Prolog Programming

Sukanta Ghosh

Prolog Basics

- The syntax and semantic of PROLOG are very close to formal logic . by this time, it must be clear to you that most program reason using logic
- PROLOG language has in it ab built in inference engine and automatic backtracking facility. it helps in efficient implementation of various search strategies.
- This language has productivity and is of program maintenance .
- PROLOG language is based on Universal Formalism of horn clauses . the positive feature of it is its immunity implementation dependencies and program tend to be uniform .
- Because of inherent and parallelism, PROLOG language can be implemented with who is on parallel machines.

Prolog Basics

- The clauses of PROLOG have a procedural and declarative meaning. Because of this understanding of language is easier.
- In PROLOG , each clause can be executed separately as though it is a separate program . Hence modular programming and testing is possible.
- PROLOG free data structure is amenable to complex data structures.
- As a interpreter, PROLOG is suitable for quick prototype an incremental system development .
- Program tracing during development is possible with modest debugging effort in PROLOG.

What is HORN Clause?

- In a Horn Clause, one condition is followed by zero or more conditions. It is represented as
- Conclusion:-
    - Condition_1,
    - Condition_2,
    - Condition_3,
    - Condition_n,

What is HORN Clause?

- The conclusion is true if and only if condition_1 is true and condition_2 is true and condition_3 is true and so on until condition_n is true.
- In simple terms, horn clause consists of a set of statements joined by logical AND's.

Robin

- The principle of resolution is, who clauses can be resolved with one another of them contains a positive literal and other contains a corresponding negative literal with same predicate symbols at the same number of arguments .
- consider the following clause:
  - ~X(a) V Y(p,q) and ~Y(p,q) V T(r,s)
- these 2 clauses can be unified to give:
  - T(r,s) V ~X(a)

Parts of Prolog Program

- It consists of set of clauses.
- A clause is either a fact or a rule.
- A fact is used to indicate a simple data relationship between elements called objects.
- Eg. Likes(kumar, toffees)
- Likes ← Relationship
- Kumar and Toffees ← Objects

Parts of Prolog Program

- A predicate is the abstract sense of relation that holds between a certain number of arguments.
- A predicate is identified by the predicate name and its arity (number of arguments).
- A predicate can have any number of arguments.

Sample database

- Likes(kumar, toffees)
- Likes(ram, aircraft)
- Likes(mani, coffee)
- Likes(ram, cars)

Queries to Database

- Once data base is created one can make queries to it.
- A simple query consists of predicate name and arguments.
- Eg. Like(ram, cars)
- Would return value true.
- Eg. Likes(muril, bike)
- Would return value false.

Queries to Database

- It is also possible that one can have a variable for an argument.
- Eg. Likes(ram, What)
- Would have the answer
- What = aircraft
- What = cars

How does PROLOG solve a query?

- It tries to match (this process is termed as unification) the arguments of the query with the facts in the database.
- If the unification succeeds, the variable is said to be instantiated.
- It is also possible that one can have variable for all the arguments.
- The query likes(Who, What), would result in:
- Who = kumar, What = toffees
- Who = ram, What = aircrafts

Compound Queries

- The queries that were posed to the systems were simple ones.
- It is also possible to pose compound queries to the system.
- For this, consider the "likes" database again.
- The query likes(mani, What), likes(kumar, What)
- Has the meaning "Is there an item which kumar and mani likes?"
- The system will respond What = toffees

Data Structure in PROLOG

- List is the most important data structure in PROLOG.
- This is nothing but a collection of ordered sequence of terms.
- The elements of the list are written between square brackets separated by commas.
- Eg. [apple, orange, mangoes, grapes]

Head and Tail of List

- The symbol "|" divides the list into two parts, the head and tail of the list.
- Eg. [apple | Rest] will result in
- Rest = [orange, mangoes, grapes]
- Head = apple
- Tail = Rest
- An empty list is represented by [ ].

Program 1: Print all members of a list.

- Write a clause that uses to recursion.
- Writelist([ ]) /* if list is empty, stop recursion */
- Writelist([H | T]) :-

  Write([H]), /* write the first element of the list */

  Writelist([T]). /* recursive call of the clause */

Program 2: Print the list in reverse order

- Rev_print([ ])   /* if the list is empty stop recursion */
- Rev_print([H | T]):-

  Rev_print(T),

  Write(H).

Program 3: Finding the length of the list

- Has_length([ ], 0).
- Has_length([H | T], N):-
- Has_length(T, N1),
- N = N + 1.

LISP Language

LISP – An AI Language

- LISP = LISt Processing
- Developed by John McCarthy.
- Symbolic Processing Language that represents information in lists and manipulation these lists to derive information.

Preliminaries of LISP

- Used for manipulating lists.
- The basic elements are treated as symbols irrespective of whether they are numeric or alphanumeric.
- Basic data element of LISP is an atom, a single word or number that stands for some object or value.

- LISP has tow types of atom: numbers and symbols.
- Number represents numerical values.
- Symbols represents combination of alphabets and numerical.
- Collection of symbols constitutes a list.
- Eg. (apple orange grapes mango)

Basic Primitives

- Arithmetic Primitives
- Boolean Primitives
- List Manipulation Primitives

Arithmetic Primitives

- Basic arithmetic operations of addition, subtraction, multiplication and division.
- Arithmetic operations are carried out on data represented in prefix form.
- E.g. If you want to add: 25+35+45+55+65
- Then it is represented as: (+25(+35(+45(+55(+65)))))
- Other arithmetic primitive are: DIFFERENCE(-), TIMES(*), QUOTIENT(/).

Boolean Primitives

- Provide a result which is Boolean in nature i.e. true or false.
- ATOM: find out whether the element is an atom or not.
- Eg: (ATOM, RAMAN) results in T
- NUMBERP: if atom is number or not.
- Eg: (NUMBERP 20) results in T
- Eg: (NUMBERP RAM) results in NIL

Boolean Primitives

- LISTP: if the input is a list or not.
- Eg: (LISTP (25 35 45)) results in T
- ZEROP: whether the number is zero or not.
- Eg: (ZEROP 26) results in NIL
- ODDP and EVENP: whether the given number is odd or even.
- Eg: (ODDP 65) results in T
- Eg: (EVENP 78) results in T

Boolean Primitives

- EQUAL: whether two given list are equal or not
- Eg: (EQUAL ‘(34 44 55) ‘(22 33 55)) results in NIL
- GREATERP: whether the first is greater then second
- Eg: (GREATERP ‘45 ’34) results in T
- LESSERP: whether the first is lesser than second
- Eg: (LESSERP ‘87 ‘31) results in T

List Manipulation Primitives

- The purpose of list manipulation primitive are for:
  - Creating a new list
  - Modifying an existing list with addition, deletion or replacement of an atom
  - Extracting portions of a list
- Values are assigned to variable by SETQ primitives.
- Eg: (SETQ A 22)   assign 22 to variable A
- Eg: (SETQ TV ONIDA)   assign TV=ONIDA

Thank You