The theory behind logic programming

As its name suggests, logic programming is based on logic, specifically a restricted form of predicate logic called horn clauses with inference done using resolution.
This lecture ...

Revises predicate logic.

Transforms formulae in the predicate calculus into clauses in conjunctive normal form.

Explains resolution as a method of inference.

Shows how Prolog uses a restricted form of clauses called horn clauses.

Briefly describes how LP can differ from Prolog.
Where did Prolog come from?

Dissatisfaction with procedural programming in the 1960s & 1970s.

Procedural programming makes programmers think about how to solve problems in terms of a particular model of a computer.

Programming should be about describing a problem – not about details of solving problems.

Logic is a brilliant description language.
A revision of predicate logic

Logic =
1. a way of expressing arguments (n.b. not argument means discussion, not parameters);
2. Formal method of deciding if the arguments are valid.

Logic is used to represent
1. propositions;
2. relationships between propositions
3. valid inferences from one proposition to another.
A revision of predicate logic

Predicate logic uses **terms** to represent objects:

Terms can be:

- **constant symbol** – single individual or concept (~atom);
- **variable symbol** – introduced with a quantifier (∀, ∃);
- **compound terms** – consisting of function symbol (= functor) and arguments (= parameters), e.g. `birth_date(year(Y), month(M), day(D))`. 
A revision of predicate logic

Expressing relationships:
 consist of predicate symbol and arguments

Atomic proposition = predicate symbol & ordered sequence of terms as arguments

\[
birth_date(\text{year}(Y), \text{month}(M), \text{day}(D))
\]

• predicate symbol = \(birth_date\)
• function symbol = \(year, month, day\).
A revision of predicate logic

Forming more complex propositions:

Combine atomic propositions using connectives:

\[ Ax \land By \lor Cz \]

We know that formulas can be rewritten – logical equivalences:

\[ P \land (Q \lor R) \equiv (P \land R) \lor (Q \land R) \]
Clausal form

Quantifiers – allow us to talk about a set of individuals and what is true about them:

∀x[Px] – P is true whatever x stands for;

∃x[Px] – there is an x such that P is true.
Clausal form

Predicate logic can be translated into clause form.

Insight: logical equivalences can be used to rewrite formulae into logical equivalent formulae.

Translation process has 6 steps.
Translating into clausal form – step 1

Remove implications – i.e. $\rightarrow$, $\leftrightarrow$

$\forall x [\text{Man}(x) \rightarrow \text{Mortal}(x)]$

is transformed into:

$\forall x [\neg \text{Man}(x) \lor \text{Mortal}(x)]$
Translating into clausal form – step 2

Remove negation inward

\[ \neg \forall x[[\text{Man}(x) \rightarrow \text{Living}(x)] \]

is transformed into:

\[ \forall x[\neg \text{Man}(x) \lor \neg \text{Living}(x)] \]

(A positive or negated atomic proposition is also called a literal.)
Translating into clausal form – step 3a

Skolemising - $\exists$

Remove existential ($\exists$) quantifiers by introducing new constant symbols = **Skolem constants**

$\exists x [\text{Female}(x) \land \text{Mother}(x, \text{edward})]$ is transformed into:

Female(pers32) $\land$ Mother(pers32, edward)
Translating into clausal form – step 3b

Skolemising – in the presence of $\forall$

We can’t use the same method as we used for $\exists$

$\forall x [\text{Human}(x) \land \exists y \text{Mother}(x, y)]$

is transformed into:

$\forall x [\text{Human}(x) \land \exists y \text{Mother}(x, \text{pers32})]$

Means pers32 is the mother of everyone.
Skolemising – in the presence of ∀

So we replace variable with functions that return the value corresponding to its argument.

∀x[Human(x) ∧ ∃yMother(x, mum_of(x))]

Returns the value of mum_of(x).
Translating into clausal form – step 4a

Moving $\forall$ outwards

Moving $\forall$ outwards doesn’t change meaning – it’s a convenience thing

$\forall x [\text{Man}(x) \rightarrow \forall y [\text{Woman}(y) \rightarrow \text{likes}(x,y)]]$

is transformed into:

$\forall x [\forall y [\text{Man}(x) \rightarrow \text{Woman}(y) \rightarrow \text{likes}(x,y)]]$
Translating into clausal form – step 4b

Making $\forall$ implicit

All variables are now existentially ($\forall$) quantified.
We don’t have to state this explicitly – just remember that this is the case.

$\forall x[\forall y[\text{Man}(x) \rightarrow \text{Woman}(y) \rightarrow \text{likes}(x,y)]]$

is transformed into:

$\text{Man}(x) \rightarrow \text{Woman}(y) \rightarrow \text{likes}(x,y)$
Translating into clausal form – step 5a

Distributing ∧ over ∨

Remember: we have removed existential quantifiers; Universal quantifiers are implicit and we’re only using the conjunctions ∧, ∨ (and ¬ in negated literals);

Now change into conjunctive normal form - i.e. no conjunctions inside disjunctions:
Translating into clausal form – step 5b

Distributing \( \land \) over \( \lor \)

Examples:

\[(A \land B) \lor C \quad \text{becomes} \quad (A \lor B) \land (B \lor C)\]
\[(A \lor B) \land C \quad \text{becomes} \quad (A \land B) \lor (B \land C)\]
Translating into clausal form – step 6a

Putting into clauses

Nesting of clauses is unnecessary:

\[
((A \land B) \land C) \equiv A \land B \land C
\]

\[
(A \land (B \land C)) \equiv A \land B \land C
\]

Order doesn’t matter:

\[
A \land B \land C \equiv C \land A \land B
\]
Translating into clausal form – step 6b

Putting into clauses

We don’t need to explicitly write $\land$ as we know it is there:

$$A \land B \land C \equiv A, B, C$$

And the logical formula has become clauses.
Translating into clausal form – step 6c

Putting into clauses

Each individual clause consists of a disjunction of literals – and we can omit the explicit $\lor$

$L_1 \lor L_2 \lor L_3 \lor L_4 \equiv L_1, L_2, L_3, L_4$

So, we’ve translated formulae into clausal form without losing meaning.
We need a notation for writing clauses.

We write clauses, one after another – order of clauses isn’t significant.

Within a clause:
• Write the unnegated literals, separated by “;”
• Write ‘:-’
• Write the negated literals (with \(~\) deleted), separated by “,” and ending with “.”
Notating clauses - 2

This is beginning to look like Prolog.

\[(\text{person(adam)} \land \text{person(eve)}) \land \\
((\text{person(x)} \lor \neg \text{mother(x, y)} \lor \neg \text{person(y)})\]

becomes:

\[
\text{person(adam)} :- \ . \\
\text{person(eve)} :- \ . \\
\text{person(x)} :- \text{mother(x, y)}, \text{person(y)}. \\
\]

*If this was “real” Prolog, we’d call these facts.*
Resolution is another rule of inference:

i.e. a way of deciding how a proposition follows from other propositions

Resolution works on formulae in clausal form – given two related clauses, it derives a new clause.

New clause is the consequence of the existing clauses.
Making inferences – resolution - 3

But resolution is *refutation complete*.

This means that, when a set of clauses is inconsistent, we will be able to derive the empty clause – i.e. there is no interpretation that makes all clauses true.

How can we use this? If we have the consistent formula \( \{ A_1, A_2, \ldots, A_n \} \), the formula \( B \) is a consequence of this formula when the formula \( \{ A_1, A_2, \ldots, A_n, \neg B \} \) is inconsistent.
Horn clauses

Horn clause = clause with no more than 1 unnegated literal.

Clausal theorem proving with horn clauses gives us sufficient power.

It’s also easier to use resolution with horn clauses – making it feasible to compute with horn clauses.
Horn clauses

2 kinds of horn clause:

**headed:**

\[
\text{mother}(X, Y) :\neg \text{parent}(X, Y), \text{female}(X).
\]

**headless:**

\[
:\neg \text{mother}(A, B).
\]

For any theorem-proving task, we need only 1 headless clause and at least one headed clause.
Prolog

Based on resolution of horn clauses:

- Uses the linear input resolution strategy.
- This restricts the choice of which clauses to resolve.
- Start with a goal (= headless clause) and resolve with one of the hypotheses (i.e. a program clause).
- When a goal matches with the head of a clause, resolve by removing the goal and replacing with the body of the rule.

This is exactly what the meta-interpreter does.
Prolog

is even more restricted

- Has a fixed way of choosing literal to be resolved – the first literal in the goal clause.
- This means that Prolog always satisfies the first subgoal before moving on to the next subgoal.
- Prolog is prone to infinite search – e.g:
  \[
  \text{goal}(X) :\text{-} \text{goal}(X), \text{ then}(Y).
  \]
- Prolog doesn’t use the occurs check in unification (see last lecture’s notes).
Logic Programming \(\equiv\) Prolog

Prolog works well on sequential architectures but it doesn’t have to be this way:

• We can change the order in which the search space is explored (see last lecture).
• We can have “proper” unification with the occurs check (see last lecture).
• We could change the matching algorithm to one based on constraints (see next lecture).
• We can radically adapt Prolog to run on parallel architectures – and use different non-determinism (see following lectures).
What should you do?

Try some verbal arithmetic problems (also known as cryptarithmetic problems).

Try some map colouring problems, e.g. colouring in the states of Australia.