Prolog

An introduction,

By Sajjad Heydari

SajjadHeydari74[at]gmail[dot]com

Prolog…

is

- Lazy.
- Based on logical rules.

is not

- Like traditional programming languages (C/C++, Java, Python)
- A functional language, it doesn’t have any function.
- A good calculator.

Term

constants

- Atoms
- Numbers
- Variables
- Relations

Terms→ Atoms

- Start with lowercase letters
- Can contain _, lowercase or uppercase letters, numbers

Example:

```
prolog.
this_is_an_atom.
number9.
```

Terms→ Numbers

- Any real number.

Example:

9
3.14
1e2

Terms→ Variables

- Start with uppercase letters or _
- Can contain _, lowercase or uppercase letters, numbers

Example:

```
A
_var
Var123
```

Anonymous Variable

_

- The Prolog variable _ (underscore) is a "don't-care" variable, which will match anything.

Example:

```
bad(Dog):-
        bites(Dog, _).
```

Terms→ Relations

- Consists of a *functor* and **n** *arguments*.
- *Functor* must be an *atom*.
- *Arguments* must be a **Term.**

Example:

```
fun(prolog).
functor(A1,argument2,_,a_complex(argument)).
```

Clauses

- Facts
- Rules
- Queries

Clauses→ Facts

- Any *term* ending with a dot(.) is a fact.
- The *term* is called the ***head***.
- A fact is considered true once prolog has parsed it.

Example:

```
prolog.

food(pizza).
likes(sajjad, pizza)
```

Clauses→ Rules

neck

head :-
body.

- Consists of a head, neck and body.
- Head is a clause (usually with variables in it).
- Body is a comma separated list of goals.
- Read:
    - If Body then Head.

Example

- Likes(Person, Thing):-
  - Food(Thing).

- is_parent(Person_A, Person_B):-
  - father(Person_A, Person_B).
- is_parent(Person_A, Person_B):-
  - mother(Person_A, Person_B).

Procedure

Example

- happy(Person):-
  - eats(Person, Food), ← and
  - likes(Person, Food).

- A ***Person*** is happy if he is eating ***Food*** and ***he*** likes the ***Food***.

Clauses⟶ Queries 1

- List of one or more goals,
- Typed at prolog interpreter ?-
- Comma separated.

Example: ?-like(sajjad,pizza).

Clauses⟶ Queries 2

- If a query doesn’t have any variable,
  - Prolog answers with true/false
- If the query includes variables,
  - Prolog tries to find all bounds of that variable that make the statement true.
    - You can get the next one by typing ;

Example:

```
?-like(sajjad,pizza).
true

?-like(sajjad,X).
X = pizza;
X = burger;
true.
```

Pressed ;

Special Commands

Negation

- not(X).

Write

- write(“String”).
- write(term).

New Line

- nl.

Trace

- trace.
- notrace.

Exit

- halt.

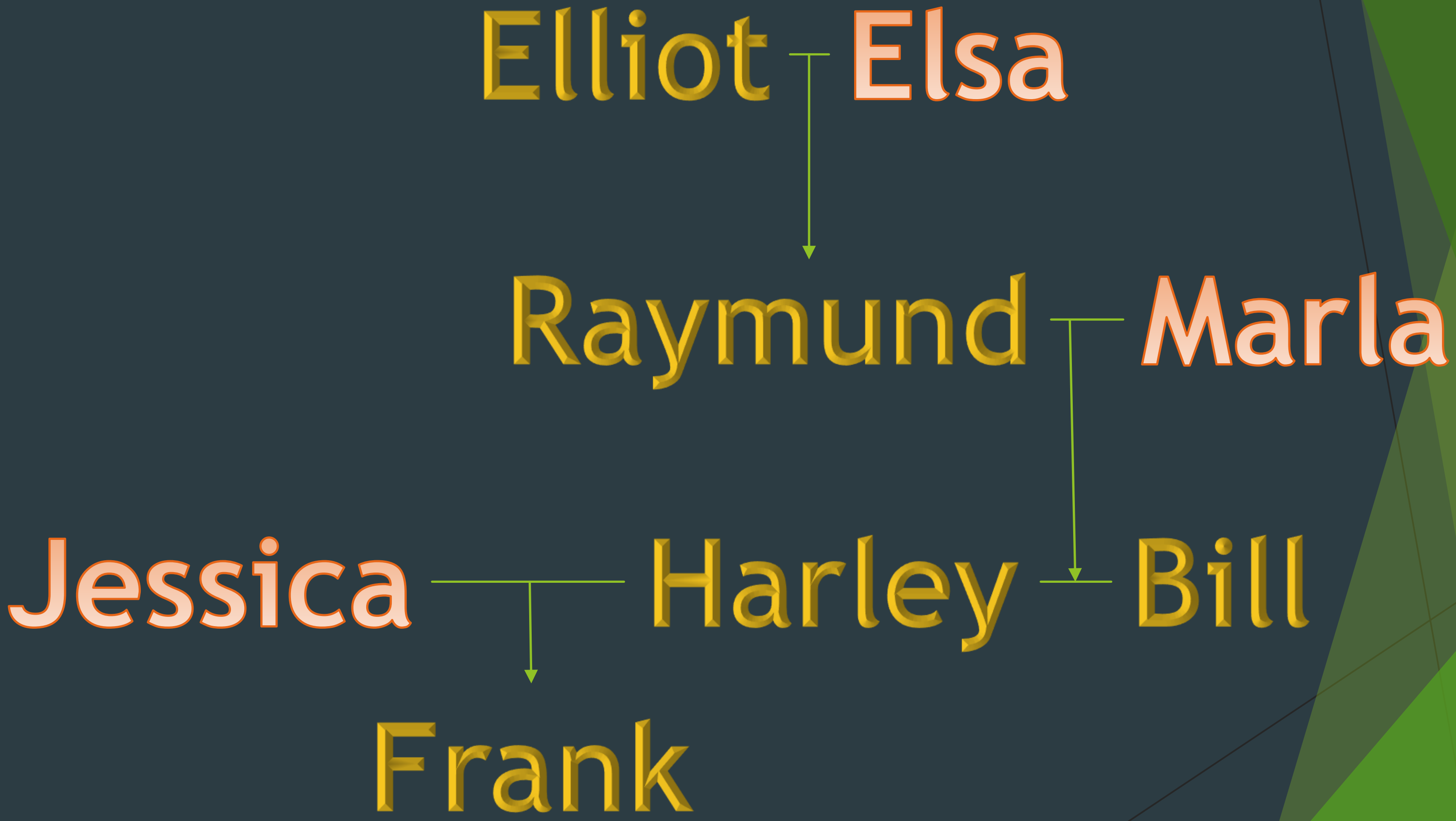
Load

- [file]
- Loads ‘file.pl’

Comment

- %Comment
- /*Multi lined Comment */

Family Tree



Elliot Elsa

Raymund Marla

Jessica Harley Bill

Frank

Mother / Father

- mother(Person_a, Person_b):-
  - parent(Person_a, Person_b),
  - female(Person_a).

- father(Person_a, Person_b):-
  - parent(Person_a, Person_b),
  - male(Person_a).

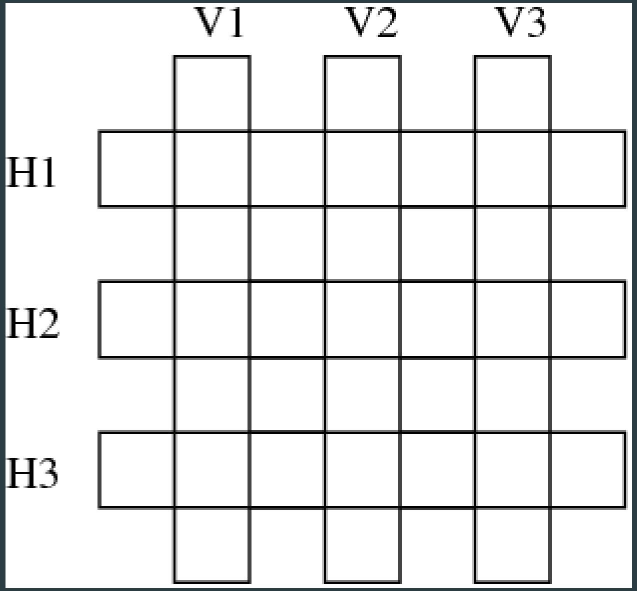
Ancestor

- ancestor(Person_a, Person_b):-
  - parent(Person_a, Person_b).

- ancestor(Person_a, Person_b):-
  - parent(Person_a, Person_c),
  - ancestor(Person_c, person_b).

Crossword Puzzle

- Fill the crossword with the following words:
    - abalone, abandon, anagram, connect, elegant, enhance

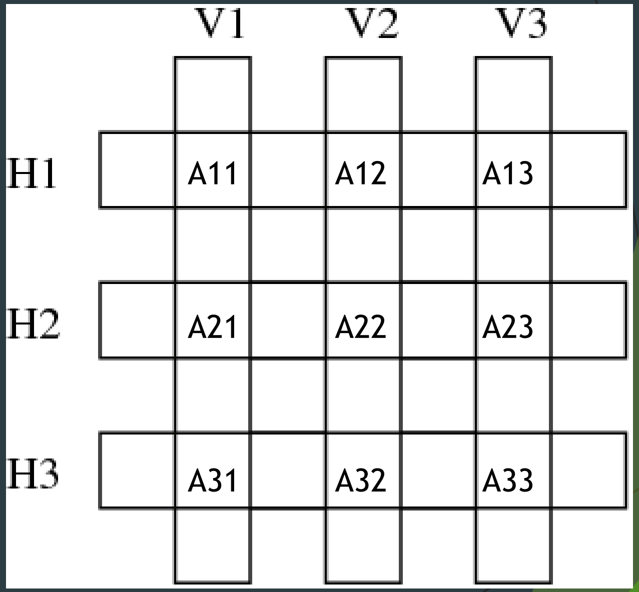


Describing words in prolog

- word(abalone, a, b, a, l, o, n, e).
- word(abandon, a, b, a, n, d, o, n).
- word(anagram, a, n, a, g, r, a, m).
- word(connect, c, o, n, n, e, c, t).
- word(elegant, e, l, e, g, a, n, t).
- word(enhance, e, n, h, a, n, c, e).

Describing Crossword

- crossword(H1, H2, H3, V1, V2, V3):-
    - word(H1, _, A11, _, A12, _, A13, _),
    - word(H2, _, A21, _, A22, _, A23, _),
    - word(H3, _, A31, _, A32, _, A33, _),
    - word(V1, _, A11, _, A21, _, A31, _),
    - word(V2, _, A12, _, A22, _, A32, _),
    - word(V3, _, A13, _, A23, _, A33, _).



Query

- ?-crossword(H1, H2, H3, V1, V2, V3).
- H1 = abalone,
- H2 = anagram,
- H3 = connect,
- V1 = abandon,
- V2 = elegant,
- V3 = enhance ;

- H1 = abandon,
- H2 = elegant,
- H3 = enhance,
- V1 = abalone,
- V2 = anagram,
- V3 = connect ;

- false.

List

- A comma-separated sequence of items, between square brackets.
- Similar to arrays.

Example:

Empty List

```
[ ]
[1, 2, 3]
[sajjad, [a,b], term_ex(1, foo) ]
```

List [H | T]

- A list could be represented as Head and Tail.
- Head is the first element.
- Tail is a list.

Example: [1, 2, 3] = [1 | [2,3]]

List [H | T], (cont.)

- Head could be a series of items.
- [a, b | T]
- [a, b, c | T]
- [a, b , c, d | T]

Example: [1, 2, 3, 4, 5] = [1 , 2| [3, 4, 5]]

Member

- member(X, [X|T]).
- member(X, [_|T]):-
  - member(X, T).

Anonymous Variable

Appending

Ex: append([1, 2, 3], [4, 5] , [1, 2, 3, 4, 5]) .

Appending → Rules

- append([], L, L).
- append([H | T] , L2, [H | L3]):-
  - append(T, L2, L3).

Appending → Query

- ?-append([1, 2, 3, 4] , [5, 6], X).

- X = [1, 2, 3, 4, 5, 6].

Prefix

- prefix(S,L):-
    - append(S, _, L).

Suffix

- suffix(S, L):-
  - append(_, S, L).

Reversing

Ex: reverse([1,2,3,4], [4,3,2,1]).

Reversing → Rules

- reverse([ ], A, A).
- reverse([H | T], A, R) : -
  - reverse(T, [H | A], R).

- reverse(A, R):-
  - reverse(A, [], R).

Reversing → Query

- ?- reverse([1,2,3,4], X).
- X = [4, 3, 2, 1].

Prolog	Description
a is b.	a = b
a + b	a + b
a – b	a – b
a * b	a x b
a / b	a / b
mod(a, b)	a mod b
a > b.	a > b
a < b.	a < b
a >= b	a >= b
a <= b	a <= b
a =:= b	a = b
a =/= b	a != b

is vs. = vs. =:=

	‘is’	=	=:=
On left	Expression	Term	Expression
On Right	Variable or Number	Term	Expression
Action	Evaluate right, compare with left	Compares the terms (doesn’t evaluate)	Evaluates and compare

Expressions can not contain Variables!

Example

- 3 is 2 + 1. Correct
- 2+1 is 3. Error
- X is 2 + 1. Correct (X = 3)

- 3 = 2 + 1. Error
- 2 + 1 = 2 + 1. Correct
- 1 + 2 = 2 + 1. Error
- X + 2 = 1 + 2. Error

- 2 + 1 =:= 1 + 2. Correct
- X + 2 =:= 1 + 2. Error

Length

- Len([a,b,c], 3).

Length → Rules

- len([], 0).
- len([H|T], X) :-
  - len(T, A),
  - X is A + 1.

Length → Queries

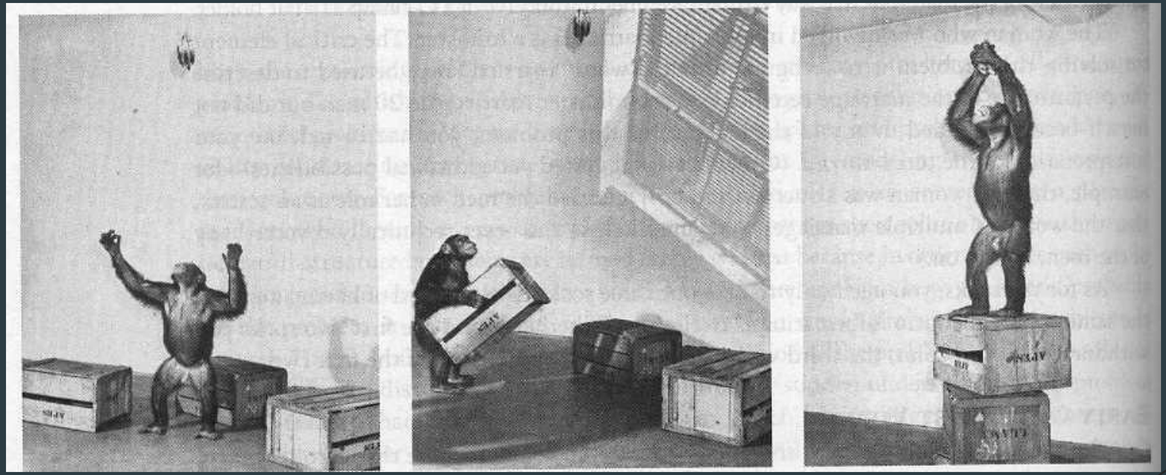
- ?- len([A,B], X).
- X = 2.

- ?- len([1,2,3],4).
- false.

Monkey and Banana

- A Monkey is on the floor.
- There is a banana on the celling, beyond reach of the monkey.
- There is a box in the corner.

- Can the monkey eat the banana?



M&B → State

- State(Monkey_X, Monkey_Y, Box_position, Has_banana)

- Monkey_x: door, middle, corner
- Monkey_y: on_floor, on_box
- Box_position: door, middle, corner
- Has_banana: yes, no

M&B → Transition → Grasp

- move(
- state(middle, on_box, middle, no),
- grasp,
- state(middle, on_box, middle, yes)
-).

M&B → Transition → Climb

- move(
- state(X, on_floor, X, Has),
- climb,
- state(X, on_box, X, Has)
-).

M&B → Transition → Push

- move(
- state(X, on_floor, X, Has),
- push,
- state(X2, on_floor, X2, Has)
-).

M&B → Transition → Walk

- move(
- state(_, on_floor, Y, Has),
- walk,
- state(_, on_floor, Y, Has)
-).

M&B → Transition → canget

- canget(state(_,_,_,yes)):-
- write("get"),nl.

M&B → Transition → canget (cont.)

- canget(State1) :-
- move(State1, Move, State2),
- canget(State2),
- write(State2), nl,
- write(Move), nl.

M&B → Query

- canget(state(door, on_floor, corner, no)).

- get
- state(middle,on_box,middle,yes)
- grasp
- state(middle,on_box,middle,no)
- climb
- state(middle,on_floor,middle,no)
- push
- state(corner,on_floor,corner,no)
- walk
- true .

Prolog

An introduction,

By Sajjad Heydari

SajjadHeydari74[at]gmail[dot]com