THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

Logic Programming Languages

References

- Concepts of Programming Languages by R. W. Sebesta.
- Prolog Programming for Artificial Intelligence by I. Bratlko

Introduction

Programs in logic languages are expressed in a form of symbolic logic

Use a logical inferencing process to produce results

Declarative rather that *procedural*:

- Only specification of *results* are stated (not detailed *procedures* for producing them)

Proposition

A logical statement that may or may not be true

- Consists of objects and relationships of objects to each other

Symbolic Logic

Logic which can be used for the basic needs of formal logic:

- Express propositions
- Express relationships between propositions
- Describe how new propositions can be inferred from other propositions

Particular form of symbolic logic used for logic programming called *predicate calculus*

Object Representation

Objects in propositions are represented by simple terms: either constants or variables

Constant: a symbol that represents an object

Variable: a symbol that can represent different objects at different times

- Different from variables in imperative languages

Compound Terms

Atomic propositions consist of compound terms

Compound term: one element of a mathematical relation, written like a mathematical function

- Mathematical function is a mapping
- Can be written as a table

Parts of a Compound Term

Compound term composed of two parts

- Functor: function symbol that names the relationship
- Ordered list of parameters (tuple)

Examples:

```
student(jon)
like(seth, osx)
like(nick, windows)
like(jim, linux)
```

Forms of a Proposition

Propositions can be stated in two forms:

- *Fact*: proposition is assumed to be true
- *Query*: truth of proposition is to be determined

Compound proposition:

- Have two or more atomic propositions
- Propositions are connected by operators

Logical Operators

Name	Symbol	Example	Meaning
negation	$\neg$	$\neg$ a	not a
conjunction	$\cap$	a $\cap$ b	a and b
disjunction	$\cup$	a $\cup$ b	a or b
equivalence	$\equiv$	a $\equiv$ b	a is equivalent to b
implication	$\supset$ $\subset$	a $\supset$ b a $\subset$ b	a implies b b implies a

Quantifiers

Name	Example	Meaning
universal	$\forall X.P$	For all X, P is true
existential	$\exists X.P$	There exists a value of X such that P is true

Clausal Form

Too many ways to state the same thing

Use a standard form for propositions

Clausal form:

$B_1 \cup B_2 \cup \ldots \cup B_n \subset A_1 \cap A_2 \cap \ldots \cap A_m$

means if all the As are true, then at least one B is true

Antecedent: right side

Consequent: left side

Predicate Calculus and Proving Theorems

A use of propositions is to discover new theorems that can be inferred from known axioms and theorems

Resolution: an inference principle that allows inferred propositions to be computed from given propositions

Resolution

Unification: finding values for variables in propositions that allows matching process to succeed

Instantiation: assigning temporary values to variables to allow unification to succeed

After instantiating a variable with a value, if matching fails, may need to *backtrack* and instantiate with a different value

Proof by Contradiction

Hypotheses: a set of pertinent propositions

Goal: negation of theorem stated as a proposition

Theorem is proved by finding an inconsistency

Theorem Proving

Basis for logic programming

When propositions used for resolution, only restricted form can be used

Horn clause - can have only two forms

- *Headed*: single atomic proposition on left side
- *Headless*: empty left side (used to state facts)

Most propositions can be stated as Horn clauses

Overview of Logic Programming

Declarative semantics

- There is a simple way to determine the meaning of each statement
- Simpler than the semantics of imperative languages

Programming is nonprocedural

- Programs do not state now a result is to be computed, but rather the form of the result

Example: Sorting a List

Describe the characteristics of a sorted list, not the process of rearranging a list

$$\text{sorted (list)} \subset \forall_j \text{ such that } 1 \leq j < n, \text{list}(j) \leq \text{list}(j+1)$$

The Origins of Prolog

University of Aix-Marseille (Calmerauer & Roussel)

- Natural language processing

University of Edinburgh (Kowalski)

- Automated theorem proving

Terms: Variables and Structures

Variable: any string of letters, digits, and underscores beginning with an uppercase letter

Instantiation: binding of a variable to a value

Lasts only as long as it takes to satisfy one complete goal

Structure: represents atomic proposition

functor (*parameter list*)

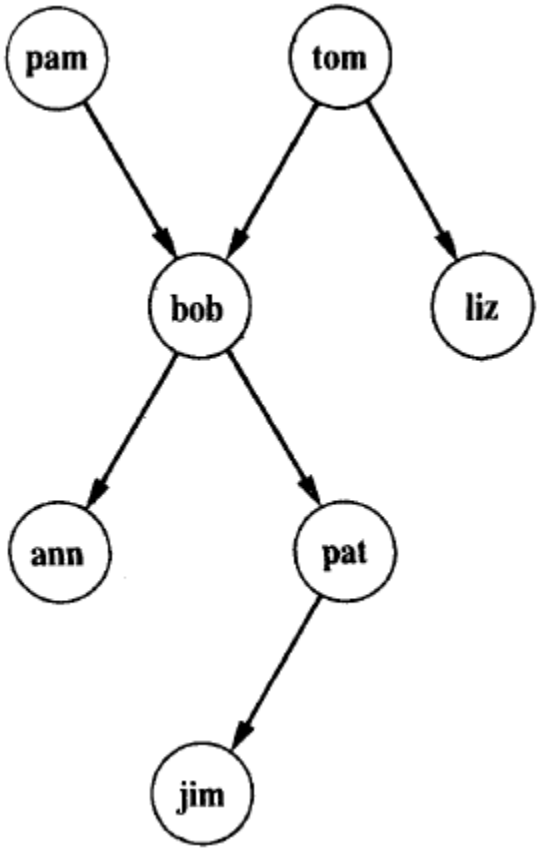
Prolog Fact

A **fact** is a predicate expression that makes a declarative statement about the problem domain. Whenever a variable occurs in a Prolog expression, it is assumed to be **universally quantified**. Note that all Prolog sentences must end with a period

- Used for the hypotheses
- Headless Horn clauses

Examples

parent(pam, bob).
parent(tom, bob).
parent(tom, liz).
parent(bob, ann).
parent(bob, pat).
parent(pat, jim).



This program consists of six clauses.
Each of these clauses declares one fact about the parent relation.
Those are binary relations.

Unary relations

The relations introduced here are male and female.

These relations are unary (or one-place) relations.

female(pam).
male(tom).
male(bob).
female(liz).
female(pat).
female(ann).
male(jim).

Rules

A **rule** is a predicate expression that uses logical implication (:-) to describe a relationship among facts.

Thus a Prolog rule takes the form

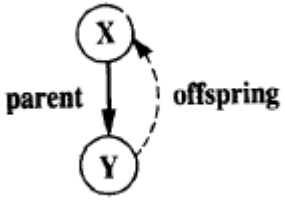
left_hand_side :- right_hand_side

This sentence is interpreted as:

left_hand_side ***if*** *right_hand_side*.

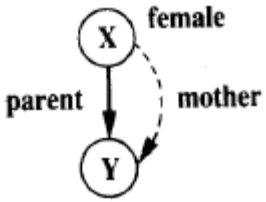
This notation is known as a **Horn clause**. In Horn clause logic, the left hand side of the clause is the conclusion, and must be a single positive literal. The right hand side contains the premises. The Horn clause calculus is equivalent to the first-order predicate calculus.

Example Rules

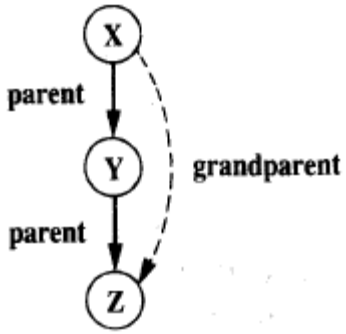


offspring(Y, X) :- parent(X, Y).

head body

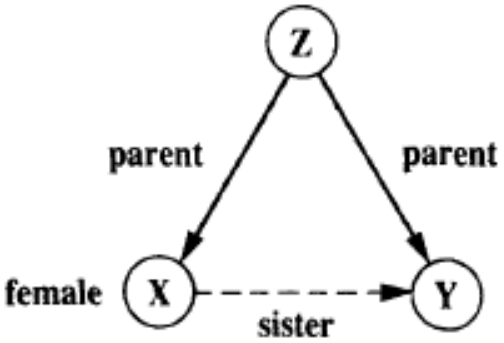


mother(X, Y) :- parent(X, Y), female(X).
i.e., for all X and Y,
X is the mother of Y if
X is a parent of Y and
X is a female.



grandparent(X, Z) :- parent(X, Y), parent(Y, Z).
i.e., for all X and Z,
X is a grandparent of X if
X is a parent of Y and
Y is a parent of X

Negation




```
sister( X, Y) :-
        parent( Z, X),
        parent( Z, Y),
        female( X),
        different( X, Y).
different(X,Y) :- not(X == Y).
```

i.e., for all X and Y,

X is the sister of Y if
Z is a parent of X and
Z is a parent of Y and
X is female and
X and Y are not the same person

Exercise

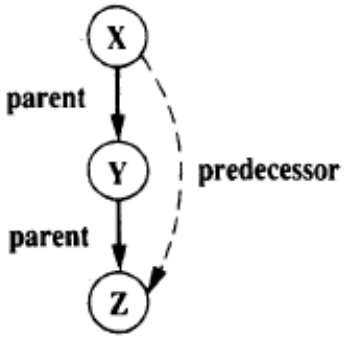
Define a rule called sibling (X, Y) in terms of the parent relations, where X is a sibling of Y if X&Y have a common parent.

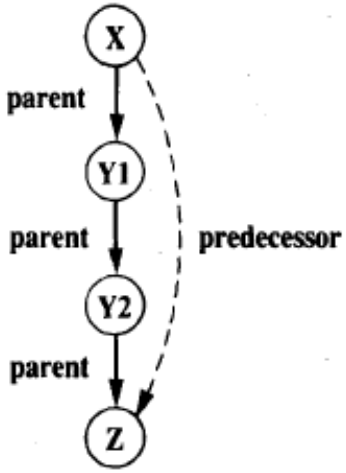
Define a rule called aunt(X, Y) in terms o f the relations parent and sister.

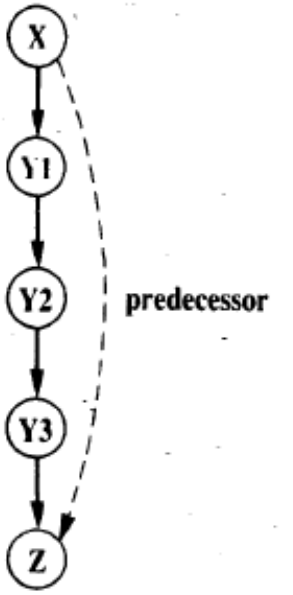
Recursive Rule Definition

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

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







For all X and Z,
X is a predecessor of Z if
there is a Y such that
(1) X is a parent of Y and
(2) Y is a predecessor of Z .

Exercise

In mathematical terms, the sequence F_n of Fibonacci numbers is defined by the recurrence relation

$F_n = F_{n-1} + F_{n-2}$ where

$F_0 = F_1 = F_2 = 1$

Create a prolog program that implements the Fibonacci numbers

Exercise

In mathematics, the factorial of a non-negative integer n, denoted by n!, is the product of all positive integers less than or equal to n, where by definition 0! = 1

Create a Prolog program to calculate the factorial

Queries

The Prolog interpreter responds to **queries** about the facts and rules represented in its database.

The database is assumed to represent what is true about a particular problem domain.

In making a query you are asking Prolog whether it can prove that your query is true. If so, it answers "yes" and displays any **variable bindings** that it made in coming up with the answer.

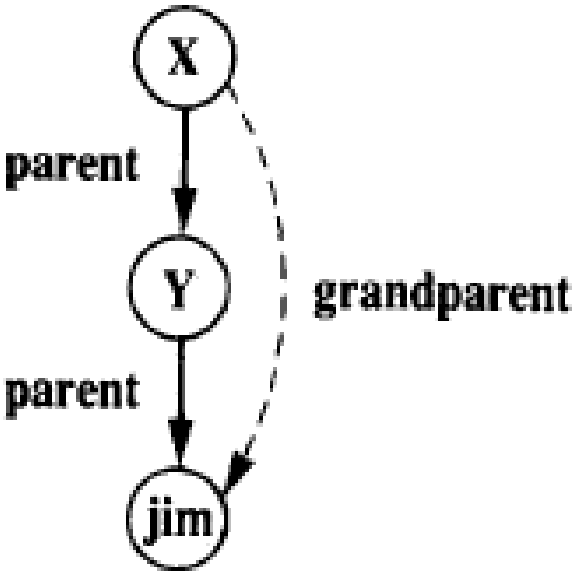
If it fails to prove the query true, it answers "No".

Examples of queries

Question	Prolog	Answer
Is Bob a parent of Pat?	?- parent(bob, pat).	yes
Is liz a parent of pat?	?- parent(liz, pat).	no
Who is Liz's parent?	?- parent(X, liz).	X = tom
Who are Bob's children?	?- parent(bob, X).	X = ann X = pat
Who is a parent of whom? Using 1st logic: Find X and Y such that X is a parent of Y	?- parent(X, Y).	X-pam Y : bob; X:tom Y : bob; X:tom Y : liz; …

More sophisticate expressions

1. Who is a grandparent of Jim?
   1. Who is a parent of Jim? Assume that this is some Y.
   2. Who is a parent of Y? Assume that this is some X .
2. Simillary, Who are Tom's grandchildren?



Prolog proposition	Answer
?- parent(Y, jim), parent(X, Y).	X = bob Y = pat
?- parent(tomn X), parent(X, Y).	X = bob Y = ann; X = bob Y = pat

Exercises

Assuming the parent relation as defined previously, what will be Prolog's answers to the following questions?

1. ?- parent(jim, X).
2. ?- parent(X, jim).
3. ?- parent(pam, X), parent(X, pat).
4. ?- parent(pm, X), parent(X, Y), parent(Y, jim).

Deficiencies of Prolog

Resolution order control

- In a pure logic programming environment, the order of attempted matches is nondeterministic and all matches would be attempted concurrently

The closed-world assumption

- The only knowledge is what is in the database

The negation problem

- Anything not stated in the database is assumed to be false

Intrinsic limitations

- It is easy to state a sort process in logic, but difficult to actually do—it doesn't know how to sort

Applications of Logic Programming

Relational database management systems

Expert systems

Natural language processing

Summary

Symbolic logic provides basis for logic programming

Logic programs should be nonprocedural

Prolog statements are facts, rules, or goals

Resolution is the primary activity of a Prolog interpreter

Although there are a number of drawbacks with the current state of logic programming it has been used in a number of areas