UNIT

Introduction to Prolog

Lecture 2

- **Emma is a Doberman pinscher and a good dog.**

gooddog(emma) ∧ isa(emma, doberman)

gooddog(emma) , isa(emma, doberman).

- **If wishes were horses, beggars would ride.**

equal(wishes, horses)→ride(beggars)

ride(beggars) :- equal(wishes, horses).

- **If Ali is happy or listens to music then he is playing guitar**

happy(Ali) ∨ listens_music(Ali) → Plays guitar (Ali)

plays_guitar(Ali):- happy(Ali); listens_music(Ali).

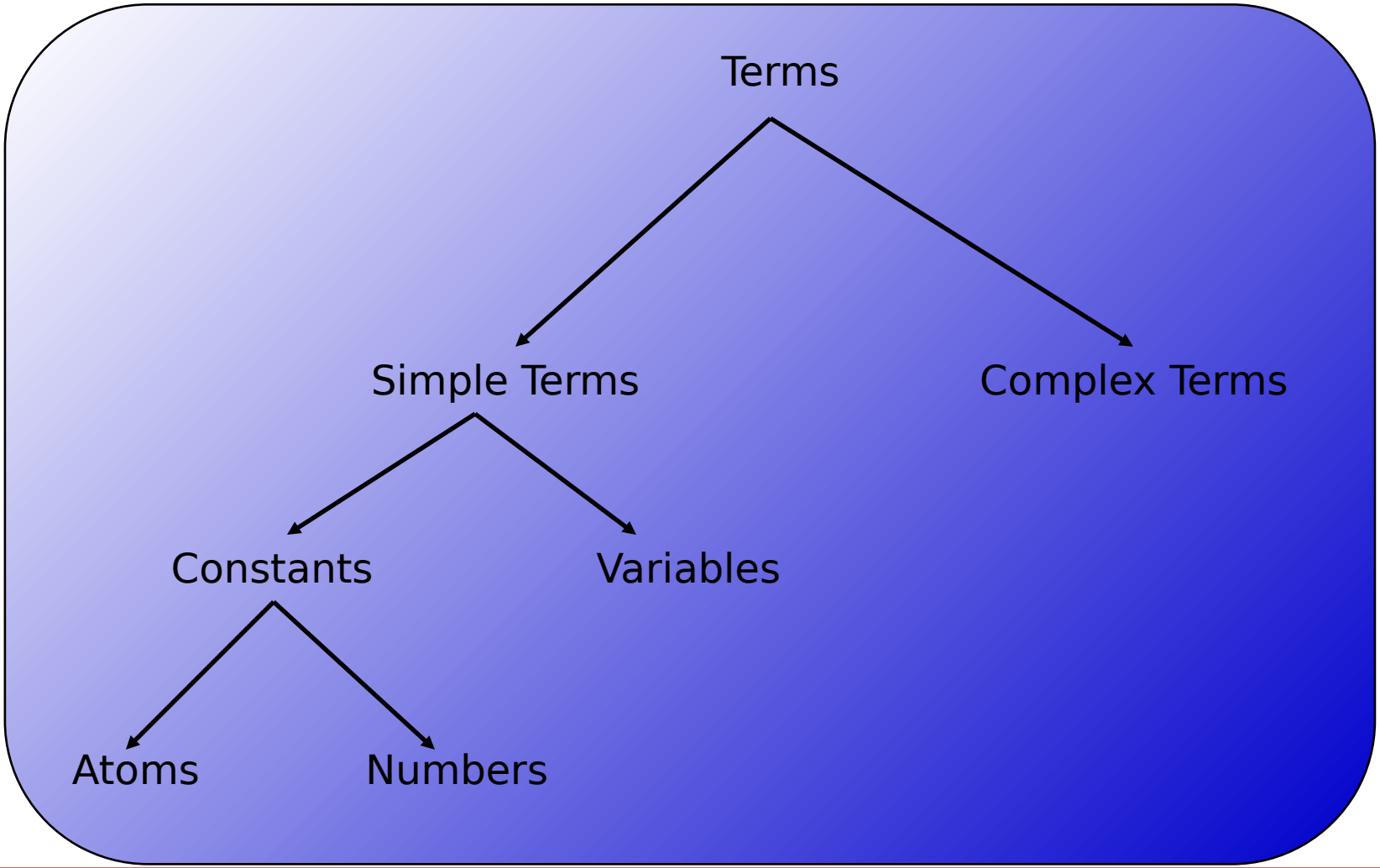
Aim of this lecture 2

- Discuss unification in Prolog
  - Show how Prolog unification differs from standard unification

- Explain the search strategy that Prolog uses when it tries to deduce new information from old, using modus ponens

$$p$$
$$p \rightarrow q$$
$$\therefore q$$

Recall Prolog Terms



Unification

- Recall previous example, where we said that Prolog unifies

  woman(X)

  with

  woman(mia)

  thereby instantiating the variable X with the atom mia.

- 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:
  - mia 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. 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
?- mia = vincent.
no
?-
?- 'mia' = mia.
Yes
?- 2='2'.
no
?- X = _5071
 Y = _5071
 yes
```

Prolog unification: =/2

```
?- mia = X.

X=mia

yes

?-
```

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

Example with complex terms


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

One last example


```
?- loves(X,X) = loves(marsellus,mia).
no
```

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(fa
    ther(father(father(father(father(father(father(father(father(father(father(fath
    er(father(father(father(father(father(father(father(father(father(father
    (father(father(father(father(father(father(father(father(father(father(f
    ather(father(father(father(father(father(father(father(father(father(fat
    her(father(father(
```

```
?- father(X) = X.
X=father(father(father(...)))
yes
?-
```

Occurs Check

- A standard unification algorithm carries out an occurs check
- If it is asked to unify a variable with another term it checks whether the variable occurs in the term
- In Prolog:

```
?- 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))).
```

Programming with Unification


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

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

```
?-
```

Programming with Unification


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

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

```
?- vertical(line(point(1,1),point(1,3))).
yes
?-
```

Programming with Unification


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

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

```
?- vertical(line(point(1,1),point(1,3))).
yes
?- vertical(line(point(1,1),point(3,2))).
no
?-
```

Programming with Unification


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

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

```
?- horizontal(line(point(1,1),point(1,Y))).
Y = 1;
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)).
Point = point(_554,3);
no
?-
```

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).
```

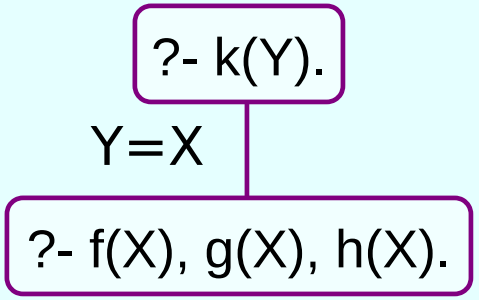
?- k(Y).

?- k(Y).

Example: search tree

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

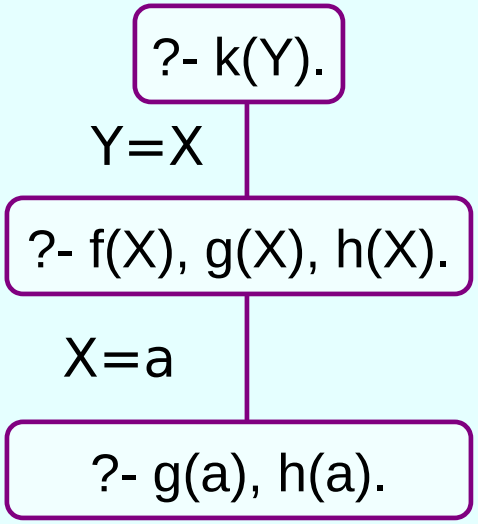
?- k(Y).



Example: search tree

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

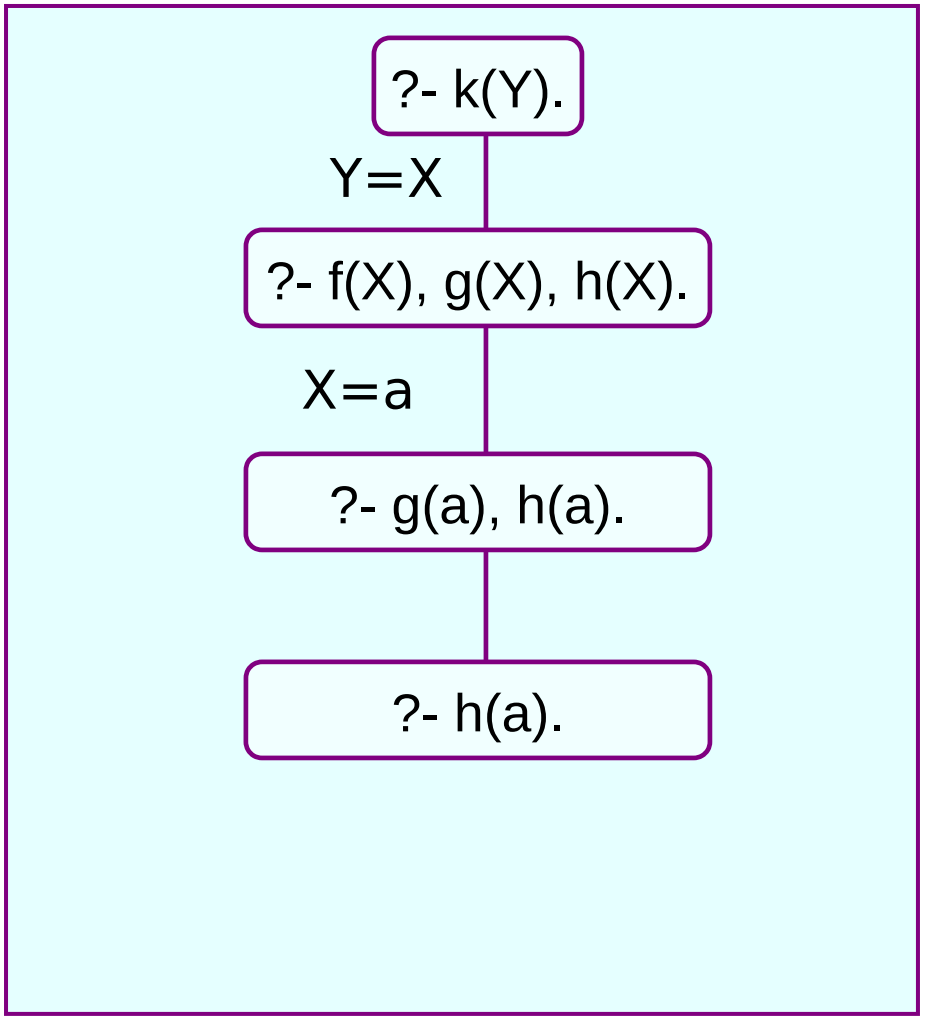
?- k(Y).



Example: search tree

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

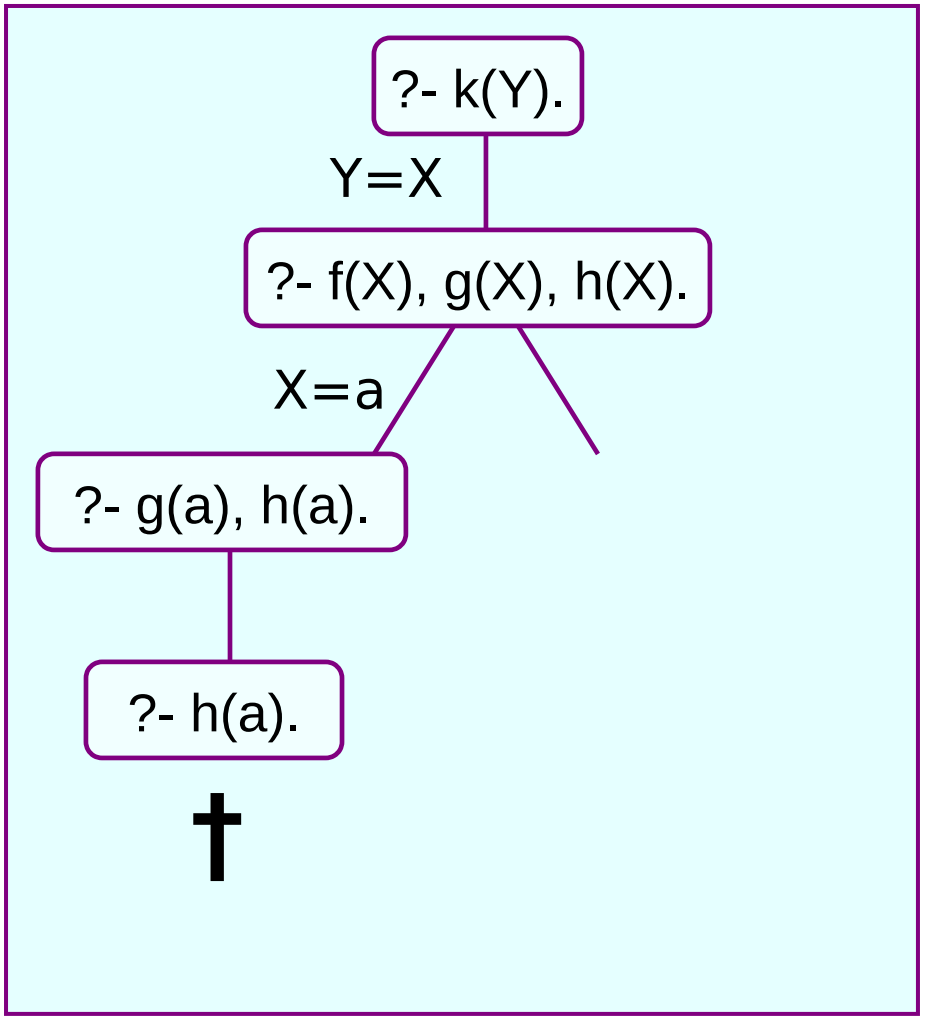
?- k(Y).



Example: search tree

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

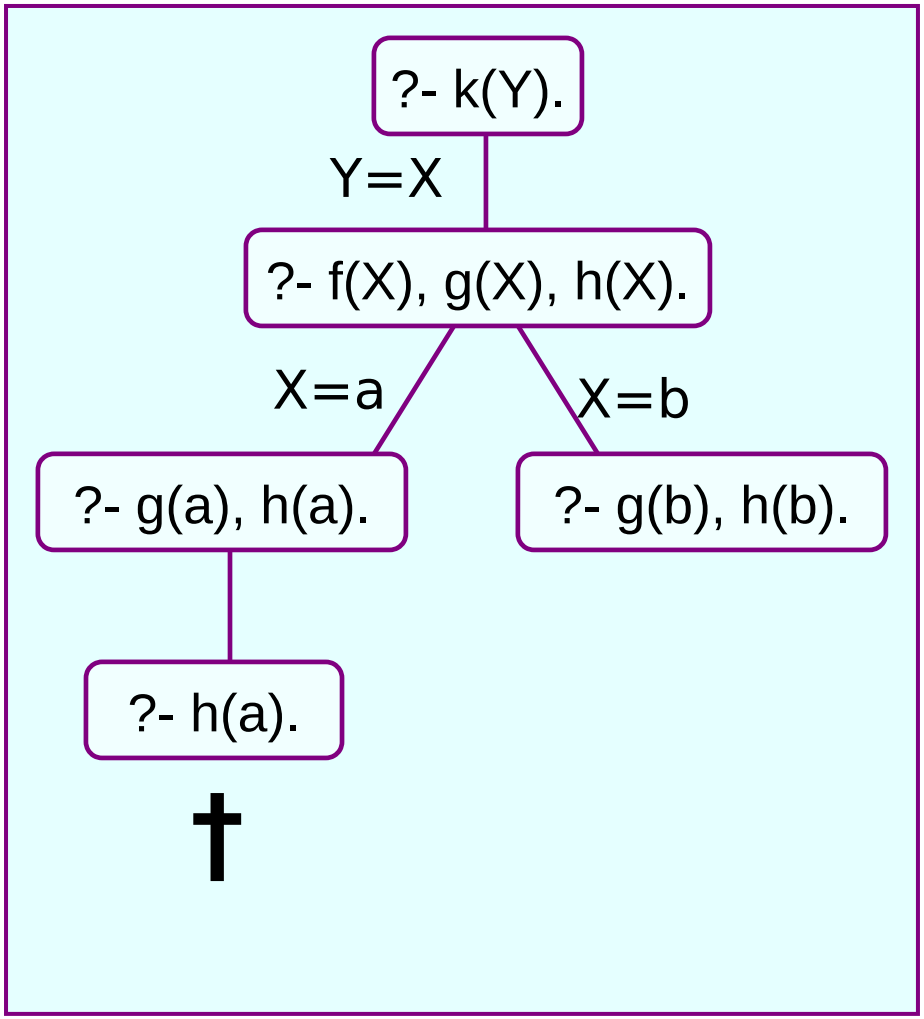
?- k(Y).



Example: search tree


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

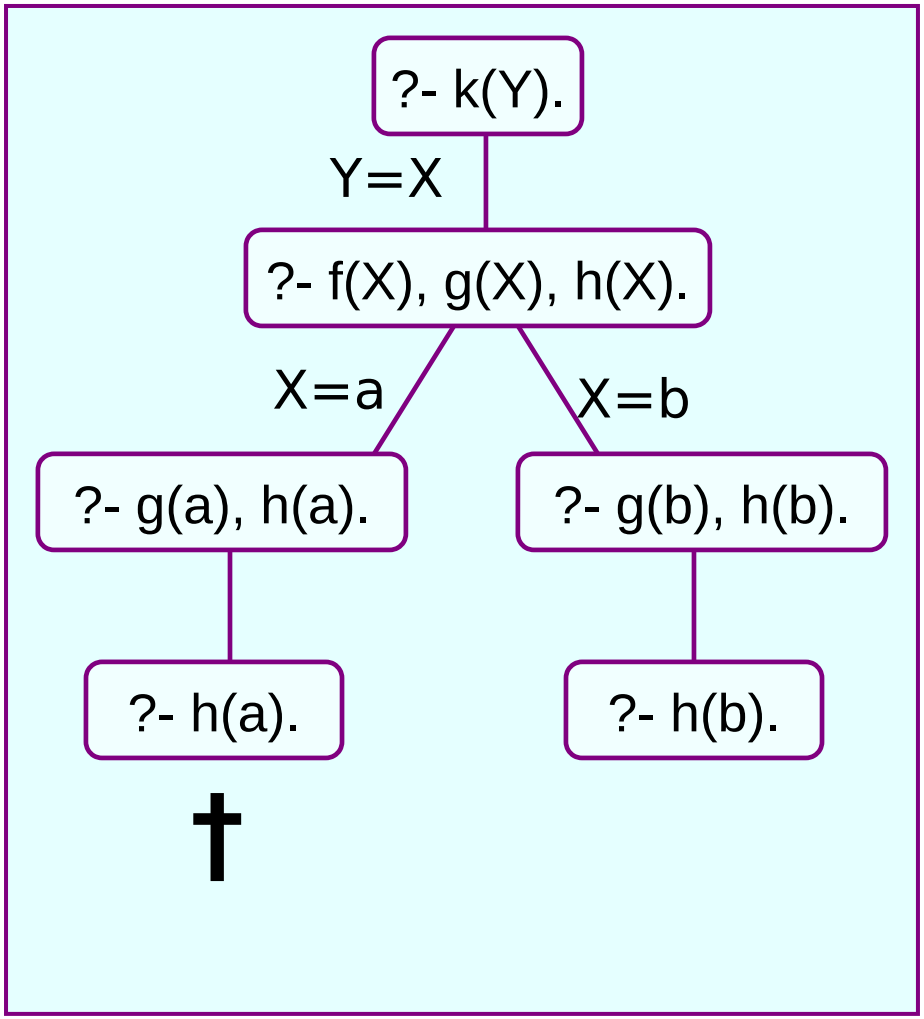
?- k(Y).



Example: search tree

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

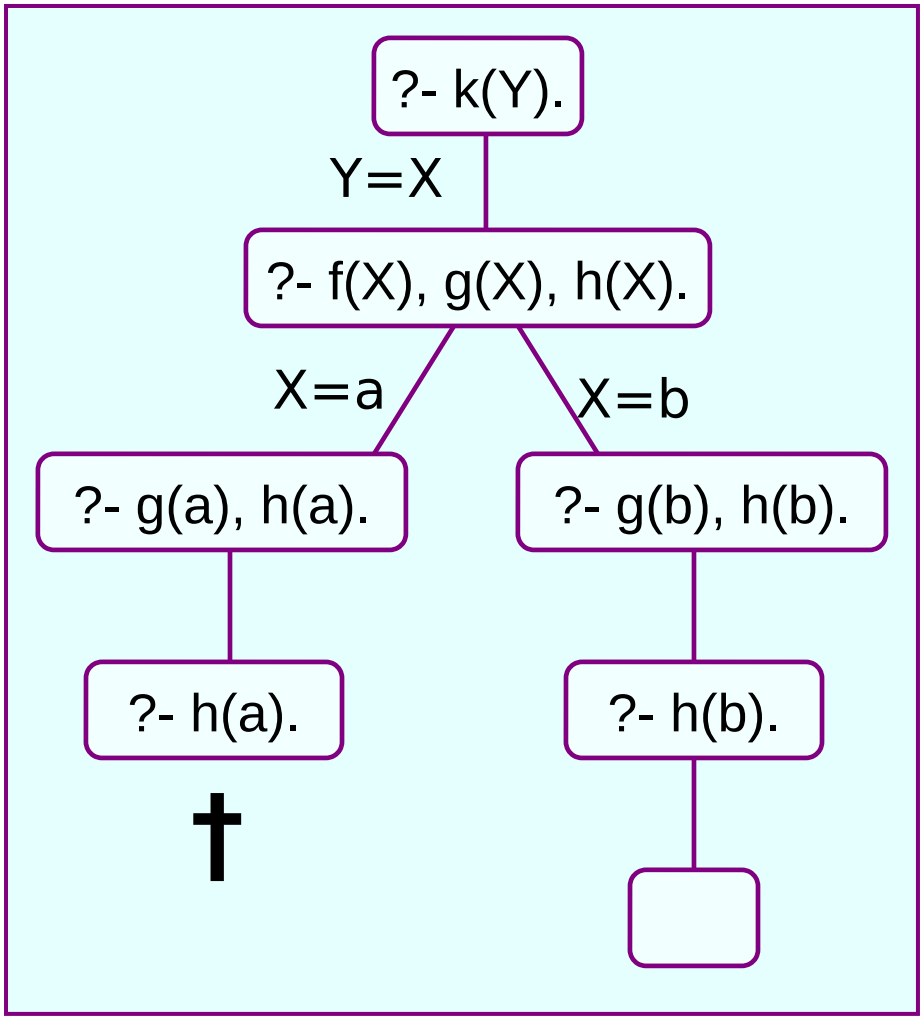
?- k(Y).



Example: search tree

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

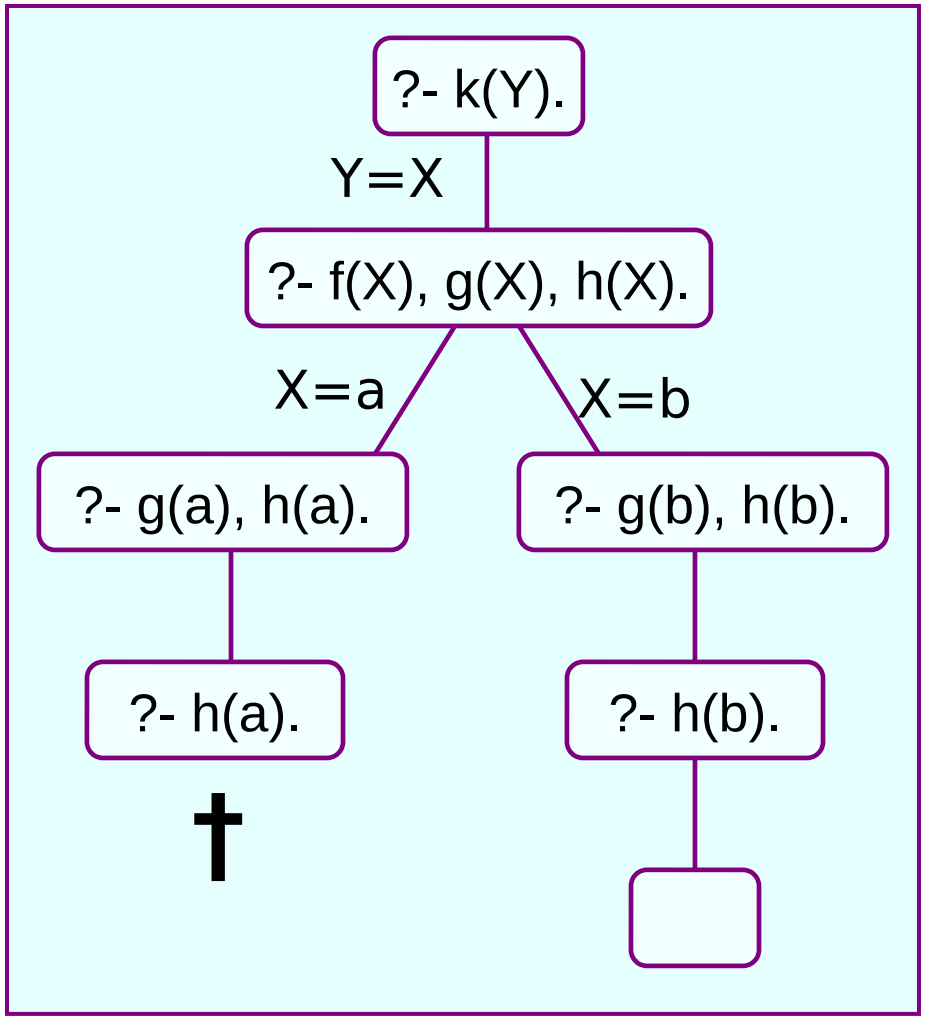
```
?- k(Y).
Y=b
```



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).
```

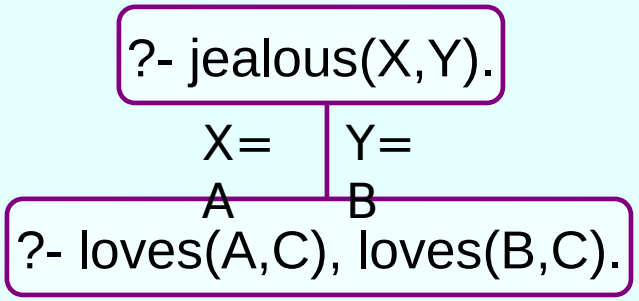
```
?- jealous(X,Y).
```

Another example

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

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

?- jealous(X,Y).

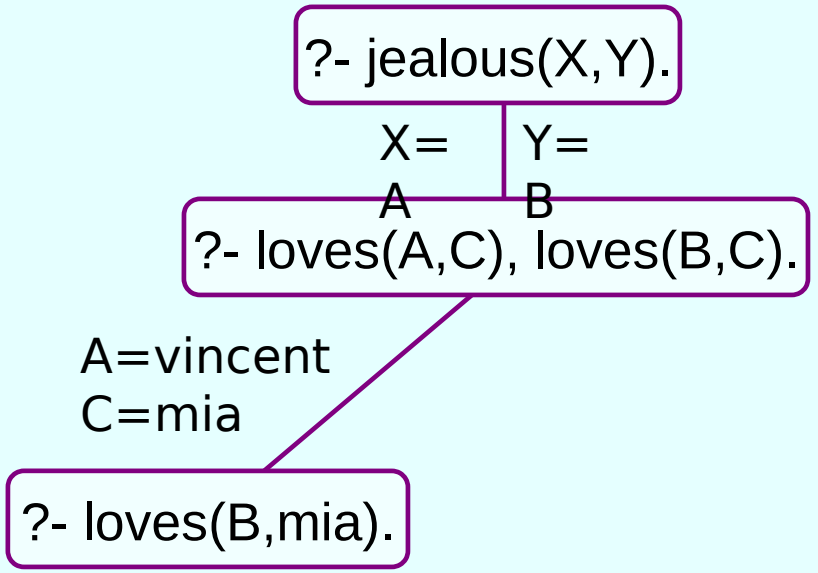


Another example

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

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

?- jealous(X,Y).

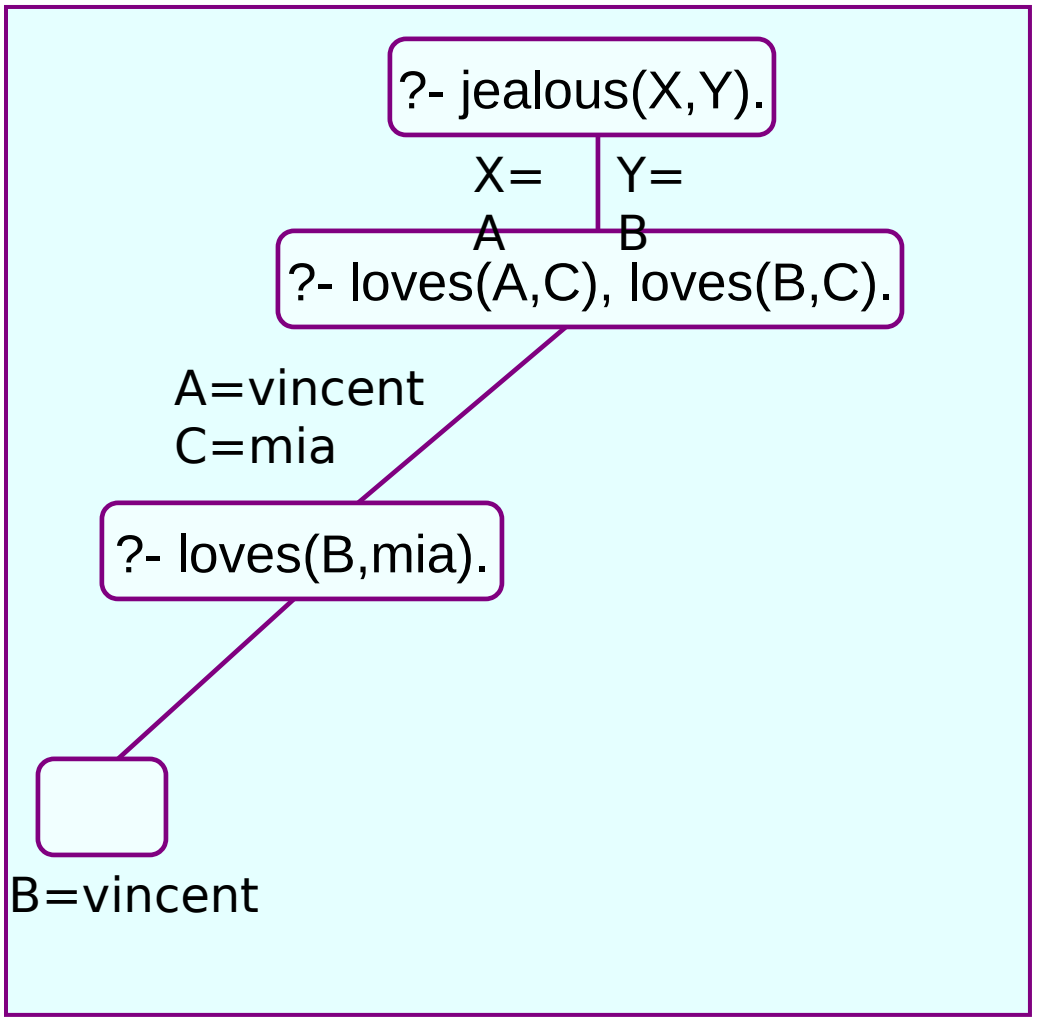


Another example

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

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

?- jealous(X,Y).
X=vincent
Y=vincent

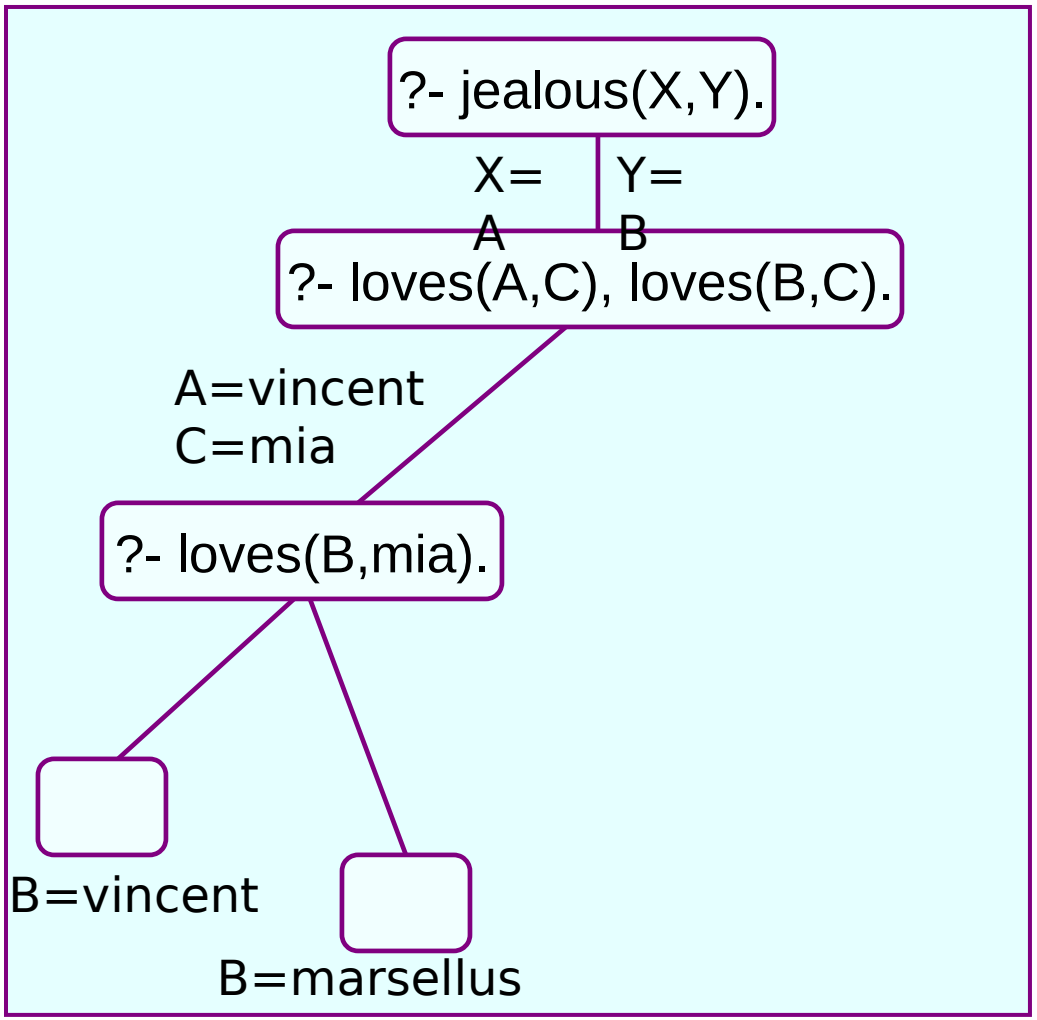


Another example


```
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

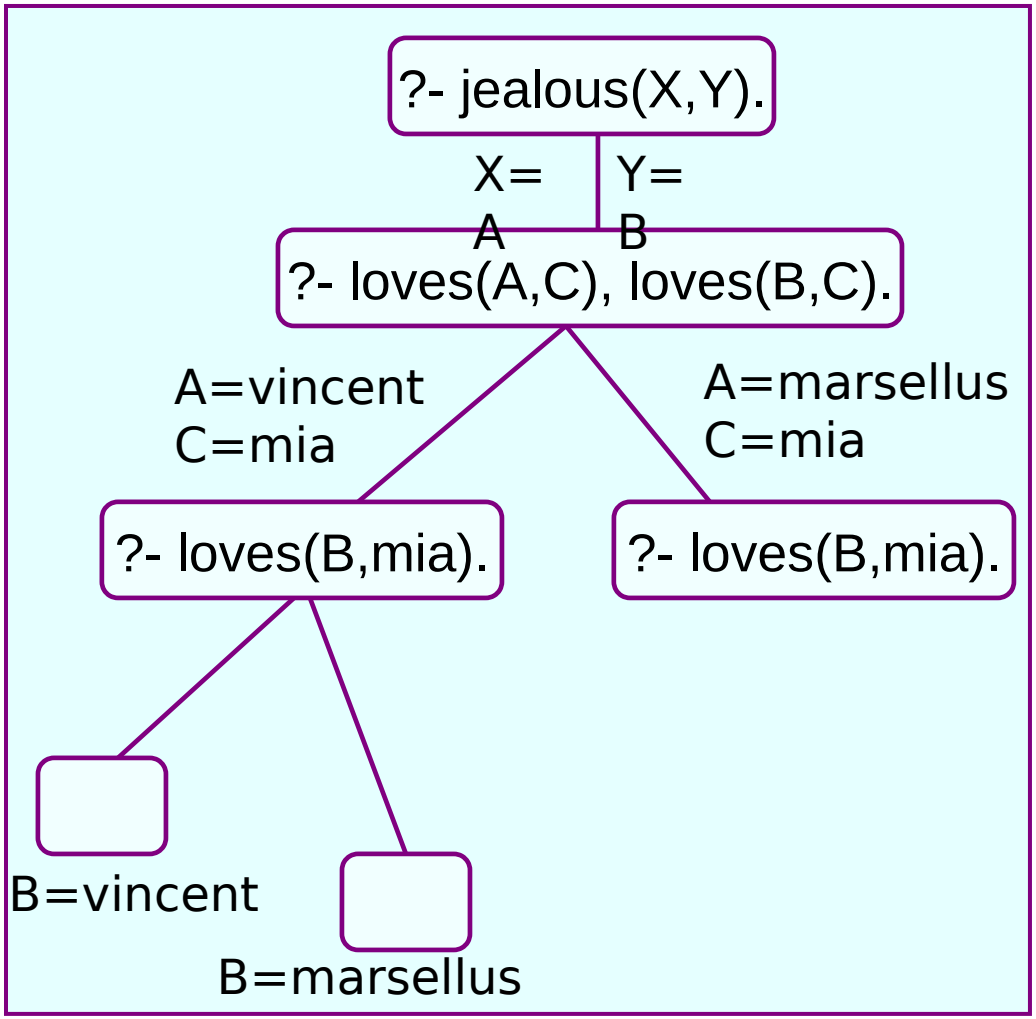


Another example


```
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;

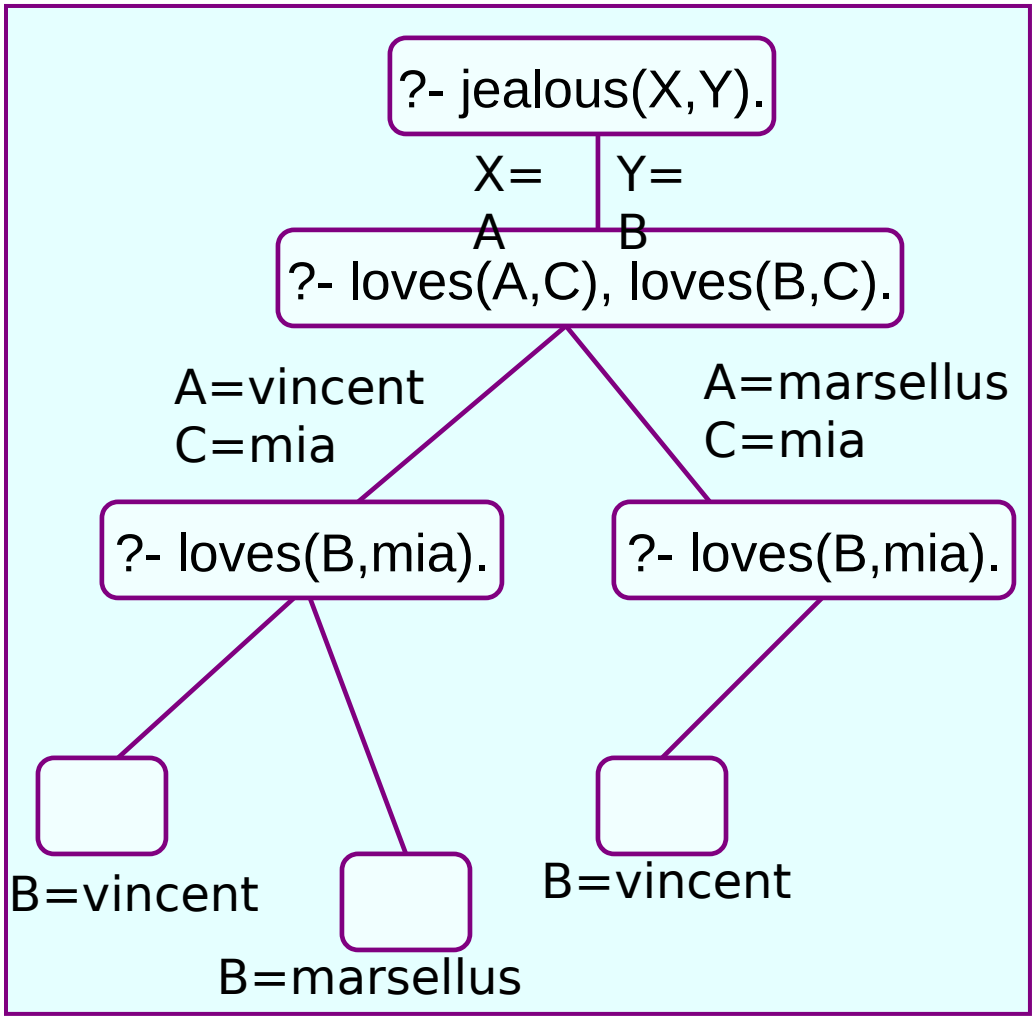


Another example


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

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

….
X=vincent
Y=marsellus;
X=marsellus
Y=vincent

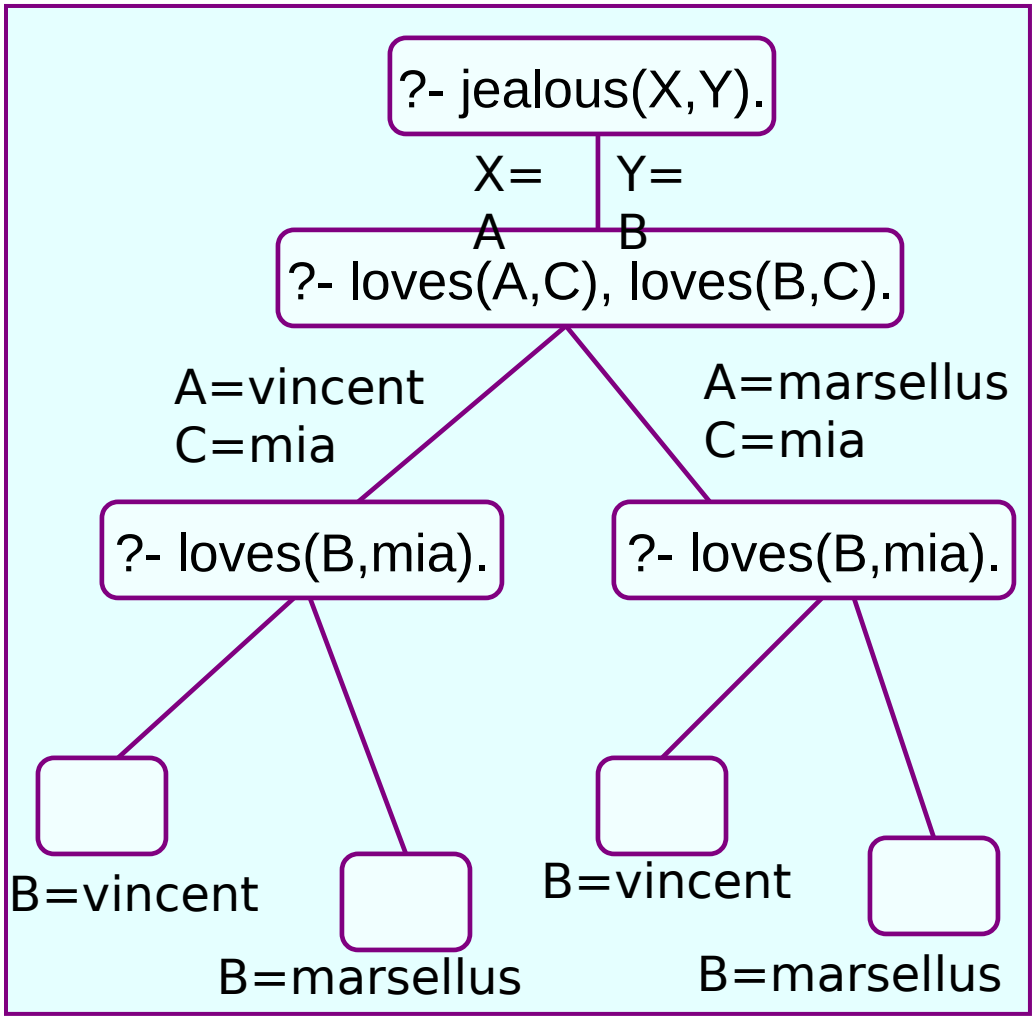


Another example


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

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

….
X=marsellus
Y=vincent;
X=marsellus
Y=marsellus

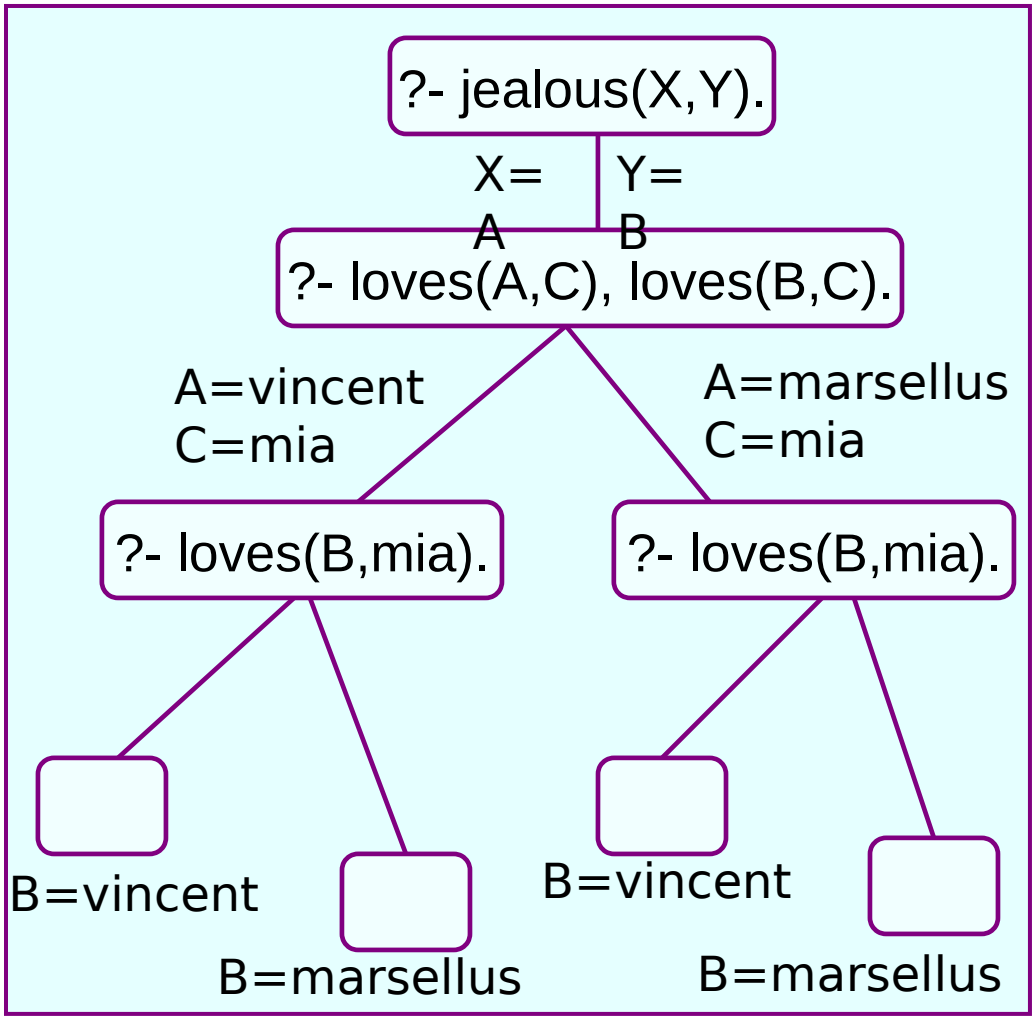


Another example


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

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

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



Exercises Chapter 2

Learn Prolog Now! Chapter 2

by Patrick Blackburn, Johan Bos, and Kristina Striegnitz

[next] [prev] [prev-tail] [tail] [up]

Chapter 2
Unification and Proof Search

This chapter has two main goals:

1. To discuss unification in Prolog, and to explain how Prolog unification differs from standard unification. Along the way, we'll introduce =/2 , the built-in predicate for Prolog unification, and unify _with _occurs _check/2 , the built-in predicate for standard unification.
2. To explain the search strategy Prolog uses when it tries to deduce new information from old using modus ponens.

[next] [prev] [prev-tail] [front] [up]

Summary of this lecture

- In this lecture we have
  - defined unification
  - looked at the difference between standard unification and Prolog unification
  - introduced search trees