More Prolog
A Simplified Simpsons (Simplisons?)

mom(marge).
dad(homer).

parent(homer, bart).
parent(marge, bart).
parent(homer, lisa).
parent(marge, lisa).
parent(homer, maggie).
parent(marge, maggie).
A Simplified Simpsons (Simplisons?)

mom(marge).
dad(homer).

parent(homer, bart).
parent(marge, bart).
parent(homer, lisa).
parent(marge, lisa).
parent(homer, maggie).
parent(marge, maggie).
A Simplified Simpsons (Simplisons?)

mom(marge).
dad(homer).

parent(homer, bart).
parent(marge, bart).
parent(homer, lisa).
parent(marge, lisa).
parent(homer, maggie).
parent(marge, maggie).

child(X, Y) :-
A Simplified Simpsons (Simplisons?)

```prolog
mom(marge).
dad(homer).

parent(homer, bart).
parent(marge, bart).
parent(homer, lisa).
parent(marge, lisa).
parent(homer, maggie).
parent(marge, maggie).

child(X, Y) :-

siblings(X, Y) :-
```
Equality

Do these things give us the same value (possibly after using the current set of bindings, \( \Theta \))? 

<table>
<thead>
<tr>
<th>term_1 == term_2, given ( \Theta )</th>
<th>term_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>atom_1</td>
<td>atom_2</td>
</tr>
<tr>
<td>Are they literally the same?</td>
<td>( \Theta[var_2] == atom_1 )</td>
</tr>
<tr>
<td></td>
<td>false</td>
</tr>
<tr>
<td>term_1 var_1</td>
<td></td>
</tr>
<tr>
<td>( \Theta[var_1] == atom_2 )</td>
<td>( \Theta[var_1] == \Theta[var_2] )</td>
</tr>
<tr>
<td></td>
<td>( \Theta[var_1] ) == ( \Theta[t_1-1, \ldots, t_{1-n}] )</td>
</tr>
<tr>
<td></td>
<td>( \Theta[t_1-1, \ldots, t_{1-n}] ) == ( \Theta[t_2-1, \ldots, t_{2-m}] )</td>
</tr>
<tr>
<td>pred_1(t_1-1, \ldots, t_{1-n})</td>
<td></td>
</tr>
<tr>
<td>false</td>
<td>( \Theta[var_2] ) == ( \Theta[t_1-1, \ldots, t_{1-n}] )</td>
</tr>
<tr>
<td></td>
<td>pred_1 == pred_2,</td>
</tr>
<tr>
<td></td>
<td>( n ) == ( m ),</td>
</tr>
<tr>
<td></td>
<td>( t_{1-i} ) == ( t_{2-i} ).</td>
</tr>
</tbody>
</table>
How Prolog works

depth-first search through possible bindings

Given a goal…
e.g., siblings(bart, Who)

…for each rule with the goal’s name…
e.g., siblings(X, Y)

…if the rule has a body with subgoals…
e.g., parent(P, X), parent(P, Y), X \( \neq \) Y.

…for each subgoal…
e.g., parent(P, X) with \{X\(\rightarrow\)bart\}

…unify subgoal with goal & bind variables as needed.
e.g., \{P\(\rightarrow\)homer, X\(\rightarrow\)bart\}

If unification fails, backtrack and try the next option.
e.g., \{P\(\rightarrow\)homer, X\(\rightarrow\)bart, Y\(\rightarrow\)bart\} but X \( \neq \) Y. Therefore, backtrack and eventually try Y\(\rightarrow\)lisa
siblings(bart, who).

mom(marge).
dad(homer).

parent(homer, bart).
parent(marge, bart).
parent(homer, lisa).
parent(marge, lisa).
parent(homer, maggie).
parent(marge, maggie).
siblings(bart, Who).
siblings(bart, Y)

X \rightarrow \text{bart}
siblings(bart, Who).
siblings(bart, Y)  \quad X \mapsto \text{bart}

parent(P, bart)
siblings(bart, Who).

siblings(bart, Y)

parent(P, bart)

parent(homer, bart).

X ↦ bart

P ↦ homer
siblings(bart, Who).
siblings(bart, Y)

parent(P, bart)

parent(homer, bart).

parent(homer, Y)

X ↦ bart

P ↦ homer

...
siblings(bart, Who).
siblings(bart, Y)  
parent(P, bart)  
parent(homer, bart).  
parent(homer, Y)  
parent(homer, bart).  
parent(homer, bart). 

X ↦ bart 
P ↦ homer 
Y ↦ bart
siblings(bart, Who).
siblings(bart, Y) \quad X \rightarrow \text{bart}

parent(P, bart)

parent(homer, bart). \quad P \rightarrow \text{homer}

parent(homer, Y)

parent(homer, bart). \quad Y \rightarrow \text{bart}

bart \nleq\n bart \quad \text{false, so backtrack}
siblings(bart, Who).

siblings(bart, Y)  

\[
X \mapsto \text{bart}
\]

parent(P, bart)  

\[
P \mapsto \text{homer}
\]

parent(homer, bart).

parent(homer, Y)
siblings(bart, Who).

siblings(bart, Y)  \quad X \mapsto bart

parent(P, bart)

parent(homer, bart).  \quad P \mapsto homer

parent(homer, Y)

parent(homer, lisa).  \quad Y \mapsto lisa

parent(homer, lisa).
siblings(bart, Who).
siblings(bart, Y)  
parent(P, bart)  
parent(homer, bart).
parent(homer, Y)  
parent(homer, lisa).
bart \implies lisa

X \mapsto bart

P \mapsto homer

Y \mapsto lisa

Who \mapsto lisa

dad(homer).

mam(marge).

parent(homer, bart).
parent(marge, bart).
parent(homer, lisa).
parent(marge, lisa).
parent(homer, maggie).
parent(marge, maggie).

BART  LISA  MAGGIE

HOMER  MARGE
siblings(X, Y) :- parent(P, X), parent(P, Y), X \( \preceq \) Y.

?- siblings(bart, lisa).
true ;
true.

?- siblings(bart, bart).
false.

?- siblings(bart, Who).
Who = lisa ;
Who = maggie ;
Who = lisa ;
Who = maggie.
How Prolog works

depth-first search through possible bindings

Given a goal…
e.g., siblings(bart, Who)

…for each rule with the goal’s name...
e.g., siblings(X, Y)

…if the rule has a body with subgoals...
e.g., parent(P, X), parent(P, Y), X \(\neq\) Y.

…for each subgoal...
e.g., parent(P, X) with \{X\rightarrow bart\}

…unify subgoal with goal & bind variables as needed.
e.g., \{P\rightarrow homer, X\rightarrow bart\}

If unification fails, backtrack and try the next option.
e.g., \{P\rightarrow homer, X\rightarrow bart, Y\rightarrow bart\} but X \(\neq\) Y. Therefore, backtrack and eventually try Y\rightarrow lisa

Prolog gives us all true statements.
siblings(X, Y) :- parent(P, X), parent(P, Y), X \== Y.

?- siblings(bart, lisa).
true ;
true.

?- siblings(bart, bart).
false.

?- siblings(bart, Who).
Who = lisa ;
Who = maggie ;
Who = lisa ;
Who = maggie.
siblings(X, Y) :- parent(P, X), parent(P, Y), X \(!= Y.

?- siblings(bart, lisa), !.
true ;

?- siblings(bart, bart).
false.

?- siblings(bart, Who), !.
Who = lisa.
siblings(X, Y) :- parent(P, X), parent(P, Y), X \( \leq \) Y.

?- siblings(bart, lisa), !. true ;

?- siblings(bart, bart). false.

?- siblings(bart, Who), !. Who = maggie.
siblings(X, Y) :- parent(P, X), parent(P, Y), X \= Y.

?- setof(Who, siblings(bart, Who), Answer).
Answer = [lisa, maggie].
siblings(X, Y) :- parent(P, X), parent(P, Y), X \(\leq Y\).

?- setof(Who, siblings(bart, Who), Answer).
Answer = [lisa, maggie].

?- setof([X, Y], siblings(X, Y), Answer).
Answer = [[bart, lisa], [bart, maggie], [lisa, bart], [lisa, maggie], [maggie, bart], [maggie, lisa]].

list of lists!
Lists in Prolog

Construction

[] empty

[F|R] cons!

[E₁, E₂, E₃] literal
Lists in Prolog

Selection, i.e., pattern matching

\[\text{length}(L, N) : - L = [], N = 0.\]
Lists in Prolog

Selection, i.e., pattern matching

\[
\text{length}(L, N) :- L = [], N = 0.
\]

is the same as

\[
\text{length}([], 0).
\]

pattern matching on the left-hand side!
Lists in Prolog

Selection, i.e., pattern matching

\[
\text{length([], 0).}
\]

\[
\text{length([F|R], N) :- length(R, M), N is M+1.}
\]
Lists in Prolog

Selection, i.e., pattern matching

\[
\text{length}([], 0).
\]

\[
\text{length}([F|R], N) :- \text{length}(R, M), N \text{ is } M+1.
\]
Lists in Prolog

Selection, i.e., pattern matching

\[
\text{length}([], 0).
\]

"don't care"

\[
\text{length}([\_|R], N) :- \text{length}(R, M), N \text{ is } M+1.
\]

math, in Prolog

pattern matching on the left-hand side!
You-try! member

```
member( , ).

member( , ) :- ...
```

?- member(3, [1,2,3,4,5]).
true

?- member(6, [1,2,3,4,5]).
false.

?- member(E, [1,2,3,4,5]).
E = 1 ;
E = 2 ;
E = 3 ;
E = 4 ;
E = 5.
You-try! **member**

*member* is built in to SWI-Prolog

```
member(E, [E|_]).
member(E, [_|R]) :- member(E, R).
```

?- member(3, [1,2,3,4,5]).
true;
false.

?- member(6, [1,2,3,4,5]).
false.

?- member(E, [1,2,3,4,5]).
E = 1 ;
E = 2 ;
E = 3 ;
E = 4 ;
E = 5.
```
The “Zebra” Puzzle

a.k.a. The “Einstein” Puzzle

five nationalities
norwegian, brit, swede, dane, german

five pets
dog, bird, zebra, cat, horse

five cigars
pallmall, winfield, dunhill, rothmans, marlboro

five beverages
tea, coffee, milk, water, beer

five house colors
red, green, yellow, blue, white

fifteen clues
(1) The norwegian lives in the first house.
...

Who owns the zebra?
The “Zebra” Puzzle

call the clues

(1) The Norwegian lives in the first house.
(2) The person living in the center house drinks milk.
(3) The Brit lives in a red house.
(4) The Swede keeps dogs as pets.
(5) The Dane drinks tea.
(6) The Green house is next to, and on the left of the White house.
(7) The owner of the Green house drinks coffee.
(8) The person who smokes Pall Mall rears birds.
(9) The owner of the Yellow house smokes Dunhill.
(10) The man who smokes Marlboro lives next to the one who keeps cats.
(11) The man who keeps horses lives next to the one who smokes Dunhill.
(12) The man who smokes Winfields drinks beer.
(13) The German smokes Rothmans.
(14) The red house is to the right of the blue.
(15) The Norwegian doesn’t live by the red, white, or green houses.
The “Zebra” Puzzle

Representation in Prolog: all possible houses, i.e., the “state space”

houses( [ H1, H2, H3, H4, H5 ]):-

H1 = [ N1, P1, S1, B1, C1 ],
H2 = [ N2, P2, S2, B2, C2 ],
H3 = [ N3, P3, S3, B3, C3 ],
H4 = [ N4, P4, S4, B4, C4 ],
H5 = [ N5, P5, S5, B5, C5 ],
perm( [N1,N2,N3,N4,N5], [norwegian, brit, swede, dane, german] ),
perm( [P1,P2,P3,P4,P5], [dog, bird, zebra, cat, horse] ),
perm( [S1,S2,S3,S4,S5], [pallmall, winfield, dunhill, rothmans, marlboro] ),
perm( [B1,B2,B3,B4,B5], [tea, coffee, milk, water, beer] ),
perm( [C1,C2,C3,C4,C5], [red, green, yellow, blue, white] ).
The “Zebra” Puzzle

Representation in Prolog: the clues

einstein(Houses) :-
  Houses = [[norwegian, _, _, _, _], _, [_, _, _, milk, _], _, _],
  member([brit, _, _, _, red], Houses),
  …
  houses( Houses ),  % it's important to have this LATE (otherwise it's 120**5)
  …

(1) The Norwegian lives in the first house.
(2) The person living in the center house drinks milk.
(3) The Brit lives in a red house.
…
1.1 Syntax of the Predicate Calculus

4-1 Definition. The syntax of the predicate calculus ($\mathcal{P}C$) consists of symbols and formulas as follows:

Symbols

parentheses: (, )

sentential connectives: $\neg$, $\lor$, $\land$, $\rightarrow$, $\leftrightarrow$

quantifiers: $\forall$, $\exists$

$\mathcal{Sc}$ letters (sentential letters): $A, B, \cdots, Z$, and any of these letters with a positive Arabic numeral subscript.

predicate symbols: An $n$-ary predicate is an uppercase letter, $A, \cdots, Z$, with the numeral $n$ as a superscript, where $n$ denotes the arity of the predicate and $0 < n$. These uppercase letters may also have numerical subscripts. Note: We will usually omit the superscript when we know the arity of a predicate.

individual constants: lowercase letters $a, \cdots, r$, with or without numerical subscripts.

individual variables: lowercase letters $s, \cdots, z$, with or without numerical subscripts.

Formulas

The set of all predicate calculus ($\mathcal{P}C$) formulas is defined recursively, beginning with the atomic formulas.

Atomic Formula:

Any single $\mathcal{Sc}$ letter, or an $n$-ary predicate followed by exactly $n$ symbols, each of which is either an individual constant or a variable.

Formula:

Any atomic formula, or any expression (finitely long string of symbols) that is obtainable by use of the following predicate calculus construction rules ($\text{PCCR}$):
Prolog is syntactic sugar for the predicate calculus.*

The Semantics of Predicate Logic as a Programming Language

M. H. V. EMDEN AND R. A. KOWALSKI
University of Edinburgh, Edinburgh, Scotland

Abstract: Semantics of first-order predicate logic can be usefully interpreted as programs. In this paper, the operational and equational semantics of predicate logic programs are defined, and the connection with the proof theory and the model theory of logic are investigated. It is concluded that operational semantics is a part of proof theory and that equational semantics is a special case of model theory semantics.

Key words and phrases: predicate logic as a programming language, semantics of programming languages, resolution theorem proving, operational versus equational semantics, NL-resolution, fuzzy characterisation.

1. Introduction

Predicate logic plays an important role in many formal models of computer programs [3, 4, 17]. Here we are concerned with the interpretation of predicate logic as a programming language [5, 16]. The Prolog system for PROgramming in LOGic, based upon the procedural interpretation, has been used for several ambitious programming tasks.
Hmmm

Racket

Prolog

Turing Machine

0 jumpn 0

Turing 1937

\lambda

Calculus

\equiv

\lambda

Calculus

\equiv

Predicate Calculus

p :- p

Church (1936), Turing (1937)
Hmmm \equiv Racket \equiv Prolog

Turing Machine 0 \text{\textit{jump}} 0 \equiv \lambda \text{Calculus} \equiv \text{Predicate Calculus}

\begin{array}{c}
0 \\
\text{\textit{jump}} 0
\end{array}

\begin{array}{c}
(\lambda (x) (x \ x)) (\lambda (x) (x \ x))
\end{array}

\begin{array}{c}
p :- p
\end{array}
Hmmm ≡ Racket ≡ Prolog

Turing Machine
0 jump n 0

λ Calculus
((λ(x)(x x))(λ(x)(x x)))

Predicate Calculus
p :- p

Turing (1937)
Church (1936), Turing (1937)