General Game Playing (GGP)
Winter term 2013/2014

2. Prolog / Datalog

Sebastian Wandelt

WBI, Humboldt-Universität zu Berlin

Organization: Group finding

- Do you need help finding a group?

Organization: Examination

- Three options:
  - Oral
  - Written
  - *Special* examination such that
    1. You pass the course if you give a short presentation about your approach (each group around 5-10 minutes) and hand in a small abstract describing your solution(around 1 A4 page)
    2. Your actual grade is determined by a multiple-choice test with 30 questions (per student, not per group!)

Which one do you prefer?

Outline

Date	What will we do?
22.10.2013	Introduction, Repetition propositional logic and FOL
29.10.2013	Repetition FOL / Datalog and Prolog
05.11.2013	Game Description Language
12.11.2013	Design of GDL games
19.11.2013	Search Algorithms 1
26.11.2013	Search Algorithms 2
03.12.2013	Incomplete information
10.12.2013	Fluent Calculus and Fluxplayer
17.12.2013	**Midterm competition**
14.01.2014	Meta-Gaming
21.01.2014	Game Theory
28.01.2014	?
04.02.2014	**Final Competition**
11.02.2014	**Exam**

Prolog:
By example

Basic workflow when using Prolog

1. Describe the situation of interest as a set of facts and rules
2. Ask a question
3. Prolog logically deduces new facts about the situation we described
4. Prolog gives us its deductions back as answers

A simple example of a prolog database

We have four facts:

```
female(mutti).
male(peerlusconi).
male(obami).
male(westerwave).
```

A simple example of a prolog database

Let us add four more facts:

```
hasTelephone(mutti).
hasTelephone(peerlusconi).
usesTelephone(mutti)
controls(obami,nsa)                %Facts can have arbitrary arity
```

Keep in mind, we still have:

```
female(mutti).
male(peerlusconi).
male(obami).
male(westerwave).
```

A simple example of a prolog database

Let us add a simple rule:

```
eavesdrops(X,Y):-
    controls(X,nsa),hasTelephone(Y),usesTelephone(Y).
```

What could that mean?

A simple example of a prolog database

Let us add a simple rule:

```
eavesdrops(X,Y):-
    controls(X,nsa),hasTelephone(Y),usesTelephone(Y).
```

What could rule that mean?

Anybody who controls nsa, eavesdrops everybody who has a telephone and uses the telephone.

A simple example of a prolog database

```
eavesdrops(X,Y):-
    controls(X,nsa),hasTelephone(Y),usesTelephone(Y).

hasTelephone(mutti).
hasTelephone(peerlusconi).
usesTelephone(mutti)
controls(obami,nsa)
female(mutti).
male(peerlusconi).
male(obami).
male(westerwave).
```

Task:

Name, by intuition, all pairs of the eavesdrops relationship!

A simple example of a prolog database

```
eavesdrops(X,Y):-
    controls(X,nsa),hasTelephone(Y),usesTelephone(Y).

hasTelephone(mutti).
hasTelephone(peerlusconi).
usesTelephone(mutti)
controls(obami,nsa)
female(mutti).
male(peerlusconi).
male(obami).
male(westerwave).
```

Task:

Name, by intuition, all pairs of the eavesdrops relationship!

Result:

eavesdrops(obami,mutti)

A simple example of a prolog database

```
eavesdrops(X,Y):-
    controls(X,nsa),hasTelephone(Y),usesTelephone(Y).

hasTelephone(mutti).
hasTelephone(peerlusconi).
usesTelephone(mutti)
controls(obami,nsa)
female(mutti).
male(peerlusconi).
male(obami).
male(westerwave).
```

Task:

What needs to be changed/added such that we have eavesdrops(obami,peerusconi)?

At least three alternatives …

Prolog implementation

- We will use SWI-Prolog from http://www.swi-prolog.org

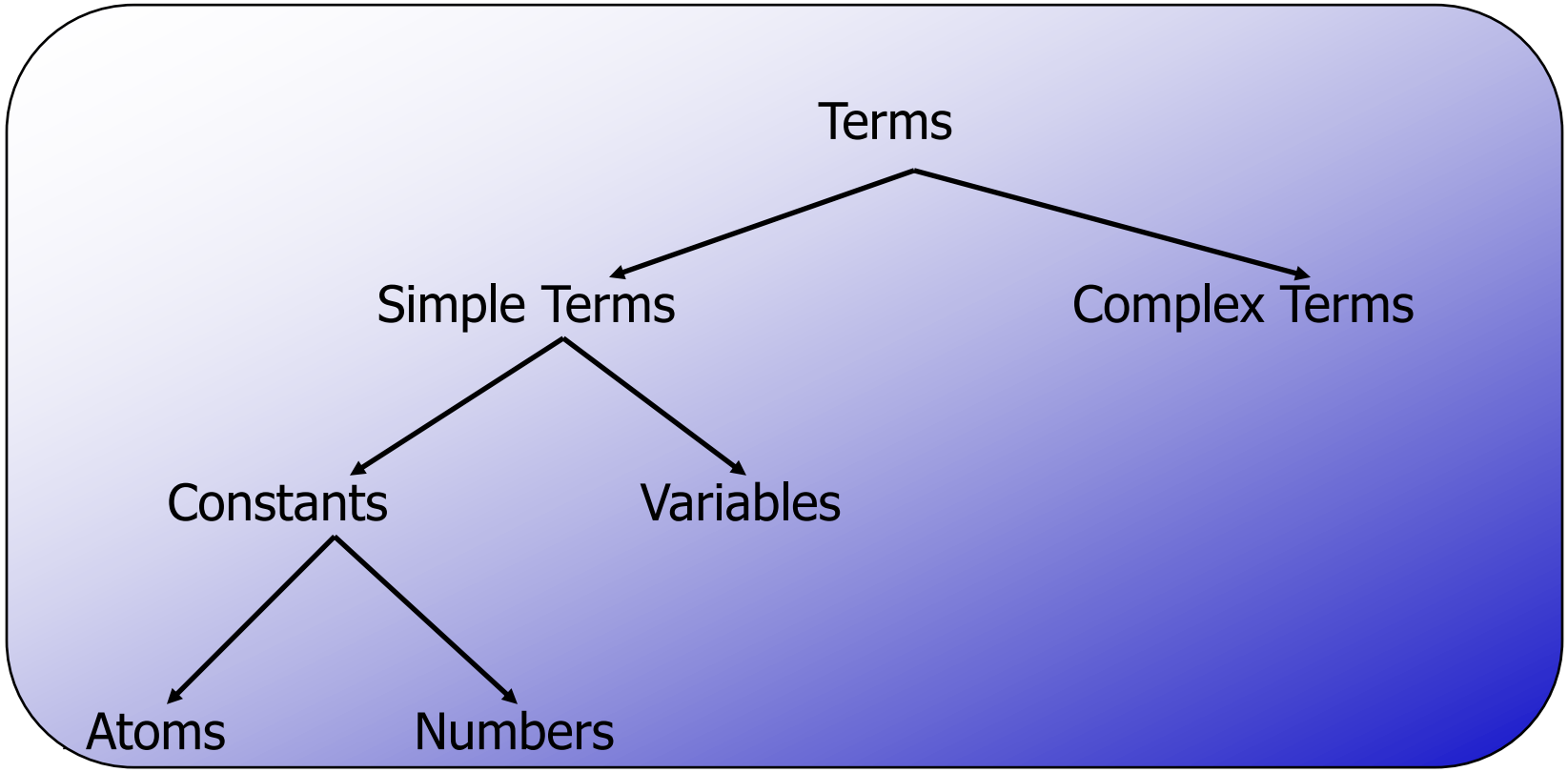
- Cheat sheet:
  - Add a fact/rule at top-level: assert(…).
  - Remove a fact/rule at top-level: retract(…)
  - Load a file with facts/rules: [‘D:\pl\test.pl’].
  - Remove the definition of a fact/rule: retract/retracall
  - Print the definition of a binary function f: listing(f/2).
  - Find next solution: hit ; key
  - Auto-completion: hit tabulator key

  - DON’T forget “.” at the end!

Prolog:
Formal introduction

Prolog Syntax

- What exactly are facts, rules and queries built from?



Atoms

- A sequence of characters of upper-case letters, lower-case letters, digits, or underscore, starting with a lowercase letter
  - *Examples*: **butch**, **big_kahuna_burger**, **playGuitar**

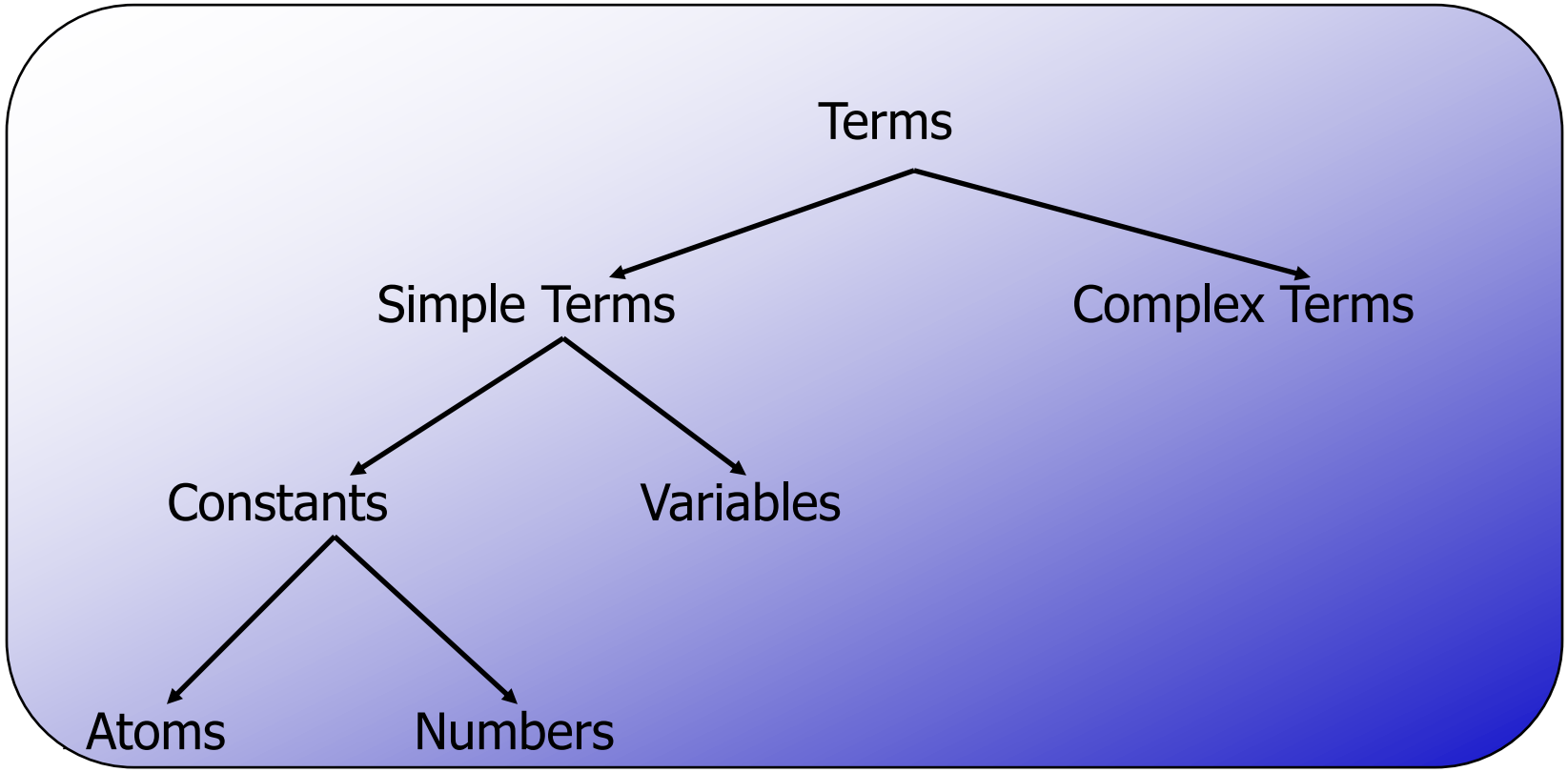
- An arbitrary sequence of characters enclosed in single quotes
  - *Examples*: **'Vincent'**, **'Five dollar shake'**, **'@$%'**

Numbers

- Integers: 12, -34, 22342
- Floats: 34573.3234

Prolog Syntax

- What exactly are facts, rules and queries built out of?



Variables

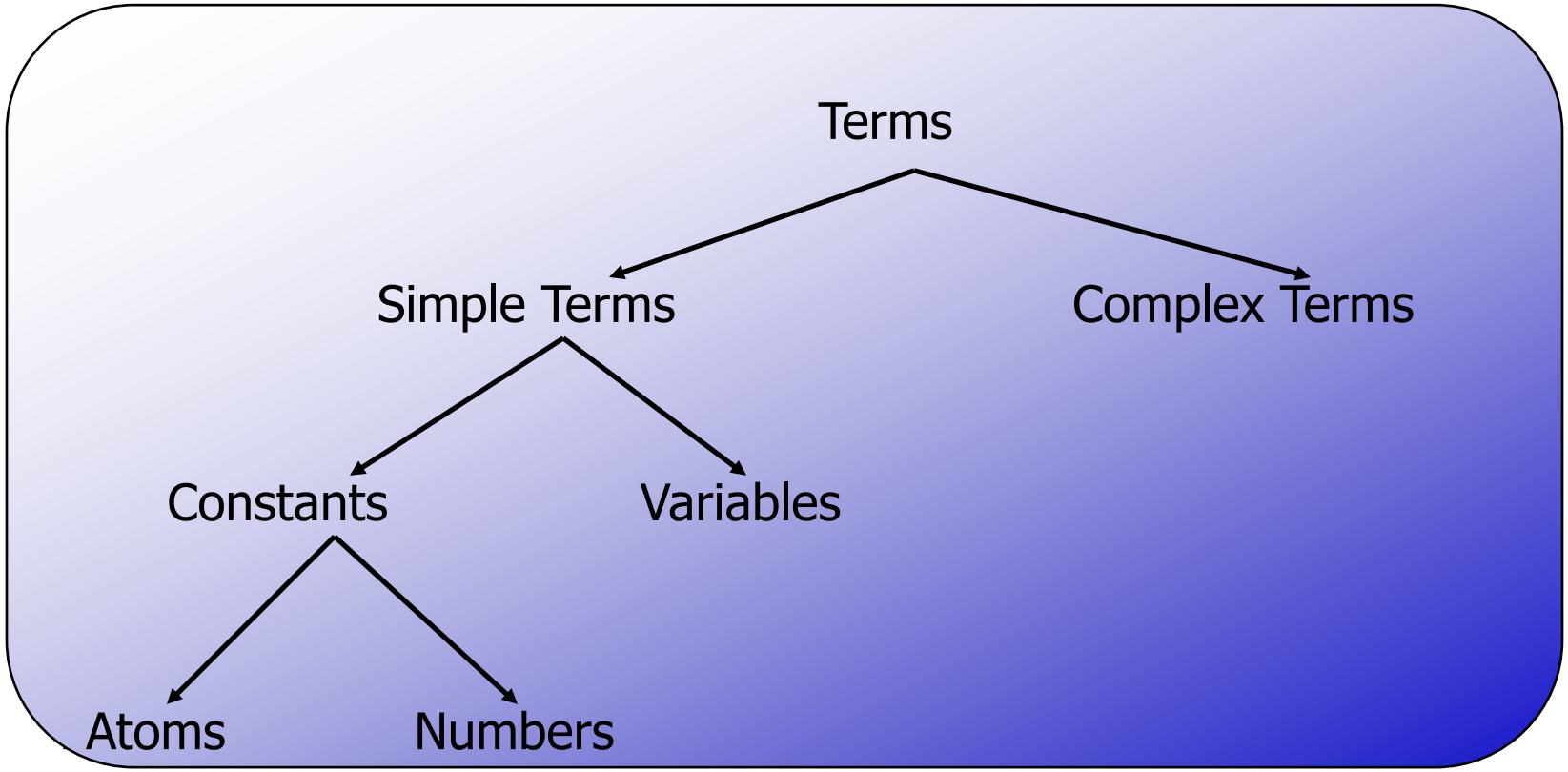
- A sequence of characters of upper-case letters, lower-case letters, digits, or underscore, starting with either an uppercase letter or an underscore

- Examples:

  X, Y, Variable, Vincent, _tag

Prolog Syntax

- What exactly are facts, rules and queries built out of?



Complex Terms

- Atoms, numbers and variables are building blocks for complex terms
- Complex terms are built out of a functor directly followed by a sequence of arguments
- Arguments are put in round brackets, separated by commas
- The functor must be an atom

Examples of complex terms

- Examples:
  - playsAirGuitar(jody)
  - loves(vincent, mia)
  - jealous(marsellus, W)

- Complex terms inside complex terms:
  - hide(X,hasfather(X,Y))

Arity

- The number of arguments a complex term has is called its arity

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```

Arity is important

- In Prolog you can define two predicates with the same functor but with different arity
- Prolog would treat this as two different predicates
- In Prolog documentation arity of a predicate is usually indicated with the suffix "/" followed by a number to indicate the arity

Facts/rules/queries

- Fact:
  - Complex term
- Rule:
  - A:- B,C,D,…
  - Where A and B and C and D and … are complex terms
  - Intuition: **If** B **and** C **and** D **and** … are true, **then** A is true
  - (There is more, but we only discuss this later)
- Query:
  - Complex term

Exercise

Task:

Define a Prolog knowledge base, consisting of facts and rules, describing the Simpsons family.

Use your imagination to define interesting rules!

Prolog: Unification

Unification

- Unification (just as in FOL) is one of the most important operations of any Prolog implementation
- For instance, Prolog unifies

  woman(X)

  with

  woman(mia)

  thereby instantiating the variable **X** with the atom **mia**.

Unification

- Working definition:
  - Two terms unify if they are the same term or if they contain variables that can be uniformly instantiated with terms in such a way that the resulting terms are equal

Unification

- This means that:
  - **mia** and **mia** unify
  - **42** and **42** unify
  - **woman(mia)** and **woman(mia)** unify
- This also means that:
  - **vincent** and **mia** do not unify
  - **woman(mia)** and **woman(jody)** do not unify

Unification

- What about the terms:
  - **luis** and **X**

Unification

- What about the terms:
  - **luis** and **X**
  - **woman(Z)** and **woman(mia)**

Unification

- What about the terms:
  - **luis** and **X**
  - **woman(Z)** and **woman(mia)**
  - **loves(mia,X)** and **loves(X,vincent)**

Instantiations

- When Prolog unifies two terms it performs all the necessary instantiations, so that the terms are equal afterwards
- This makes unification a powerful programming mechanism

Revised Definition 1/3

1. If T_1 and T_2 are constants, then
   T_1 and T_2 unify if they are the same atom, or the same number.

Revised Definition 2/3

1. If T_1 and T_2 are constants, then T_1 and T_2 unify if they are the same atom, or the same number.
2. If T_1 is a variable and T_2 is any type of term, then T_1 and T_2 unify, and T_1 is instantiated to T_2. (and vice versa)

Revised Definition 3/3

1. If T_1 and T_2 are constants, then T_1 and T_2 unify if they are the same atom, or the same number.
2. If T_1 is a variable and T_2 is any type of term, then T_1 and T_2 unify, and T_1 is instantiated to T_2. (and vice versa)
3. If T_1 and T_2 are complex terms then they unify if:
   a) They have the same functor and arity, and
   b) all their corresponding arguments unify, and
   c) the variable instantiations are compatible.

Prolog unification: =/2

?- mia = mia.

yes

?-

Prolog unification: =/2

?- mia = mia.

yes

?- mia = vincent.

no

?-

Prolog unification: =/2

?- mia = X.

X=mia

yes

?-

How will Prolog respond?

?- X=mia, X=vincent.

How will Prolog respond?

?- X=mia, X=vincent.

no

?-

Why? After working through the first goal, Prolog has instantiated X with **mia**, so that it cannot unify it with **vincent** anymore. Hence the second goal fails.

Example with complex terms

?- k(s(g),Y) = k(X,t(k)).

Task:

Unifiable? If yes, what are possible variable bindings after unification?

Example with complex terms

?- k(s(g),Y) = k(X,t(k)).

X=s(g)

Y=t(k)

yes

?-

Prolog and unification

- Prolog does not use a standard unification algorithm
- Consider the following query:

  ?- father(X) = X.

- Do these terms unify or not?

Infinite terms

?- father(X) = X.

X=father(father(father(father(father(father(father(father(father(father(father(
father(father(father(father(father(father(father(father(father(father(father(
father(father(father(father(father(father(father(father(father(father(father(
father(father(father(father(father(father(father(father(father(father(father(
father(father(father(father(father(father(father(father(father(father(father(
father(father(father(

Infinite terms

?- father(X) = X.

X=father(father(father(…))))

yes

?-

Occurs Check

- Prolog does, by default, not perform an occurs check
- Why does it make sense to leave out the occurs check?

Occurs Check

- Prolog does, by default, not perform an occurs check
- Why does it make sense to leave out the occurs check?
- Answer
  - For reasons of efficiency
  - When unifying a term t1 with a term t2, the time complexity is reduced from O(|t1|)+O(|t2|) to min(O(|t1|), O(|t2|)).

Occurs Check

- In Prolog, you can enforce the occurs check seperatly:

  ?- unify_with_occurs_check(father(X), X).
  no

Programming with Unification

```
vertical( line(point(X,Y),
               point(X,Z))).

horizontal( line(point(X,Y),
                 point(Z,Y))).
```

```
?- horizontal(line(point(2,3),Point)).
```

What will be the answer of Prolog?

Programming with Unification

```
vertical( line(point(X,Y),
               point(X,Z))).

horizontal( line(point(X,Y),
                 point(Z,Y))).
```

```
?- horizontal(line(point(2,3),Point)).

Point = point(_554,3);

no

?-
```

Prolog: Reasoning

Proof Search

- Now that we know about unification, we are in a position to learn how Prolog searches a knowledge base to see if a query is satisfied.
- In other words: we are ready to learn about proof search

Example

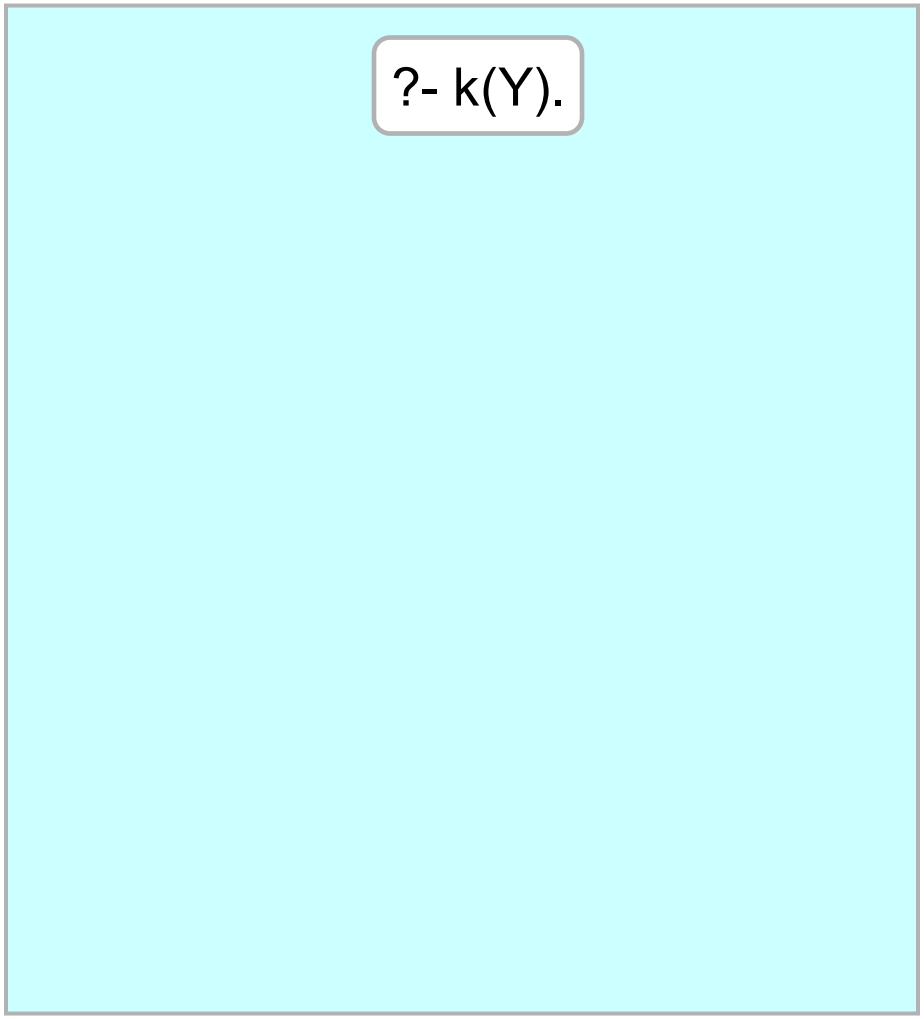
```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

```
?- k(Y).
```

Example: search tree

```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

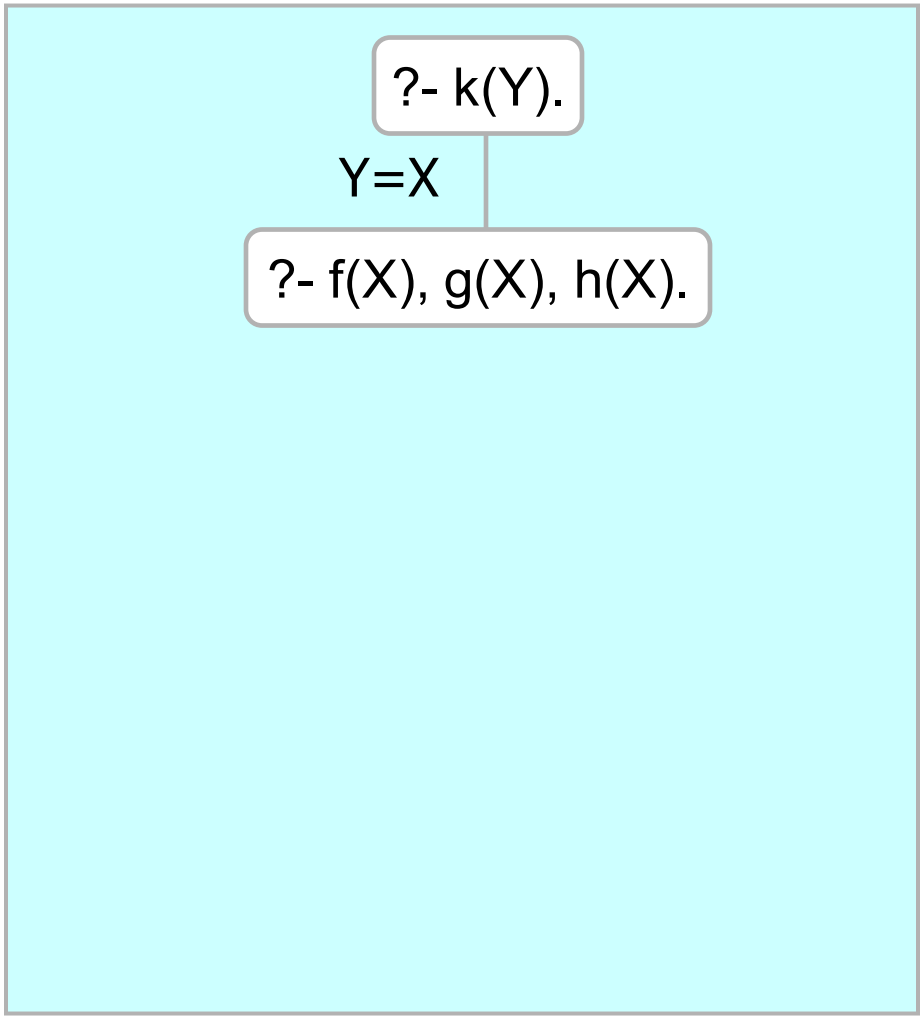
```
?- k(Y).
```



Example: search tree

```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

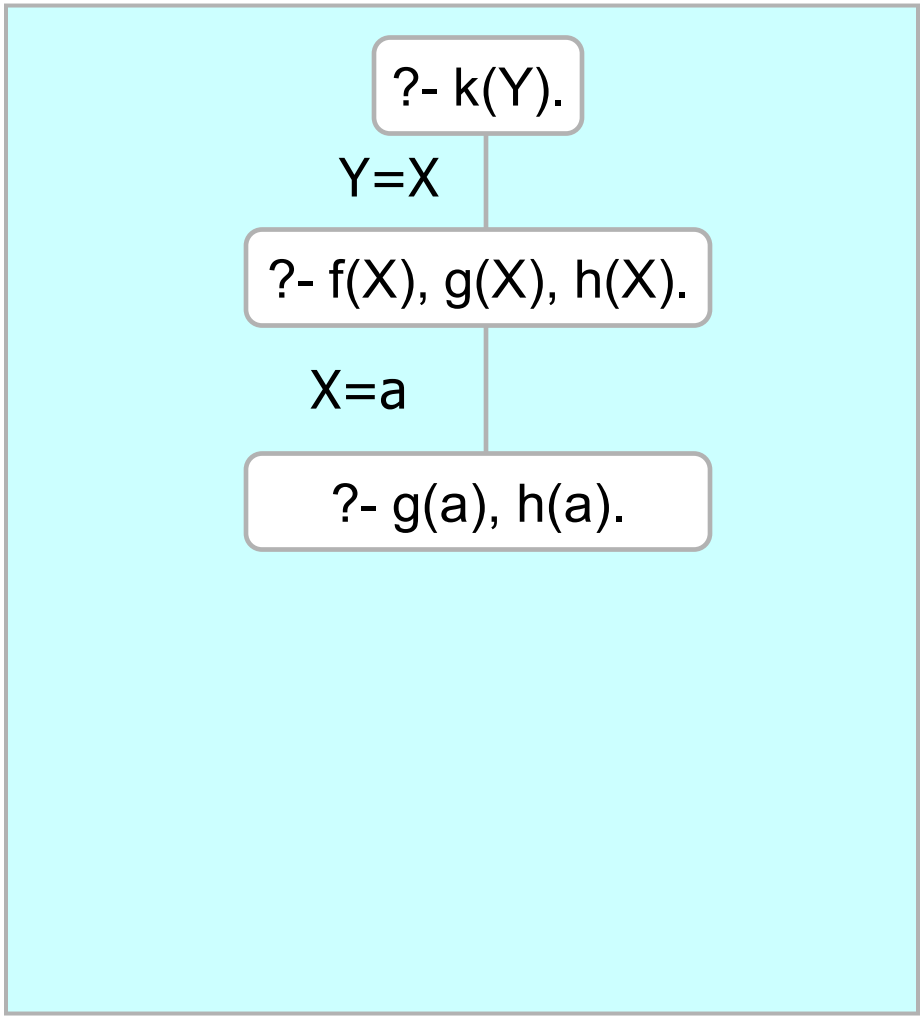
```
?- k(Y).
```



Example: search tree

```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

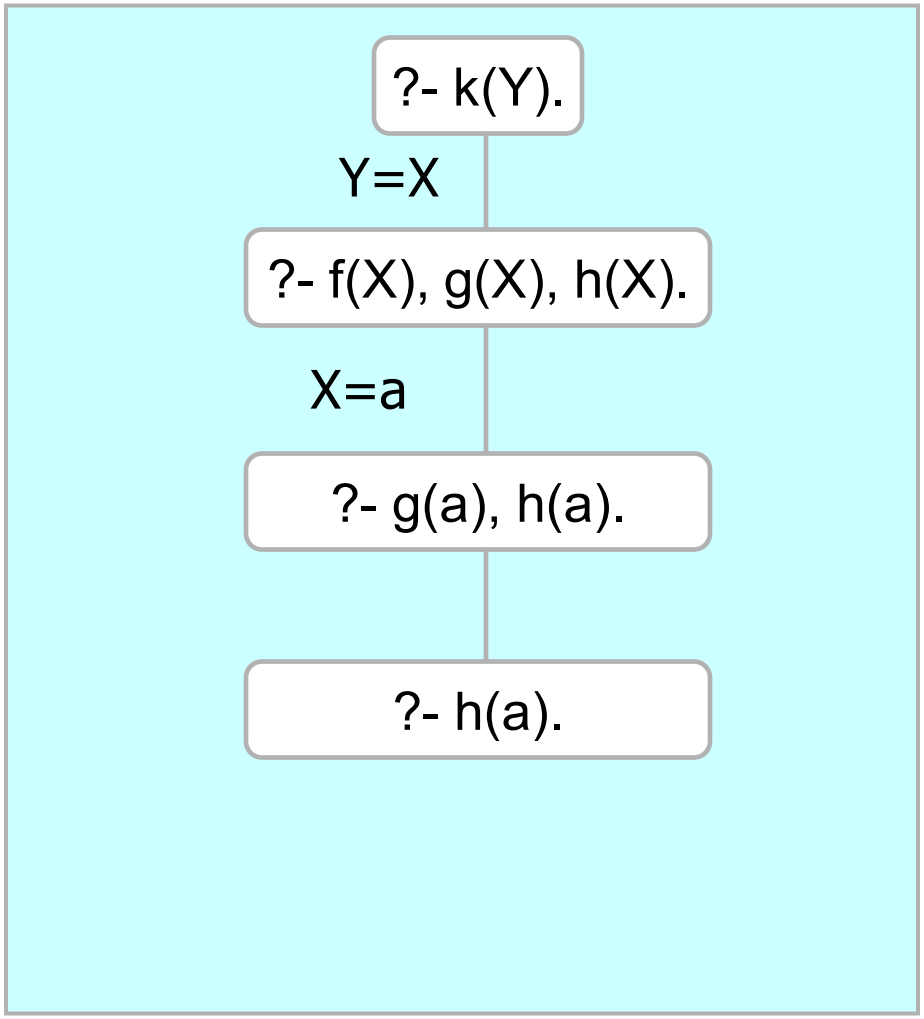
```
?- k(Y).
```



Example: search tree

```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

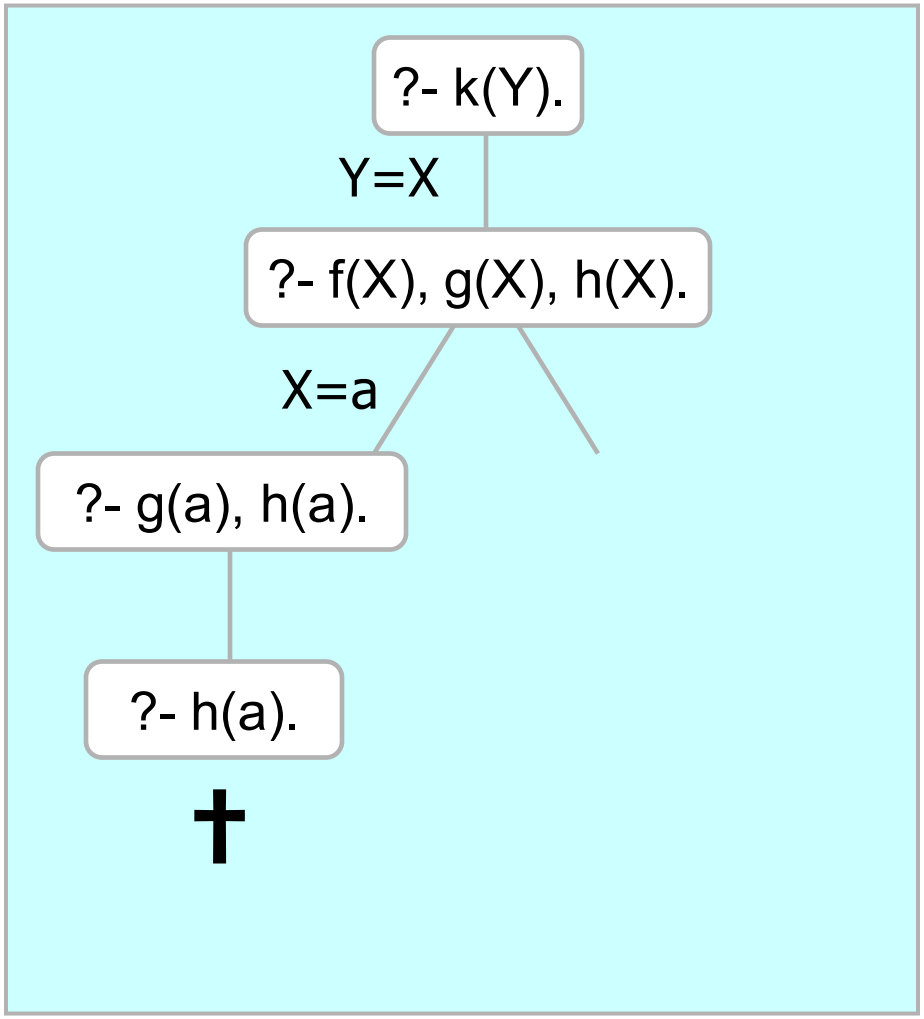
```
?- k(Y).
```



Example: search tree

```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

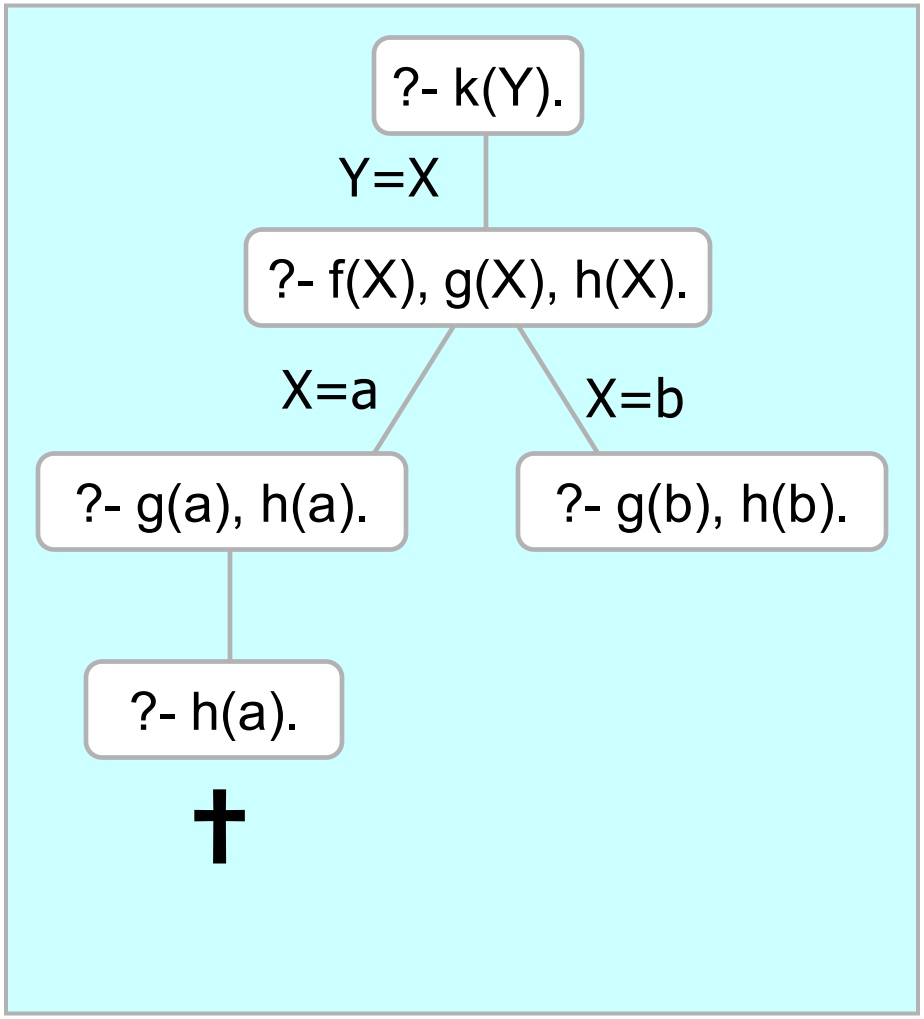
```
?- k(Y).
```



Example: search tree

```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

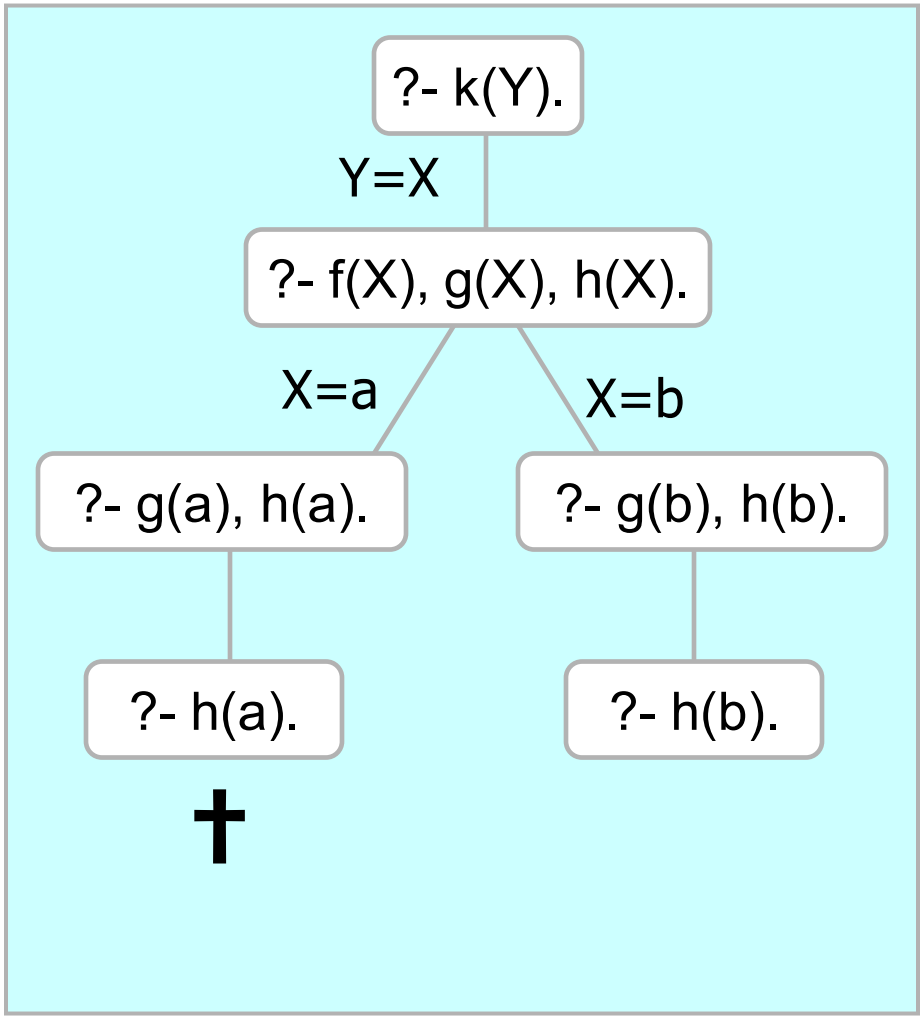
```
?- k(Y).
```



Example: search tree

```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

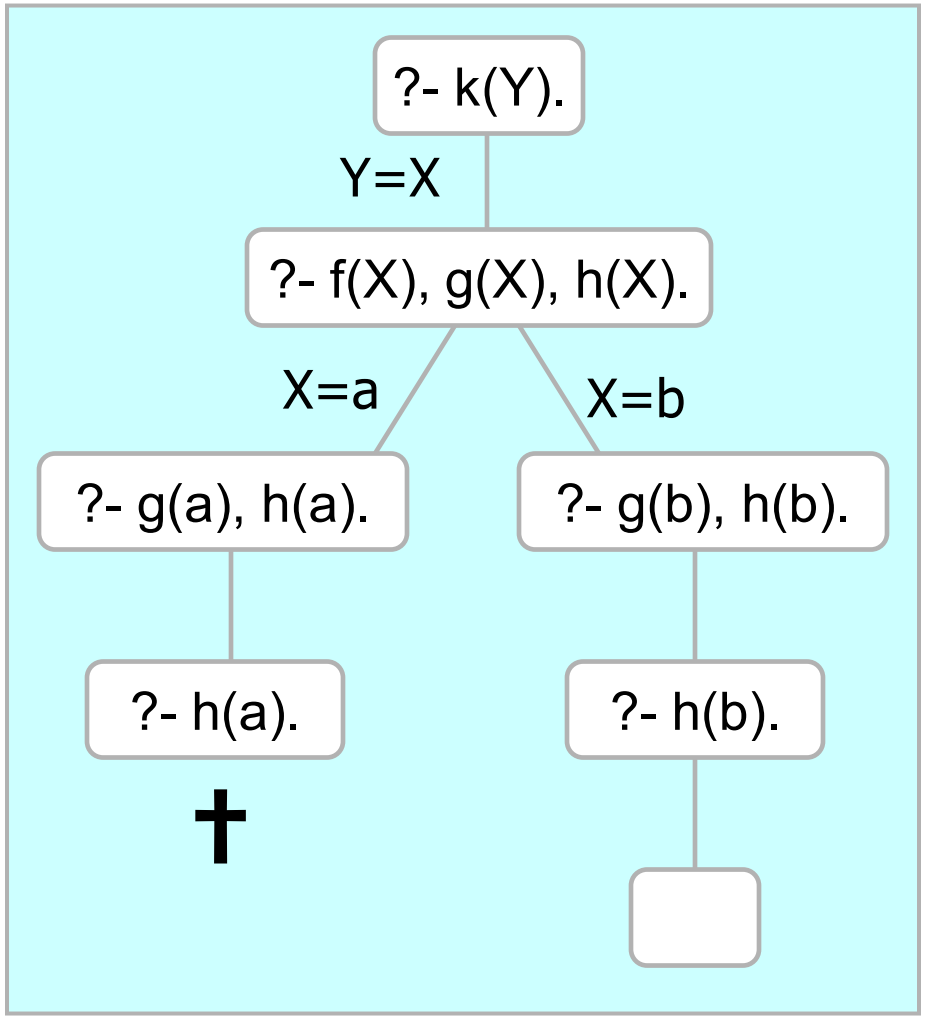
```
?- k(Y).
```



Example: search tree

```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

```
?- k(Y).
Y=b
```



Example: search tree

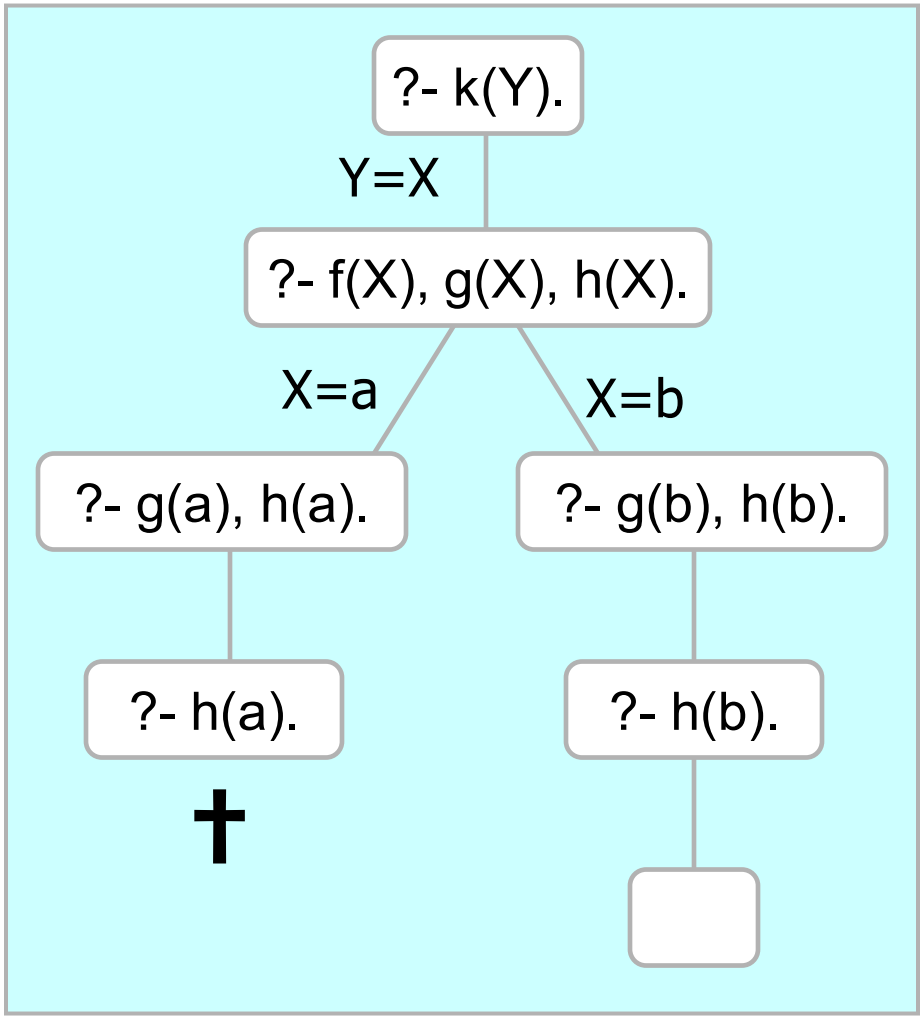
```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

```
?- k(Y).

Y=b;

no

?-
```



Another example

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

jealous(A,B):-
    loves(A,C),
    loves(B,C).
```

```
?- jealous(X,Y).
```

Another example

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

jealous(A,B):-
    loves(A,C),
    loves(B,C).
```

```
?- jealous(X,Y).
```

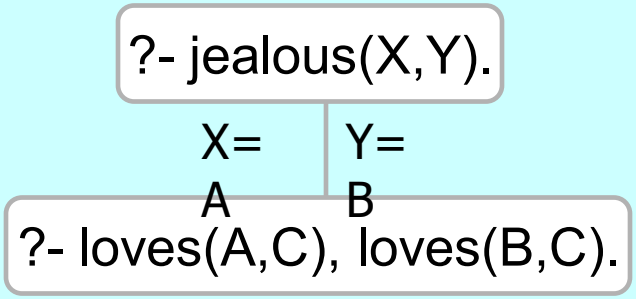
?- jealous(X,Y).

Another example

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

jealous(A,B):-
    loves(A,C),
    loves(B,C).
```

```
?- jealous(X,Y).
```

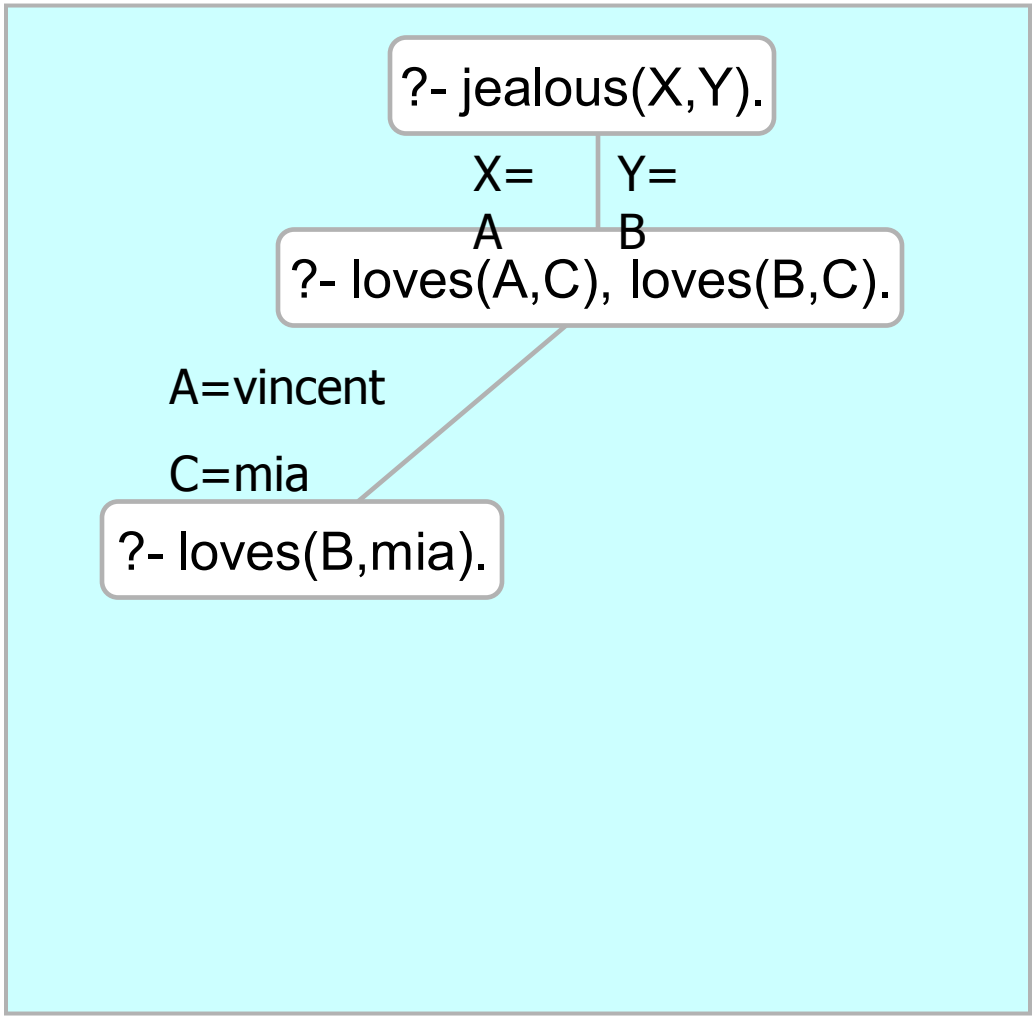


Another example

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

jealous(A,B):-
    loves(A,C),
    loves(B,C).
```

```
?- jealous(X,Y).
```



Another example

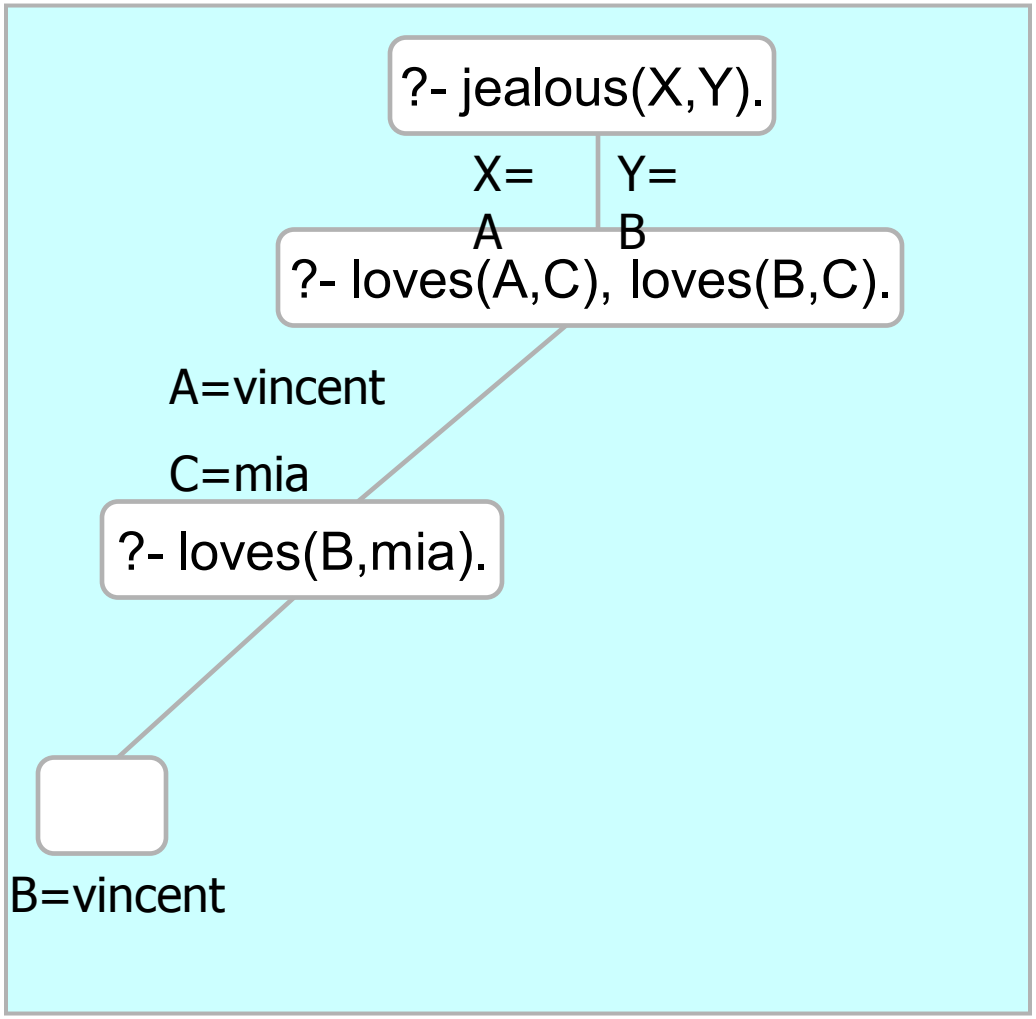
```
loves(vincent,mia).
loves(marsellus,mia).

jealous(A,B):-
    loves(A,C),
    loves(B,C).
```

```
?- jealous(X,Y).

X=vincent

Y=vincent
```

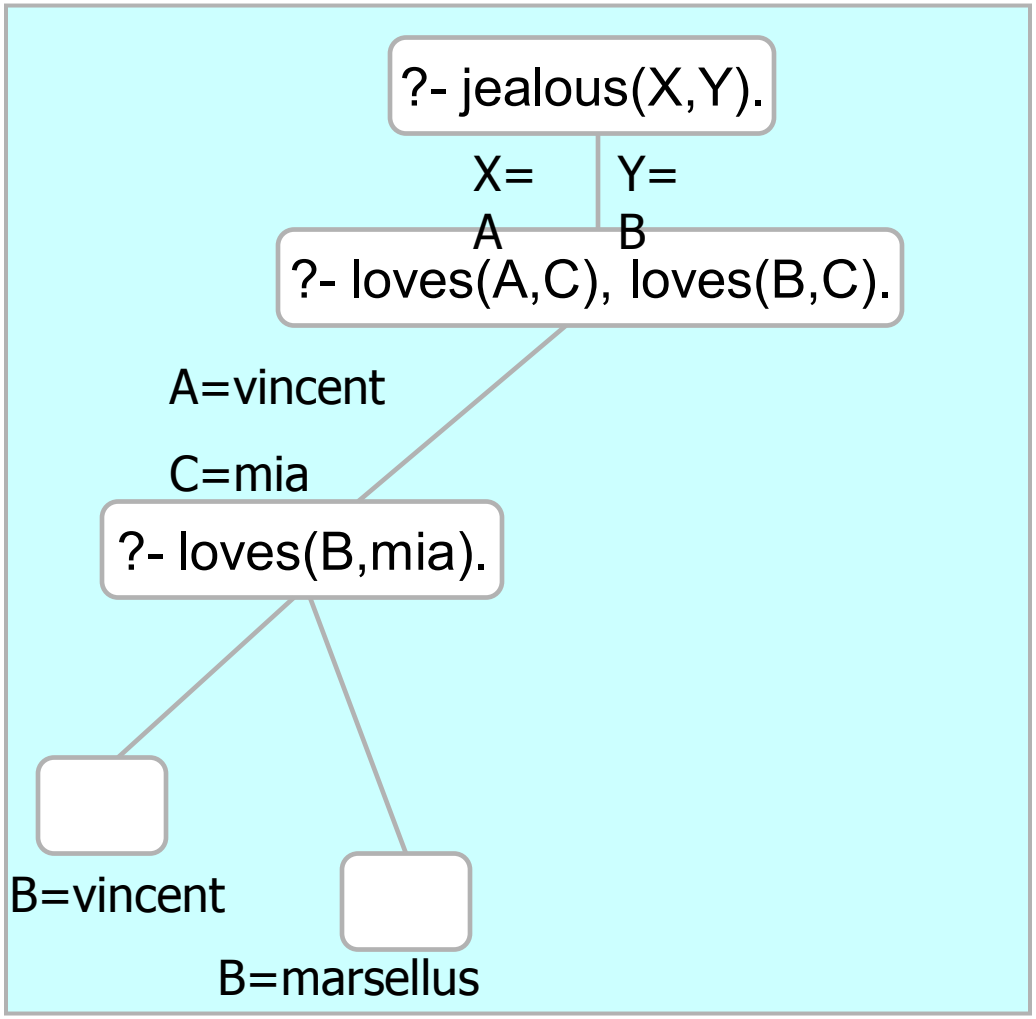


Another example

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

jealous(A,B):-
    loves(A,C),
    loves(B,C).
```

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

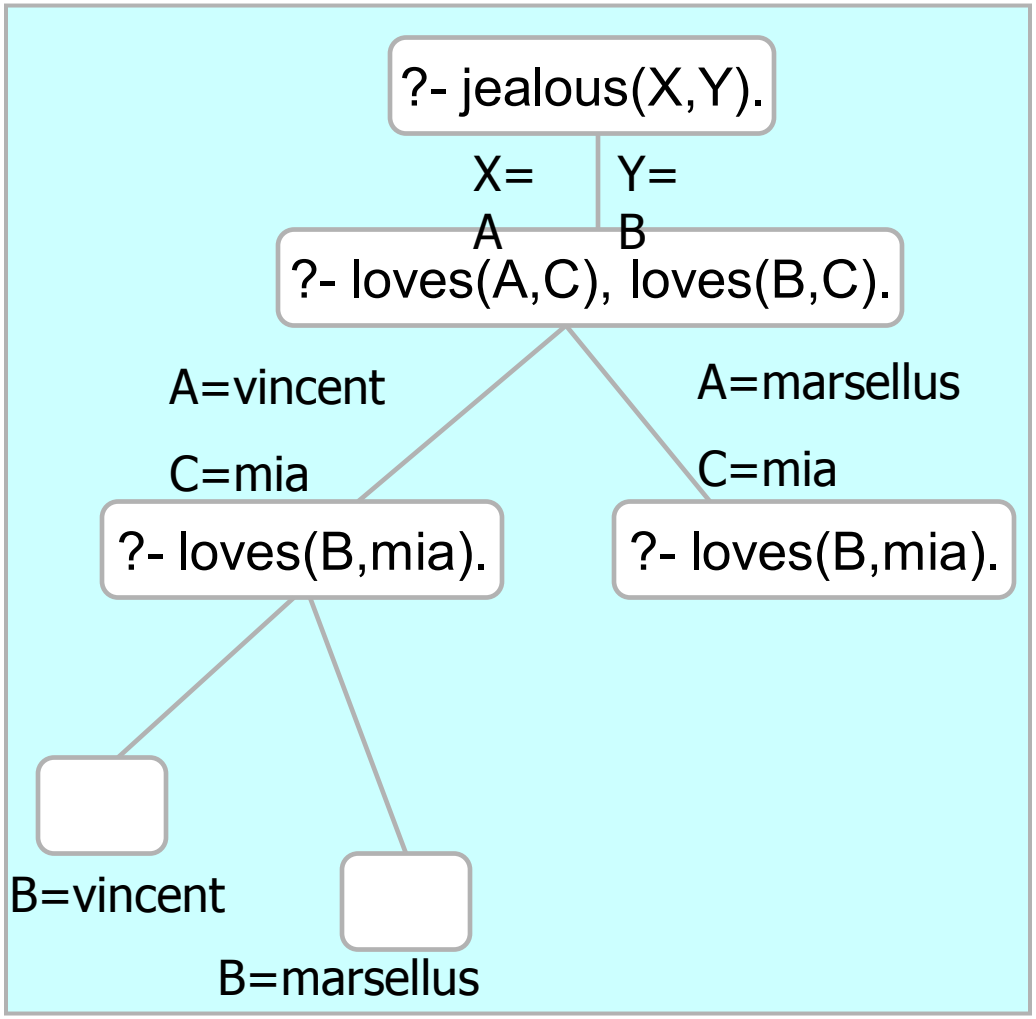


Another example

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

jealous(A,B):-
    loves(A,C),
    loves(B,C).
```

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



Another example

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

jealous(A,B):-
    loves(A,C),
    loves(B,C).
```

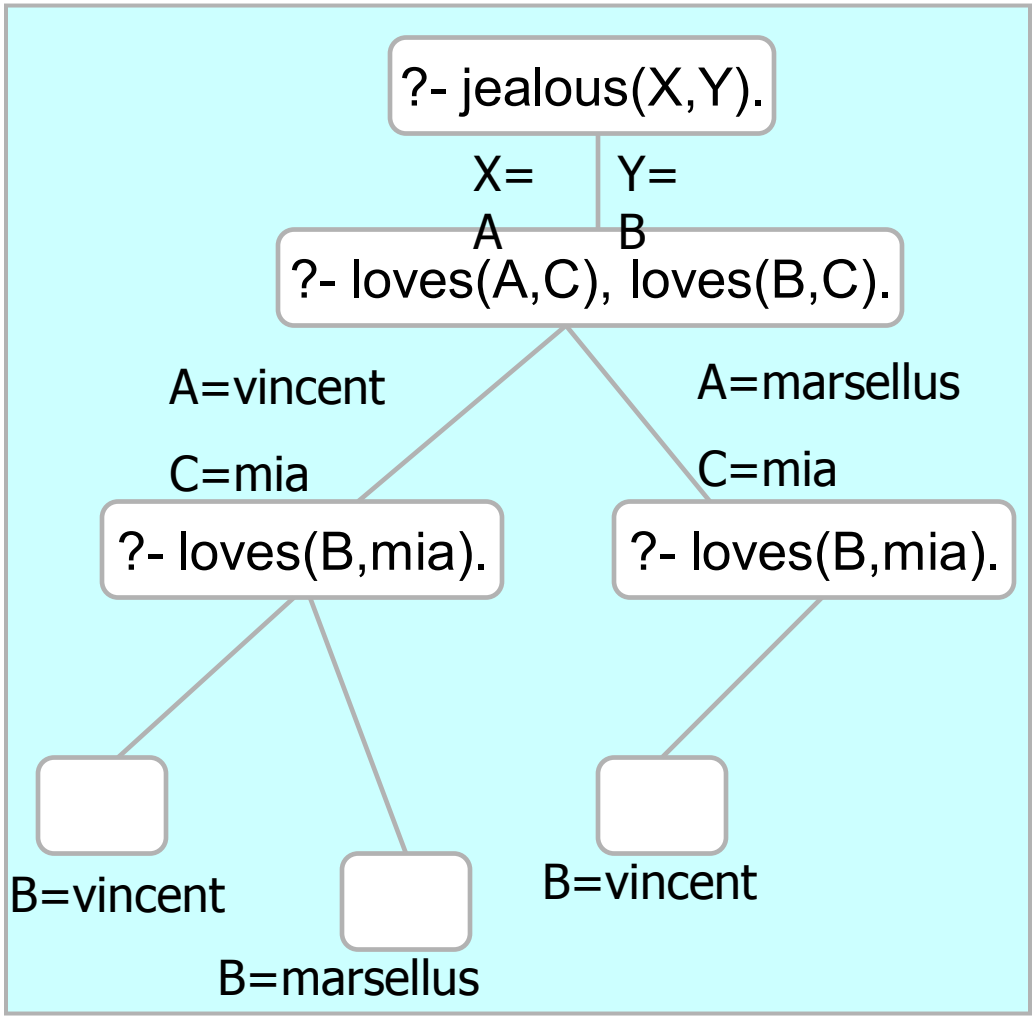
….

X=vincent

Y=marsellus;

X=marsellus

Y=vincent



Another example

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

jealous(A,B):-
    loves(A,C),
    loves(B,C).
```

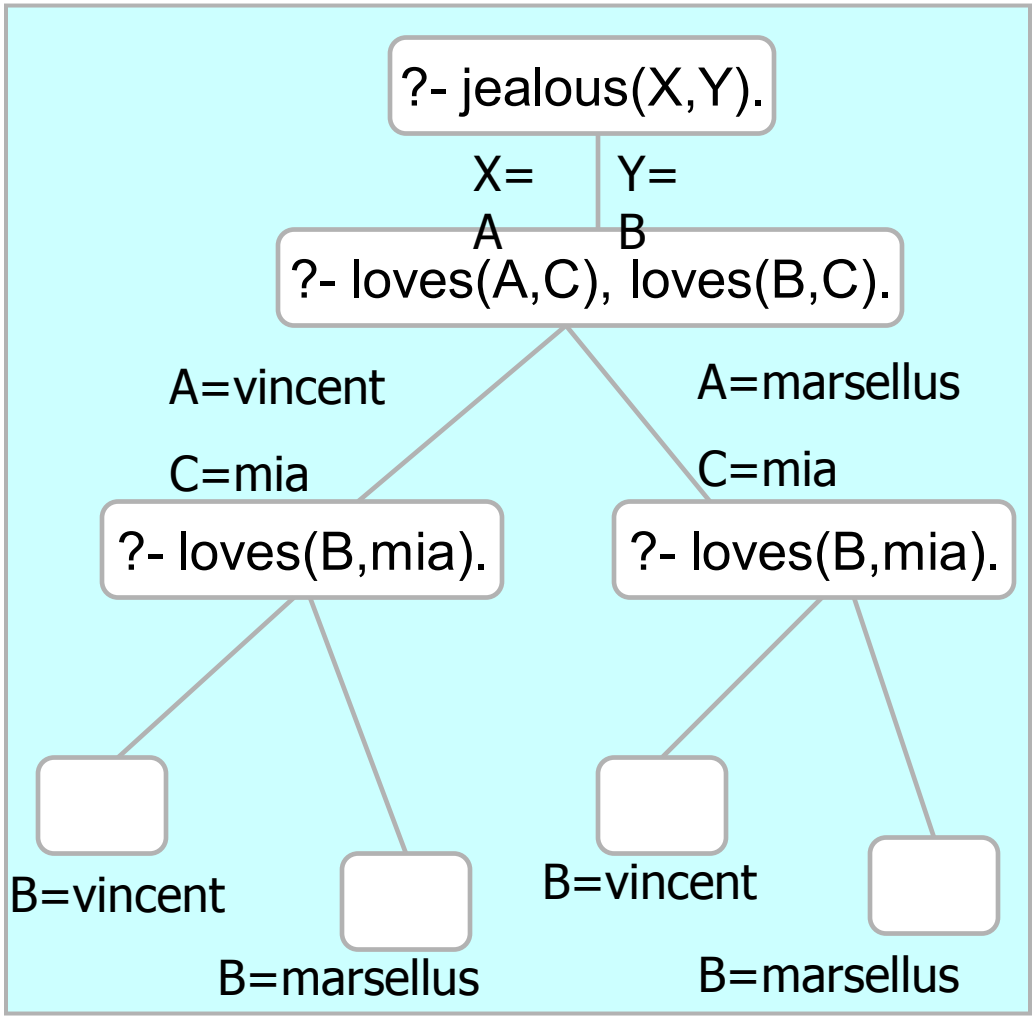
….

X=marsellus

Y=vincent;

X=marsellus

Y=marsellus

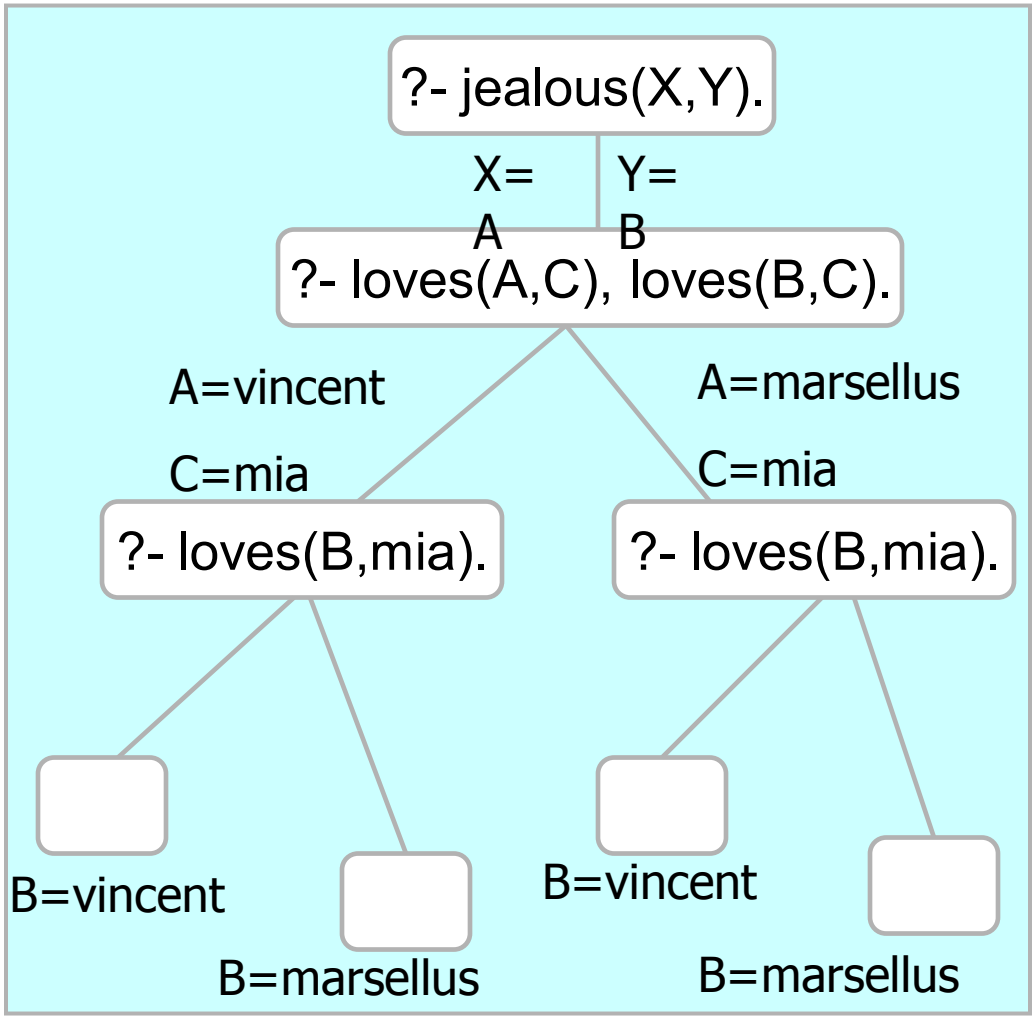


Another example

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

jealous(A,B):-
    loves(A,C),
    loves(B,C).
```

```
….
X=marsellus
Y=vincent;
X=marsellus
Y=marsellus;
no
```



Exercise

Task: Solve the puzzle "Tower of Hanoi" using Prolog.

For three disks, the output of your program should similar to:

?- hanoi(3,left,middle,right).

Move top disk from left to middle

Move top disk from left to right

Move top disk from middle to right

Move top disk from left to middle

Move top disk from right to left

Move top disk from right to middle

Move top disk from left to middle

Prolog: Collecting solutions

Consider this database

```
child(martha,charlotte).
child(charlotte,caroline).
child(caroline,laura).
child(laura,rose).

descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z),
               descend(Z,Y).
```

```
?- descend(martha,X).

X=charlotte;

X=caroline;

X=laura;

X=rose;

no
```

Collecting solutions

- There may be many solutions to a Prolog query
- However, Prolog generates solutions one by one
- Sometimes we would like to have *all* the solutions to a query in one go

=>

- Prolog has three built-in predicates that do this: **findall**/3, **bagof**/3 and **setof**/3
- In essence, all these predicates collect all the solutions to a query and put them into a single list
- But there are important differences between them

findall/3

- The query

```
?- findall(O,G,L).
```

  produces a list **L** of all the objects **O** that satisfy the goal **G**
  - Always succeeds
  - Unifies **L** with empty list if **G** cannot be satisfied

A findall/3 example

```
child(martha,charlotte).
child(charlotte,caroline).
child(caroline,laura).
child(laura,rose).

descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z),
               descend(Z,Y).
```

```
?- findall(X,descend(martha,X),L).
L=[charlotte,caroline,laura,rose]
yes
```

Other findall/3 examples

```
child(martha,charlotte).
child(charlotte,caroline).
child(caroline,laura).
child(laura,rose).

descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z),
               descend(Z,Y).
```

?- findall(X,descend(rose,X),L).

What is the output of Prolog?

Other findall/3 examples

```
child(martha,charlotte).
child(charlotte,caroline).
child(caroline,laura).
child(laura,rose).

descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z),
               descend(Z,Y).
```

```
?- findall(X,descend(rose,X),L).
L=[ ]
yes
```

findall/3 is sometimes rather crude

```
child(martha,charlotte).
child(charlotte,caroline).
child(caroline,laura).
child(laura,rose).

descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z),
               descend(Z,Y).
```

```
?- findall(Chi,descend(Mot,Chi),L).
```

What is the output of Prolog?

findall/3 is sometimes rather crude

```
child(martha,charlotte).
child(charlotte,caroline).
child(caroline,laura).
child(laura,rose).

descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z),
               descend(Z,Y).
```

```
?- findall(Chi,descend(Mot,Chi),L).

L=[charlotte,caroline,laura, rose,
    caroline,laura,rose,laura,rose,rose]

yes
```

bagof/3

- The query

```
?- bagof(O,G,L).
```

  produces a list **L** of all the objects **O** that satisfy the goal **G**
  - Only succeeds if the goal **G** succeeds
  - Binds free variables in **G**

Using bagof/3

```
child(martha,charlotte).
child(charlotte,caroline).
child(caroline,laura).
child(laura,rose).

descend(X,Y):-
    child(X,Y).
descend(X,Y):-
    child(X,Z),
    descend(Z,Y).
```

```
?- bagof(Chi,descend(Mot,Chi),L).

Mot=caroline

L=[laura, rose];

Mot=charlotte

L=[caroline,laura,rose];

Mot=laura

L=[rose];

Mot=martha

L=[charlotte,caroline,laura,rose];

no
```

Prolog: What's left?

Prolog in programming languages

- Using Prolog from Java
  - http://www.gnu.org/software/gnuprologjava/
- C++-Interface for SWI-Prolog
  - http://www.swi-prolog.org/pldoc/package/pl2cpp.html
- Python interface for SWI-Prolog
  - http://code.google.com/p/pyswip/
- Prolog-interpreter written in Scala
  - http://code.google.com/p/styla/

Prolog

- Prolog is much, much more, than what we have seen
  - Lists
  - Negation as failure
  - I/O
  - Cut operator
- A nice manual can be found here:
  - http://www.csupomona.edu/~jrfisher/www/prolog_tutorial/contents.html
- During the lecture, we will look at these things, only whenever it becomes necessary

Datalog?

- Datalog=Prolog minus non-terminating queries ☺
  - Datalog does not allow function symbols
  - Datalog does not allow negation (not or … \+)

Acknowledgements

- Prolog material based on introduction at
  - http://www.learnprolognow.org/lpnpage.php?pageid=teaching
  - © Patrick Blackburn, Johan Bos & Kristina Striegnitz