Prolog - PROgramming in LOGic

CS 3711

An overview of the Prolog Language, its structure, uses and its shortcomings!

Presented to:

Dr. Rosebrugh

CS 3711 - Project. By Andy McIntyre, Aaron Webb, John Mullane and Johnny Miller.
Dec. 1, 1999

What is PROLOG?

- Prolog is a Declarative or logical Language.
- Prolog takes only facts and rules to arrive at goals.
- The programmer doesn't tell it how to solve.
- For solving logic and decision problems, Prolog is ideal.
- Typical applications: AI, Database apps, proving theorems, symbolic evaluation (i.e. differentiation).

How does Prolog work?

- Prolog is based on 'Horn Clauses'
- Horn Clauses are a subset of 'Predicate Logic'
- Predicate logic is a way of simply defining how reasoning gets done in logic terms.
- Predicate Logic is a syntax for easily reading and writing Logical ideas.

Predicate Logic…

- To transform an English sentence to Predicate Logic, we remove unnecessary terms.
- This leaves only the relationship and the entities involved, known as arguments.
- Ex: A pie is good = good(pie)
- The relation is ‘good’, the relation’s argument is ‘pie’.
- In Prolog, we call the relation’s name (e.g. “good”) the ‘Functor’. A relation may include many arguments after the functor.

Rules in Prolog

- To infer facts from other facts, Prolog uses Rules.
- The programmer defines rules similarly to the facts.
- Ex: Bill likes cars if they are red =
  `likes(bill, cars):- red(cars).`
- By the way, in prolog, ':-' is pronounced 'if'.

Prolog Queries

- Based on the Rules and Facts, Prolog can answer questions we ask it
- This is known as querying the system.
- We may want to ask, "What does Jim like?"
- In Prolog syntax, we ask: `likes(jim,What).`
  Note: capital W on what

Part I - Program Structure

What you'll need to know about the layout in order to use Prolog!

Putting It Together: *Parts of a Prolog program*

- All programs written in Prolog contain at least 4 parts:
- DOMAINS
- PREDICATES
- CLAUSES
- GOALS

What is DOMAIN?

- The section of code where we define the legal values for any type that is not defined as a standard type. This may include aliasing of types (renaming).
- Domain declarations can also be used to define structures that are not defined by the standard domains.

What are PREDICATES?

- The PREDICATES section is where we define predicates to be used in the CLAUSES section and define the domains for their arguments.
- Symbolic name of a relation
- We found it best to think of predicate declarations as function prototypes.
- Ex: age(string, integer)

What are CLAUSES

- Clauses are the heart of the program.
- A clause is an instance of a predicate, followed by a period.
- Clauses are of two types:
- Facts
- Rules

Clause Facts:

- Facts in the CLAUSE section are relations that are known to be true by the programmer.
- Self standing basis for inference
- A property of an object or a relation between objects.
- Ex: red(car)

Clause Rules:

- Used to infer other facts
- Property or relation known given the fact that some other set of relations are known.
- Ex: Jane can eat food if it is a vegetable on the doctor's list.
- `Can_eat(jane, X):- food(X), vegetable(X), doc_list(X).`

What are GOALS:

- Part of program where queries are made.
- Can be singular or compound.
- Each part of a compound goal is known as a subgoal.
- To satisfy a compound goal (or query) each subgoal must itself be satisfied by the system.

Compound Goals

- Useful when we want to find Conjunction or Disjunction of a number of goals.
- Ex: What are the things do Bill *and* Cindy both like (in common)?
- In Prolog:

```
likes(bill, What), likes(cindy, What).
```

- Similarly, we use ‘;’ as an OR for system queries. (Note: AND takes precedence over OR in multiple subgoal cases.)

Part II: Syntax and Semantics

Making sense of some Prolog code - much of this is likely compiler specific. Variables, wild card, terms, statement delimiters - facts by ., lists, compound data objects and functors. AND OR, arithmetic (integer and real (infix notation), order of operations, comparison, no loops all recursion(121), I/O.

Terms

- Prolog programs are composed of terms.
- Terms are *constants*, *variables*, or *structures*.
- *Constants* are names for specific objects or relationships. The two types are Atoms and integers. The :- symbol is also considered an atom.
- Atoms are things which must begin with a lowercase letter and may be followed by digits, underscores or any case letters.
- If needed, an atom may contain anything and be place inside ‘ ’.

Variables

- Variables are any string that begins with an uppercase sign or an underscore.
- Variables stand for something that cannot be named at the current time.
- Similar to a pronoun in English.
- The single '_' is considered the anonymous variable. It means don't care.
- Variables are instantiated (bound to values) as the program progresses

Atom and Variable Examples:

- Atom Examples:
  a_boy, peanut, 'Jack-Smith', i12345.
- Not Atoms:
  231as, Jack-Smith, _crack.
- Variables Examples:
  Answer, X, I_like, _Marbles.
- Not Variables:
  mother, 3blind_mice.

Structures

- Structures represent the atomic proposition of predicate calculus. The general form is `functor(parameter list).`
- Facts, and relations and rules are considered structures.
- Compound structures – a structure can contain substructures. E.g. instead of

```
drives(john,car)
```

we could have

```
drives(john,car(mercedes))
```

which can greatly aid readability.

Arithmetic in Prolog

- Prolog provides built in functions for doing arithmetic, similar to most major languages.
- Provides built in random number generator.
- Integers and reals have standard operations defined (*,/,+,-)
- In-fix notation is used

Arithmetic continued

Order of operations (for all types):

1. Parentheses
2. Left to right *, /, div, mod
3. Left to right +, -

Other Arithmetic Functions:

X mod Y

Returns the remainder (modulos) of X divided by Y.

X div Y

Returns the quotient of X divided by Y.

abs(X)

If X is bound to a positive value val, `abs(X)` returns that value; otherwise, it returns -1 * val.

cos(X)

The trigonometric functions require that X be bound to

sin(X)

a value representing an angle in radians.

tan(X)

Returns the tangent of its argument.

.

More Functions!

exp(X)

e raised to the value to which X is bound.

ln(X)

Logarithm of X, base e.

log(X)

Logarithm of X, base 10.

sqrt(X)

Square root of X.

random(X)

Binds X to a random real; 0 <= X < 1.

random(X, Y)

Binds Y to a random integer; 0 <= Y < X.

trunc(X)

Truncates X. The result still being a real

val(domain,X)

Explicit conversion between numeric domains

Assignment vs. Comparison (=)

Ex:

```
X = Y+Z*20.
```

The variable X is assigned the value X+Z*20 if X is uninstantiated

X is compared to Y+Z*20 if X has been instantiated earlier on.

(i.e. There is no == for comparison. Tricky. Compiler specific.)

Some dialects of prolog use 'is' for assignment and = for comparison.

Relational operations

return succeed or fail

Symbol	Relation
<	Less than
>	Greater than
>=	Greater or equal
=<	Less than or equal
<> Or ><	Not equal

Types available for Comparison

- Prolog the previous operations for comparison on integers and reals (duh), strings, characters and symbols.
- Characters are converted to ASCII values which are then compared.
- Strings are compared character by character until a pair are not equal. Shorter strings are considered < larger strings.
- Symbols cannot be compared directly due to storage mechanism. They must first be bound to variables or written as strings.

I/O

- I/O can be performed on virtually any device. Standard is keyboard/video.
- Prolog uses write, readln, readint, readch, readreal (some other readtypes as well – all are covered – many compatible).

EX:

```
GOAL

    write("Enter an integer!"),
    readint(Int),
```

Loops

- Prolog provides no support for loops, though it can be 'tricked' into looping. Ex:

```
CLAUSES
        repeat.
        repeat:-repeat.
        typewriter:-
                repeat,
                readchar(C),
                write(C),
                C = '\r',!.
```

Recursion

- Repetition is normally done through recursion. The standard example is:

```
factorial(1, 1) :- !.

factorial(X, FactX) :-
    Y = X-1,
    factorial(Y, FactY),
    FactX = X*FactY.
```

Part III: Data types

Description of Prolog's built-in data types and which are compatible.

Data Types

- Char

A character, implemented as an unsigned byte. Syntactically, it is written as a character surrounded by single quotation marks: 'a'.

- Real

A floating-point number, implemented as 8 bytes in accordance with IEEE conventions; equivalent to C's ***double***.

Data Types

- String

  A pointer to a null terminated sequence of characters (as in C), defined by:

  1. a sequence of letters, numbers and underscores, provided the first character is lower-case; or
  2. a character sequence surrounded by a pair of double quotation marks.

Data Types

- Integer

  A signed quantity, having the natural size for the machine/platform architecture in question.

- Symbol

  A sequence of characters, implemented as a pointer to an entry in a hashed symbol-table, containing strings. The syntax is the same as for strings.

Data Types

- List

  The Prolog equivalent to arrays.
  Lists in prolog are, however, dynamic and similar to LISP in that you can only use CAR and CDR equivalents to access elements. [Head|Tail]

Data Types

- Other built in Domains (probably compiler specific!)

short, ushort, long, ulong, unsigned, byte, word, dword.

These are mostly similar to their C counterparts.

Note: No operator overloading is permitted in Prolog! (see reason for =<)

Part IV: Crazy Things in Prolog

Backtracking, Cuts, Fails, Dynamic Facts!

Lets get Crazy!

Backtracking

- The process used by Prolog to re-satisfy goals
- When a goal or subgoal fails, Prolog attempts to re-satisfy each of the subgoals in turn.
- When no other subgoals can be satisfied, it attempts to find an alternative clause for the goal. Variables are returned to their original values (called uninstantiated).
- Proceeds to satisfy any alternate clauses.

Backtracking

- Example:

```
male(alan).
male(gary).
female(margaret).
parent(alan, gary).
parent(alan, margaret).
mother(X,Y) :- parent(X,Y), female(Y).
father(X,Y) :- parent(X,Y), male(Y).
```

Goal: `?- mother(alan,Mom)`

The "Cut"

- Syntactically described by !
- Special mechanism which controls (prevents) backtracking.
- As a goal, the "cut" always succeeds and cannot be re-satisfied
- Similar to GOTO – powerful but usually ugly
- Once a "cut" is encountered, backtracking cannot retreat past this point
- Two types: Red cut and a Green cut.

Green Cuts!

A 'Green Cut' is when:

- You incorporate a cut when you know that after a certain stage in the clauses, no new meaningful solutions will be generated
- You can save time and memory by eliminating extraneous searching that the system would otherwise need to discover on its own, the hard way.
- The program would run fine without the cut, it is merely an optimization.

Red Cut!

- Changes the logic of a program
- Correct solutions depend on using the cut for omitting certain subgoals from being considered.
- Makes for much worse readability and maintainability.

The "Green Cut" Example

- Example: trapping the input.

```
CLAUSES

        r(X) :- X = 1 , !, write("you entered 1.").
        r(X) :- X = 2 , ! ,write("you entered 2.").
        r(X) :- X = 3 , ! ,write("you entered 3.").
        r(_) :- write("This is a catchall clause.").

GOAL

        r(1).
```

The "Red Cut" Example

- Example: Library access! (from Programming in Prolog 2nd Ed. P. 76.)

```
CLAUSES

        facility(Pers, Fac):- book_overdue(Pers,
        Book),!,basic_facility(Fac).

        facility(Pers, Fac):-general_facility(Fac).

        basic_facility(reference).
        basic_facility(enquiries).

        additional_facility(borrowing).
        additional_facility(inter_library_loan).

        general_facility(X):-basic_facility(X).
        general_facility(X):-additional_facility(X).

        book_overdue('A. Webb', book69).

GOAL

        facility(A. Webb,Y).
```

Fail

- Another built in predicate used for controlling the backtracking process.
- Can be used in the GOAL or CLAUSES sections
- Essentially the opposite of the cut.
- Causes a forced failure in a chain of goals
- Equivalent to adding an impossible goal. (Ex. 2=3)
- Encourages backtracking to continue.

Fail Example:

- Nicer output:

```
CLAUSES

        father(leonard,katherine).
        father(jack, christine).
        father(leonard, jane).

        list_dads:-father(X,Y),
        write(X," is ",Y,"'s father\n"),

        fail.

        list_dads.

GOAL

        list_dads.
```

Dynamic Facts!

- A cool aspect of Prolog is that Facts (clauses) can be added and removed from the program during execution.
- Similarly, facts can be removed on the fly.
- This introduces the possibility of dynamic rules, growing databases and low level 'learning' within a program.
- The dynamic facts can be easily saved to a file and consulted in a later instance of the program.

Dynamic Facts Example:
(building a greedy wish list)

```
CLAUSES
        wish_list(500, 256, 19).
        wish_list(600, 64, 21).

        is_sexy(Clock, RAM, Monitor):-
        Clock >= 450,
        (RAM >= 256;
        Monitor >= 19),
        not(wish_list(Clock, RAM, Monitor)).

GOAL

        consult("list.dba"),
        (is_sexy(733, 512, 21),
        asserta(wish_list(Clock, RAM, Monitor),
        Save("list.dba").
```

Part V: Evaluation!

The perks vs. the complaints

A Whole new approach

- We found 'thinking logically' odd at times
- Prolog's facts and rules seemed strangely minimal for solving problems.
- Recursive calls rather than looping made for some headache.
- Result: learning the language wasn't all that trivial.

Readability and Writability

- Very tough to learn how to trace the way a program goes.
- Backtracking isn't a natural way to trace a program's flow
- Cuts seem very unreadable and tough to see where they should go. Especially Red cuts.
- Cuts and fails make for tough maintainability
- Cuts are often easy to miss. (they are compared to gotos by many authors!)

Data types and control structures

- Domains were easy to understand, though some seemed redundant.
- Control structures are practically non existent. (Should they exist in a declarative language?)
- Prolog doesn't have strict type checking, however our compiler did have some loose rules in place. (May cause headaches for maintenance)

Cost and Reliability

- Many free compilers available
- So many dialects of Prolog available that code is unlikely to be portable between compilers.
- Quite a steep learning curve, so may cost in terms of training and maintenance.

Limitations

- Closed World assumption
- Lack of resolution order control
- The Negation problem (instantiation problems when using not)

Overall

- Prolog seems to be a nice tool for strict logic problems and recursive problems.
- Visual Prolog seems to be trying to push Prolog into mainstream use by adding many OOP features. We do not suggest it for a general use language.
- Its readability and program flow make the language quite difficult for use and maintenance
- Efficient for areas of AI, but inconceivable for most applications.

That's all.