CS F214 - Logic in CS
Prolog – Lecture 1

BITS Pilani
Pilani Campus

Jagat Sesh Challa

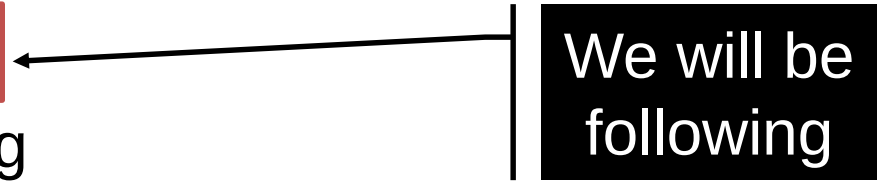
Source

Learn Prolog Now

http://www.learnprolognow.org/

Prolog Interpreters

- Many interpreters available in market
  - JIProlog
  - SWI-Prolog
  - Open Prolog
  - Strawberry Prolog
  - GNU Prolog
  - etc.



Today's Lecture

- Introduction to Prolog
  - Facts, Rules and Queries
  - Prolog Syntax
- Unification
- Proof search

Prolog

- *"Programming with Logic"*
- Very different from other programming languages
  - Declarative (not procedural)
  - Recursion (no "for" or "while" loops)
  - Relations (no functions)
  - Unification

Prolog and Web Applications

Prolog programs are often smaller

- smallness encourages well written code
- hence, easier to maintain

Source:

http://www.pathwayslms.com/swipltuts/

Basic idea of Prolog

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

Consequences

- Think declaratively, not procedurally
  - Challenging
  - Requires a different mindset
- High-level language
  - Not as efficient as, say, C
  - Good for rapid prototyping
  - Useful in many AI applications (knowledge representation, inference)

Knowledge Base 1

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

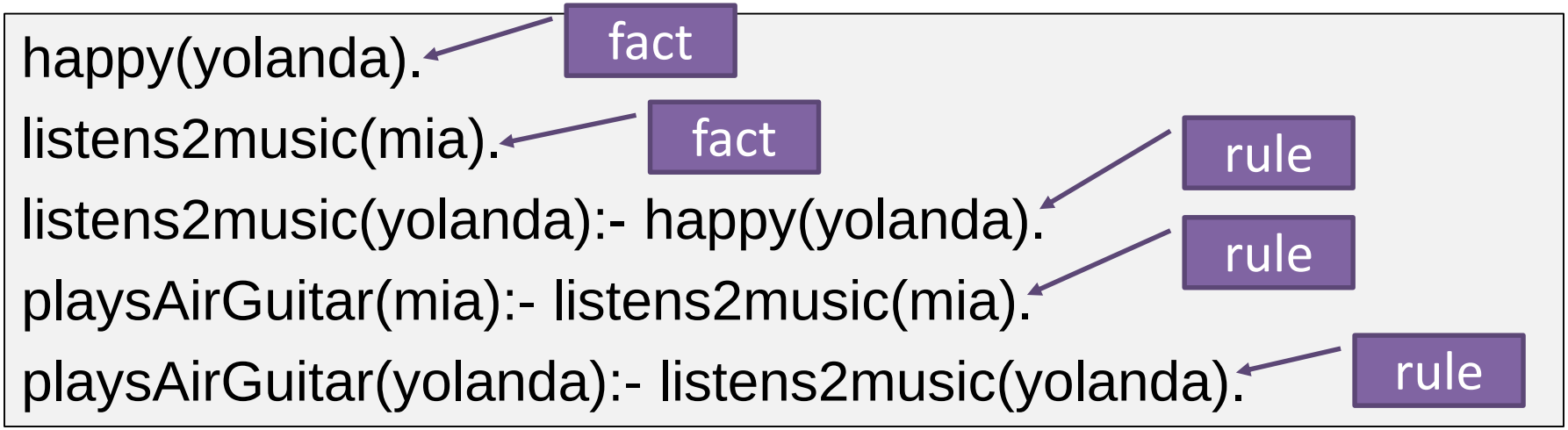
```
?- woman(mia).
yes
?- playsAirGuitar(jody).
yes
?- playsAirGuitar(mia).
no
```

Knowledge Base 1

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

```
?- tatooed(jody).
ERROR: predicate tattoed/1 not defined.
?- party.
yes
?- rockConcert.
no
```

Knowledge Base 2



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).
```

```
?- 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

innovate achieve lead

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

Expressing Disjunction

innovate achieve lead

```
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 & Logic

- Clearly, Prolog has something to do with logic...

	Prolog	Logic
Implication	A:-B	$B \rightarrow A$
Conjunction	A,B	$A \wedge B$
Disjunction	A;B	$A \vee B$

- Use of inference (modus ponens)
- Negation (?)

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 and 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).
X=mia
yes
?-
```

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

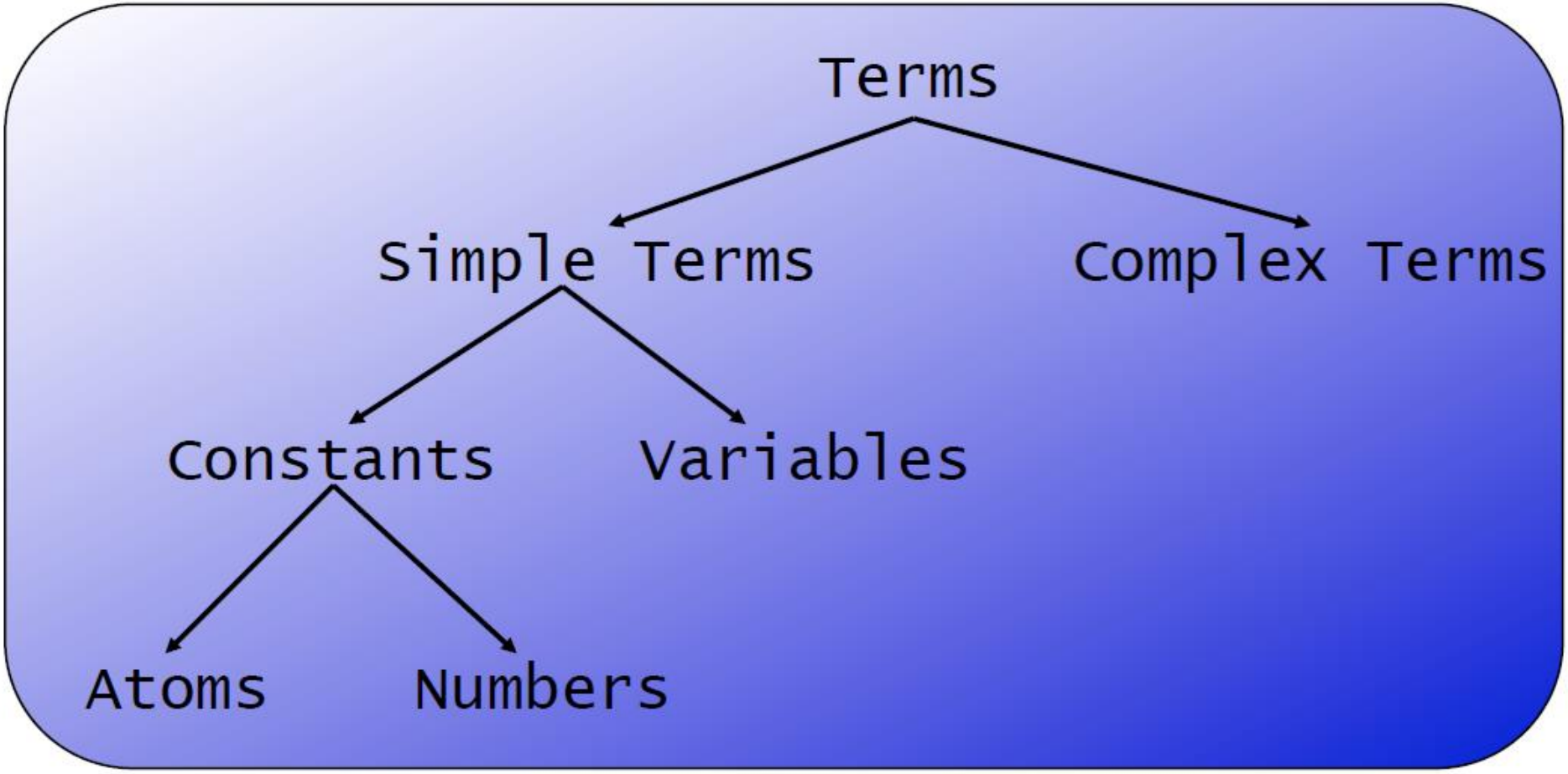
```
loves(vincent,mia).
loves(marsellus,mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).
jealous(X,Y):- loves(X,Z), loves(Y,Z).
```

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

Syntax of Prolog

- Q: What exactly are facts, rules and queries built out of?
- A: Prolog terms

Prolog terms



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, 0.3435

Variables

- A sequence of characters of uppercase 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))))

Arity

- 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

- 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

innovate achieve lead

BITS Pilani

Pilani Campus

Unification in Prolog

Unification

- Recall the previous example, where we said that Prolog unifies

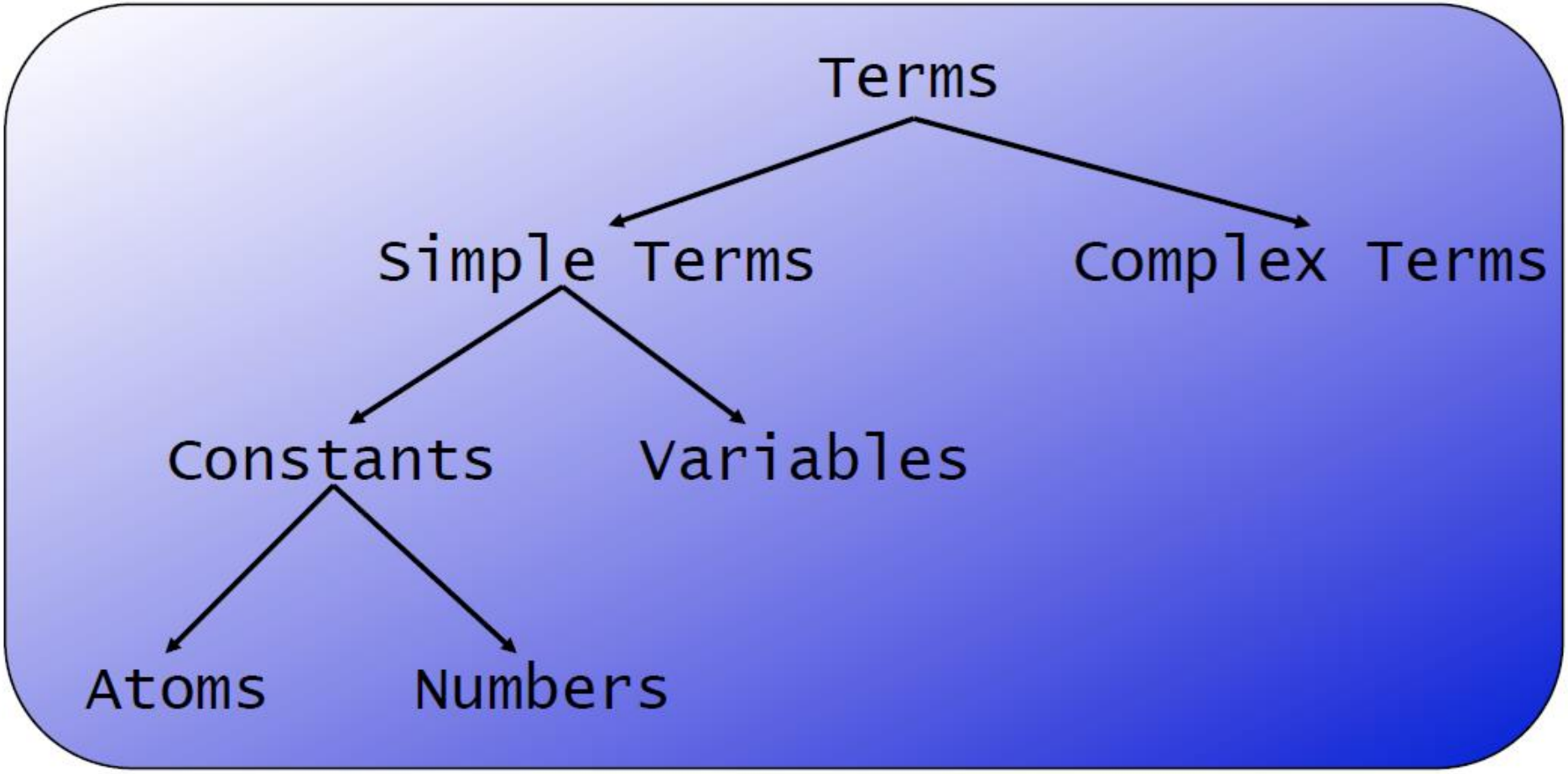
woman(X)

with

woman(mia)

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

Recall Prolog Terms



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**
  - **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 very 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:
    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

?- mia = vincent.

no

?- mia = X.

X=mia

yes

?-

How will Prolog respond?

innovate achieve lead

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

Example with Complex Terms


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

One last example

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

no

?-

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

innovate achieve lead

```
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
?- horizontal(line(point(2,3),Point)).
Point = point(_554,3);
no
```

Exercise

Which of the following pairs of terms unify? Where relevant, give the variable instantiations that lead to successful unification.

1. bread = bread
2. 'Bread' = bread
3. 'bread' = bread
4. Bread = bread
5. bread = sausage
6. food(bread) = bread
7. food(bread) = X
8. food(X) = food(bread)
9. food(bread,X) = food(Y,sausage)
10. food(bread,X,beer) = food(Y,sausage,X)
11. food(bread,X,beer) = food(Y,kahuna_burger)
12. food(X) = X
13. meal(food(bread),drink(beer)) = meal(X,Y)
14. meal(food(bread),X) = meal(X,drink(beer))

Exercise

We are working with the following knowledge base:

```
house_elf(dobby).
witch(hermione).
witch('McGonagall').
witch(rita_skeeter).
magic(X):- house_elf(X).
magic(X):- wizard(X).
magic(X):- witch(X).
```

Which of the following queries are satisfied? Where relevant, give all the variable instantiations that lead to success.

1. ?- magic(house_elf).
2. ?- wizard(harry).
3. ?- magic(wizard).
4. ?- magic('McGonagall').
5. ?- magic(Hermione).

innovate achieve lead

BITS Pilani

Pilani Campus

Proof Search

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 and search trees

Example

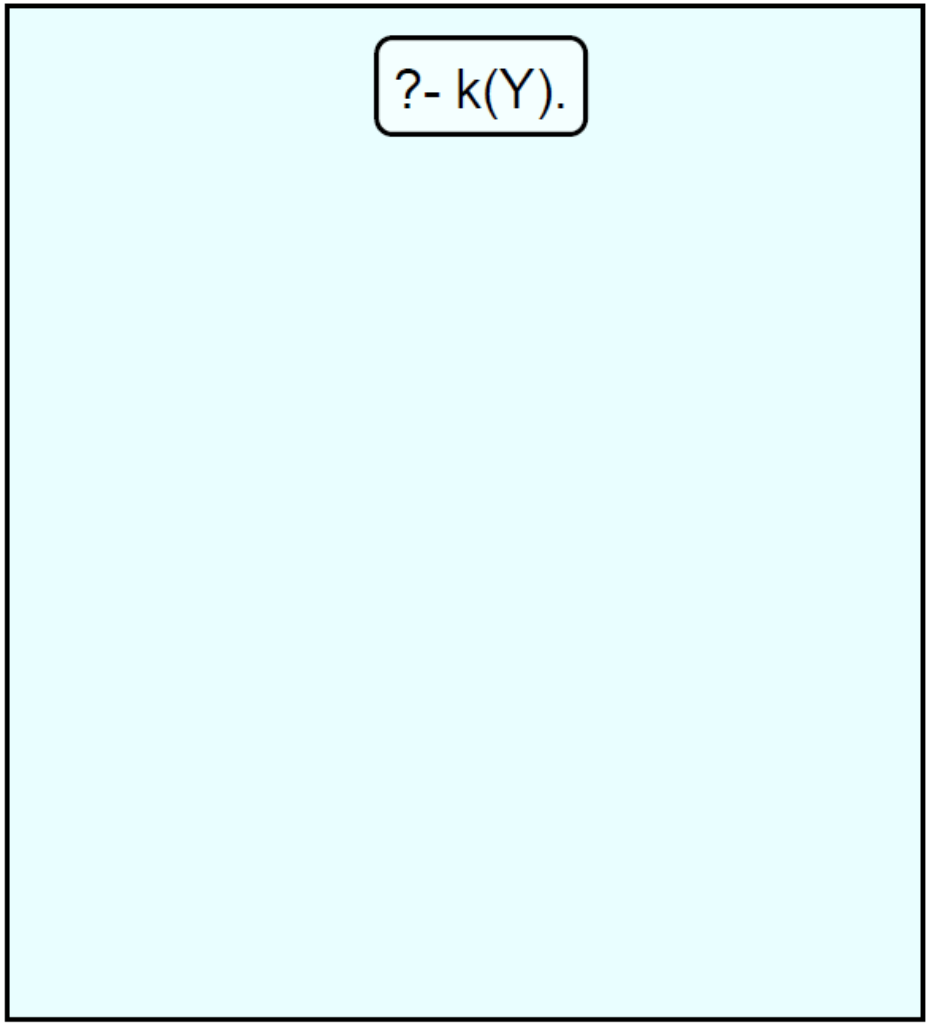
```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

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

Example: Search Tree

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

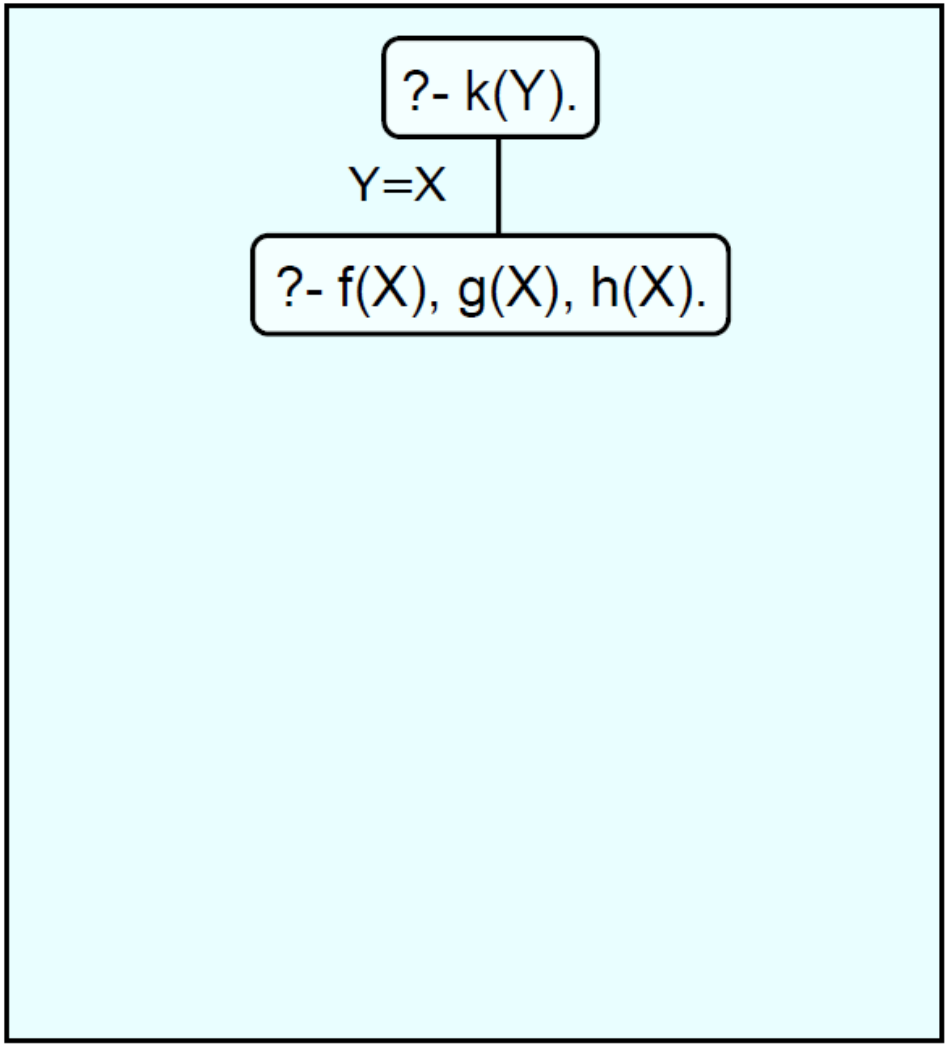
```
?- k(Y).
```



Example: Search Tree

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

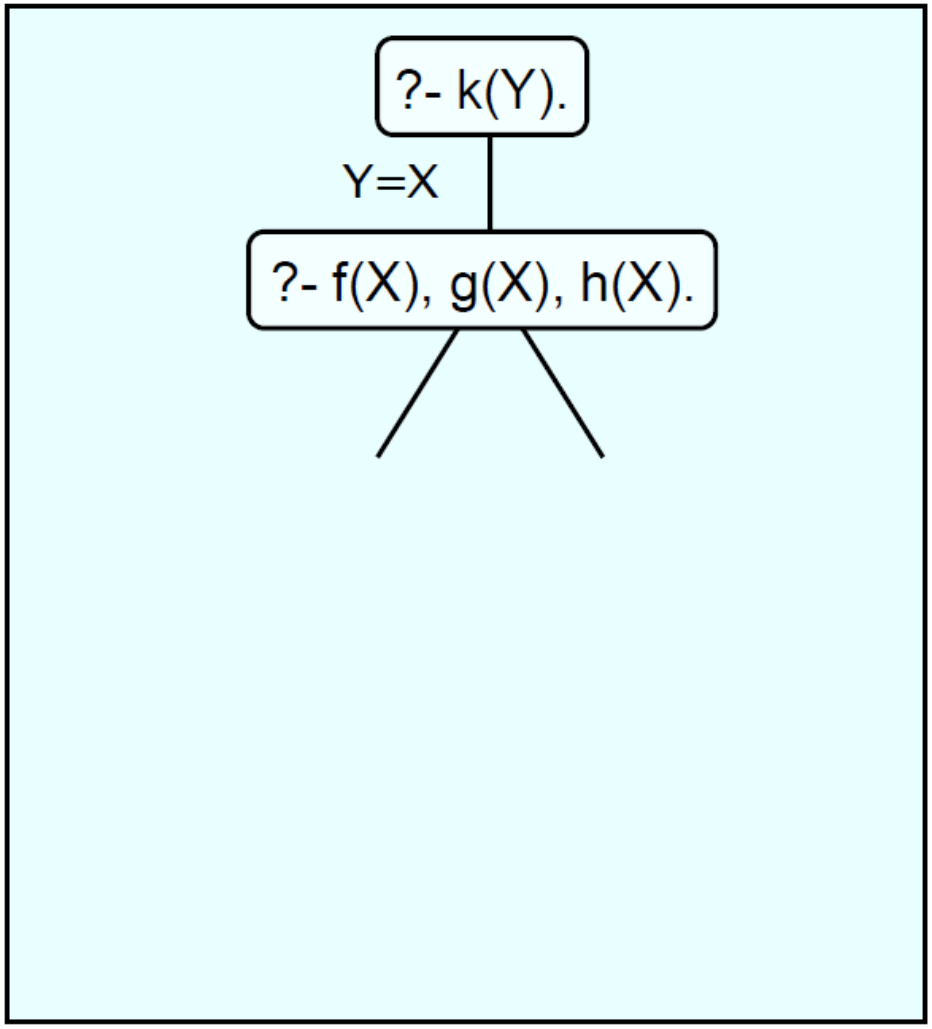
```
?- k(Y).
```



Example: Search Tree

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

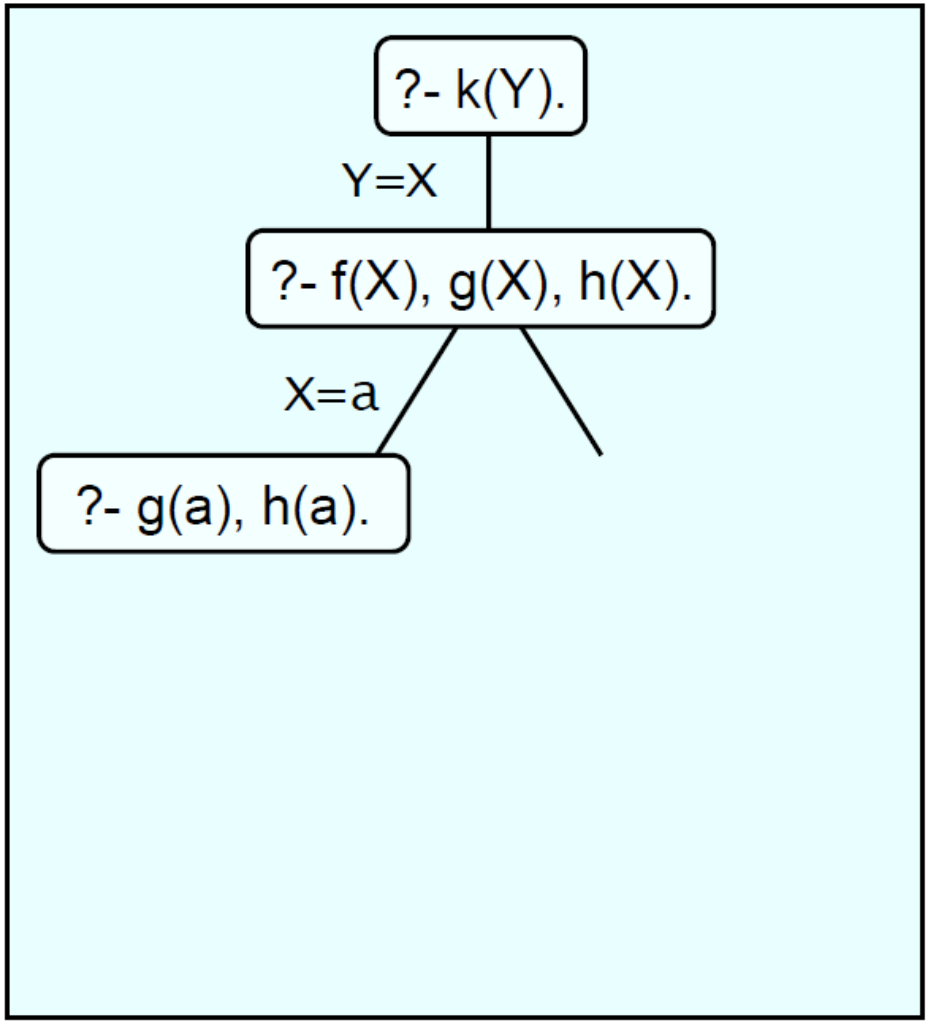
```
?- k(Y).
```



Example: Search Tree

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

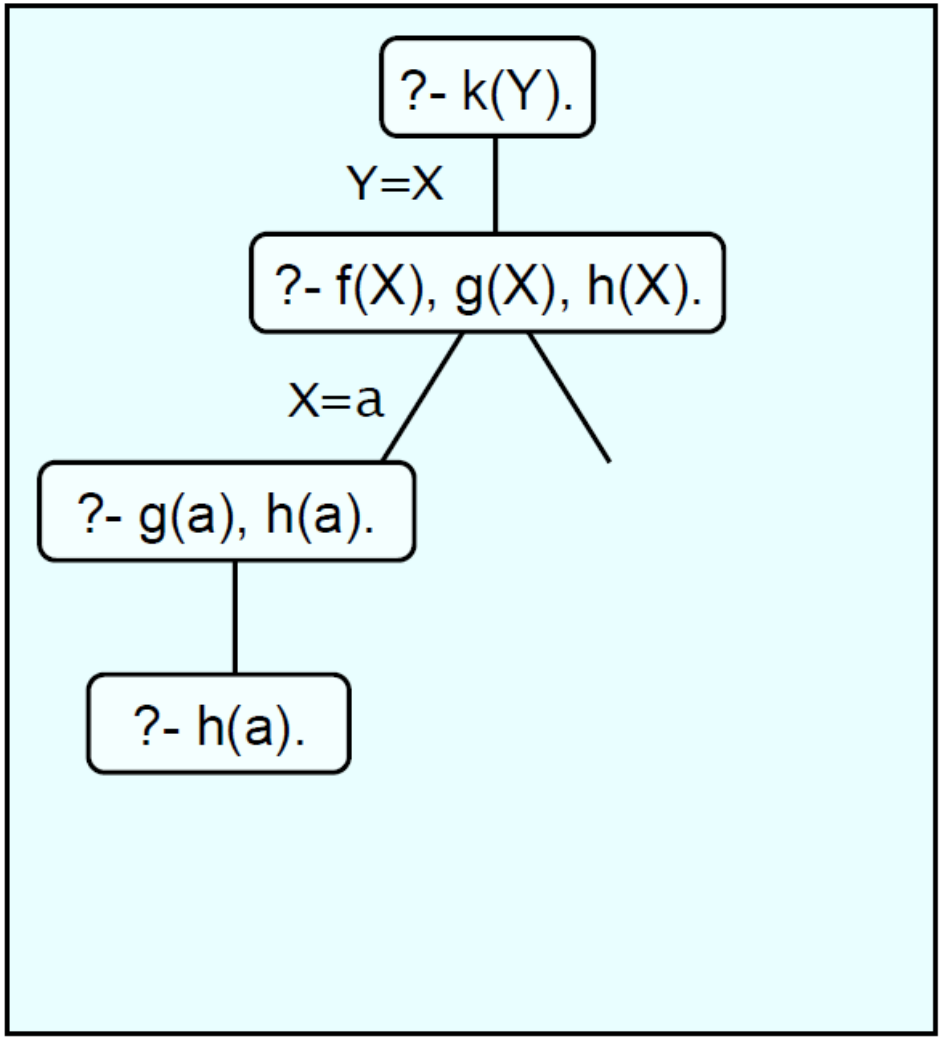
```
?- k(Y).
```



Example: Search Tree

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

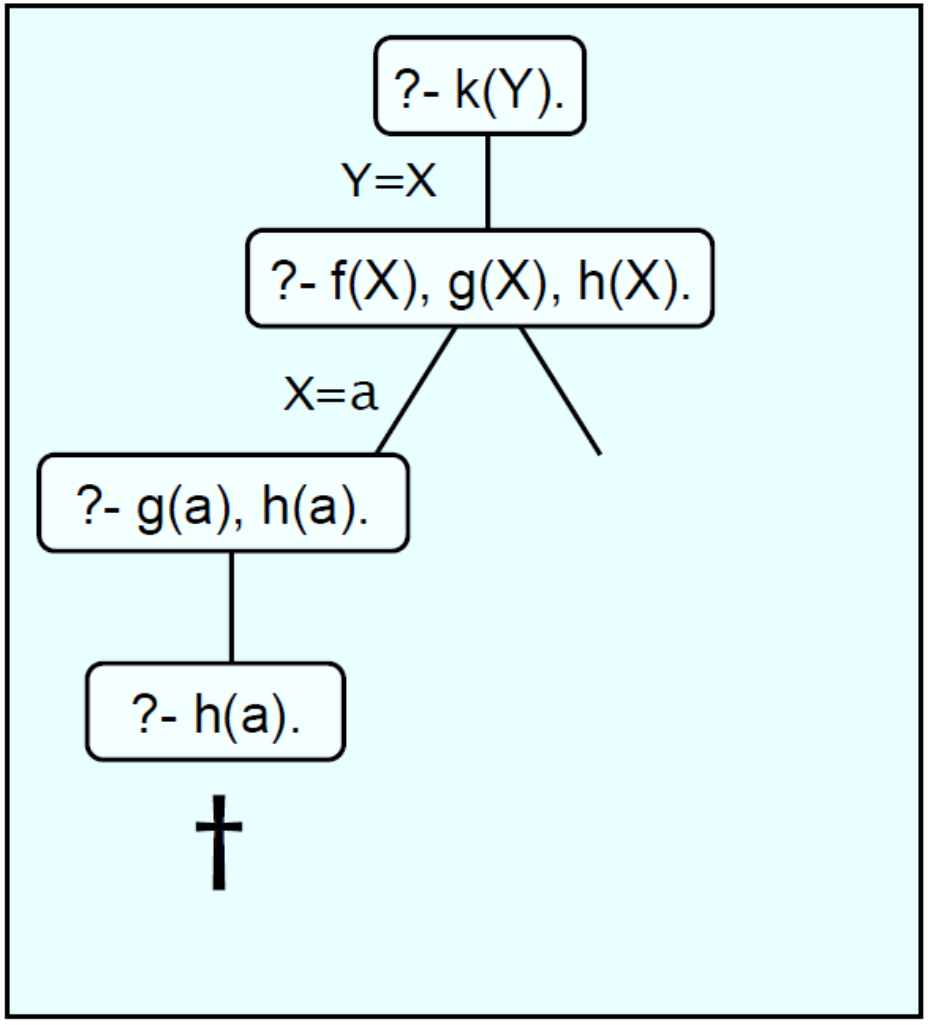
```
?- k(Y).
```



Example: Search Tree

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

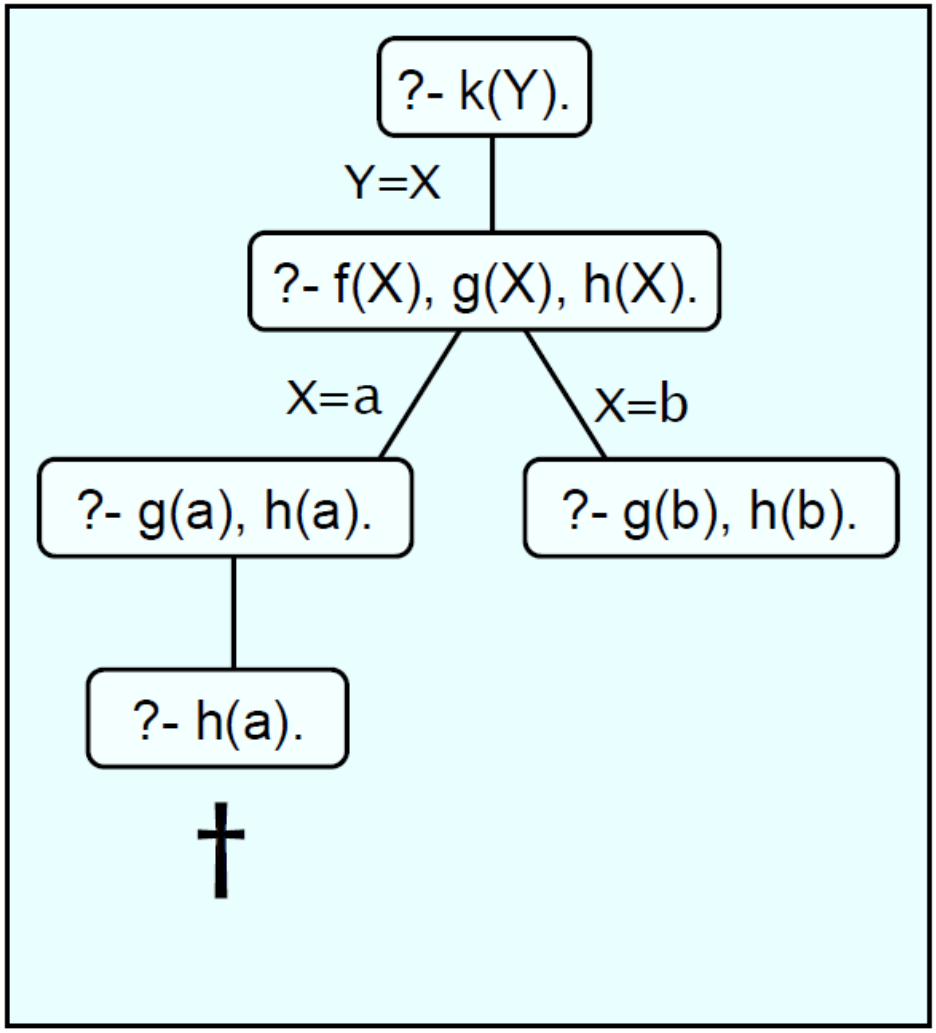
```
?- k(Y).
```



Example: Search Tree

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

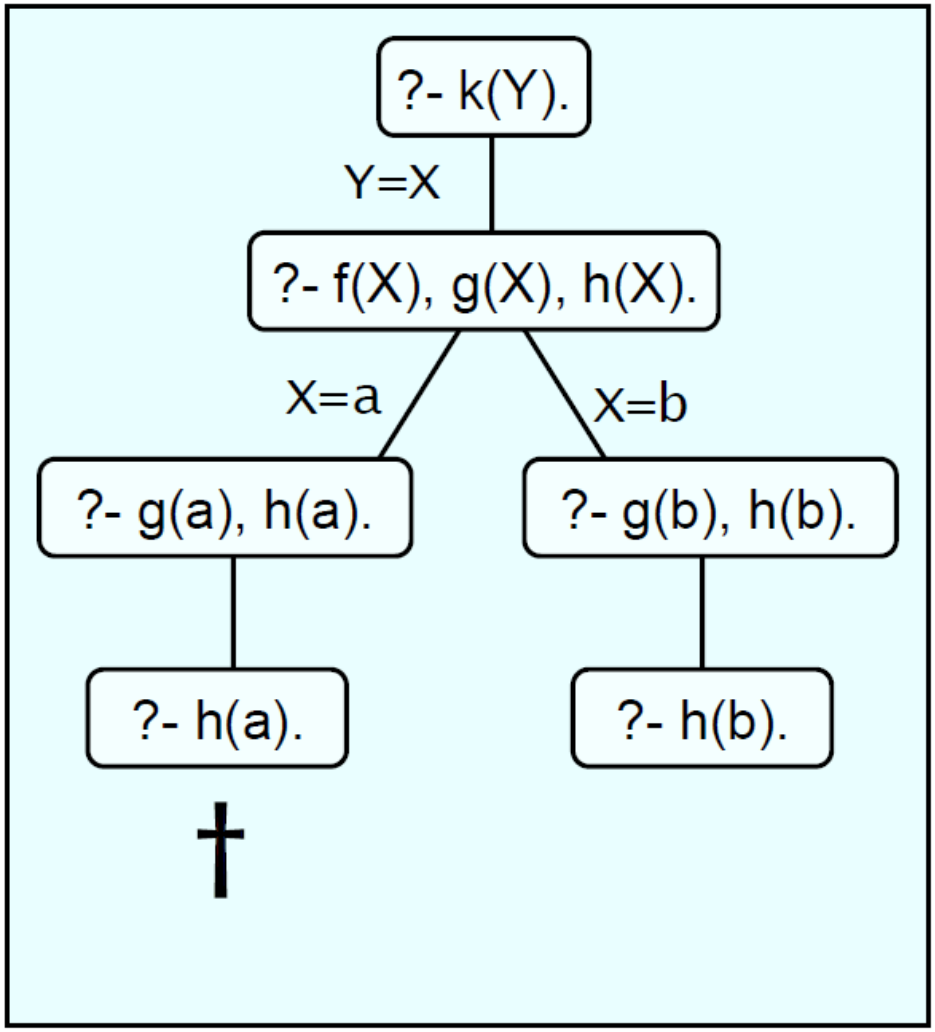
```
?- k(Y).
```



Example: Search Tree

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

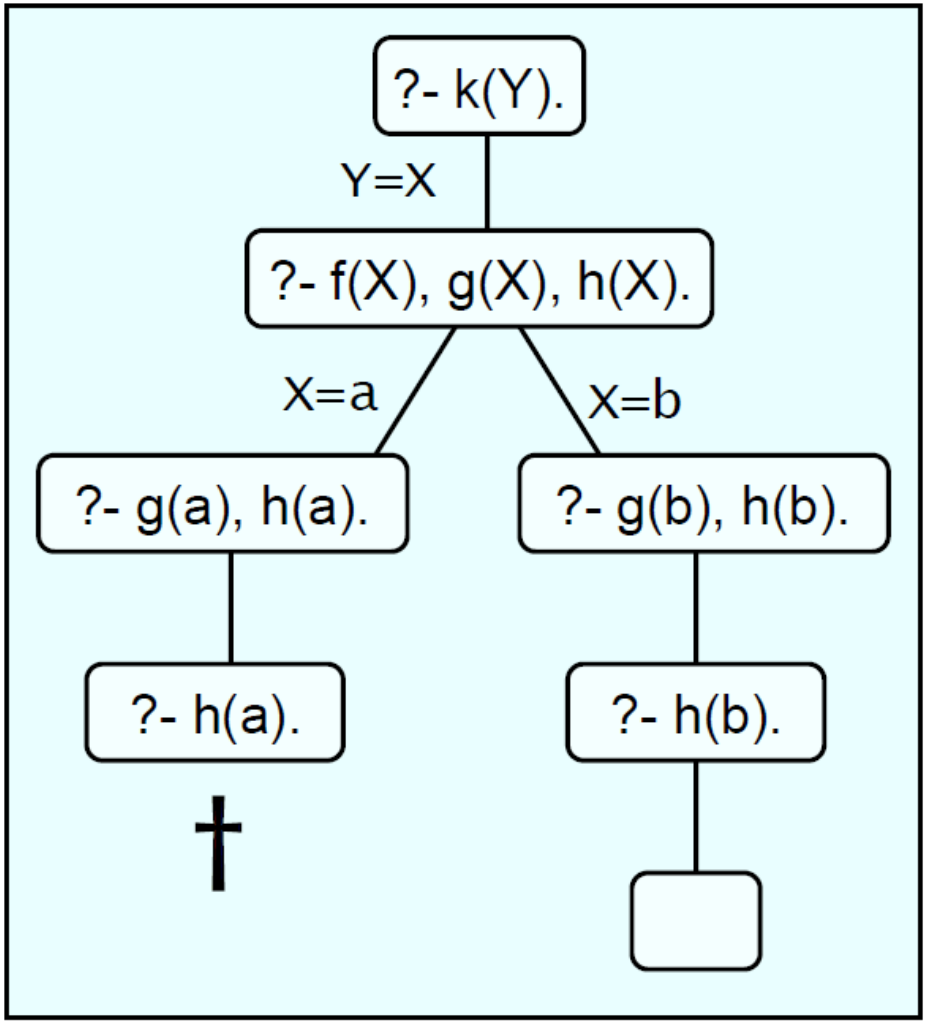
```
?- k(Y).
```



Example: Search Tree

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

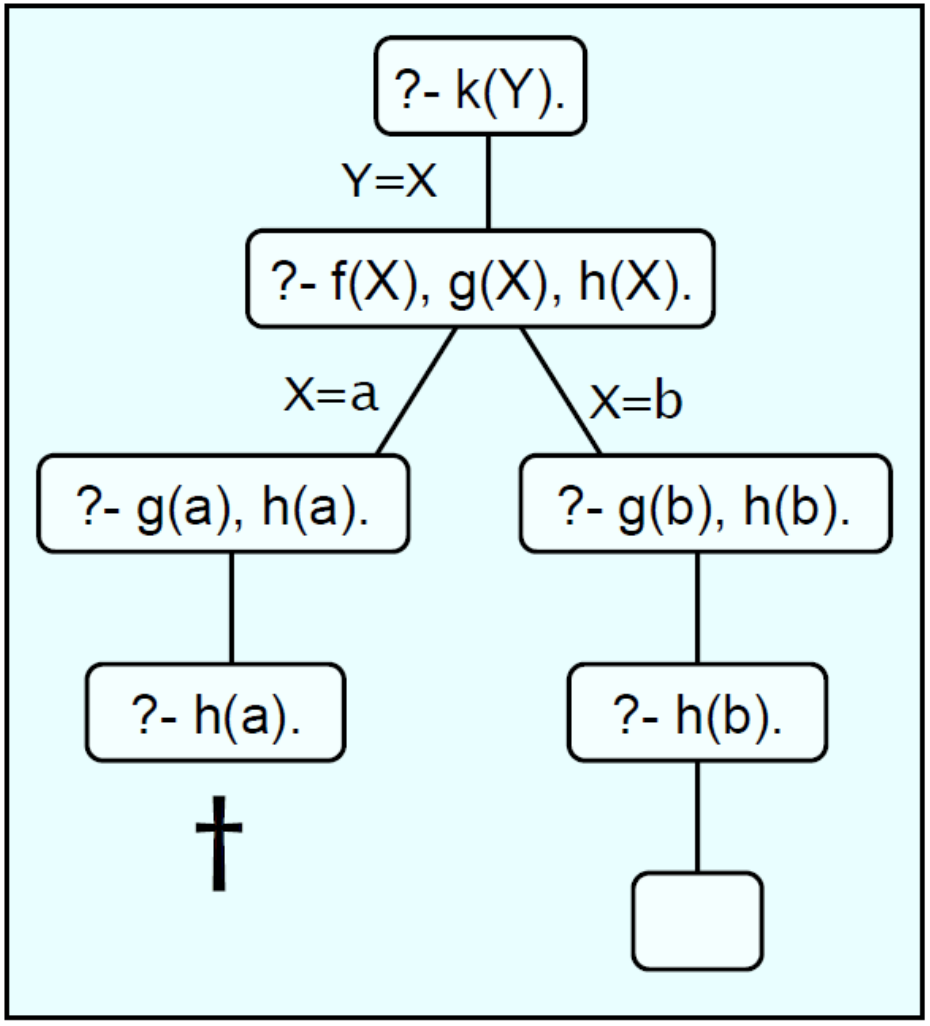
```
?- k(Y).
Y=b
```



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
?-
```



Exercise

We are working with the following knowledge base:

```
house_elf(dobby).
witch(hermione).
witch('McGonagall').
witch(rita_skeeter).
magic(X):- house_elf(X).
magic(X):- wizard(X).
magic(X):- witch(X).
```

Draw the search tree for:

```
?- magic(Hermione).
```

Next…

Recursion

Lists

Arithmetic