Introduction to Logic Programming and Prolog

What is Logic Programming?

There are many (overlapping) perspectives on logic programming

- Computations as Deduction
- Theorem Proving
- Non-procedural Programming
- Algorithms minus Control
- A Very High Level Programming Language
- A Procedural Interpretation of Declarative Specifications

Computation as Deduction

- Logic programming offers a slightly different paradigm for computation: *COMPUTATION IS LOGICAL DEDUCTION*
- It uses the language of logic to express data and programs.

  Forall X and Y, X is the father of Y if

  X is a parent of Y and the gender of X is male.
- Current logic programming languages use first order logic (FOL) which is often referred to as first order predicate calculus (FOPC).
- The first order refers to the constraint that we can quantify (i.e. generalize) over objects, but not over functions or relations. We can express *"All elephants are mammals"* but not

  "for every continuous function f,

  if n<m and f(n)<0 and f(m)>0 then

  there exists an x such that

  n<x<m and f(x)=0"

Theorem Proving

- Logic Programming uses the notion of an *automatic theorem prover* as an interpreter.
- The theorem prover derives a desired solution from an initial set of axioms.
- Note that the proof must be a "constructive" one so that more than a true/false answer can be obtained.
- E.G. The answer to

  exists x such that x = sqrt(16)
- should be

  x = 4 or x = -4
- rather than

  true

Non-procedural Programming

- Logic Programming languages are non-procedural programming languages.
- A non-procedural language one in which one specifies **WHAT** needs to be computed but not **HOW** it is to be done.
- That is, one specifies:
  - the set of objects involved in the computation
  - the relationships which hold between them
  - the constraints which must hold for the problem to be solved
- and leaves it up the the language interpreter or compiler to decide **HOW** to satisfy the constraints.

Algorithms Minus Control

- Nikolas Wirth (architect of Pascal) used the following slogan as the title of a book:

  Algorithms + Data Structures = Programs
- Bob Kowalski offers a similar one to express the central theme of logic programming:

  Algorithms = Logic + Control
- We can view the LOGIC component as:

  A specification of the essential logical constraints of a particular problem
- and CONTROL component as:

  Advice to an evaluation machine (e.g. an interpreter or compiler) on how to go about satisfying the constraints)

A Very High Level Language

- A good programming language should not encumber the programmer with non-essential details.
- The development of programming languages has been toward freeing the programmer of more and more of the details...
  - ASSEMBLY LANGUAGE: symbolic encoding of data and instructions.
  - FORTRAN: allocation of variables to memory locations, register saving, etc.
  - ALGOL: environment manipulations
  - LISP: memory management
  - ADA: name conflicts
  - ML: explicit variable type declarations
  - JAVA: Platform specifics
- Logic Programming Languages are a class of languages which attempt to free us from having to worry about many aspects of explicit control.

A Procedural Interpretation of Declarative Specifications

- One can take a logical statement like the following:

  Forall X and Y, X is the father of Y if X is a parent of Y and the gender of X is male.
- which would be expressed in Prolog as:

  father(X,Y) :- parent(X,Y), gender(X,male).
- and interpret it in two slightly different ways:
  - **declaratively** - as a statement of the truth conditions which must be true if a father relationship holds.
  - **procedurally** - as a description of what to do to establish that a father relationship holds.

Some Underlying Ideas

Logic Programming languages typically embody a number of useful ideas:

- pattern invoked procedures
- unification matching
- built-in failure driven search mechanism
- Deductive database
- rule-based programming

Pattern Invoked Procedures

- Carl Hewitt (MIT) first articulated the useful notion of specifying a procedure to call by a

  description of the inputs offered and the results desired
- rather than the conventional mechanism:

  the procedure name
- This frees the programmer from the requirement of knowing the procedure name.
- This suggests that there may be NO or SEVERAL procedures which may match the pattern.

Unification

- Unification is a pattern matching operation between two terms, both of which can contain variables.
- A *substitution* is an assignment of variables to values.
- Two terms unify if there is a substitution that makes the terms identical.

  Unifying f(X,2) and f(3,Y) produces X=3, Y=2
- A most general unifier (mgu) is a substitution that unifies the terms w/o 'over-assigning' any variables.
- The result of applying a most general unifier to a set of terms results in a most general instance (mgi).
- E.g., unify f(X,Y) and f(1,A)
  - A substitution: X=1, Y=2, A=2
  - The mgu: X=1, Y=A
  - The mgi: f(1,Y)

Search

- a Logic Programming 'procedure' can either fail or succeed. If it succeeds, it may have computed some additional information (conveyed by instantiating variables).
- Question: What if it fails.....? Answer: find another way to try to make it succeed.
- Most logic programming languages use a simple, fixed search strategy to try alternatives:
  - if a goal succeeds and there are more goals to achieve, then remember any untried alternatives and go on to the next goal.
  - if a goal succeeds and there are no more goals to achieve, then stop with success.
  - if a goal fails and there are alternate ways to solve it, then try the next one.
  - if a goal fails and there are no alternate ways to solve it and there is a previous goal, then propagate failure back to the previous goal.
  - if a goal fails and there are no alternate ways to solve it and no previous goal then stop with failure.

Deductive Database

- Most logic programming languages have a common database which any procedure can access and modify.
- It is sometimes called an assertional database.
- It is similar to the blackboard model of program communication. (and for that matter to the Fortran COMMON mechanism)
- The database is used to represent both PROGRAMS and DATA in a uniform way.
- Datalog

Rule-Based Programming

- Logic Programming languages provide one kind of rule-based programming environment.
- Programs are usually made up of many "independent" rules, each one of which captures a part of the computation.
  - toEnroll(X,freshman,cse110) do ...
  - toEnroll(X,underGrad,cse???) do ...
  - toEnrol(X,grad,cse???) do ...
  - toEnrol(X,grad,cis???) do ...
  - ...
- Advantages of this approach include modularity, easy of adding additional capabilities, ease of understanding each case.
- This idea shows up in the programming language ML as well.

A Short History

1965 Efficient theorem provers. Resolution (Alan Robinson)

1969 Theorem Proving for problem solving. (Cordell Green)

1969 PLANNER, theorem proving as programming (Carl Hewett)

1970 Micro-Planner, an implementation (Sussman, Charniak and Winograd)

1970 Prolog, an implementation (Alain Colmerauer)

1972 Book: Logic for Problem Solving. (Kowalski)

1977 DEC-10 Prolog, an efficient interpreter/compiler (Warren and Pereira)

1982 Japan's 5th Generation Computer Project

~1985 Datalog and deductive databases

1995 Prolog interpreter embedded in NT

PROLOG is the FORTRAN of Logic Programming

- Prolog is the only widely used logic programming language.
- As a Logic Programming language, it has a number of advantages
  - simple, small, fast, easy to write good compilers for it.
- and disadvantages
  - It has a fixed control strategy.
  - It has a strong procedural aspect
  - limited support parallelism or concurrency or multi-threading.

The Zebra Puzzle

- Here is a classic example of a constraint satisfaction puzzle.
- There are five houses, each of a different color and inhabited by men of different nationalities, with different pets, drinks, and cigarettes.
- Given the facts to the right, who drinks water and who owns the zebra?

- the englishman lives in the red house
- the spaniard owns the dog.
- coffee is drunk in the green house
- the ukrainian drinks tea.
- the green house is immediately to the right of the ivory house.
- the old gold smoker owns snails.
- kools are being smoked in the yellow house.
- milk is drunk in the middle house.
- the norwegian lives in the first house on the left.
- the camel smoker lives next to the fox owner.
- kools are smoked in the house next to the house where the horse is kept.
- the lucky strike smoker drinks orange juice.
- the japanese smokes parlaiments.
- the norwegian lives next to the blue house.

A Solution - Preliminaries

- We begin by defining some binary relations between people:
  - X hasLeftNeighbor Y - X has Y as his immediate left neighbor.
  - X rightOf Y - X lives somewhere to the right of Y.
  - X nextTo Y - X and Y live in adjacent houses.
  - X isNot Y - X and Y are distinct people.
- Take hasLeftNeighbor as a primitive relation, and define the others:
  - X nextTo Y if X hasLeftNeighbor Y or Y hasLeftNeighbor X.
  - X rightOf Y if X hasLeftNeighbor Y or X hasLeftNeighbor Z and Z rightOf Y.
  - X isNot Y if X rightOf Y or Y rightOf X.
- We will introduce constant symbols to stand for the five people:
  - rightGuy, midrightGuy, middleGuy, midleftGuy, leftGuy.
- We will define a predicate differ which holds if all of its arguments are distince people:

  differ(X1,X2,X3,X4,X5) if X1 isNot X2 and X1 isNot X3 and X1 isNot X4 and X1 isNot X5 and X2 isNot X3 and X2 isNot X4 and X2 isNot X5 and X3 isNot X4 and X3 isNot X5 and X4 isNot X5.

A Prolog Solution

```
:- op(500, xfy,
    [hasLeftNeighbor,rightOf,nextTo,isNot]).

rightGuy hasLeftNeighbor midrightGuy.
midrightGuy hasLeftNeighbor middleGuy.
middleGuy hasLeftNeighbor midleftGuy.
midleftGuy hasLeftNeighbor leftGuy.

X nextTo Y :- X hasLeftNeighbor Y.
X nextTo Y :- Y hasLeftNeighbor X.

X rightOf Y :- X hasLeftNeighbor Y.
X rightOf Y :- X hasLeftNeighbor Z, Z rightOf
    Y.

X isNot Y :- X rightOf Y.
X isNot Y :- Y rightOf X.
```

```
differ(X1,X2,X3,X4,X5) :-
  X1 isNot X2, X1 isNot X3, X1 isNot X4, X1 isNot X5,
  X2 isNot X3, X2 isNot X4, X2 isNot X5,
  X3 isNot X4, X3 isNot X5,
  X4 isNot X5.

?- Englishman = RedHouser,
   Spaniard = DogOwner,
   CoffeeDrinker = GreenHouser,
   Ukranian = TeaDrinker,
   GreenHouser hasLeftNeighbor IvoryHouser,
   WinstonSmoker = SnailOwner,
   KoolSmoker = YellowHouser,
   MilkDrinker = middleGuy,
   Norwegian = leftGuy,
   CamelSmoker nextTo FoxOwner,
   KoolSmoker nextTo HorseOwner,
   LuckySmoker = OJDrinker,
   Japanese = ParliamentSmoker,
   Norwegian nextTo BlueHouser,
  differ(GreenHouser,YellowHouser,RedHouser,IvoryHouser,
       BlueHouser),

    differ(ZebraOwner,FoxOwner,HorseOwner,SnailOwner,DogOw
    ner),
  differ(OJDrinker,MilkDrinker,TeaDrinker,CoffeeDrinker,
       WaterDrinker),
  differ(Englishman,Spaniard,Norwegian,Japanese,Ukranian),
  differ(KoolSmoker,WinstonSmoker,ParliamentSmoker,
       LuckySmoker,CamelSmoker).
```

Answer

```
| ?- [zebra].
Englishman = middleGuy
RedHouser = middleGuy
Spaniard = midrightGuy
DogOwner = midrightGuy
CoffeeDrinker = rightmostGuy
GreenHouser = rightmostGuy
Ukranian = midleftGuy
TeaDrinker = midleftGuy
IvoryHouser = midrightGuy
WinstonSmoker = middleGuy
SnailOwner = middleGuy
KoolSmoker = leftmostGuy
YellowHouser = leftmostGuy
MilkDrinker = middleGuy
Norwegian = leftmostGuy
CamelSmoker = midleftGuy
FoxOwner = leftmostGuy
HorseOwner = midleftGuy
LuckySmoker = midrightGuy
OJDrinker = midrightGuy
Japanese = rightmostGuy
ParliamentSmoker =
    rightmostGuy
BlueHouser = midleftGuy
ZebraOwner = rightmostGuy
WaterDrinker = leftmostGuy
```