Prolog

- Logic programming (declarative)
  - Goals and subgoals
- Prolog Syntax
- Database example
  - rule order, subgoal order, argument invertibility, backtracking model of execution, negation by failure, variables
- Data structures (lists, trees)
  - Recursive Functions: append, member
  - Lazy evaluation, terms as trees, Prolog search trees
- Goal-oriented semantics

Intro to Logic Programming

- Specifies relations between objects
  larger (2, 1), father (tom, jane)
- Separates control in PL from description of desired outcome
  father(X, jane) :- male(X), parent(X, jane).
- Computation engine: theorem proving and recursion
  - Higher-level PL than imperative languages
  - More accessible to non-technical people
Horn Clauses

- Conjunct of 0 or more conditions which are atomic formulae in predicate logic (constants, predicates, functions)
  \[ h_1 \land h_2 \land \ldots \land h_n \rightarrow c \]
- Means \( c \) if \( h_1, h_2, \ldots, h_n \) are all true
- Can have variables in the \( h_i \)'s or \( c \)
  \[ c(x_1,x_2,\ldots,x_m) \text{ if } h(x_1,x_2,\ldots,x_m,y_1,\ldots,y_k) \]
  means for all objects \( x_1, x_2, \ldots, x_m \), \( c \) holds if there are objects \( y_1, \ldots, y_k \) such that \( h \) holds.
  
  \[ \text{father}(X, \text{jane}) :- \text{male}(X), \text{parents}(X, Y, \text{jane}) \]

Logic Programming

- Goal-oriented semantics
  - goal is true for those values of variables which make each of the subgoals true
    - \( \text{father}(X, \text{jane}) \) will be true if \( \text{male}(X) \) and \( \text{parents}(X, Y, \text{jane}) \) are true with specific values for \( X \) and \( Y \)
  - recursively apply this reasoning until reach rules that are facts.
  - called \textit{backwards chaining}
Logic Programming

• Nondeterminism
  - Choice of rule to expand subgoal by
  - Choice of subgoal to explore first

    father(X,jane):- male(X), parents(X, Y, jane).
    father (X,jane):- father (X,Y), brother(Y, jane).

which rule to use first? which subgoal to explore first?
- Prolog tries rules in sequential order and proves subgoals from left to right. - Deterministic!

    father(X, jane) :- male(X), parent(X, jane).

Victoria Database Program

male(albert).  
male(edward).  
female(alice).  
female(victoria).  
parents(edward,victoria,albert).  
parents(alice,victoria,albert).
?- male(albert).
  yes
?- male(alice).
  false
?- female(X).
  X = alice ;
  X = victoria.

By responding <cr> you quit the query ; <cr> you continue to find another variable binding that makes the query true, if possible.
Victoria Example

- Problem: facts alone do not make interesting programs possible. Need variables and deductive rules.

```
?-female(X).          ; a query or proposed fact
X = alice ;          ; asks for more answers
X = victoria.        ; if user types <cr> then no more answers given when no more satisfying answers are left.
```

Variable X has been unified to all possible values that make female(X) true.
- Performed by pattern match search

---

Prolog Syntax in EBNF

```
<term> → <integer> | <atom> | <variable> |
        <functor> (<term> {, <term>} )

<rule> → <predicate> (<term> {,<term>} ):-
        <term> {, <term>} . | <fact>

<fact> → <functor> (<term>) {, <term>} ).

<query> → ?- <functor>(<term>{,<term>}).
```

[a proposed fact that must be proven]
Prolog Syntax, cont.

- Prolog program consists of facts, rules, and queries
- A query is a proposed fact, needing to be proven
  - If query has no variables and is provable, answer is yes
  - If query has variables, the proof process causes some variables to be bound to values which are reported (called a substitution)
- Variable names are capitalized, predicate names and constants are lower case.
- Names (e.g., predicates, functors with terms, clauses) come from first order logic.

Victoria Example, cont.

sister_of(X,Y) :- female(X),parents(X,M,F),
          parents(Y,M,F).
?- sister_of(alice, Y).
Y = edward
?- sister_of(X,Y),
X = alice
Y = edward ;
X = Y, Y = alice ;
false

3. female(alice).
4. female(victoria).
5. parents(edward,victoria,albert).
6. parents(alice,victoria,albert).
first answer from 3.+6.+5.
second answer from 3.+6.+6.

Subgoal order, argument invertibility, backtracking, rule order
Victoria Example, cont.

\[
sis(X,Y) :- \text{female}(X), \text{parents}(X,M,F), \text{parents}(Y,M,F), \neg(X==Y).
\]

?- sis(X,Y).
\[X = \text{alice}\]
\[Y = \text{edward} ;\]
\[false\]

\(\neg(P)\) succeeds when \(P\) fails;
called negation by failure

Negation by Failure

\[
\text{not}(X) :- X, !, \text{fail}.
\]
\[
\text{not}(\_ ) .
\]

if \(X\) succeeds in first rule, then the goal fails because of the last term.

if we type “;” the cut (!) will prevent us from backtracking over it or trying the second rule so there is no way to undue the fail.

if \(X\) fails in the first rule, then the goal fails because subgoal \(X\) fails. the system tries the second rule which succeeds, since “\(\_\)” (don’t care variable) unifies with anything.
Lists

- list  head  tail
- [a, b, c]  a  [b, c]
- [a, [b, c], d]  a  [ [b, c], d]
- [X | Y]  X  Y

A list consists of a sequence of terms

Unifying Lists

- [X, Y, Z] = [john, likes, fish]
  - X = john, Y = likes, Z = fish

- [cat] = [X | Y]
  - X = cat, Y = []

- [1 | 2] versus [1, 2]
  - 1 2
  - 1 2 [ ]
Lists

• Sequence of elements separated by commas, or
• \[ \text{first element} \mid \text{rest of list} \]
  • Like Scheme notation [\text{car}(	ext{list}) \mid \text{cdr}(	ext{list})]
• \[ [\text{the} \mid Y] \mid Z \] = [\[X, \text{hare}\] \mid [\text{is, here}]]

Lists

\[ [X, \text{abc}, Y] =? [X, \text{abc} \mid Y] \]
there is no value binding for Y, to make these two trees isomorphic.

Be careful of unifications that mix list syntax modes.
Lists

\[ a, b \mid Z \] =? \[ X \mid Y \]
\[ X = a, \ Y = [b \mid Z], \ Z = _ \]
look at the trees to see why this works!
\[ a, b, c \] = \[ X \mid Y \]
\[ X = a, \ Y = [b, c] \]

Member_of Function

\begin{verbatim}
member(A, [A | B]).
member(A, [B | C]) :- member(A, C).
\end{verbatim}

goal-oriented semantics: can get value assignment for goal member(A, [B|C]) by showing truth of subgoal member(A, C) and retaining value bindings of the variables
Example

?- member(a, [a, b]).
   yes
?- member(a, [b, c]).
   false
?- member(X, [a, b, c]).
   X = a ;
   X = b ;
   X = c ;
   false

Try this last query with trace.

Example

?- member(a, [b, c, X]).
   X = a ;
   false
?- member(X, Y).
   X = _123
   Y = [ X / _124 ] ;
   X = _123
   Y = [ _125, X / _126 ] ;
   X = _123
   Y = [ _127, _128, X / _129 ]

Lazy evaluation of a priori unbounded list structure. Unbound X variable is first element, then second element, then third element, in a sequence of generated lists of increasing length.
1. member(A, [A | B]).
2. member(A, [B | C]) :- member(A, C).

?- member(X, [a,b,c]).
match rule 1. member(A, [A | B]) so X = A = a, B = [b,c]  
   X = a ;
match rule 2. member(A, [B | C]) so X = A, B = a, C = [b,c]  
   then evaluate subgoal member(X, [b,c])  
   match rule 1. member(A', [A' | B']) so X = b, B' = [c]   
   X = b ;
match rule 2. member(A', [B' | C']) so X = A', B' = b, C' = [c]   
   then evaluate subgoal member(X, [c])  
   match rule 1. member(A'', [A'' | B'']) so X = A'', B'' = [ ]   
   X = c ;
match rule 2. member(A'', [B'' | C'']) so X = A'', B'' = c, C'' = [ ],  
   but member(X, [ ]) is unsatisfiable, no
Another Search Tree

member(a, [b,c,X])

fail, a can’t unify with b

member(a,[c, X])

fail, a can’t unify with c

member(a,[ X]).

X=a, B”=[ ]

success

fail, can’t unify [ ] with a list

fail, ditto

1. member(A,[A | B]).
2. member(A,[B | C]) :- member (A,C).

Prolog Search Trees

• A formalism to consider all possible computation paths
  • Leaves are success nodes or failures where computation cannot proceed
  • To model Prolog, leftmost subgoal is tried first
  • Label edges with variable bindings that occur by unification

- There can be infinite branches in the tree, representing non-terminating computations (performed lazily (i.e., generate as needed) by Prolog):
**Another Member_of Function**

Equivalent set of rules:

\[
\text{mem}(A, [A|\_]) .
\]
\[
\text{mem}(A, [\_|C]) :- \text{mem}(A,C).
\]

Can examine search tree and see the variables which have been excised were auxiliary variables in the clauses.

---

**Append Function**

append ([ ],A,A).

append([A|B],C,[A|D]) :- append(B,C,D).

• **Build a list**
  
  ?- append([a],[b],Y).
  \[ Y = [a,b] \]

• **Break a list into constituent parts**

  ?- append(X,[b],[a,b]).
  \[ X = [a] \]

  ?- append([a],Y,[a,b]).
  \[ Y = [b] \]
More Append

?- append(X, Y, [a, b]).
X = []
Y = [ a, b ] ;
X = [a]
Y = [b] ;
X = [a,b]
Y = [] ;
false

append ([ ], A, A).
append([A|B], C, [A|D]):- append(B, C, D).

Still More Append

• Generating an unbounded number of lists

?- append(X, [b], Y).
X = []
Y = [b] ;
X = [ _169]
Y = [ _169, b] ;
X = [ _169, _170 ]
Y = [ _169, _170, b] ;
etc.

append ([ ], A, A).
append([A|B], C, [A|D]):- append(B, C, D).
Common Beginner’s Errors

• Compile-time
  - Forget ending “.”
  - Misspelled functors
  - Need to override precedences with (..)

• Runtime
  - Infinite loops - check your recursion
  - Variables instantiated with unexpected values
  - Circular definitions
  - Giving wrong numbers of arguments to a clause