Logic Programming and Learning

Logic Programming (LP)

and
Inductive Logic Programming (ILP)

When

II Semester 2003–2004

Who

Ashwin Srinivasan: ashwin@cse.iitd.ernet.in

How to do this course

Lectures. Main topics, theoretical results, some examples.

Laboratory. Lots of examples, practical use of LP and ILP.

Every week

1. Attend lectures

2. Do at least 1 laboratory session

Lecture Schedule

1. Propositional Logic Programming
2. First-order Logic Programming
3. Computations and Answers
4. Introduction to Model Theory
5. Introduction to Proof Theory
6. Proof Theory (contd.)
7. Subsumption Theorem, Generality Orderings and Lattices
8. Generality Orderings and Lattices
9. Introduction to ILP Theory
10. ILP Theory (contd.)
11. ILP Implementation
12. ILP Experimental Method
13. ILP Applications
14. Introduction to Learning Theory (if time permits)

Lab Schedule

1. Proplog
2. Datalog
3. Prolog
4. Introduction to ILP
5. Generalisation in ILP
6. Generalisation in ILP (contd.)
7. Scientific Discovery with ILP

Symbolic Logic as a computer language

2 stages in software development

1. Specification
   - usually not computer executable
   - correct
2. Implementation
   - computer executable
   - correct
   - efficient

Consider: $\underline{x} = (x_1 x_2 \ldots x_n)$, a sequence of numbers

– Now examine the specifications:

S1 $\underline{x}$ is ordered if $\forall i, j \ (i < j) \Rightarrow (x_i < x_j)$

S2 $\underline{x}$ is ordered if $\forall i \ (x_i < x_{i+1})$

– But what about implementation?

  * S1 is $O(n^2)$ but S2 is $O(n)$

  * An implementation of S2 in C:

```
typedef struct listelem{
int val;
struct listelem *next;
}
typedef struct listelem *next;

ordered(x)
list(x);
{
register list l;
int xi, ok;

if (!x) return 0;
xi = x->val; ok = 1;
for (l = x->next ; l ; l = l->next)
if (!(ok = xi < l->val)) break;
else xi = l->val;
return (ok);
}
```

Logic programming is about writing specifications in symbolic logic *and* executing them directly on a computer

The standard formalism is as follows:

Specifications written in a subset of first-order predicate logic (“clausal form”)

↓

A particular inference system to execute statements written in clausal form (“resolution”)

Historic aside

1965. Robinson discovers resolution

1972. Kowalski introduces clausal form as programs

1973. Colmerauer implements Prolog

1976. 1^{st} Logic Programming Workshop at Imperial College

1977. Clark links negation and finite failure

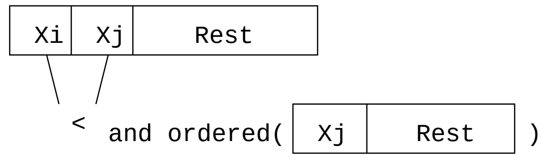
1981. Japan announces 5^{th} Generation Computer Systems project

1984. Lloyd publishes book

Examples of logic programs

A sequence with 1 element is ordered

```
ordered([X]).
ordered([Xi,Xj|Rest]):-
   Xi < Xj,
   ordered([Xj|Rest]).
```




```
factorial(0,1).
factorial(N,M):-
   N1 is N - 1,
   factorial(N1,M1),
   M is N*M1.
```

n! = n*(n-1)!

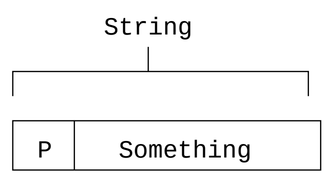
```
likes(steve,Anyone):-
   buys(Anyone,ilp_book).
```

Steve likes anyone who buys the ILP book

```
buys(ilp_book,X):-
   smart(X),
   rich(X).
```

The book may not be very cheap!

```
prefix(P,String):-
   append(Something,P,String).
```



P is a prefix of a string, if you can append something to P to give the string!

But, this is jumping ahead

– We will start with propositional logic programs

Computing with propositions

Propositions are symbols to which we will assign a truth value of either *true* (t, or 1) or *false* (f, or 0) but not both. For e.g. *paris_is_in_england* (*false*) *sarek_is_a_vulcan* (*true*)

> Usually the symbols $p, q \ldots$ will be used to denote propositions.

Propositions may be joined together using connectives like $\wedge$ (and), $\vee$ (or), and $\sim$. Recall the truth-tables:

p	q	$p \wedge q$	$p \vee q$	$\sim p$	$\sim q$
f	f	f	f	t	t
f	t	f	t	t	f
t	f	f	t	f	t
t	t	t	t	f	f

One more truth-table is of interest. This concerns the connective $\leftarrow$. The statement $p \leftarrow q$ is to be read as "if q then p".

p	q	$p \leftarrow q$
f	f	t
f	t	f
t	f	t
t	t	t

If you have not seen this before, it may be surprising. For e.g.

$flatworld$	$humanmonkeys$	$flatworld \leftarrow humanmonkeys$
f	f	t

Note: $p \leftarrow q \equiv p \vee \sim q \equiv \sim q \vee p$

Clauses

Statements of the form
$p_1 \vee p_2 \ldots \;\leftarrow\; q_1 \wedge q_2 \ldots$ are called *clauses*

$p_1 \vee p_2 \ldots$ is sometimes called the *head* of the clause, and $q_1 \wedge q_2 \ldots$ the *body*

If the head has *exactly* 1 proposition without a $\sim$, and the body does not have any $\sim$ symbols, then the clause is called a *definite* clause. Thus:

Clause	Definite clause?
$p \;\leftarrow\; q \wedge r$	$\surd$
$p \vee q \;\leftarrow\; r \wedge s$	$\times$
$p \;\leftarrow\; q \wedge \sim r$	$\times$
$p \;\leftarrow$	$\surd$

A note on syntax

You may see the following variants:

- The symbol $\leftarrow$ written as ":-"

- The symbol $\wedge$ written as ","

- The symbol $\vee$ written as ";"

- The statement $p \leftarrow$ written as simply "p"

- Clauses terminated with a "."

In the laboratory, the clause $p \leftarrow q \wedge r$ is written as:

p:- q, r.

A Proplog "expert" system

Here are some rules for identifying animals:

```
is_mammal :- has_hair.
is_mammal :- has_milk.
is_bird :- has_feathers.
is_bird :- can_fly, has_eggs.
is_carnivore :- is_mammal, eats_meat.
is_carnivore :- has_pointed_teeth, has_claws, has_pointy_eyes.
cheetah :- is_carnivore, has_tawny_colour, has_dark_spots.
tiger :- is_carnivore, has_tawny_colour, has_black_stripes.
tiger :- is_carnivore, has_tawny_colour, has_black_stripes.
penguin :- is_bird, cannot_fly, can_swim.
```

Now here are some statements about a particular animal:

```
has_hair.           fat.
lazy.               big.
has_green_eyes.     has_tawny_colour.
nice.               eats_people.
eats_meat.          has_black_stripes.
```

What are the logical consequences of all the clauses?

Proplog: not expressive enough

Suppose we wanted to represent facts about more than 1 animal

- Animals 1 ($peter$) and 2 (bob) are both hairy. We will need 2 propositions: *has_hair_peter* and *has_hair_bob*.

- But what about the clause *is_mammal* $\leftarrow$ *has_hair*. That is, how do we derive the logical consequences that $peter$ and bob are mammals?

- We need to replace the "mammal" clause with 2 new ones:

  is_mammal_peter $\leftarrow$ *has_hair_peter*

  is_mammal_bob $\leftarrow$ *has_hair_bob*

- Now, we have to also rewrite *is_carnivore* $\leftarrow$ *is_mammal*, *eats_meat*.

Further, suppose we find out about a third animal ($fred$) ...

– Clearly, this is tedious. We want to be in a position to say:

Peter has hair Bob has hair

"Any animal that has hair is a mammal"

We need *predicates*, *functions* and *variables*

First-order logic: alphabet

Constant symbols. Name specific objects. Start with a lower-case letter ($peter$, $mcmxii$ etc.)

Function symbols. Name a functional relationship between objects. Start with a lower-case letter (sin,cos, $+$ etc.)

Variable symbols. Stand for objects or functions without naming them explicitly. Start with an upper-case letter (X, Y etc.)

Predicate symbols. Name a relation on the world of objects. Start with a lower-case letter (son, $\leq$ etc.)