Knowledge Representation and Learning UE (351.010)
Learning in Logic Assignment

Johannes Fürnkranz, Van Quoc Phuong Huynh, Josef Küng

05.2024

The aim of this assignment is to help students get familiar with Inductive Logic Programming (ILP) in general. You will practise with particular two implementations Aleph and Popper. Aleph is an advanced ILP system with many different options. Popper is a newly developed ILP system based on a scientific publication at `https://link.springer.com/article/10.1007/s10994-020-05934-z`.

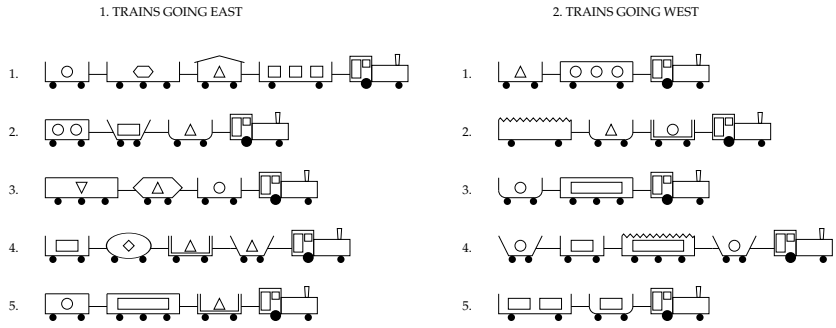
1 Preparation

1. Aleph:
   (a) Download and install Aleph. In Moodle, we provide you with a slightly modified version 'aleph_swi.pl' that should work with recent versions of SWI-Prolog.[1]
   (b) Take a look a the manual at `http://www.cs.ox.ac.uk/activities/programinduction/Aleph/` and familiarize yourself with the basic workings of the system.
   (c) You can find a set of simple examples at `http://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/misc/examples.zip`. Download and save them, you will need them for the subsequent tasks.
2. Popper: Follow the link `https://github.com/logic-and-learning-lab/Popper` for installation and usage instructions. However, since Popper is supported fully on Linux-like operating systems (OS) only, there are two optional installations for you.
   (a) If your machine is with Linux-like OS, it's recommended to follow exactly instructions from the above link. With this option, Popper can work with differnt SAT solvers, e.g. with "UWrMaxSat", "WMaxCDCL" solves, Popper may find better solutions.
   (b) Otherwise, i.e. Windows OS, you ignore the 'pysat' installation and use a slightly modified version of Popper (Popperw) uploaded on the Moodle of the course (**do not install Popper from the above link**). With this option, Popper only works with "clingo" SAT solver.

[1] Note that Aleph is also available as a SWI-Prolog package, which can be installed via `pack_install(aleph)` and activated via `use_module(library(aleph))`. This uses a slightly different interface (cf. `https://www.swi-prolog.org/pack/file_details/aleph/prolog/aleph.pl`). However, we could not get it to work. Let us know if you have more luck...

2 Getting Started

Machine Learning pioneer Ryszard Michalski has come up with the following by now classical example for a task that requires first-order logic:



The task is to find a rule that discriminates eastbound trains from westbound trains, based on the size and type of cars (number of wheels) and what loads they carry. As trains have different numbers of cars, it is hard to represent this in a conventional, propositional learning setting.

You can find this example encoded for Aleph in the `examples/trains` directory. Load Aleph (`consult('aleph-swi')`), change into the example directory (e.g., `chdir(examples/trains)`), and load all files with `read_all(train)`. You will probably get quite a few warnings, which you can ignore (but you should not get any errors).

1. Look at how the positive and negative examples and, in particular, the background knowledge is encoded in the files `train.[f,n,b]`. Try to find a solution to the problem by hand.

2. Use Aleph to learn a solution by calling `induce`. You can now look at the learned rule(s) with `show(theory)`, at the examples with `show(pos)` and `show(neg)`. Familiarize yourself with other options of `show/1` as well.

   Inspect the learned rule, interpret it, and compare it to what you have got in (a).

3. Look at the bottom clause (`show(bottom)`), try to interpret it, and show that the learned rule subsumes the bottom clause.

For Popper, in `examples/zendo1`, you find an example of Zendo game encoded for Popper. Zendo is a logic inductive game, the rule and information for the game can be found at `https://en.wikipedia.org/wiki/Zendo_(game)`. You will see three files "exs.pl", "bk.pl", and "bias.pl" which respectively defines positive/negative examples, the information about the problem, and the search space for Popper. The details for usage can be found in the provided link.

Exercise 1 *(30 points)*

1. *Encode the above problem of train discrimination for Popper, compare Popper's solution with Aleph's. (15 points)*

2. *Encode the above example of Zendo game (`examples/zendo1`) for Aleph, compare Aleph's solution with Popper's. (15 points)*

3 Recursion

In the folders `examples/recursion` (of Aleph) and `examples/length` (of Popper) you can find files that allow the ILP systems to learn recursive rules for `member/2` and `length/2` respectively.

Exercise 2 *(35 points)*
Choose two out of the following four recursive list processing predicates `sum([4, 2, 6], 12)`, `max([4,2,6], 6)`, `element_at(2, [a,b,c], b)`, `last([b,a,c], c)`*, encode the chosen two for both Aleph and Popper. Then compare their corresponding solutions.*

4 Generalization

At `https://www.doc.ic.ac.uk/~shm/applications.html` there is a list of simple sample applications where ILP approaches such as Aleph have been successfully used. You choose one application from the list and stick with it for exercise 3.

Exercise 3 *(35 points)*

1. *Learn a rule set with Aleph. For the moment only use the* `train`*-data and the background knowledge. You do not need to change any of the examples, but the naming and the declarations in the background knowledge file have to be adapted (e.g., make use of mode restrictions in order to restrict the search space). Manually verify and interpret found rules or part of them if the rule set is large. (15 points)*

2. *The files also contain positive and negative test examples. Test the theory on the provided test set (*Hint*: You have to test twice, once on the positive and once on the negative examples). Report the error rate on the test set (combined for both sets, i.e., the percentage of all examples that have been misclassified). (5 points)*

3. *Learn a rule set with Popper and compare the Popper's rule set with that of Aleph. (15 points)*

5 Try something interesting*

Find an interesting and new application of ILP and try to solve it by Aleph or Popper. You should register your choice in the shared worksheet "Register list" (tab "Assignment 2"). Some ideas:

- something related to your first project
- something related to knowledge bases and the Semantic Web (which will follow later in the course)
- something related to games (e.g., learning a simple but imperfect chess playing strategy from the other chess database mentioned above)
- etc.

Note that this may turn out to be harder than you think, so try to aim for a simple task. Try hard to succeed, but it is also o.k. to provide a report on a failure (why didn't it work as expected).

6 Submission

You only report your solution for the exercises. Section 5 (with *) is optional for a bonus of maximum 15 points.

Please upload your submission on Moodle until **June 9th, 2024, 23h:59'**. Any submissions after the deadline will be penalized 20% from the total mark of this assignment. All files must be compressed in a .zip file with the file name in format "**group_xx.zip**", e.g. "group_01.zip". The package includes:

1. All files to be processed by Aleph and Popper
2. A presentation slides file
3. A report for your solutions