CPSC 312

Functional and Logic Programming

15 September 2015

Piazza

- Please join Piazza if you haven't already

piazza.com/ubc.ca/winterterm12015/cpsc312

- That's where you will find these lectures, assignments, ask your questions, answer others' questions, etc.

- There is no course website.

- The Syllabus is on Piazza, please download and read to know what is expected of you. Let me know if you have any questions.

Grading Scheme

5 Assignments	15%
2 Projects	30%
Midterm exam	20%
Final exam	35%

+ (Bonus) Participation 5-10%

In order to pass the course you must obtain 50% ovearall, plus

- pass the final exam
- as well as each of the two projects

Office Hours

- Instructor: (Sara Sagaii)
  - Tuesdays 10-11am
  - Thursdays 9:30-10:30am
  - ICCS 187
- Rui Ge:
  - Wednesdays 10-12am
  - Table 1 at DLC
- Susanne Bradley:
  - Fridays 11:30-1:30
  - Table 1 at DLC

Waitlist woes

- Class is now full; 9 left on the waiting list.
- Two important dates:
  - **Sept. 22nd (Tue.)** – last day for dropping first-term courses without a "W"
  - **Oct. 16th (Fri.)** – last day to withdraw from a first-term course.
- If you are on the waiting list:
  - your only option is to continue as a regular student until you've lost hope (I don't recommend retaining hope Sep 23rd onward).
  - I have absolutely no information about your chance of getting in or your place in the queue.

Questions

As we said..

- In Logic (declarative) Programming, we are concerned with what to do, not how to do it...*Prolog already knows how*.
- A prolog program is a set of rules and facts.

Separation of Program and Computation

"program = set of axioms

computation = constructive proof of a goal statement from the program" (p. 4)

"A **logic program** is a set of axioms, or rules, defining relations between objects. A **computation** of a logic program is a deduction of the consequences of the program." (p. 12)

Separation of knowledge and goals

- As you read in the first chapter...there are three types of statement in Prolog: **Facts**, **Rules**, **Queries**.

- Facts and Rules express knowledge of the problem domain.

- Queries ask for solutions based on that knowledge.

Chapter 1: The Basic Constructs

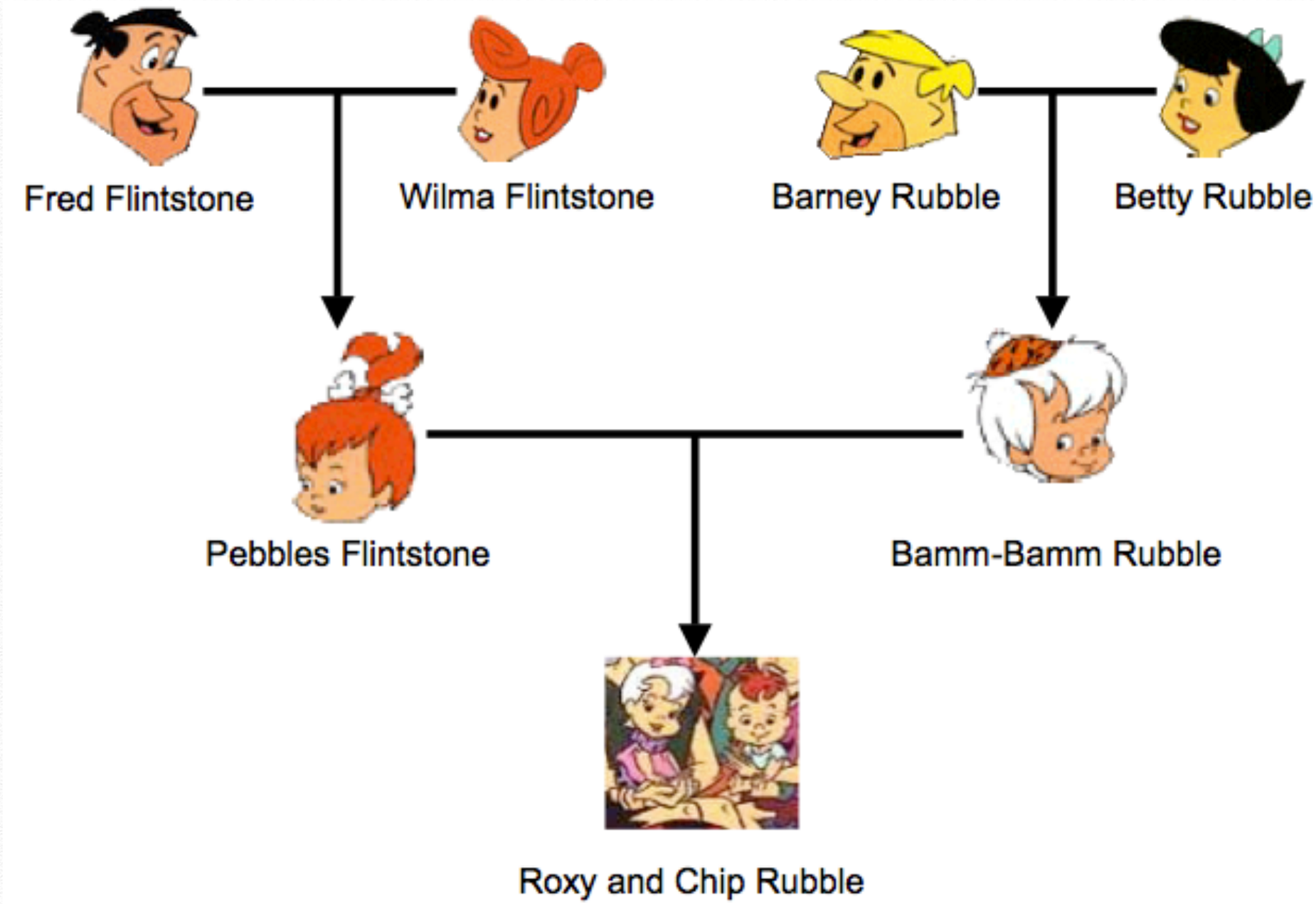
What did you think of the reading?

An informal intro

- Your textbook takes a mathematical logic approach to introducing logic programming and Prolog. That's not very useful for learning to program in Prolog.

- Rather than throw lots of theory at you, let's begin with a simple problem domain, cook up a little Prolog code, and see what happens.

The Flintstones family tree



Facts

- What kinds of facts could we state about the world of the Flintstones based on what we've seen?

Fred is Pebbles' father.
Fred is married to Wilma.
Wilma is Pebbles' mother.
Wilma is married to Fred.
Pebbles is Fred's daughter.
Pebbles is Wilma's daughter.
Fred's first name is Fred.
Fred is older than Pebbles.

- What else can you come up with?

There's a lot if you consider we're still working on just Fred, Wilma, and Pebbles!

Facts

- What does the first fact look like in Prolog?

  English: Fred is Pebbles' father.

Facts

- What does the first fact look like in Prolog?

English: Fred is Pebbles' father.

Prolog: `father(fred,pebbles).`

Facts

- What does the first fact look like in Prolog?

  English: Fred is Pebbles' father.

  Prolog: `father(fred,pebbles).`

- What does this Prolog fact mean to you?

Facts

- What does the first fact look like in Prolog?

  English: Fred is Pebbles' father.

  Prolog: `father(fred,pebbles).`

- What does this Prolog fact mean to you?
- Why doesn't it mean Pebbles is Fred's father?

Facts

- What does the first fact look like in Prolog?

  English: Fred is Pebbles' father.

  Prolog: `father(fred,pebbles).`

- What does this Prolog fact mean to you?
- Why doesn't it mean Pebbles is Fred's father?
- What does this fact mean to Prolog?

Facts

- What does the first fact look like in Prolog?

  English: Fred is Pebbles' father.

  Prolog: `father(fred,pebbles).`

- What does this Prolog fact mean to you?
- Why doesn't it mean Pebbles is Fred's father?
- What does this fact mean to Prolog?
- A *fact* in Prolog is a statement that declares a relationship between objects.

Facts

Here are more facts:

```
father(fred,pebbles).
father(barney,bamm-bamm).
father(bamm-bamm,roxy).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).
```

Facts

Here are more facts:

```
father(fred,pebbles).
father(barney,bamm-bamm).
father(bamm-bamm,roxy).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).
```

This also just happens to be a Prolog program!

Facts

Here are more facts:

```
father(fred,pebbles).
father(barney,bamm-bamm).
father(bamm-bamm,roxy).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).
```

This also just happens to be a Prolog program!

A finite set of facts constitutes a program.

Questions

IDE

- Any simple text editor or editors with built-in Prolog mode (Vim or Emacs), or more modern editors with Prolog plug-ins such as Notepad++ and Sublime Text.

- Plug-in for Sublime Text: https://packagecontrol.io/packages/Prolog

- Plug-in for Notepad++: http://docs.notepad-plus-plus.org/index.php?title=User_Defined_Language_Files (Look for SWI Prolog)

IDE

If you want to experiment:

- XPG: http://xgp.sourceforge.net/
- SWI-Prolog PceEmacs is a work in progress: http://www.swi-prolog.org/IDE.html
- SICStus Prolog IDE (SPIDER) based on Eclipse is in public beta: https://sicstus.sics.se/spider/
- ProDT http://prodevtools.sourceforge.net/
- and many more.

SWI-Prolog

- Download and Install: http://www.swi-prolog.org/
- Also available on lab computers:
  - typing either `swipl` (or `pl` in older versions) should open an swi-prolog console.
- On your machines, make sure swipl is added to your path. Change to your program directory, then type `swipl` (or `pl`).

Running SWI-Prolog

```
% swipl

Welcome to SWI-Prolog (Multi-threaded, Version
5.3.16) Copyright (c) 1990-2003 University of
Amsterdam. SWI-Prolog comes with ABSOLUTELY NO
WARRANTY. This is free software, and you are
welcome to redistribute it under certain
conditions. Please visit http://www.swi-
prolog.org for details.

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

?-
```

The Manual

- Quick Start guide from SWI-Prolog's Reference Manual http://www.swi-prolog.org/pldoc/man?section=quickstart

- Ch. 2 Overview: http://www.swi-prolog.org/pldoc/man?section=overview

- Ch. 4 Built-in Predicates: http://www.swi-prolog.org/pldoc/man?section=builtin

Basic Commands

to load a program:

```
?- consult('flintstones.pl').
```

or

```
?- [flintstones].
```

to reload a program after editing:

```
?- make.
```

to unload a program:

```
? - unload_file('flintstones.pl').
```

to end your session and exit:

```
?- halt.
```

Back to the Flintstones

Here are more facts:

```
father(fred,pebbles).
father(barney,bamm-bamm).
father(bamm-bamm,roxy).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).
```

This also just happens to be a Prolog program!

A finite set of facts constitutes a program.

Querying Flintstones

```
?- [flintstones].
true.
```

A finite set of facts constitutes a program.

```
father(fred,pebbles).
father(barney,bamm-bamm).
father(bamm-bamm,roxy).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).
```

Querying Flintstones

```
?- [flintstones].
true.

?- father(fred,pebbles).
```

A finite set of facts constitutes a program.

```
father(fred,pebbles).
father(barney,bamm-bamm).
father(bamm-bamm,roxy).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).
```

Querying Flintstones

```
?- [flintstones].
true.

?- father(fred,pebbles).
true.

?- mother(pebbles,chip).
```

A finite set of facts constitutes a program.

```
father(fred,pebbles).
father(barney,bamm-bamm).
father(bamm-bamm,roxy).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).
```

Querying Flintstones

```
?- [flintstones].
true.

?- father(fred,pebbles).
true.

?- mother(pebbles,chip).
true.

?- mother(fred,pebbles).
```

A finite set of facts constitutes a program.

```
father(fred,pebbles).
father(barney,bamm-bamm).
father(bamm-bamm,roxy).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).
```

Querying Flintstones

```
?- [flintstones].
true.

?- father(fred,pebbles).
true.

?- mother(pebbles,chip).
true.

?- mother(fred,pebbles).
false.
```

A finite set of facts constitutes a program.

```
father(fred,pebbles).
father(barney,bamm-bamm).
father(bamm-bamm,roxy).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).
```

Prolog Program Structure

- A Prolog program is a declarative, logical model of a problem; It's not a collection of instructions that describe how to solve the problem. the model consists of facts and rules.

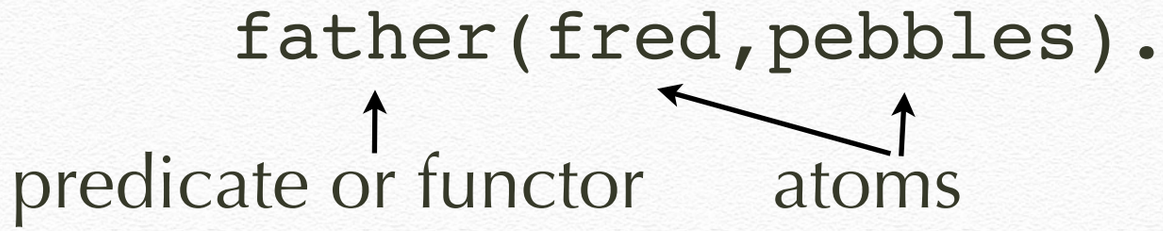
- We supply a Prolog program with a query: a logical statement that *may or may not be true*.

- Prolog will try to prove the query and will report the result. So think of the query as a theorem to be proven or a goal to be reached.

Questions

Syntax and Terminology

Fact Syntax



- **atoms** are the names of objects and start with lower case letters.
- **Predicates** or **functors** are the names of relations, and also begin with lower case letters.
- **arity** is the number of arguments a predicate takes; `father` has an arity of 2.
- predicates are characterized by their name and arity, such as `father/2`

Rule Syntax

if (←) and(^)

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

head body

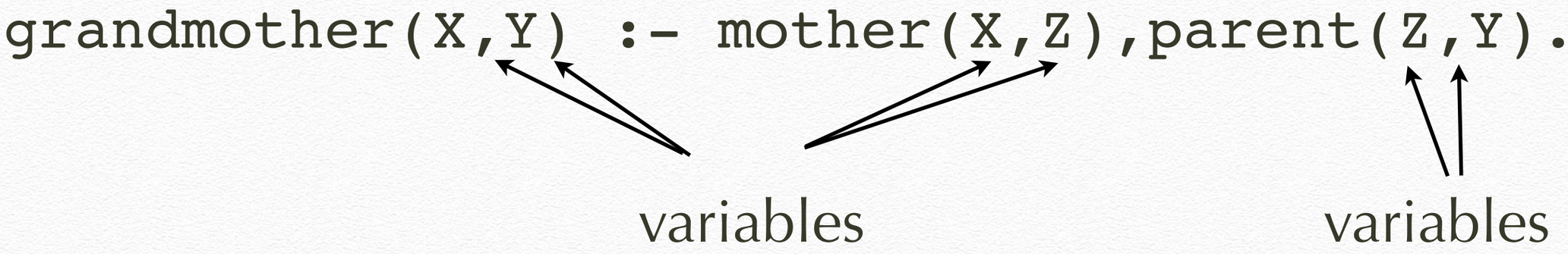
You can read this as

"X is a grandmother of Y only if X a mother of Z and Z is a parent of Y."

You could also read this as

"to prove that X is grandmother of Y, you must prove that X is the mother of Z and Z is the parent of Y."

Variables



Variables stand for *general unspecified objects*.

They're **not** names of memory locations where you store data.

Variables begin with upper case letters.

(e.g., X, Name, M324)

Query Syntax

```
?- father(fred,pebbles).
```

```
?- father(X,pebbles).
```

Same as fact syntax, except you provide a query at the ?- prompt (In your textbook, queries are often followed by a ? just so you'll know it's a query).

Even though syntactically they are similar, facts and queries have represent different semantics: "fred is the father of pebbles" vs "is fred the father of pebbles?"

*what is the meaning of a variable in a query?

Quiz

- What is the semantic difference between a variable in a query and a variable in a fact?

```
?- father(fred,X).
```

vs

```
father(fred,X).
```

Quiz

```
?- father(fred,X).
```

vs

```
father(fred,X).
```

The variable X in the query is "Existentially Quantified", but in the fact it is "Universally Quantified".

In the query, it means: is there a value for X such that this relation is true. In English: Fred must be the father of *someone* (in order for the query to be true).

In the fact, it means: for any value of X, such relation is true. In English: Fred is the father of *everyone*.

Research Question

If you actually include such a fact

```
father(fred,X).
```

in a Prolog program, you will get a warning upon loading the program intro the interpreter.

What is that warning, what does it mean, why is it a warning (i.e. what could the issue be) and what is a way to eliminate it?

More Terminology

- A **definite clause** is either a fact or a rule.
- A **program** is a finite set of definite clauses.
- A fact is a special case of a rule -- it's a rule with a head but no body.

A second example*

parent(tom, bob).

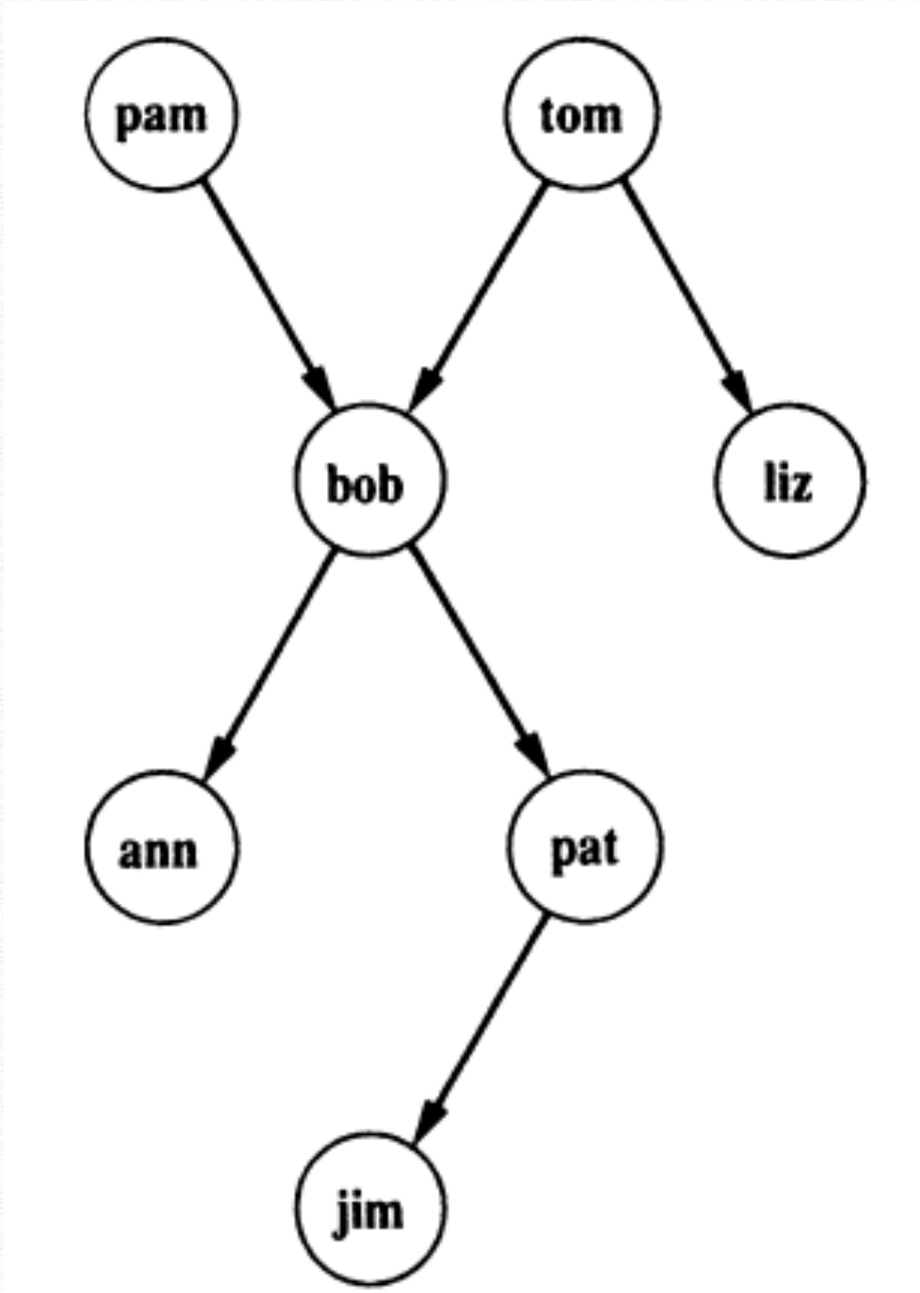
parent(pam, bob).

parent(tom, liz).

parent(bob, ann).

parent(bob, pat).

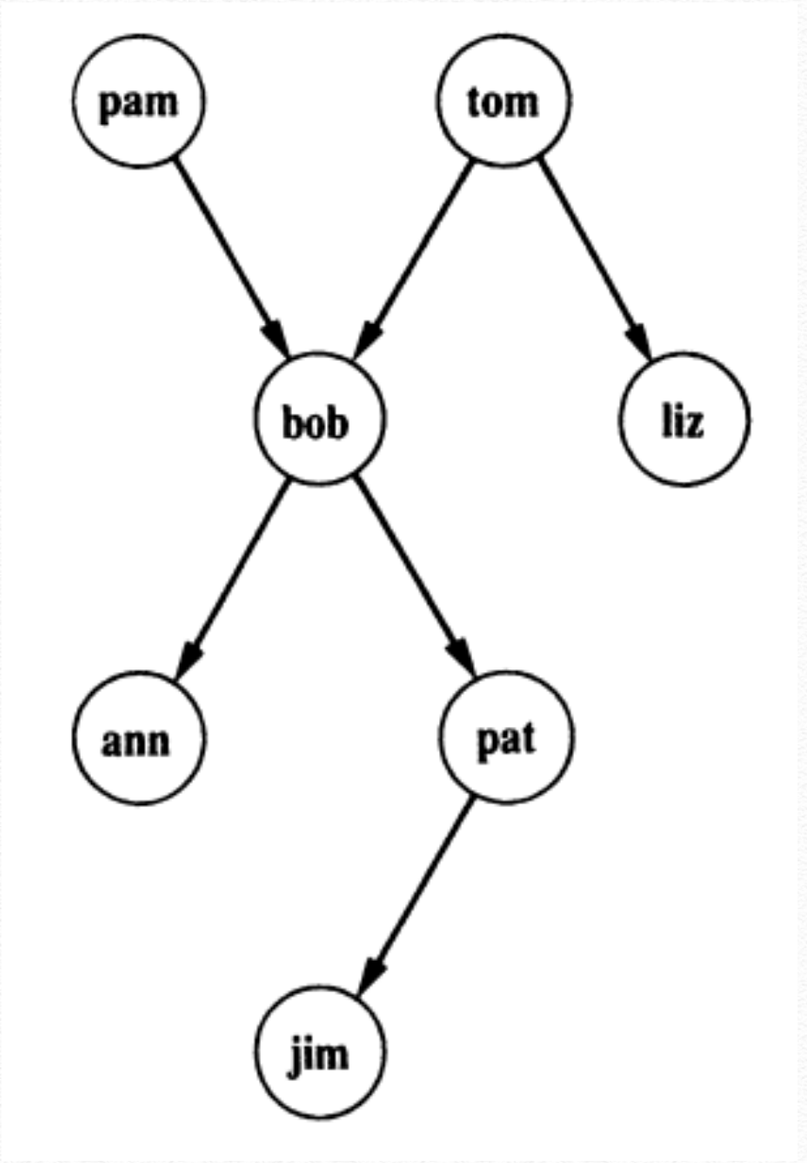
parent(pat, jim).



*from *PROLOG: Programming for Artificial Intelligence* by Ivan Bratko, Fourth Edition.

Let's Query

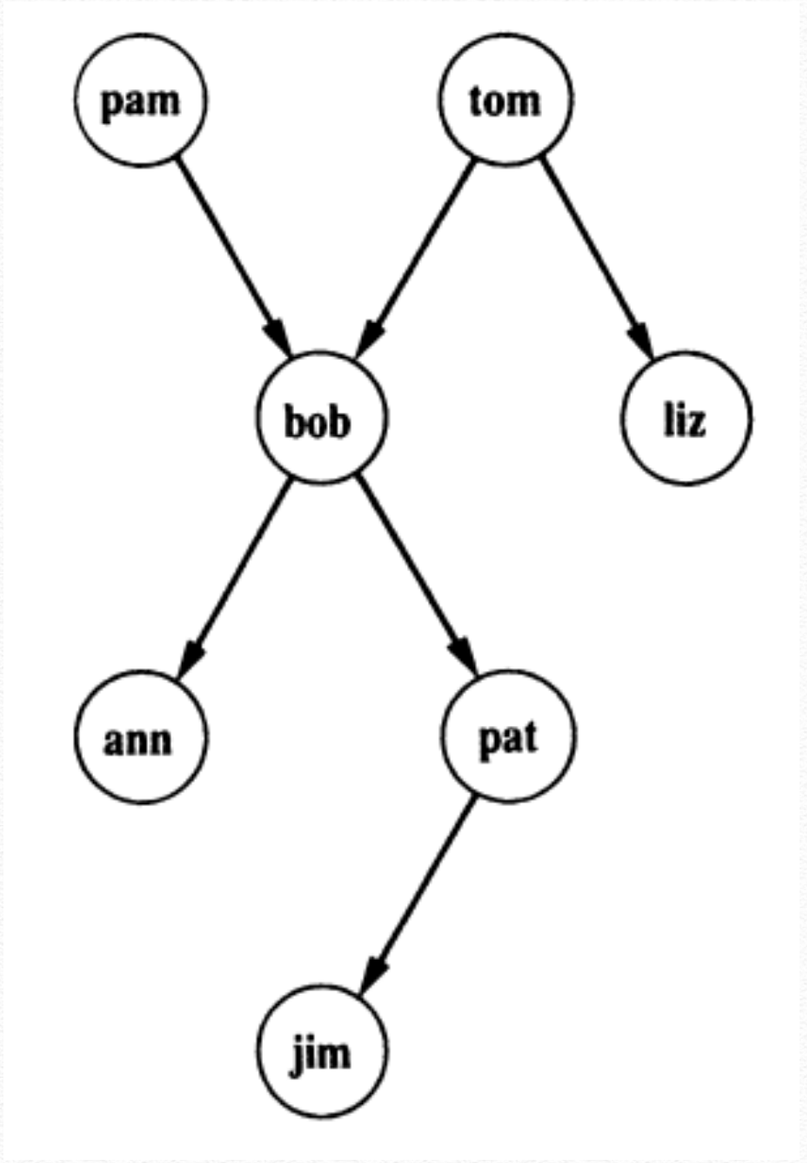
English: Is Bob a parent of Pat?



Let's Query

English: Is Bob a parent of Pat?

```
?- parent(bob, pat).
```

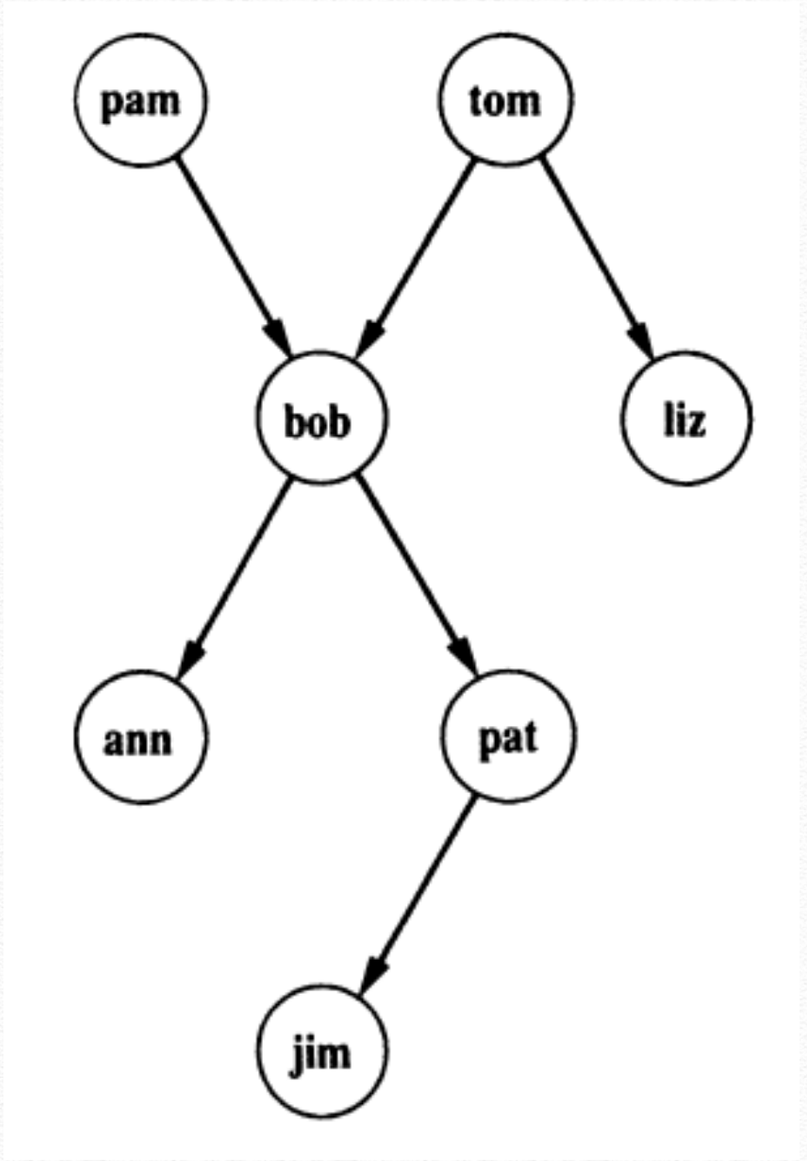


Let's Query

English: Is Bob a parent of Pat?

```
?- parent(bob, pat).
true.

?- parent(bob, liz).
```

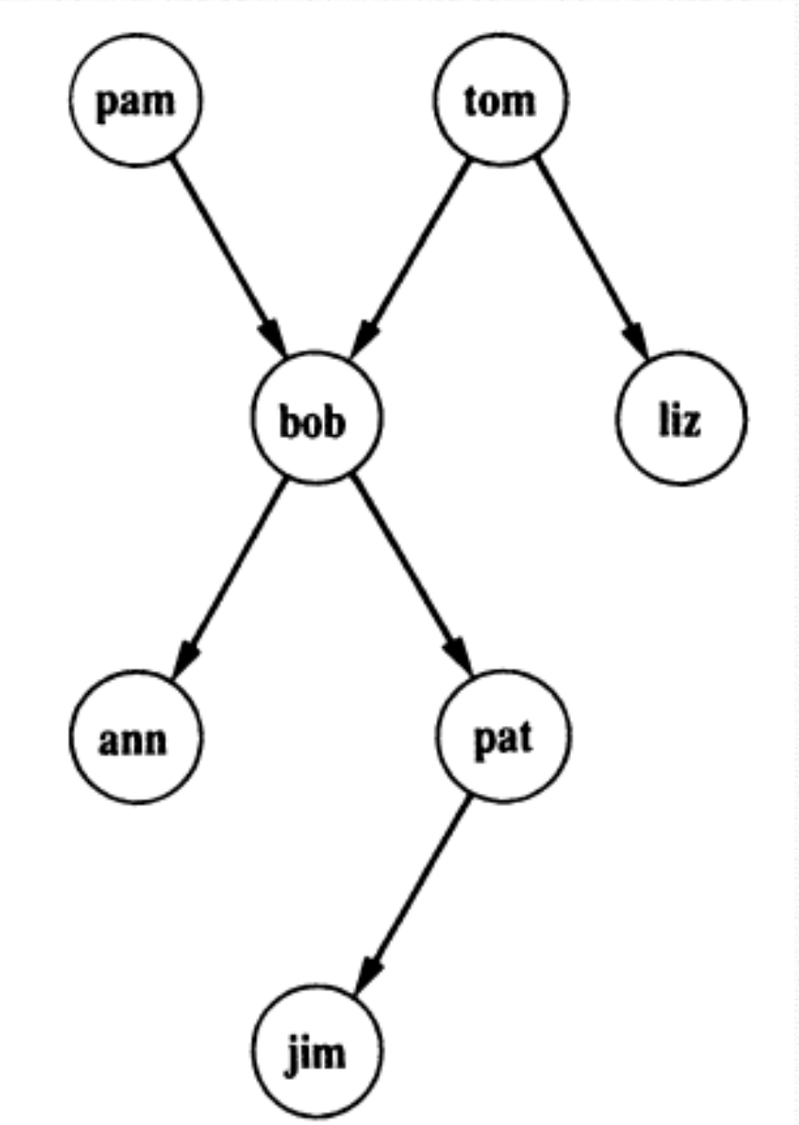


Let's Query

English: Is Bob a parent of Pat?

```
?- parent(bob, pat).
true.

?- parent(bob, liz).
false.
```



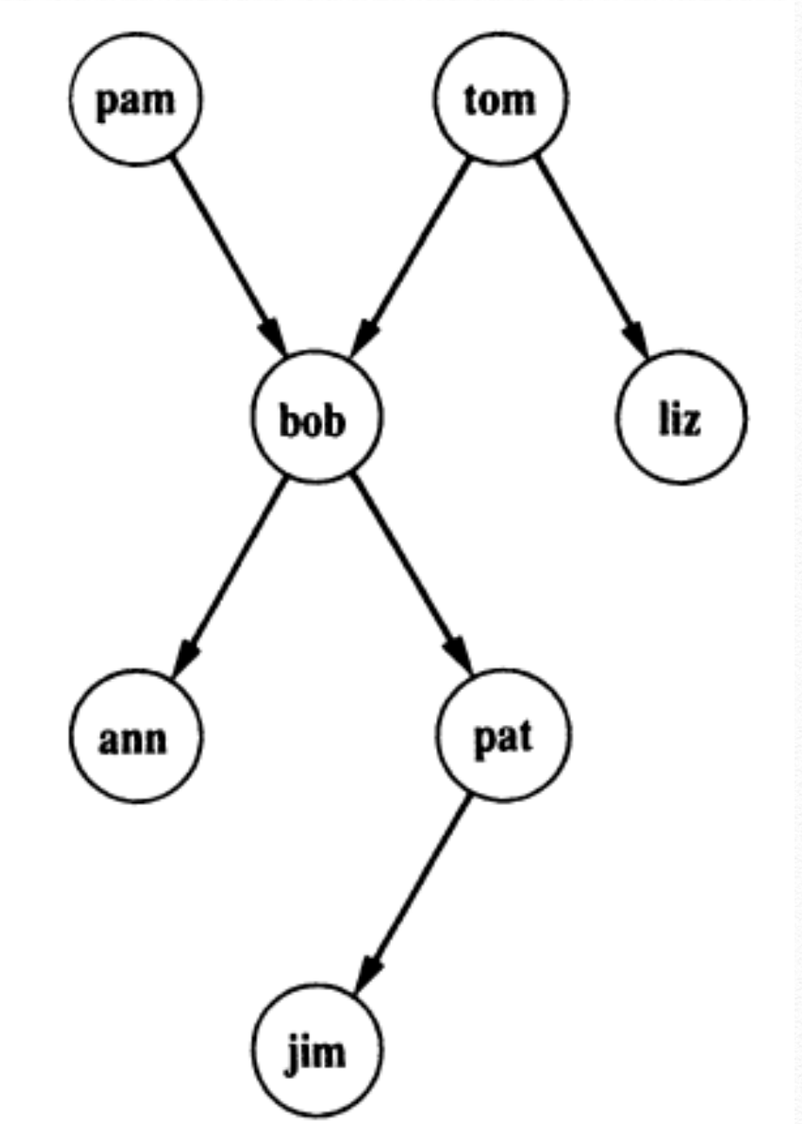
Let's Query

English: Is Bob a parent of Pat?

```
?- parent(bob, pat).
true.
```

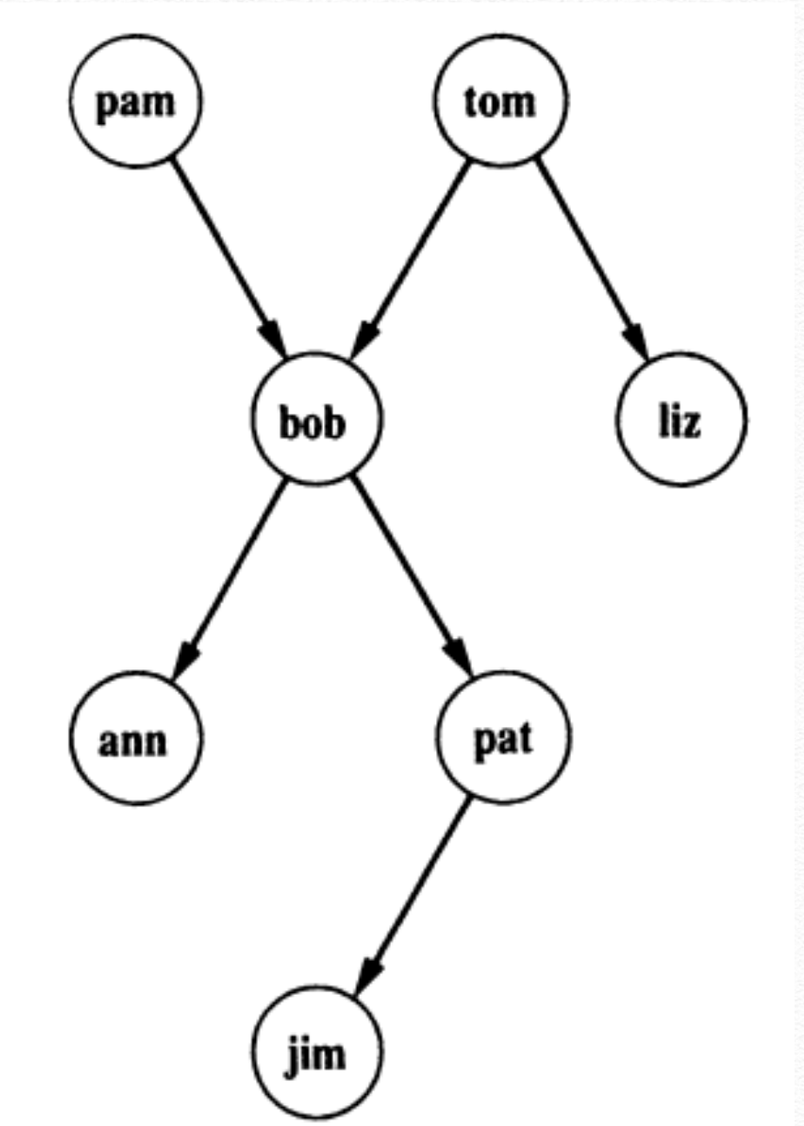
```
?- parent(bob, liz).
false.
```

Prolog operates under the Closed World Assumption. If it is not stated as a fact or can not be deduced from rules, it is not true.



Let's Query

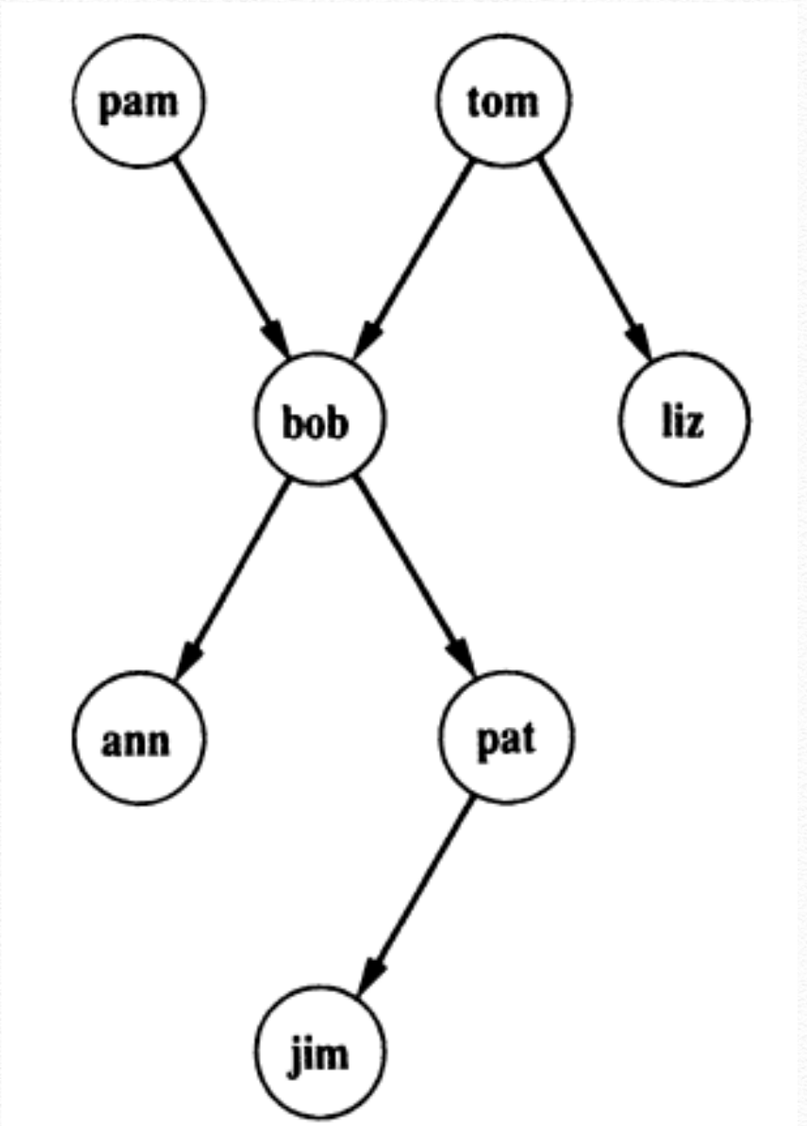
English: Who is the parent of liz?



Let's Query

English: Who is the parent of liz?

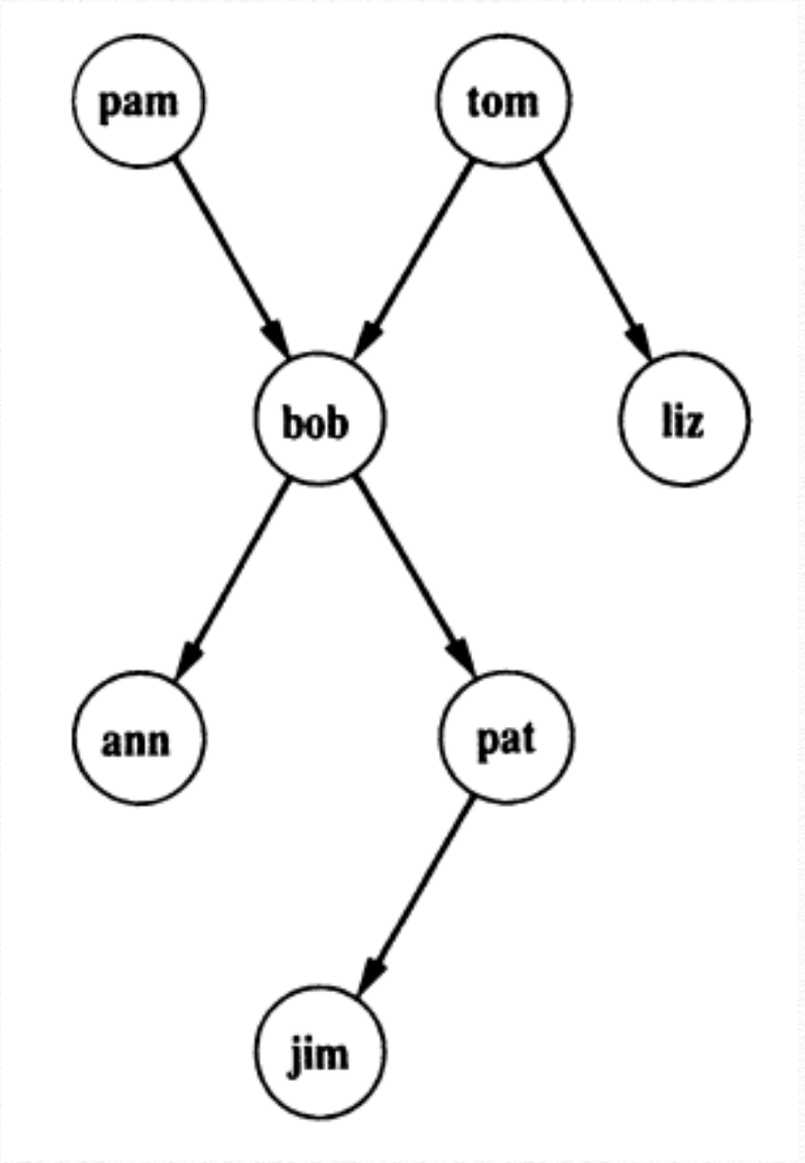
```
?- parent(X,liz).
```



Let's Query

English: Who is the parent of liz?

```
?- parent(X,liz).
X = tom.
```

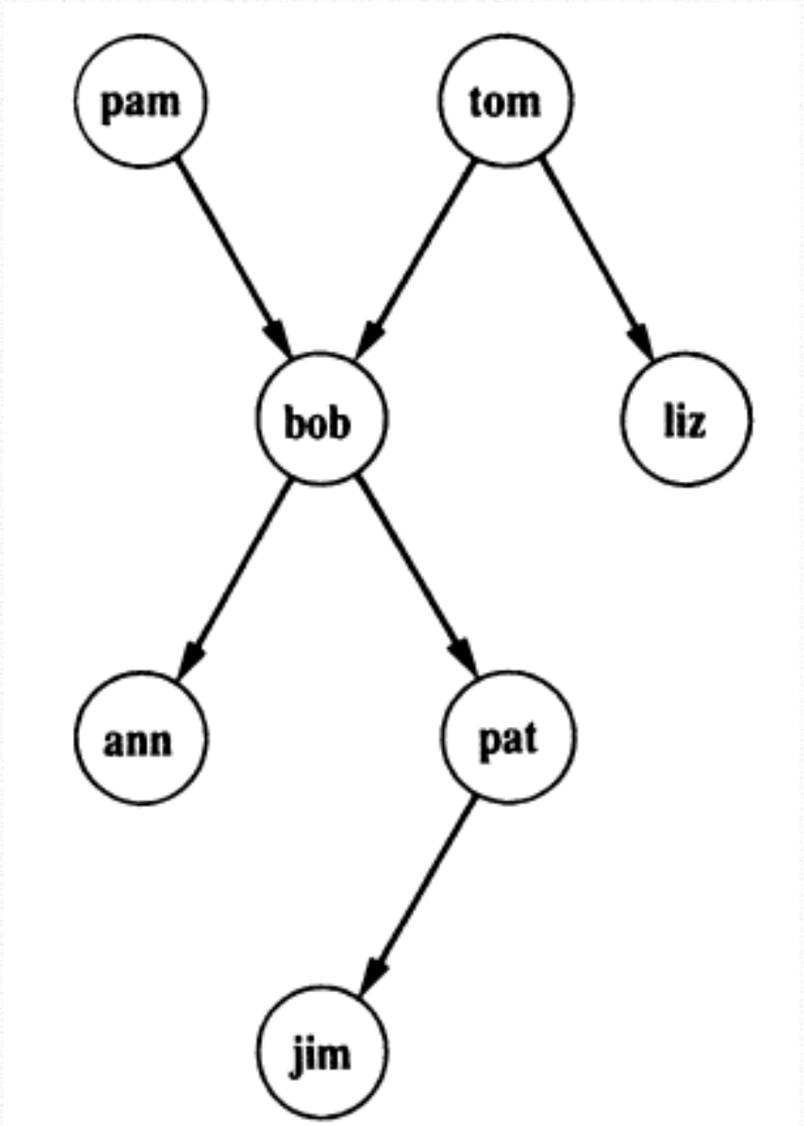


Let's Query

English: Who is the parent of liz?

```
?- parent(X,liz).
X = tom.
```

Prolog will tell us what the value of X should be, according to the program, in order for the above query to be true. If it doesn't find any way to *resolve* X and make the query true it will return false.



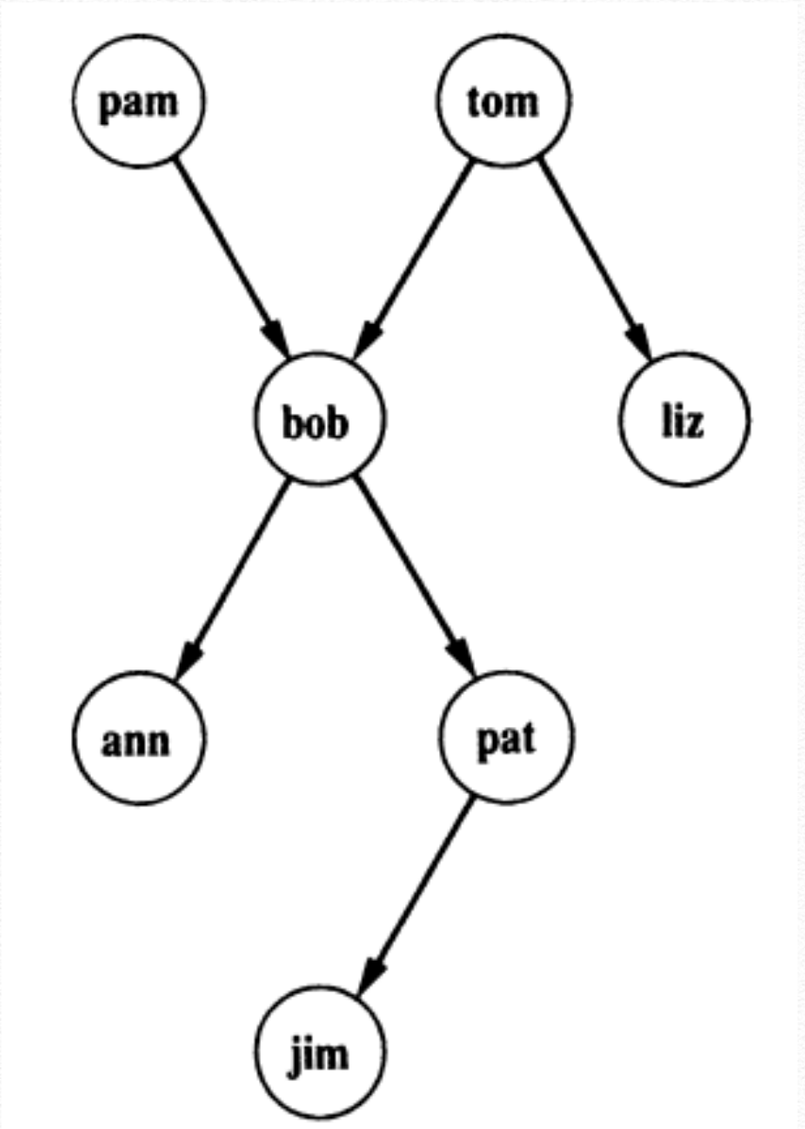
Let's Query

English: Who is the parent of liz?

```
?- parent(X,liz).
X = tom.
```

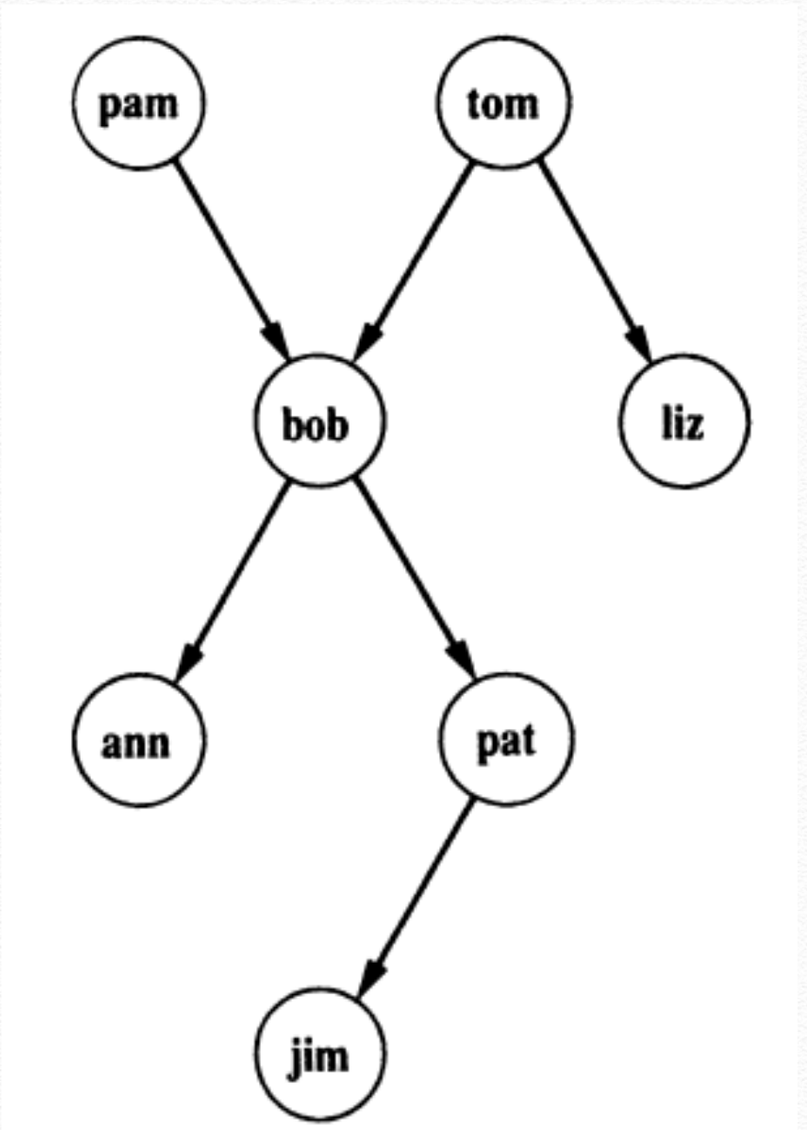
Prolog will tell us what the value of X should be, according to the program, in order for the above query to be true. If it doesn't find any way to *resolve* X and make the query true it will return false.

```
?- parent(X,pam).
false.
```



Let's Query

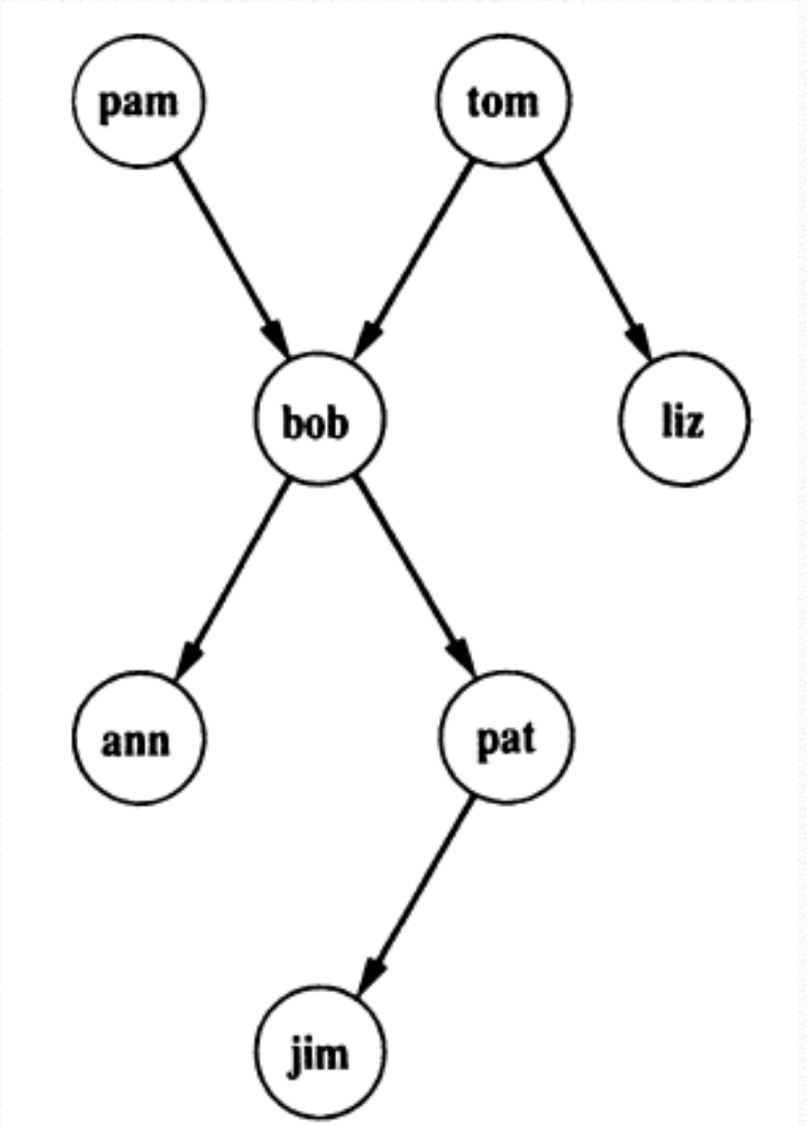
English: Who are Bob's children?



Let's Query

English: Who are Bob's children?

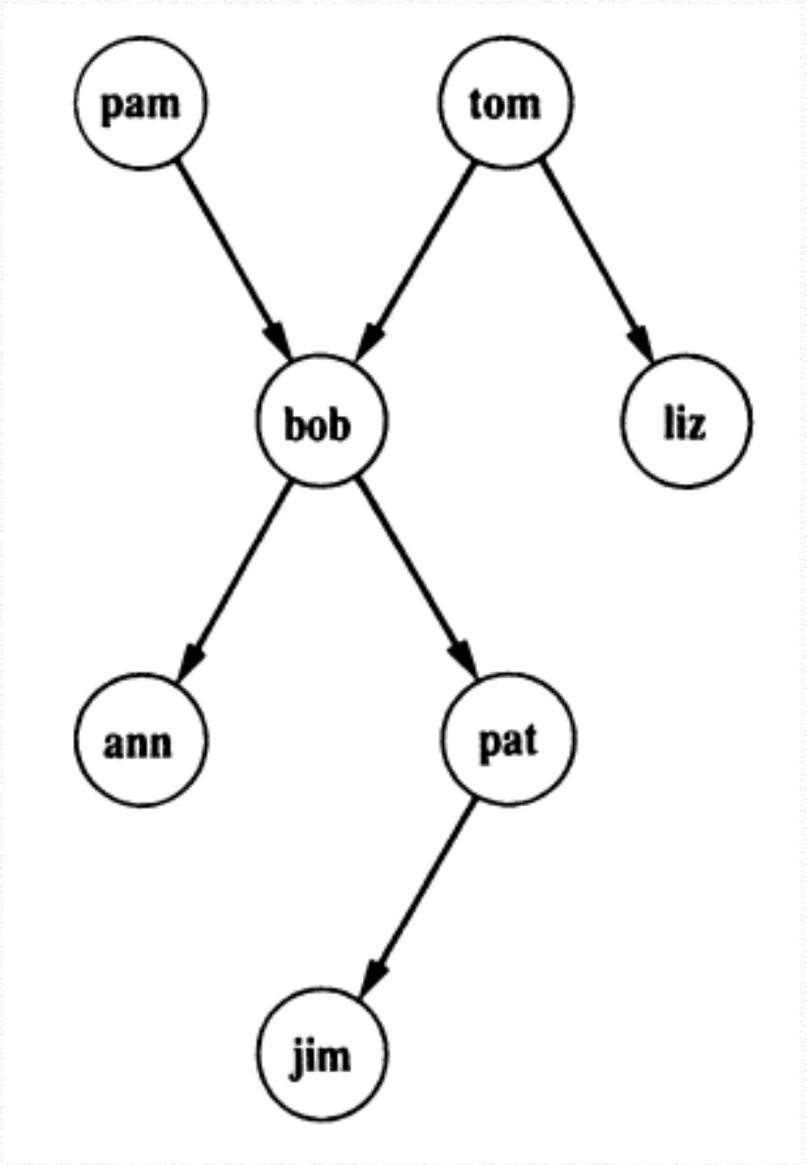
```
?- parent(bob,X).
```



Let's Query

English: Who are Bob's children?

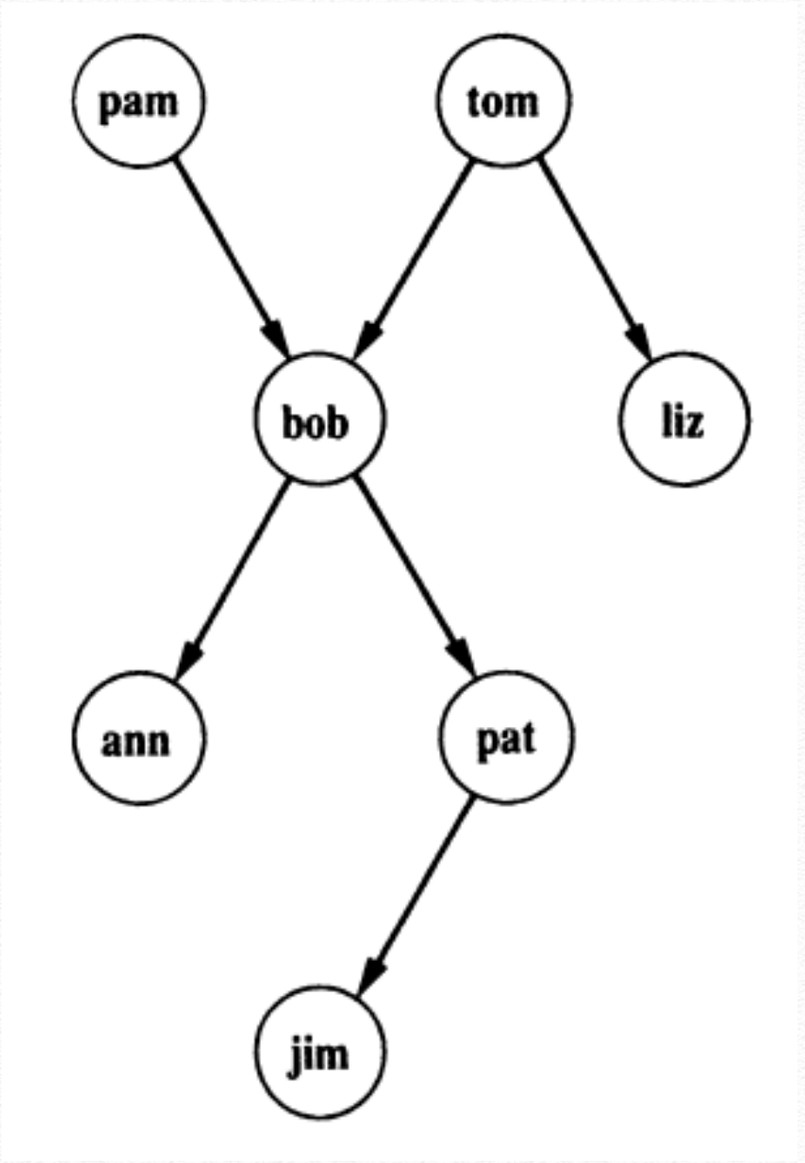
```
?- parent(bob,X).
X = ann
```



Let's Query

English: Who are Bob's children?

```
?- parent(bob,X).
X = ann ;
X = pat.
```

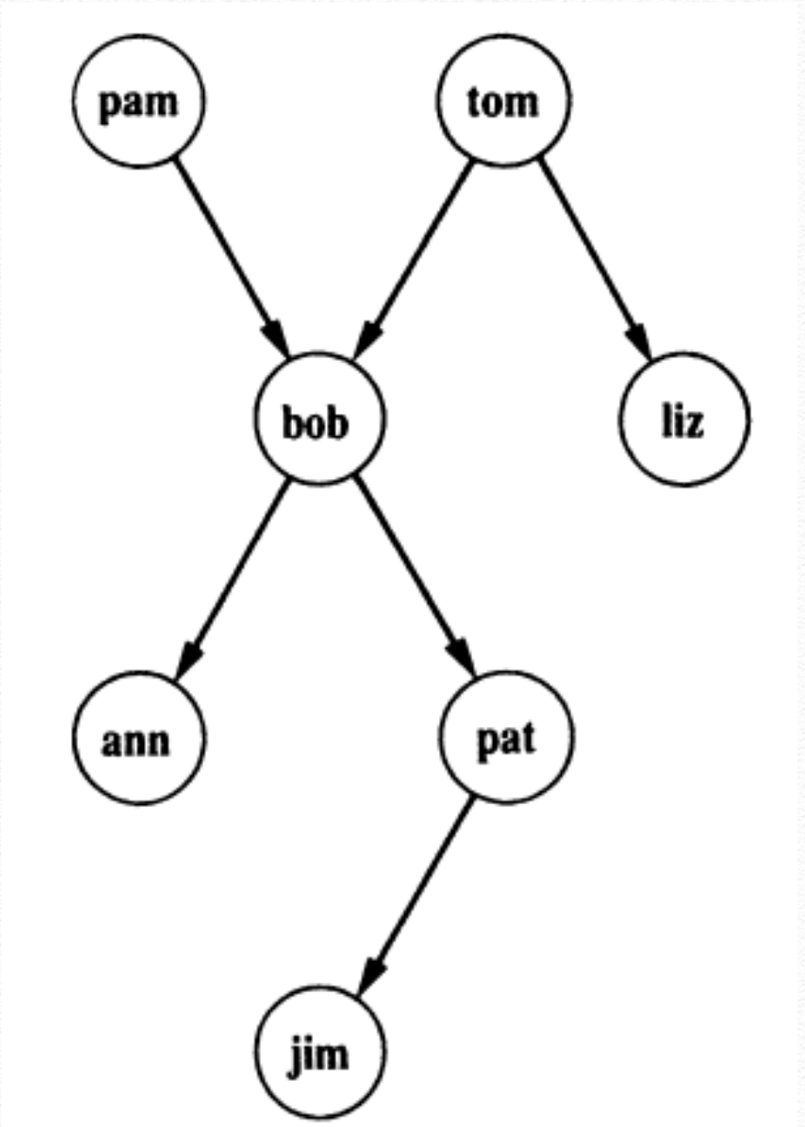


Let's Query

English: Who are Bob's children?

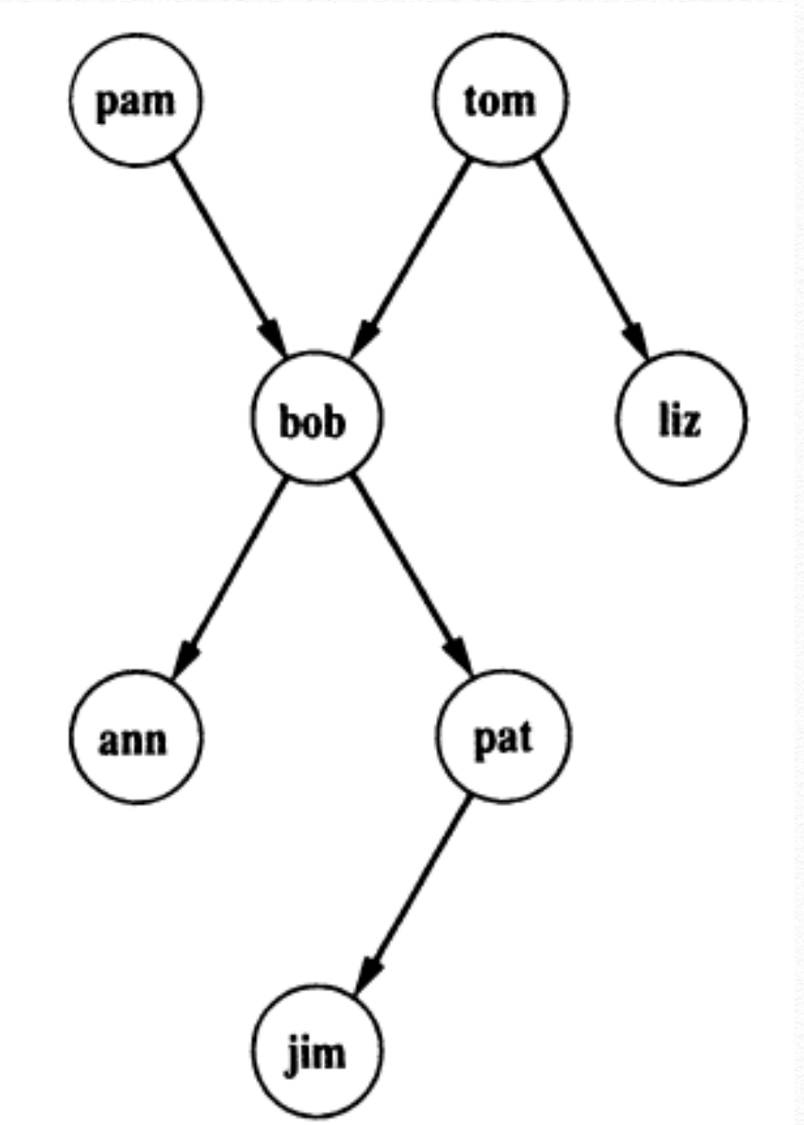
```
?- parent(bob,X).
X = ann ;
X = pat.
```

When there are more than one way to resolve X, Prolog gives the first answer and waits for instruction. Pressing semi-colon gives more answers, pressing enter terminates.



Let's Query

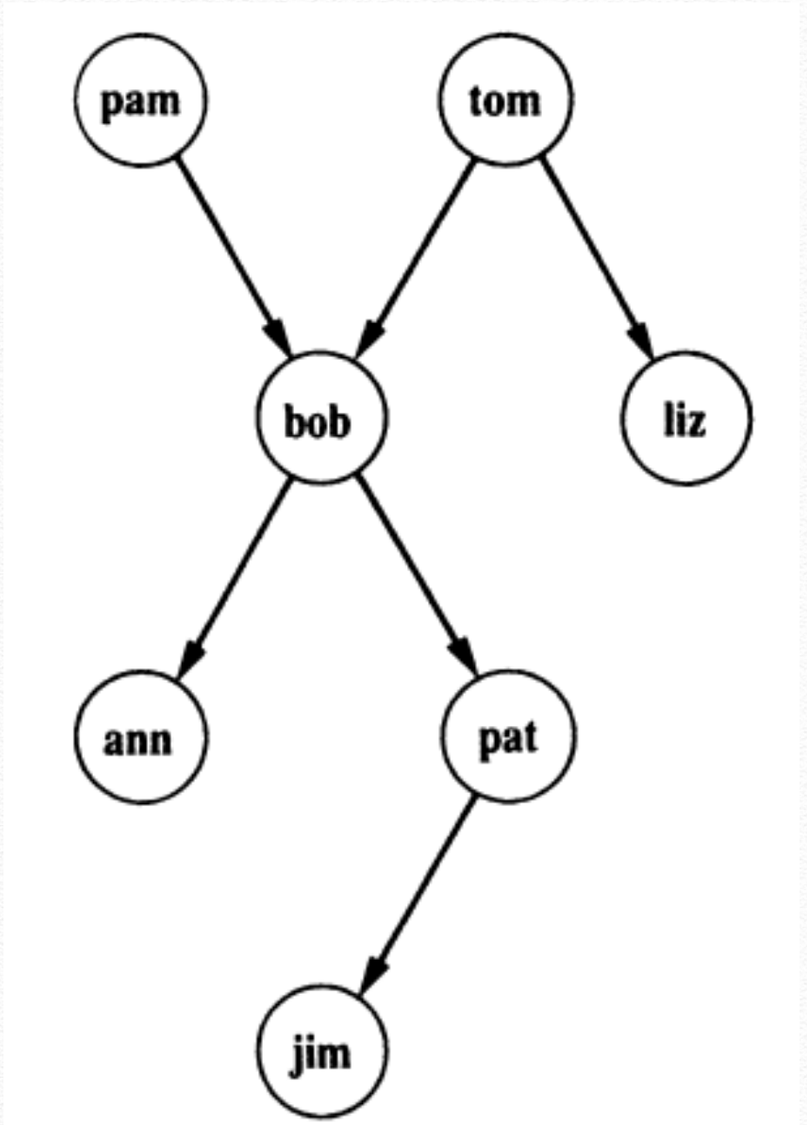
English: Who is a parent of whom?



Let's Query

English: Who is a parent of whom?

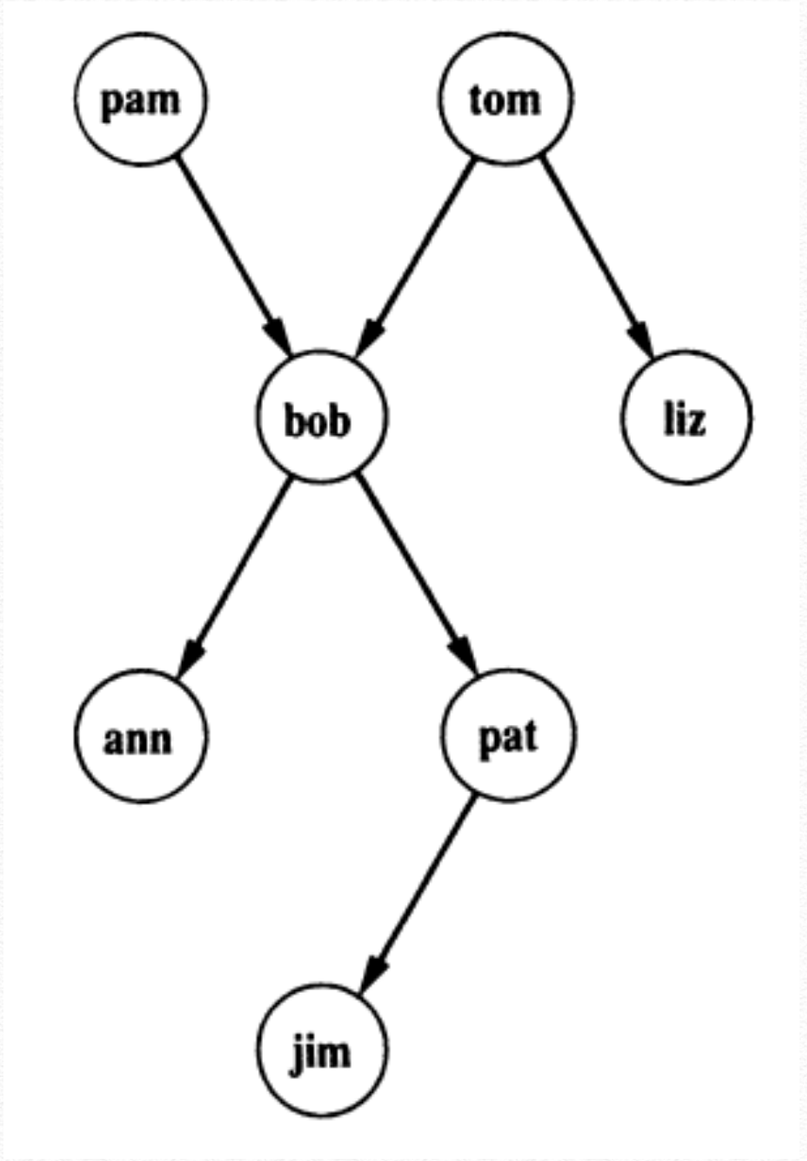
```
?- parent(X,Y).
```



Let's Query

English: Who is a parent of whom?

```
?- parent(X,Y).
X = tom,
Y = bob ;
X = pam,
Y = bob ;
X = tom,
Y = liz ;
..
```



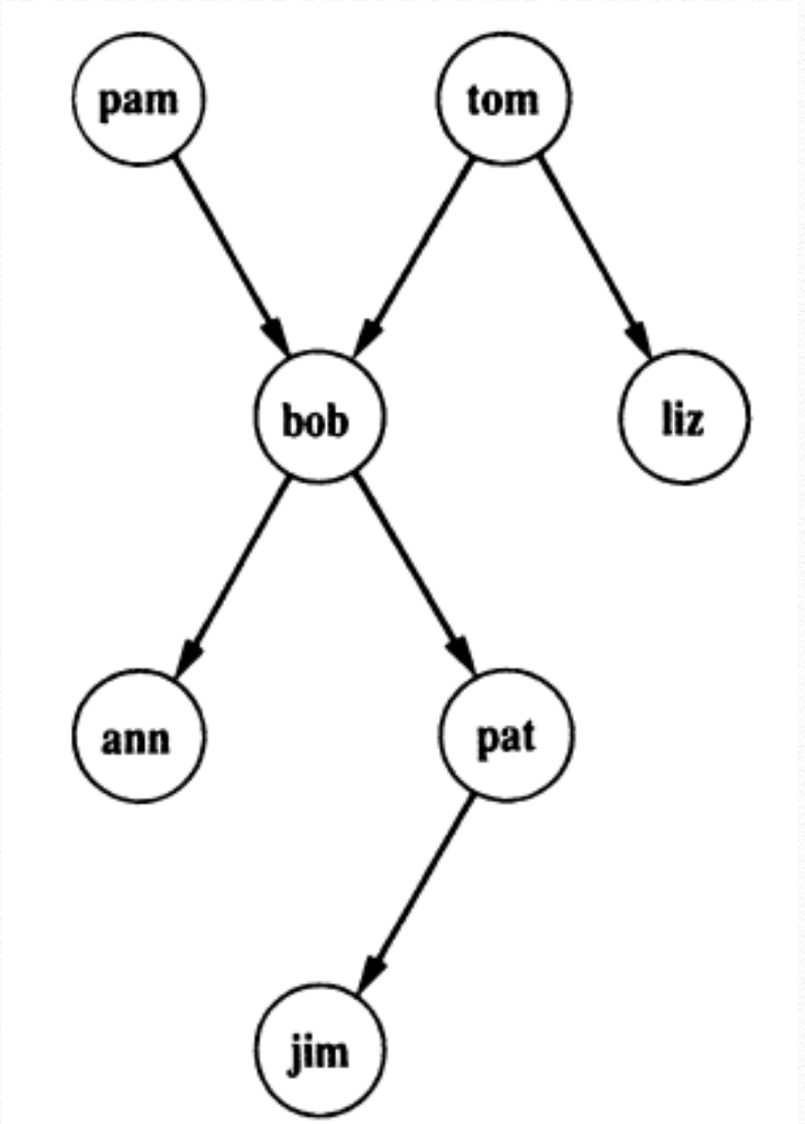
Let's Query

English: Who is a parent of whom?

```
?- parent(X,Y).
X = tom,
Y = bob ;
X = pam,
Y = bob ;
X = tom,
Y = liz ;
..
```

What happens if we write instead:

```
?- parent(X,X).
```



Let's Query

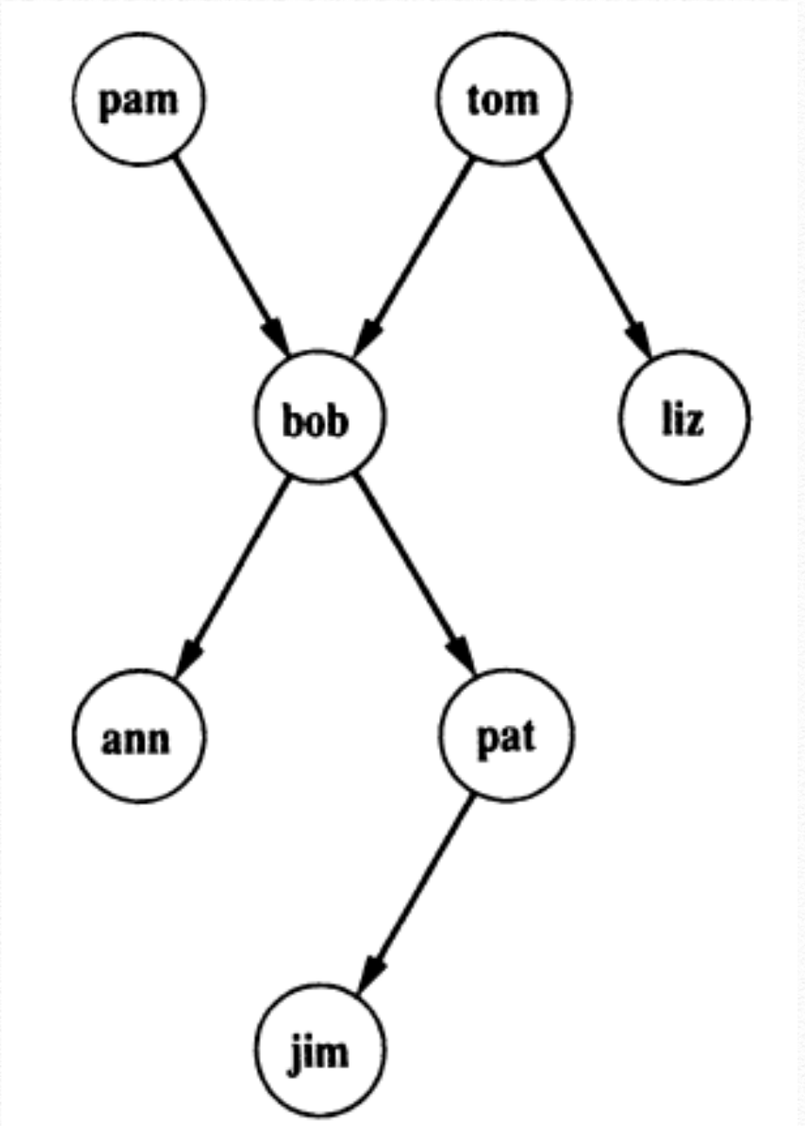
English: Who is a parent of whom?

```
?- parent(X,Y).
X = tom,
Y = bob ;
X = pam,
Y = bob ;
X = tom,
Y = liz ;
..
```

What happens if we write instead:

```
?- parent(X,X).
false.
```

Why?



Questions

A Simple Abstract Interpreter

Most of chapter 1 is preparation for the introduction of the Simple Abstract Interpreter on page 22.

This interpreter represents the theoretical underpinnings for a real Prolog interpreter, but it doesn't work exactly the way a real Prolog interpreter does. It's an *abstraction* -- a simplification -- but it'll give you a good sense of what happens with your programs.

A Simple Abstract Interpreter

Input: A ground **goal** G and a **program** P

Output: *yes* if G is a **logical consequence** of P,
no otherwise

Algorithm:

Initialize **resolvent** to goal G (the query)
while resolvent not empty do
choose a goal A from the resolvent
choose a ground instance of a clause
`A' :- B`$_1$`,…,B`$_n$ from program P
such that A and A' are identical
(if no such goal and clause exist, exit
the while loop)
replace A by `B`$_1$`,…,B`$_n$ in the resolvent
If the resolvent is empty, then output yes,
else output no

Modus Ponens

Modus Ponens is the basic rule of inference in logic.

- noun. (logic) the principle that whenever a conditional statement and its antecedent are given to be true its consequent may be validly inferred. In Latin it means "rule that affirms by affirming".

If P implies Q, and we know that P is true,
then we can infer that Q is true:

$$\frac{P \rightarrow Q \qquad P}{Q}$$

- *if it's Tuesday this must be Belgium* and *it's Tuesday* so *this must be Belgium*

Generalizing Modus Ponens

If

h `:-` $b_1, b_2, \ldots, b_m$

is a clause in the program, and each b_i has been computed from the program, then h can be computed from the program.

- In Prolog instead of the if arrow (→) we have '**:-**' that works like a **reverse arrow** (←).

Step 0. Initialize

Initialize **resolvent** to goal G (the query)

When we present a query to the interpreter, we're really saying "Here's a theorem, go prove it…go show that it can be derived from or is a logical consequence of the program."

So a query like

```
?- a1,a2,…,am.
```

becomes the initial resolvent for our abstract interpreter

Next Step

while resolvent not empty do:
choose a goal A from the resolvent

The proof procedure chooses (arbitrarily)* an atom or conjunct from:

$a_1, a_2, \ldots, a_m$ (the query)

Let's say the procedure selected a_i

*it can be shown that the order of selection is irrelevant in the outcome

Next Step

choose a ground instance of a clause

`A' :- B`$_1$`,…,B`$_n$ from program P

such that A and A' are identical

(if no such goal and clause exist, exit the while loop)

The proof procedure *chooses** a clause from the program whose head matches a_i

For example: `a`$_i$ `:- b`$_1$`,…,b`$_x$`.`

*this is a special kind of choosing. we will come back to it later

Next Step

replace `A` by $B_1, \ldots, B_n$ in the resolvent

The proof procedure then resolves the resolvent:

$$a_1, a_2, \ldots, a_i, \ldots, a_m$$

with the chosen clause from the program:

$$a_i \text{ :- } b_1, \ldots, b_x.$$

yielding:

$$a_1, a_2, \ldots, b_1, \ldots, b_x, \ldots, a_m$$

The Loop

Keep doing this until all the atoms in the body of the resolvent are true:

$a_1, a_2, a_3, \ldots, a_m$

$a_1, \texttt{true}, a_3, \ldots, a_m$

$a_1, a_3, \ldots, a_m$

$a_1, \texttt{true}, \ldots, a_m$

$a_1, \ldots, a_m$.

The Loop

Keep doing this until all the atoms in the body of the resolvent are true:

$a_1, a_2, a_3, \ldots, a_m$

$a_1, \texttt{true}, a_3, \ldots, a_m$

$a_1, a_3, \ldots, a_m$

$a_1, \texttt{true}, \ldots, a_m$

$a_1, \ldots, a_m.$

Each iteration of the loop is a single application of the modus ponens and is called a **reduction.**

The Loop

Keep doing this until all the atoms in the body of the resolvent are true:

$a_1, a_2, a_3, \ldots, a_m$

$a_1, \texttt{true}, a_3, \ldots, a_m$

$a_1, a_3, \ldots, a_m$

$a_1, \texttt{true}, \ldots, a_m$

$a_1, \ldots, a_m$.

Each iteration of the loop is a single application of the modus ponens and is called a **reduction.**

A sequence of answer resolvents that ends with an empty resolvent is called a derivation. If so, the query is proved to be a logical consequence of the program. This process is often called **definite clause resolution** (or simply **resolution**).

Questions?

Questions?

- How about an example?

Exercise 1

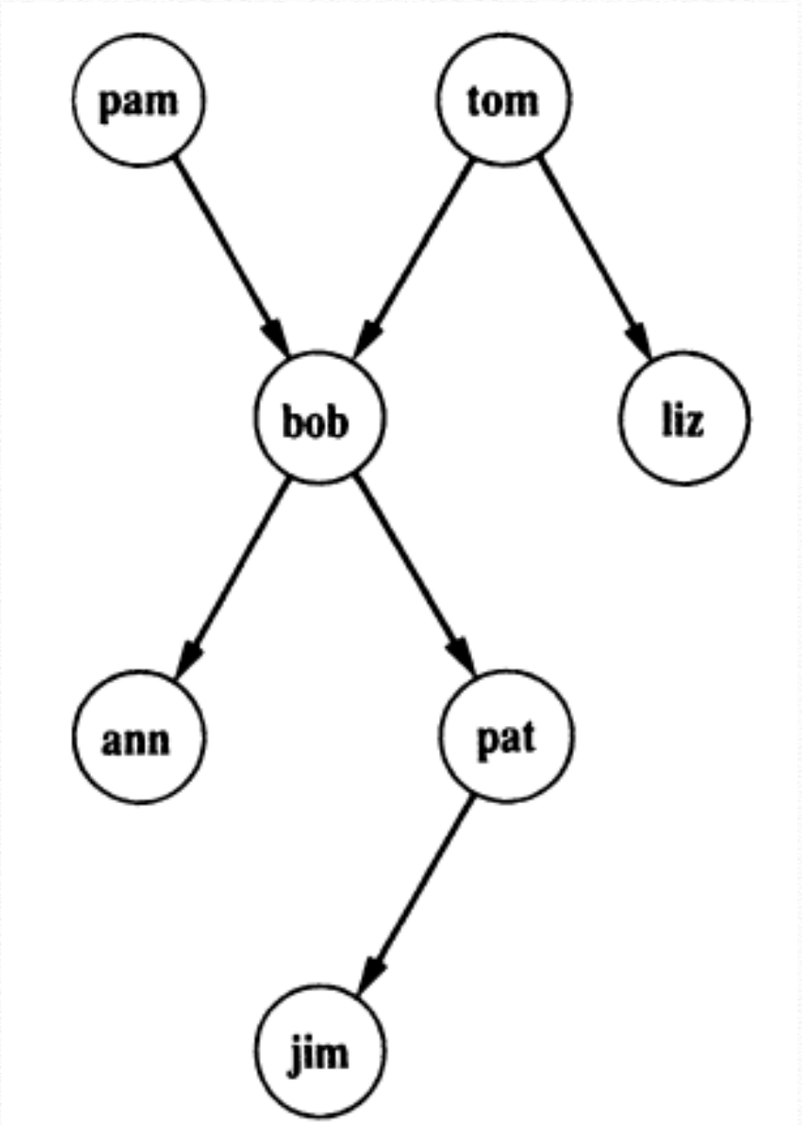
Assuming the parent relation as defined in the previous slides, what will be Prolog's answers to the following queries?

```
?- parent(jim,X).

?- parent(X,jim).

?- parent(pam,X),parent(X,pat).

?- parent(pam,X),parent(X,Y),
parent(Y,jim).
```



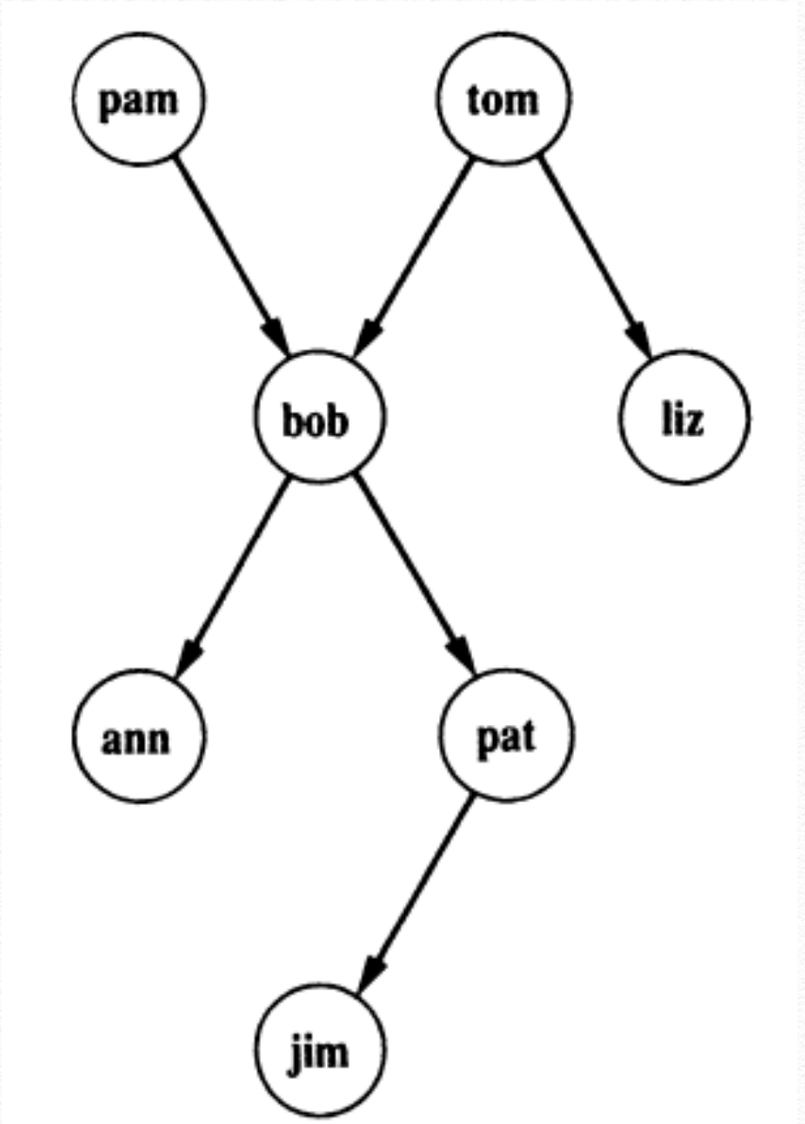
Exercise 2

Formulate in Prolog the following queries about the parent relation:

A. Who is Pat's parent?

B. Does Liz have a child?

C. Who is Pat's grandparent?



Next Class

- We will talk some more about computational model of Prolog, we will talk about Database Programming and Recursion.
- Read Chapter 2 & 3 from the Art of Prolog.