Introduction to Prolog

Basanta Joshi, PhD
joshibasanta@gmail.com

Prolog

PROLOG: PROgramming in LOGic

Developed in 1970 in France

Declarative Language unlike LISP, PASCAL, C, . . . which are procedural languages

No assignment statements

No goto's

No if-then-else structures

No loops

The user describes the problem in the form of facts and rules and PROLOG applies logical reasoning to find the answer.

Prolog Features

Prolog is a combination of the following key ideas:

- If- then rules with variables
- Relational databases (w/ terms for data structuring)
- Backward Chaining to try to prove goals
- Unification to match goals to rule conclusions.
- Backtracking to try all possibilities

Prolog Compilers

There are several free PROLOG Compilers available

SWI Prolog is a non-commercial product. This is on the PCs in Memorial.

***Amzi*! Prolog** is available on a 90 day free trial offer.

Visual Prolog has a student version

Strawberry Prolog is a nice light version

Prolog Definitions

PROLOG works with facts, relations and rules.

A *fact* is a unit of information which is assumed to be true.

A *relation* combines two or more facts.

A *rule* is a conditional assertion of a fact.

Example

A description of Ann's and Sue's world

Word Description.	*PROLOG Version*
A doll is a toy.	*toy(doll).*
Snoopy is a toy.	*toy(snoopy).*
Ann plays with Snoopy.	*plays(ann,snoopy).*
Sue likes everything Ann likes.	*likes(sue,Y) :- likes(ann,Y);*
Ann likes the toys she plays with	*likes(ann,X) :- toy(X),plays(ann,X).*

Every PROLOG line is a clause

A constant is lower case

A variable is upper case

"Running Prolog" – Queries

Given the description of Ann's and Sue's world, the question might be asked: "What does Sue like?"

PROLOG: |?-likes(sue,X).

X = snoopy

PROCESS: Prolog tries to find a value for all variables in a query so as to make the query true.

Sue likes X if Ann likes X

Ann likes X if X is a toy and Ann plays with X

Snoopy is a toy and Ann plays with Snoopy

So, Sue likes Snoopy

Prolog Execution

To enter a PROLOG program:

|?-consult(user).

<rules/facts/relations>

end-of-file.

or CRTL - D

Alternative: Save the program in a file and use |?-consult(<filename>).

Query Structure

A query returns a single answer, if you want to find alternative answers, respond to a PROLOG result with a "*;*" and PROLOG will seek another way to make the query true.

It will report the new answer or "*no*" if there are not other solutions.

Query Forms

```
|?- likes(sue,snoopy).
Yes
|?-likes(sue,ann).
no
```

```
|?- toy(X),likes(sue,X)
```

Prolog Programming

Prolog Programming

GOAL: Build a travel database and write PROLOG query routines

Begin by defining the structure of any relationships

travel(carrier,origin,destination,type)

```
travel(amtrak, new-york, boston,train).
travel(nj-transit, new-york, boston,train).
travel(amtrak, boston, portland,train).
travel(greyhound, boston, portland,bus).
travel(amtrak, new-york, washington,train).
travel(peoples, burlington, new-york,plane).
```

Prolog Rules

A Prolog program consists of rules and facts

GOAL: Construct a rule that will identify competitors.

What is the definition of a competitor in this small travel world?

Two carriers are competitors if they travel between the same two cities.

variables

and

if

competitor(Carrier1,Carrier2) :-

travel(Carrier1, CityA, CityB, _),

travel(Carrier2, CityA, CityB, _).

anonymous variable

Expanded Rules

GOAL: Develop a rule that will determine if it is possible to travel between two cities

We can travel between A and B if there is a carrier which starts at A and ends at B.

```
can-travel(CityA, CityB) :-

    travel(_, CityA, CityB, _).
```

Don't care about the carrier name

or type

Problem

Given this travel base and the can-travel rule what is the response to the query:

can-travel(new-york,portland).

Is it correct? If not, what is the problem?

How should a more general can-travel rule be expressed in English?

We can travel from A to B if we can travel from A to C and then from C to B for any number of intermediate cities, C.

travel(amtrak, new-york, boston,train).
travel(nj-transit, new-york, boston,train).
travel(amtrak, boston, portland,train).
travel(greyhound, boston, portland,bus).
travel(amtrak, new-york, washington,train).
travel(peoples, burlington, new-york,plane).

Recursive Rules

A more general can-travel rule involves two versions of the can-travel relation:

PROLOG will try this first

```
can-travel(CityA, CityB) :-
    travel(_, CityA, CityB, _).
```

Then this rule if the first fails

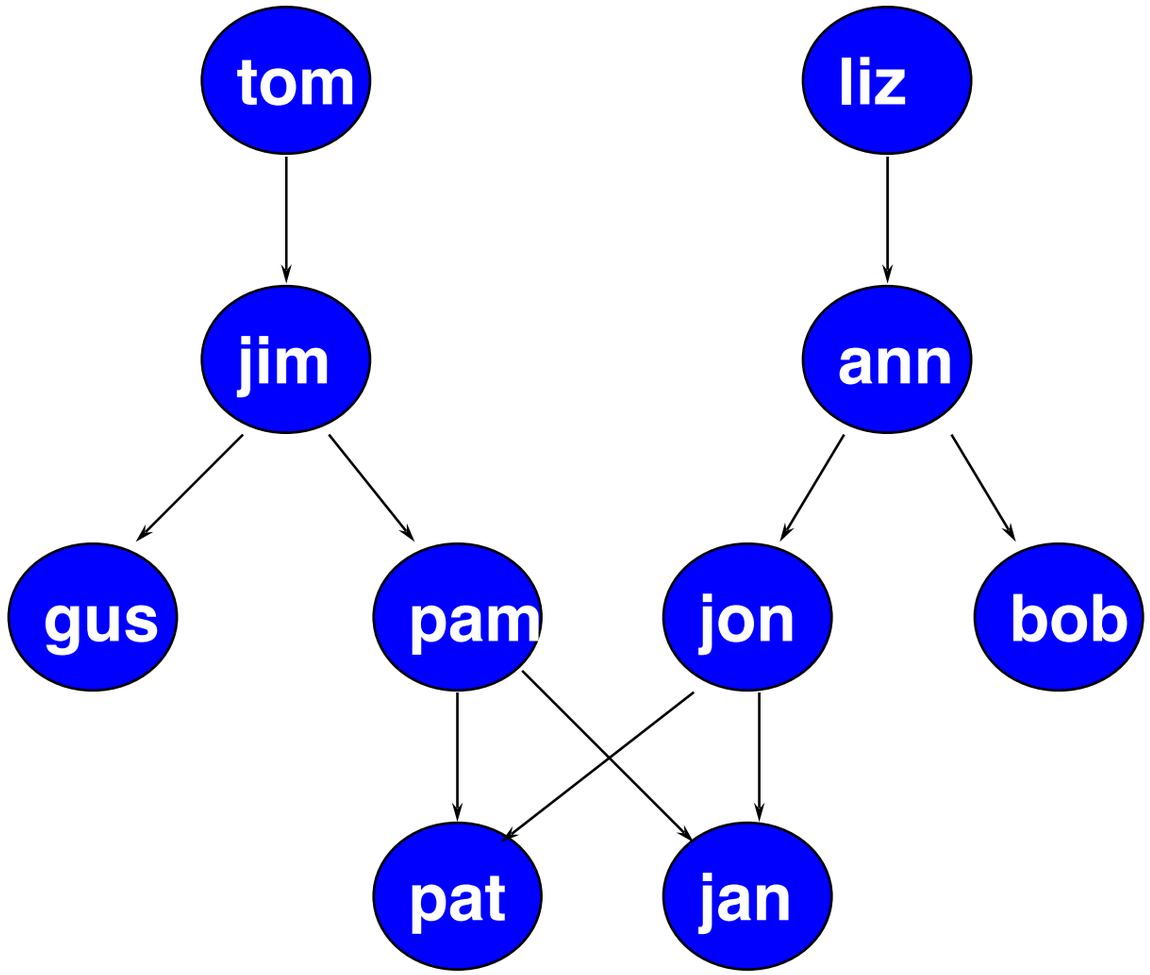
```
can-travel(CityA, CityB) :-
    travel(_, CityA, CityC, _),

can-travel(CityC, CityB).
```

Example 2 – Family Tree

Given a family tree – represent it in Prolog:

```
parent(tom, jim).
parent(jim, gus).
parent(jim, pam).
parent(pam, pat).
parent(pam, jan).
parent(liz, ann).
parent(ann, jon).
parent(ann, bob).
parent(jon, pat).
parent(jon, jan).
```



Queries

We can ask several questions about this database:

Is pam a parent of pat?

```
?- parent(pam, pat).
yes
```

Is pam a parent of anyone?

```
?- parent(pam, X).
X = pat    ;
X =  jan   ;
no
```

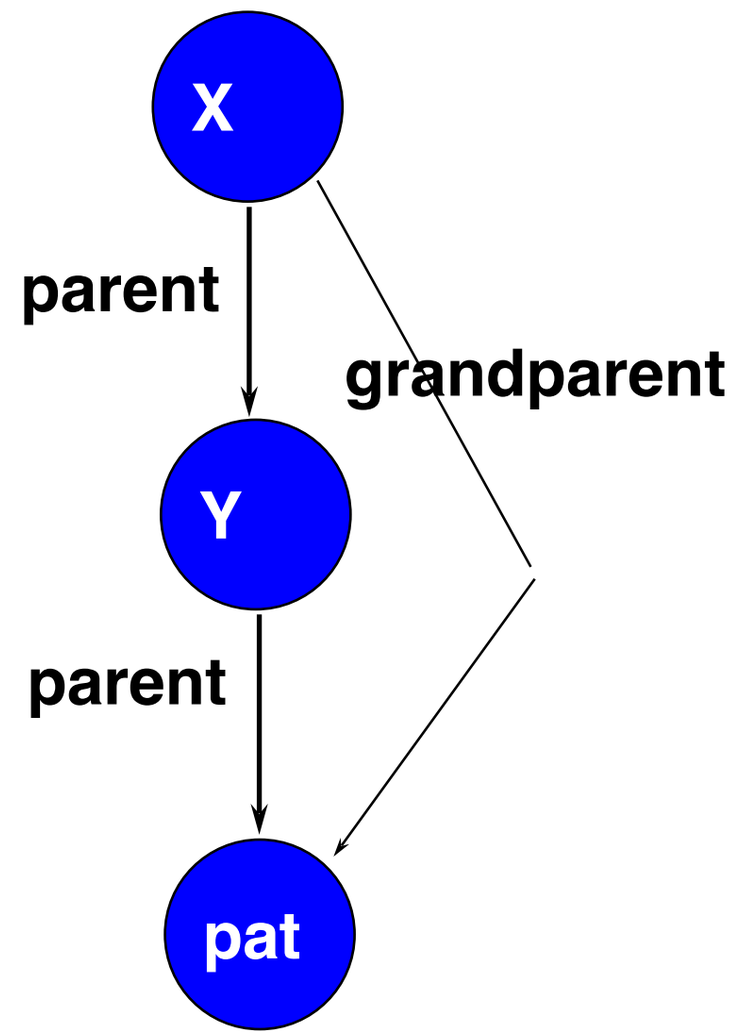
Do gus and jan share the same parent?

```
?- parent(X, gus), parent(X, jan)
X = jim
```

Grandparent Rule

To determine a grandparent:

?- parent(X,Y),parent(Y,pat).
X = jim
Y = pam



Defining a Predecessor

For parents

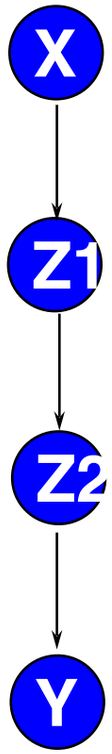
```
predecessor(X, Y) :- parent(X, Y).
```

For grandparents

```
predecessor(X, Y) :-
    parent(X, Z1), parent(Z1, Y).
```

For great grandparents

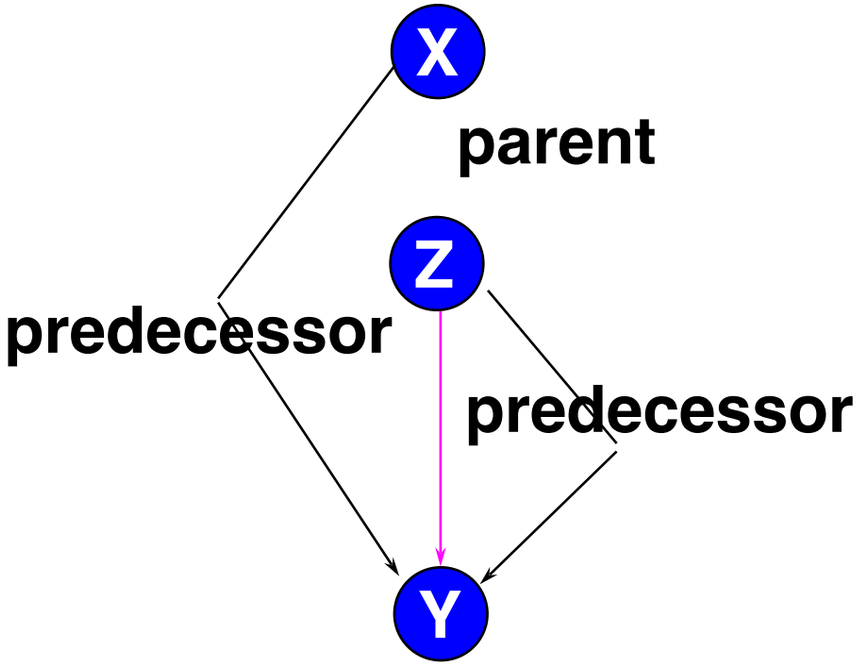
```
predecessor(X, Y) :- parent(X, Z1),
                     parent(Z1, Z2),
                     parent(Z2, Y).
```



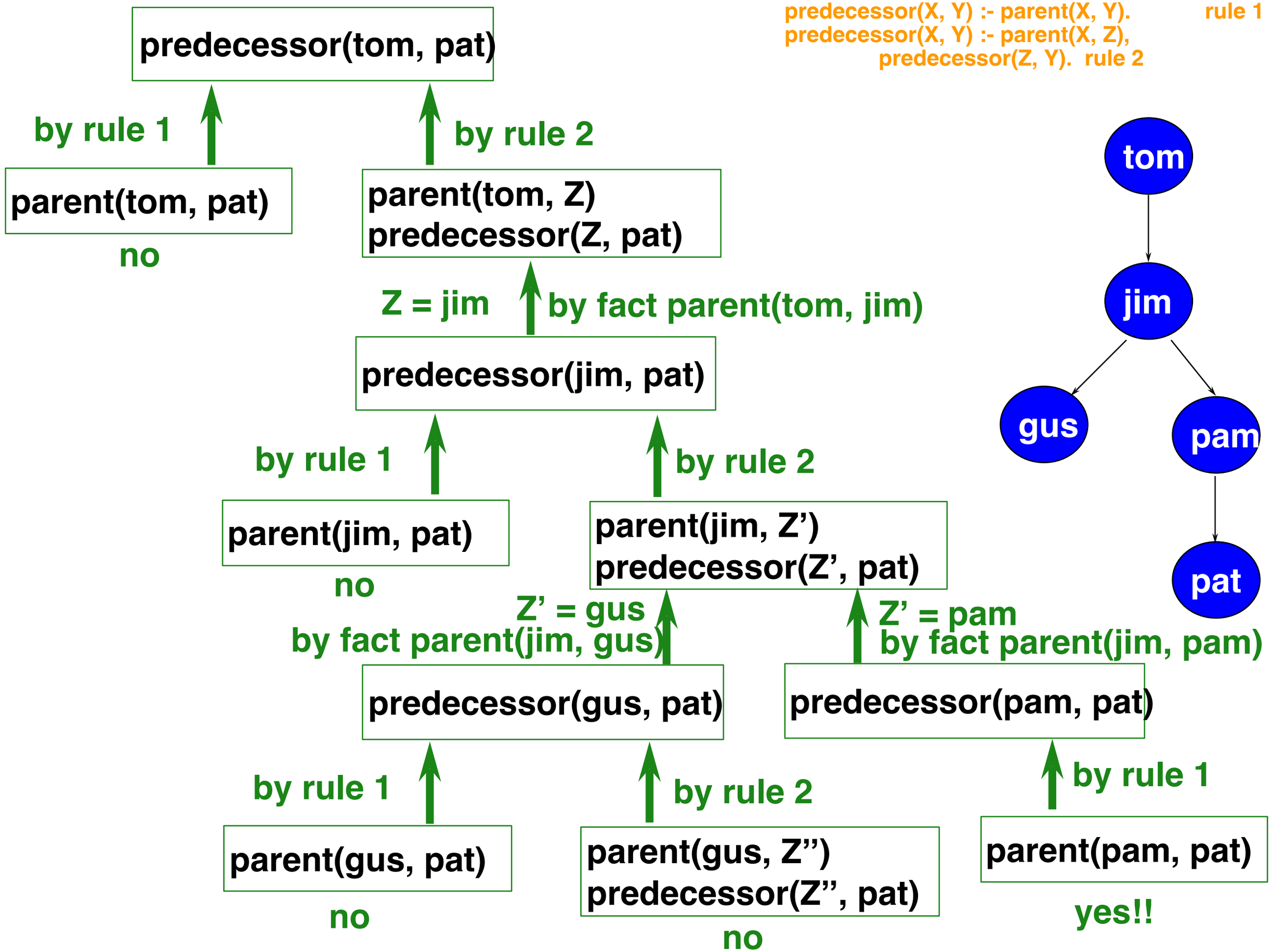
Recursive Predecessor Rule

```
predecessor(X, Y) :- parent(X, Y).

predecessor(X, Y) :- parent(X, Z),
                     predecessor(Z, Y).
```



predecessor(X, Y) :- parent(X, Y). rule 1
predecessor(X, Y) :- parent(X, Z),
predecessor(Z, Y). rule 2



Prolog Order

Prolog searches for clauses (rules & facts) from top to bottom

Prolog attempts goals from left to right

Caution

Good advice

- Put simplest clauses first
- Put simplest goals on the left

Although logically correct, Prolog can't handle the following

```
predecessor(X, Y) :- predecessor(Z, Y), parent(X, Z).
predecessor(X, Y) :- parent(X, Y).
```

Prolog I/O

Prolog I/O

There are two I/O commands in PROLOG

write(X). - writes a term X to the current output

read(X). - reads a term X from the current input.

EXAMPLE: given color(blue) color(red) then

```
|?- color(X), write(X), nl.
   blue   ;
   red
```

```
|?- read(X), write(X).
   test.
   test
```

Prolog Prompt

It is useful to provide a prompt for a read operation:

Define a rule input(X)
Write the prompt
Read the value

```
input(X) :-
    write('>'),
    read(X).
```

Running Prolog

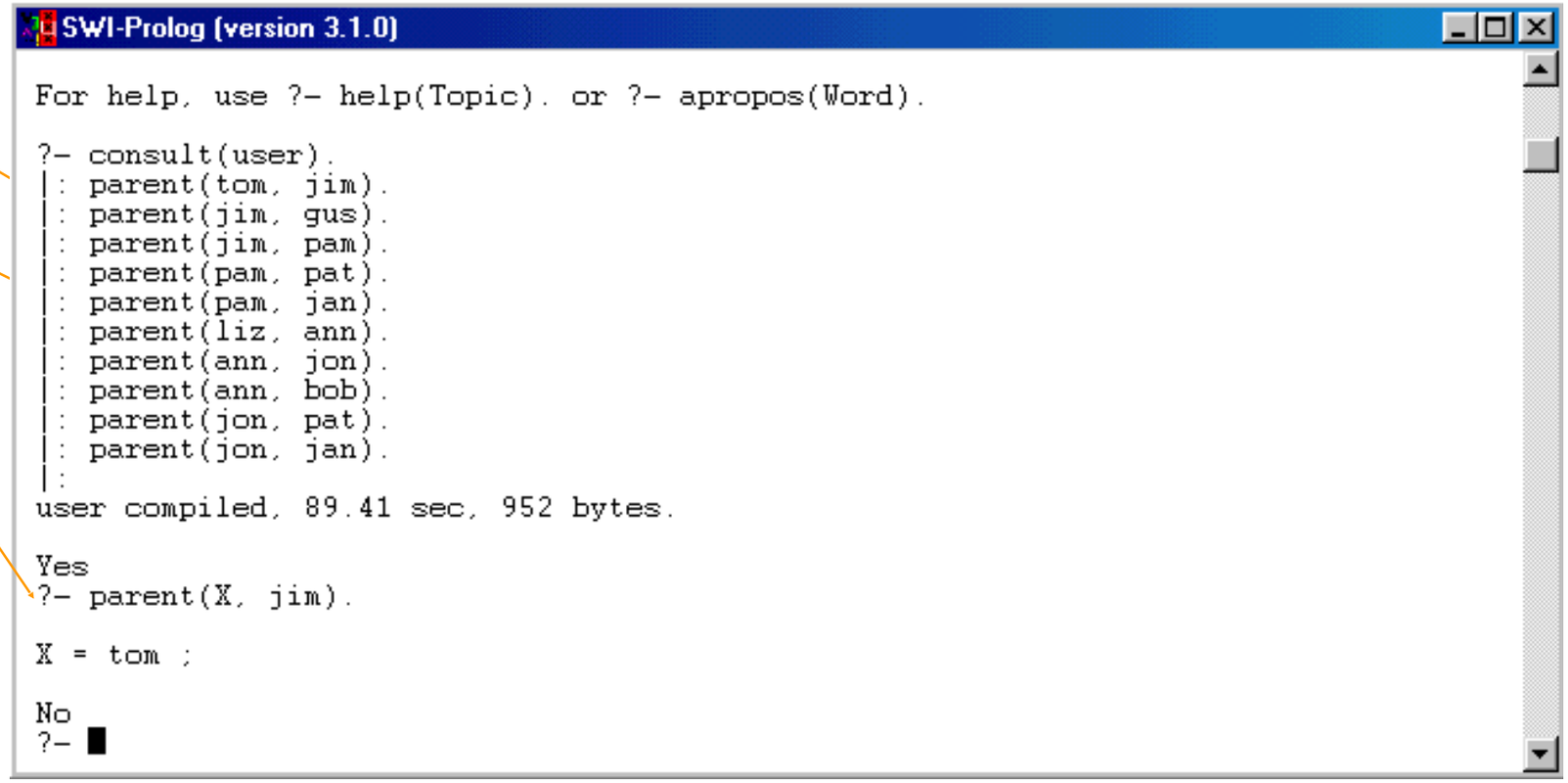
SWI Prolog 1

Run SWI and a Prolog window opens

Start a session

Enter the code

Enter a query



SWI Prolog 2

To save a program and run it at a later time

- Create the program as a set of rules and facts in an ASCII file
- Save it with a .pl extension
- Start SWI Prolog and enter

  consult(<filename>).
- The file will be read into Prolog and you are now ready to enter queries

Example Program

Create a text file with the following set of rules and facts:

```
parent(tom, jim).
parent(jim, gus).
parent(jim, pam).
parent(pam, pat).
parent(pam, jan).
parent(liz, ann).
parent(ann, jon).
parent(ann, bob).
parent(jon, pat).
parent(jon, jan).
```

New Facts

```
female(pam).
female(pat).
female(liz).
female(ann).
female(jan).
male(tom).
male(jim).
male(gus).
male(jon).
male(bob).
```

Mother Rule:
X is the mother of Y if
X is a parent of Y and
X is a female.

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

Example Run

Start SWI and consult your program file:

Make a query

```
SWI-Prolog (version 4.0.9)
Copyright (c) 1990-2000 University of Amsterdam.
Copy policy: GPL-2 (see www.gnu.org)

For help, use ?- help(Topic). or ?- apropos(Word).

?- consult(\family).
% \family compiled 0.00 sec, 2,412 bytes

Yes
?- mother(X, pat).

X = pam

Yes
?-
```

Prolog Examples

Difficult Rule

Using the family tree data base consider the rule:

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

Unfortunately Prolog succeeds on

```
?- sister(ann, ann).
```

Solution

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

where X\= Y is only true if X is not equal to Y

Prolog Arithmetic

An assignment like statement is in the form:

`X is E .`

The arithmetic expression **`E`** is evaluated and the variable **`X`** is instantiated to the result.

```
?- N is 3 * 4.
N = 12
yes
```

```
?- X is 3 + 4/5.
X = 3.8
```

Example 1

PROBLEM: Find the factorial of a nonnegative integer, n

= is a test, not an assignment

*ENGLISH: F is the factorial of N if either N=0 and F = 1 or N>0 and F=N*F1 where F1 is the factorial of N-1.*

```
factorial(N,F) :- N=0, F=1.

factorial(N,F) :- N>0,
                  N1 is N - 1,
                  factorial(N1,F1),
                  F is N*F1.
```

Example 2

Define a rule to find the maximum of two numbers X and Y

Relation: *max(X,Y,Max)*

English: X is the max if X > Y

Max(X,Y,X) :- X > Y.

English: Y is the max if Y >= X

Max(X,Y,Y) :- Y >= X.

Example 3

Problem: Find the square of a number

```
dosquares :-write('Next item
please: '),
read(X),
 process(X).
process(stop) :- !.
 process(N) :-C is N * N,
 write('Square of
 '),write(N),write(' is '),
 write(C), nl,
 dosquares.
```

```
?- dosquares.
Next item please:
5.
Square of 5 is
25.
Next item please:
12
Square of 12 is
144
Next item please:
stop
yes
```