6/5/2021

Luka Leko, Paul Knoll

Symbolic AI: Task 2

Group 01:

Contents

[1. Getting Started 2](#_Toc73810949)

[Basic Aleph File Structure 2](#_Toc73810950)

[2. Recursion 5](#_Toc73810951)

[(a) Learn the member predicate: 5](#_Toc73810952)

# Getting Started

We first tried to understand what Aleph does by testing it out. For this we installed it as instructed in the task specification and then executed Aleph on the Train example from Ryszard Michalski. This is what we have learned:

## Basic Aleph File Structure

The whole Aleph Logic is contained within a single .pl file, which we will call aleph-swi.pl. For Aleph to be able to induce a set of rules (also called a theory), it needs the following three files contained within the same folder:

1. The .f file: It contains the set of positive examples. The examples are formatted as simple prolog facts. Aleph will consider those as correct during his learning procedure:

.f file for the Train example:

**eastbound(east1).**

**eastbound(east2).**

**eastbound(east3).**

**eastbound(east4).**

**eastbound(east5).**

1. The .n file: It contains the set of negative examples. It looks the same as .f file, except that Aleph will consider those facts as wrong during his learning procedure.
2. The .b file: This file is the most interesting out of the three. The Background knowledge is in the form of Prolog clauses that encode information relevant to the domain. This file also contains language and search restrictions for Aleph. Amongst other things, this file mostly contains:
   1. Modes. These declare the mode of call for predicates that can appear in any clause hypothesised by Aleph.
   2. Determinations. Determination statements declare the predicates that can be used to construct a hypothesis.
   3. Type definitions.
   4. You can also set certain variables used by Aleph with set(variableName, value).

When one wants to run Aleph to learn rules to distinguish between positive and negative examples, one has to:

1. Open swipl,
2. Consult the aleph-swi.pl file.
3. Read\_all(filename). This will make Aleph load the example.
4. Call Induce. This will initiate the learning step. Based on the configuration within the Background Knowledge file, this will produce a certain set of rules. During the learning, Aleph will also print out the theory that he has learned, including a matrix stating how many of the positive and negative examples one can correctly identify given that theory. Used on the trains example, Aleph is able to find a theory consisting of only a single rule which is able to correctly predict every positive example and exclude every negative example:

[theory]

[Rule 1] [Pos cover = 5 Neg cover = 0]

eastbound(A) :-

has\_car(A,B), short(B), closed(B).

[Training set performance]

Actual

+ -

+ 5 0 5

Pred

- 0 5 5

5 5 10

Accuracy = 1

The solution that we have found by hand to distinguish between eastbound trains and westbound trains looks as follows:

**eastbound(A) :-**

**has\_car(A, B), has\_car(A, C), has\_car(A, D), shape(C, triangle, 1).**

With this we can correctly predict all positive examples as correct, and only wrongly predict one negative example as correct.

We can also look at the bottom clauses, i.e. the clauses that contain all literals that we have allowed through the background knowledge. You can do so by calling **show(bottom)**. The bottom clause for the first example looks as follows:

eastbound(A) :-

has\_car(A,B), has\_car(A,C), has\_car(A,D), has\_car(A,E), short(E), short(C), closed(C), long(D), long(B), open\_car(E), open\_car(D), open\_car(B), shape(E,rectangle), shape(D,rectangle), shape(C,rectangle), shape(B,rectangle), wheels(E,2), wheels(D,3), wheels(C,2), wheels(B,2),

load(E,circle,1), load(D,hexagon,1), load(C,triangle,1), load(B,rectangle,3).

We can see that it does indeed contain all literals that were also included in the rule that Aleph has found during learning.

# Recursion

## Learn the member predicate:

The member predicate contains the following files:

Just like with the train example, we now lets Aleph learn a theory by first loading all files with read\_all(filename) and then calling induce.