Prologue to Prolog 101
A Lecture for COSC-6421

Parke Godfrey
Fall 2004
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

\[-a \lor b \quad -b \lor -f \lor h\]
\[-a \lor c \quad -c \lor -d \lor h\]
\[-b \lor d \lor e \quad -e \lor -g \lor h\]
\[-c \lor f \lor g \quad a\]

Prove \(h\).

Search can be hard. Theorem proving can be hard.

---

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

\[a \lor -b \lor -c\]

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: \( a \leftarrow b_1, \ldots, b_n \).  
   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 clause var  
   assert retract not “\+”  
   univ “=..” equivalent “==” meta-variables!

5. Search Prunning/Commit

   cut “!”
Grandmothers and Grandfathers

\[
\text{grandmother}(GM, X) \leftarrow \text{mother}(GM, P), \\
\quad \text{parent}(P, X). \\
\]
\[
\text{grandfather}(GF, X) \leftarrow \text{father}(GF, P), \\
\quad \text{parent}(P, X). \\
\]
\[
\text{parent}(M, X) \leftarrow \text{mother}(M, X). \\
\text{parent}(F, X) \leftarrow \text{father}(F, X). \\
\]
\[
\text{mother}(julie, parke). \\
\text{father}(blan, parke). \\
\text{mother}(ruby, judith). \\
\text{father}(alvin, judith). \\
\text{mother}(lallage, blan). \\
\text{father}(albert, blan). \\
\]
\[
\leftarrow \text{grandmother}(G, parke). \\
\leftarrow \text{grandmother}(lallage, X). \\
\]
\[
G = ruby; \quad X = parke; \\
G = lallage; \quad no \\
no
\]
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 recursion 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

← setof (GM, grandmother (GM, parke), GMs).

GMs = [lallage, ruby];

no

assert

← student (X).

no

← assert (student (parke)).

yes

← student (X).

X = parke;

no

meta-variables

exec_list ([X|Xs]) ← X, exec_list (Xs).

exec_list ([[]]).
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.

\[\text{sort} \ (As, \ Zs) \leftarrow \text{same\_length} \ (As, \ Zs),\]
\[\quad \text{perm} \ (As, \ Zs),\]
\[\quad \text{ordered} \ (Zs).\]

\[\text{perm} \ (As, \ [A \mid Zs]) \leftarrow \text{choose} \ (A, \ As, \ Rest),\]
\[\quad \text{perm} \ (Rest, \ Zs).\]
\[\text{perm} \ ([], \ []).\]

\[\text{same\_length} \ ([\_ \mid As], \ [\_ \mid Zs]) \leftarrow \text{same\_length} \ (As, \ Zs).\]
\[\text{same\_length} \ ([], \ []).\]

\[\text{choose} \ (A, \ [A \mid As], \ As).\]
\[\text{choose} \ (A, \ [B \mid As], \ [B \mid Zs]) \leftarrow \text{choose} \ (A, \ As, \ Zs).\]

\[\text{ordered} \ ([A, \ B \mid As]) \leftarrow A < B, \ \text{ordered} \ ([B \mid As]).\]
\[\text{ordered} \ ([A]).\]
\[\text{ordered} \ ([\_]).\]
Problem with Specs [cont.]

A better sort of sort.

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

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

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

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

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

\+ is \textit{not}
,

\textit{and}
;

\textit{or} \quad \text{(also used to enumerate answers)}
!

\textit{cut}
:- \quad \textit{if} \quad (\leftarrow )

[\textit{Head} | \textit{Tail}] \text{ is a list.}

\textit{Head} \text{ is the first term in list.} \quad (\textit{car} \text{ for you Lispites})
\textit{Tail} \text{ is the first term in list.} \quad (\textit{cdr} \text{ for you Lispites})

[\textit{First}, \textit{Second} | \textit{Tail}] \text{ is valid notation too.} \quad [] \text{ is the empty list.}

[\textit{First}, \textit{Second}, \textit{Third}] \text{ is a completely enumerated list.}

Variables names always start \textsc{capitalized}.

Constants begin with lowercase (or are single-quoted).

---

How do you load clauses from a file?

In the Prolog session, type: \textit{consult} ((\textit{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
