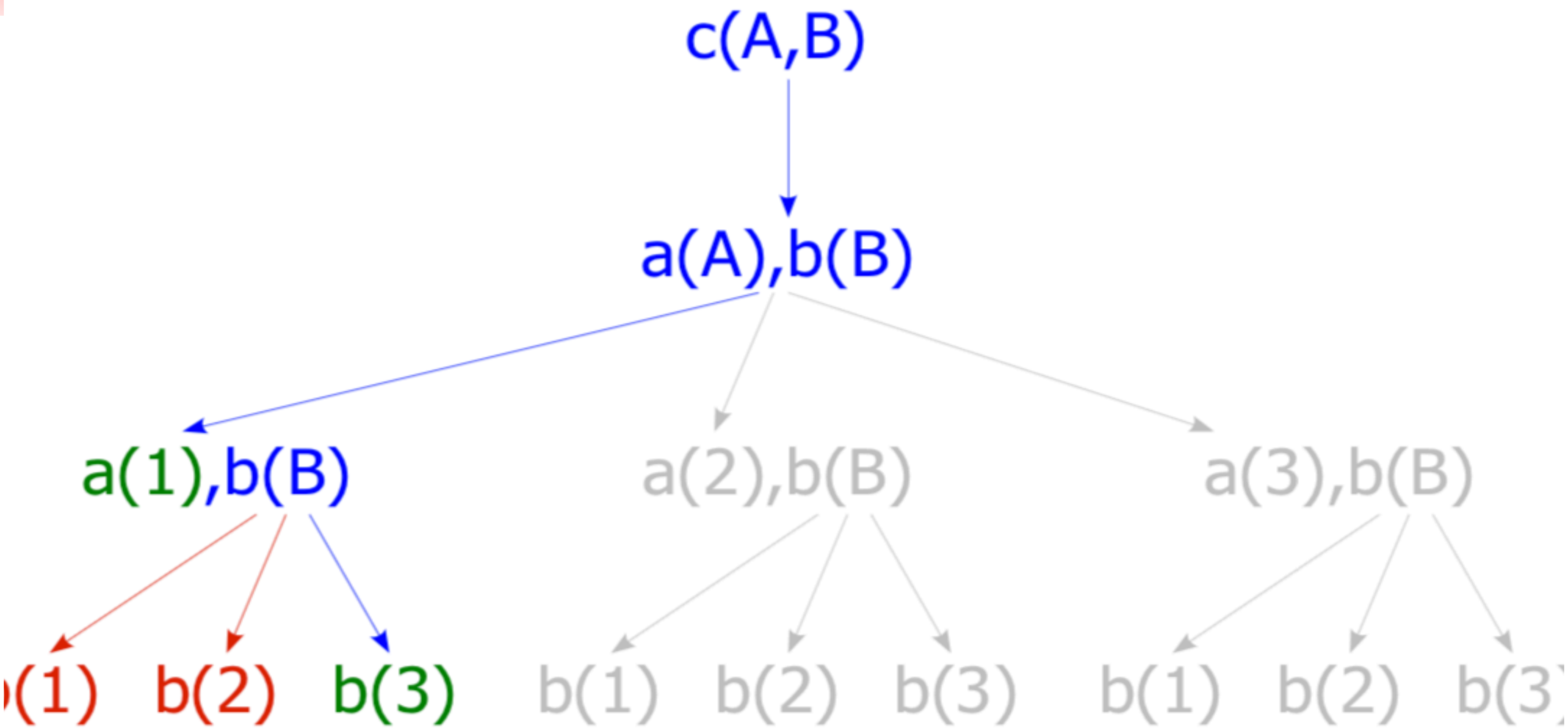


Backtrack to find another solution



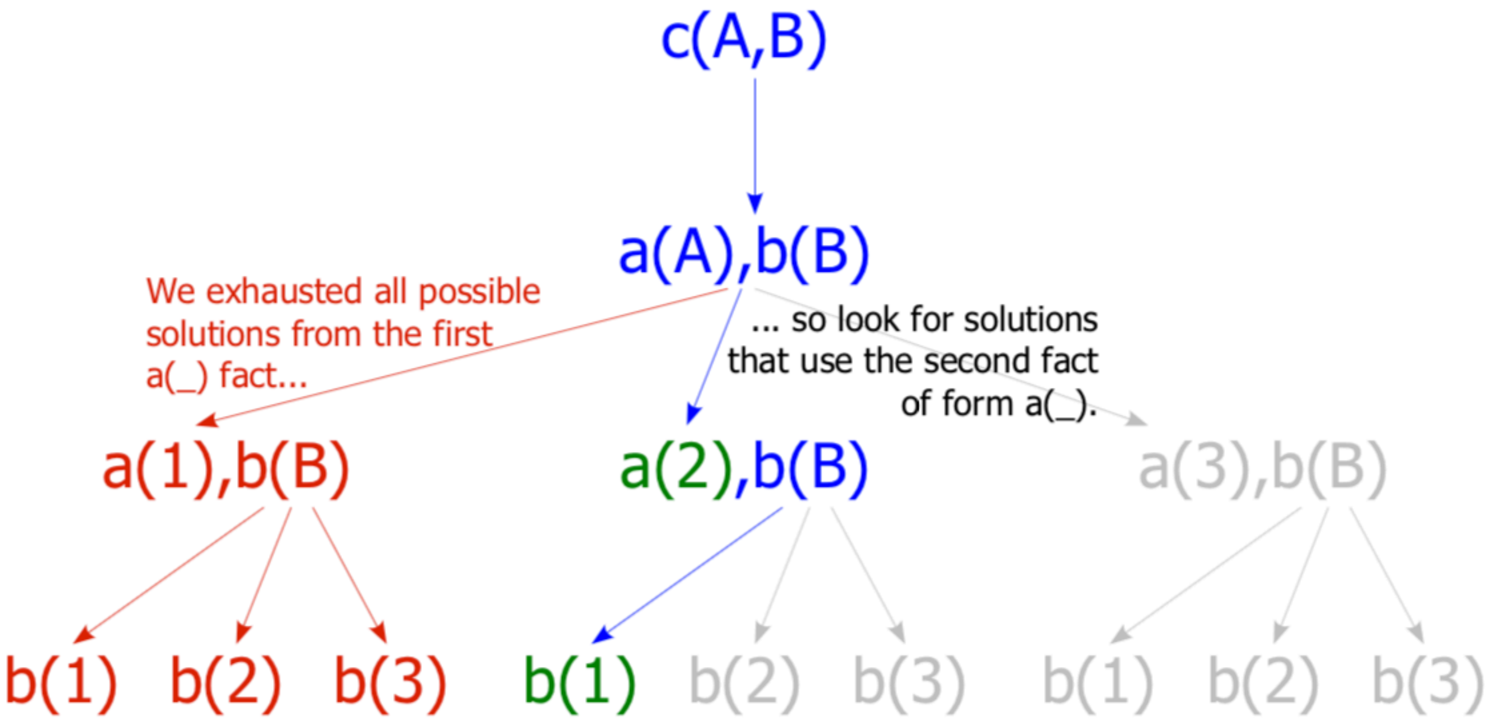
Variable bindings : A= 1, B=2

Backtrack to find another solution



Variable bindings : A= 1, B=3

Backtrack to find another solution



Variable bindings : A= 2, B=1

Declarative Meaning

- Let P be program and target G
- A goal G is true (this is a logical follow from P), if and only if:
  - (1) There is a clause C in P that is valid
  - (2) There is clause instance I of C such that
    - (a) the head of I is identical to G, and
    - (b) all the goals in the body of I are true

- In general, a question to Prolog is a *list* of goals separated by comas. A list of goals is true if all the goals in the list are true for some instantiation of variables.
- The values of the variables result from the most general instantiation.

Declarative and Procedural Meaning of programs

- Let look at the clause:
  P :- Q, R.

- Declarative reading of the clause:
  - P is true if Q and R are true.
  - From Q and R follows P.

- Procedural reading:
  - To solve the problem P, solve Q and then R.
  - To prove P, first prove Q and then R.

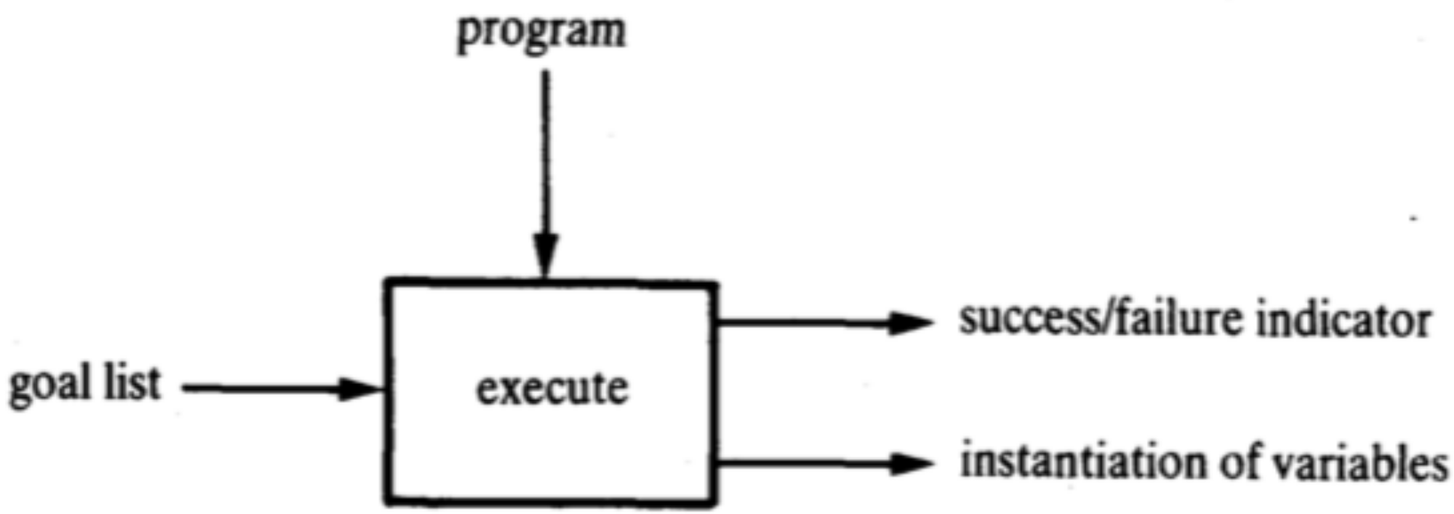
Declarative and Procedural Meaning of programs

- A & B is logically equivalent to B & A

- Declarative meaning - only the relations defined by the program - *What* will be the output of a program
- The order of the goals in the clauses ***does not influence*** the declarative meaning

- The procedural meaning – how the relations are actually derived by the Prolog system
  - The algorithm
- The order of the goals in the clauses **influence** the procedural meaning

Procedural meaning



Input/output view of the procedure that executes a list of goals.

It shows how Prolog prove goals

A procedural meaning is an algorithm for executing a list of goals with respect to a given program