Unification and Proof Search in Prolog

Module of Logics and Artificial Intelligence course

Unification – Starting example recap

The knowledge base contains only one fact:

male(ned).

the simplest query is:

male(X).

Prolog **unifies** the two complex terms by **instantiating** the variable **X** with **ned.**

Unification

Unification is possible between two terms:

- if they are the same term
- if they contain variables that can be uniformly instantiated with terms in such a way that the resulting terms are equal

Examples:

ned and **ned** unify
42 and **42** unify
male(ned) and **male(ned)** unify
ned and **arya** do not unify
male(ned) and **male(richard)** do not unify

Instantiations

When Prolog unifies two terms, it instantiates any needed variable and and that instantiations are used afterwards.

What Prolog responds to:

?-X=ned.

?-male(X)=male(ned).

?-father(ned,X)=father(X,arya).

?-father(ned,X)=father(Z,arya).

?-X=ned,X=arya.

Instantiations

When Prolog unifies two terms, it instantiates any needed variable and and that instantiations are used afterwards.

What Prolog responds to:

?-X=ned.

X = ned.

?-male(X)=male(ned).

X = ned.

?-father(ned,X)=father(X,arya).

false.

?-father(ned,X)=father(Z,arya).

X = arya.

Z= ned.

?-X=ned,X=arya.

false.

Recap

- **If T1 and T2 are constants**, then T1 and T2 **unify** if they are the **same atom**, or the same number

- **If T1 is a variable and T2 is any type of term**, then T1 and T2 **unify**, and **T1 is instantiated to T2** (and vice versa)

- **If T1 and T2 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**.

More examples

What Prolog responds to:

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

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

?-father(X,X)=father(ned,arya).

More examples

What Prolog responds to:

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

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

?-father(X,X)=father(ned,arya).
False.

Prolog and unifications

What Prolog responds to:
?-male(X)=X.

Do these terms unify or not?

Prolog and unifications

What Prolog responds to:
?-male(X)=X.

Do these terms unify or not?

X=male(male(male(male(male(male(male(male(male(male(male(mal
e(male(male(male(male(male(male(male(male(male(male(male(mal
e(male(male(male(male(male(male(male(male(male(male(male(mal
e(male(male(…
Infinite terms..

In Prolog:
X = male(X).

Prolog unifies without **occurs check**

Unification without occurs check

Unification without occurs check can lead to **unsound inference**.

Example
A resolution proof can be found for the non-theorem:

$$\forall x \exists y\ parent(y, x) \Rightarrow \exists y \forall x\ parent(y, x)$$

because of the lack of occurs check make **X** unifiable with **f(X)** after skolemization.

Prolog – unification with occurs check:
?- unify_with_occurs_check(male(X), X).
False.

SLD resolution

Selective **L**inear **D**efinite clause resolution is a **sound** and **complete inference rule** for **Horn clauses**.

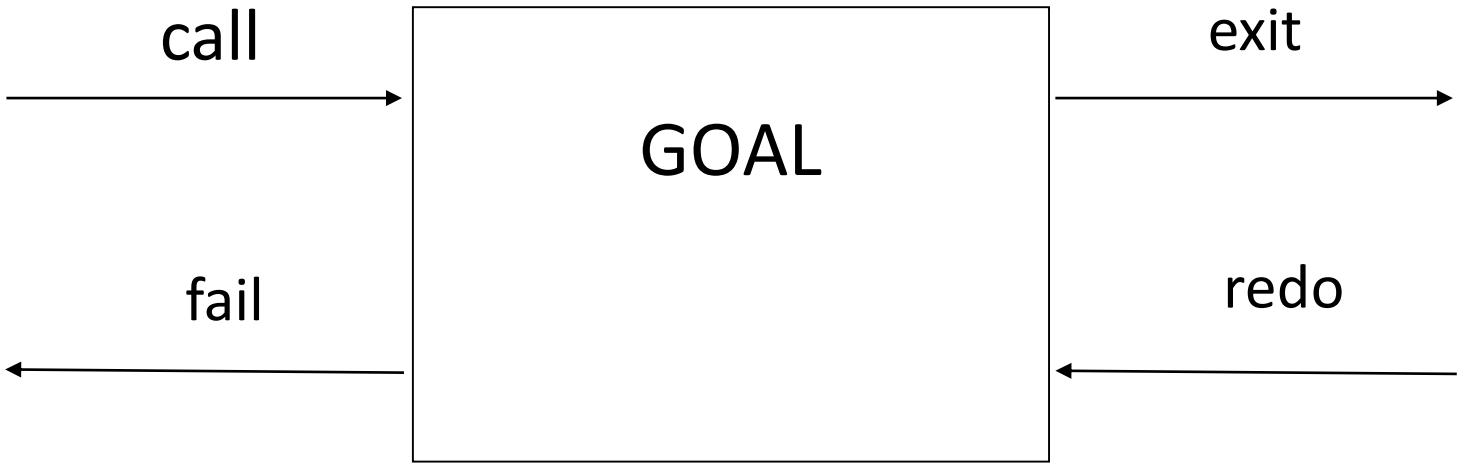
New clauses are derived via **backward reasoning**, using an input clause as goal-reduction procedure.

The unifying substitution steps:

- **Input from the subgoal to the body** of the procedure,
- **Output from the head to remaining subgoals**.

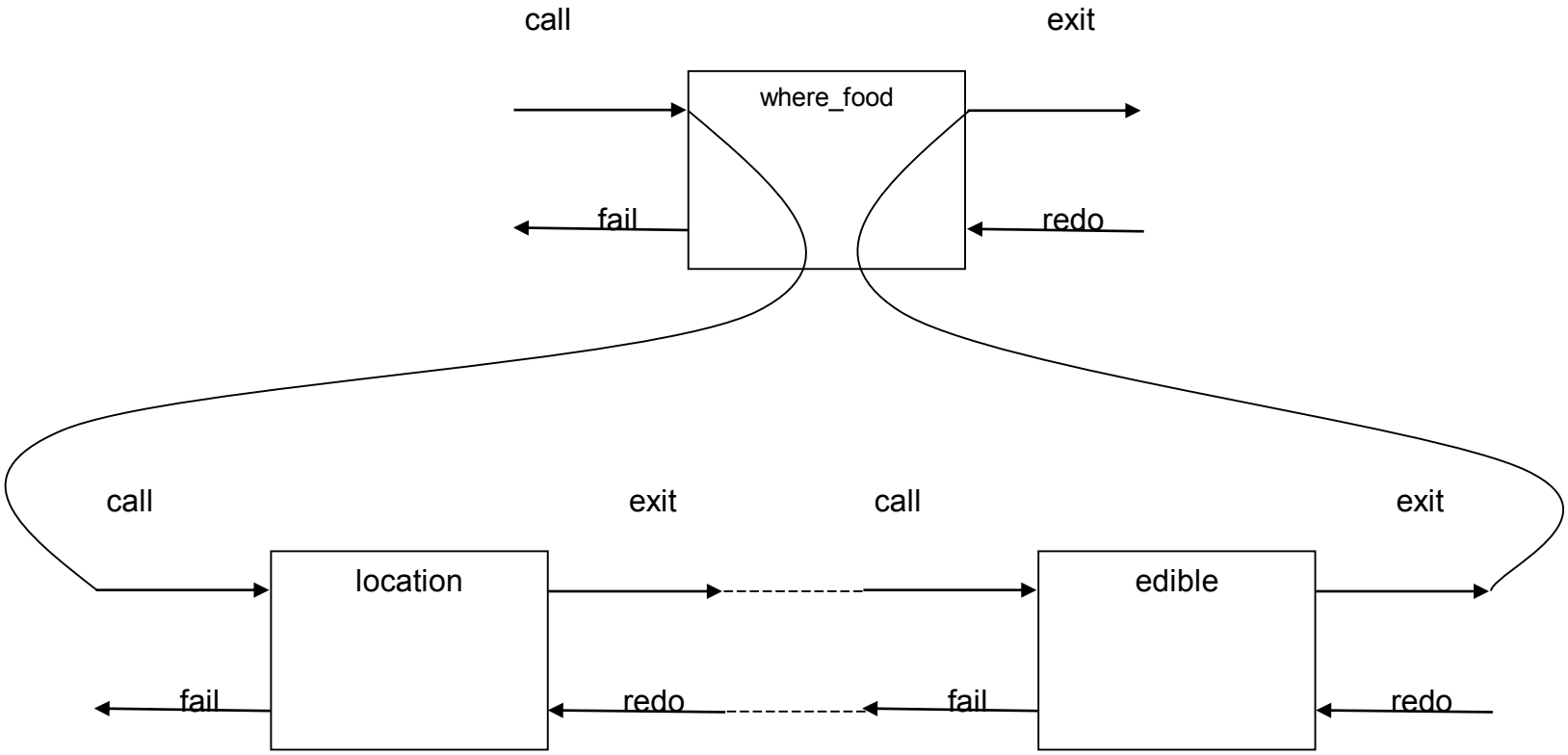
Goal representation

The query can be observed as an input-output black box:



A rule call

What Prolog does when a rule is called:
where_food(X,Y) :- location(X,Y),edible(X).



Trace

The trace command can be useful in order to analyze Prolog steps within the query execution.

8 ?- trace.

[trace] 8 ?- where_food(X,Y).

Call: (7) where_food(_G457, _G458) ? creep

Call: (8) location(_G457, _G458) ? creep

Exit: (8) location(desk, office) ? creep

Call: (8) edible(desk) ? Creep

…

X and **Y** are instantiated with internal anon variables **_G457** and **_G458.**

The query execution

When a query is submitted to the Prolog interpreter, **it tries to prove that the query can be logically derived from the program**.
This consists in **finding a substitution** such that the query can be derived from the knowledge base.

parent(richard,ned).
parent(ned,sansa).
child(Y,X):-parent(X,Y).

?-child(X,Y).

The unification **X/richard Y/ned** let the query to be derived from the program.

Finding for a solution

When a Prolog program is executed, a goal and a rule are involved, the search of a solution can be represented through a tree in which the Prolog interpreter looks for the solution among all alternatives. This **AND-OR** tree is named **search tree**.

The Prolog interpreter uses a **depth first search** strategy with **backtracking**.

The search tree is built evaluating alternatives **one by one** and exploring the sub-goals **from left to right**, analyzing the following sub-goal only when the previous one is satisfied.

Backtracking

When a goal fails, the interpreter uses the **backtracking** technique and **explores the alternative branches**.
When the navigation goes back the **instantiations are annulled** following an inverted order**, from the right to the left**.

The interpreter looks for further matches to satisfy the goal, using different clauses because unification algorithm does not contemplate alternatives.

Example

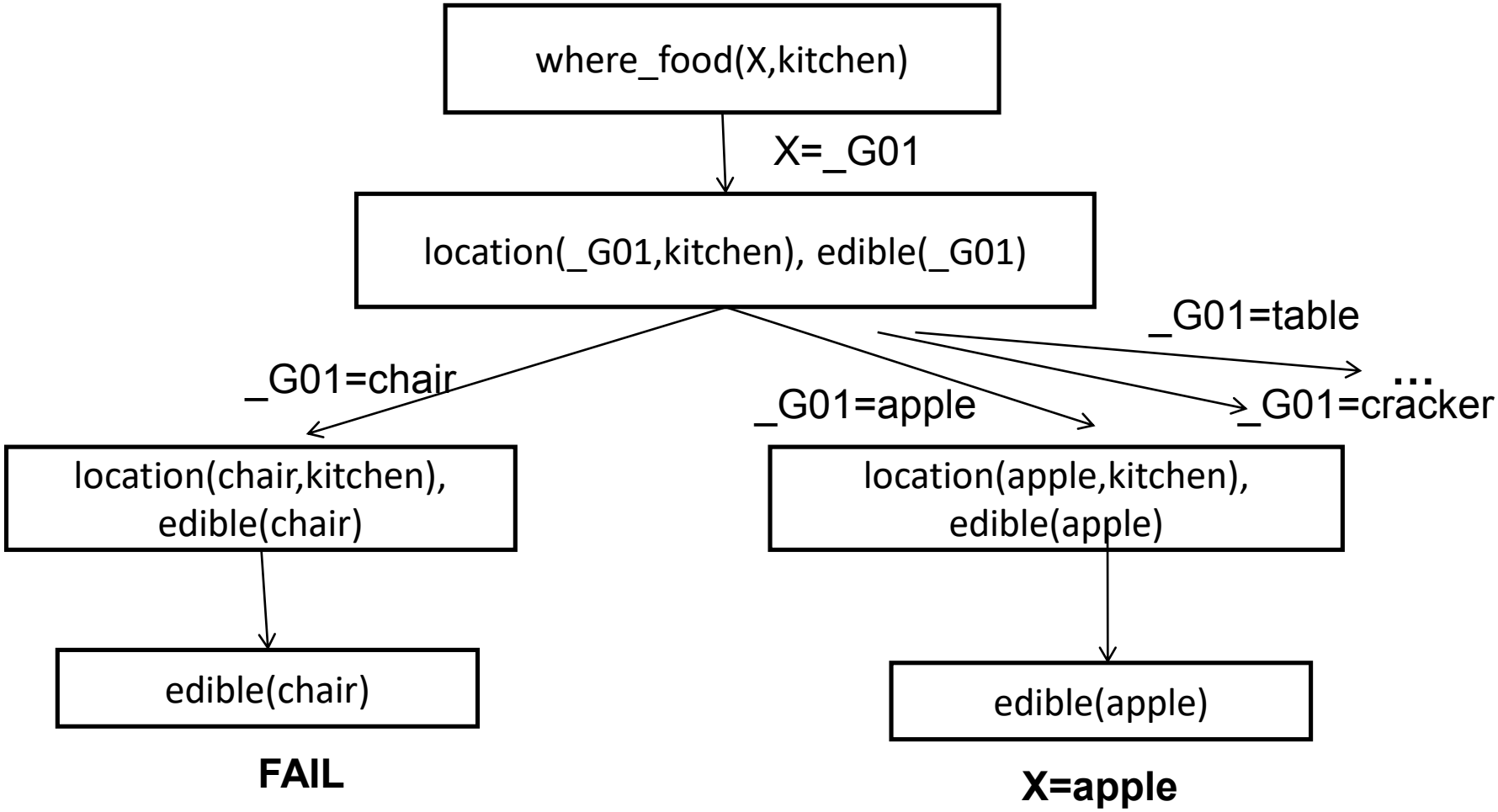
Knowledge base:

```
location(chair,kitchen).
location(apple,kitchen).
location(crackers,kitchen).
location(table,kitchen).
edible(apple).
edible(crackers).
where_food(X,Y)  :-  location(X,Y),edible(X).
```

Query:

```
?- where_food(X,kitchen).
```

Example


where_food(X,kitchen)
X=_G01
location(_G01,kitchen), edible(_G01)
_G01=chair
_G01=apple
_G01=table
...
_G01=cracker
location(chair,kitchen), edible(chair)
location(apple,kitchen), edible(apple)
edible(chair)
edible(apple)
FAIL
X=apple

Example

If the **cardinality** of the instances of the relation **is** previously **known** we could **evaluate** the **efficiency** of the program.
If the number of edible objects is lower than the total number of objects we could switch the subgoals from

where_food(X,Y) :- location(X,Y),edible(X).

to

where_food(X,Y) :- edible(X), location(X,Y).

Using a first fail heuristic to explore a potentially smaller tree.

What search tree is built in this case?

Example

Knowledge base:

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

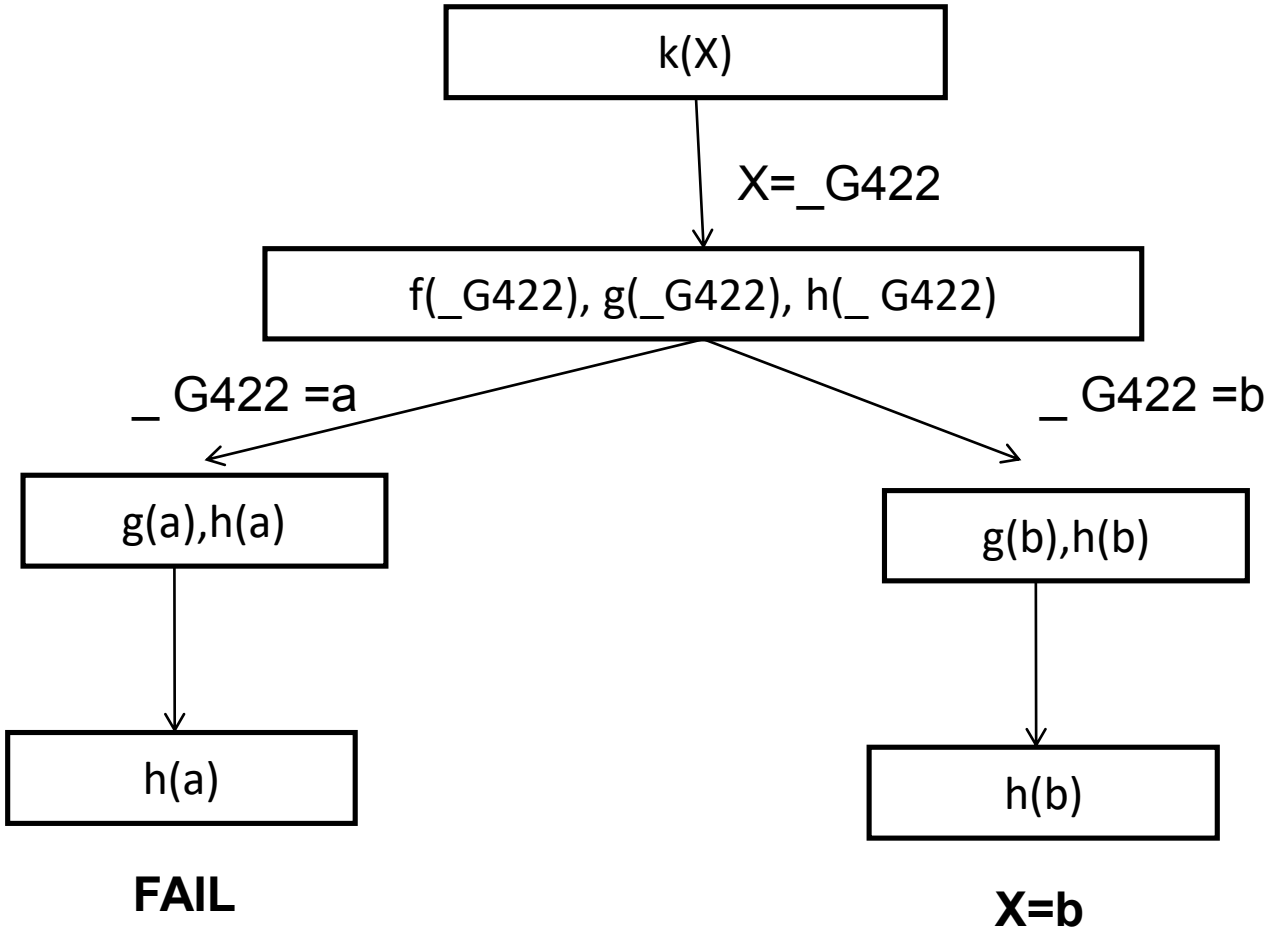
Query:

?- k(X).

Example

```
[trace] 12 ?- k(X).
   Call: (6) k(_G422) ? creep
   Call: (7) f(_G422) ? creep
   Exit: (7) f(a) ? creep
   Call: (7) g(a) ? creep
   Exit: (7) g(a) ? creep
   Call: (7) h(a) ? creep
   Fail: (7) h(a) ? creep
   Redo: (7) f(_G422) ? creep
   Exit: (7) f(b) ? creep
   Call: (7) g(b) ? creep
   Exit: (7) g(b) ? creep
   Call: (7) h(b) ? creep
   Exit: (7) h(b) ? creep
   Exit: (6) k(b) ? creep
X = b.
```

Example


k(X)
X=_G422
f(_G422), g(_G422), h(_ G422)
_ G422 =a
_ G422 =b
g(a),h(a)
g(b),h(b)
h(a)
h(b)
FAIL
X=b

Example

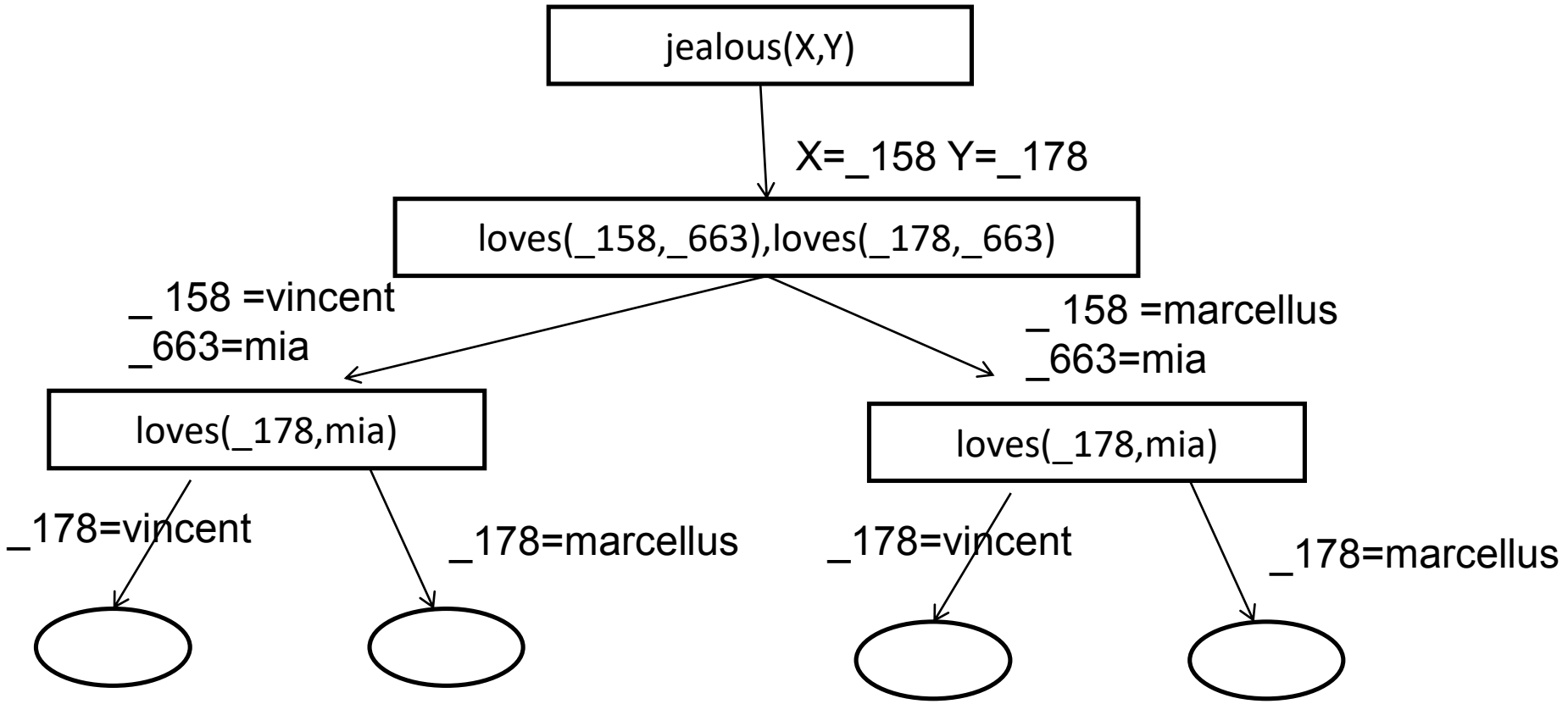
Knowledge base:

loves(vincent,mia).
loves(marcellus,mia).
jealous(X,Y) :-loves(X,Z),loves(Y,Z).

Query:

?- jealous(X,Y).

Example


jealous(X,Y)
X=_158 Y=_178
loves(_158,_663),loves(_178,_663)
_ 158 =vincent
_663=mia
_ 158 =marcellus
_663=mia
loves(_178,mia)
loves(_178,mia)
_178=vincent
_178=marcellus
_178=vincent
_178=marcellus