Prologue

Goal: Convert you to the way of Prolog, especially you Lisp heathens (or to introduce you to Prolog and its many merits).

I. The Genesis of Prolog

II. Prolog, the language

III. The Merits of Prolog

IV. Why Prolog?
   A. Prolog vs. Lisp
   B. Why Prolog for AI?

V. The Cannibals-and-Missionaries Problem

VI. Homework
Theorem Proving

\[
\neg a \lor \neg b \\
\neg a \lor \neg c \\
\neg \phi \lor \neg d \lor \neg e \\
\neg c \lor \neg f \lor \neg g \\
\]

Prove \( h \).

Search can be hard. Theorem proving can be hard.

---

A Horn clause has one or no positive atoms in it.

\[
a \lor \neg b \lor \neg c
\]

can be rewritten as

\[
a \leftarrow b, c.
\]

---

Procedural = Declarative

Logic can be used as a programming language!

---

Prolog, the language

1. Clauses, Facts, and Queries

    Clause: \( a \leftarrow \phi_1, \ldots, \phi_n \).

    Fact: \( a \).

    Query: \( \leftarrow a_1, \ldots, a_n \).

2. Matching (unification)

3. Built-in control

   - Proof by refutation
   - One inference rule: resolution.

   - Choosing clauses: first in list to match; to last in list to match
   - Choosing goals: from left-to-right in goal list

4. Meta-predicates

    setof \quad \text{clause} \quad \text{var}
    \text{assert} \quad \text{retract} \quad \text{not} \quad \text{"\rightarrow\"}
    \text{univ} \quad \text{".\"} \quad \text{equivalent} \quad \text{"==\"} \quad \text{meta-variables!}

5. Search Pruning/Commit

    cut \quad \text{\"!\"}
Grandmothers and Grandfathers

\[ \text{grandmother (CM, X)} \leftarrow \text{mother (CM, P)}, \]
\[ \quad \text{parent (P, X)}. \]
\[ \text{grandfather (GF, X)} \leftarrow \text{father (GF, P)}, \]
\[ \quad \text{parent (P, X)}. \]
\[ \text{parent (M, X)} \leftarrow \text{mother (M, X)}. \]
\[ \text{parent (F, X)} \leftarrow \text{father (F, X)}. \]

\[ \text{mother (julie, parke)}. \quad \text{father (ben, parke)}. \]
\[ \text{mother (ruby, judith)}. \quad \text{father (alvin, judith)}. \]
\[ \text{mother (lallage, blan)}. \quad \text{father (albert, blan)}. \]
\[ + \text{grandmother (G, parke)}. \quad + \text{grandmother (lallage, X)}. \]
\[ G = \text{ruby}; \]
\[ X = \text{parke}; \]
\[ G = \text{lallage}; \]
\[ \text{no} \]

Why Prolog?

**Prolog vs. Lisp** (a sibling rivalry)
- the not-invented-here syndrome
- relational vs. functional

---

Why Prolog for AI?

- easy to write meta-programs
  - Prolog is its own meta-language!
  - code = data
- is an "interpreted" language
  - good debugging facilities
  - needed for meta-programming
- based on the recursive paradigm
- no typing!
- Prolog is based on first-order logic
  - Logic is good for AI.
- is declarative
  (not prescriptive)
The Merits of Prolog
Neat Features of Prolog

● Non-determinism (backtracking)
  – Can find alternate answers/solutions for free!

● Invertability
  – Call any predicate with any instantiation pattern.
    (Well, sometimes …)

● Unification
  – Pattern matching for free!

● Built-in Search
  – A free refutation proof system.
  – Specs are executable. (Well, kind of …)
    Do not have to write one’s own search mechanism for every problem.

● Built-in database features
  – `assert` and `retract`

Meta-Predicates
a.k.a. Extra-Logical Predicates

setof/findall

\[
\begin{align*}
\text{setof} & \quad \langle \text{GM, grandmother (GM, parke), GMs}. \\
\text{GMs} & \quad = \quad \text{[lallah, ruby]}. \\
& \quad \text{no}
\end{align*}
\]

\[\begin{align*}
\text{assert} & \quad \langle \text{student (X)}. \\
\text{no} & \quad \langle \text{assert (student (parke))}. \\
\text{yes} & \quad \langle \text{student (X)}. \\
X & \quad = \quad \text{parke}; \\
\text{no} & \quad \langle \text{meta-variables} \\
\text{exec_list} & \quad \langle \langle X, Xs \rangle \rangle \quad \leftarrow \quad X, \text{exec_list} (Xs). \\
\text{exec_list} & \quad \langle [] \rangle.
\end{align*}\]
Executable Specifications

Program = Logic + Control

A goal of logic programming is to be able to execute specifications as code.

In Prolog, the control mechanism is built in.

Problem with Specs

Some specs are more equal than others.

\[
\begin{align*}
\text{sort} &\ (As, Zs) \leftarrow \text{sameLength} (As, Zs), \\
&\ \quad \quad \text{perm} (As, Zs), \\
&\ \quad \quad \text{ordered} (Zs), \\
\text{perm} &\ (As, [A|Zs]) \leftarrow \text{choose} (A, As, Rest), \\
&\ \quad \quad \text{perm} (Rest, Zs), \\
\text{perm} &\ ([], []). \\
\text{sameLength} &\ ([|As], [|Zs]) \leftarrow \text{sameLength} (As, Zs), \\
&\ \quad \quad \text{sameLength} ([|], [|]). \\
\text{choose} &\ (A, [A|As], As), \\
\text{choose} &\ (A, [B|As], [B|Zs]) \leftarrow \text{choose} (A, As, Zs), \\
\text{ordered} &\ ([A, B|As]) \leftarrow A < B, \text{ordered} ([|B|As]), \\
\text{ordered} &\ ([|]), \\
\text{ordered} &\ ([]).
\end{align*}
\]
Problem with Specs [cont.]

A better sort of sort.

\[\text{sort} ([A|As], Zs) \leftarrow \text{divide_list} (A, As, Fs, Ls),\]
\[\text{sort} (Fs, \text{OrdFs}),\]
\[\text{sort} (Ls, \text{OrdLs}),\]
\[\text{append} (\text{OrdFs}, [A|\text{OrdLs}], Zs).\]

\[\text{sort} ([], []).\]

\[\text{divide_list} (A, [F|As], [F|Fs], Ls) \leftarrow\]
\[A \triangleright F,\]
\[\text{divide_list} (A, As, Fs, Ls).\]

\[\text{divide_list} (A, [L|As], Fs, [L|Ls]) \leftarrow\]
\[A \triangleright= L,\]
\[\text{divide_list} (A, As, Fs, Ls).\]

\[\text{divide_list} (A, [ ], [], [ ]).\]

Pragmatics

\[\vdash \text{is not}\]
\[, \text{is and}\]
\[; \text{is or} \quad (\text{also used to enumerate answers})\]
\[! \text{is cut}\]
\[:- \text{is if} \quad (\leftarrow )\]

[Head|Tail] is a list.

Head is the first term in list. (car for you Lispers)
Tail is the first term in list. (cdr for you Lispers)
[First, Second|Tail] is valid notation too. [] is the empty list.
[First, Second, Third] is a completely enumerated list.

Variables names always start CAPITALIZED.
Constants begin with lower case (or are single quoted).

How do you load clauses from a file?

In the Prolog session, type: consult ((filename)).

Every clause (rule, query, or fact) must end in a period!
Books on Prolog

Prolog Books (On reserve in AVW Library)


Manuals

The SICSTUS Manual.

Logic for Problem Solving


Books on Logic Programming