Logic Foundations of Prolog & its AI Applications

Dr. Mathias Fonkam
Outline

- Why Prolog?
- The Programming Paradigm - declarative
- History of Prolog & motivating AI applications
- Prolog Implementations: SWI, AMZI etc
- Prolog by Example:
  - Defining facts
  - Queries
  - Rules
- Logic Basis: Horn-clauses + Resolution
- AI Applications of Prolog
  - Constraint-based problems
  - Expert Systems
  - Natural Language Processing
Why Prolog?

• Excellent for AI symbolic processing – AI renaissance!!
  – theorem proving – Knowledge representation & reasoning
  – Natural language process – language translators (DCG)
  – Expert systems, deductive & decision support systems
  – Automated search – game playing, constraint-based problems
• Excellent for CS research – great for rapid prototyping!!
• New language & programming paradigm – declarative
• Sound Math basis – Horn-clause logic & computation of relations (kind of generalizes relational DB theory)
  – Correctness/completeness proofs of its programs can be shown
  – Higher level abstraction (code applies to far more domains!)
• Easy & consistent syntax – means concise coding
• Built-in pattern-matching & backtracking: AI need!
The Prolog Paradigm

**Imperative** (procedure driven) programming

\[ \text{Programs} = \text{Algorithms} + \text{Data Structures} \]

\[ \Rightarrow \text{Structured Programming} \]

\[ \Rightarrow \text{OOP: Programs} = \text{Classes} + \text{Objects} \]

OOP: Think in Objects that encapsulate data & routines!

**Logic programming?** - **Declarative**

\[ \text{Programs} = \text{Rules} + \text{Facts} \]

\[ \text{Programs} = \text{Logic} + \text{Control} \]

**Functional programming:** \( \lambda \)-calculus!

\[ \text{Composition of Functions} = \text{Programs} \]
History

• Kowalski: late 60’s Logician who showed logical proof can support computation.

• Colmerauer: early 70’s Developed early version of Prolog for natural language processing.

• Warren: mid 70’s First version of Prolog that was efficient.
Implementations of Prolog

- [https://github.com/jiprolog/jiprolog/jiprolog](https://github.com/jiprolog/jiprolog/jiprolog)
Prolog programming idea

- A *program* consists of a collection of universally quantified logical assertions -- axioms that characterize the essential properties of the problem domain.
- Solutions are computed by *proving* logical consequences of the chosen axioms.
Prolog programming idea

• Prolog programs are invoked through *queries* -- existentially quantified logical assertions

• The Prolog system determines if a *query is a logical consequence* of the program -- and if so, values of constituent unknowns must also be reported – with given KB

Prolog is a *General Problem-solver*!!
A1. Facts. Examples:

Consider the following fact described in natural language as a sentence in English:

John likes Mary.

This is a fact with two objects: John, Mary and one relation: likes.

We can represent in ProLog as:

likes (john, mary) .

likes is the predicate name. john, mary are named arguments. Predicate and arguments are in lower case letters. Fact terminated by . (dot) character?
A1. Facts. More Examples:

Sentence in English: John is a student.

an object: John

a relation: student status (studentship)

Fact in Prolog: student(john).
A2. Facts. More Examples:

Phrase in English: Nature
zero objects, no objects

Fact in Prolog: nature.
A Prolog KB of Facts

parent(tom,bob).
parent(pam,bob).
parent(tom,liz).
parent(bob,ann).
parent(bob,pat).
parent(pat,jim).

• A relation is a collection of facts
• parent is the name of a relation – as function?
Queries to our Family KB:

\[
\begin{align*}
& \text{parent(tom,bob).} \\
& \text{parent(pam,bob).} \\
& \text{parent(tom,liz).} \\
& \text{parent(bob,ann).} \\
& \text{parent(bob,pat).} \\
& \text{parent(pat,jim).}
\end{align*}
\]

====================

? - parent(bob,pat).
yes
? - parent(liz,pat).
no
More Queries: Meaning?

\[
\text{parent}(\text{tom}, \text{bob}).
\]
\[
\text{parent}(\text{pam}, \text{bob}).
\]
\[
\text{parent}(\text{tom}, \text{liz}).
\]
\[
\text{parent}(\text{bob}, \text{ann}).
\]
\[
\text{parent}(\text{bob}, \text{pat}).
\]
\[
\text{parent}(\text{pat}, \text{jim}).
\]

====================

?- \text{parent}(\text{tom}, \text{ben}).
?- \text{parent}(\text{X}, \text{liz}).
?- \text{parent}(\text{bob}, \text{X}).
?- \text{parent}(\text{X}, \text{Y}).
Composite Queries – Explain!

parent(tom,bob).
parent(pam,bob).
parent(tom,liz).
parent(bob,ann).
parent(bob,pat).
parent(pat,jim).

--------------------

?- parent(Y,jim), parent(X,Y). % ??
?- parent(X,Y), parent(Y,jim). % ??
?- parent(tom,X), parent(X,Y). % ??
?- parent(X,ann), parent(X,pat). % ??
Defining Relations by Rules

- The offspring relation:
  For all X and Y,
  
  Y is an offspring of X if
  X is a parent of Y

- Translates to Prolog clause:
  
  \(\text{offspring}(Y,X) :- \text{parent}(X,Y).\)

- Alternative reading:
  For all X and Y,
  
  if X is a parent of Y,
  then Y is an offspring of X
Rules

- Rules like facts are clauses.
- A rule has a condition and a conclusion.
- The conclusion of a Prolog rule is its head.
- The condition of a Prolog rule is its body.
- If the condition of a rule is true, then it follows that its conclusion is true also.
How Prolog Rules are Used

• Prolog rules allow to compute new facts from old
• Given the offspring relation rule:
  
  ```prolog
test OFFSPRING (Y, X) :- parent (X, Y).
  ```

  – if (X,Y) is in the parent relation, then (Y,X) is in the offspring relation
• With this rule and goal:
  
  ```prolog
?- offspring (Y, X)
  ```

  – the goal succeeds if \textit{parent} (X, Y) succeeds
• Procedurally, when a goal matches the head of a rule, Prolog sets up its body as a new goal
Example

parent(tom,bob).
parent(pam,bob).
parent(tom,liz).
parent(bob,ann).
parent(bob,pat).
parent(pat,jim).

Offspring(Y,X) :- parent(X,Y).

?- offspring(liz,tom).
More Family Relations

parent(tom,bob). parent(pam,bob).
parent(tom,liz). parent(bob,ann).
parent(bob,pat). parent(pat,jim).

male(tom). male(bob). male(jim).
female(pam). female(liz). female(ann)

offspring(Y,X) :- parent(X,Y).

mother(X,Y) :- parent(X,Y),female(X).
father(X,Y) :- ??.
grandfather(X,Z) :- ??.
sister(X,Y) :- ??.
brother(X,Y) :- ??.

====================
Predecessor (Ancestor)?

predecessor(X,Z) :- parent(X,Z).
predecessor(X,Z) :-
    parent(X,Y),
    parent(Y,Z).
predecessor(X,Z) :-
    parent(X,Y1),
    parent(Y1,Y2),
    parent(Y2,Z).

etc. When do we stop? -- Use recursion!

predecessor(X,Z) :- parent(X,Z).
predecessor(X,Z) :- parent(X,Y), predecessor(Y,Z).

etc. Why does this work?
Declarative vs Procedural

- Prolog programs can be understood two ways: declaratively and procedurally.  
  \[ P : - Q, R \]

- **Declarative Way**
  - \( P \) is true if \( Q \) and \( R \) are true

- **Procedural Way**
  - To solve problem \( P \), first solve \( Q \) and then \( R \)  
  (or) To satisfy \( P \), first satisfy \( Q \) and then \( R \)
What is difference?

- Procedural way does not only define logical relation between the head of the clause and the goals in the body, but also the order in which the goal are processed.
• Relations & Functions
  – A relation from $A$ to $B$ is subset of $A \times B$
    (sounds like?)
  – A function from $A$ to $B$ is a relation $R$ from $A$ to $B$ admitting only many-to-one mappings
  – Functions are special cases of relations
  – Examples:
    \begin{align*}
    A &= \{a_1,a_2,a_3,a_4\} \quad B = \{b_1,b_2,b_3\} \\
    R &= \{(a_1,b_1),(a_2,b_2),(a_2,b_3),(a_4,b_3)\} \\
    F &= \{(a_1,b_1),(a_2,b_2),(a_3,b_3),(a_4,b_3)\}
    \end{align*}
Logical Basis: Propositions

1. Booleans \textit{true} and \textit{false} are propositions.

2. A \textit{symbol} is a proposition which has value \textit{true} or \textit{false}.

3. If \( p \) and \( q \) are propositions, then so are:
   
   a) \( \neg p \): negation of \( p \).
   
   b) \( p \land q \): conjunction of \( p \) and \( q \).
   
   c) \( p \lor q \): disjunction of \( p \) and \( q \), and
   
   d) \( P \implies q \): implication from \( p \) to \( q \).
Logical Basis: Predicates

1. Each predicate denotes a relation among constants.
2. Each predicate is defined as a set of terms. Only DS!
3. Terms are relation instances whose arguments can be constants or quantified variables (\( \forall \) or \( \exists \) quantified):

Examples:

\[
\begin{align*}
\text{parent}(\text{ted}, \text{derek}). \\
\text{parent}(\text{ted}, \text{dena}). \\
\text{parent}(\text{ted}, \text{dustin}). \\
\text{parent}(\text{derek}, \text{adams}). \\
\forall X, Y. (\text{parent}(X, Y) \Rightarrow \text{ancestor}(X, Y)). \\
\forall X, Y. (\exists Z. \text{parent}(X, Z) \land \text{ancestor}(Z, Y) \\
\quad \Rightarrow \text{ancestor}(X, Y)).
\end{align*}
\]
Logical Basis: Theorem Proving

- KB predicates serve as axioms
- Queries become theorems
  - A general theorem-prover (refutation proof) attempts to prove query as theorem of KB - in the process instantiating variables – serving to extract data from KB!
Logical Basis:  Theorem Proving

**Example**

parent(ted,derek).
parent(ted,dena).
parent(ted,dustin).
parent(derek,adams).

\[ \forall X,Y. (\text{parent}(X,Y) \Rightarrow \text{ancestor}(X,Y)). \]
\[ \forall X,Y. (\exists Z. \text{parent}(X,Z) \land \text{ancestor}(Z,Y) \Rightarrow \text{ancestor}(X,Y)). \]

=======================================

parent(ted, derek)?
\exists X. parent(ted,X)?
\exists X,Y. parent(X,Y)?
\exists X. ancestor(ted,X)?
Resolution Inference Rule

- Predicates serve as axioms
- Queries serve as theorems to prove
- **Resolution as the inference rule**
  \[ KB[p, p \Rightarrow q] |- q \]
- Proving theorems by **backward deduction**.
Horn Clauses

• Horn clauses are a subset of predicate logic
• Each Horn clause implication has form:
  \[ p_1, \ldots, p_n \Rightarrow q \]
  where \( n \geq 0 \)

• The **Resolution inference** rule is most efficient with Horn Clauses (proven sound and complete)
  – It can prove all theorems by backward deduction (completeness) and only the correct ones (soundness)!
Notation for Horn Clauses

Facts (universal): <fact> ::= <term>.
  parent(ted,derek)
  parent(ted,dena).
  parent(ted,dustin).
  parent(derek,adams).

Rules (∀ or ∃): <rule> ::= <term> :- <terms>.
  ancestor(X, Y) :- parent(X, Y).
  ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).

Queries (existential): <query> ::= ?- <terms>.
  ?- parent(ted, X).
  ?- ancestor(ted,X).
Clauses and Predicates

- A **clause** is a fact or a rule.
- A **literal** is a relation occurrence in a clause.
- A clause **head** is the literal before the ‘:-’ in a clause.
- A clause **body** consists of the literals after a ‘:-’ in a clause.
- A **predicate** is the set of clauses whose head has the same predicate symbol and arity (number of arguments).
An Example

parent(ted, derek) :- true.
parent(ted, dена) :- true.
parent(ted, dustin) :- true.
parent(derek, adams) :- true.

ancestor(X, Y) :- parent(X, Y).
ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y)
Unification

- Terns $T_1$ and $T_2$ unify if $\sigma(T_1) = \sigma(T_2)$ for some substitution $\sigma$.
- Substitution $\sigma$ is called a unifier of $T_1$ and $T_2$.

**Example:** $\sigma = \{H \rightarrow a, \ T \rightarrow b\}$

$\{H \rightarrow a, \ T \rightarrow b\} \ (f(a,T))$

$= \{H \rightarrow a, \ T \rightarrow b\} \ (f(H,b))$

$= f(a,b)$
Resolution

Let

\[ Q_i = a_1, a_2, \ldots, a_n \]

Be a query, and

\[ b : b_1, b_2, \ldots, b_m \]

Be a rule in the KB.

If \( \sigma \) is the most general unifier of \( a_1 \) and \( b \), and

\[ Q_{i+1} = \sigma (b_1, b_2, \ldots, b_m, a_2, \ldots, a_n) , \]

Then we say \( Q_i \) is reduced to \( Q_{i+1} \) via a resolution step.
An Example

The most general unifier for the query

\(?- \text{ancestor}(X, \text{adams}).\)

And the KB rule:

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

Is:

\[ \sigma = \{ Y \mapsto \text{adams} \} \]

Hence, the query after the resolution of these 2 is:

\[ \sigma(\text{parent}(X, Y)) = \text{parent}(X, \text{adams}) \]
Applications of Prolog

- Natural language process – DCG – more powerful that CFG
- Expert Systems
  - Expert System shells out there supporting diagnostics, planning, design etc
- Constraint-driven programming for solving certain types of problems:
  - Search-based problems (constraints prone the search space
  - Example Constraint type problems:
    - S.E.N.D. + M.O.R.E. = M.O.N.E.Y
    - \(d_1d_2\ldots d_9\), none of digits is zero, none is repeated and \(d_1\ldots d_i\) is divisible by \(i\)
Applications: Expert Systems

- An expert system is **software that attempts to reproduce the performance of one or more human experts**, 
- Use human knowledge to solve problems that normally would require human intelligence 
- Embody some non-algorithmic expertise (heuristics) 
- Represent the expertise knowledge as data or rules within the computer 
  - Can be called upon when needed to solve problems 

Categories 
- Diagnostic applications, servicing: 
  - People 
  - Machinery 
- Play chess 
- Make financial planning decisions 
- Configure computers 
- Monitor real time systems 
- Underwrite insurance policies 
- Perform many other services which previously required human expertise
Major Components
Major Roles of Individuals
Resources

- Wikipedia Page

- Building Expert Systems in Prolog

- Artificial Intelligence - A modern Approach by Stuart Russell, Peter Norvig

- PC AI – Expert Systems
  - Available at: [http://www.pcai.com/web/ai_info/expert_systems.html](http://www.pcai.com/web/ai_info/expert_systems.html)

- VisiRule Web Demos (A simple example): [http://www.lpa.co.uk/vrs_dem.htm](http://www.lpa.co.uk/vrs_dem.htm)
Thank You
For
Your attention