Reading: Sethi, Chapter 11.

- Overview
- Predicate Calculus
- Substitution and Unification
- Introduction to Prolog
- Prolog Inference Rules
- Programming in Prolog
  - Recursion
  - List Processing
  - Arithmetic
  - Higher-order programming
  - Miscellaneous functions
- Conclusion
Prolog

Programming in Logic

• Idea emerged in early 1970’s; most work done at Univ. of Edinburgh.

• Based on a subset of first-order logic.
  – Feed it theorems and pose queries, system does the rest.

• main uses:
  – Originally, mainly for natural language processing.
  – Now finding uses in database systems and even rapid prototyping systems of industrial software.

• Popular languages: Prolog, XSB, LDL, Coral, Datalog, SQL.
Logic Programming Framework

Query:
Is \( q(X_1, \ldots, X_N) \) true?

Programming Environment

Knowledge Base:

\textit{Facts & Rules}

Proof Procedure

Answer:
“\textit{Yes}/\textit{No}”
& variable bindings
Declarative Languages

In its purest form, Logic programming is an example of *declarative programming*.

Popular in database systems and artificial intelligence.

Declarative specifications: Specify what you want, but not how to compute it.

Example. Find $x$ and $y$ such that

\[
\begin{align*}
3x + 2y &= 1 \\
x - y &= 4
\end{align*}
\]

A method (program) for solving these is how to get values for $x$ and $y$. But all we gave was a *specification*, or *declaration* of what we want. Hence the name.
Examples

• "Retrieve the telephone number of the person whose name is Tom Smith" (easy)

• "Retrieve the telephone number of the person whose address is 13 Black St" (hard)

• "Retrieve the name of the person whose telephone number is 123-3445" (hard)

Each command specifies \textit{what} we want but not \textit{how} to get the answer. A database system would use a different algorithm for each of these cases.

Can also return multiple answers:
• "Retrieve the names of \textit{all} people who live on Oak St."
Algorithm = Logic + Control

- Users specify “logic” — what the algorithm does — using logical rules and facts.

- “Control” — how the algorithm is to be implemented — is built into Prolog.

i.e., Search procedures are built into Prolog. They apply logical rules in a particular order to answer user questions.

**Example.** P if Q₁ and Q₂ and ... and Qₖ can be read as

to deduce P:
  
deduce Q₁
  
deduce Q₂
  
...  
deduce Qₖ

Users specify what they want using classical first-order logic (predicate calculus).
Classical First-Order Logic

- The simplest kind of logical statement is an atomic formula. *e.g.*,
  
  man(tom) (tom is a man)
  
  woman(mary) (mary is a woman)
  
  married(tom, mary)
  (tom and mary are married)

- More complex formulas can be built up using logical connectives: $\land$, $\lor$, $\sim$, $\forall X$, $\exists X$. *e.g.*,
  
  smart(tom) $\lor$ dumb(tom)
  
  smart(tom) $\lor$ tall(tom)
  
  $\sim$ dumb(tom)
  
  $\exists X$ married(tom, X)
  (tom is married to something)
  
  $\forall X$ loves(tom, X)
  (tom loves everything)
  
  $\exists X$ [married(tom, X) $\land$ female(X) $\land$ human(X)]
  (tom is married to a human female)
Logical Implication

rich(tom) \lor \sim\text{smart}(tom)

This implies that if tom is smart, then he must be rich. So, we often write this as

rich(tom) \leftarrow \text{smart}(tom)

In general, \( P \leftarrow Q \) and \( Q \to P \) are abbreviations for \( P \lor \sim Q \).

For example,

\[ \forall X [ (\text{person}(X) \land \text{smart}(X)) \to \text{rich}(X)] \]

(every person who is smart is also rich)

\[ \exists X \ \text{mother}(john,X) \]

(john has a mother)

\[ \exists X [\text{mother}(john,X) \land \forall Y \text{mother}(john,Y) \to Y = X] \]

(john has exactly one mother)
Horn Rules

Logic programming is based on formulas called Horn rules. These have the form

\[ \forall x_1 \ldots x_k \ [A \leftarrow B_1 \land B_2 \ldots \land B_j] \]

where \( k, j \geq 0 \).

For example,

\[ \forall x, y \ [A(x) \leftarrow B(x, y) \land C(y)] \]
\[ \forall x \ [A(x) \leftarrow B(x)] \]
\[ \forall x \ [A(x, d) \leftarrow B(x, e)] \]
\[ A(c, d) \leftarrow B(d, e) \]
\[ \forall x \ A(x) \]
\[ \forall x \ A(x, d) \]
\[ A(c, d) \]

Note that atomic formulas are also Horn rules, often called facts.

A set of Horn rules is called a Logic Program.
**Logical Inference with Horn Rules**

Logic Programming is based on a simple idea: From rules and facts derive more facts

---

**Example 1.** Given the facts \( A, B, C, D \), and these rules:

\[
\begin{align*}
(1) & \quad E \leftarrow A \land B \\
(2) & \quad F \leftarrow C \land D \\
(3) & \quad G \leftarrow E \land F
\end{align*}
\]

From (1), derive \( E \)
From (2), derive \( F \)
From (3), derive \( G \)

---

**Example 2.** Given these facts:

\[
\begin{align*}
\text{man(plato)} & \quad (\text{“plato is a man”}) \\
\text{man(socrates)} & \quad (\text{“socrates is a man”})
\end{align*}
\]

and this rule:

\[
\forall X \ [\text{man}(X) \rightarrow \text{mortal}(X)]
\]

(“all men are mortal”)

derive: \( \text{mortal(plato)}, \ \text{mortal(socrates)} \).
Recursive Inference

Example.

Given:

\[ \forall X \ [\text{mortal}(X) \rightarrow \text{mortal(son_of}(X))] \]
\[ \text{mortal}(\text{plato}) \]

Derive:

\[ \text{mortal}(\text{son_of}(\text{plato})) \]
  (using \( X = \text{plato} \))

\[ \text{mortal}(\text{son_of}(\text{son_of}(\text{plato}))) \]
  (using \( X = \text{son_of}(\text{plato}) \))

\[ \text{mortal}(\text{son_of}(\text{son_of}(\text{son_of}(\text{plato})))) \]
  (using \( X = \text{son_of}(\text{son_of}(\text{plato})) \))

...

This kind of inference simulates recursive programs (as we shall see).
Logic Programming

Horn rules correspond to programs, and a form of Horn inference corresponds to execution.

For example, consider the following rule:

$$\forall X, Y \ p(X) \leftarrow q(X,Y) \land r(X,Y) \land s(X,Y)$$

Later, we shall see that this rule can be interpreted as a program, where

- $p$ is the program name,
- $q, r, s$ are subroutine names,
- $X$ is a parameter of the program, and
- $Y$ is a local variable.
Non-Horn Formulas

The following formulas are *not* Horn:

\[ A \rightarrow \sim B \]

\[ A \lor B \]

\[ A \lor B \leftarrow C \]

\[ \exists X \ [A(X) \leftarrow B(X)] \]

\[ A \leftarrow (B \leftarrow C) \]

\[ \forall X \ [\text{flag}(X) \rightarrow [\text{red}(X) \lor \text{white}(X)]] \]

("every flag is red or white")

\[ \forall X \ \exists Y \ [\text{wife}(X) \rightarrow \text{married}(X,Y)] \]

("every wife is married to someone")
Non-Horn Inference

Inference with non-Horn formulas is more complex than with Horn rules alone.

**Example.**

\[
\begin{align*}
A & \leftarrow B \\
A & \leftarrow C \\
B \lor C & \quad \text{(non-Horn)}
\end{align*}
\]

We can infer \( A \), but must do case analysis:

- either \( B \) or \( C \) is true.
  - if \( B \) then \( A \)
  - if \( C \) then \( A \)

Therefore, \( A \) is true in all cases.

Non-Horn formulas do not correspond to programs, and non-Horn inference does not correspond to execution.
Logical Equivalence

Many non-Horn formulas can be put into Horn form using two methods:

(1) logical equivalence
(2) skolemization

Example 1. Logical Equivalence.
\[
\sim A \leftarrow \sim B \equiv \sim A \lor \sim(\sim B) \\
\equiv \sim A \lor B \\
\equiv B \lor \sim A \\
\text{(Horn)} \quad \equiv B \leftarrow A
\]

Logical Laws:
\[
\sim \sim A \equiv A \\
\sim (A \lor B) \equiv \sim A \land \sim B \\
A \lor (B \land C) \equiv (A \lor B) \land (A \lor C) \\
A \leftarrow B \equiv A \lor \sim B
\]

Example 2. Logical Equivalence.
\[
A \leftarrow (B \lor C) \equiv A \lor \sim(B \lor C) \\
\equiv A \lor (\sim B \land \sim C) \\
\equiv (A \lor \sim B) \land (A \lor \sim C) \\
\text{(Horn)} \quad \equiv (A \leftarrow B) \land (A \leftarrow C)
\]
Example 3. Logical Equivalence.

\[
A \leftarrow (B \leftarrow C) \equiv A \lor \sim (B \leftarrow C) \\
\equiv A \lor \sim (B \lor \sim C) \\
\equiv A \lor (\sim B \land \sim \sim C) \\
\equiv A \lor (\sim B \land C) \\
\equiv (A \lor \sim B) \land (A \lor C)
\]
(non Horn) \equiv (A \leftarrow B) \land (A \lor C)

In general, rules of the following form cannot be converted into Horn form:

\[
\forall x[(A_1 \lor \ldots \lor A_n) \leftarrow (B_1 \land \ldots \land B_m)]
\]

For example,

\[
(A \lor B) \leftarrow (C \land D) \\
(A \lor B) \leftarrow C \\
(A \lor B) \\
\forall x \ [A(x) \lor B(x)] \leftarrow [C(x) \land D(x)]
\]

i.e., if it is possible to infer a non-trivial disjunction from a set of formulas, then the set is inherently non-Horn.

(A rule like \( p \lor q \leftarrow q \) infers a trivial disjunction, since the rule is a logical tautology. Such rules can simply be ignored.)
Skolemization

Non-Horn formulas like $\exists x \ A(x)$ can be converted to Horn form.

Example 1.

Replace (1) $\exists x \ \text{mother}(\text{john},x)$ \hspace{1cm} (non-Horn)
\hspace{1cm} with (2) $\text{mother}(\text{john},m)$ \hspace{1cm} (Horn)

Here, $m$ is a new constant symbol, called a skolem constant, that stands for the (unknown) mother of john.

Note: (1) $\not\equiv$ (2), but they say (almost) the same thing. In particular, (1) can sometimes be replaced by (2) during inference, as we shall see.
Example 2. A non-Horn formula:

(3)  \( \forall x \ [\text{person}(x) \rightarrow \exists y \ \text{mother}(x,y)] \)
     ("every person has a mother")

Let \( m(x) \) stand for the (unknown) mother of \( x \). Then, we can replace (3) by a Horn rule:

(4)  \( \forall x \ [\text{person}(x) \rightarrow \text{mother}(x,m(x)) \])

\( m(x) \) is called a **skolem function**.

It is an artificial name we have created.

*e.g.*, \( m(\text{mary}) \) denotes the mother of mary.
\( m(\text{tom}) \) denotes the mother of tom.
\( m(\text{jfk}) \) denotes the mother of jfk.

So, we only need \( \exists x \) because we don’t have a *name* for \( x \). By creating artificial names (skolem symbols), we can eliminate many \( \exists \)'s, and convert many formulas to Horn rules, which Prolog can then use.

Skolemization is a technical device for doing inference.
Inference with Skolemization

(1) $\forall X \ [\text{man}(X) \rightarrow \text{person}(X)]$
   ("every man is a person")

(2) $\forall X \ \exists Y \ [\text{person}(X) \rightarrow \text{mother}(X,Y)]$
   ("every person has a mother"—non Horn)

(3) $\forall X,Y \ [\text{mother}(X,Y) \rightarrow \text{loves}(Y,X)]$
   ("every mother loves her children")

(4) $\text{man}(\text{plato})$  ("plato is a man")

**Question.** $\exists Y \ \text{loves}(Y,\text{plato})$
   ("does someone love plato?")

**Step 1.** Skolemize (2) to get a Horn rule:
   (2') $\forall X \ [\text{person}(X) \rightarrow \text{mother}(X, \text{m}(X))]$

**Step 2.** Use Horn inference:

- $\text{person}(\text{plato})$ from (1)
- $\text{mother}(\text{plato}, \text{m}(\text{plato}))$ from (2')
- $\text{loves}(\text{m}(\text{plato}), \text{plato})$ from (3)

**Thus.** $\exists Y \ \text{loves}(Y,\text{plato})$

_i.e._, $Y = \text{m}(\text{plato}).$ So, answer is YES.
Skolem Dependencies

(1)  \( \exists x \, \forall y \, p(x, y) \)
    skolemizes to  \( \forall y \, p(a, y) \),
    where \( a \) is a skolem constant.

(2)  \( \forall y \, \exists x \, p(x, y) \)
    skolemizes to  \( \forall y \, p(b(y), y) \),
    where \( b \) is a skolem function.

    i.e., in (2), \( x \) depends on \( y \).
    But in (1), \( x \) is independent of \( y \).

(3)  \( \forall x \, \forall y \, \exists z \, q(x, y, z) \)
    skolemizes to  \( \forall x \, \forall y \, q(x, y, c(x, y)) \),
    where \( c \) is a skolem function of both \( x \) and \( y \).

    i.e., in (3), \( z \) depends on both \( x \) and \( y \).
Skolem Dependencies — Concrete Examples

\[ \exists X \ \forall Y \ \text{loves}(X,Y) \quad (\text{“someone loves everybody”}) \]
\[ \Rightarrow \ \forall Y \ \text{loves}(p,Y) \quad (\text{“p loves everybody”}) \]

\[ \forall X \ \exists Y \ \text{mother}(X,Y) \quad (\text{“everyone has a mother”}) \]
\[ \Rightarrow \ \forall X \ \text{mother}(X,m(X)) \]
\[ \quad (\text{“m}(X) \text{ is the mother of } X”) \]

\[ \forall X \ \forall Y \ \exists Z \ \text{owns}(X,Y) \ \rightarrow \ \text{document}(Z,X,Y) \]
\[ \quad (\text{“if } X \text{ owns } Y, \text{ then there is a document, } Z, \text{ saying that } X \text{ owns } Y”) \]
\[ \Rightarrow \ \forall X \ \forall Y \ \text{owns}(X,Y) \ \rightarrow \ \text{document}(\text{d}(X,Y),X,Y) \]
\[ \quad (\text{“d}(X,Y) \text{ is a document saying that } X \text{ owns } Y”) \]