Meta-Interpretive Learning: Achievements and Challenges, Final Report

SHRUTI NAIR, SBU ID - 111481332

December 20, 2017

Abstract

This report provides an implementation and a deep understanding on the concepts of Meta Interpretive Learning with its effective usage for three different problems. Those are Parent generation, stair case problem and polygon representation problem. The report comprises of the steps included for adding extra test cases using stair case problem and polygon representation problem. In addition to it, certain clauses have been refractured. It further includes the challenges posed by the current meta learning model and the recent work going towards it.

Keywords - *Meta Interpretive Learning, Meta Rules*

Contents

1 Technologies Used

The code has been developed in pure prolog. Some of the code has been taken from the copyrighted code by Andrew Copper from Imperial College London.The platform used for developing this project is SWI Prolog. The source code basically contains one file called 'metagol' which is used as the basic code for developing MIL and the rest of the files including 'parent' , 'staircase' and 'polygon' have been used to test the metagol file and it's subsequent result.

2 Meta Interpretive Learning

2.1 What is Meta Interpretive Learning?

To clearly understand the concept of Meta Interpretive Learning, we need to first understand how does a meta interpreter actually work.

2.1.1 Meta Interpreter

We already know that Prolog has it's own proof strategy to solve the goal by unifying all the clauses in the given database. However, if we are already provided with the prolog database,we can actually manipulate the data as if it were any other sort of data i.e - programming where the data is bits of program, rather than information about entities in the world. This technique is called meta interpretation.[1]

Prolog meta-interpreter takes a Prolog program and a goal and attempts to prove that the goal logically follows from the program. When we create a meta interpreter, we design a whole new proof strategy for solving the goal.

2.1.2 Meta Interpretive Learning - MIL

Meta Interpretive Learning is a new and recent advancement in Inductive Logic Programming towards Machine Learning[2]. It's novel aspect lies in it's capability of introducing sub-definitions while learning a predicate definition. This allows the decomposition into a hierarchical structure of reusable parts.

2.1.3 Difference between Meta Interpretive Learning and Meta Interpreter

A meta-interpretive learner in addition to the capabilities of the meta interpreter also fetches higher-order meta-rules whose heads unify with the goal, and saves the resulting meta-substitutions to form a program.

3   Working Example

If we are given with various examples of ancestor relation and the background knowledge of father and mother facts only,then a system must not only be able to learn ancestor as a recursive definition but also simultaneously invent parent to learn these definitions.

Let us suppose we are given with a Prolog relation as follows:

Family relations (Dyadic)

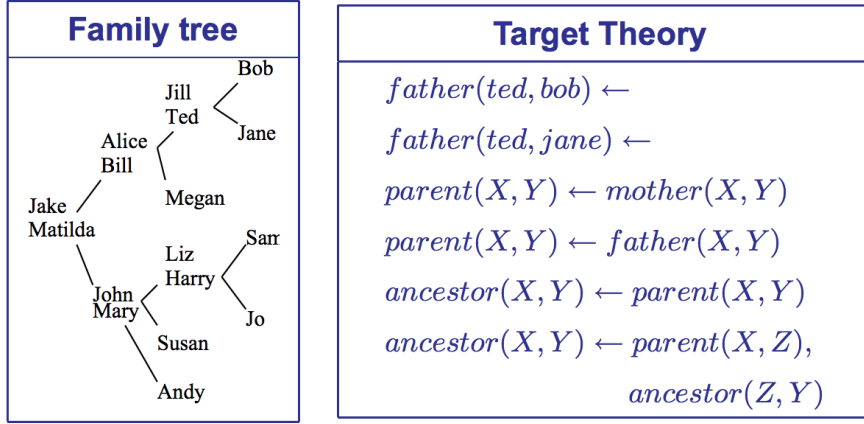| Family tree	Target Theory
Bob Jill Ted Jane Alice Bill Megan Jake Matilda Sam Liz Harry John Mary Jo Susan Andy	$father(ted, bob) \leftarrow$ $father(ted, jane) \leftarrow$ $parent(X, Y) \leftarrow mother(X, Y)$ $parent(X, Y) \leftarrow father(X, Y)$ $ancestor(X, Y) \leftarrow parent(X, Y)$ $ancestor(X, Y) \leftarrow parent(X, Z),$ $ancestor(Z, Y)$

Figure 1: Family relation example

The meta interpreter execution for the above problem is as follows :

- In order to execute the meta interpreter,a user gives meta rules which are higher-order expressions describing the forms of clauses permitted in hypothesied programs.

Metarules

Name	Meta-Rule	Order
Instance	$P(X, Y) \leftarrow$	$True$
Base	$P(x, y) \leftarrow Q(x, y)$	$P \succ Q$
Chain	$P(x, y) \leftarrow Q(x, z), R(z, y)$	$P \succ Q, P \succ R$
TailRec	$P(x, y) \leftarrow Q(x, z), P(z, y)$	$P \succ Q,$ $x \succ z \succ y$

Figure 2: Meta Rule Table

- Each of the meta rules are given with an order constraint to execute safe termination.

- The meta interpreter proves the examples and saves the substitutions for existentially quantified variables found in the associated meta-rules for any successful proof.

- The interpreter recursively proves a series of atomic goals by matching them against the heads of meta-rules.

$$prove([], BK, BK).$$

$$prove([Atom|As], BK, BK\_H) : -$$

$$metarule(Name, MetaSub, (Atom \text{ :- } Body), Order),$$

$$Order,$$

$$save\_subst(metasub(Name, MetaSub), BK, BK\_C),$$

$$prove(Body, BK\_C, BK\_Cs),$$

$$prove(As, BK\_Cs, BK\_H).$$

Figure 3: Generalized meta interpreter

- After testing the Order constraint, save substr checks whether the meta-substitution is already in the program and otherwise adds it to form an augmented program. On completion the returned program, by construction, derives all the examples.

- When these substitutions are applied to meta rules, they result in a first order logic predicates programs.

4 Implementation

The various modules used for it involves the construction of learning task and sequence and proving the models based on the background knowledge. The metagol's file has been refractured by efficiently implementing the constructor clause which the author suggested needs serious changes.

4.1 Parent Predicate generation

The first test case generated for the problem is the parent predicate generation problem. Here is the code snippet which shows how to run the parent predicate for the metagol. I have set both positive and negative examples to run through the model. The positive and negative examples has been chosen for testing the correctness.

4.2 Staircase problem

The staircase generation problems involves the background knowledge of horizontal and vertical lengths. The same knowledge has been used for creating both positive and negative cases to check it's correctness. The meta rules remained the same.

A staircase is represented as a set of ordered planes, where the background predicates vertical and horizontal describe adjacent planes.

stair(X,Y) :- a(X,Y).

stair(X,Y) :- a(X,Z), stair(Z,Y).

a(X,Y) :- vertical(X,Z), horizontal(Z,Y).

```
:- use_module('metagol').

%% first-order background knowledge
mother(ann,amy).
mother(ann,andy).
mother(amy,amelia).
mother(linda,gavin).
father(steve,amy).
father(steve,andy).
father(gavin,amelia).
father(andy,spongebob).

%% predicates that can be used in the learning
prim(mother/2).
prim(father/2).

%% metarules
metarule([P,Q],([P,A,B]:-[[Q,A,B]])).
metarule([P,Q,R],([P,A,B]:-[[Q,A,B],[R,A,B]])).
metarule([P,Q,R],([P,A,B]:-[[Q,A,C],[R,C,B]])).

%% learning task
a :-
  %% positive examples
  Pos = [
    parent(ann,amy),
    parent(steve,amy),
    parent(ann,andy),

  ],
```
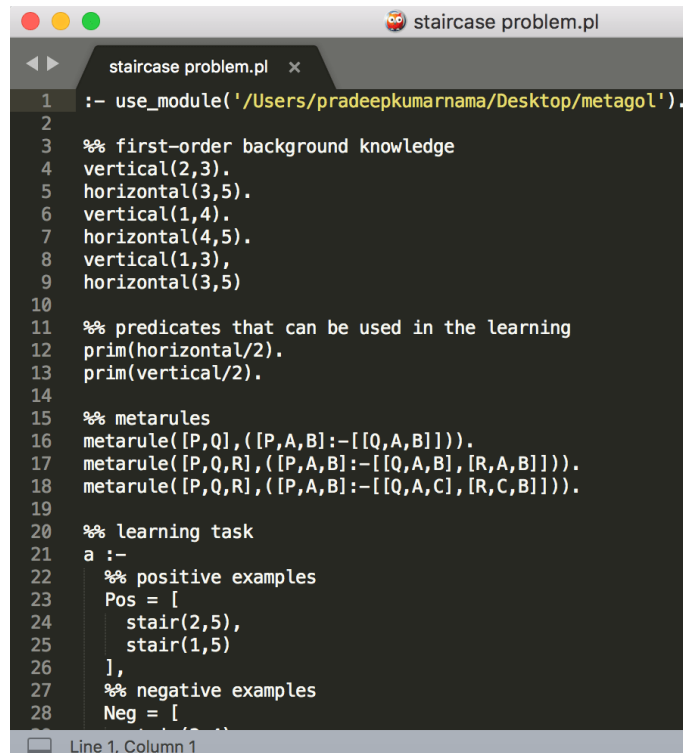
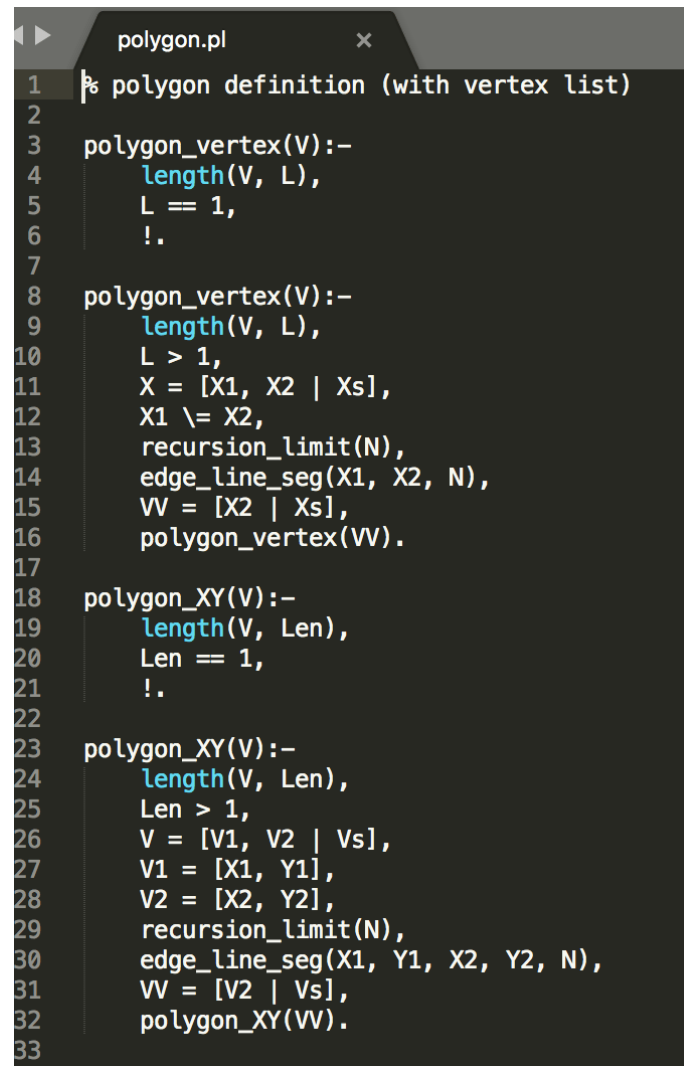Figure 4: Parent Predicate tester

```
staircase problem.pl

staircase problem.pl  ×

1   :- use_module('/Users/pradeepkumarnama/Desktop/metagol').
2
3   %% first-order background knowledge
4   vertical(2,3).
5   horizontal(3,5).
6   vertical(1,4).
7   horizontal(4,5).
8   vertical(1,3),
9   horizontal(3,5)
10
11  %% predicates that can be used in the learning
12  prim(horizontal/2).
13  prim(vertical/2).
14
15  %% metarules
16  metarule([P,Q],([P,A,B]:-[[Q,A,B]])).
17  metarule([P,Q,R],([P,A,B]:-[[Q,A,B],[R,A,B]])).
18  metarule([P,Q,R],([P,A,B]:-[[Q,A,C],[R,C,B]])).
19
20  %% learning task
21  a :-
22    %% positive examples
23    Pos = [
24      stair(2,5),
25      stair(1,5)
26    ],
27    %% negative examples
28    Neg = [

Line 1, Column 1
```

Figure 5: Stair Case Problem

5

4.3 Polygon Construction Problem

The polygon construction problem involves learning the shape of the polygon. I believe this can be better implemented using OpenCV however I have managed to construct the basics of the polygon testing file which can be used to run on the metagol file. The code snippet is below.

```
polygon.pl                    ×
1   % polygon definition (with vertex list)
2
3   polygon_vertex(V):-
4       length(V, L),
5       L == 1,
6       !.
7
8   polygon_vertex(V):-
9       length(V, L),
10      L > 1,
11      X = [X1, X2 | Xs],
12      X1 \= X2,
13      recursion_limit(N),
14      edge_line_seg(X1, X2, N),
15      VV = [X2 | Xs],
16      polygon_vertex(VV).
17
18  polygon_XY(V):-
19      length(V, Len),
20      Len == 1,
21      !.
22
23  polygon_XY(V):-
24      length(V, Len),
25      Len > 1,
26      V = [V1, V2 | Vs],
27      V1 = [X1, Y1],
28      V2 = [X2, Y2],
29      recursion_limit(N),
30      edge_line_seg(X1, Y1, X2, Y2, N),
31      VV = [V2 | Vs],
32      polygon_XY(VV).
33
```

Figure 6: Stair Case Problem

5 Challenges

A number of open challenges exist for Meta-Interpretive Learning. These include the following.

- **Generalise beyond Dyadic logic**

  Many Prolog programs are more natural to represent when represented with more than two arguments.

- **Deal with classification noise**

  Most data sources for machine learning contain both classification and attribute noise.

- **Active learning**

  Most forms of machine learning are passive in the sense that they take a given training data set and generate a model. Active learning involves proposing and testing instances which are classified either by a user of by carrying out experiments in the real world.

- **Efficient problem decomposition**

  Finding efficient ways of decomposing the definitions in MIL is one of the hardest open problems in the field.

6 Conclusion and Goals Achieved

The program has been designed for executing an meta interpretive learning model for testing various applications onto it.

I have achieved three goals in regarding to this project :-

1) Created three applications, one involving using parent as a predicate for running the metagol file.

2) Implemented the staircase problem and polygon regeneration (not completely) so that it can be run for meta gol file.

3) Refractured the code for construction clause which the author needed re modification.

7 Acknowledgment

I would like to thank Professor Paul Fodor for his support and encouragement during the course of the project. I was able to deliver the project with results with which I am highly satisfied with. A huge part of the credit for that goes to Professor Fodor.

References

[1] Tim Smith *Meta-interpretation*. http://www.inf.ed.ac.uk/teaching/courses/aipp/lecture_slides/ 17_meta_interp.pdf.

[2] Muggleton, S., Lin, D., Tamaddoni-Nezhad *Meta-interpretive learning of higher-order dyadic datalog: predicate invention revisited*. Mach. Learn. 100(1), 49–73 (2015). http://www.doc.ic.ac.uk/shm/Papers/metagolDMLJ.pdf.