ruby-prolog

Logical programming with Ruby declarations.

2009.02.09 - Preston Lee <preston.lee@openrain.com> - Http://prestonlee.com

Logical programming paradigm.

- **Declarative**.
  We define the rules for the world, but not how to process them.

- **Inherently recursive**.
  The interpreter will automatically recurse to evaluate a query behind the scenes.

- **Turing complete**.
  You can program without objects or functions, if you so choose.

- **Artificial intelligence**.
  It's easier to define and resolve complex logical problems when we think mathematically.

Core Prolog concepts.

- **Rules**
  - Generic semantic definitions (*predicates*) of how your world works via *clauses*.
  - Declarations are formal mathematics (*predicate logic*) in programming syntax.
- **Facts**
  - Assertions of truth about the world.
  - Bob is Charlie's father.
  - Bob is Dale's father.
- **Queries**
  - Attempts to resolve an unknown logical statement using the rule given the facts.

Project history.

- **tiny_prolog** resolver / unification implementation from teh internets. Various small additional patches.

- **Refactored** to be **object-oriented**, not mess with method_missing globally, play nice with **garbage collection** and support **multiple simultaneous contexts**.

- **Test cases** from scratch, and various snippets ported from Prolog.

Simple family tree, in Prolog.

- **Rules**
  - We are siblings if we share a parent.
  - A father is a parent.
  - A mother is a parent.
- **Facts**
  - Alice is Charlie's mother.
  - Bob is Charlie's father.
  - Bob is Dale's father.
- **Queries**
  - Who are Alice's siblings?

```
sibling(X, Y)      :- parent_child(Z, X), parent_child(Z, Y).

parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).

mother_child(alice, charlie).
father_child(bob, charlie).
father_child(bob, dale).

/*
* Who are Charle's siblings?
* sibling(charlie, X).
*
* Who are Charlie's parents?
* parent_child(X, sally).
*/
```

Imperative programming support.

```
/*************************************************
 * 99 Bottles of Beer
 * Paul J. DeMarco  9/20/2002
 * beer.pro
 * To execute start gprolog (others may work)
 * consult('beer.pro').
 * drink(beer,99).
 *************************************************/
english(beer,0):-
                        write('no more bottle of beer').
english(beer,1):-
                        write('1 bottle of beer').
english(beer,X):-
                        X >= 2,
                        write( X ) ,
                        write(' bottles of beer').

drink(beer,X):- X >= 1,
                        english(beer,X),
                        write(' on the wall, '),
                        english(beer,X),
                        write(', take one down, pass it around\n'),
                        X1 is X - 1,
                        english(beer,X1),
                        write(' on the wall.\n'),
                        drink(beer, X1).
```

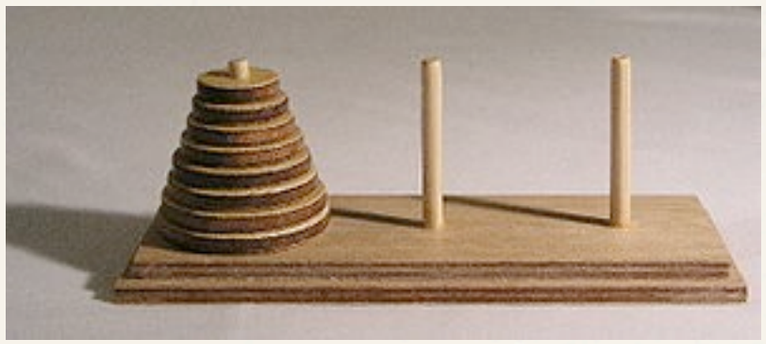
ruby-prolog

- Prolog-like DSL.
- Object-oriented wrapper.
- Not as complete as Prolog.

```
c = RubyProlog::Core.new
c.instance_eval do

  vendor['dell'].fact
  vendor['apple'].fact

  model['ultrasharp'].fact
  model['xps'].fact
  model['macbook'].fact
  model['iphone'].fact

  manufactures['dell', 'ultrasharp'].fact
  manufactures['dell', 'xps'].fact
  manufactures['apple', 'macbook'].fact
  manufactures['apple', 'iphone'].fact

  is_a['xps', 'laptop'].fact
  is_a['macbook', 'laptop'].fact
  is_a['ultrasharp', 'monitor'].fact
  is_a['iphone', 'phone'].fact

  kind['laptop']
  kind['monitor']
  kind['phone']

  model[:M] <<= [manufactures[:V, :M]]

  vendor_of[:V, :K] <<= [vendor[:V],
                manufactures[:V, :M], is_a[:M, :K]]

  p query(is_a[:K, 'laptop'])
  p query(vendor_of[:V, 'phone'])

end
```

Complex logical reasoning.

The Towers of Hanoi



Two implementations.

Prolog

```
move(1,X,Y,_) :-
    write('Move top disk from '),
    write(X),
    write(' to '),
    write(Y),
    nl.

move(N,X,Y,Z) :-
    N>1,
    M is N-1,
    move(M,X,Z,Y),
    move(1,X,Y,_),
    move(M,Z,Y,X).

/* move(3,left,right,center). */
```

ruby-prolog

```
c = RubyProlog::Core.new
c.instance_eval do

  move[0,:X,:Y,:Z] <<= :CUT
  move[:N,:A,:B,:C] <<= [
    is(:M,:N){|n| n - 1}, # reads as "M IS N - 1"
    move[:M,:A,:C,:B],
    write_info[:A,:B],
    move[:M,:C,:B,:A]
  ]
  write_info[:X,:Y] <<= [
    write["move a disc from the "],
    write[:X], write[" pole to the "],
    write[:Y], writenl[" pole "]
  ]

  move[3,"left","right","center"]

end
```

ACL Example

examples/acls.rb

Ideas for the future.

- **active_prolog** - Logical interface to relational ActiveRecord objects.
- **logical_authentication** - Easy custom ACL creation and enforcement.
- **logical_search** - Custom DB search query builder using English-like predicates.
- **...**

Thanks!

- **Code**: http://github.com/preston/ruby-prolog/tree/master
- **Releases**: http://rubyforge.org/projects/ruby-prolog/