



INTRODUCTION TO PROLOG PROGRAMMING

Tutorial for
Programming Languages Laboratory (CS 431)

Indian Institute of Technology Guwahati

Prolog

- **Pro**gramming with **logic**
- Declarative programming paradigm
- Deduces new facts about a situation based on provided facts(**Knowledge Base**)
- Answers queries related to the situation

Prolog: Application

- Useful in many AI applications
 - Expert systems for solving complex problems
 - Decision support system

Reference Material

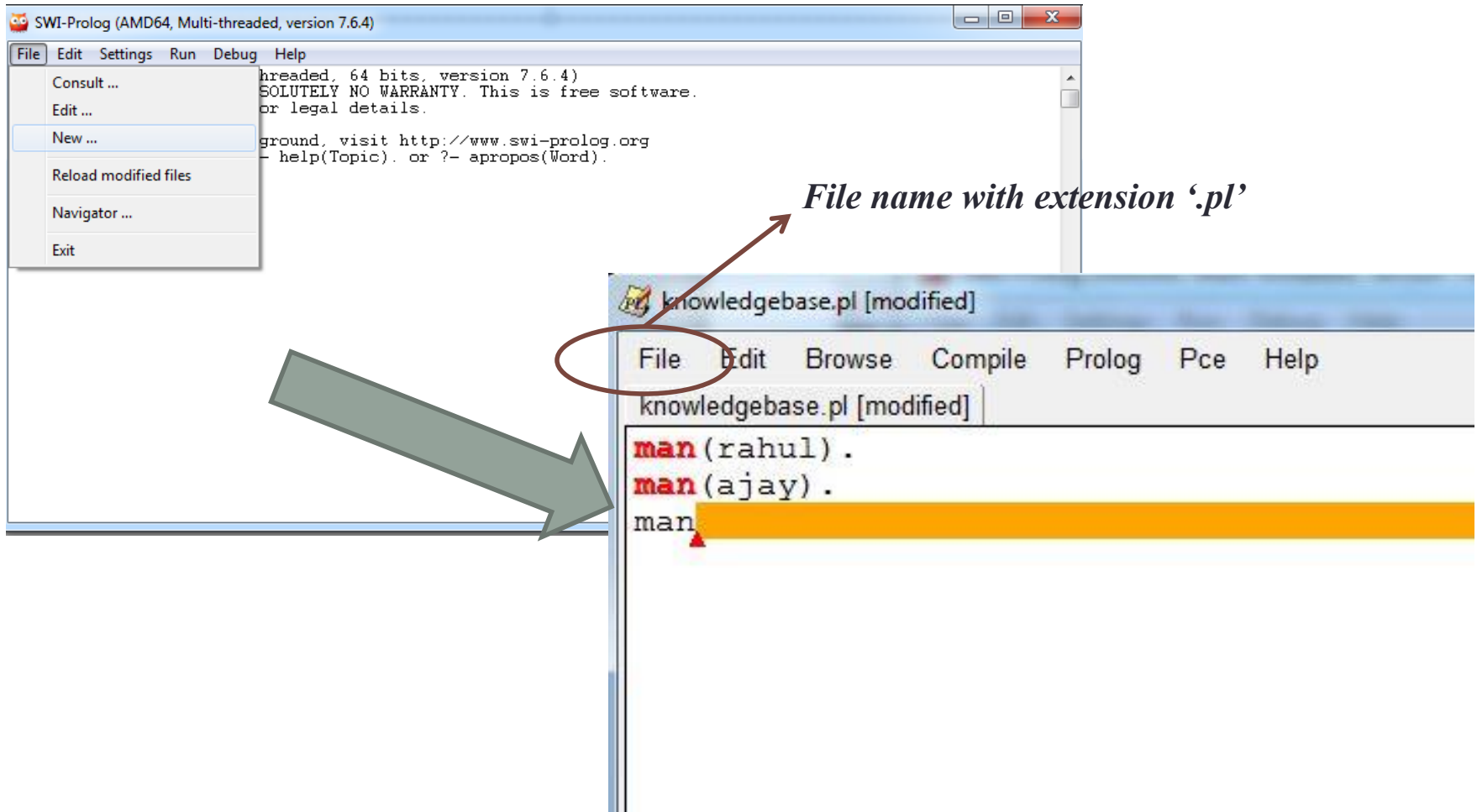
- Blackburn, Patrick, Johannes Bos, and Kristina Striegnitz. “*Learn prolog now!*”, Vol. 7. No. 7. Londres: College Publications, 2006.
- Online Version
<http://www.learnprolognow.org/lpnpage.php?pageid=online>

Prolog Interpreter

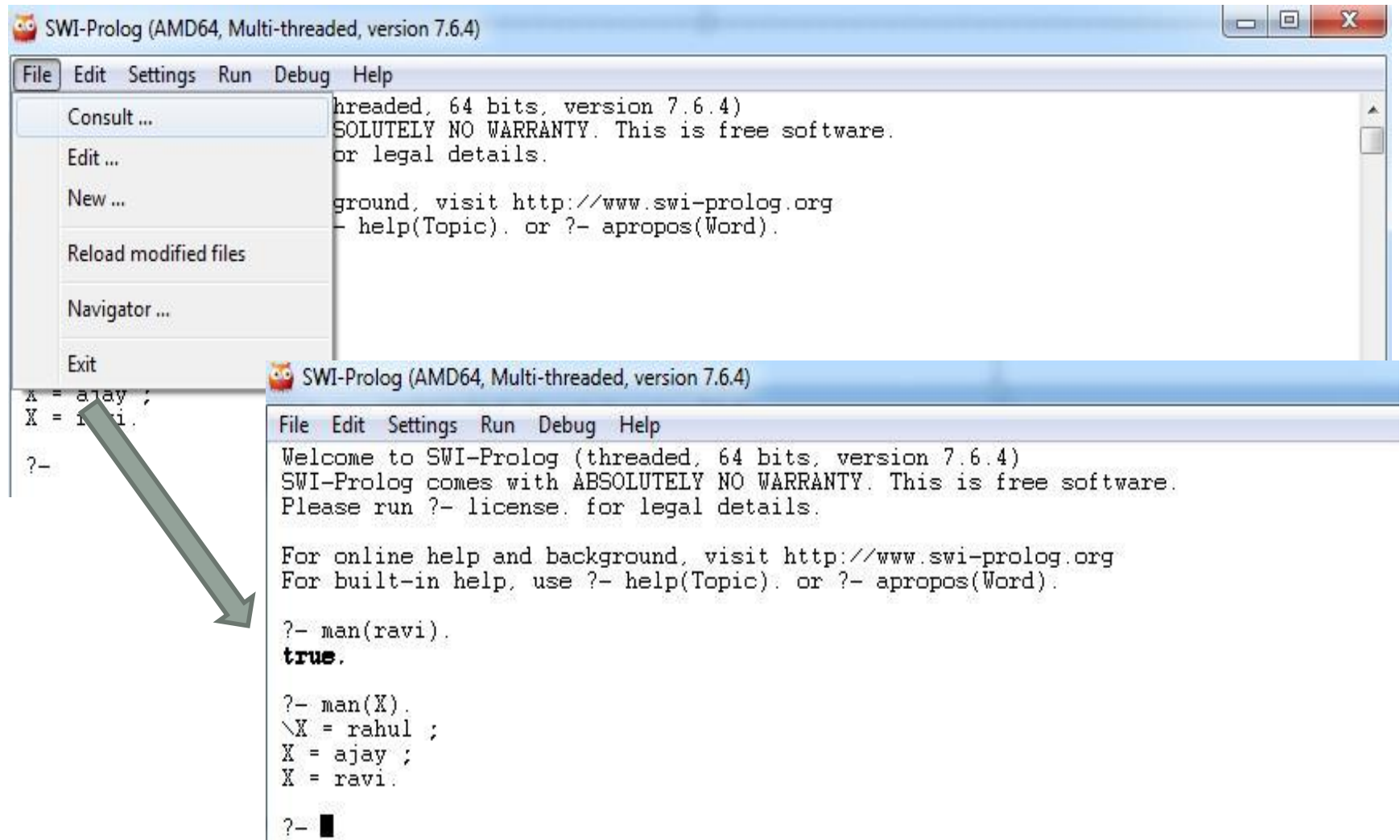
- SWI Prolog
 - Freely available
 - Works with Windows, Linux and Mac OS
 - ISO compliant
- Download interpreter and relevant material:
<http://www.swi-prolog.org/>



SWI Prolog: Basics



SWI Prolog: Basics



The screenshot displays the SWI-Prolog (AMD64, Multi-threaded, version 7.6.4) application window. The 'File' menu is open, showing options: Consult ..., Edit ..., New ..., Reload modified files, Navigator ..., and Exit. The main text area contains the Prolog startup message and help information. A second window, also titled 'SWI-Prolog (AMD64, Multi-threaded, version 7.6.4)', shows the Prolog REPL. A large green arrow points from the 'Exit' option in the first window's File menu to the REPL window.

```
File Edit Settings Run Debug Help
Consult ...
Edit ...
New ...
Reload modified files
Navigator ...
Exit
```

hreaded, 64 bits, version 7.6.4)
ABSOLUTELY NO WARRANTY. This is free software.
for legal details.
ground, visit <http://www.swi-prolog.org>
- help(Topic). or ?- apropos(Word).

```
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 7.6.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.


For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- man(ravi).
true.

?- man(X).
\X = rahul ;
X = ajay ;
X = ravi.

?-
```


Knowledge Base

 knowledgebase.pl

File Edit Browse Compile Prolog Pce Help

knowledgebase.pl

```
man(rahul).  
man(ajay).  
man(ravi).  
swim(ajay).  
dance.
```

 SWI-Prolog (AMD64, Multi-threaded, version 7.6.4)

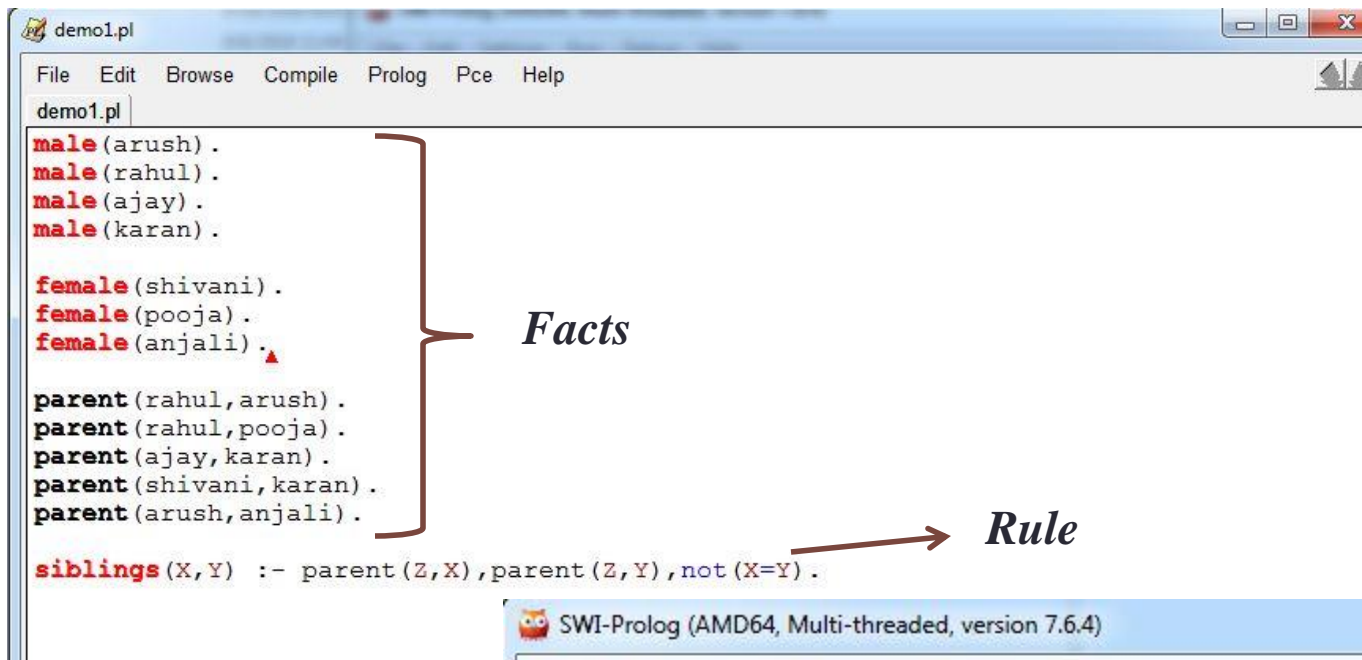
File Edit Settings Run Debug Help

```
?- man(rahul).  
true.  
  
?- man(ajay).  
true.  
  
?- man(atif).  
false.  
  
?- swim(ravi).  
false.  
  
?- swim(ajay).  
true.  
  
?- dance.  
true.  
  
?- dance(ajay).  
ERROR: Undefined procedure: dance/1  
ERROR:         However, there are definitions for:  
ERROR:         dance/0  
false.  
  
?- play(ajay).  
ERROR: Undefined procedure: play/1 (DWIM could not correct goal)  
?-
```


Basic structure of a Prolog program

- Three basic constructs of Prolog:
 - Facts
 - Rules
 - Queries

Basic structure of a Prolog program



```
demo1.pl
File Edit Browse Compile Prolog Pce Help
demo1.pl
male(arush).
male(rahul).
male(ajay).
male(karan).

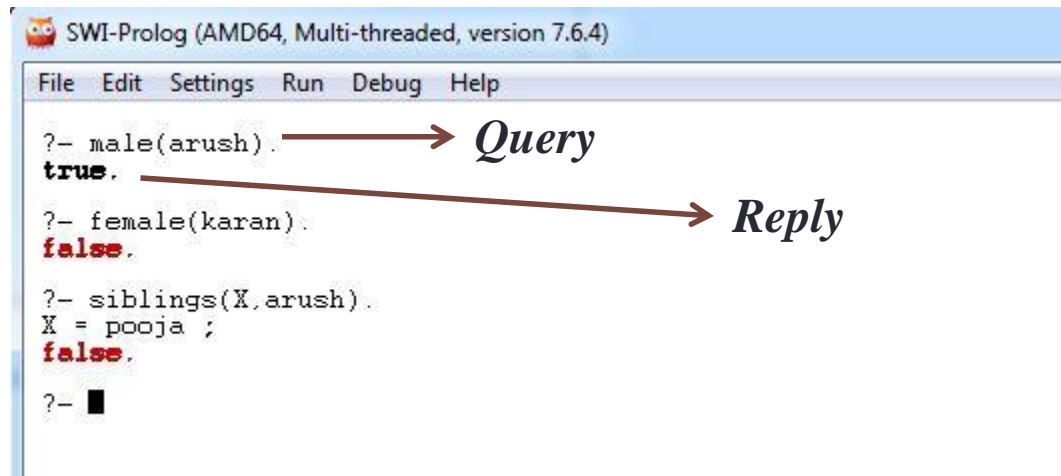
female(shivani).
female(pooja).
female(anjali).

parent(rahul,arush).
parent(rahul,pooja).
parent(ajay,karan).
parent(shivani,karan).
parent(arush,anjali).

siblings(X,Y) :- parent(Z,X),parent(Z,Y),not(X=Y).
```

Facts

Rule



```
SWI-Prolog (AMD64, Multi-threaded, version 7.6.4)
File Edit Settings Run Debug Help

?- male(arush).
true.

?- female(karan).
false.

?- siblings(X,arush).
X = pooja ;
false.

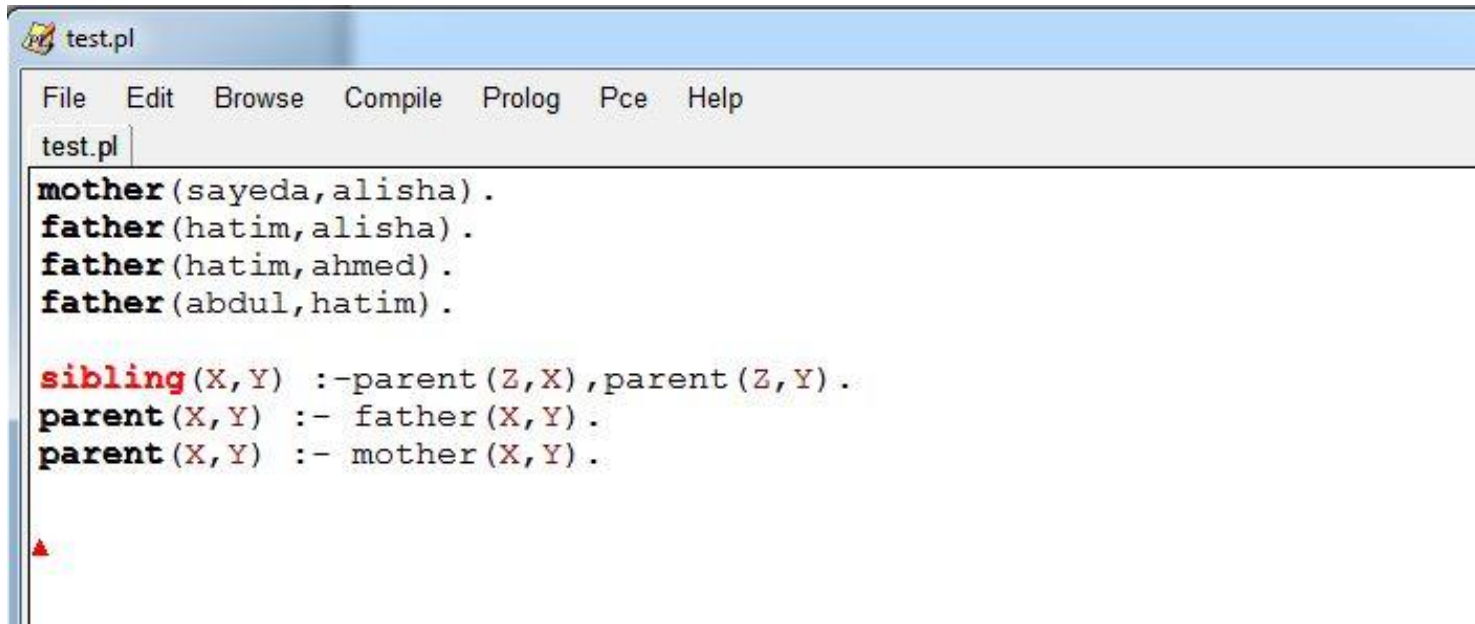
?-
```

Query

Reply

Clauses

- Clause:
 - Fact (Base Clause)
 - Rule

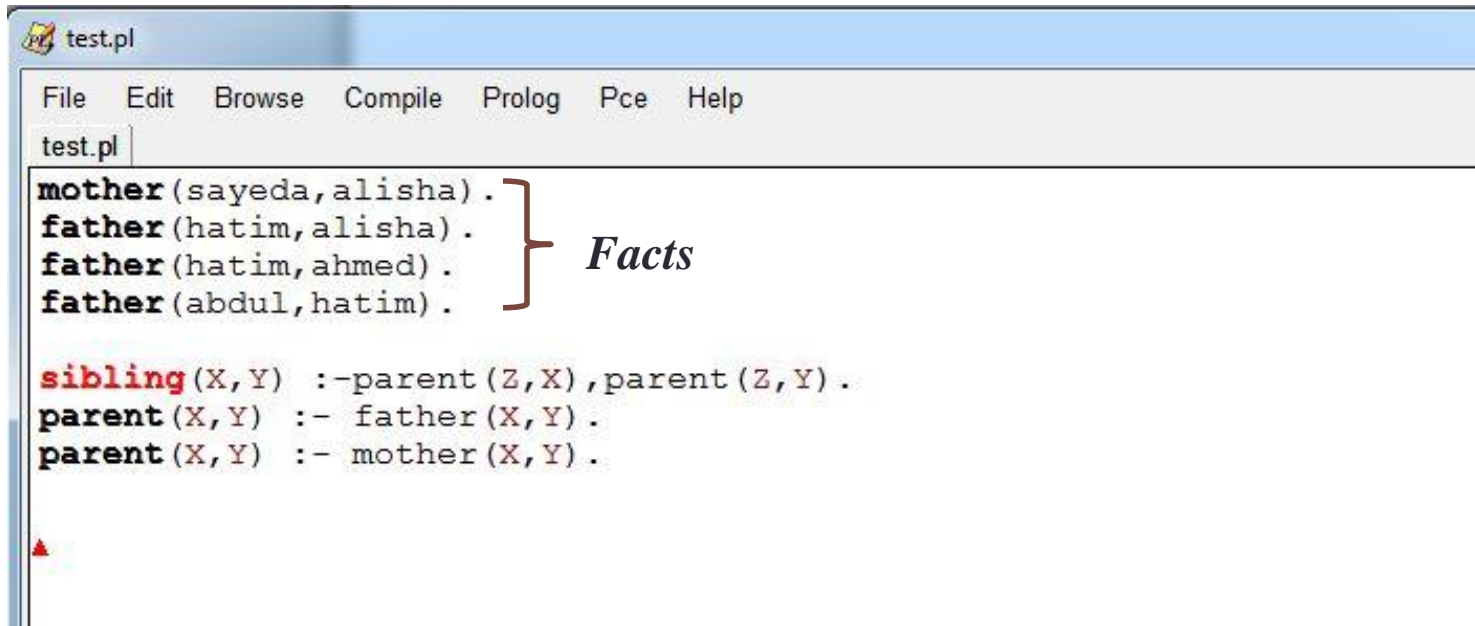


```
test.pl
File Edit Browse Compile Prolog Pce Help
test.pl
mother(sayed,aisha).
father(hatim,aisha).
father(hatim,ahmed).
father(abdul,hatim).

sibling(X,Y) :-parent(Z,X),parent(Z,Y).
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
```

Clauses

- Clause:
 - Fact (Base Clause)
 - Rule



```
test.pl
File Edit Browse Compile Prolog Pce Help
test.pl
mother(sayed,a,alisha).
father(hatim,a,alisha).
father(hatim,a,ahmed).
father(abdul,h,hatim).
} Facts

sibling(X,Y) :-parent(Z,X),parent(Z,Y).
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
```

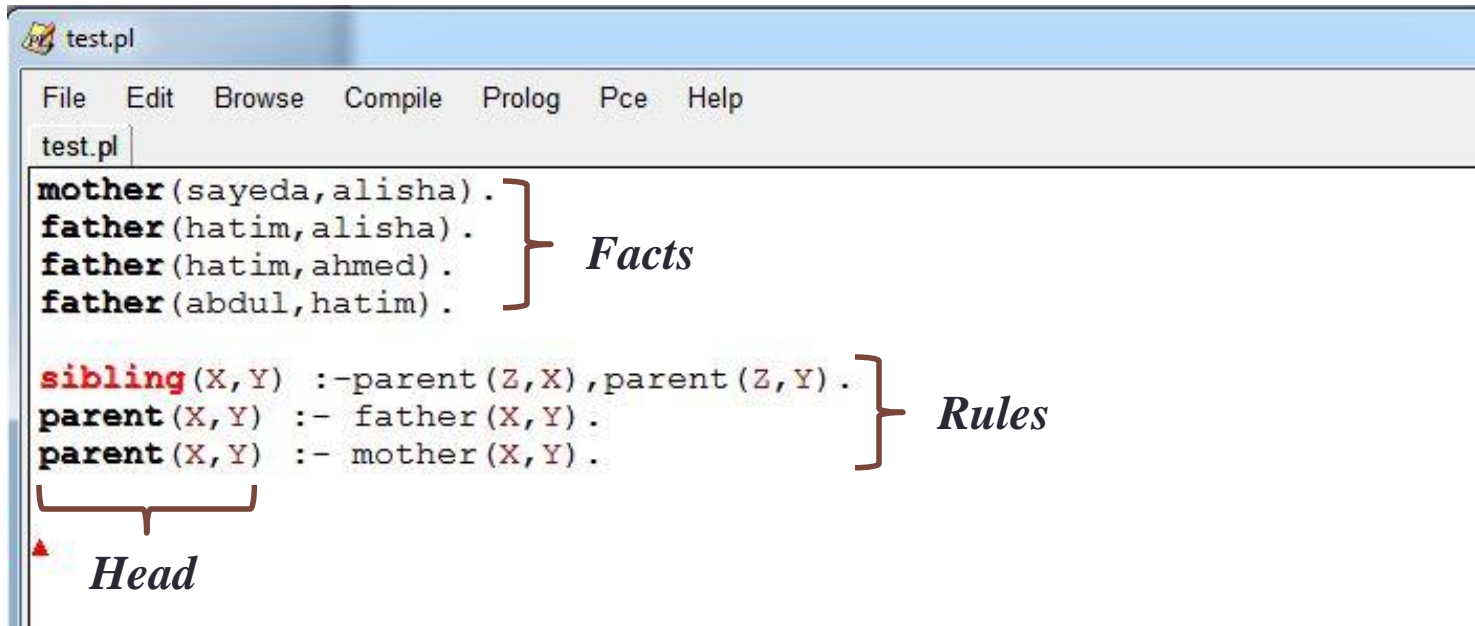
Clauses

- Clause:
 - Fact (Base Clause)
 - Rule

```
test.pl
File Edit Browse Compile Prolog Pce Help
test.pl
mother(sayed, alisha).
father(hatim, alisha).
father(hatim, ahmed).
father(abdul, hatim).
sibling(X, Y) :- parent(Z, X), parent(Z, Y).
parent(X, Y) :- father(X, Y).
parent(X, Y) :- mother(X, Y).
```

Clauses

- Clause:
 - Fact (Base Clause)
 - Rule



```
test.pl
File Edit Browse Compile Prolog Pce Help
test.pl
mother(sayed,alisha).
father(hatim,alisha).
father(hatim,ahmed).
father(abdul,hatim).

sibling(X,Y) :-parent(Z,X),parent(Z,Y).
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
```

The screenshot shows a Prolog editor window titled "test.pl" with a menu bar (File, Edit, Browse, Compile, Prolog, Pce, Help). The code is as follows:

```
test.pl
File Edit Browse Compile Prolog Pce Help
test.pl
mother(sayed,alisha).
father(hatim,alisha).
father(hatim,ahmed).
father(abdul,hatim).

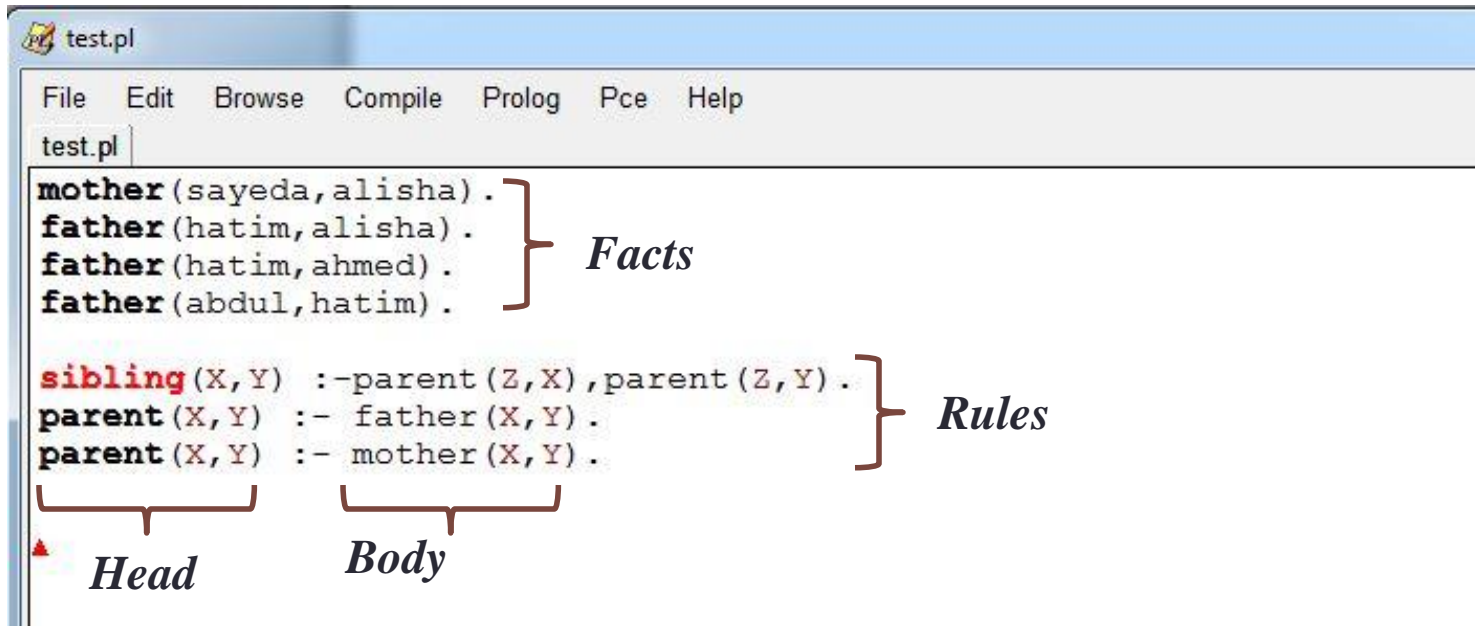
sibling(X,Y) :-parent(Z,X),parent(Z,Y).
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
```

Annotations in the image:

- A bracket on the right groups the first four lines (the facts) under the label *Facts*.
- A bracket on the right groups the last three lines (the rules) under the label *Rules*.
- A bracket on the left under the first rule's head `sibling(X,Y)` is labeled *Head*.

Clauses

- Clause:
 - Fact (Base Clause)
 - Rule



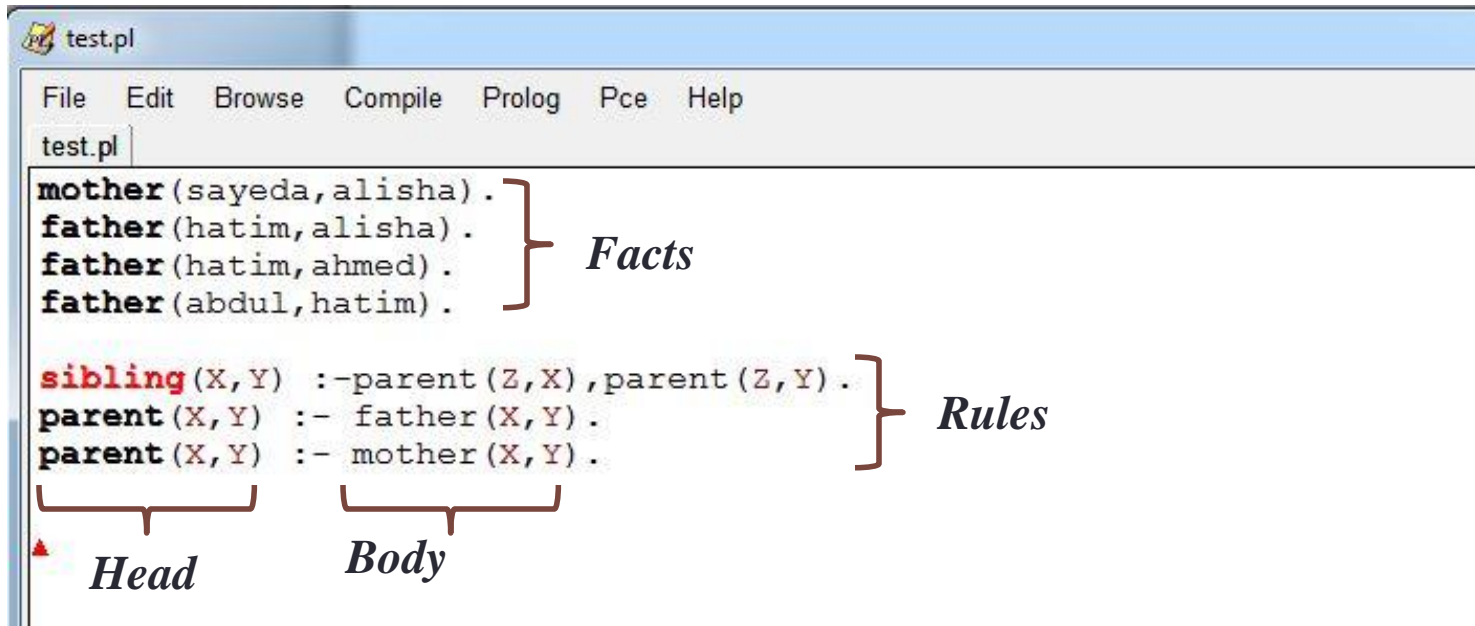
```
test.pl
File Edit Browse Compile Prolog Pce Help
test.pl
mother(sayed,alisha).
father(hatim,alisha).
father(hatim,ahmed).
father(abdul,hatim).

sibling(X,Y) :-parent(Z,X),parent(Z,Y).
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
```

The screenshot shows a Prolog editor window titled "test.pl". The code contains four clauses. The first three are facts: `mother(sayed,alisha).`, `father(hatim,alisha).`, `father(hatim,ahmed).`, and `father(abdul,hatim).`. These are grouped by a right curly bracket labeled *Facts*. The next three are rules: `sibling(X,Y) :-parent(Z,X),parent(Z,Y).`, `parent(X,Y) :- father(X,Y).`, and `parent(X,Y) :- mother(X,Y).`. These are grouped by a right curly bracket labeled *Rules*. Within the rules, the first part of each rule (e.g., `sibling(X,Y)`) is bracketed and labeled *Head*, and the part after the colon (e.g., `:-parent(Z,X),parent(Z,Y).`) is bracketed and labeled *Body*.

Clauses

- Clause:
 - Fact (Base Clause)
 - Rule



```
test.pl
File Edit Browse Compile Prolog Pce Help
test.pl
mother(sayed,alisha).
father(hatim,alisha).
father(hatim,ahmed).
father(abdul,hatim).

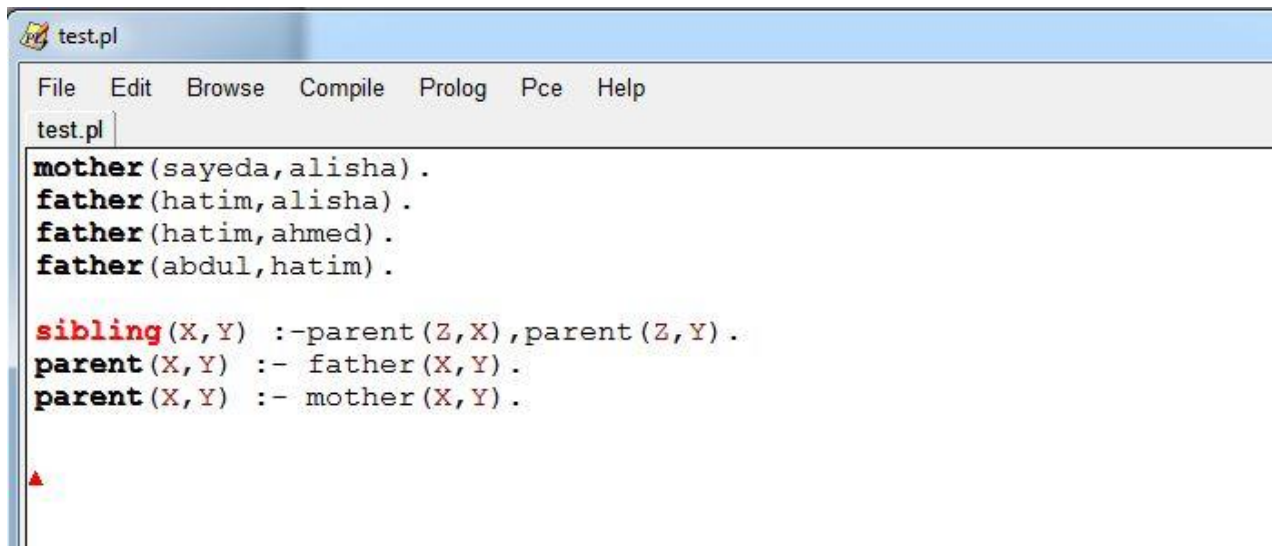
sibling(X,Y) :-parent(Z,X),parent(Z,Y).
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
```

The screenshot shows a Prolog editor window titled "test.pl" with a menu bar (File, Edit, Browse, Compile, Prolog, Pce, Help). The code contains four facts and three rules. Annotations include a right curly brace labeled "Facts" grouping the four fact clauses, and a right curly brace labeled "Rules" grouping the three rule clauses. Below the rules, two horizontal curly braces are shown: the first under the first rule is labeled "Head" and points to the predicate "sibling"; the second under the two subsequent rules is labeled "Body" and points to the body of those rules.

7 Clauses: 4 Facts and 3 Rules

Predicates

- A prolog program consists of one or more predicates
- Predicates are clauses having the same **functor** name and **arity**

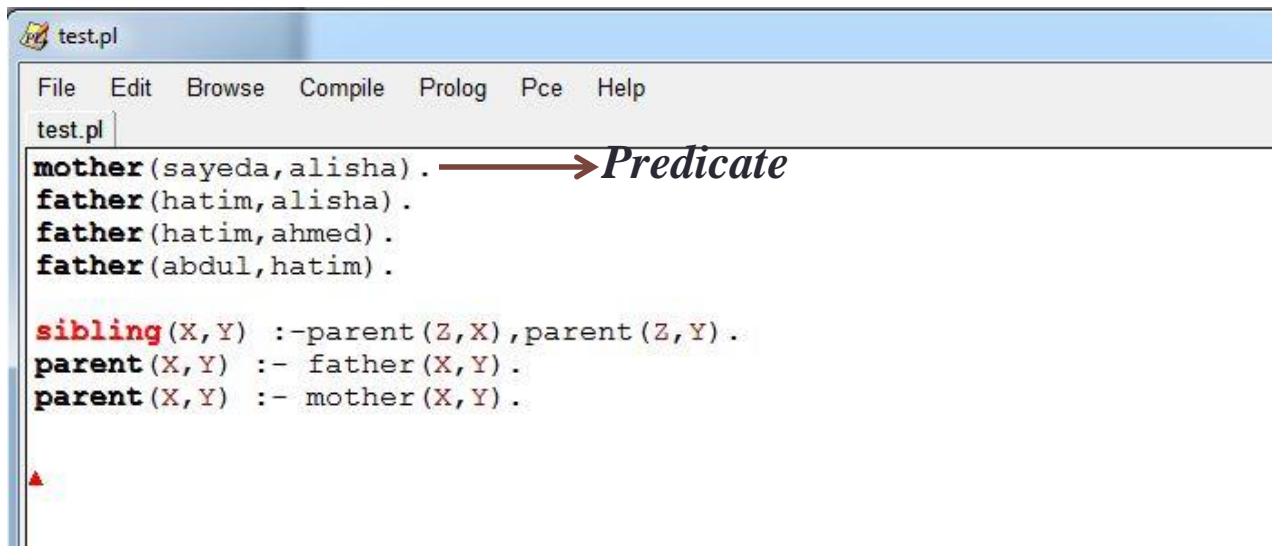


```
test.pl
File Edit Browse Compile Prolog Pce Help
test.pl
mother(sayed,a,alisha).
father(hatim,a,alisha).
father(hatim,a,ahmed).
father(abdul,h,hatim).

sibling(X,Y) :-parent(Z,X),parent(Z,Y).
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
```

Predicates

- A prolog program consists of one or more predicates
- Predicates are clauses having the same **functor** name and **arity**

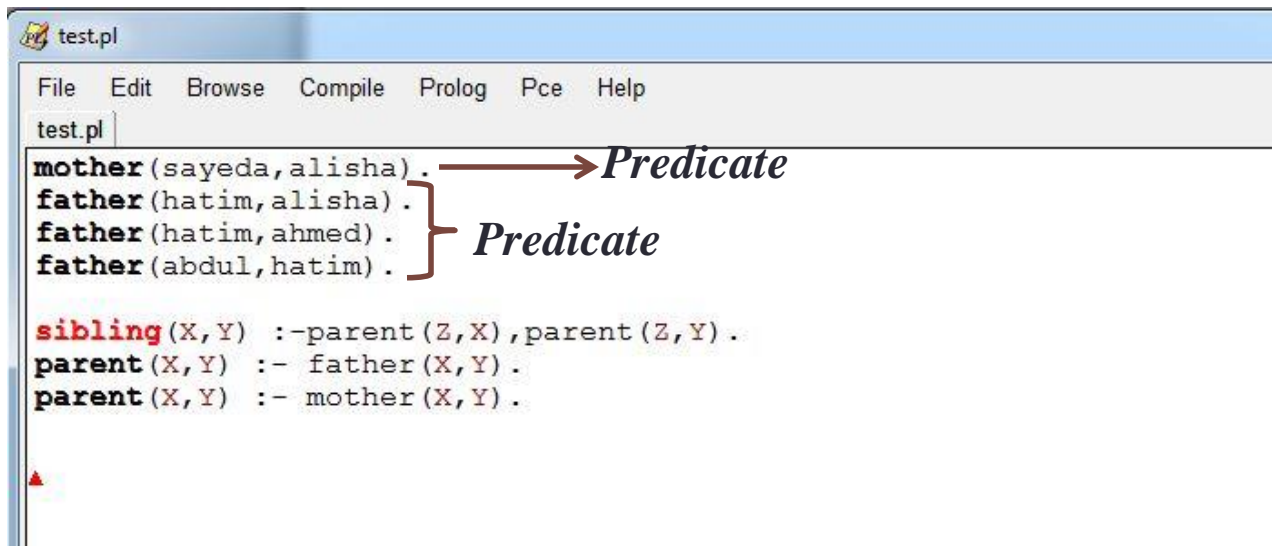


```
test.pl
File Edit Browse Compile Prolog Pce Help
test.pl
mother(sayeda,alisha). —————→ Predicate
father(hatim,alisha).
father(hatim,ahmed).
father(abdul,hatim).

sibling(X,Y) :-parent(Z,X),parent(Z,Y).
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
```

Predicates

- A prolog program consists of one or more predicates
- Predicates are clauses having the same **functor** name and **arity**

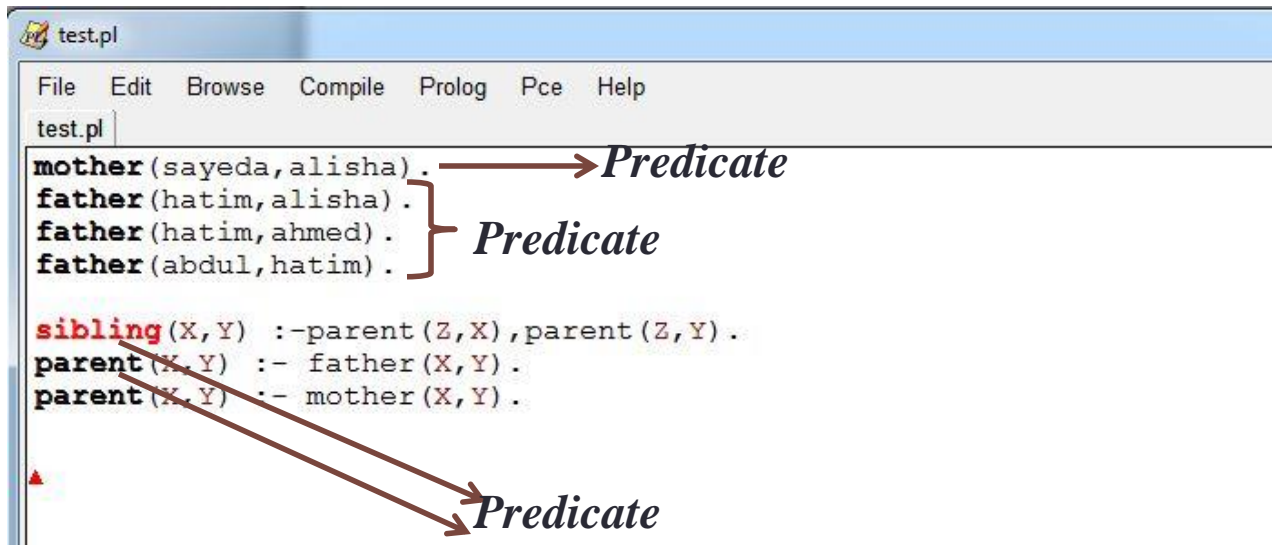


```
test.pl
File Edit Browse Compile Prolog Pce Help
test.pl
mother(sayed,alisha).
father(hatim,alisha).
father(hatim,ahmed).
father(abdul,hatim).
sibling(X,Y) :-parent(Z,X),parent(Z,Y).
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
```

The screenshot shows a Prolog editor window titled "test.pl". The code defines several predicates: `mother(sayed,alisha).`, `father(hatim,alisha).`, `father(hatim,ahmed).`, `father(abdul,hatim).`, `sibling(X,Y) :-parent(Z,X),parent(Z,Y).`, `parent(X,Y) :- father(X,Y).`, and `parent(X,Y) :- mother(X,Y).`. Annotations with arrows and a bracket point to the functor names and their arities: `mother(2)`, `father(2)`, and `sibling(2)` are labeled as "Predicate".

Predicates

- A prolog program consists of one or more predicates
- Predicates are clauses having the same **functor** name and **arity**



```
test.pl
File Edit Browse Compile Prolog Pce Help
test.pl
mother(sayed,alisha).
father(hatim,alisha).
father(hatim,ahmed).
father(abdul,hatim).

sibling(X,Y) :-parent(Z,X),parent(Z,Y).
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
```

The image shows a Prolog IDE window titled "test.pl". The code contains several predicate definitions. Annotations with arrows and a bracket identify the predicates:

- An arrow points from the word **Predicate** to the `mother(sayed,alisha).` line.
- A bracket groups the `father(hatim,alisha).`, `father(hatim,ahmed).`, and `father(abdul,hatim).` lines, with an arrow pointing from the word **Predicate** to the bracket.
- Two arrows point from the word **Predicate** to the `sibling(X,Y) :-parent(Z,X),parent(Z,Y).` and `parent(X,Y) :- father(X,Y).` lines.

Predicates

- A prolog program consists of one or more predicates
- Predicates are clauses having the same **functor** name and **arity**

```
test.pl
File Edit Browse Compile Prolog Pce Help
test.pl
mother(sayed,alisha).
father(hatim,alisha).
father(hatim,ahmed).
father(abdul,hatim).

sibling(X,Y) :-parent(Z,X),parent(Z,Y).
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
```

Predicate

Predicate

Functor

Predicate

Predicates

- A prolog program consists of one or more predicates
- Predicates are clauses having the same **functor** name and **arity**

```
test.pl
File Edit Browse Compile Prolog Pce Help
test.pl
mother(sayed, alisha).
father(hatim, alisha).
father(hatim, ahmed).
father(abdul, hatim).
sibling(X,Y) :- parent(Z,X), parent(Z,Y).
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
```

Annotations in the image:

- Arrow from `mother(sayed, alisha).` to *Predicate*
- Brace from `father(hatim, alisha).`, `father(hatim, ahmed).`, and `father(abdul, hatim).` to *Predicate*
- Arrow from `parent(Z,Y)` in `sibling(X,Y) :- parent(Z,X), parent(Z,Y).` to *arity=2*
- Brace from `parent(X,Y) :- father(X,Y).` and `parent(X,Y) :- mother(X,Y).` to *Functor*
- Arrow from `parent` in the rules to `parent` in the `sibling` rule to *Predicate*

Predicates

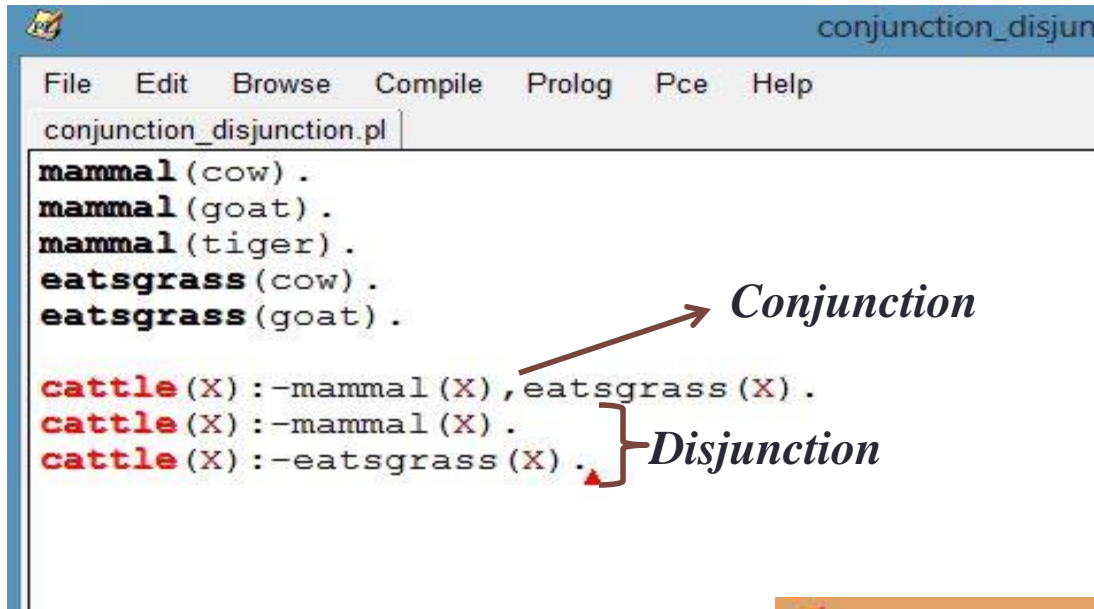
- A prolog program consists of one or more predicates
- Predicates are clauses having the same **functor** name and **arity**

```
test.pl
File Edit Browse Compile Prolog Pce Help
test.pl
mother(sayed, alisha).
father(hatim, alisha).
father(hatim, ahmed).
father(abdul, hatim).

sibling(X, Y) :- parent(Z, X), parent(Z, Y).
parent(X, Y) :- father(X, Y).
parent(X, Y) :- mother(X, Y).
```

- In the above program, there are 4 predicates

Conjunction and Disjunction

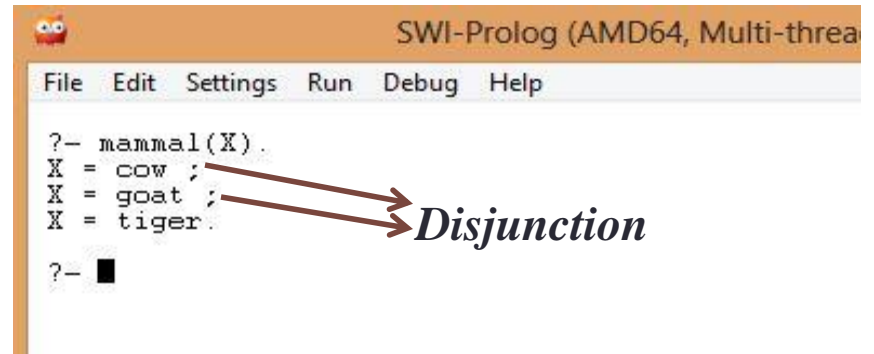


```
conjunction_disjunction.pl
mammal(cow).
mammal(goat).
mammal(tiger).
eatsgrass(cow).
eatsgrass(goat).

cattle(X):-mammal(X),eatsgrass(X).
cattle(X):-mammal(X).
cattle(X):-eatsgrass(X).
```

Conjunction

Disjunction



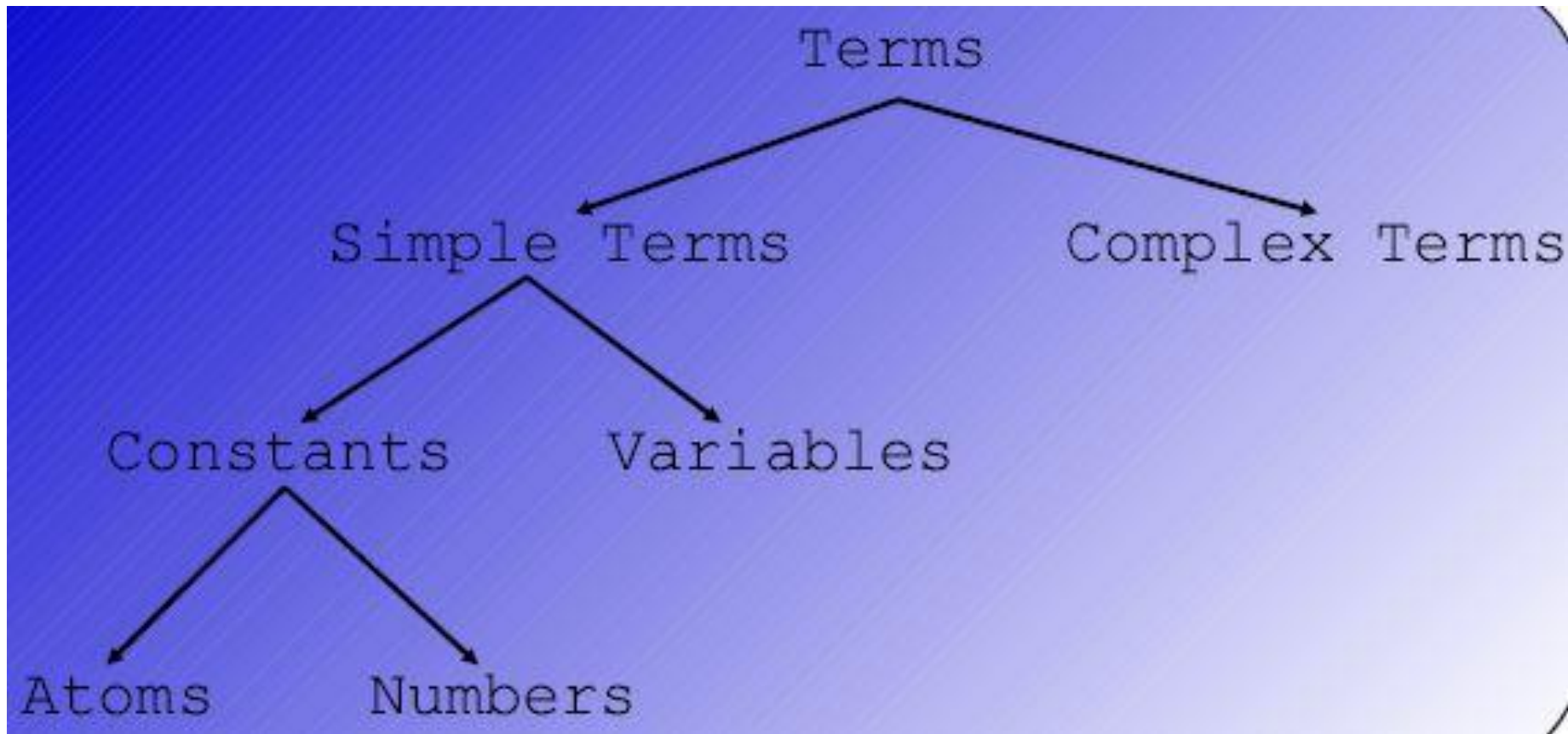
```
SWI-Prolog (AMD64, Multi-threaded)
File Edit Settings Run Debug Help

?- mammal(X).
X = cow ;
X = goat ;
X = tiger ;
?-
```

Disjunction

Dissecting Prolog Constructs

- Facts, Rules and Queries are made up of terms.



Atoms

- A sequence of characters of upper-case letters, lower-case letters, digits, or underscore, starting with a **lowercase letter**
 - eg. **animal, likes_eating_flesh, loves2Play**
- An arbitrary sequence of characters enclosed in single quotes
 - eg. **'Manish', 'One man army', '\$#*%'**
- A sequence of special characters
 - *eg.* **: , ; . :-**

Numbers

- Integer

81,133,-48

- Float

765.2344

Variables

- A sequence of characters of uppercase letters, lower-case letters, digits, or underscore, starting with either an **uppercase letter** or an **underscore**
 - eg. X,Y, Amount, _score

Complex terms

- Complex terms is made up of atoms, numbers and variables
- It consists of a functor followed by a sequence of arguments
- Arguments are put in round brackets, separated by commas
- The functor must be an atom

eg. `playsFootball(Sameer)`

`loves(Ayesha,Danish)`

Nested complex terms

`works(X, mother(mother(bina)))`

Arity

- Number of arguments in a complex term
 - `man(rahul)` \rightarrow arity=1
 - `parent(johan,isabela)` \rightarrow arity=2, and so on
- Representation:
- `loves(mohan, anjali)` \rightarrow loves/2
- In prolog if two predicates have same functor but different arity they are treated as different predicates

Instantiation

- It is a process of binding a value to a variable
eg. $X = \text{rahul}$ (variable X is bound to the value 'rahul')
 $Y = 3 + 2$ (variable Y is bound to the value $3 + 2$)
- Fully instantiated:- if the bounded value does not contain variable
eg. $X = \text{mother}(\text{rohan})$
- Partially instantiated:- if the bounded value contains variable
eg. $X = \text{father}(Y)$

Unification

- Unification is a pattern matching mechanism in prolog.

term X = term Y (= Unification)

- **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

- Unification is a pattern matching mechanism in prolog.

term $X = \text{term } Y$ (= Unification)

- **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

- Unification is a pattern matching mechanism in prolog.

term X = term Y (= Unification)

- **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

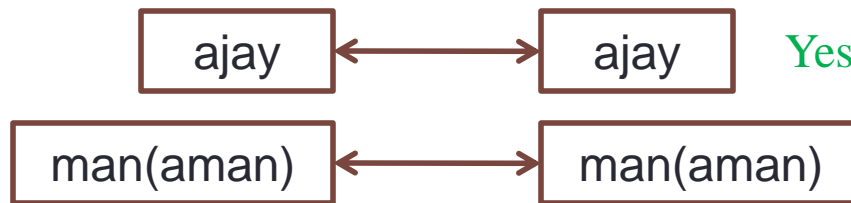


Unification

- Unification is a pattern matching mechanism in prolog.

term X = term Y (= Unification)

- 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

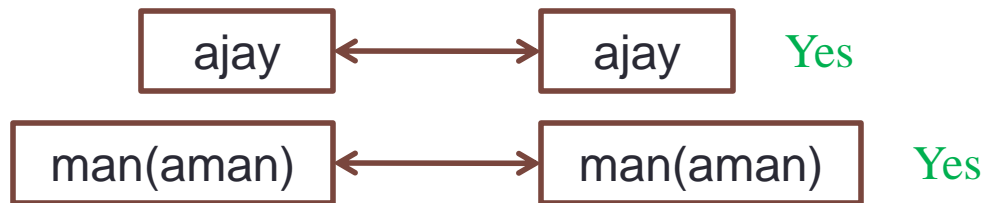


Unification

- Unification is a pattern matching mechanism in prolog.

term X = term Y (= Unification)

- 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

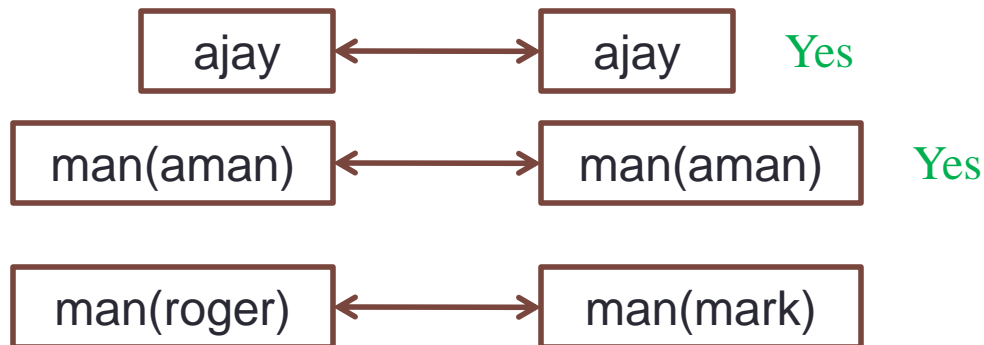


Unification

- Unification is a pattern matching mechanism in prolog.

term X = term Y (= Unification)

- 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

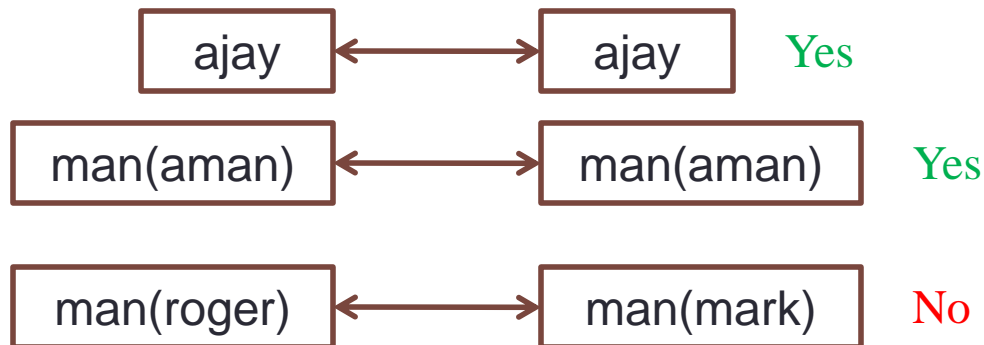


Unification

- Unification is a pattern matching mechanism in prolog.

term X = term Y (= Unification)

- 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

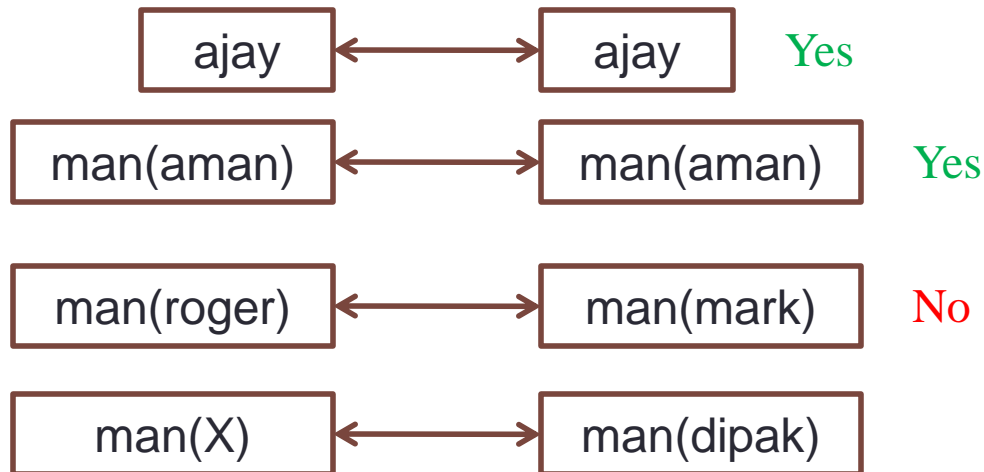


Unification

- Unification is a pattern matching mechanism in prolog.

term $X = \text{term } Y$ (= Unification)

- 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

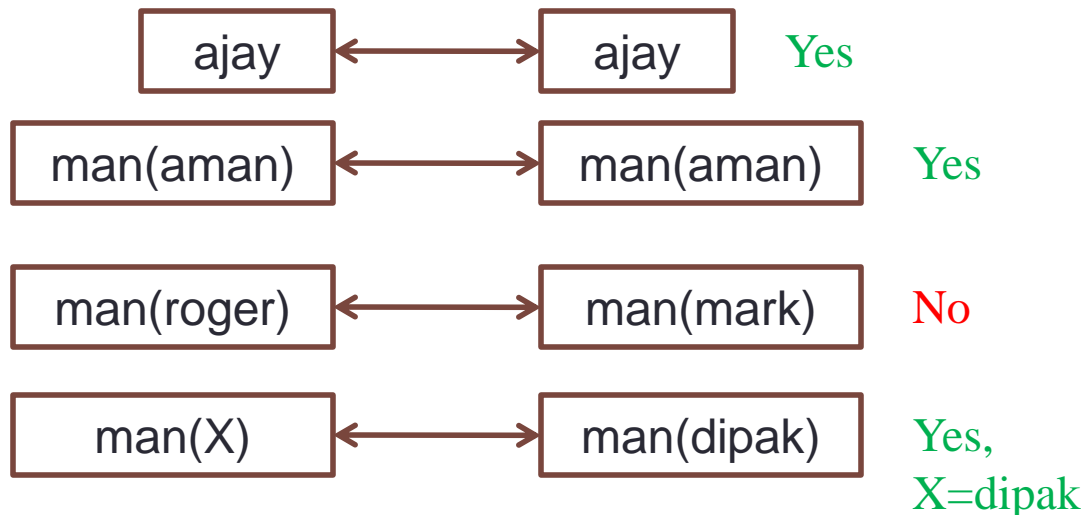


Unification

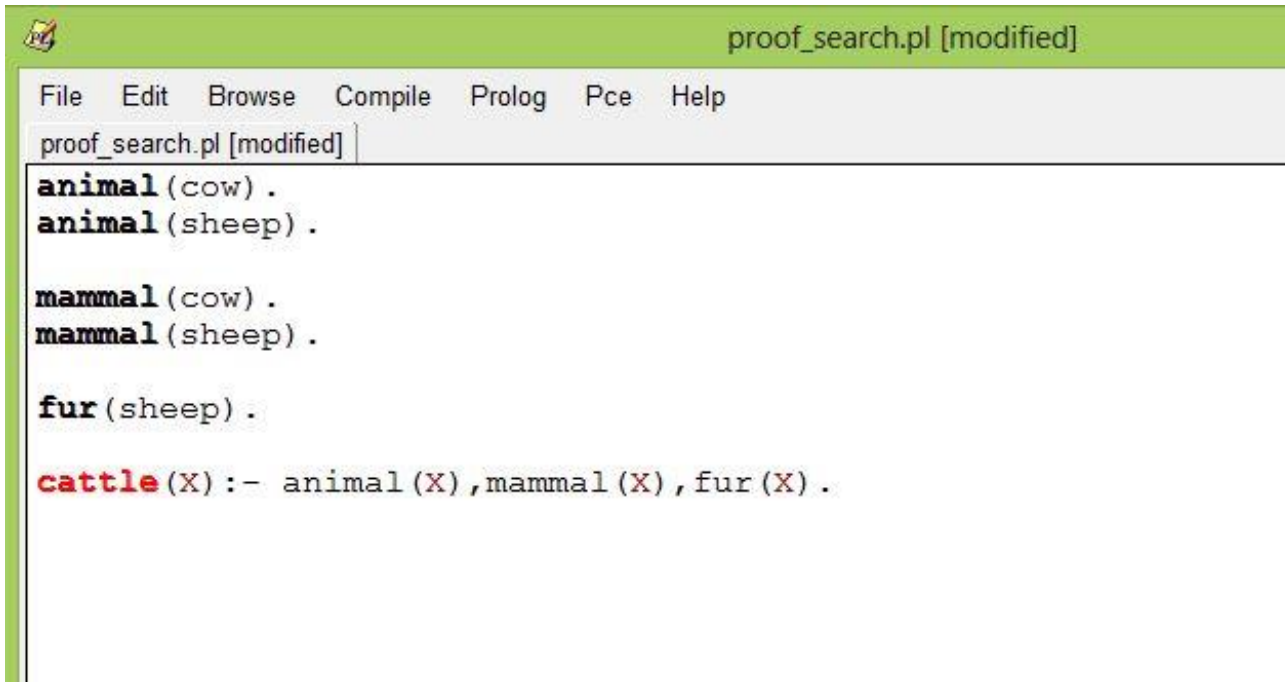
- Unification is a pattern matching mechanism in prolog.

term $X = \text{term } Y$ (= Unification)

- 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



Proof Search and Backtracking



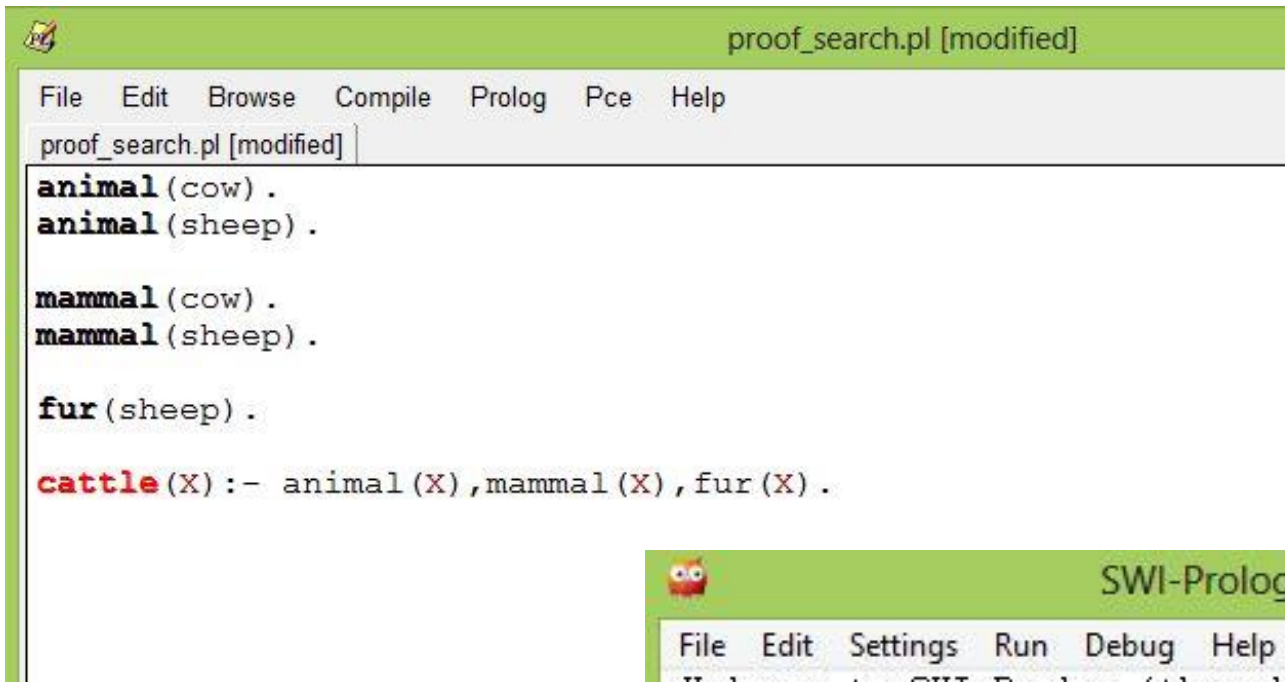
```
proof_search.pl [modified]
File Edit Browse Compile Prolog Pce Help
proof_search.pl [modified]
animal(cow) .
animal(sheep) .

mammal(cow) .
mammal(sheep) .

fur(sheep) .

cattle(X) :- animal(X), mammal(X), fur(X).
```

Proof Search and Backtracking

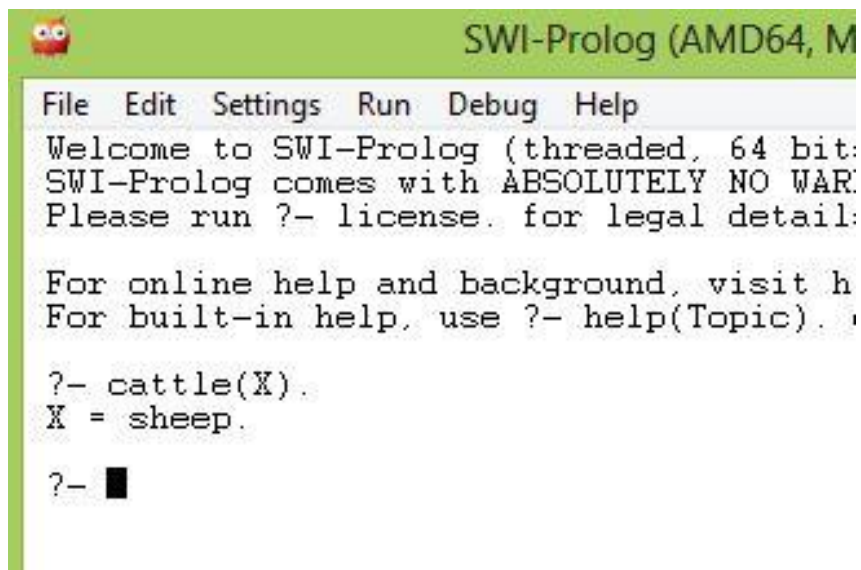


```
proof_search.pl [modified]
File Edit Browse Compile Prolog Pce Help
proof_search.pl [modified]
animal(cow) .
animal(sheep) .

mammal(cow) .
mammal(sheep) .

fur(sheep) .

cattle(X) :- animal(X), mammal(X), fur(X) .
```



```
SWI-Prolog (AMD64, M...
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bit:
SWI-Prolog comes with ABSOLUTELY NO WAR
Please run ?- license. for legal detail:

For online help and background, visit h
For built-in help, use ?- help(Topic).

?- cattle(X).
X = sheep.

?-
```

Proof Search and Backtracking

- How does prolog search in the knowledge base and check if the query is satisfied?

Proof Search and Backtracking

- How does prolog search in the knowledge base and check if the query is satisfied?
 - By a process called 'Proof Search'

Proof Search and Backtracking

```
animal(cow).  
animal(sheep).
```

```
mammal(cow).  
mammal(sheep).
```

```
fur(sheep).
```

```
cattle(X):- animal(X),mammal(X),fur(X).
```

```
?- cattle(Y).  
Y = sheep.
```

```
?- ■
```

Proof Search and Backtracking

cattle(Y)

```
animal(cow).  
animal(sheep).
```

```
mammal(cow).  
mammal(sheep).
```

```
fur(sheep).
```

```
cattle(X):- animal(X),mammal(X),fur(X).
```

```
?- cattle(Y).  
Y = sheep.
```

```
?- ■
```

Proof Search and Backtracking

```
animal(cow).  
animal(sheep).  
  
mammal(cow).  
mammal(sheep).  
  
fur(sheep).  
  
cattle(X) :- animal(X), mammal(X), fur(X).
```

```
?- cattle(Y).  
Y = sheep.
```

```
?- ■
```

cattle(Y)

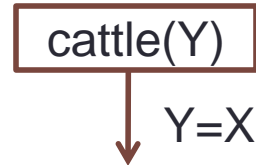


Proof Search and Backtracking

```
animal(cow).  
animal(sheep).  
  
mammal(cow).  
mammal(sheep).  
  
fur(sheep).  
  
cattle(X) :- animal(X), mammal(X), fur(X).
```

```
?- cattle(Y).  
Y = sheep.
```

```
?- ■
```

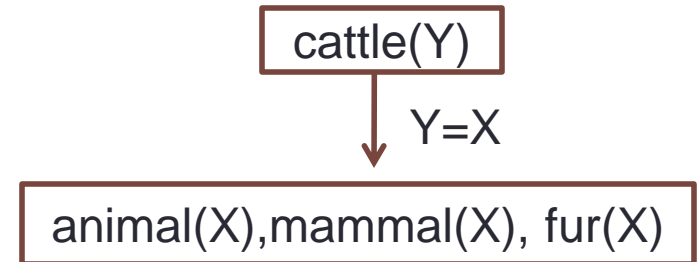


Proof Search and Backtracking

```
animal(cow).  
animal(sheep).  
  
mammal(cow).  
mammal(sheep).  
  
fur(sheep).  
  
cattle(X) :- animal(X), mammal(X), fur(X).
```

```
?- cattle(Y).  
Y = sheep.
```

```
?- ■
```

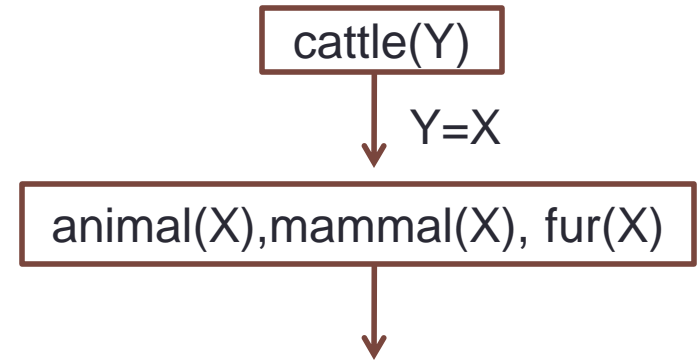


Proof Search and Backtracking

```
animal(cow).  
animal(sheep).  
  
mammal(cow).  
mammal(sheep).  
  
fur(sheep).  
  
cattle(X) :- animal(X), mammal(X), fur(X).
```

```
?- cattle(Y).  
Y = sheep.
```

```
?- ■
```

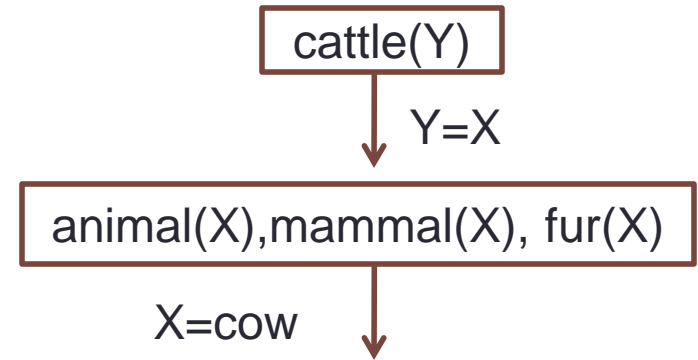


Proof Search and Backtracking

```
animal(cow).  
animal(sheep).  
  
mammal(cow).  
mammal(sheep).  
  
fur(sheep).  
  
cattle(X) :- animal(X), mammal(X), fur(X).
```

```
?- cattle(Y).  
Y = sheep.
```

```
?- ■
```

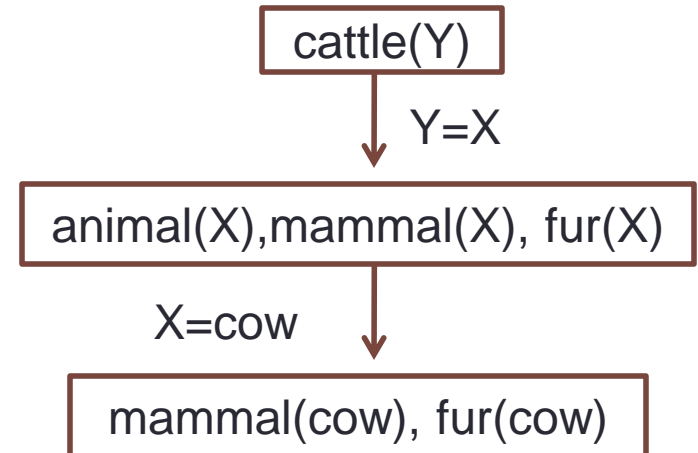


Proof Search and Backtracking

```
animal(cow).  
animal(sheep).  
  
mammal(cow).  
mammal(sheep).  
  
fur(sheep).  
  
cattle(X) :- animal(X), mammal(X), fur(X).
```

```
?- cattle(Y).  
Y = sheep.
```

```
?- ■
```

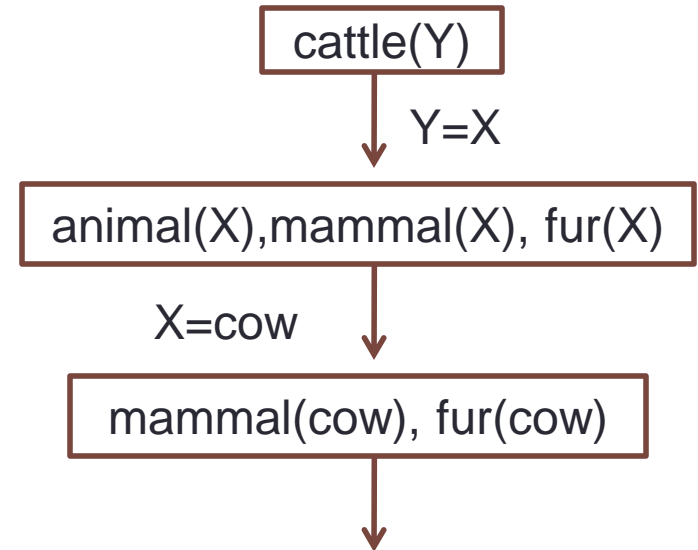


Proof Search and Backtracking

```
animal(cow).  
animal(sheep).  
  
mammal(cow).  
mammal(sheep).  
  
fur(sheep).  
  
cattle(X) :- animal(X), mammal(X), fur(X).
```

```
?- cattle(Y).  
Y = sheep.
```

```
?- ■
```

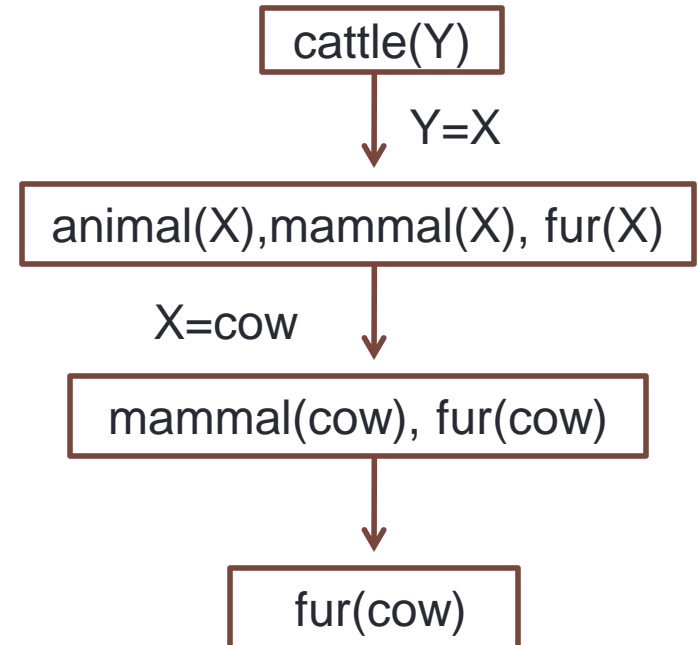


Proof Search and Backtracking

```
animal(cow).  
animal(sheep).  
  
mammal(cow).  
mammal(sheep).  
  
fur(sheep).  
  
cattle(X) :- animal(X), mammal(X), fur(X).
```

```
?- cattle(Y).  
Y = sheep.
```

```
?- ■
```

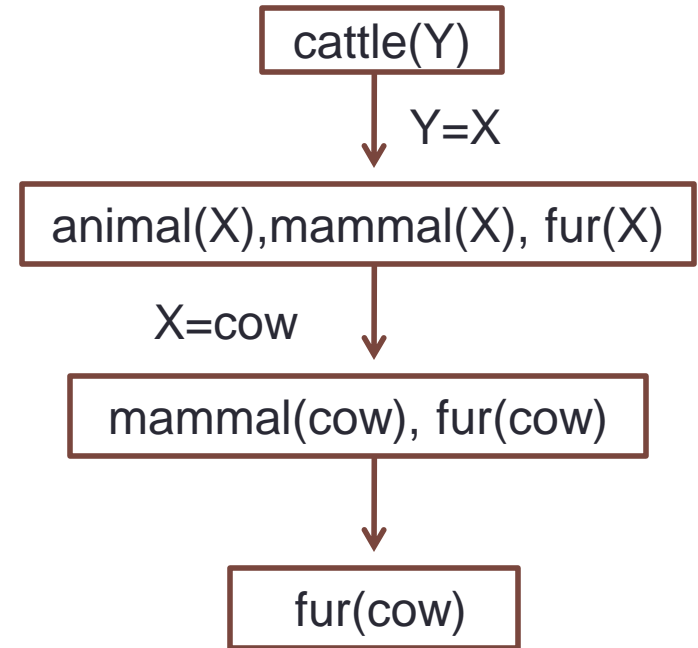


Proof Search and Backtracking

```
animal(cow).  
animal(sheep).  
  
mammal(cow).  
mammal(sheep).  
  
fur(sheep).  
  
cattle(X) :- animal(X), mammal(X), fur(X).
```

```
?- cattle(Y).  
Y = sheep.
```

```
?- ■
```



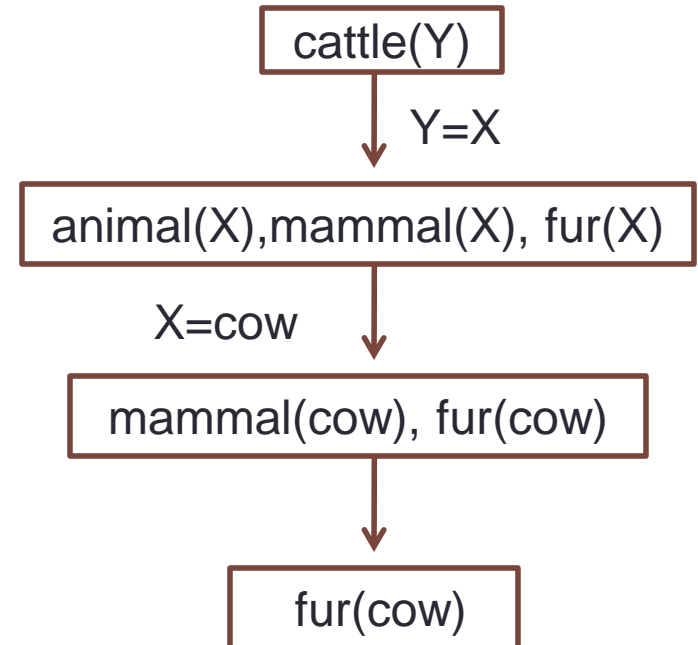
Can't be satisfied

Proof Search and Backtracking

```
animal(cow).  
animal(sheep).  
  
mammal(cow).  
mammal(sheep).  
  
fur(sheep).  
  
cattle(X) :- animal(X), mammal(X), fur(X).
```

```
?- cattle(Y).  
Y = sheep.
```

```
?- ■
```

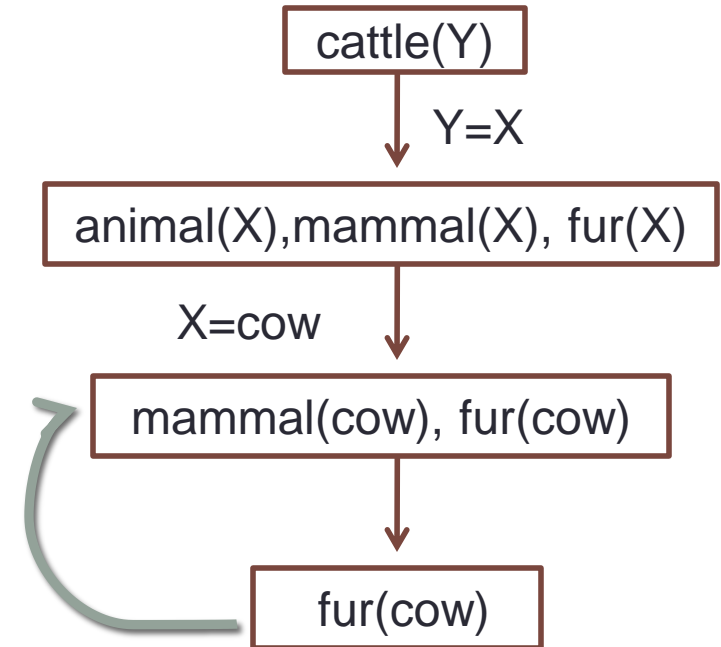


Proof Search and Backtracking

```
animal(cow).  
animal(sheep).  
  
mammal(cow).  
mammal(sheep).  
  
fur(sheep).  
  
cattle(X) :- animal(X), mammal(X), fur(X).
```

```
?- cattle(Y).  
Y = sheep.
```

```
?- ■
```

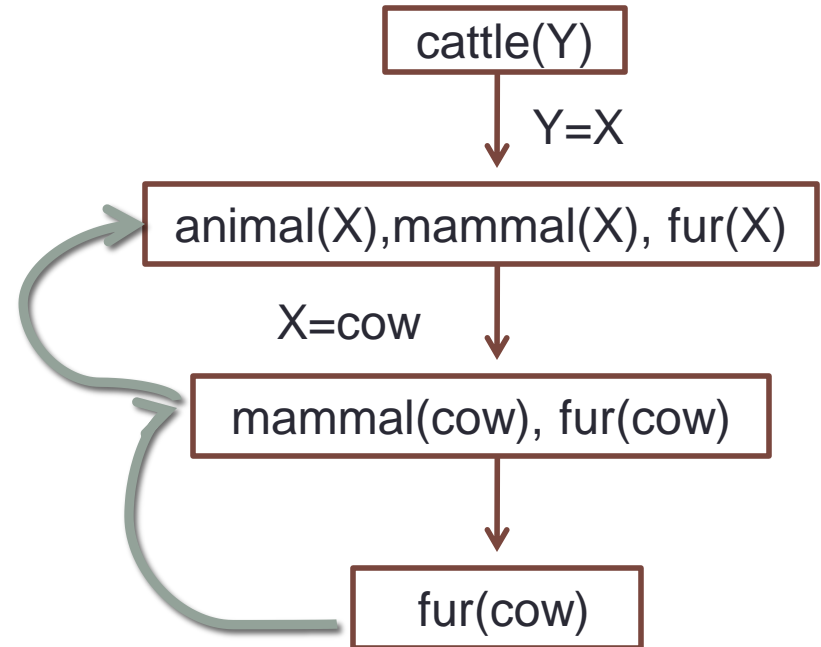


Proof Search and Backtracking

```
animal(cow).  
animal(sheep).  
  
mammal(cow).  
mammal(sheep).  
  
fur(sheep).  
  
cattle(X) :- animal(X), mammal(X), fur(X).
```

```
?- cattle(Y).  
Y = sheep.
```

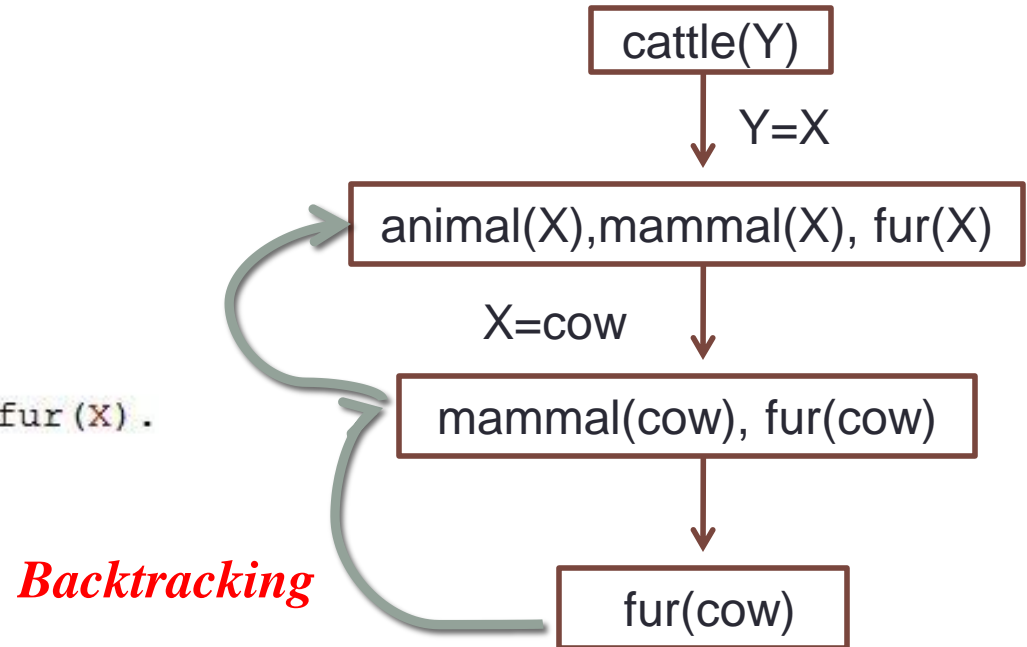
```
?- ■
```



Proof Search and Backtracking

```
animal(cow).  
animal(sheep).  
  
mammal(cow).  
mammal(sheep).  
  
fur(sheep).  
  
cattle(X) :- animal(X), mammal(X), fur(X).
```

```
?- cattle(Y).  
Y = sheep.  
  
?- ■
```



Proof Search and Backtracking

```
animal(cow).  
animal(sheep).
```

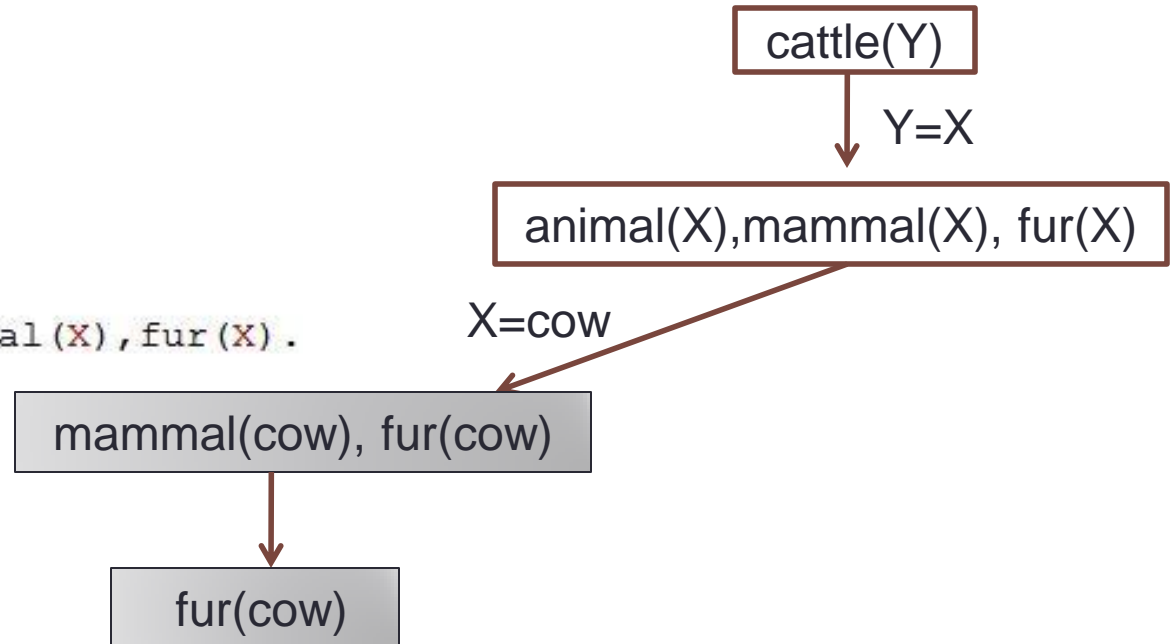
```
mammal(cow).  
mammal(sheep).
```

```
fur(sheep).
```

```
cattle(X) :- animal(X), mammal(X), fur(X).
```

```
?- cattle(Y).  
Y = sheep.
```

```
?- ■
```



Proof Search and Backtracking

```
animal(cow).  
animal(sheep).
```

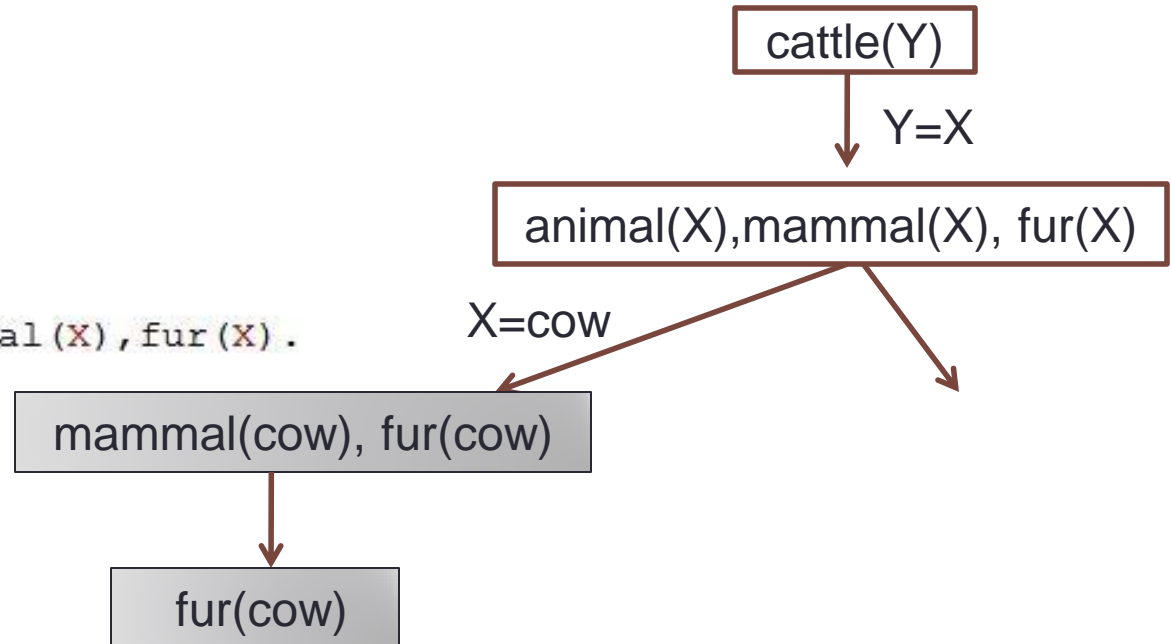
```
mammal(cow).  
mammal(sheep).
```

```
fur(sheep).
```

```
cattle(X) :- animal(X), mammal(X), fur(X).
```

```
?- cattle(Y).  
Y = sheep.
```

```
?- ■
```



Proof Search and Backtracking

```
animal(cow).  
animal(sheep).
```

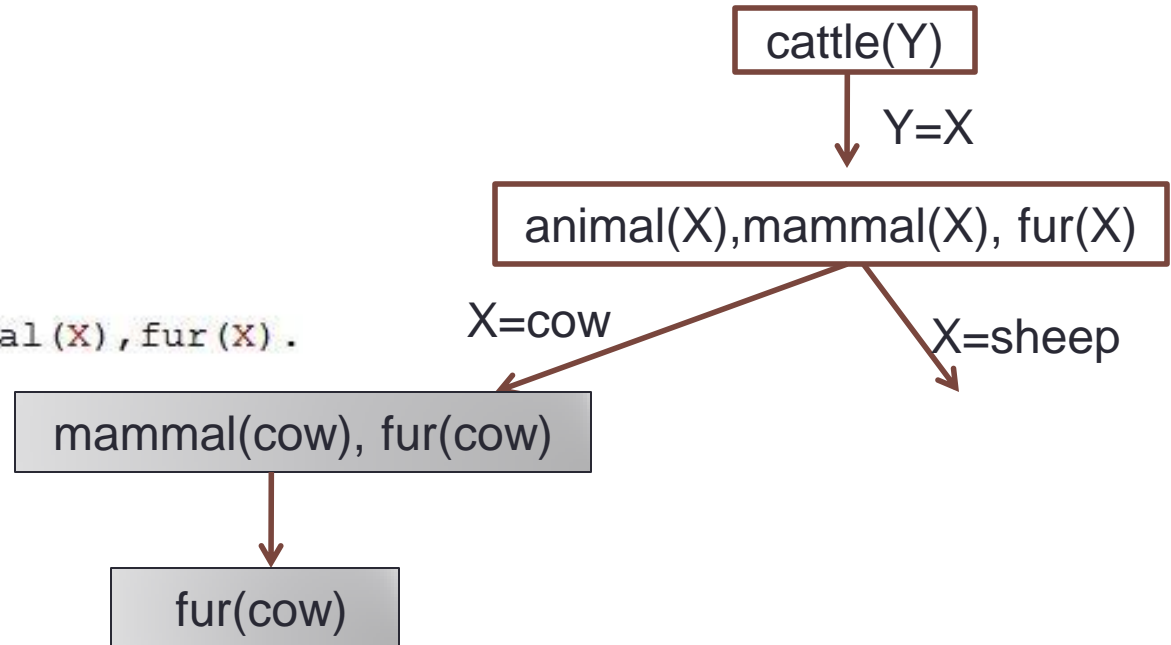
```
mammal(cow).  
mammal(sheep).
```

```
fur(sheep).
```

```
cattle(X) :- animal(X), mammal(X), fur(X).
```

```
?- cattle(Y).  
Y = sheep.
```

```
?- ■
```



Proof Search and Backtracking

```
animal(cow).  
animal(sheep).
```

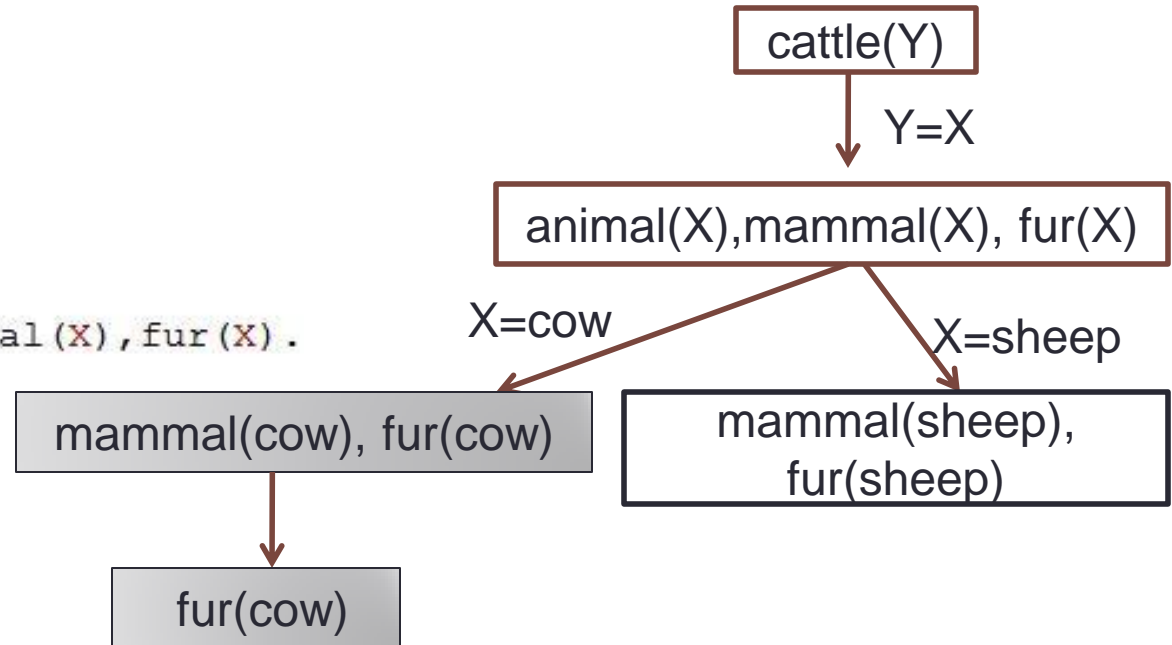
```
mammal(cow).  
mammal(sheep).
```

```
fur(sheep).
```

```
cattle(X) :- animal(X), mammal(X), fur(X).
```

```
?- cattle(Y).  
Y = sheep.
```

```
?- ■
```



Proof Search and Backtracking

```
animal(cow).  
animal(sheep).
```

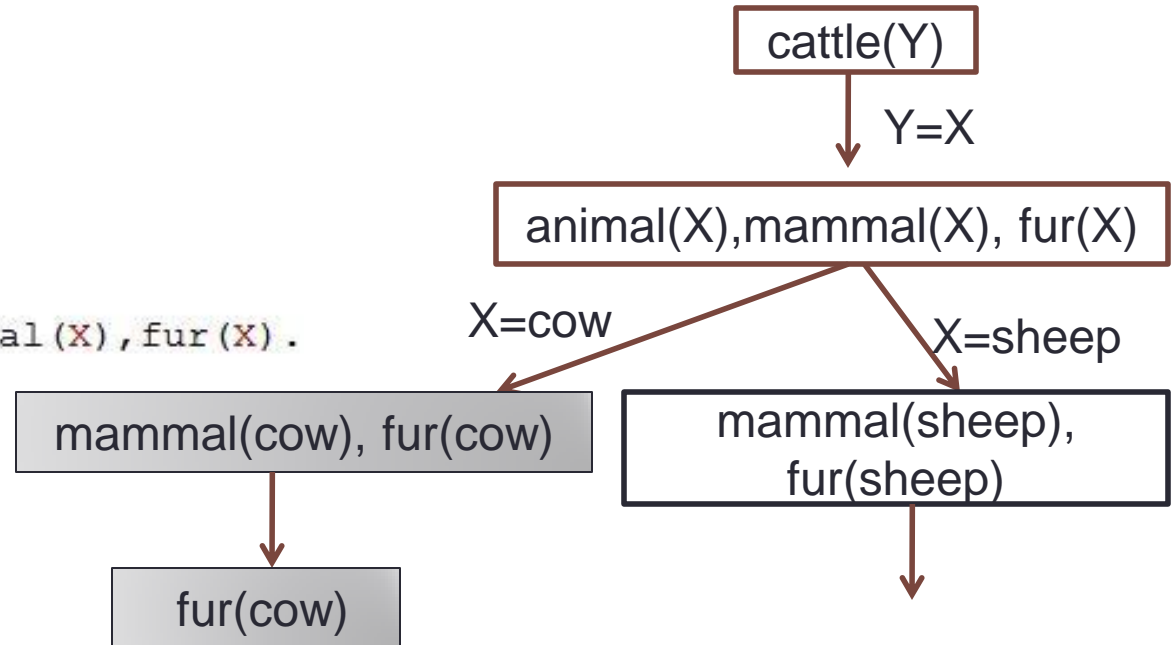
```
mammal(cow).  
mammal(sheep).
```

```
fur(sheep).
```

```
cattle(X) :- animal(X), mammal(X), fur(X).
```

```
?- cattle(Y).  
Y = sheep.
```

```
?- ■
```



Proof Search and Backtracking

```
animal(cow).  
animal(sheep).
```

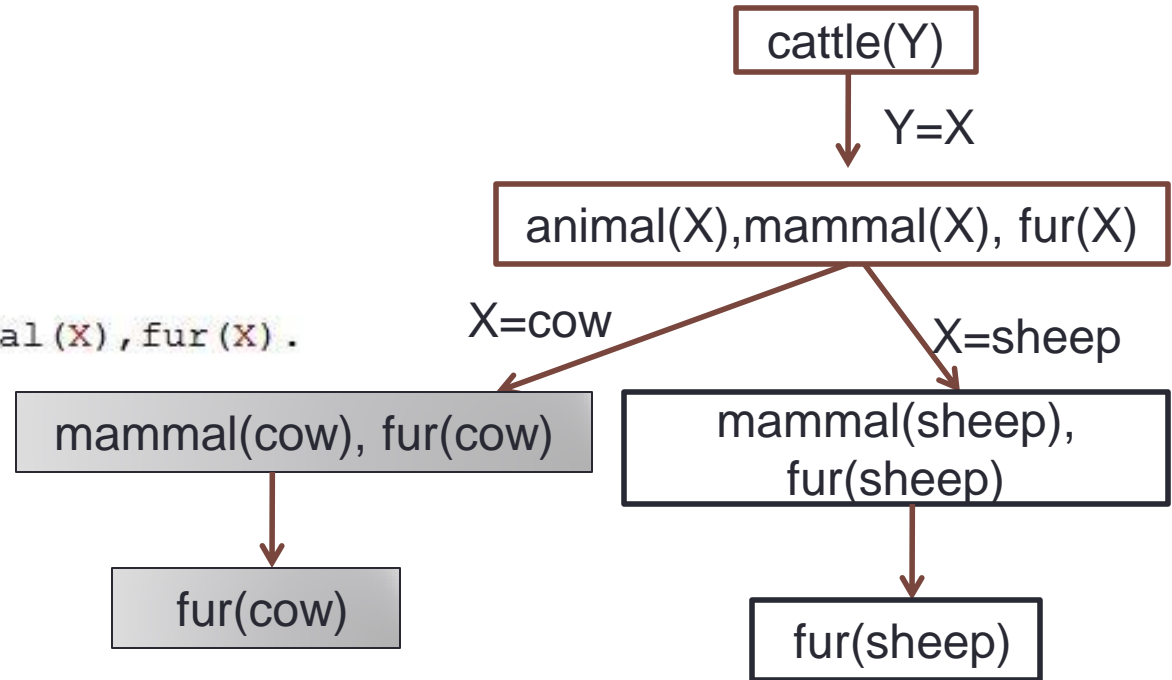
```
mammal(cow).  
mammal(sheep).
```

```
fur(sheep).
```

```
cattle(X) :- animal(X), mammal(X), fur(X).
```

```
?- cattle(Y).  
Y = sheep.
```

```
?- ■
```



Proof Search and Backtracking

```
animal(cow).  
animal(sheep).
```

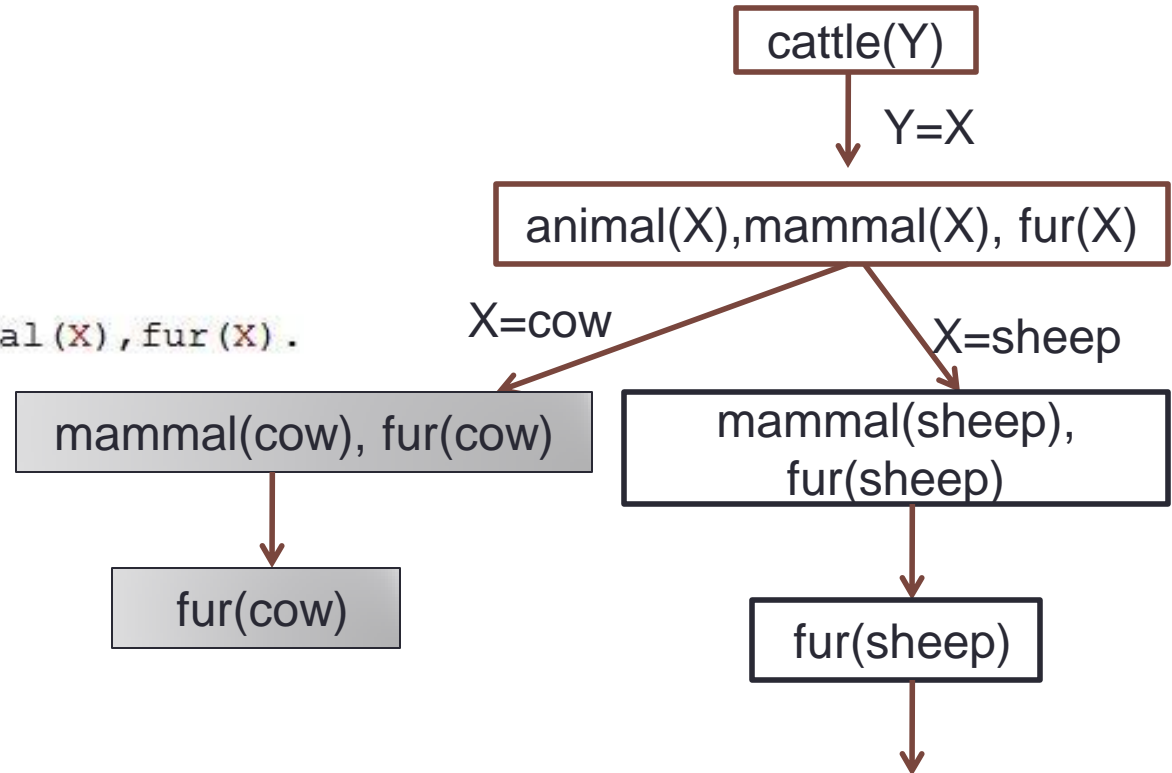
```
mammal(cow).  
mammal(sheep).
```

```
fur(sheep).
```

```
cattle(X) :- animal(X), mammal(X), fur(X).
```

```
?- cattle(Y).  
Y = sheep.
```

```
?- ■
```



Proof Search and Backtracking

```
animal(cow).  
animal(sheep).
```

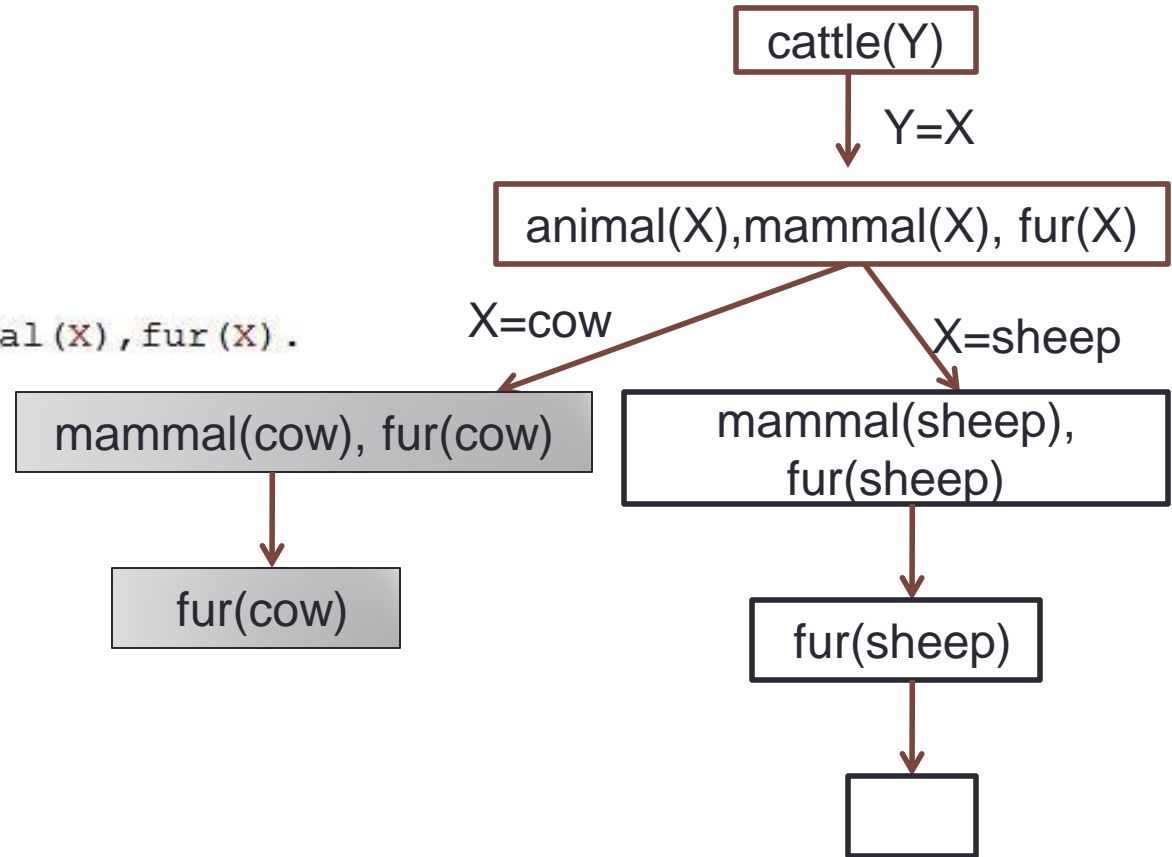
```
mammal(cow).  
mammal(sheep).
```

```
fur(sheep).
```

```
cattle(X) :- animal(X), mammal(X), fur(X).
```

```
?- cattle(Y).  
Y = sheep.
```

```
?- ■
```

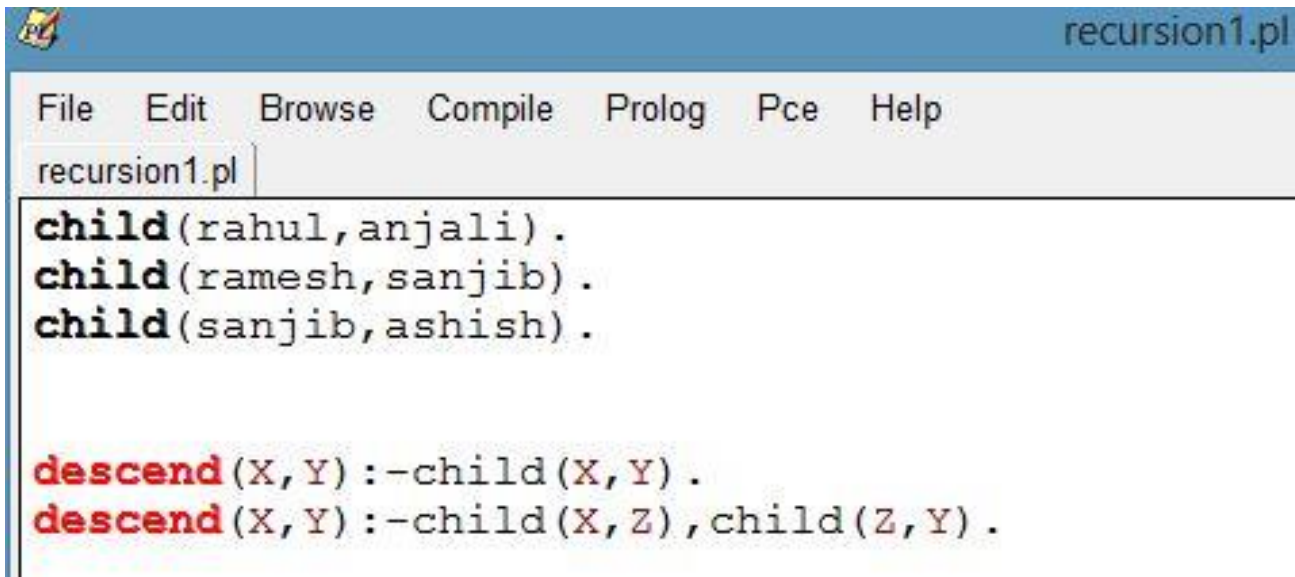


Recursion

- In prolog, predicates are allowed to be defined recursively
- A predicate is recursively defined if one or more rules in its definition refers to itself

Recursion

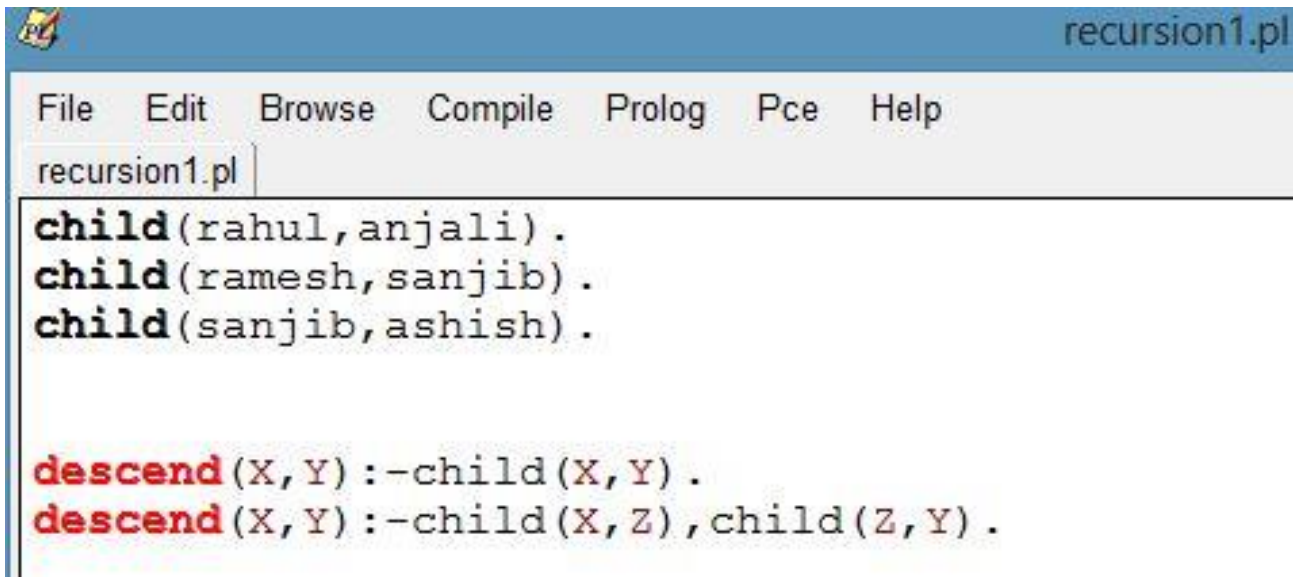
Recursion



```
recursion1.pl
File Edit Browse Compile Prolog Pce Help
recursion1.pl
child(rahul,anjali) .
child(ramesh,sanjib) .
child(sanjib,ashish) .

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

Recursion



```
recursion1.pl  
  
child(rahul,anjali).  
child(ramesh,sanjib).  
child(sanjib,ashish).  
  
descend(X,Y):-child(X,Y).  
descend(X,Y):-child(X,Z),child(Z,Y).
```


```
?- descend(ramesh,ashish).
```

```
true.
```

```
?- ■
```

Recursion

Recursion

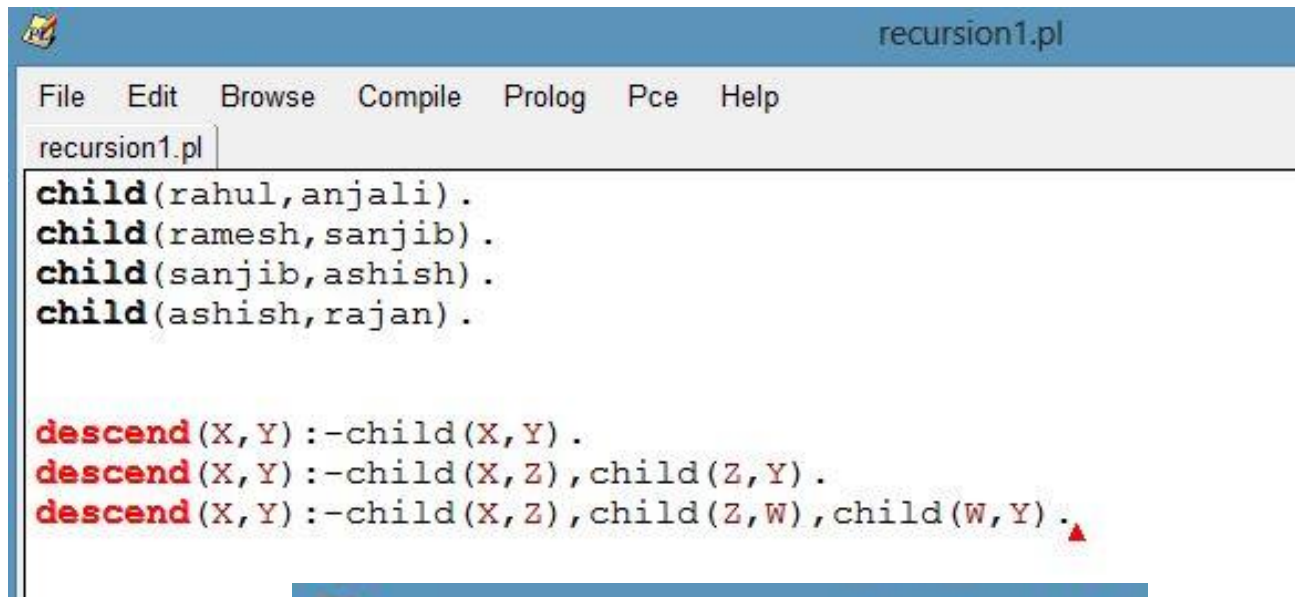


```
recursion1.pl

child(rahul,anjali).
child(ramesh,sanjib).
child(sanjib,ashish).
child(ashish,rajan).

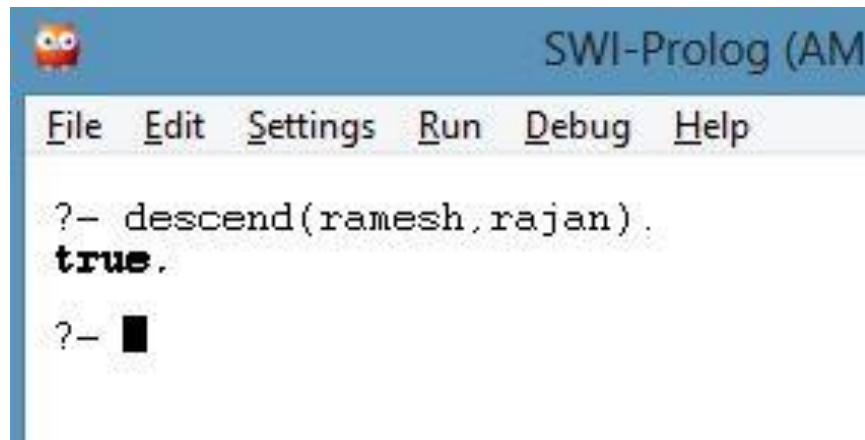
descend(X,Y):-child(X,Y).
descend(X,Y):-child(X,Z),child(Z,Y).
descend(X,Y):-child(X,Z),child(Z,W),child(W,Y).
```

Recursion



```
recursion1.pl
File Edit Browse Compile Prolog Pce Help
recursion1.pl
child(rahul,anjali).
child(ramesh,sanjib).
child(sanjib,ashish).
child(ashish,rajan).

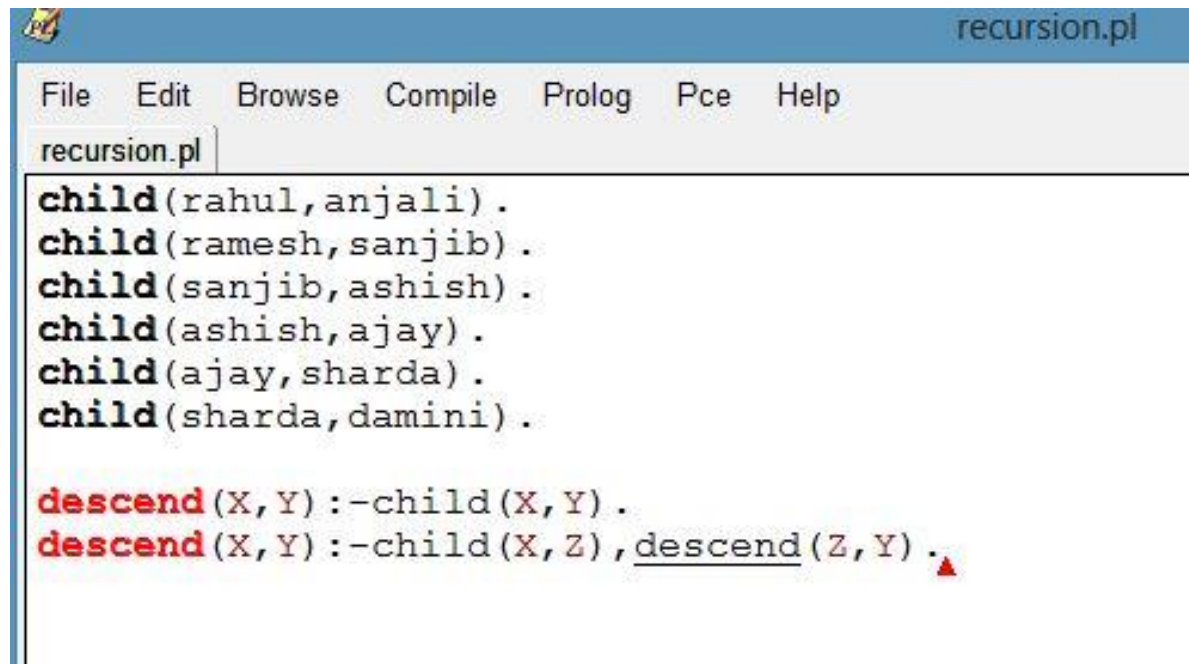
descend(X,Y):-child(X,Y).
descend(X,Y):-child(X,Z),child(Z,Y).
descend(X,Y):-child(X,Z),child(Z,W),child(W,Y).
```



```
SWI-Prolog (AM...
File Edit Settings Run Debug Help
?- descend(ramesh,rajan).
true.
?-
```

Recursion

Recursion



```
recursion.pl
File Edit Browse Compile Prolog Pce Help
recursion.pl
child(rahul,anjali).
child(ramesh,sanjib).
child(sanjib,ashish).
child(ashish,ajay).
child(ajay,sharda).
child(sharda,damini).

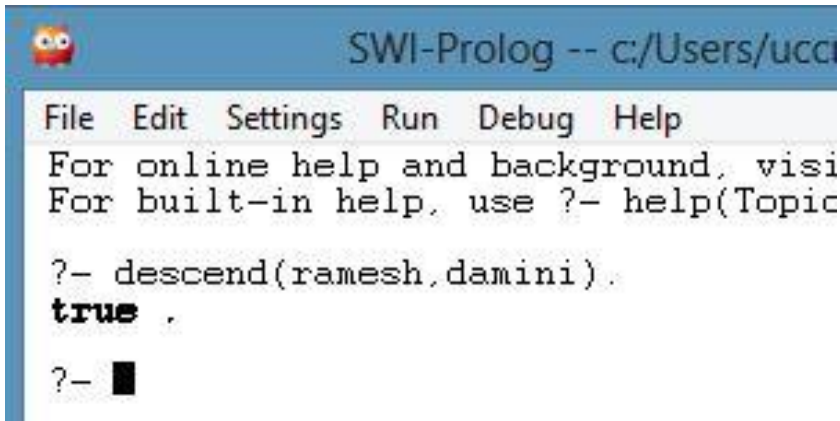
descend(X,Y):-child(X,Y).
descend(X,Y):-child(X,Z),descend(Z,Y).
```

Recursion



```
recursion.pl
File Edit Browse Compile Prolog Pce Help
recursion.pl
child(rahul,anjali).
child(ramesh,sanjib).
child(sanjib,ashish).
child(ashish,ajay).
child(ajay,sharda).
child(sharda,damini).

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



```
SWI-Prolog -- c:/Users/ucca
File Edit Settings Run Debug Help
For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic)

?- descend(ramesh,damini).
true.

?-
```

Lists

- List in prolog is a finite sequence of elements
- It can be homogeneous and heterogeneous
- A list is enclosed in a square bracket
- Examples:

[rajan,rahul,babita,shreya,ajay]

[babita, 3.4,X,9,nilufer]

[X, man(sam), animal(tiger)]

[[1,2,3],[a,b,c]]

[]

Lists

- Length of a list = number of elements
- Special list with no elements → empty list []
- A non empty list has two parts:
 - Head
 - Tail
- Head: First element of the list
- Tail: Remaining elements of the list
- Tail of a list is also a list

Lists: Examples

- [arif, binay, sonal, mitali]
- [[], 2.5, fruit(X),[a,[b,c]]]
- [ramesh]

Lists: Examples

- [arif, binay, sonal, mitali]
 - Head: arif
 - Tail: [binay, sonal, mitali]
- [[], 2.5, fruit(X),[a,[b,c]]]
- [ramesh]

Lists: Examples

- [arif, binay, sonal, mitali]
 - Head: arif
 - Tail: [binay, sonal, mitali]
- [[], 2.5, fruit(X),[a,[b,c]]]
 - Head:[]
 - Tail: [2.5, fruit(X),[a,[b,c]]]
- [ramesh]

Lists: Examples


- [arif, binay, sonal, mitali]
 - Head: arif
 - Tail: [binay, sonal, mitali]
- [[], 2.5, fruit(X),[a,[b,c]]]
 - Head: []
 - Tail: [2.5, fruit(X),[a,[b,c]]]
- [ramesh]
 - Head: ramesh
 - Tail: []

Lists: Operator ‘|’

- The operator ‘|’ is used to differentiate head and tail of a list

Lists: Operator ‘|’

- The operator ‘|’ is used to differentiate head and tail of a list

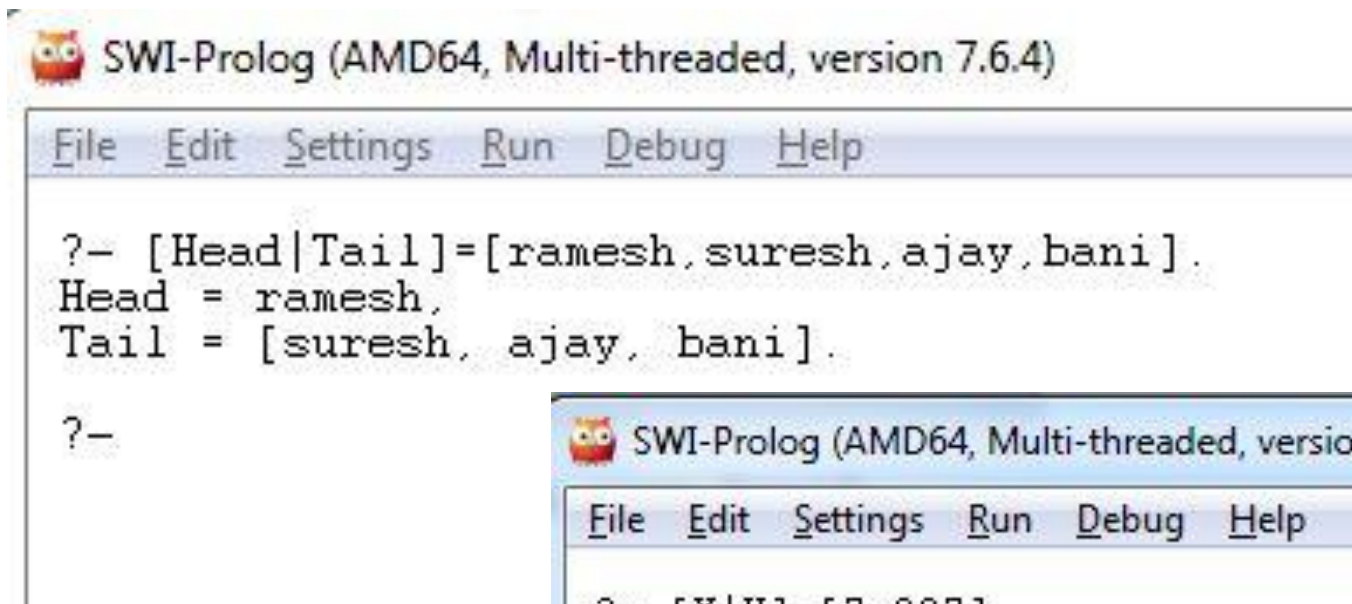
 SWI-Prolog (AMD64, Multi-threaded, version 7.6.4)

File Edit Settings Run Debug Help


```
?- [Head|Tail]=[ramesh,suresh,ajay,bani].  
Head = ramesh,  
Tail = [suresh, ajay, bani].  
  
?-
```

Lists: Operator '|'

- The operator '|' is used to differentiate head and tail of a list




```
?- [Head|Tail]=[ramesh,suresh,ajay,bani].  
Head = ramesh,  
Tail = [suresh, ajay, bani].  
  
?-
```




```
?- [X|Y]=[7.987].  
X = 7.987,  
Y = [].  
  
?- ■
```

Lists: Operator '|'

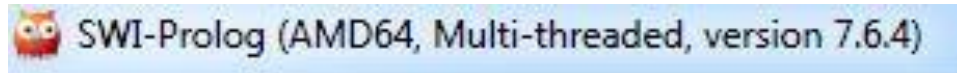
Lists: Operator ‘|’

 SWI-Prolog (AMD64, Multi-threaded, version 7.6.4)
File Edit Settings Run Debug Help


?- [X,Y|Z]=[123,anisha,87.5,good].
X = 123,
Y = anisha,
Z = [87.5, good].

?- 

Lists: Operator ‘|’

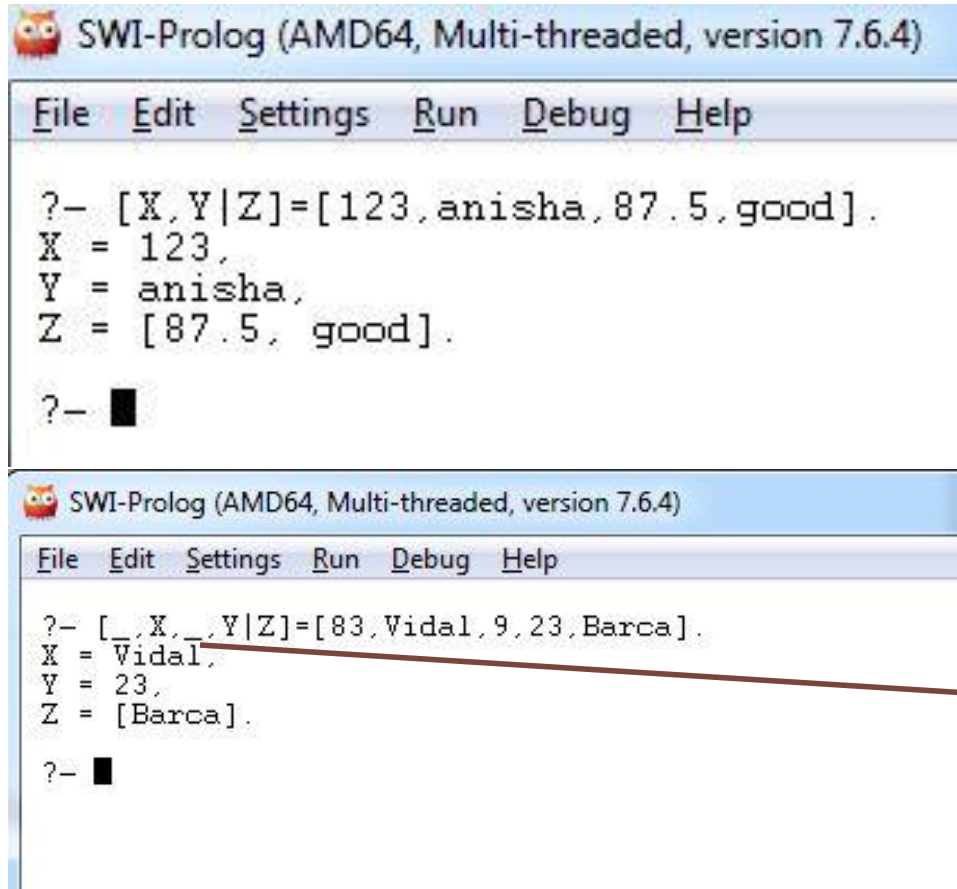


```
?- [X,Y|Z]=[123,anisha,87.5,good].  
X = 123,  
Y = anisha,  
Z = [87.5, good].  
  
?-
```



```
?- [_,X_,Y|Z]=[83,Vidal,9,23,Barca].  
X = Vidal,  
Y = 23,  
Z = [Barca].  
  
?-
```

Lists: Operator '|'



The image shows two screenshots of the SWI-Prolog (AMD64, Multi-threaded, version 7.6.4) IDE. The top screenshot shows a query `?- [X,Y|Z]=[123,anisha,87.5,good].` with bindings `X = 123,`, `Y = anisha,`, and `Z = [87.5, good].`. The bottom screenshot shows a query `?- [_,X_,Y|Z]=[83,Vidal,9,23,Barca].` with bindings `X = Vidal,`, `Y = 23,`, and `Z = [Barca].`. A red arrow points from the underscore in the second query to the text 'Anonymous variable'.

```
SWI-Prolog (AMD64, Multi-threaded, version 7.6.4)
File Edit Settings Run Debug Help

?- [X,Y|Z]=[123,anisha,87.5,good].
X = 123,
Y = anisha,
Z = [87.5, good].

?- █

SWI-Prolog (AMD64, Multi-threaded, version 7.6.4)
File Edit Settings Run Debug Help

?- [_,X_,Y|Z]=[83,Vidal,9,23,Barca].
X = Vidal,
Y = 23,
Z = [Barca].

?- █
```

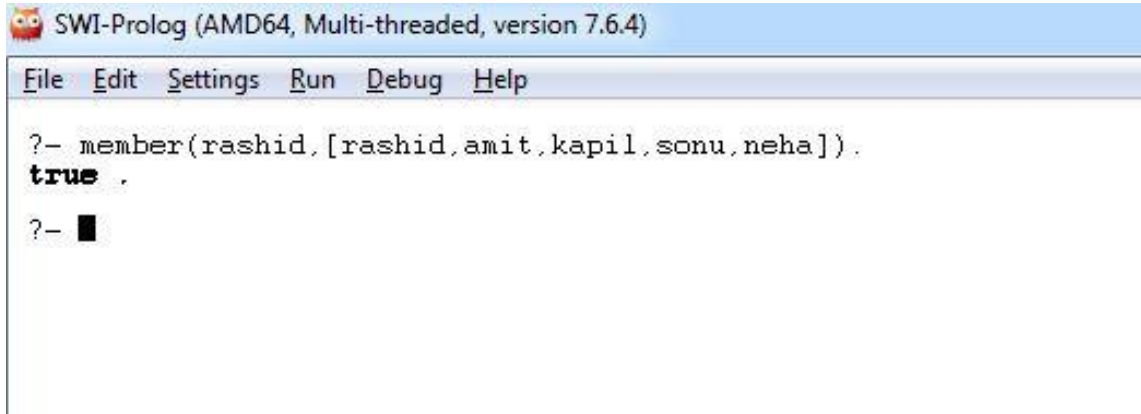
→ *Anonymous variable*

Lists: 'member' predicate

- The member/2 predicate is used to check if an element is present in a list

Lists: 'member' predicate

- The member/2 predicate is used to check if an element is present in a list

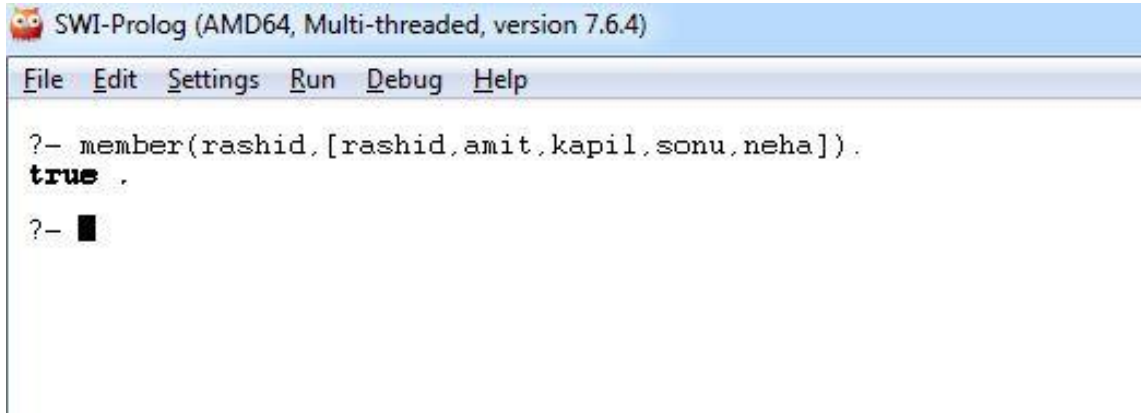


A screenshot of the SWI-Prolog (AMD64, Multi-threaded, version 7.6.4) window. The window has a title bar with the text "SWI-Prolog (AMD64, Multi-threaded, version 7.6.4)". Below the title bar is a menu bar with the following items: File, Edit, Settings, Run, Debug, Help. The main text area contains the following text:

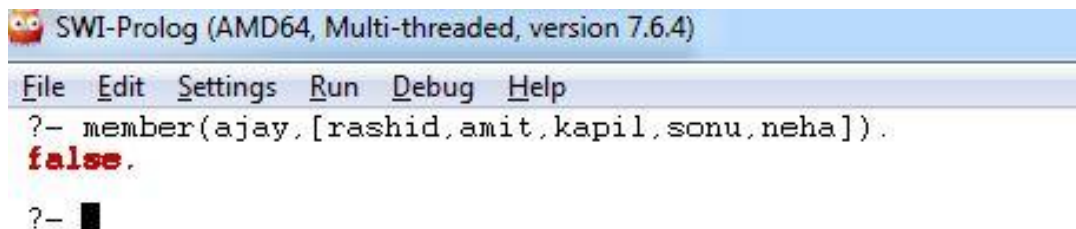
```
?- member(rashid,[rashid,amit,kapil,sonu,neha]).  
true .  
?- ■
```

Lists: 'member' predicate

- The member/2 predicate is used to check if an element is present in a list



```
SWI-Prolog (AMD64, Multi-threaded, version 7.6.4)
File Edit Settings Run Debug Help
?- member(rashid,[rashid,amit,kapil,sonu,neha]).
true.
?- █
```



```
SWI-Prolog (AMD64, Multi-threaded, version 7.6.4)
File Edit Settings Run Debug Help
?- member(ajay,[rashid,amit,kapil,sonu,neha]).
false.
?- █
```

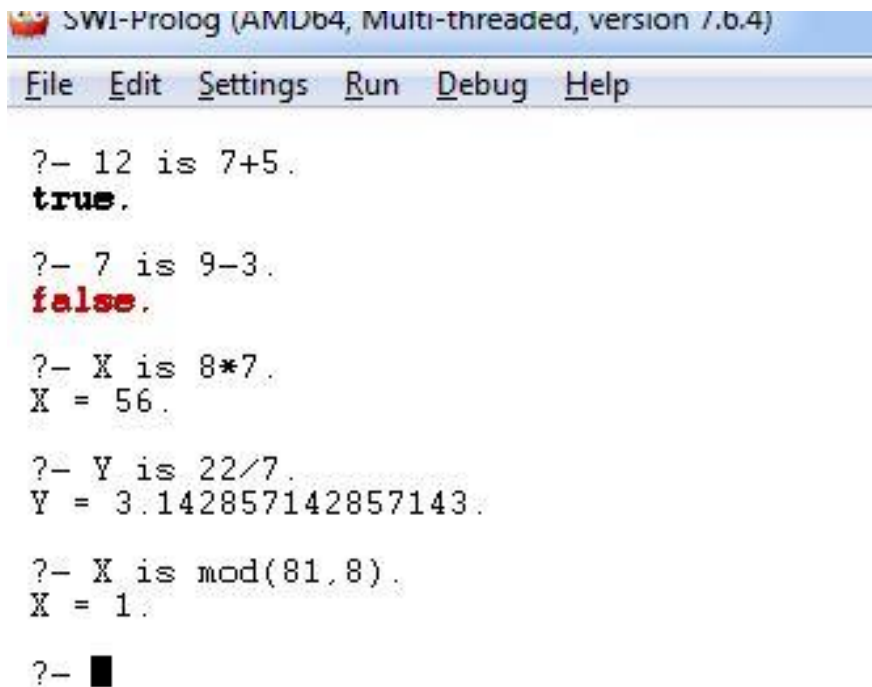
Arithmetic

- Prolog allows basic arithmetic operations
 $+$, $-$, $*$, $/$, **mod**

Arithmetic

- Prolog allows basic arithmetic operations

+, -, *, / , mod



```
SWI-Prolog (AMD64, Multi-threaded, version 7.6.4)
File Edit Settings Run Debug Help

?- 12 is 7+5.
true.

?- 7 is 9-3.
false.

?- X is 8*7.
X = 56.

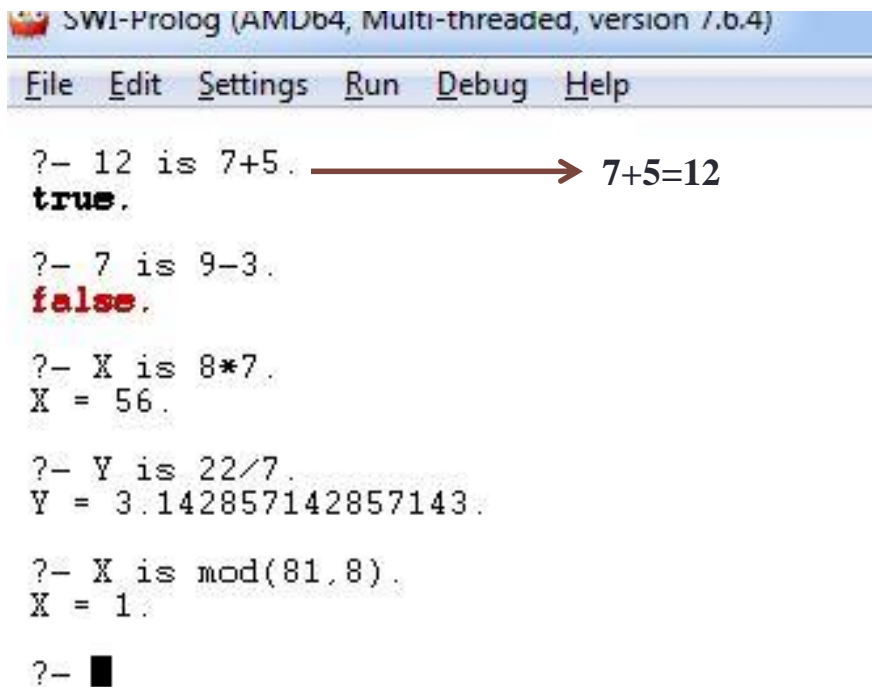
?- Y is 22/7.
Y = 3.142857142857143.

?- X is mod(81,8).
X = 1.

?- ■
```

Arithmetic

- Prolog allows basic arithmetic operations
 $+$, $-$, $*$, $/$, mod



The screenshot shows the SWI-Prolog (AMD64, Multi-threaded, version 7.6.4) interface. The menu bar includes File, Edit, Settings, Run, Debug, and Help. The main window displays several Prolog queries and their results:

```
?- 12 is 7+5.  $\longrightarrow$  7+5=12
true.

?- 7 is 9-3.
false.

?- X is 8*7.
X = 56.

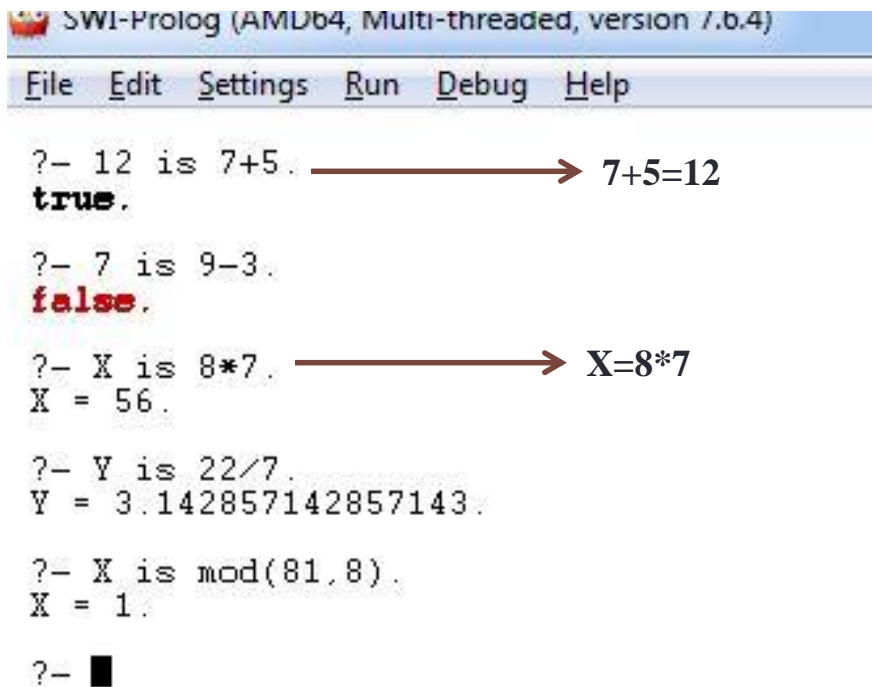
?- Y is 22/7.
Y = 3.142857142857143.

?- X is mod(81,8).
X = 1.

?-  $\blacksquare$ 
```


Arithmetic

- Prolog allows basic arithmetic operations
 $+$, $-$, $*$, $/$, **mod**



```
SWI-Prolog (AMD64, Multi-threaded, version 7.6.4)
File Edit Settings Run Debug Help

?- 12 is 7+5. -----> 7+5=12
true.

?- 7 is 9-3.
false.

?- X is 8*7. -----> X=8*7
X = 56.

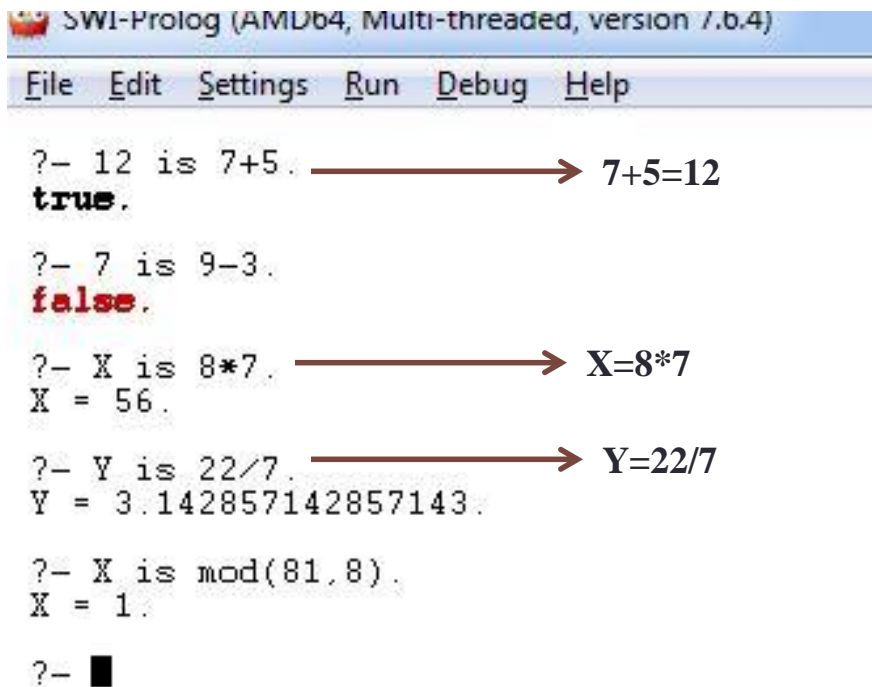
?- Y is 22/7.
Y = 3.142857142857143.

?- X is mod(81,8).
X = 1.

?- ■
```

Arithmetic

- Prolog allows basic arithmetic operations
 $+$, $-$, $*$, $/$, **mod**



```
SWI-Prolog (AMD64, Multi-threaded, version 7.6.4)
File Edit Settings Run Debug Help

?- 12 is 7+5. -----> 7+5=12
true.

?- 7 is 9-3.
false.

?- X is 8*7. -----> X=8*7
X = 56.

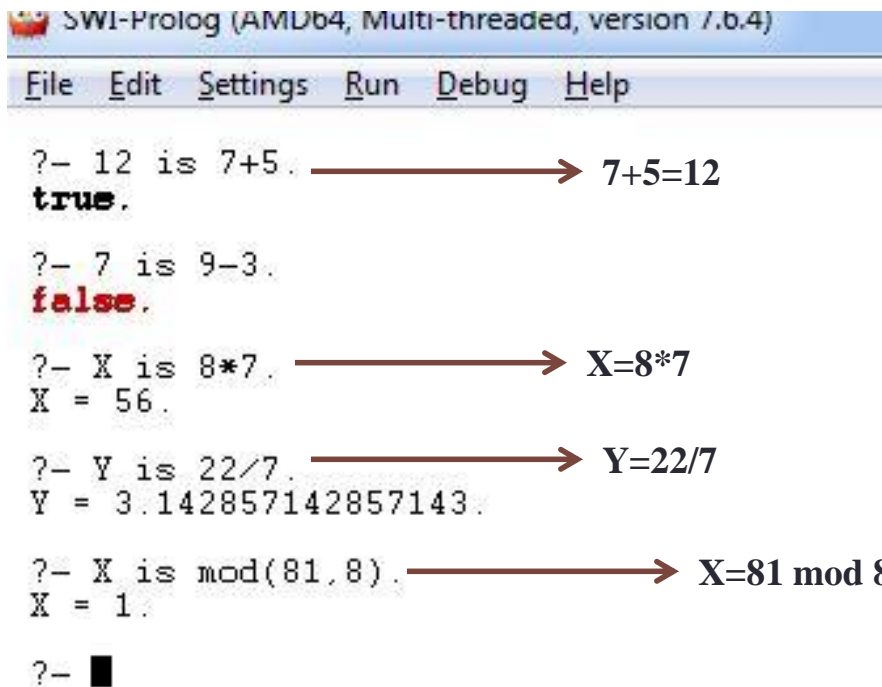
?- Y is 22/7. -----> Y=22/7
Y = 3.142857142857143.

?- X is mod(81,8).
X = 1.

?- ■
```

Arithmetic

- Prolog allows basic arithmetic operations
 $+$, $-$, $*$, $/$, mod



```
SWI-Prolog (AMD64, Multi-threaded, version 7.6.4)
File Edit Settings Run Debug Help

?- 12 is 7+5.  $\longrightarrow$  7+5=12
true.

?- 7 is 9-3.
false.

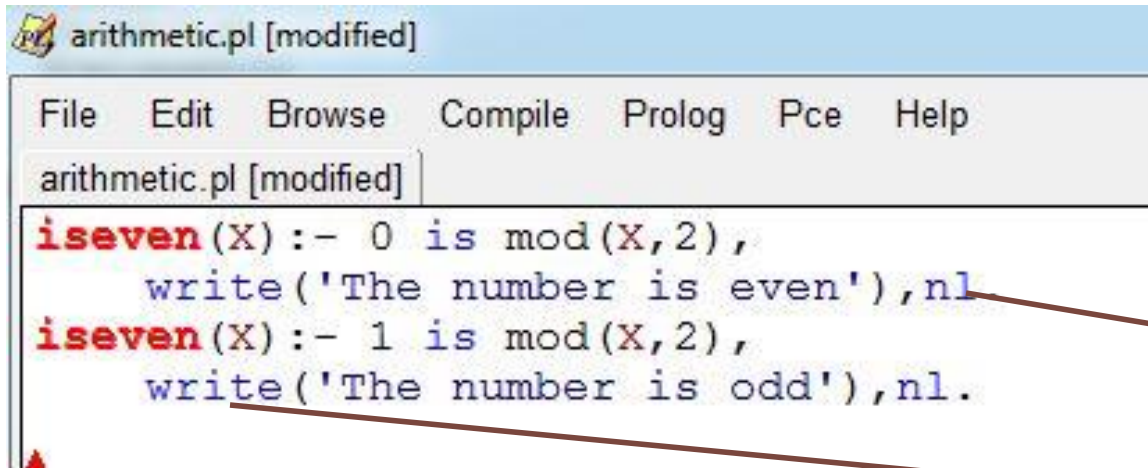
?- X is 8*7.  $\longrightarrow$  X=8*7
X = 56.

?- Y is 22/7.  $\longrightarrow$  Y=22/7
Y = 3.142857142857143.

?- X is mod(81,8).  $\longrightarrow$  X=81 mod 8
X = 1.

?- ■
```

Predicates with arithmetic

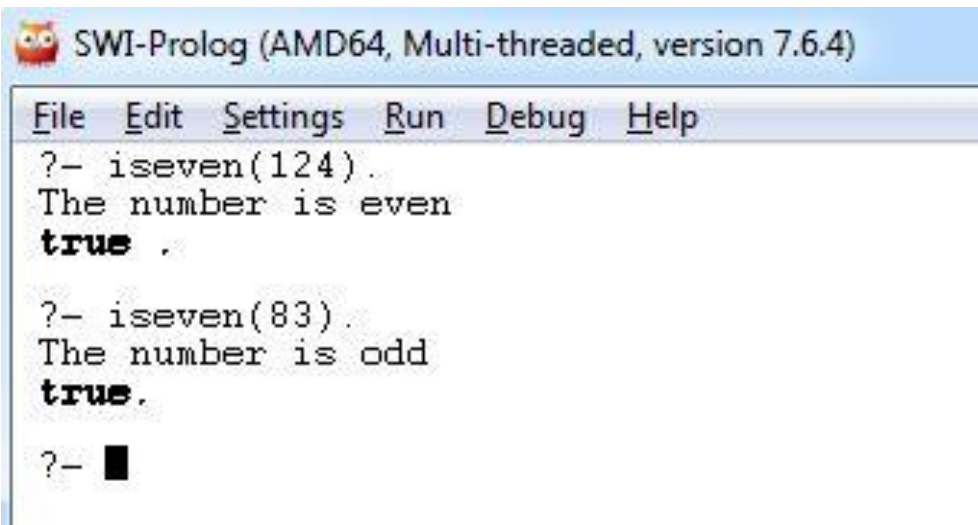


The screenshot shows a Prolog editor window with a menu bar (File, Edit, Browse, Compile, Prolog, Pce, Help) and a text area containing two Prolog predicates. The first predicate, `iseven(X) :- 0 is mod(X,2), write('The number is even'),nl`, is on one line. The second predicate, `iseven(X) :- 1 is mod(X,2), write('The number is odd'),nl.`, is on the next line. A red arrow points from the end of the first line to the text "New Line". Another red arrow points from the `nl` in the second line to the text "Predicate to print".

```
arithmetic.pl [modified]
File Edit Browse Compile Prolog Pce Help
arithmetic.pl [modified]
iseven(X) :- 0 is mod(X,2),
    write('The number is even'),nl
iseven(X) :- 1 is mod(X,2),
    write('The number is odd'),nl.
```

New Line

Predicate to print



The screenshot shows the SWI-Prolog console window with the title "SWI-Prolog (AMD64, Multi-threaded, version 7.6.4)". The console displays the results of two queries: `?- iseven(124).` which outputs "The number is even" and `true .`, and `?- iseven(83).` which outputs "The number is odd" and `true.`. The prompt `?-` is followed by a black square cursor.

```
SWI-Prolog (AMD64, Multi-threaded, version 7.6.4)
File Edit Settings Run Debug Help
?- iseven(124).
The number is even
true .
?- iseven(83).
The number is odd
true.
?- 
```

The is/2 predicate

- In prolog the $+$, $-$, $*$ and $/$ does not evaluate anything on their own
- They are predicates with arity 2.
eg. $9-2$ is actually $-(9,2)$
- To carry out the evaluation of the expressions we need to use built-in is/2 predicate

The is/2 predicate

- In prolog the $+$, $-$, $*$ and $/$ does not evaluate anything on their own
- They are predicates with arity 2.
eg. $9-2$ is actually $-(9,2)$
- To carry out the evaluation of the expressions we need to use built-in `is/2` predicate



```
SWI-Prolog (AMD64, Multi-threaded, version 7.6.4)
File Edit Settings Run Debug Help

?- X=9-2.
X = 9-2.

?- X is 9-2.
X = 7.

?- 
```

The is/2 predicate

- In prolog the +,-,* and / does not evaluate anything on their own
- They are predicates with arity 2.
eg. 9-2 is actually $-(9,2)$
- To carry out the evaluation of the expressions we need to use built-in is/2 predicate

```
SWI-Prolog (AMD64, Multi-threaded, version 7.6.4)
File Edit Settings Run Debug Help
?- X=9-2.
X = 9-2.

?- X is 9-2.
X = 7.

?-
```

```
SWI-Prolog (AMD64, Multi-threaded, version 7.6.4)
File Edit Settings Run Debug Help
?- is(X,-(9,2)).
X = 7.

?- 9-2 is X.
ERROR: Arguments are not sufficiently instantiated
ERROR: In:
ERROR:      [8] 9-2 is _6278
ERROR:      [7] <user>
?-
```

Comparison operators

- Operators to compare numbers:

 SWI-Prolog (AMD64, Multi-threaded, version 7.6.4)

File Edit Settings Run Debug Help

?- 4<7.

true.

?- 9=<9.

true.

?- 9=<10.

true.

?- 8=:3+5.

true.

?- 7=\19.

true.

?- 21>=20.

true.

?- 21>=21.

true.

?- 33>21.

true.

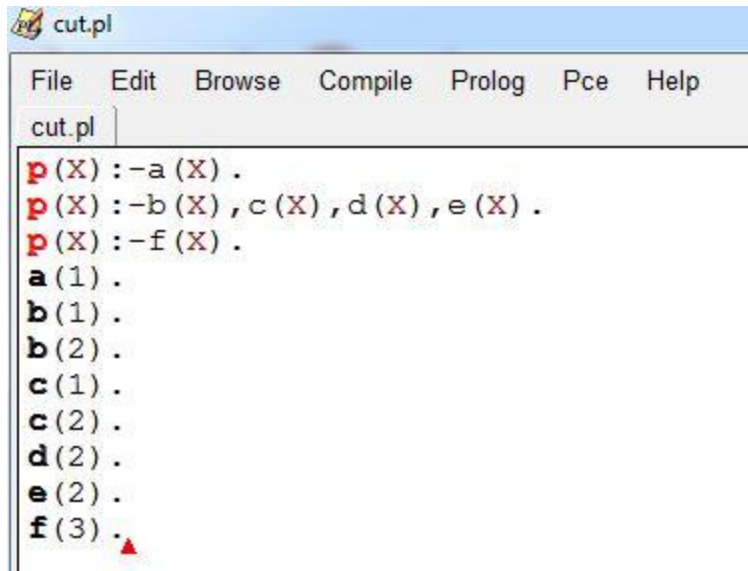
?-

Cut

- Backtracking is a characteristic feature of Prolog
- However it can lead to inefficiency:
 - Prolog may waste time in exploring all possibilities unnecessarily
 - It should have some control mechanism
- The cut predicate (`!/0`) provides a way for controlled backtracking
- The cut predicate always succeeds

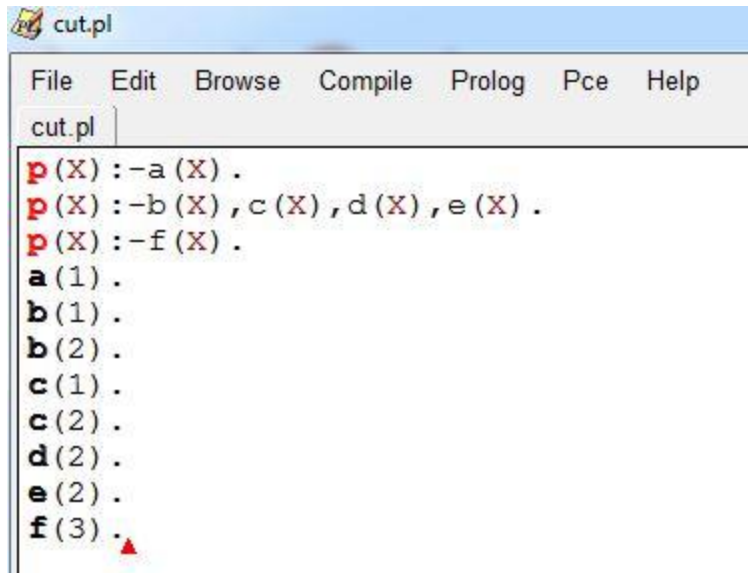
Example without Cut

Example without Cut



```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X):-a(X).
p(X):-b(X),c(X),d(X),e(X).
p(X):-f(X).
a(1).
b(1).
b(2).
c(1).
c(2).
d(2).
e(2).
f(3).
```

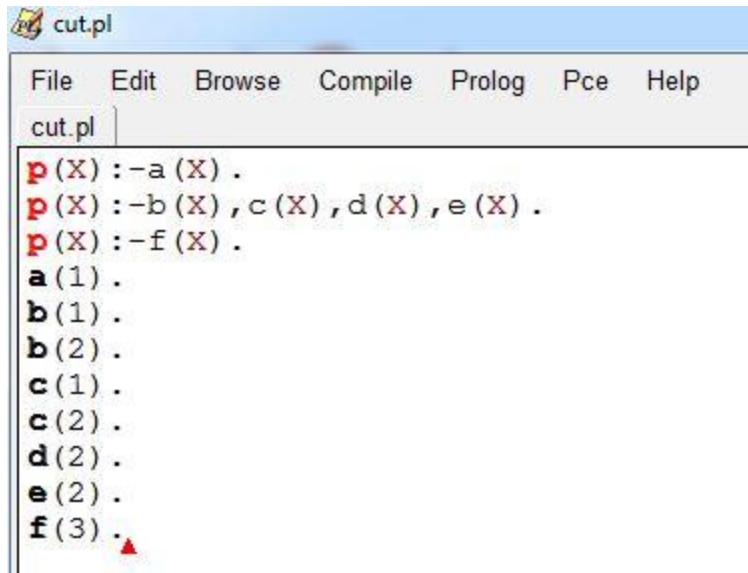
Example without Cut



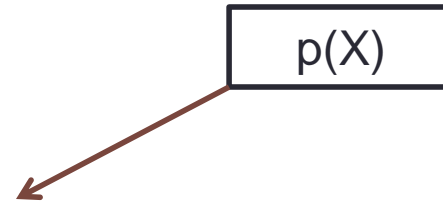
```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X):-a(X).
p(X):-b(X),c(X),d(X),e(X).
p(X):-f(X).
a(1).
b(1).
b(2).
c(1).
c(2).
d(2).
e(2).
f(3).
```

p(X)

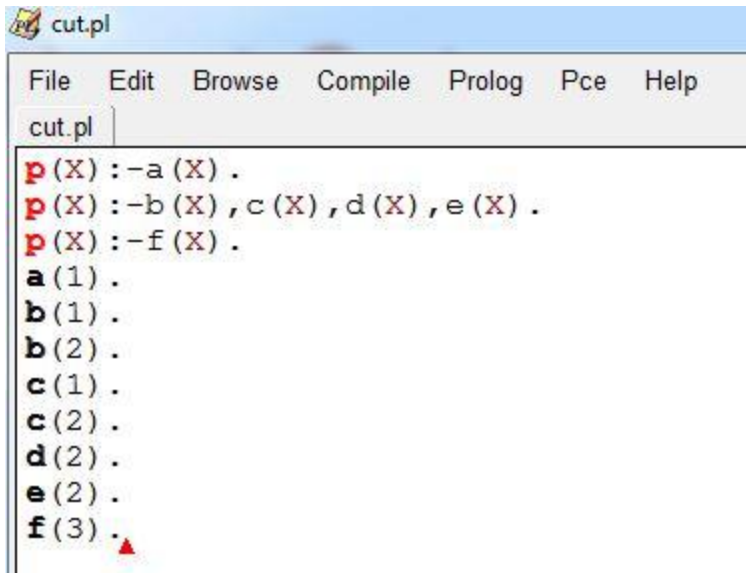
Example without Cut



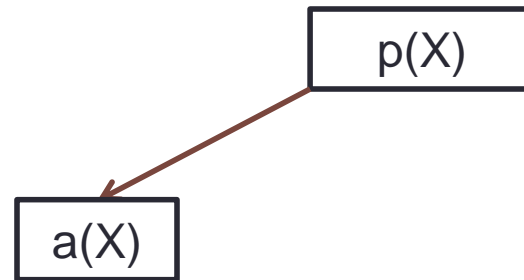
```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X) :- a(X) .
p(X) :- b(X), c(X), d(X), e(X) .
p(X) :- f(X) .
a(1) .
b(1) .
b(2) .
c(1) .
c(2) .
d(2) .
e(2) .
f(3) .
```



Example without Cut

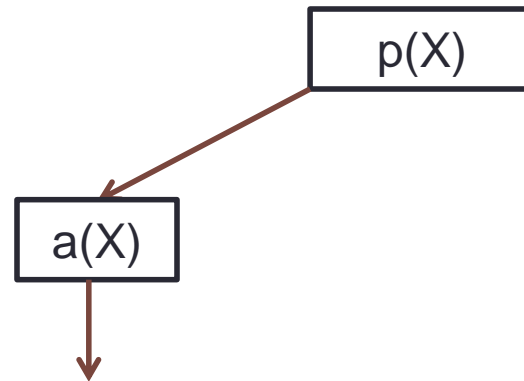


```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X):-a(X).
p(X):-b(X),c(X),d(X),e(X).
p(X):-f(X).
a(1).
b(1).
b(2).
c(1).
c(2).
d(2).
e(2).
f(3).
```



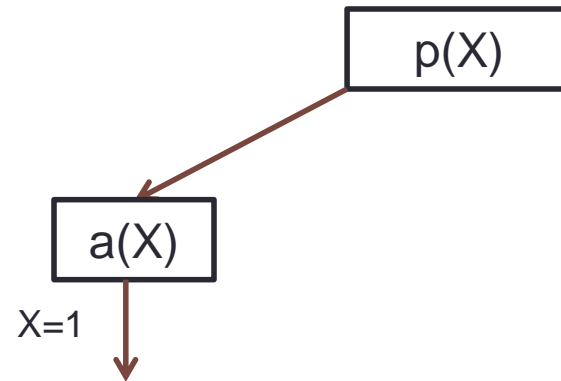
Example without Cut

```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X):-a(X).
p(X):-b(X),c(X),d(X),e(X).
p(X):-f(X).
a(1).
b(1).
b(2).
c(1).
c(2).
d(2).
e(2).
f(3).
```

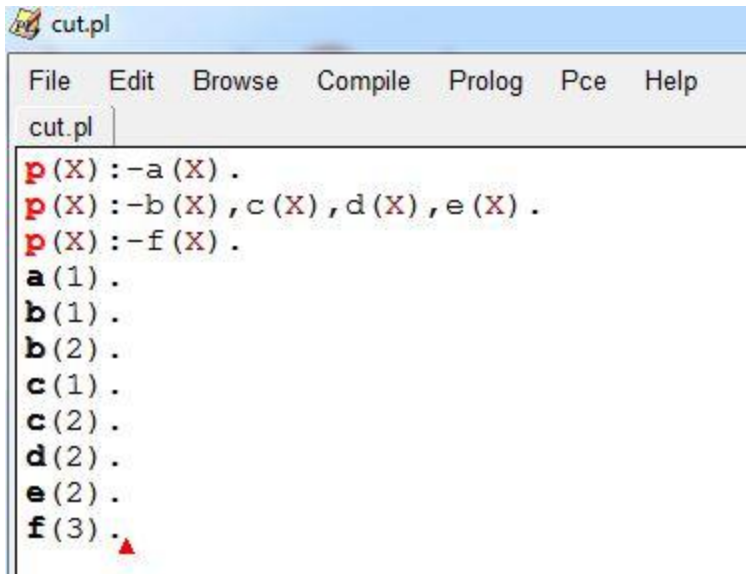


Example without Cut

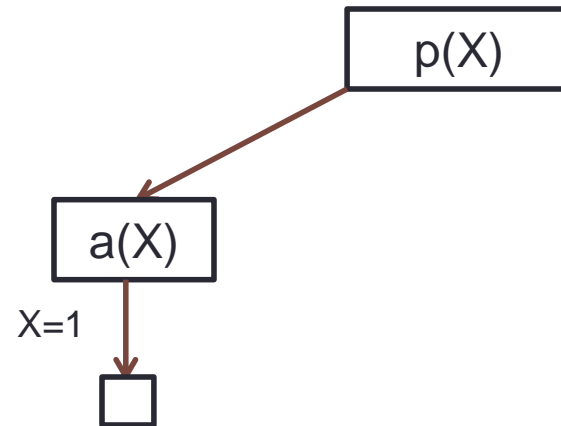
```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X):-a(X).
p(X):-b(X),c(X),d(X),e(X).
p(X):-f(X).
a(1).
b(1).
b(2).
c(1).
c(2).
d(2).
e(2).
f(3).
```



Example without Cut

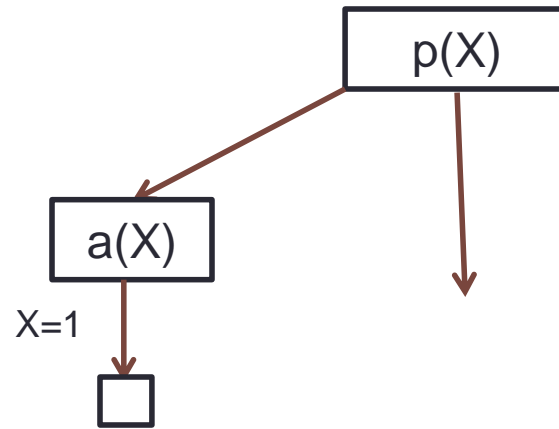


```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X):-a(X).
p(X):-b(X),c(X),d(X),e(X).
p(X):-f(X).
a(1).
b(1).
b(2).
c(1).
c(2).
d(2).
e(2).
f(3).
```



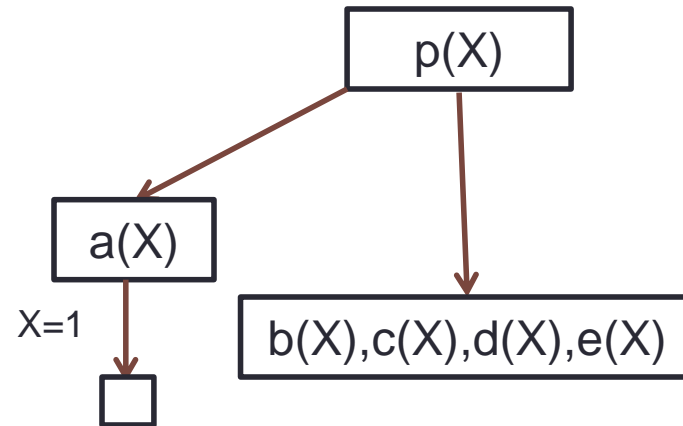
Example without Cut

```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X):-a(X).
p(X):-b(X),c(X),d(X),e(X).
p(X):-f(X).
a(1).
b(1).
b(2).
c(1).
c(2).
d(2).
e(2).
f(3).
```



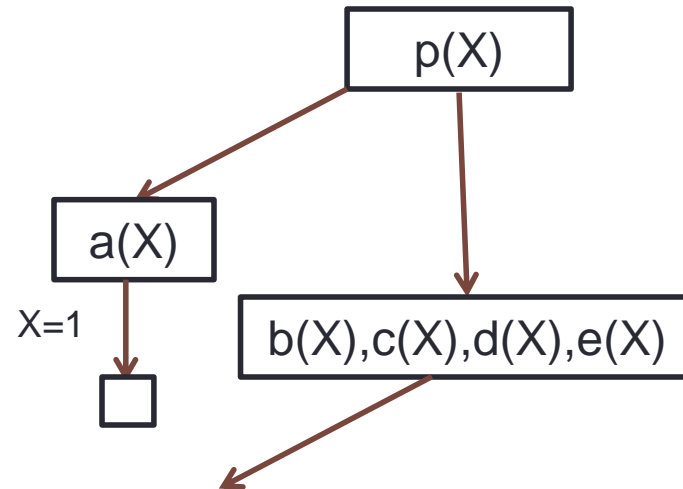
Example without Cut

```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X):-a(X).
p(X):-b(X),c(X),d(X),e(X).
p(X):-f(X).
a(1).
b(1).
b(2).
c(1).
c(2).
d(2).
e(2).
f(3).
```



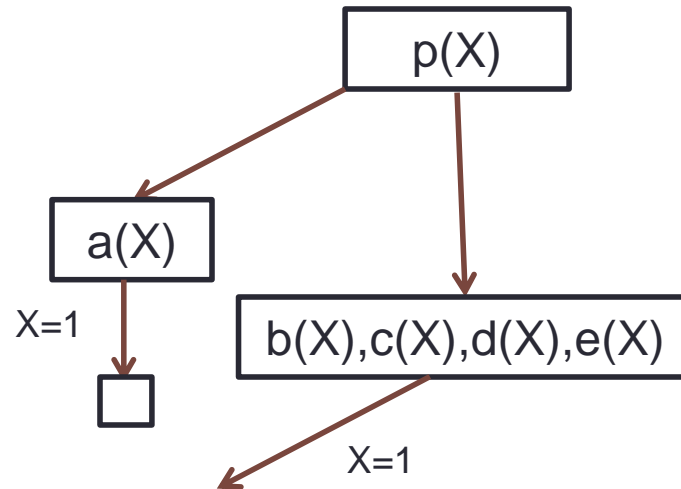
Example without Cut

```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X):-a(X).
p(X):-b(X),c(X),d(X),e(X).
p(X):-f(X).
a(1).
b(1).
b(2).
c(1).
c(2).
d(2).
e(2).
f(3).
```



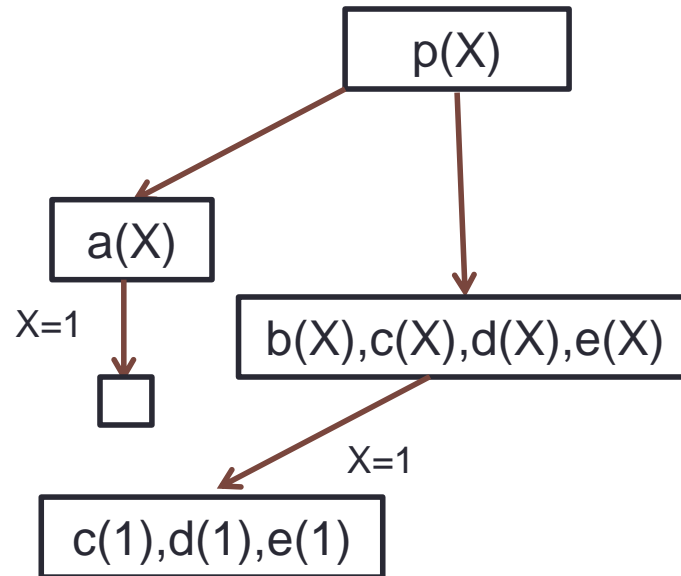
Example without Cut

```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X):-a(X).
p(X):-b(X),c(X),d(X),e(X).
p(X):-f(X).
a(1).
b(1).
b(2).
c(1).
c(2).
d(2).
e(2).
f(3).
```



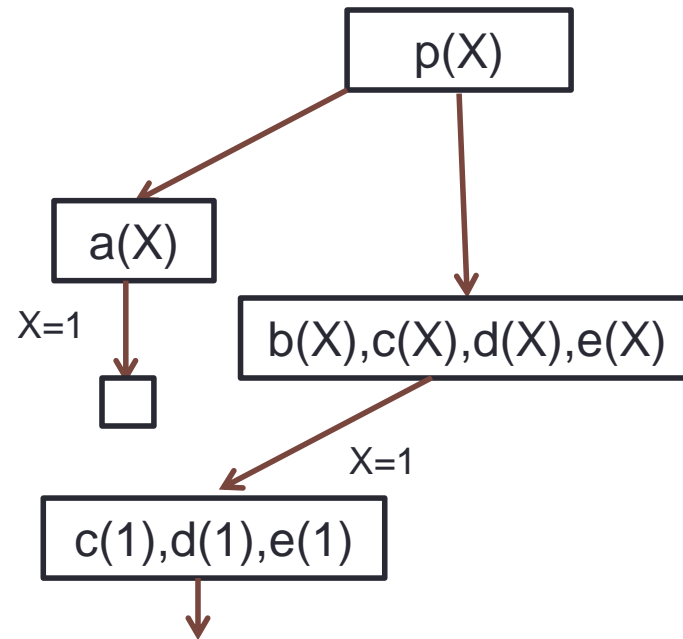
Example without Cut

```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X):-a(X).
p(X):-b(X),c(X),d(X),e(X).
p(X):-f(X).
a(1).
b(1).
b(2).
c(1).
c(2).
d(2).
e(2).
f(3).
```



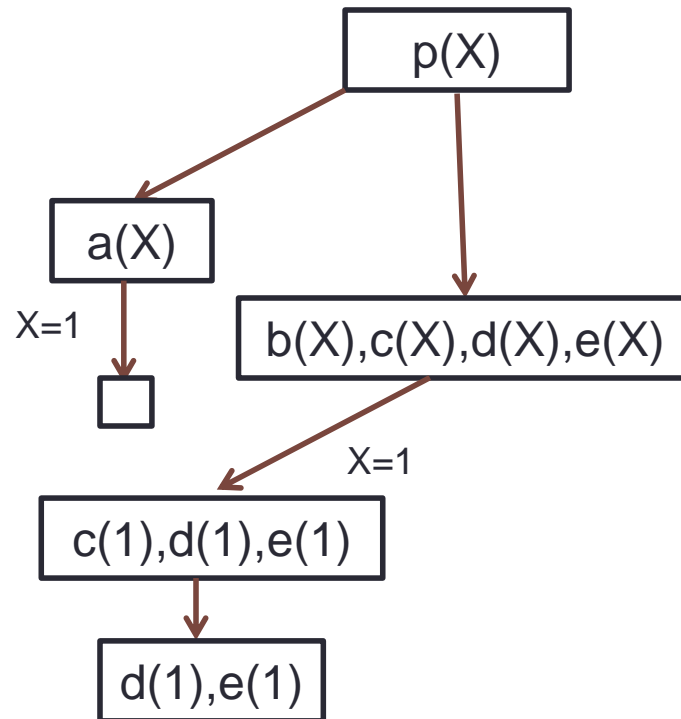
Example without Cut

```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X):-a(X).
p(X):-b(X),c(X),d(X),e(X).
p(X):-f(X).
a(1).
b(1).
b(2).
c(1).
c(2).
d(2).
e(2).
f(3).
```



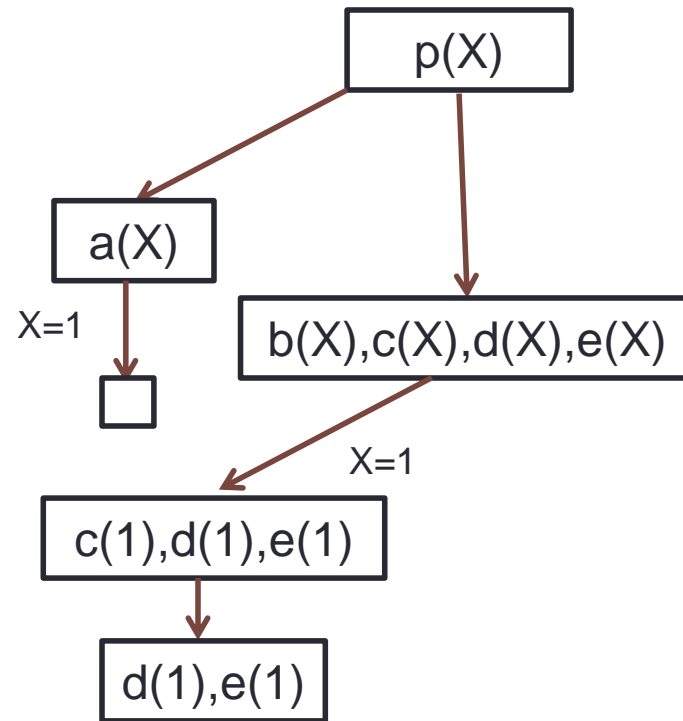
Example without Cut

```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X):-a(X).
p(X):-b(X),c(X),d(X),e(X).
p(X):-f(X).
a(1).
b(1).
b(2).
c(1).
c(2).
d(2).
e(2).
f(3).
```



Example without Cut

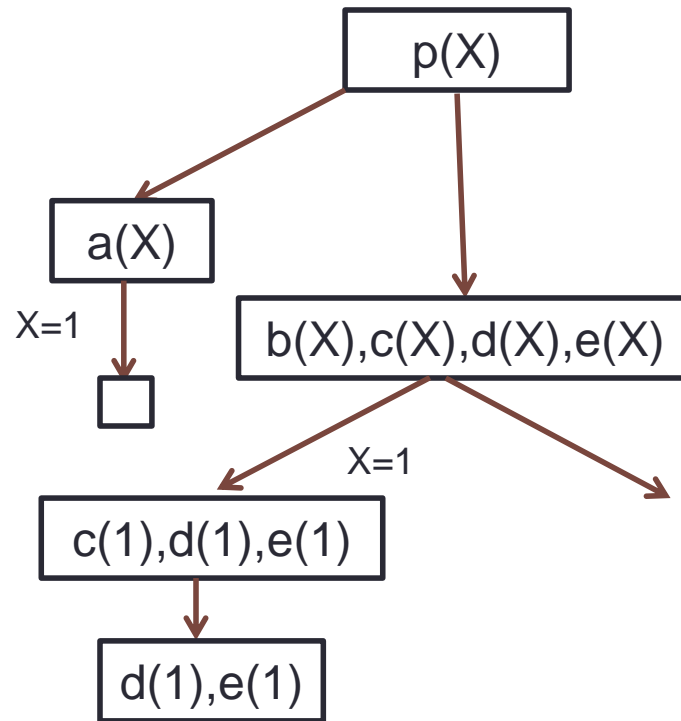
```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X):-a(X).
p(X):-b(X),c(X),d(X),e(X).
p(X):-f(X).
a(1).
b(1).
b(2).
c(1).
c(2).
d(2).
e(2).
f(3).
```



Can't be satisfied

Example without Cut

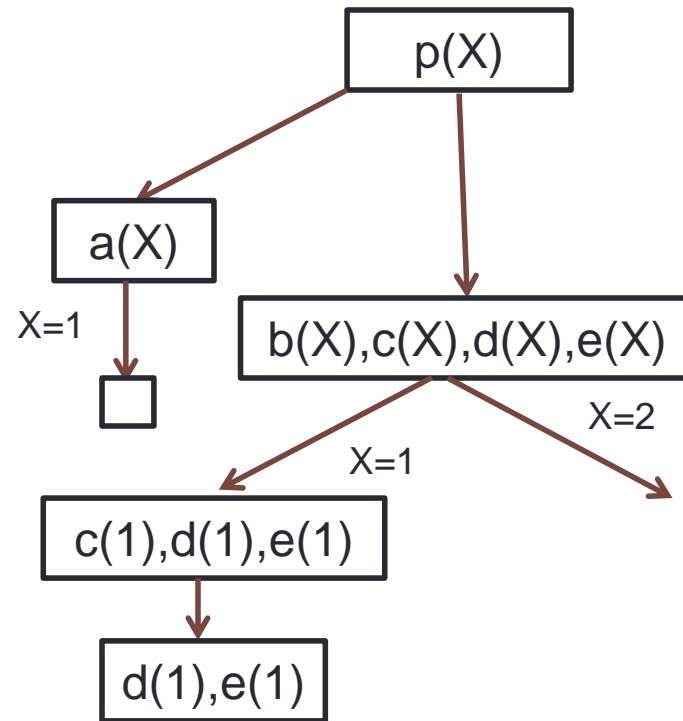
```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X):-a(X).
p(X):-b(X),c(X),d(X),e(X).
p(X):-f(X).
a(1).
b(1).
b(2).
c(1).
c(2).
d(2).
e(2).
f(3).
```



Can't be satisfied

Example without Cut

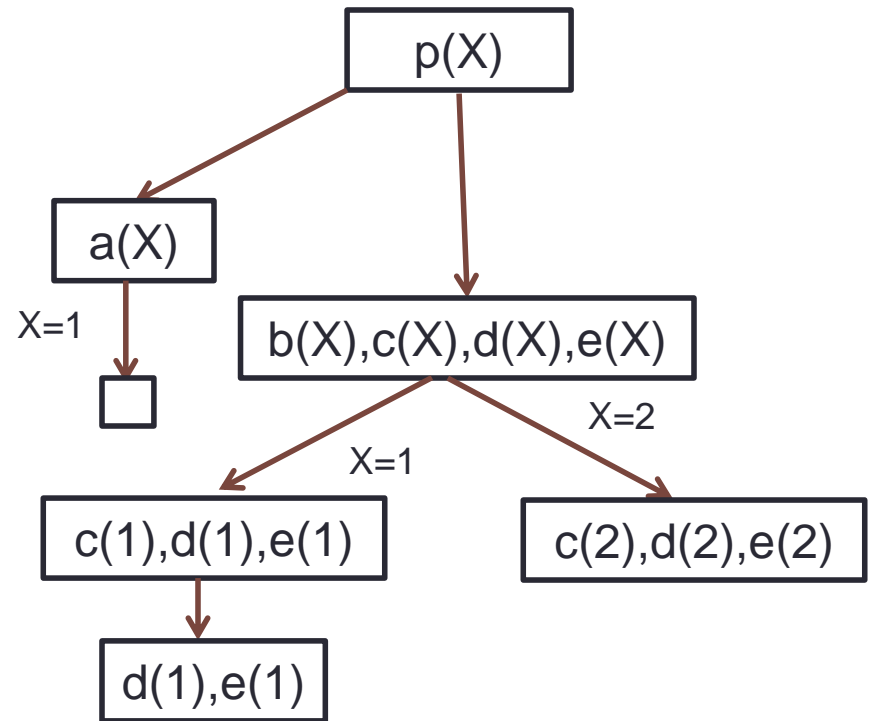
```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X):-a(X).
p(X):-b(X),c(X),d(X),e(X).
p(X):-f(X).
a(1).
b(1).
b(2).
c(1).
c(2).
d(2).
e(2).
f(3).
```



Can't be satisfied

Example without Cut

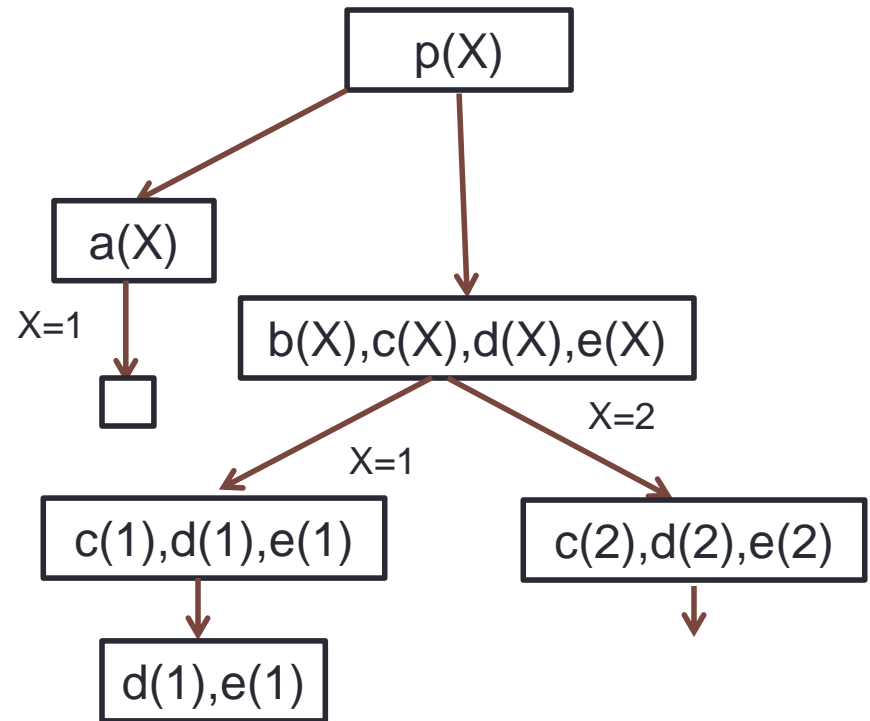
```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X):-a(X).
p(X):-b(X),c(X),d(X),e(X).
p(X):-f(X).
a(1).
b(1).
b(2).
c(1).
c(2).
d(2).
e(2).
f(3).
```



Can't be satisfied

Example without Cut

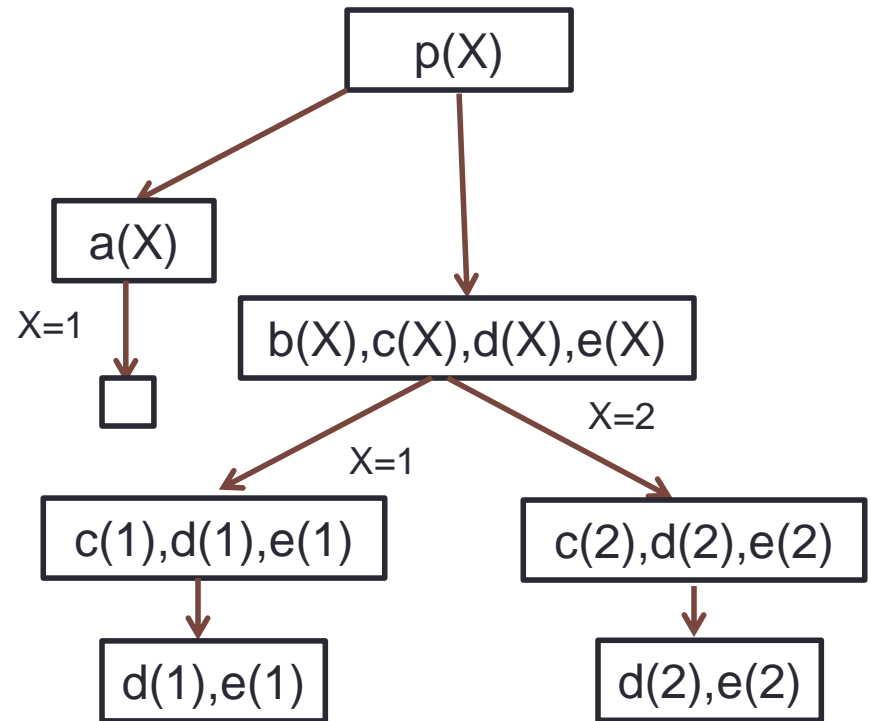
```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X):-a(X).
p(X):-b(X),c(X),d(X),e(X).
p(X):-f(X).
a(1).
b(1).
b(2).
c(1).
c(2).
d(2).
e(2).
f(3).
```



Can't be satisfied

Example without Cut

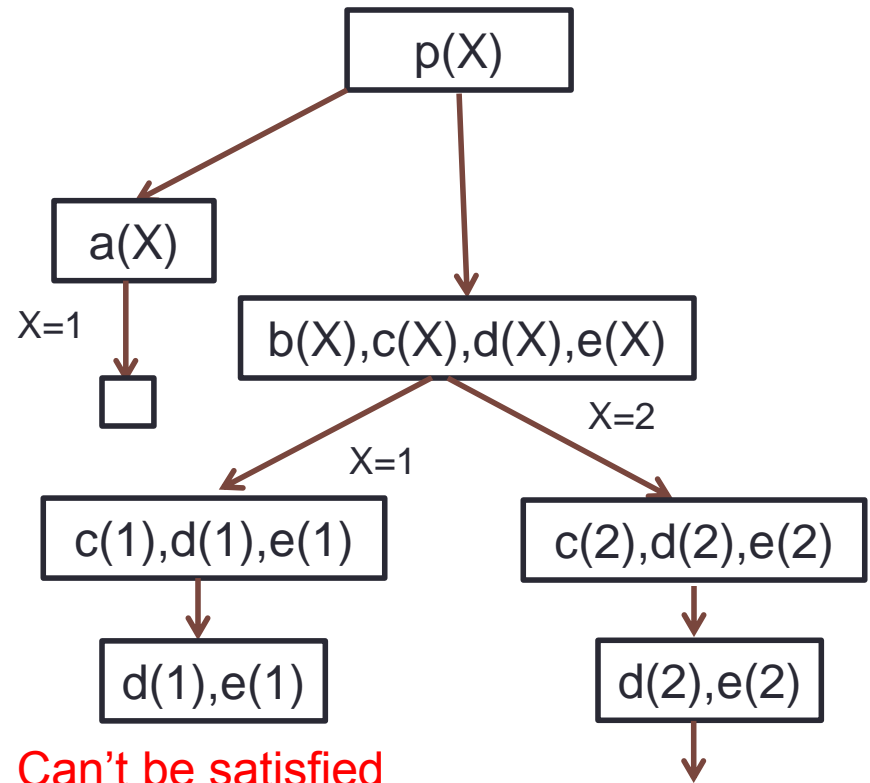
```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X) :- a(X) .
p(X) :- b(X), c(X), d(X), e(X) .
p(X) :- f(X) .
a(1) .
b(1) .
b(2) .
c(1) .
c(2) .
d(2) .
e(2) .
f(3) .
```



Can't be satisfied

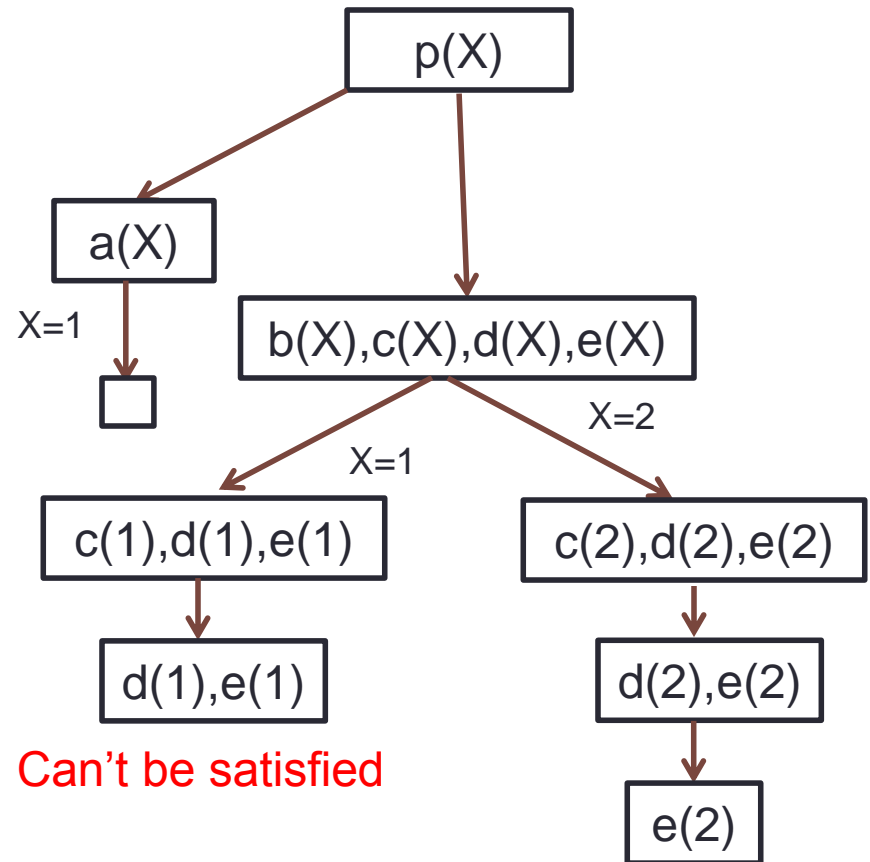
Example without Cut

```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X) :- a(X) .
p(X) :- b(X), c(X), d(X), e(X) .
p(X) :- f(X) .
a(1) .
b(1) .
b(2) .
c(1) .
c(2) .
d(2) .
e(2) .
f(3) .
```



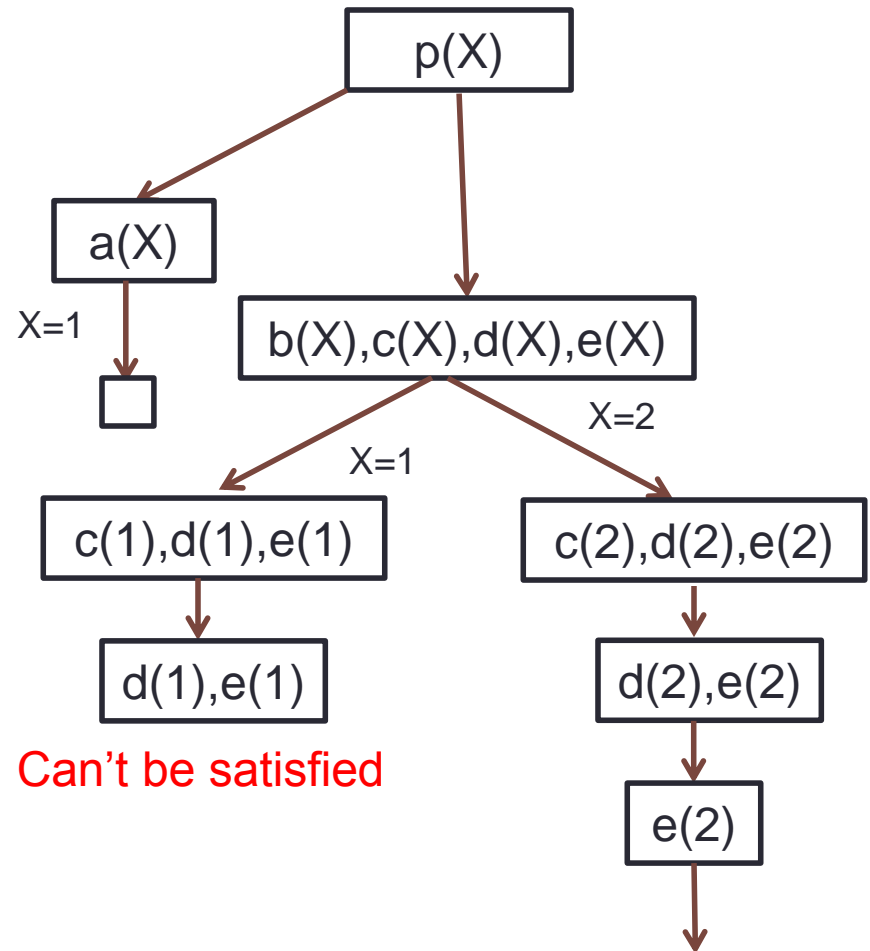
Example without Cut

```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X):-a(X).
p(X):-b(X),c(X),d(X),e(X).
p(X):-f(X).
a(1).
b(1).
b(2).
c(1).
c(2).
d(2).
e(2).
f(3).
```



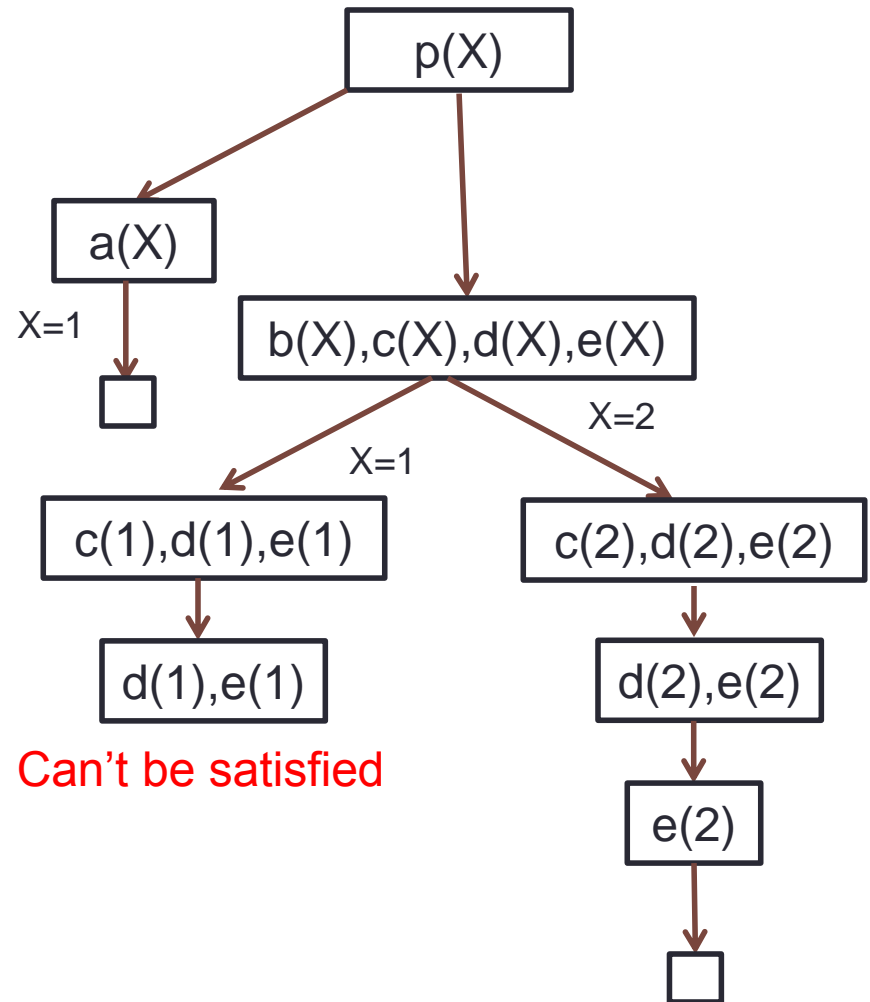
Example without Cut

```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X) :- a(X) .
p(X) :- b(X), c(X), d(X), e(X) .
p(X) :- f(X) .
a(1) .
b(1) .
b(2) .
c(1) .
c(2) .
d(2) .
e(2) .
f(3) .
```



Example without Cut

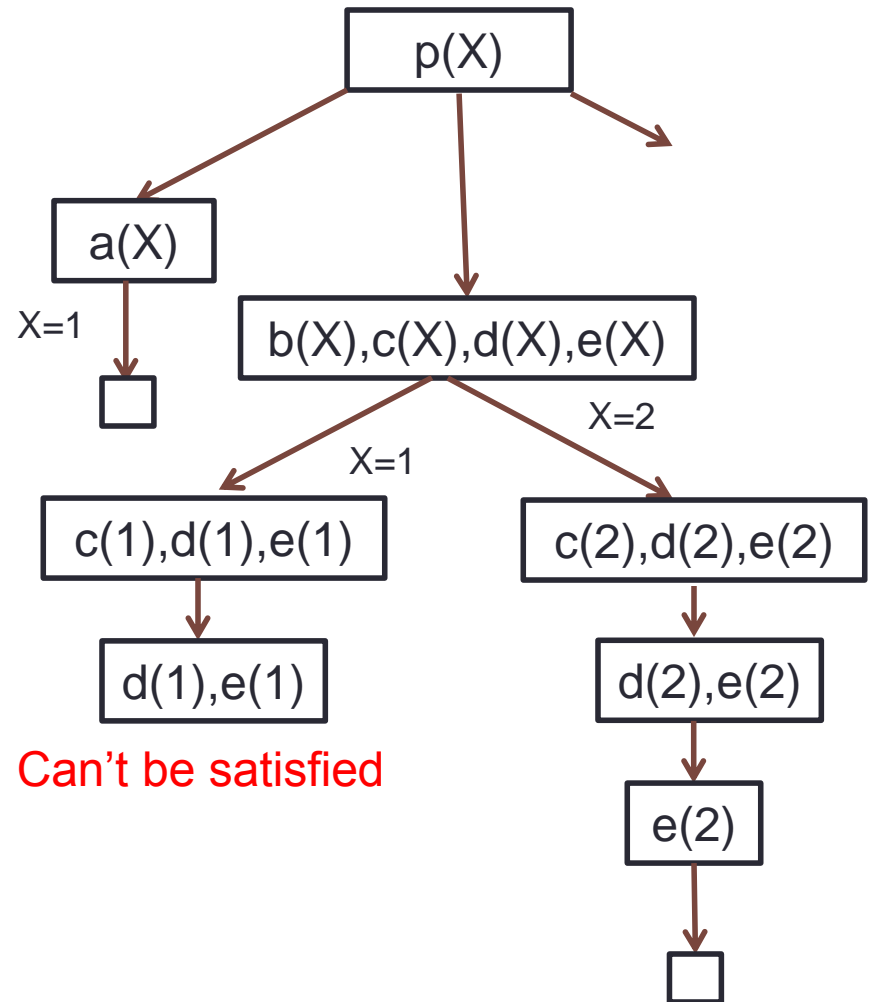
```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X):-a(X).
p(X):-b(X),c(X),d(X),e(X).
p(X):-f(X).
a(1).
b(1).
b(2).
c(1).
c(2).
d(2).
e(2).
f(3).
```



Can't be satisfied

Example without Cut

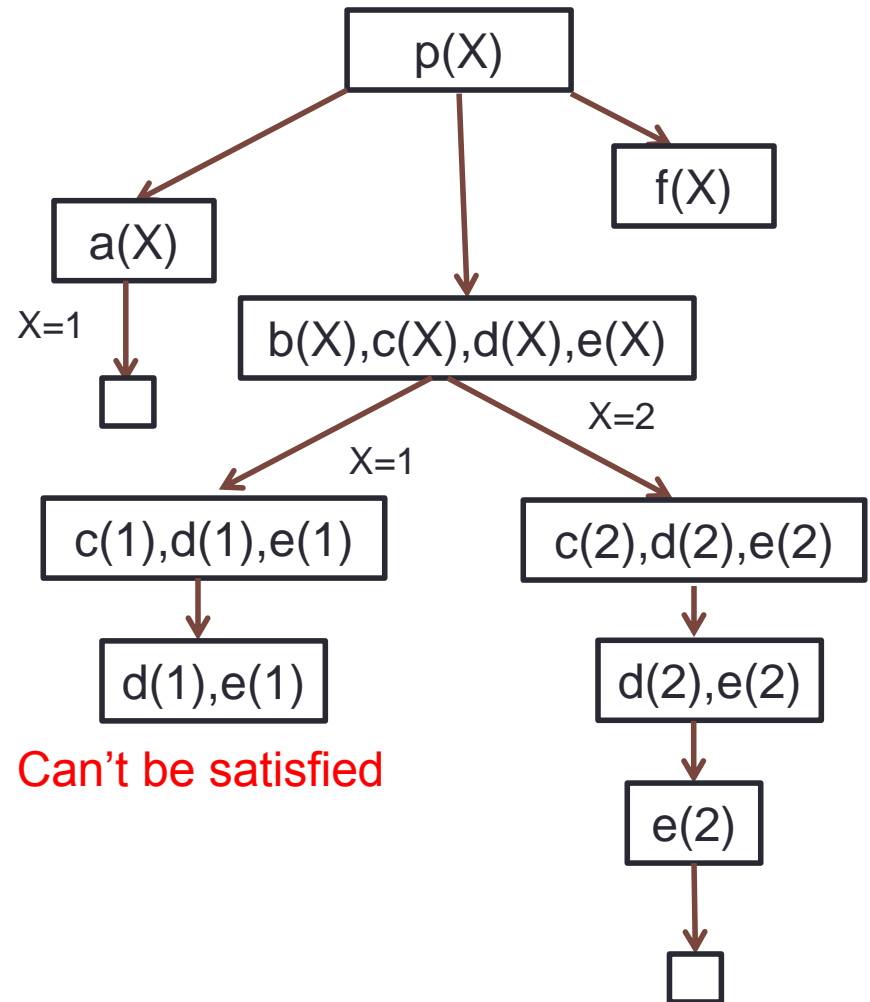
```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X):-a(X).
p(X):-b(X),c(X),d(X),e(X).
p(X):-f(X).
a(1).
b(1).
b(2).
c(1).
c(2).
d(2).
e(2).
f(3).
```



Can't be satisfied

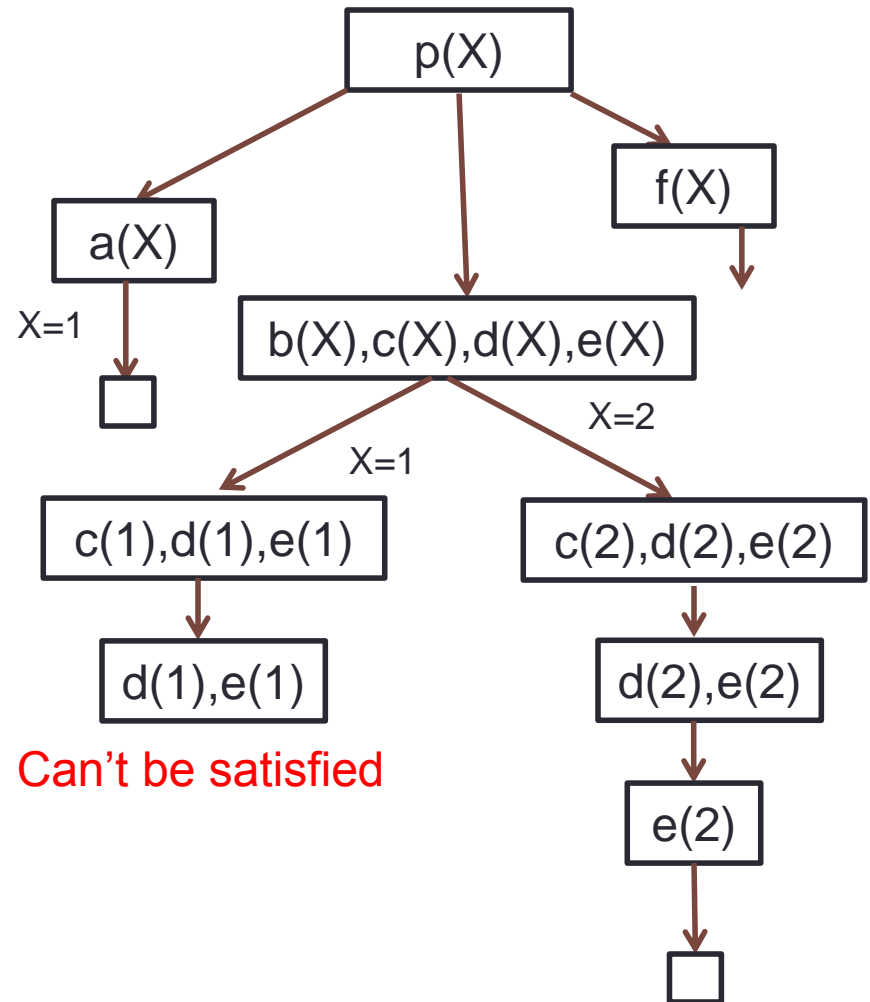
Example without Cut

```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X):-a(X).
p(X):-b(X),c(X),d(X),e(X).
p(X):-f(X).
a(1).
b(1).
b(2).
c(1).
c(2).
d(2).
e(2).
f(3).
```



Example without Cut

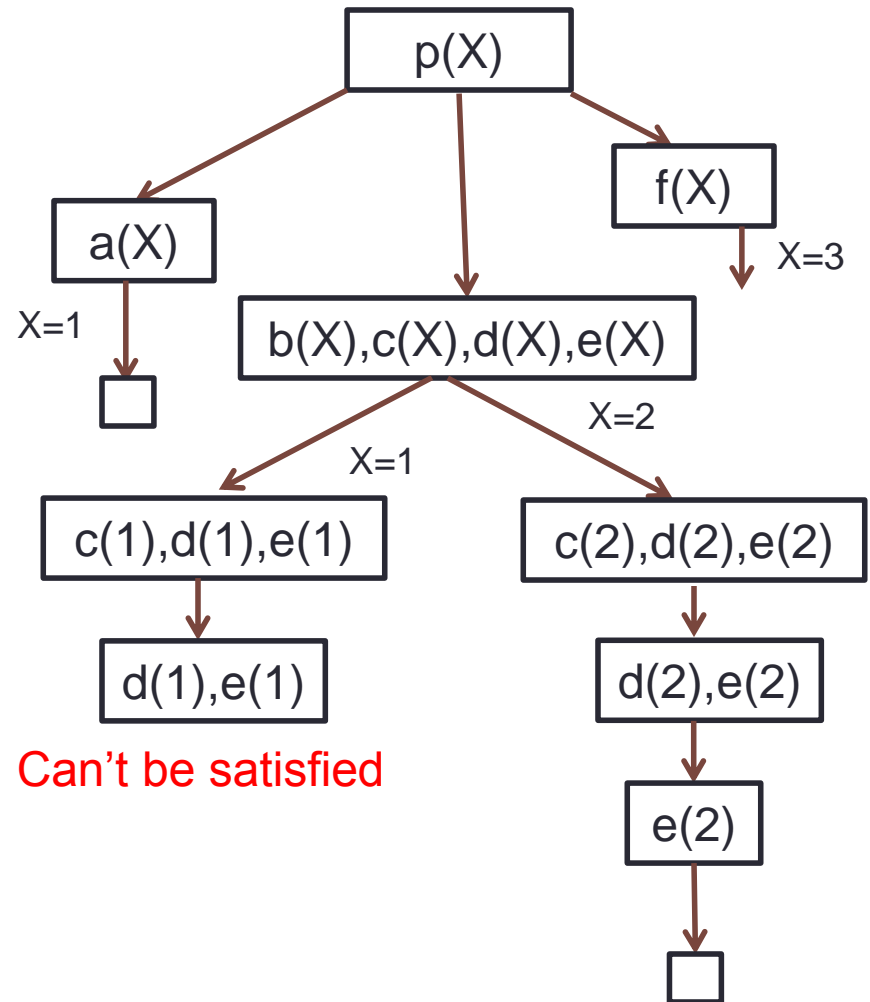
```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X):-a(X).
p(X):-b(X),c(X),d(X),e(X).
p(X):-f(X).
a(1).
b(1).
b(2).
c(1).
c(2).
d(2).
e(2).
f(3).
```



Can't be satisfied

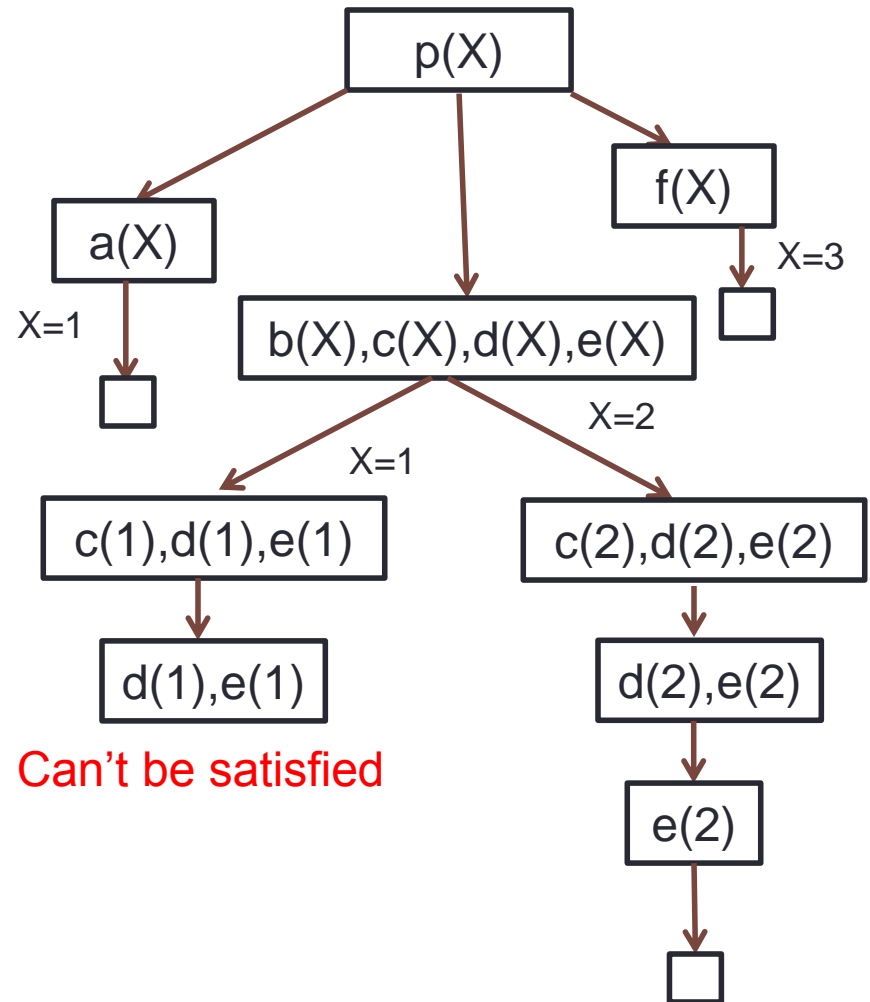
Example without Cut

```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X):-a(X).
p(X):-b(X),c(X),d(X),e(X).
p(X):-f(X).
a(1).
b(1).
b(2).
c(1).
c(2).
d(2).
e(2).
f(3).
```



Example without Cut

```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X):-a(X).
p(X):-b(X),c(X),d(X),e(X).
p(X):-f(X).
a(1).
b(1).
b(2).
c(1).
c(2).
d(2).
e(2).
f(3).
```

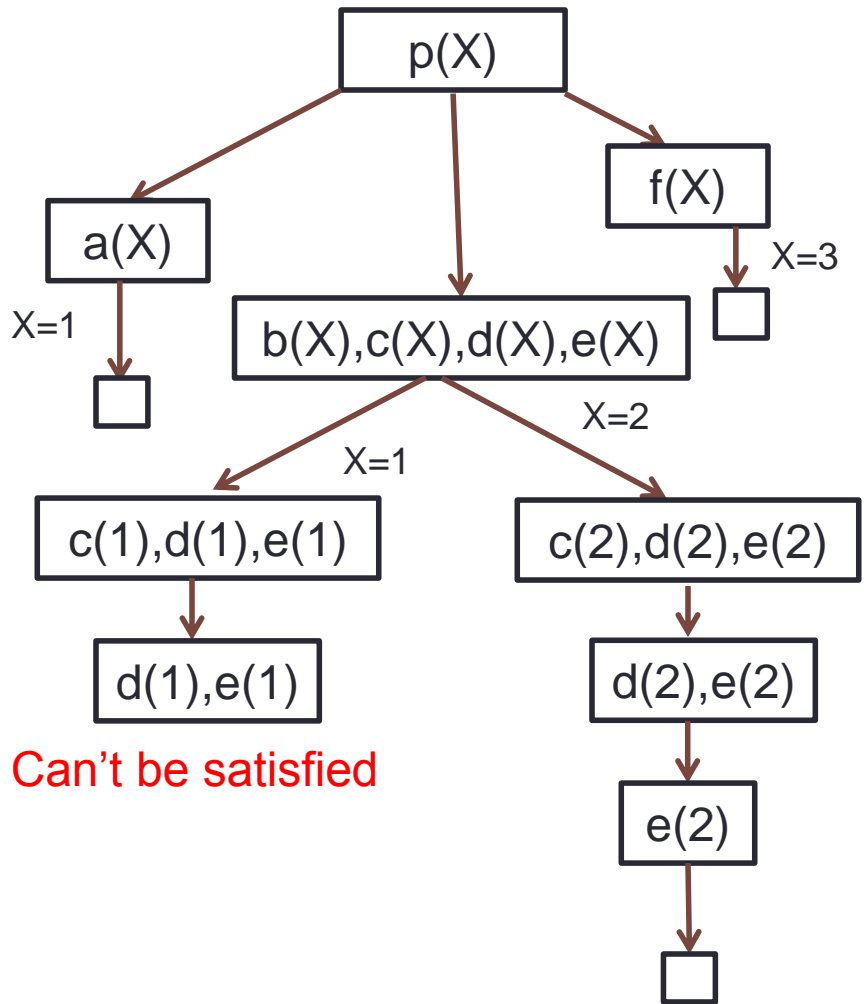


Can't be satisfied

Example without Cut

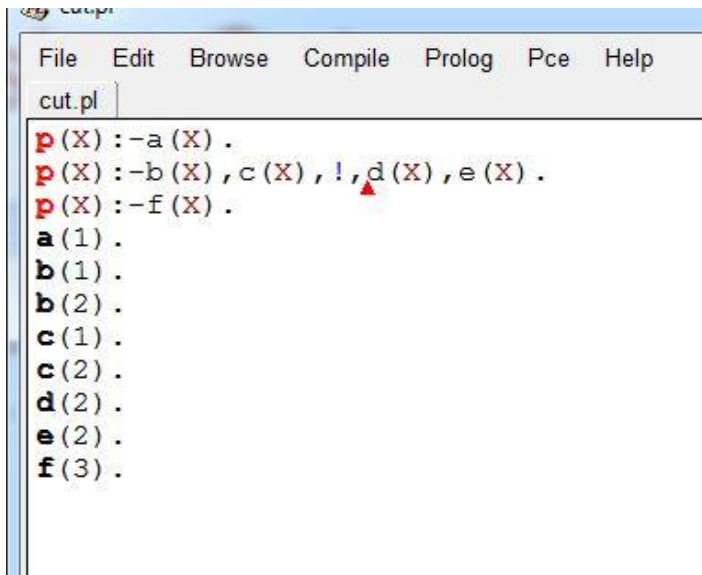
```
cut.pl
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X) :- a(X) .
p(X) :- b(X), c(X), d(X), e(X) .
p(X) :- f(X) .
a(1) .
b(1) .
b(2) .
c(1) .
c(2) .
d(2) .
e(2) .
f(3) .
```

```
SWI-Prolog (AMD64, Multi-threaded, version 7.6.4)
File Edit Settings Run Debug Help
?- p(X) .
X = 1 ;
X = 2 ;
X = 3 ;
?-
```



Example using Cut

Example using Cut



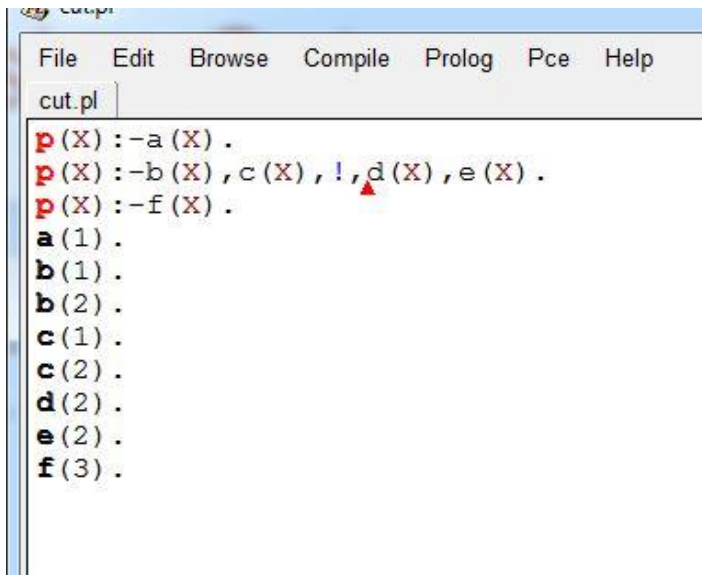
```
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X) :- a(X) .
p(X) :- b(X), c(X), !, d(X), e(X) .
p(X) :- f(X) .
a(1) .
b(1) .
b(2) .
c(1) .
c(2) .
d(2) .
e(2) .
f(3) .
```

Example using Cut

$p(X)$

```
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X) :- a(X) .
p(X) :- b(X), c(X), !, d(X), e(X) .
p(X) :- f(X) .
a(1) .
b(1) .
b(2) .
c(1) .
c(2) .
d(2) .
e(2) .
f(3) .
```

Example using Cut

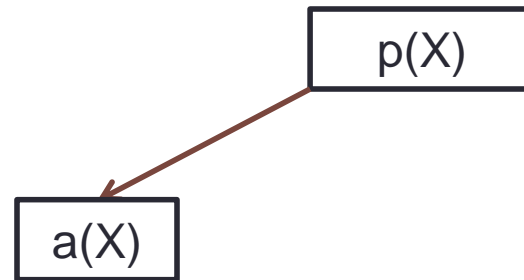


```
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X) :- a(X) .
p(X) :- b(X) , c(X) , ! , d(X) , e(X) .
p(X) :- f(X) .
a(1) .
b(1) .
b(2) .
c(1) .
c(2) .
d(2) .
e(2) .
f(3) .
```

p(X)

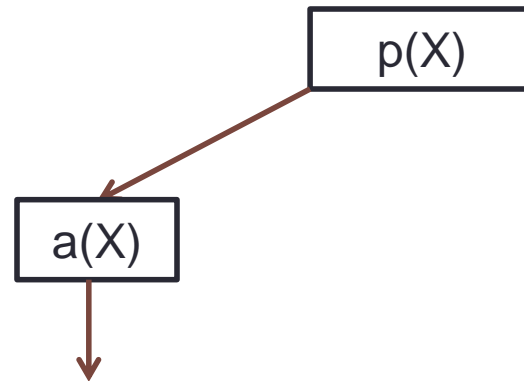
Example using Cut

```
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X) :- a(X) .
p(X) :- b(X), c(X), !, d(X), e(X) .
p(X) :- f(X) .
a(1) .
b(1) .
b(2) .
c(1) .
c(2) .
d(2) .
e(2) .
f(3) .
```



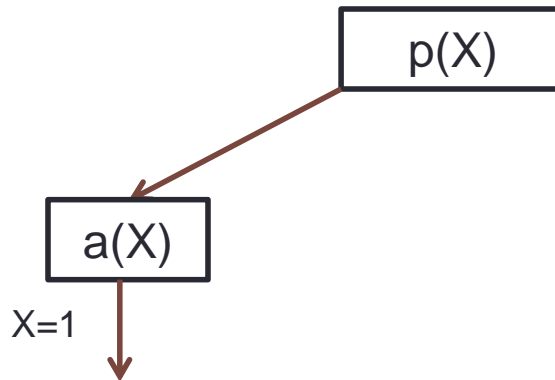
Example using Cut

```
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X) :- a(X) .
p(X) :- b(X), c(X), !, d(X), e(X) .
p(X) :- f(X) .
a(1) .
b(1) .
b(2) .
c(1) .
c(2) .
d(2) .
e(2) .
f(3) .
```



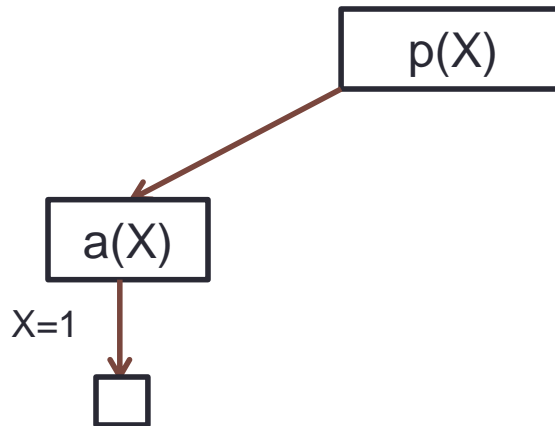
Example using Cut

```
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X) :- a(X) .
p(X) :- b(X), c(X), !, d(X), e(X) .
p(X) :- f(X) .
a(1) .
b(1) .
b(2) .
c(1) .
c(2) .
d(2) .
e(2) .
f(3) .
```



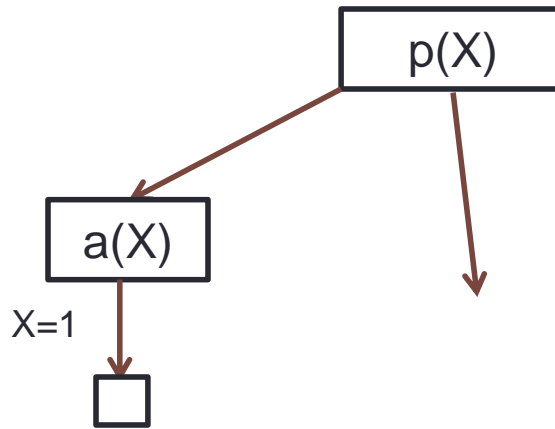
Example using Cut

```
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X) :- a(X) .
p(X) :- b(X), c(X), !, d(X), e(X) .
p(X) :- f(X) .
a(1) .
b(1) .
b(2) .
c(1) .
c(2) .
d(2) .
e(2) .
f(3) .
```



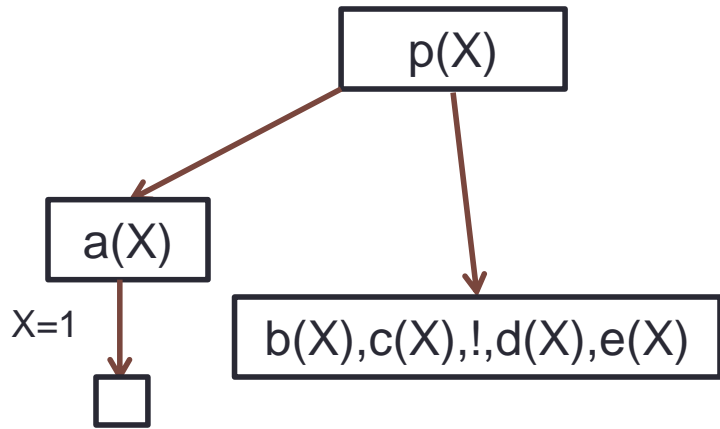
Example using Cut

```
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X) :- a(X) .
p(X) :- b(X), c(X), !, d(X), e(X) .
p(X) :- f(X) .
a(1) .
b(1) .
b(2) .
c(1) .
c(2) .
d(2) .
e(2) .
f(3) .
```



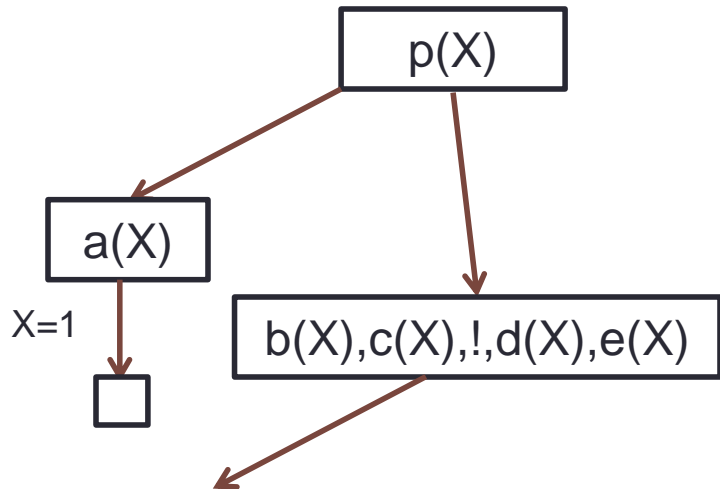
Example using Cut

```
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X) :- a(X) .
p(X) :- b(X), c(X), !, d(X), e(X) .
p(X) :- f(X) .
a(1) .
b(1) .
b(2) .
c(1) .
c(2) .
d(2) .
e(2) .
f(3) .
```



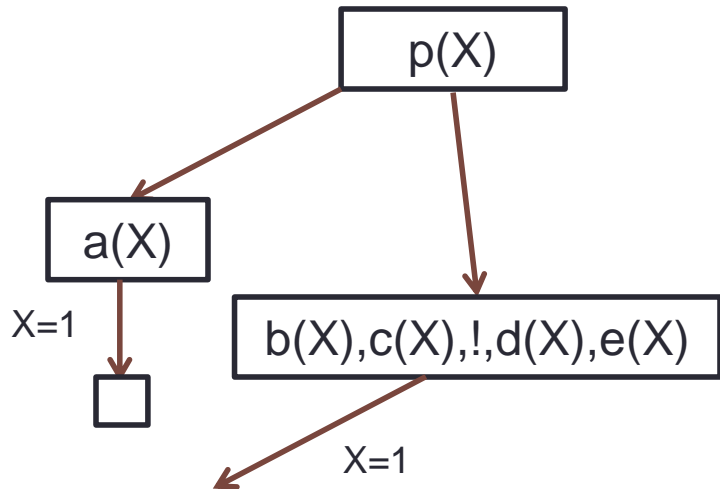
Example using Cut

```
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X) :- a(X) .
p(X) :- b(X), c(X), !, d(X), e(X) .
p(X) :- f(X) .
a(1) .
b(1) .
b(2) .
c(1) .
c(2) .
d(2) .
e(2) .
f(3) .
```



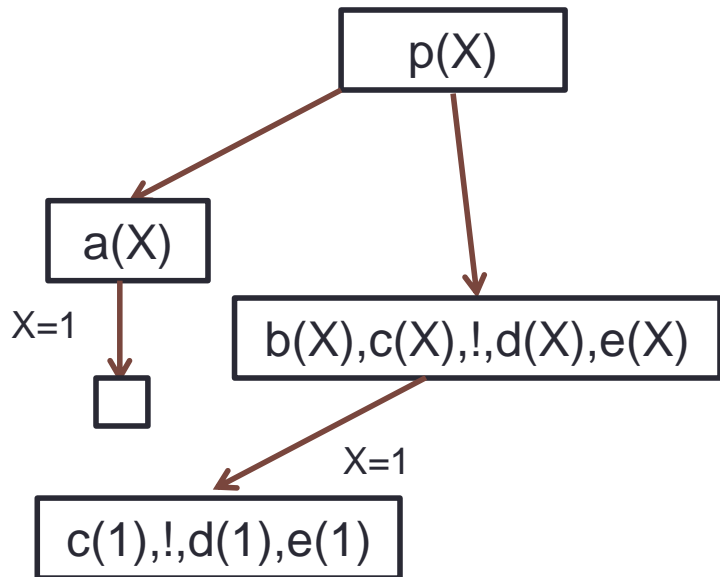
Example using Cut

```
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X) :- a(X) .
p(X) :- b(X), c(X), !, d(X), e(X) .
p(X) :- f(X) .
a(1) .
b(1) .
b(2) .
c(1) .
c(2) .
d(2) .
e(2) .
f(3) .
```



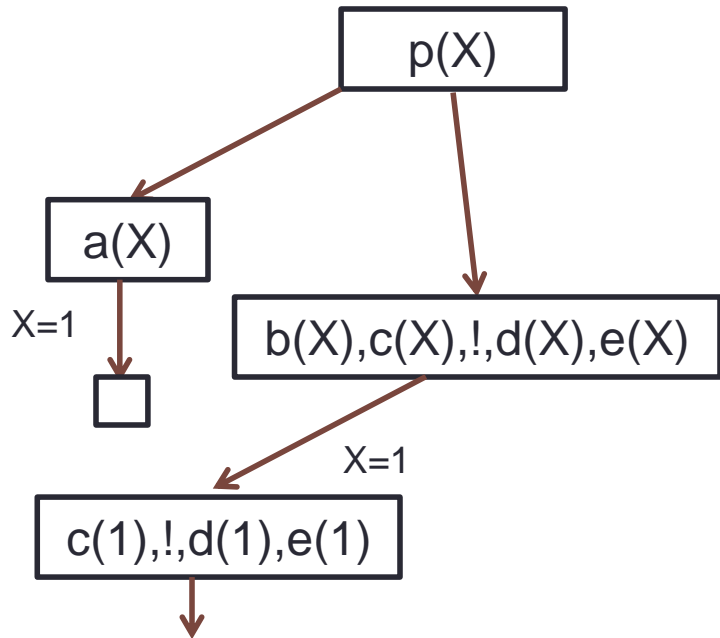
Example using Cut

```
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X) :- a(X) .
p(X) :- b(X), c(X), !, d(X), e(X) .
p(X) :- f(X) .
a(1) .
b(1) .
b(2) .
c(1) .
c(2) .
d(2) .
e(2) .
f(3) .
```



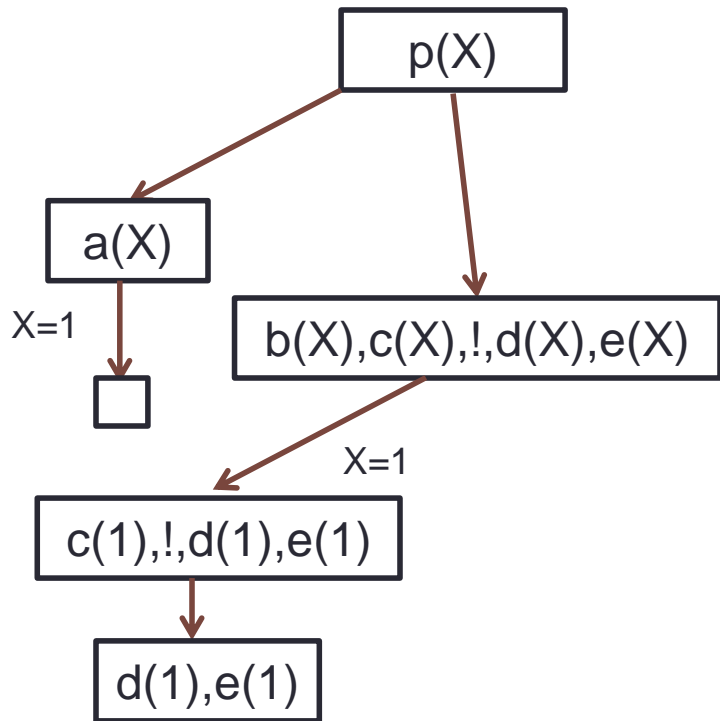
Example using Cut

```
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X) :- a(X) .
p(X) :- b(X), c(X), !, d(X), e(X) .
p(X) :- f(X) .
a(1) .
b(1) .
b(2) .
c(1) .
c(2) .
d(2) .
e(2) .
f(3) .
```



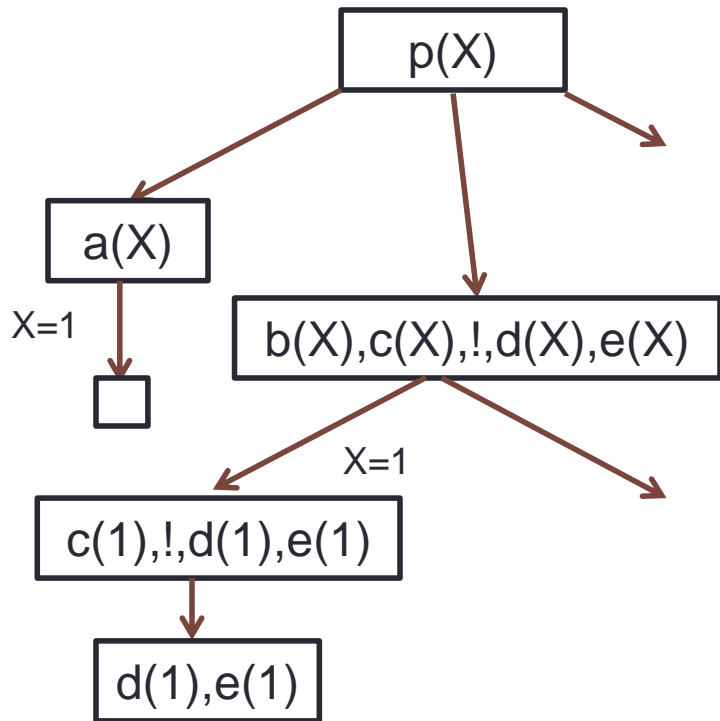
Example using Cut

```
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X) :- a(X) .
p(X) :- b(X), c(X), !, d(X), e(X) .
p(X) :- f(X) .
a(1) .
b(1) .
b(2) .
c(1) .
c(2) .
d(2) .
e(2) .
f(3) .
```



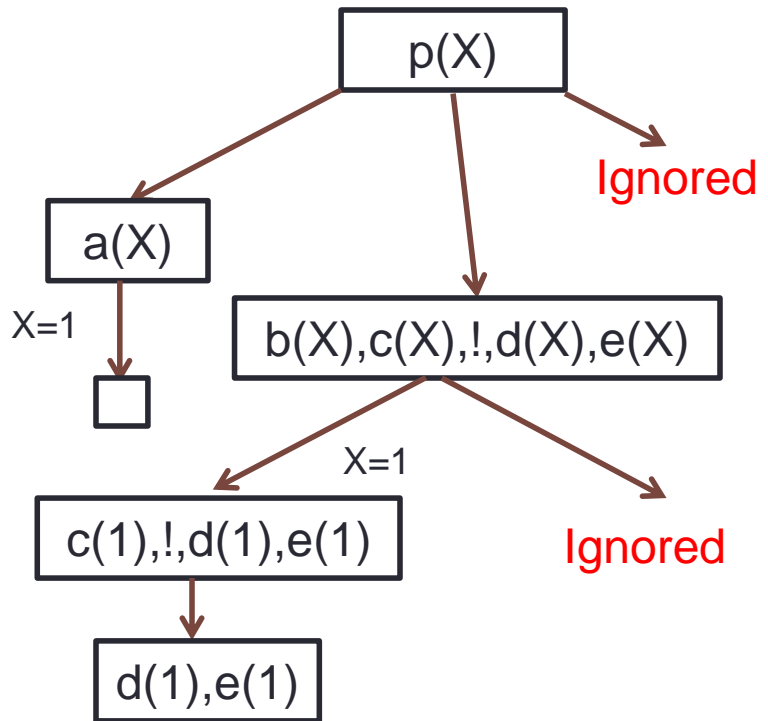
Example using Cut

```
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X) :- a(X) .
p(X) :- b(X), c(X), !, d(X), e(X) .
p(X) :- f(X) .
a(1) .
b(1) .
b(2) .
c(1) .
c(2) .
d(2) .
e(2) .
f(3) .
```



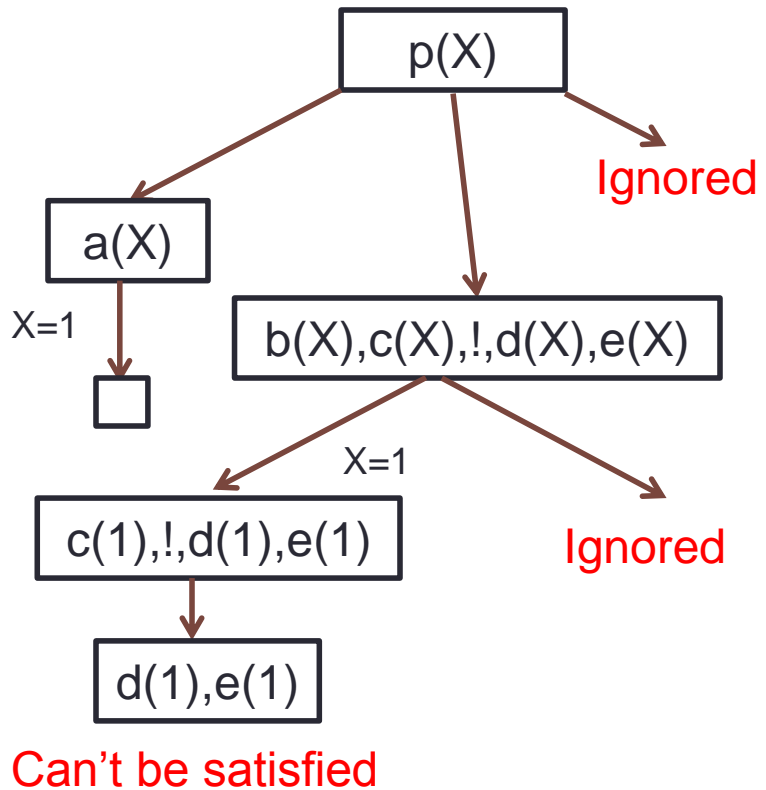
Example using Cut

```
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X) :- a(X) .
p(X) :- b(X), c(X), !, d(X), e(X) .
p(X) :- f(X) .
a(1) .
b(1) .
b(2) .
c(1) .
c(2) .
d(2) .
e(2) .
f(3) .
```



Example using Cut

```
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X) :- a(X) .
p(X) :- b(X), c(X), !, d(X), e(X) .
p(X) :- f(X) .
a(1) .
b(1) .
b(2) .
c(1) .
c(2) .
d(2) .
e(2) .
f(3) .
```



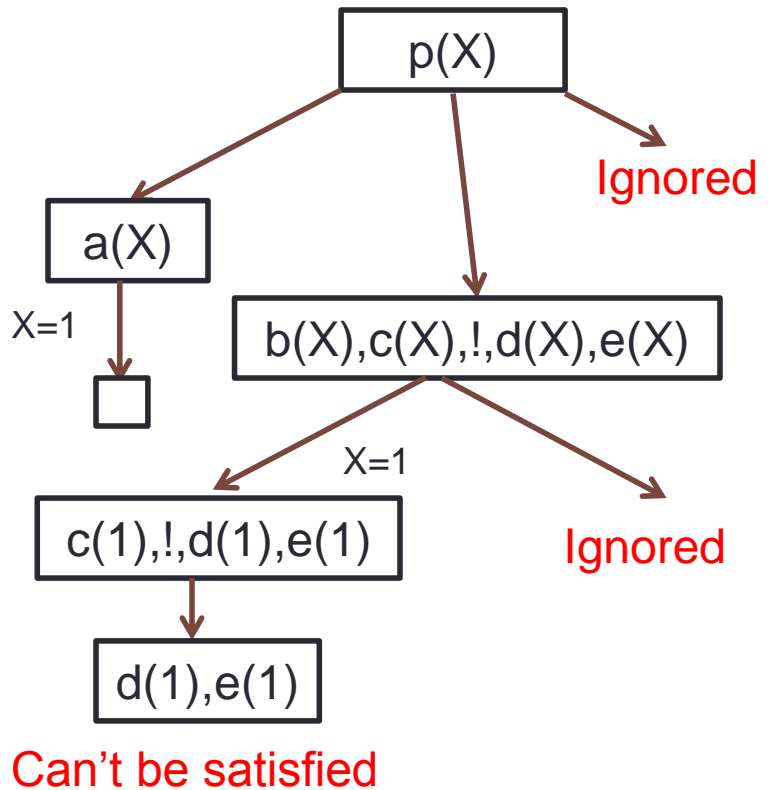
Example using Cut

```
File Edit Browse Compile Prolog Pce Help
cut.pl
p(X) :- a(X) .
p(X) :- b(X), c(X), !, d(X), e(X) .
p(X) :- f(X) .
a(1) .
b(1) .
b(2) .
c(1) .
c(2) .
d(2) .
e(2) .
f(3) .
```

```
SWI-Prolog (Am64, multi-threaded, version 7.0.4)
File Edit Settings Run Debug Help

?- p(X) .
X = 1 .

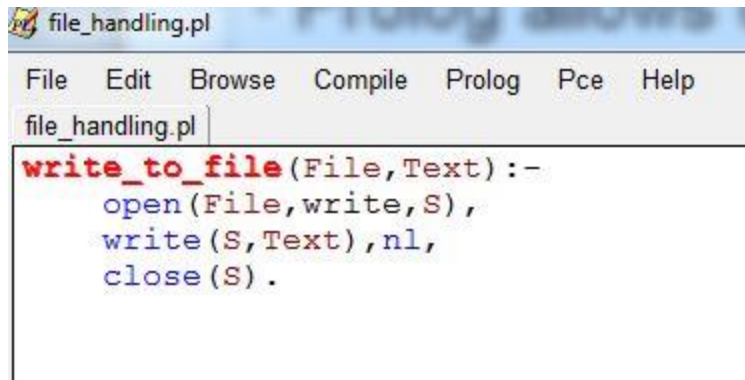
?-
```



File handling

- Prolog allows us to read data from a file and write data to a file.
- Operations on file: read, write and append

Writing to a file

A screenshot of a Prolog IDE window titled 'file_handling.pl'. The window has a menu bar with 'File', 'Edit', 'Browse', 'Compile', 'Prolog', 'Pce', and 'Help'. Below the menu bar is a tab labeled 'file_handling.pl'. The main text area contains the following Prolog code:

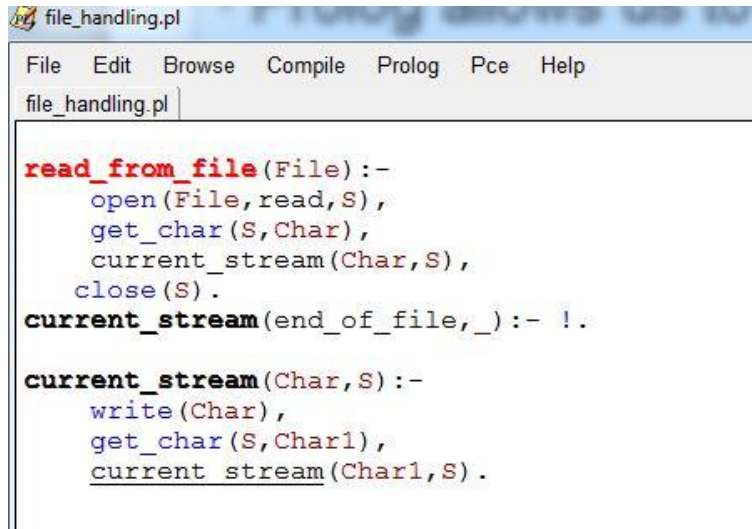
```
write_to_file(File,Text):-  
    open(File,write,S),  
    write(S,Text),nl,  
    close(S).
```

File handling

Reading from a file

File handling

Reading from a file



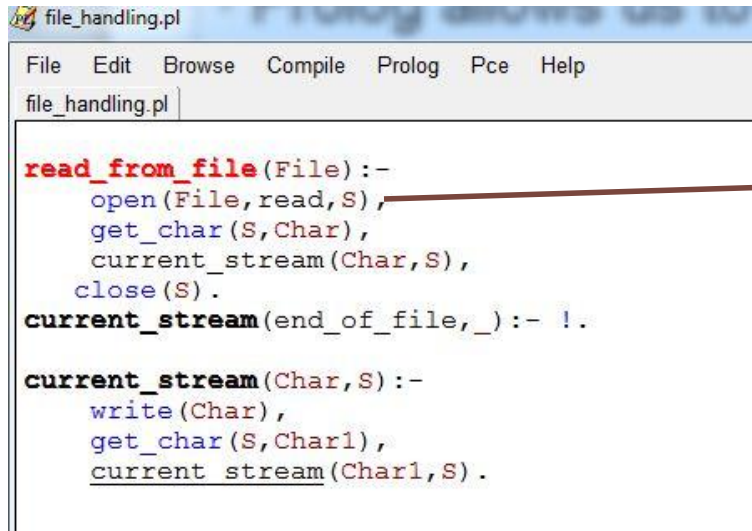
```
file_handling.pl
File Edit Browse Compile Prolog Pce Help
file_handling.pl

read_from_file(File):-
    open(File,read,S),
    get_char(S,Char),
    current_stream(Char,S),
    close(S).
current_stream(end_of_file,_):- !.

current_stream(Char,S):-
    write(Char),
    get_char(S,Char1),
    current_stream(Char1,S).
```

File handling

Reading from a file



```
file_handling.pl
File Edit Browse Compile Prolog Pce Help
file_handling.pl

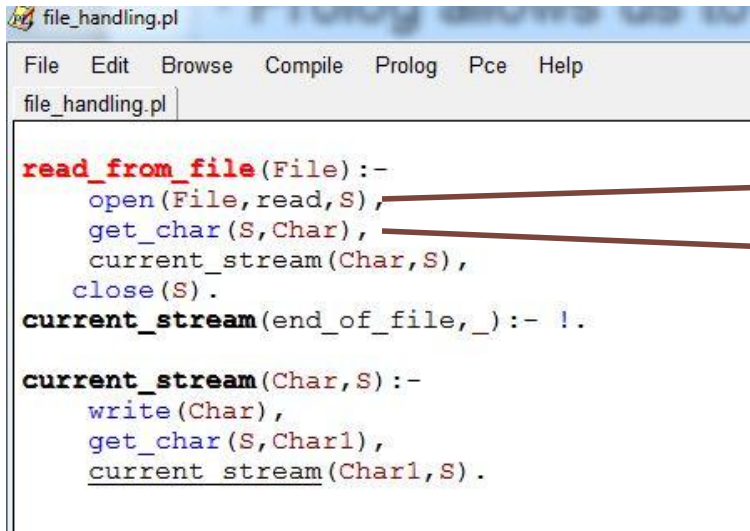
read_from_file(File):-
    open(File,read,S),
    get_char(S,Char),
    current_stream(Char,S),
    close(S).
current_stream(end_of_file,_):- !.

current_stream(Char,S):-
    write(Char),
    get_char(S,Char1),
    current_stream(Char1,S).
```

Open the file, File in read mode

File handling

Reading from a file



```
file_handling.pl
File Edit Browse Compile Prolog Pce Help
file_handling.pl

read_from_file(File):-
    open(File,read,S),
    get_char(S,Char),
    current_stream(Char,S),
    close(S).
current_stream(end_of_file,_):- !.

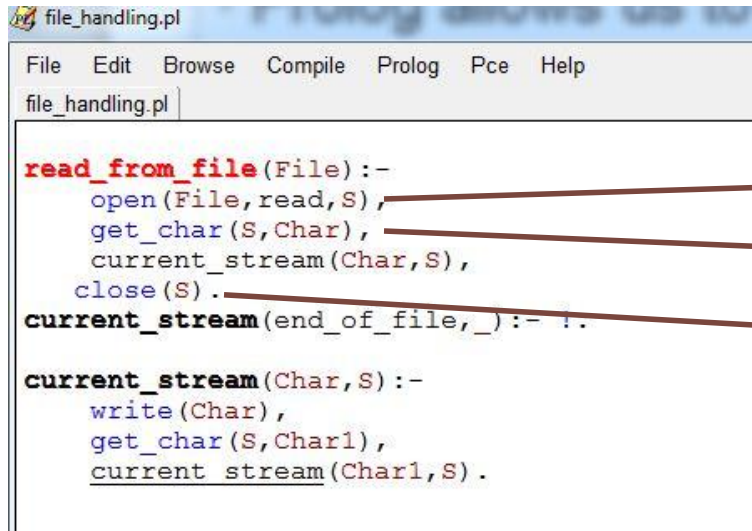
current_stream(Char,S):-
    write(Char),
    get_char(S,Char1),
    current_stream(Char1,S).
```

→ *Open the file, File in read mode*

→ *Reading character from the stream, S*

File handling

Reading from a file



```
file_handling.pl
File Edit Browse Compile Prolog Pce Help
file_handling.pl

read_from_file(File):-
    open(File,read,S),
    get_char(S,Char),
    current_stream(Char,S),
    close(S).
current_stream(end_of_file,_):-!.
current_stream(Char,S):-
    write(Char),
    get_char(S,Char1),
    current_stream(Char1,S).
```

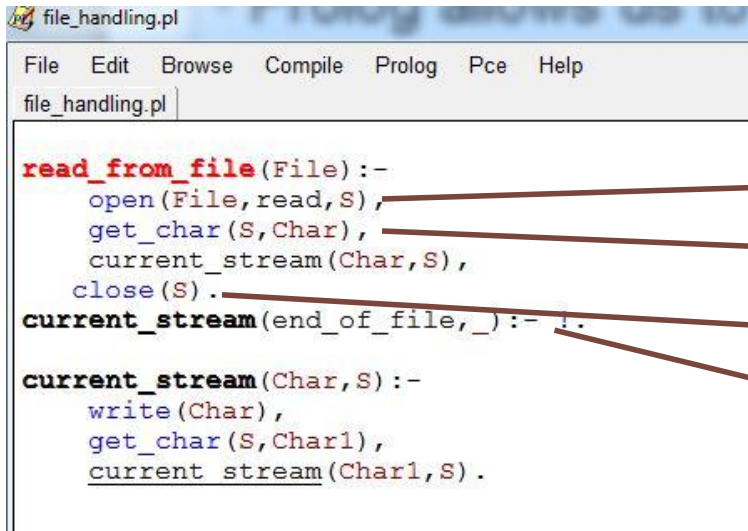
→ *Open the file, File in read mode*

→ *Reading character from the stream, S*

→ *Closing the file*

File handling

Reading from a file



```
file_handling.pl
File Edit Browse Compile Prolog Pce Help
file_handling.pl

read_from_file(File):-
    open(File,read,S),
    get_char(S,Char),
    current_stream(Char,S),
    close(S).
current_stream(end_of_file,_):-!.
current_stream(Char,S):-
    write(Char),
    get_char(S,Char1),
    current_stream(Char1,S).
```

→ *Open the file, File in read mode*

→ *Reading character from the stream, S*

→ *Closing the file*

→ *Cut(Stop when end of the file)*

File handling

Reading from a file

```
file_handling.pl
File Edit Browse Compile Prolog Pce Help
file_handling.pl

read_from_file(File):-
    open(File,read,S),
    get_char(S,Char),
    current_stream(Char,S),
    close(S).
current_stream(end_of_file,_):-!.
current_stream(Char,S):-
    write(Char),
    get_char(S,Char1),
    current_stream(Char1,S).
```

→ *Open the file, File in read mode*

→ *Reading character from the stream, S*

→ *Closing the file*

→ *Cut(Stop when end of the file)*

```
SWI-Prolog (AMD64, Multi-threaded, version 7.6.4)
File Edit Settings Run Debug Help
?- write_to_file('test.txt','This is a prolog tutorial').

true.

?- read_from_file('test.txt').
This is a prolog tutorial
true.

?- █
```

Built-in prolog predicates

- Many useful built-in predicates.
- Check the following link:

http://www.gprolog.org/manual/html_node/gprolog024.html

THANK

YOU!!