

Declarative Programming

- Knowledgebases and queries in propositional logic are made up of propositions and connectives.
- Predicate logic adds the notion of *predicates* and *variables*.
- We take a non-theoretical approach to predicate logic by introducing *declarative programming*.
- Declarative programming is the use of mathematical logic to describe the logic of computation without describing its control flow.
- Useful for: expert systems, diagnostic systems, machine learning, parsing text, theorem proving, ...

Declarative Programming

- Programmer gives a declarative specification of the problem, using the language of logic.
- The programmer should not have to tell the computer what to do.
- To get information, the programmer simply asks a query.

Datalog

- Prolog is a declarative (logical) programming language and stands for **PRO**gramming in **LOG**ic
- We only look at a subset of the language which is (roughly) equivalent to **Datalog**.

<http://www.learnprolognow.org>



Basic idea of Datalog/Prolog

- Describe the situation of interest
- Ask a question
- Prolog logically deduces new facts about the situation we described
- Prolog gives us its deductions back as answers

Consequences

- Think declaratively, not procedurally
 - Challenging
 - Requires a different mindset
 - Has similarities with other programming paradigms such as functional programming.
- High-level language
 - Not as efficient
 - Useful in many AI applications

The interpreter

- Prolog has an interactive interpreter
- After starting the interpreter, it can start reading your Prolog files and answer your queries.
- To exit Prolog simply type the command `halt.` (note the full-stop)
- Prolog program files usually have the extension `.pl` or `.pro`

Where is the program written?

- Facts and Rules are stored in one or more files forming our Knowledge Base
- Files containing KB are loaded into the interpreter
- After changing these files, the files should be loaded again to be effective
- Queries are asked in the interactive mode in front of the question prompt: ?-

Reading Files

- `consult (filename) .`
 - Reads and compiles a Prolog source file. Example:
 - `consult ('/home/user/prolog/sample.pl') .`
- `reconsult (filename) .`
 - Reconsult a changed source files. Example
 - `reconsult ('/home/user/prolog/sample.pl') .`
- `['filename'] .` (valid but don't use it; can be confused with lists!)
 - `['/home/user/prolog/sample.pl'] .`
- `make .`
 - Reconsult all changed source files.

Knowledge Base 1

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

Knowledge Base 1

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

?-

Knowledge Base 1

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

?- woman(mia).

Knowledge Base 1

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

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

Knowledge Base 1

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

```
?- woman(mia).  
yes  
?- playsAirGuitar(jody).
```

Knowledge Base 1

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

```
?- woman(mia).  
yes  
?- playsAirGuitar(jody).  
yes  
?-
```

Knowledge Base 1

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

```
?- woman(mia).  
yes  
?- playsAirGuitar(jody).  
yes  
?- playsAirGuitar(mia).  
no
```

Knowledge Base 1

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

?- tattooed(jody).

Knowledge Base 1

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

?- tattooed(jody).
no
?-

Knowledge Base 1

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

```
?- tattooed(jody).  
ERROR: predicate tattooed/1 not defined.  
?-
```

Knowledge Base 1

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

?- party.

Knowledge Base 1

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

?- party.
yes
?-

Knowledge Base 1

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

?- rockConcert.

Knowledge Base 1

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

?- rockConcert.
no
?-

Knowledge Base 2

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

Knowledge Base 2

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

Knowledge Base 2

happy(yolanda).

fact

listens2music(mia).

fact

listens2music(yolanda):- happy(yolanda).

playsAirGuitar(mia):- listens2music(mia).

playsAirGuitar(yolanda):- listens2music(yolanda).

Knowledge Base 2

happy(yolanda).

fact

listens2music(mia).

fact

listens2music(yolanda):- happy(yolanda).

rule

playsAirGuitar(mia):- listens2music(mia).

playsAirGuitar(yolanda):- listens2music(yolanda).

Knowledge Base 2

happy(yolanda).

fact

listens2music(mia).

fact

listens2music(yolanda):- happy(yolanda).

rule

playsAirGuitar(mia):- listens2music(mia).

rule

playsAirGuitar(yolanda):- listens2music(yolanda).

Knowledge Base 2

The diagram shows a knowledge base with five statements. The first two are facts, and the last three are rules. Each statement has a blue callout box pointing to it with the word 'fact' or 'rule'.

happy(yolanda). **fact**

listens2music(mia). **fact**

listens2music(yolanda):- happy(yolanda). **rule**

playsAirGuitar(mia):- listens2music(mia). **rule**

playsAirGuitar(yolanda):- listens2music(yolanda). **rule**

Knowledge Base 2

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



head

body

Knowledge Base 2

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

?-

Knowledge Base 2

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

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

Knowledge Base 2

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

```
?- playsAirGuitar(mia).  
yes  
?- playsAirGuitar(yolanda).  
yes
```


Clauses

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

*There are five clauses in this knowledge base:
two facts, and three rules.*

The end of a clause is marked with a full stop.

Predicates

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

*There are three **predicates**
in this knowledge base:
happy, listens2music, and playsAirGuitar*

Knowledge Base 3

happy(vincent).

listens2music(butch).

playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).

playsAirGuitar(butch):- happy(butch).

playsAirGuitar(butch):- listens2music(butch).

Expressing Conjunction

```
happy(vincent).  
listens2music(butch).  
playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).  
playsAirGuitar(butch):- happy(butch).  
playsAirGuitar(butch):- listens2music(butch).
```

The comma "," expresses conjunction in Prolog

Knowledge Base 3

```
happy(vincent).  
listens2music(butch).  
playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).  
playsAirGuitar(butch):- happy(butch).  
playsAirGuitar(butch):- listens2music(butch).
```

```
?- playsAirGuitar(vincent).  
no  
?-
```

Knowledge Base 3

```
happy(vincent).  
listens2music(butch).  
playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).  
playsAirGuitar(butch):- happy(butch).  
playsAirGuitar(butch):- listens2music(butch).
```

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

Expressing Disjunction

```
happy(vincent).  
listens2music(butch).  
playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).  
playsAirGuitar(butch):- happy(butch).  
playsAirGuitar(butch):- listens2music(butch).
```

```
happy(vincent).  
listens2music(butch).  
playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).  
playsAirGuitar(butch):- happy(butch); listens2music(butch).
```

Prolog and Logic

- Clearly Prolog has something to do with logic
- Operators
 - Implication :-
 - Conjunction ,
 - Disjunction ;
- Use of modus ponens

Knowledge Base 4

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

loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).

Prolog Variables

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

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

```
?- woman(X).
```

Variable Instantiation

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

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

```
?- woman(X).  
X=mia
```

Asking Alternatives

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

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

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

Asking Alternatives

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

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

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

Asking Alternatives

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

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

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

Asking Alternatives

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

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

```
?- woman(X).  
X=mia;  
X=jody;  
X=yolanda;  
no
```

Knowledge Base 4

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

loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).

?- loves(marsellus,X), woman(X).

Knowledge Base 4

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

loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).

?- loves(marsellus,X), woman(X).
X=mia
?-

Knowledge Base 4

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

loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).

?- loves(pumpkin,X), woman(X).

Knowledge Base 4

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

loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).

?- loves(pumpkin,X), woman(X).
no
?-

Knowledge Base 5

```
loves(vincent,mia).  
loves(marsellus,mia).  
loves(pumpkin, honey_bunny).  
loves(honey_bunny, pumpkin).  
  
jealous(X,Y):- loves(X,Z), loves(Y,Z).
```

Knowledge Base 5

```
loves(vincent,mia).  
loves(marsellus,mia).  
loves(pumpkin, honey_bunny).  
loves(honey_bunny, pumpkin).  
  
jealous(X,Y):- loves(X,Z), loves(Y,Z).
```

```
?- jealous(marsellus,W).
```

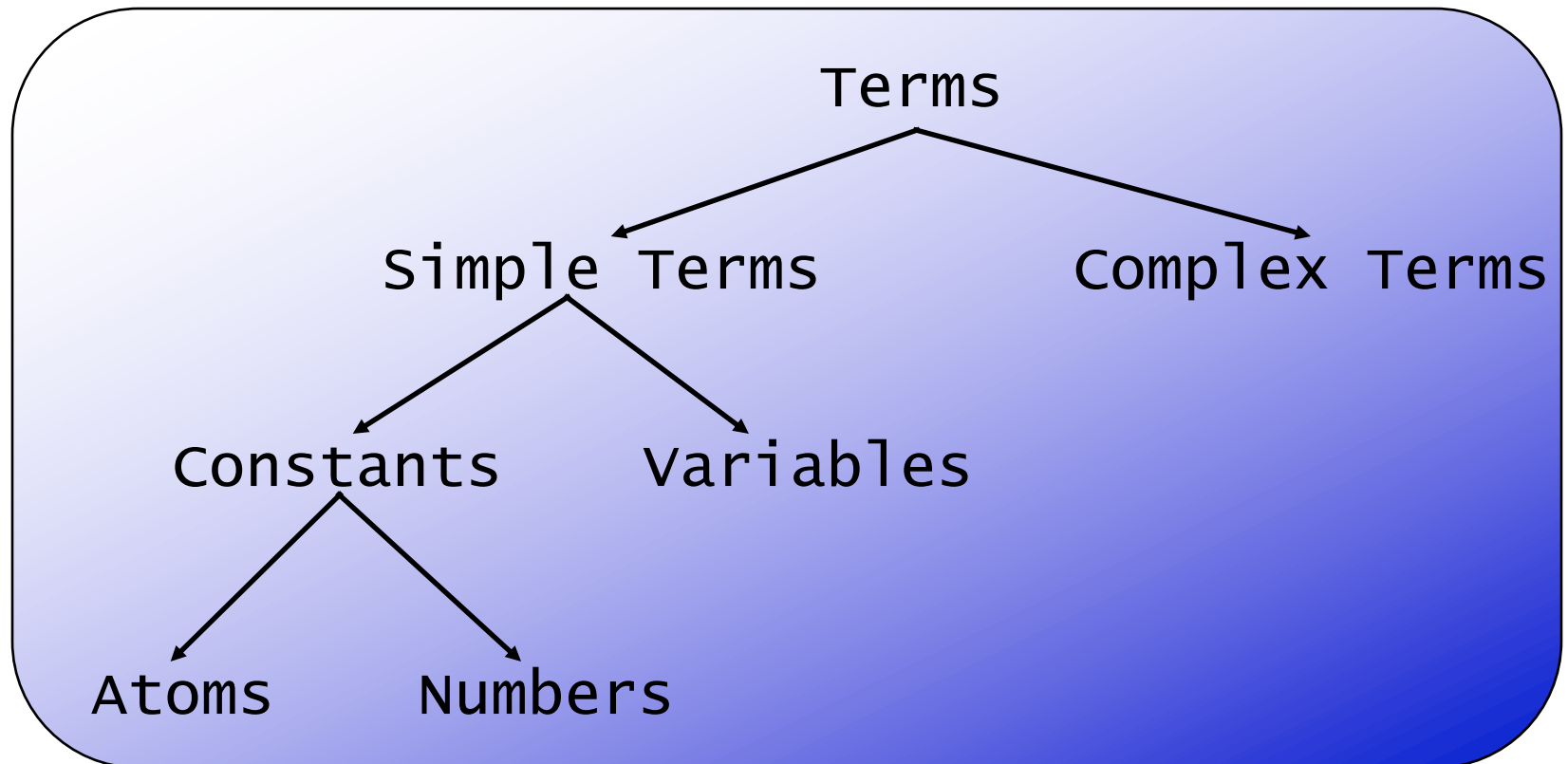
Knowledge Base 5

```
loves(vincent,mia).  
loves(marsellus,mia).  
loves(pumpkin, honey_bunny).  
loves(honey_bunny, pumpkin).  
  
jealous(X,Y):- loves(X,Z), loves(Y,Z).
```

```
?- jealous(marsellus,W).  
W=vincent  
?-
```

Prolog Syntax

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



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'**, **'@\$%'**
- A sequence of special characters
 - *Examples:* **:**, **,**, **;**, **.**, **:-**

Numbers

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

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

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 we have seen before:
 - `playsAirGuitar(jody)`
 - `loves(vincent, mia)`
 - `jealous(marsellus, W)`
- Complex terms inside complex terms:
 - `hide(X, father(father(father(butch))))`

Ariety

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

woman(mia) is a term with arity 1
loves(vincent,mia) has arity 2
father(father(butch)) arity 1

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

Example of Arity

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

- This knowledge base defines
 - happy/1
 - listens2music/1
 - playsAirGuitar/1

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:
 - **mia** and **X**

Unification

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

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 1/3

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:
 1. They have the same functor and arity, and
 2. all their corresponding arguments unify, and
 3. 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

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

Example with complex terms

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

$X = s(g)$

$Y = t(k)$

yes

?-

Example with complex terms

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

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).

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
- Prolog has a specific way of answering queries:
 - Search knowledge base from top to bottom
 - Processes clauses from left to right
 - Backtracking to recover from bad choices

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).

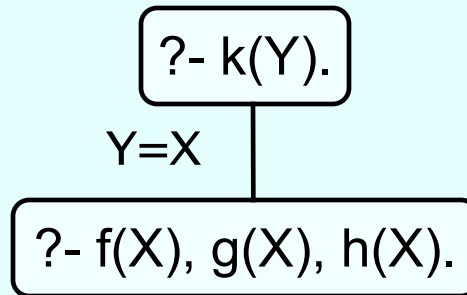
?- k(Y).

?- k(Y).

Example: search tree

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

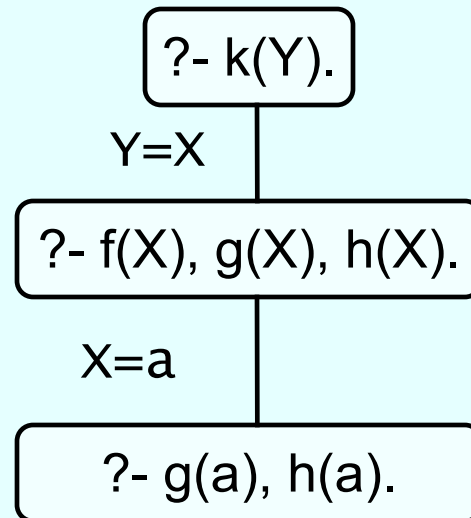
?- k(Y).



Example: search tree

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

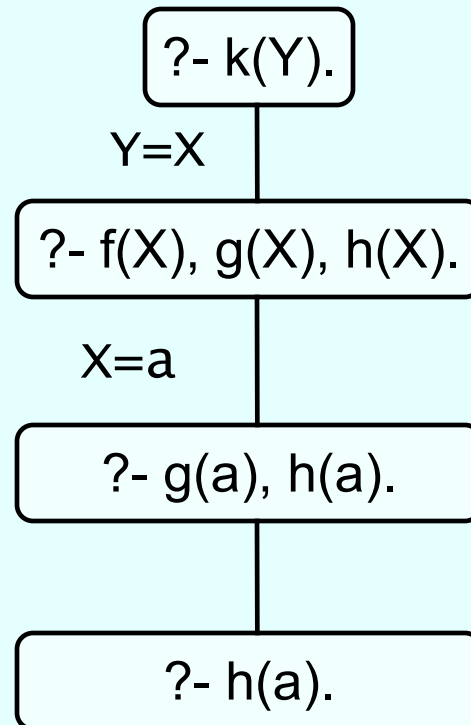
?- k(Y).



Example: search tree

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

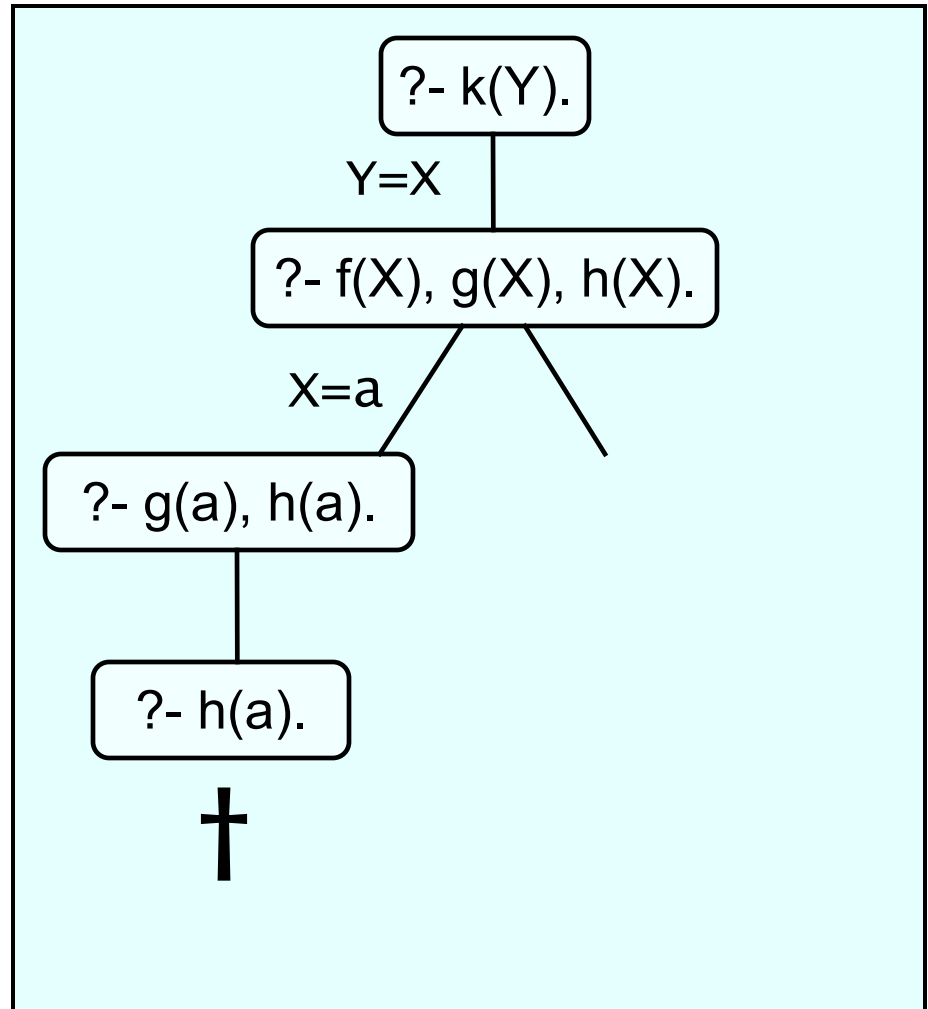
?- k(Y).



Example: search tree

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

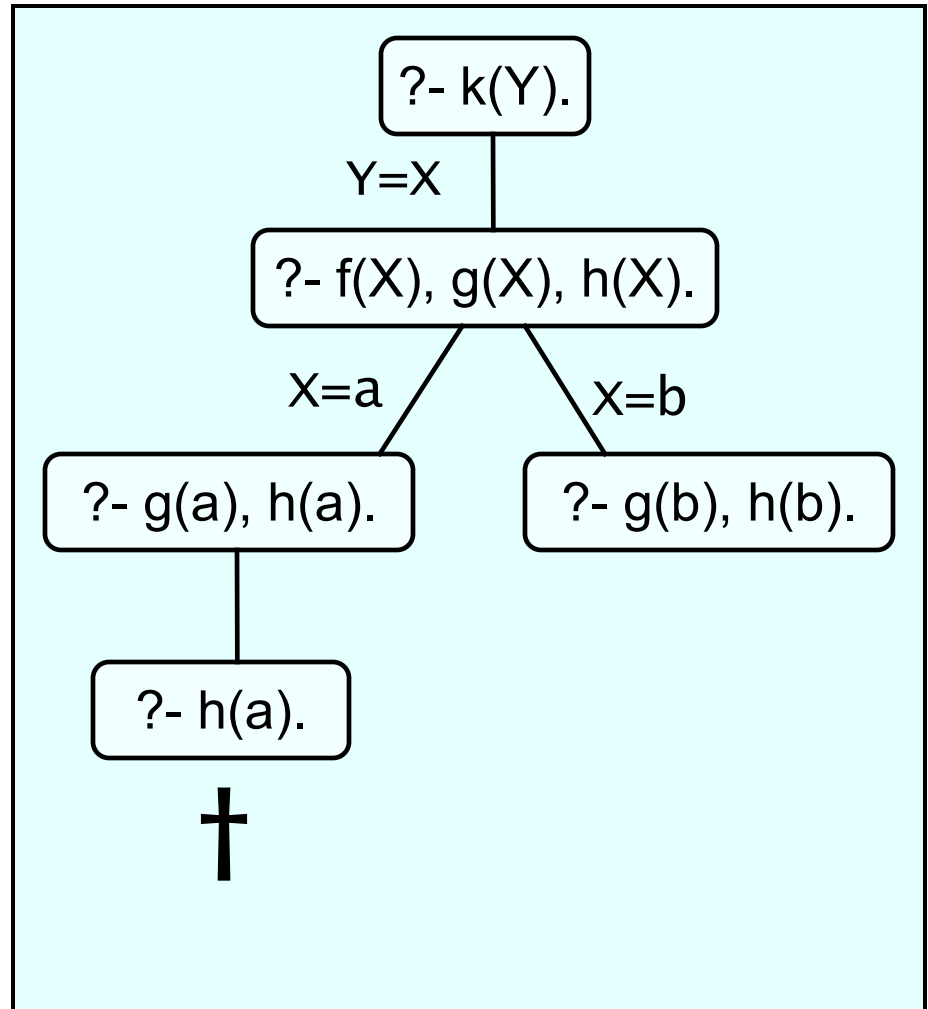
?- k(Y).



Example: search tree

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

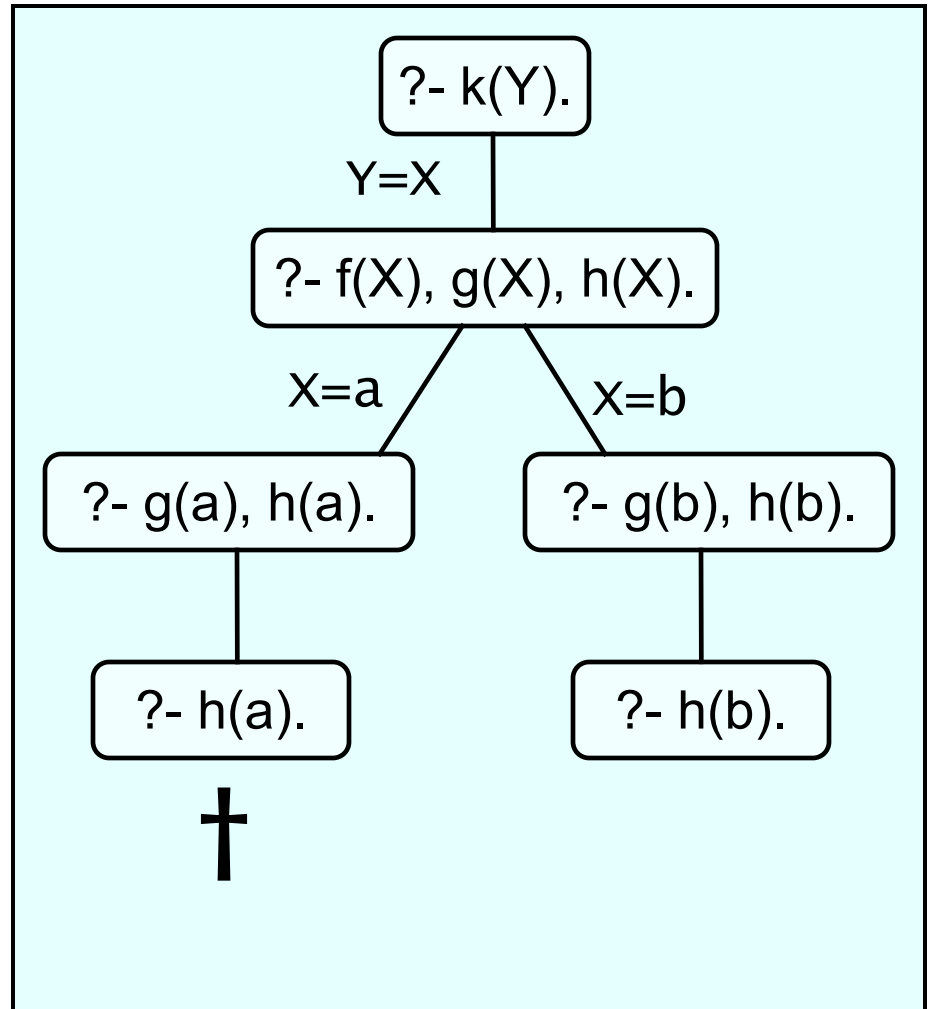
?- k(Y).



Example: search tree

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

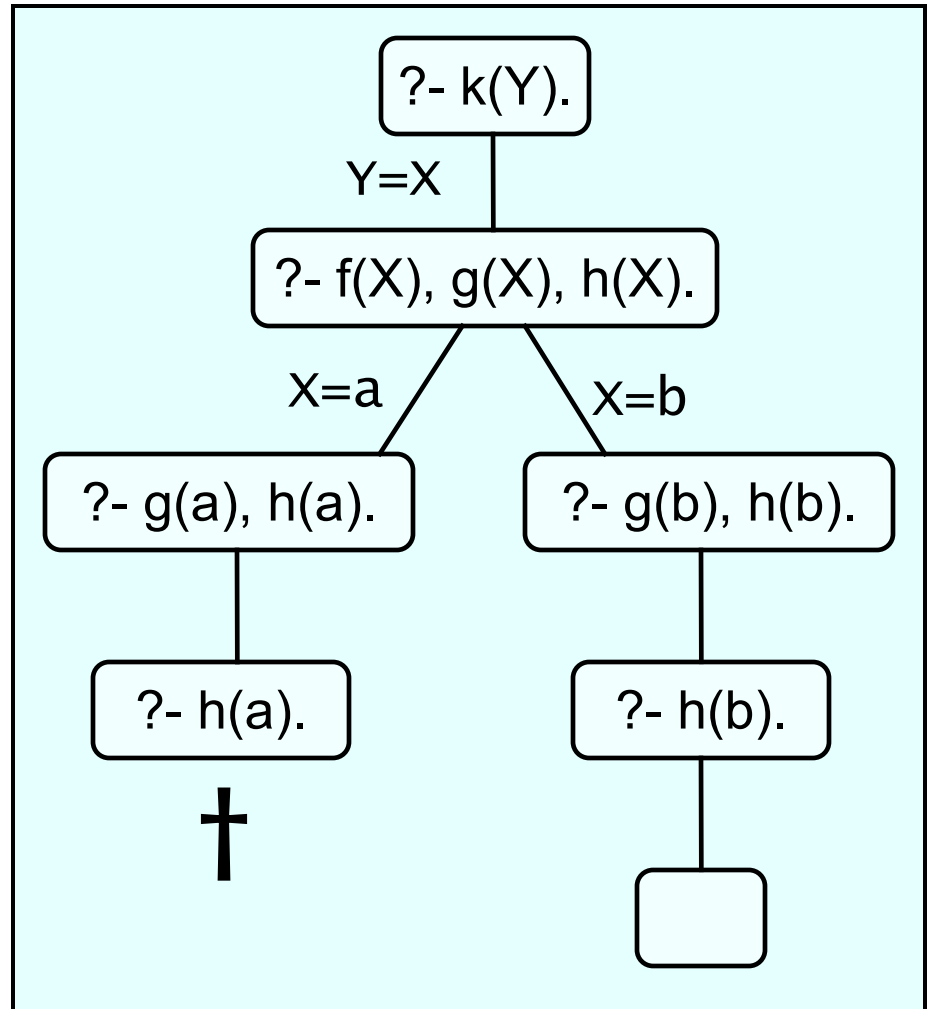
?- k(Y).



Example: search tree

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

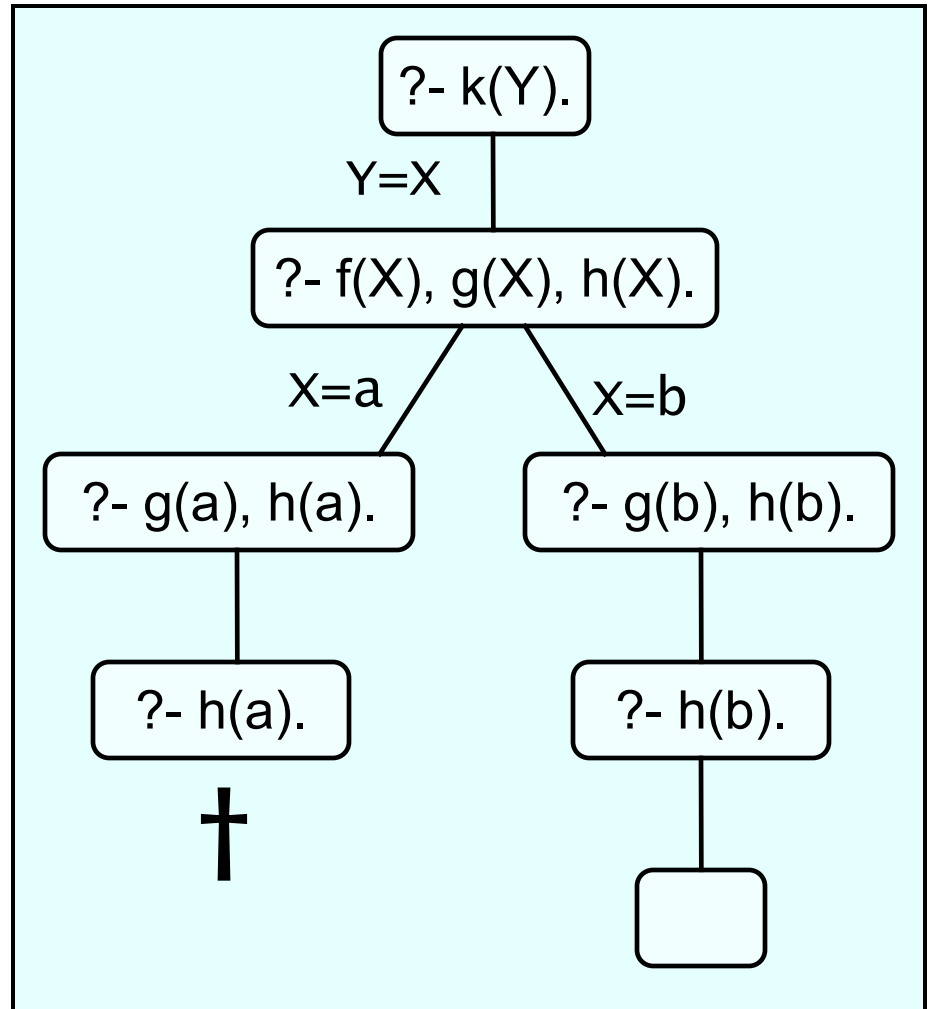
?- k(Y).
Y=b



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).
```

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

X=A

Y=B

```
?- loves(A,C), loves(B,C).
```

Another example

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

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

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

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

X=A Y=B

```
?- loves(A,C), loves(B,C).
```

A=vincent

C=mia

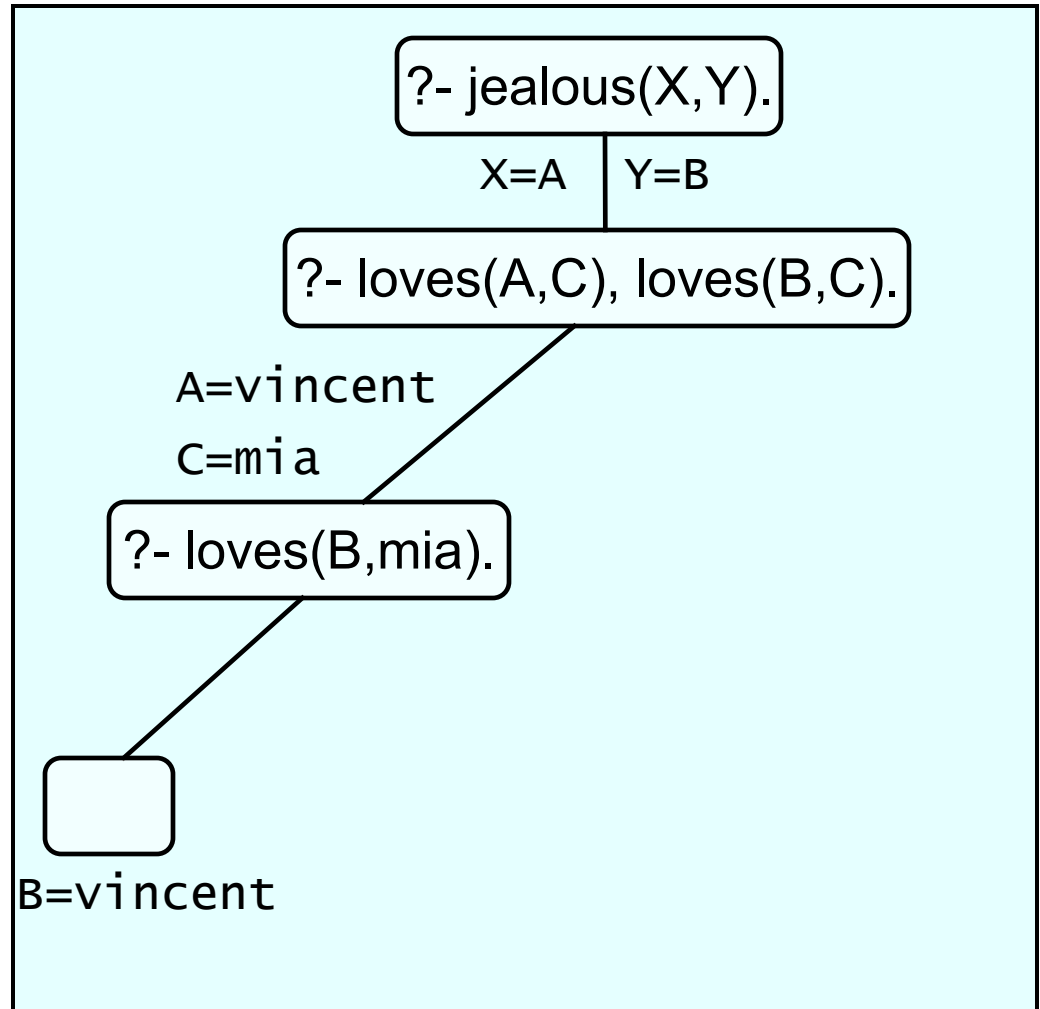
```
?- loves(B,mia).
```

Another example

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

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

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

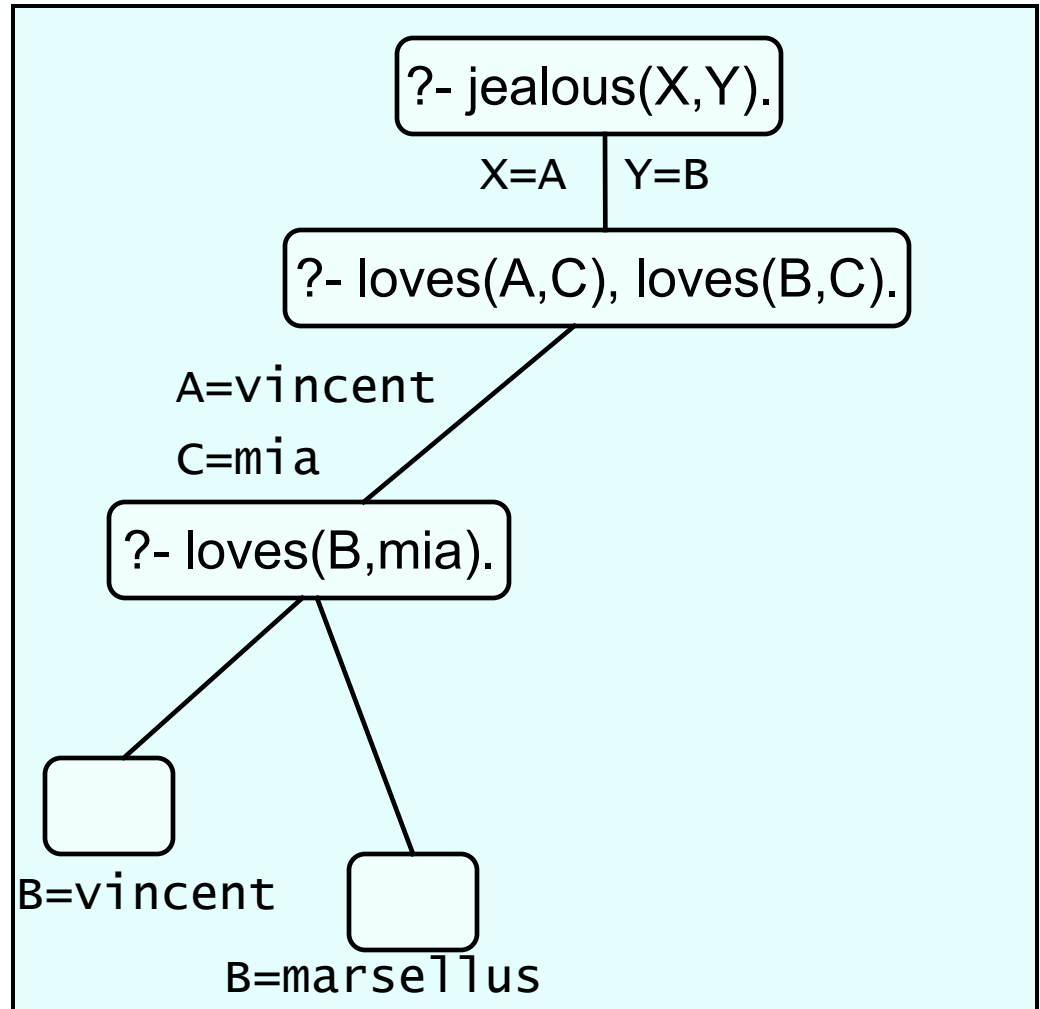


Another example

```
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
```

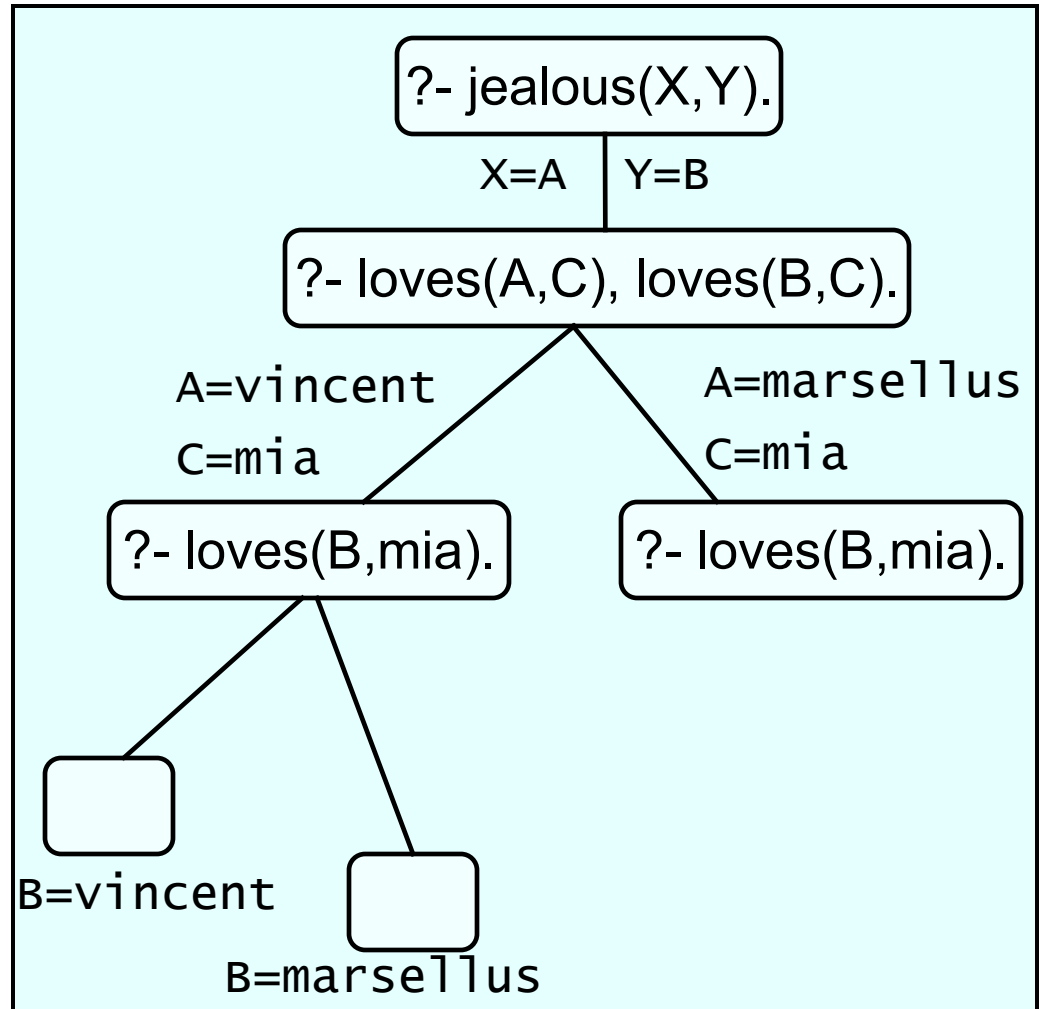


Another example

```
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;
```



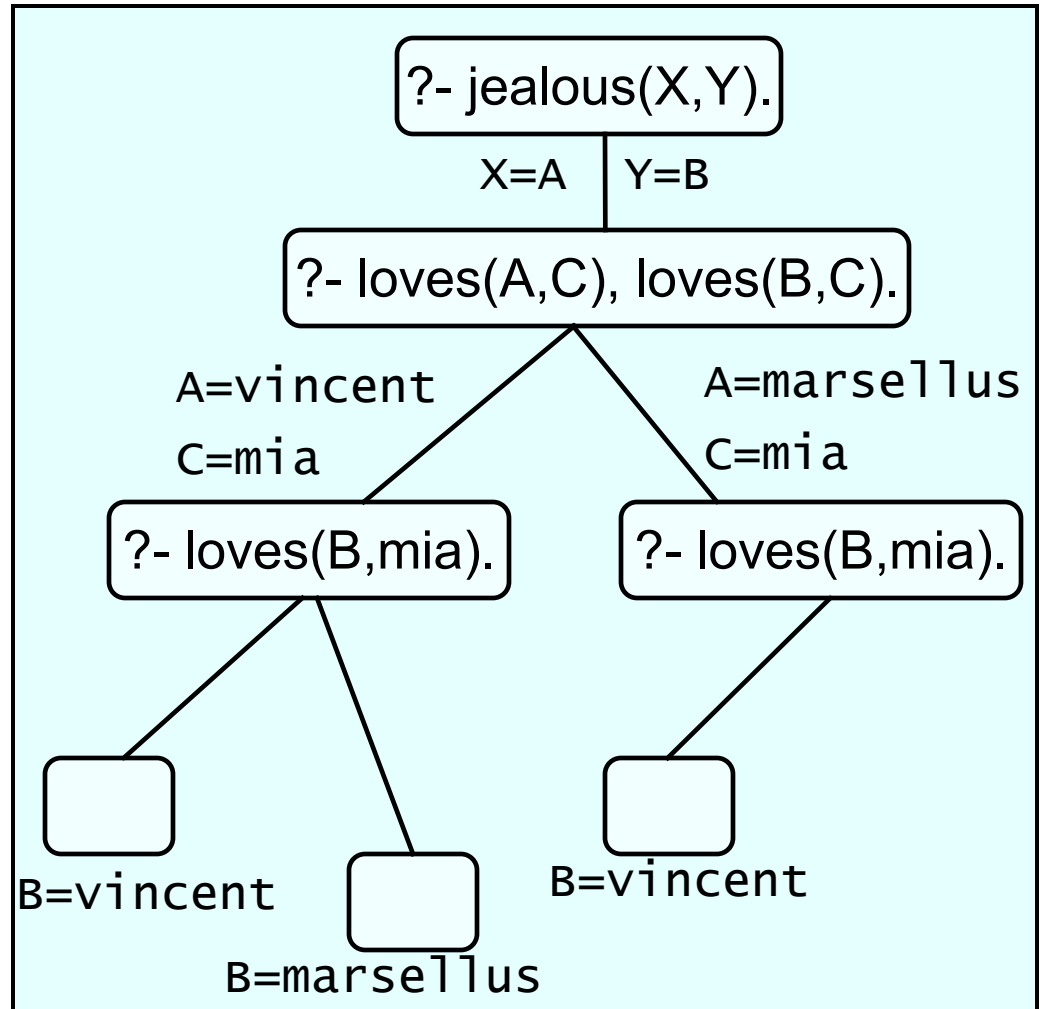
Another example

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

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

....

```
X=vincent  
Y=marsellus;  
X=marsellus  
Y=vincent
```

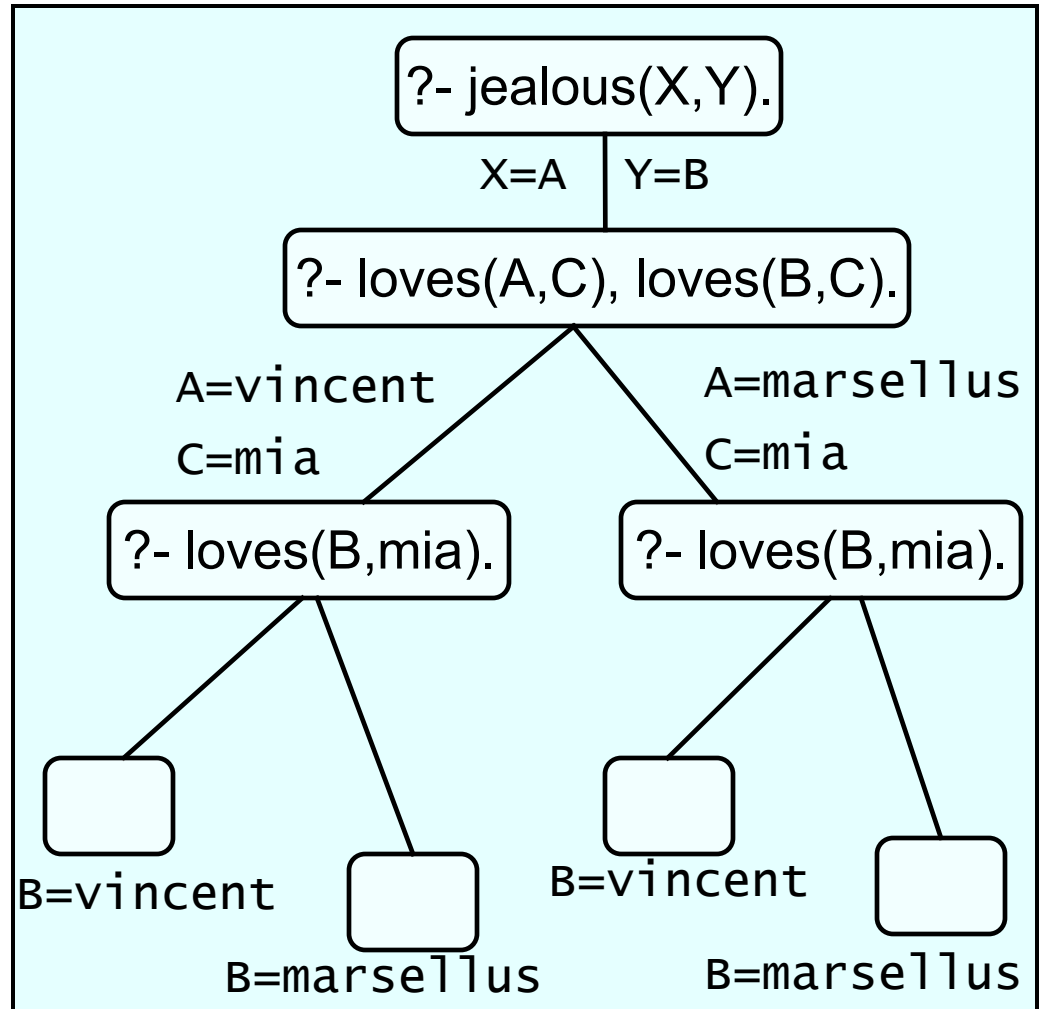


Another example

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

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

```
....  
X=marsellus  
Y=vincent;  
X=marsellus  
Y=marsellus
```

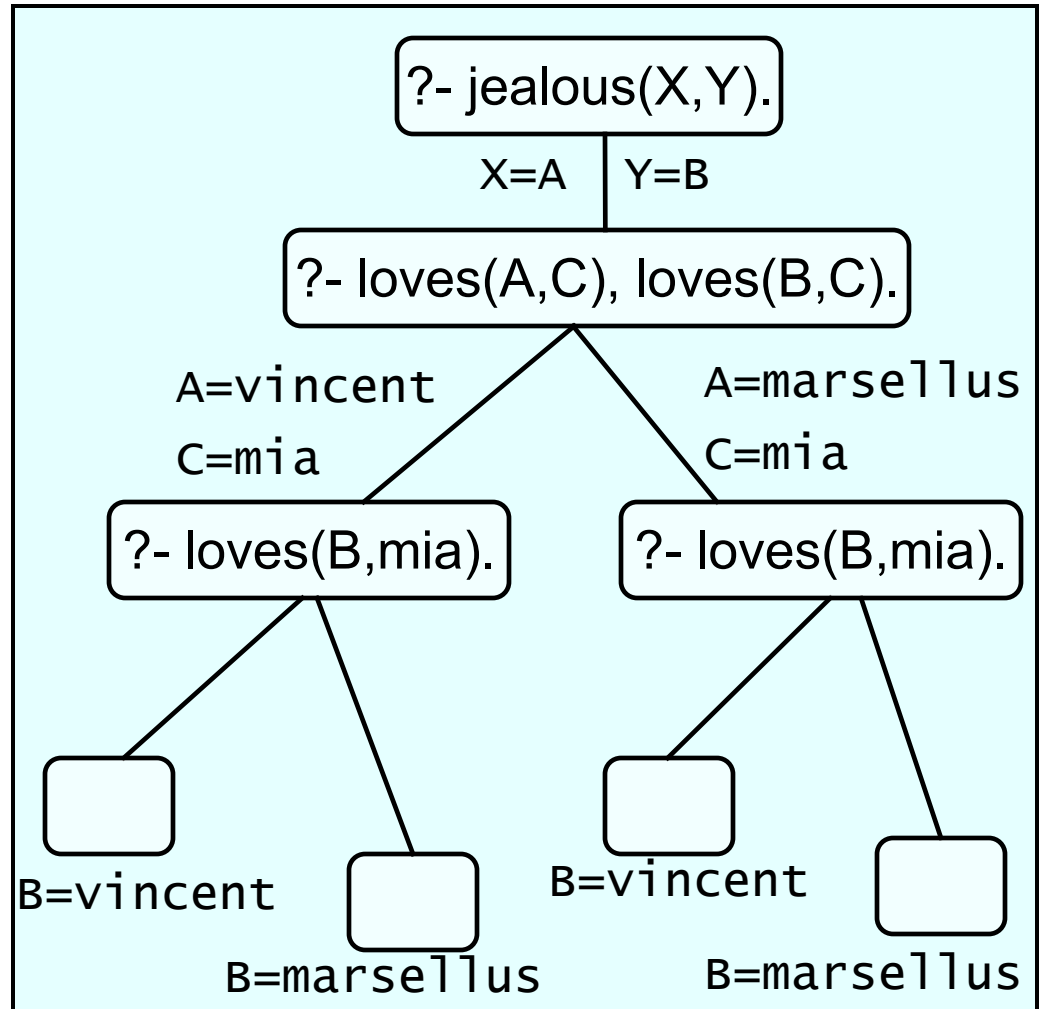


Another example

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

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

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



Recursive Definitions

- Prolog predicates can be defined recursively
- A predicate is recursively defined if one or more rules in its definition refers to itself

Example: Descendant

```
child(anna,bridget).  
child(bridget,caroline).  
child(caroline,donna).  
child(donna,emily).
```

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

?-

Example: Descendant

```
child(anna,bridget).  
child(bridget,caroline).  
child(caroline,donna).  
child(donna,emily).
```

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

```
?- descend(anna,donna).  
yes
```


Another recursive definition

p:- p.

?- p.

Another recursive definition

p:- p.

?- p.

ERROR: out of memory

Example: Successor

- Suppose we use the following way to write numerals:
 1. **0** is a numeral.
 2. If **X** is a numeral, then so is **succ(X)**.

Example: Successor

```
numeral(0).  
numeral(succ(X)):- numeral(X).
```

Example: Successor

```
numeral(0).  
numeral(succ(X)):- numeral(X).
```

```
?- numeral(succ(succ(succ(0)))).  
yes  
?-
```

Example: Successor

```
numeral(0).  
numeral(succ(X)):- numeral(X).
```

```
?- numeral(X).
```

Example: Successor

```
numeral(0).  
numeral(succ(X)):- numeral(X).
```

```
?- numeral(X).  
X=0;  
X=succ(0);  
X=succ(succ(0));  
X=succ(succ(succ(0)));  
X=succ(succ(succ(succ(0))))
```

Example: Addition

?- add(succ(succ(0)),succ(succ(succ(0))), Result).
Result=succ(succ(succ(succ(succ(0)))))
yes

Example: Addition

```
add(0,X,X).
```

%%% base clause

```
?- add(succ(succ(0)),succ(succ(succ(0))), Result).
```

```
Result=succ(succ(succ(succ(succ(0)))))
```

```
yes
```

Example: Addition

```
add(0,X,X).                                %%% base clause
```

```
add(succ(X),Y,succ(Z)):-                    %%% recursive clause  
    add(X,Y,Z).
```

```
?- add(succ(succ(0)),succ(succ(succ(0))), Result).  
Result=succ(succ(succ(succ(succ(0)))))  
yes
```

Examples from the book

descend1.pl

```
child(anna,bridget).  
child(bridget,caroline).  
child(caroline,donna).  
child(donna,emily).
```

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

```
?- descend(A,B).  
A=anna  
B=bridget
```

descend2.pl

```
child(anna,bridget).  
child(bridget,caroline).  
child(caroline,donna).  
child(donna,emily).
```

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

```
?- descend(A,B).  
A=anna  
B=emily
```

descend3.pl

```
child(anna,bridget).  
child(bridget,caroline).  
child(caroline,donna).  
child(donna,emily).
```

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

```
?- descend(A,B).  
ERROR: OUT OF LOCAL STACK
```

descend4.pl

```
child(anna,bridget).  
child(bridget,caroline).  
child(caroline,donna).  
child(donna,emily).
```

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

```
?- descend(A,B).
```

Using dif predicate

```
mother(X, Y) :- parent(X, Y), female(X).  
sister(X, Y) :-  
    parent(Z, X),  
    parent(Z, Y),  
    female(X).
```

- What is wrong with this rule?
 - Any female is her own sister
- Solution?
 - Use dif (\neq) predicate: ..., $X \neq Y$.

Comments

- Multi-line :

```
/* This is a comment  
   This is another comment */
```

- Short :

```
% This is also a comment
```

Notation of Predicate Descriptions

- + Argument must be fully instantiated.

Think of the argument as *input*.

- Argument must be unbound.

Think of the argument as *output*.

- ? Either instantiated or unbound

Think of the argument as either *input* or *output*

Examples:

Write a predicate `sum(+List, ?Sum)` so that ...

Write a predicate `append(?List1, ?List2, ?List1AndList2)` such that ...

Side effects

- Some built-in “Predicates” may have side effects.
- Example: The built-in predicate `write(+Term)`

```
print_if_positive(X) :- X > 0, write(X).
```

Logical quantification

- Variables that appear in the head of a rule are universally quantified.
- Variables that only appear in the body of a rule are existentially quantified.

`path(X, Y) :- edge(X, Z), path(Z, Y).`

For all nodes X and Y, there is a path from X to Y if *there exists* a node Z such that there is an edge from X to Z and there is path from Z to Y.