CPS331 Lecture: Introduction to Prolog Last revised January 14, 2014

Materials:

1. Copy of Clocksin and Mellish to show
2. Handout and projectable of isa.pro program from Problem Set 3
3. Handout on using PrologJ
4. Projectable of problem and projectable and demo program for "Garden Dilemma" puzzle

I. Background

A. Prolog is one of two programming languages widely used in AI work - the other being LISP. LISP is the more commonly used in the US, but Prolog is widely used here as well as in other parts of the world.

B. Prolog is programming language based on predicate calculus. The name is a contraction for "Programming in Logic", and the language was originally developed in conjunction with work on automated theorem proof.

C. Probably the best known book on Prolog is

Clocksin, William F. and Christopher S. Mellish. *Programming in Prolog* (Berlin: Springer-Verlag). Currently in its 6th edition, copyright 2013.

SHOW

D. Prolog is utterly different from any other programming language you are likely to have studied.

1. If you are not a CS student, that means that you're really at no disadvantage compared to CS students when it comes to learning the language. In fact, it has been said (and I think with a significant element of truth) that it is easier for a philosopher to learn Prolog than it is a Computer Scientist.

2. If you are a CS student, the differences can be summed up in two key words:

   a) Prolog is higher-level than other languages you know.

   b) Prolog is non-procedural. In brief, a Prolog program specifies what is to be done, rather than specifying how to do it.

   c) Therefore, in order to learn Prolog, you must consciously lay aside some of the thought habits you have formed from working with procedural languages.

II. **Representing Facts Rules, and Queries**

A. When we write in Prolog, we are typically constructing one of three things:

1. A FACT - something that we are asserting to be true. (Similar to the predicates in predicate calculus.)

2. A RULE - something that allows us to infer new knowledge from existing facts and rules. (Similar to implications in predicate calculus)

3. A GOAL - either a query to be answered based on existing facts and rules, or a subgoal generated in the process of answering the query.

4. Typically, the facts and rules reside in a database which you consult. An initial query is entered interactively to initiate computation - which typically results in further goals being generated.

B. Let's look at examples of these from the Prolog program you will be working with for Problem Set 3.

PROJECT isa.pro

1. The program contains a number of facts.

   Examples: isa(snoopy, beagle). and subsequent

   a) A fact has the form of a predicate calculus predicate.

   b) Each fact is followed by a period.

2. The program also contains a number of rules.

   Examples: the two infer rules at the end of the program

   a) A rule has the general form

      head :- body

      (1) This is equivalent to the predicate calculus implication

          body → head

          or, using the rule notation we will look at soon, the rule

          IF body
          THEN head

          The designers of Prolog reversed the order to focus attention on the head, because Prolog uses backwards chaining - so when we are trying to satisfy some goal, we are looking for a rule whose head unifies with it

      (2) A rule is also always followed by a period

      (3) A fact can be thought of as a rule with an empty body, or as a rule whose body consists of the single goal true (which always succeeds)

i.e. `isa(garfield, cat).`

is exactly equivalent to

`isa (garfield, cat) :- true.`

(which may actually be the way a Prolog implementation represents it internally)

(4) In Prolog, rules often have bodies that are composed of more than one subgoal, with the subgoals separated by "," - which stands for "and" in Prolog.

b) Example: the first rule in the example program says "It is possible to infer that something holds if that something is recorded in the database." (`call` is a builtin predicate in Prolog that attempts to satisfy a goal.)

Example: turn tracing on and execute
`infer(isa(garfield, cat)).`

c) Example: the second rule says "It is possible to infer that some property holds for some entity if

The entity belongs to some class (it "isa" something)
And: We can infer that the property holds for that class
And: Inheritance is not blocked by some negative fact

Example: turn tracing on and execute
`infer(isa(garfield, carnivore)).`

d) The univ operator (`=..`) is something we will discuss later. What it does here is allows us to take our original goal apart into the name of the property (`Property`), the entity we are concerned about (`Who`), and - if present - a `Value`. We then discover some `Class` that `Who` belongs to and construct a new inference - does that `Property` hold for that `Class`?

(1) Example: suppose we were trying to infer that garfield eats meat.

Initial goal `infer(eat(garfield, meat))`.
Since garfield is a cat, we now try to infer `eat(cat, meat).`
This, in turn, leads us to try to infer `eat(carnivore, meat)`

(2) Turn tracing on a demonstrate

3. Computation is initiated by entering an initial goal (called a query). This may, in turn generate new goals

Example: refer to trace of infer(eat(garfield, meat))

Initial goal (query): infer(eat(garfield, meat))
Subgoal: eat(garfield, meat) - fails (not explicitly recorded in database)
Subgoal: isa(garfield, Class) - succeeds with Class = cat
Subgoal: eat(cat, meat) - fails (not explicitly recorded in database)
Subgoal: isa(cat, Class) - succeeds with Class = carnivore
Subgoal: eat(carnivore, meat) - succeeds
Subgoal: not ~(eat(garfield, meat)) - succeeds
(CWA, called negation as failure in Prolog)

C. Prolog facts, rules, and goals are constructed from basic building blocks called terms. Prolog has five basic types of term

1. Atoms

a) An atom is a symbolic name for some entity, value, or concept our program must deal with. An atom corresponds to a predicate calculus constant or predicate name.

Examples from isa.pro: garfield, cat, meat

b) In Prolog, there are four ways of writing an atom

(1) A name composed of letters (either case), digits, and the underscore character. The first character must be a lower case letter.

We have already seen some examples from isa.pro
Also: a_red_apple, a11, aBC

But not: _a, 1a, ABC

(2) A sequence of one or more special characters from the following list:

+ - * / ~ < > = \ ` ^ : . ? @ # $ & %

Examples from isa.pro: ~ (in facts), :-, =.. (in rules at end)
But not: a, a

(3) The following one or two character strings:

! ; [] {} (in some contexts:) ,

Examples from isa.pro: , (in rules at end)

(4) Any string of characters enclosed by single quotes. (A ' may be embedded by doubling or escaping it with a backslash.)

Ex: (from Eliza): 'don''t'

2. Numbers. an integer or a real number, written in the conventional way.

Ex: 1, 127, -123, 1.0 3.14159 6.02e23

These are also correspond to predicate calculus constants.

3. Variables

a) A Prolog variable (which corresponds to a predicate calculus variable) is a sequence of letters (either case), digits, and underscores. The first character must be an upper case letter or an underscore.

Examples from isa.pro: Fact, Property, Who, Value, Class
Also: X, XYZ, _XYZ, _xyz, This_is_a_long_name

b) Variable names of the form _0, _1, _2 etc. are generated internally by Prolog during execution. Therefore, you should not use these names explicitly.

c) A single underscore, by itself, is an anonymous variable. If it occurs several times in a given formula, each occurrence is treated as different.

Ex: In the rule p(X) :- q(X)
the two occurrences of X must unify to the same object.

But in the rule p(_) :- q(_),
the two variables could unify differently.

d) Prolog variables typically appear in rules and goals. They are permitted, but less common, in facts. In a rule (or a fact) a variable is regarded as universally quantified; in a goal, it is regarded as existentially quantified.

Ex: dog(X) :- barks(X) is the rule "X is a dog if X barks"
dog(X), as a fact, would claim that everything is a dog!
dog(X), as goal, would ask "is there some such that
X is a dog?"

e) As Prolog is attempting to satisfy a goal, it assigns values to variables - a process technically known as instantiation

Example: isa(garfield, X) results in X becoming instantiated to cat (Demo)

4. Structures.

   a) A Prolog structure consists of an atom (called its functor), followed by a parenthesized list of comma-separated arguments. The number of arguments is called its arity

      Ex: isa(garfield, cat) is a structure of arity 2
      The functor is isa.
      The arguments are garfield and cat

      When writing a structure in this way, it is essential that no spaces separate the functor and the left parenthesis - e.g.

      `roams(X)` is a legitimate structure, but `roams (X)` is not, and would cause the Prolog reader to report a syntax error

   b) There is an alternate way of writing structures for certain atoms that are defined as operators.

      Example from isa.pro: ~purr is equivalent to ~(purr)
      Property =.. [PropertyName, Who | Value]
      is equivalent to
      =..(Property, [PropertyName, Who | Value]

      (1) Certain operators (e.g =..) are builtin to Prolog

      (2) The user can define atoms as operators - e.g. the definition for ~ at the start of isa.pro

   c) A Prolog structure can be used to correspond to a predicate calculus predicate. (There are other uses we won't talk about)

5. Lists

a) A Prolog list consists of a left bracket, followed by a series of comma-separated arguments, followed by a right bracket.

Example from eliza:
```
[thank, you, for, chatting, with, me, '.', have,
a, nice, day,'!']
```

b) An empty list can be written as a left bracket followed immediately be a right bracket - with no space in between

Ex: `[]`, but <u>not</u> `[ ]`

c) A vertical bar, followed by a single argument, can appear just before the right bracket. The argument following the bar stands for the entire rest of the list - and is itself a list

Example: From isa.pro: `[ PropertyName, Class | Value]`

This is would match `[ eat, carnivore, meat ]`
with `PropertyName` instantiated to `eat`, `Class` instantiated to `carnivore`, and `Value` instantiated to `[ meat ]` - (note list)

This would also match: `[ roam, beagle ]`
with `PropertyName` instantiated to roam, `Class` instantiated to beagle, and `Value` instantiated to `[]` (the empty list)

d) The builtin operator `=..` (pronounced univ) can be used to convert between structures and the corresponding list -

Example in isa.pro:

Suppose we invoke the final rule with the goal
`infer(eat(garfield, meat))`

The first univ would instantiate `PropertyName` to `eat`, `Who` to `garfield`, and `Value` to `[ meat ]` (note that it would be a list)

Since the isa call would instantiate `Class` to `cat`, the second univ would instantiate `ClassProperty` to `eat(cat, meat)`

This would result in the new goal infer(eat(cat, meat)), which would in turn create the new goal infer(eat(carnivore, meat)), which would match a fact in the database. (We demonstrated this earlier)

D. Instead of using the standard predicate calculus symbols for operators, Prolog uses symbols for operators that occur on standard computer keyboards.

1. The comma is used for "and".

   Example: the rule above

2. The semicolon is used for "or". However, this is rather rare.

3. Negation is written as "not"or "\+", depending on the dialect of Prolog being used. (The dialect we are using recognizes either, and treats them as equivalent.)

III.Handling Queries

A. We said earlier that a Prolog QUERY has the form of a fact, but is entered to the interpreter rather than appearing in a program.

Example: the queries we used to initiate computation in the isa program

1. The interpreter reports whether the query is true (a fact, or can be inferred from one or more rules).

   Example: Demo using isa.pro

   ```
   infer(isa(garfield, cat)), infer(isa(garfield, dog)).
   ```

2. If the query contains one or more variables, the interpreter also reports the value(s) of the variable(s) that make the query true.

Demo: `infer(eat(mickey, X))`

3. If the query contains one or more variables, the interpreter will allow the user to ask about other ways to satisfy the query by using different values for the variables.

   Example: find all solutions to `infer(isa(garfield, X))`

B. Backtracking

1. Sometimes, in attempting to satisfy a query, Prolog will attempt one or more approaches that don't "work". In this case, Prolog will backtrack until it finds an approach that does work.

   Example: Suppose we want to know what individuals eat cheese.

   This could be answered by the compound query:

   `individual(X), infer(eat(X, cheese)).`

   a) Prolog finds the two solutions `mickey` and `minnie`

      DEMO

   b) It is instructive to see the process involved in finding those wo solutions. To see this, we can add some code to write out each individual being tried.

      `individual(X), write(X), nl, infer(eat(X, cheese))`

      DEMO

      Note that Prolog tries individuals in the order in which they appear in the database. It therefore tries snoopy, nermal, and garfield before mickey but cannot infer that any of them eat cheese. In each case, the failure of the "eat" goal forces backtracking to occur.

2. We will see a more significant use of backtracking in our final example.

3. Actually, when we enter "`;`" at after Prolog reports a variable binding, we are actually initiating backtracking as well - the "`;`" in effect says "I'm not satisfied with that answer - please give me another."

IV. **Using our Prolog.**

A. DISTRIBUTE, GO OVER "Using" handout

B. Prolog operates in two modes:

1. In question answering mode, anything typed is taken as a goal to be satisfied. This is denoted by the ?- prompt. You may think of input in this mode as being preceded by "Please tell me ..."

   DEMO: `Y is 2 + 2.`

2. In consultation mode, Prolog treats all input as facts/rules to be entered in the database for later use in answering questions. You may think of input to this mode as being interpreted as if it were prefixed by "The following is true ...".

   a) Prolog may consult a disk file, as we have done in the examples thus far, or it may consult you directly from the keyboard

      The easiest way to consult a file (though there are others) is to choose Consult from the Interpreter

   b) To consult from the console, type `[user]`

      DEMO: `[user].`

(1) The prompt changes consulting during the consultation.

DEMO:
```
isa(bullwinkle, moose).
isa(moose, mammal).
eats(X, grass) :- isa(X, moose).
```

(2) When through, click the EOF button to return to query mode.

DEMO using above: `infer(eat(X, grass)).`

(Note: we used a rule for eat, instead of fact, to illustrate that both rules and facts can be added in consultation mode!)

c) You should not consult the same file twice - this may result in duplication of predicates and/or errors

DEMO: Turn off directive enforcement, then consult `isa.pro` again, then try `infer(isa(garfield, X))`. Note that each answer comes out twice!

d) Instead, after changing a file, you can reconsult it - which results in replacing old definitions with new ones.

DEMO: Edit isa.pro to change isa fact for garfield to dog
Reconsult isa.pro
Try `infer(isa(garfield, X))`

C. In either mode, comments may be entered enclosed in /* */, or preceded by % (single line comment).

Note examples in isa.pro

D. If it is necessary to abort processing of a Prolog query (e.g. because there is an infinite loop, you can click the Abort button.

   DEMO:

   ```
   repeat, write(hello), nl, fail.
   ```
   Abort the above.

E. To exit the interpreter, the simplest way is to just close the console window.

V. **A More Complicated Example**

A. PROJECT Garden Dilemma Puzzle

B. DEMO - `solve(F, L, B), printSolution(F, L, B).`

C. PROJECT, discuss Prolog program

1. Basic strategy of the "solve" goal - generate and test. Backtracking is used to generate all permutations until one is found that works.

2. "list processing" predicates

VI. There is a **lot** more that could be said about Prolog, but for our purposes this should be sufficient for now