First contact with *Lear*

Dominique Bouthinon & Henry Soldano

L.I.P.N, UMR-CNRS 7030, Université Paris-Nord,
93430 Villetaneuse, France
dominique.bouthinon@lipn.univ-paris13.fr
henry.soldano@lipn.univ-paris13.fr

1 Introduction

Lear is a relational concept-learning system implemented in swi-prolog (`http://www.swi-prolog.org/index.txt` [1]). It learns a set of first order clauses from ambiguous examples represented as clausal theories. *Lear* is based on theoretical ideas described in [2].

2 Directories of *Lear*

Once you have downloaded (and uncompressed) *Lear* you have a directory *lear* containing three directories:

- *programs* containing the prolog files implementing *Lear*, and the python scripts used to perform miscellaneous learnings,
- *examples* containing the example files,
- *documentation* containing the documentation files.

3 Configure your system

Lear can be executed on every system that can run swi-prolog. To configure and execute properly *Lear* you must:

1. go to the *programs* directory and make executable all files **.pl* and **.sh*,
2. install swi-prolog and set the configuration file of your system with the path to swi-prolog (`http://www.swi-prolog.org/index.txt`),
3. set an environment variable *LEAR_DIR* with the path to the directory *programs* mentioned at the previous section. As an illustration assume that the path to *lear* is `usr/local/lear` then set the variable as follows: `export LEAR_DIR=/usr/local/lear/programs`,
4. edit the file *topLear3.pl* located in the directory *programs* and replace the first line `#! /usr/local/bin/swipl` by the access to the place where you have installed *swipl*.

4 Representing ambiguous examples and background knowledge in *Lear* format

We give hereafter an illustration of the representation of the background knowledge and the ambiguous examples with a problem where we want to learn the definition of the *father* relation.

4.1 Background knowledge

The background knowledge, shared by all examples, is put in a file *father.int_bkg*.

```
      % specify the target relation
target_relation(father(_X,_Y)).
      % types of the arguments of the target relation
target_mode(father(person, person)).

/* declaration of the types of the arguments of the predicates
   used to describe the examples and the hypotheses */

mode(person(name)).
mode(parent(name, +name)).   (+ indicates the second argument person
                                   must be instantiated for Lear to consider this atom)
mode(male(name)).
mode(female(name)).
mode(age(name, year)).

               /* parameters relative to variables appearing in the hypotheses */
      % all variables of type person will be under OI constrainst
types_under_oi([name]).
      % variables of types appearing in the following list can be replaced by constants
types_instantiable([year]).

     /* all clauses appearing in the list are irrelevant (here empty list) */
irrelevant([]).

/* let empty */
non_abducible_predicates([]).

     /* clauses describing background knowledge */
background_knowledge(
   [

          % noone is parent of himself
     <- parent(X,X),
          % Y cannot be parent of X whether X is parent of Y
     <- parent(X,Y) and parent(Y,X),
```

```
    person(X) <- parent(X,Y),
    person(Y) <- parent(X,Y),
    male(X) or female(X) <- person(X)

      % noone is both male and female,
    <- male(X) and female(X)

  ]
 ).
```

4.2 Ambiguous examples

The file *father.int* contains the ambiguous examples.

```
 /* ========================== POSITIVE EXAMPLES ============================ */

target(1, pos, father(john, mary)).
pos(1, [male(john), female(mary), <- parent(mary, john)]).
domains(1, pos, [person([john, mary])]).

target(2, pos, father(david, steve)).
pos(2, [parent(david, steve), male(david), male(steve)]).
domains(2, pos, [person([david, steve])]).

/* ========================== NEGATIVE EXAMPLES ============================ */

target(1, neg, father(katy, ellen)).
neg(1, [parent(katy, ellen), female(ellen)]).
domains(1, neg, [person([katy, ellen])]).

target(2, neg, father(oliver, philipp)).
neg(2, [male(oliver), male(philipp)]).
domains(2, neg, [person([oliver, philipp])]).
```

Each ambiguous example is described by three predicates $target/3$, $pos/2$ (or $neg/2$ for the negative examples) and $domains/3$. The first argument of each predicate is a number that identify the example, and the second argument of *target* and *domains* identify whether this is a positive or a negative example. Note that a positive and a negative example can have the same number, they will be discriminated by their second argument.

The third argument of *target* describes a ground instance of the target relation. This instance is assumed true whether the example is positive, false whether it is negative. As an illustration *father(john, mary)* associated with the positive

example 1 means that *john* is *mary*'s father, while *father(katy, ellen)* associated with the negative example 3 means that *katy* is *ellen*'s father.

The second argument of *pos* (*neg* for a negative example) is a clausal theory stipulating what is known from the point of view of the considered example. In example 1, $[male(john),\ female(mary), \leftarrow parent(mary, john)]$ means that *john* is a man, *mary* is a woman and *mary* is not a parent of *john*. There is an ambiguity because the examples are incomplete: every unknown fact is indeterminate. So we do not know neither whether *john* is a parent of *mary*, nor whether *john* is a also woman, nor whether *mary* is a also a man. The role of the background knowledge provided by *parent.int_bkg* is to limit this ambiguity (this background theory is shared by all the examples). For instance, the clause $\leftarrow male(X)\ and\ female(X)$ allows *Lear* to establish that *john* is not a woman and *mary* is not a man. The clausal theory in *pos* (*neg*) describing an example in the predicate *pos* or *neg* may contains variables which will be implicitly ground with the constants declared in the *domains* predicate (see next paragraph). Also notice the possibility to use general clauses of the form $a_1\ or \ldots or\ a_m \leftarrow b_1\ and \ldots and\ b_n$ in the clausal theories associated with examples and background knowledge.

The third argument of *domains* is a list containing the types of each object involved in the example. For instance, $[name([john, mary]), age([13, 45])]$ means that *john* and *mary* are of the type *name*, and that 13 and 45 are of the type *age*. Note that objects that appear neither in *target* nor in *pos* (*neg*) can be referenced in *domains*. This means that we know nothing on such objects. For instance, assume the declaration $domains(1, pos, [name([john, mary]), pet([titi])])$ associated with example 1. It means that example 1 also contains an object *titi* of type *pet*, but we know nothing about it. Nevertheless, *titi* will be considered by *Lear* in each predicate that accepts a *pet* as argument.

5 Learn with *Lear*

You can learn with *Lear* whenever you have the ambiguous examples file (*father.int*) and the background knowledge file (*father.int_bkg*).

To make a simple learning use the *learn.sh* script:

```
learn.sh father father
```

The first argument is the name (without extension) of the file containing the ambiguous examples, the second one is the name (without extension) of the file containing the background knowledge. Here the two names are equals, but they could be different. The results of the learning are displayed at the screen:

```
================== FINAL SET OF RULES ====================
father(A,B) :- parent(A,B), male(A).
end of learning.
elapsed time : 0 h 0 m 0 s
```

The script *learn.sh* contains two declarations and two instructions:

```
fileEx=$1                              # prefix of the file containing
                      # the ambiguous examples (<file>.int)
fileBkg=$2                              # prefix of the file containing
                      # the background knowledge (<file>.int_bkg)

topLear3.pl -e $fileEx -B $fileBkg   # build the extentional examples
topLear3.pl -f $file                 # learn
```

`topLear3.pl` is the main prolog program that handles the *Lear* options. Notice the two steps in the script:

1. `topLear3.pl -e $fileEx -b $fileBkg`. The option $-e$ means that *Lear* must convert each intentional ambiguous example contained in *fileEx.int* in an extensional ambiguous example represented as a set of ground clauses. The option *-B* declares that the background knowledge used in this step is in the file *fileBkg.int_bkg*. Two files are created at the end of this step: *fileEx.pl* containing the extensional ambiguous examples and *fileEx.bkg* containing the background knowledge in a new format.
2. `topLear3.pl -f $fileEx` learns the target concept from the two files created in 1.

Notice that the first step, which may cost times, must be achieved only once. So the second step (learning phase) can be done several times, with different options, from the same files *fileEx.pl* and *fileBkg.bkg*.

To display the available options associated with `topLear3.pl` enter:
topLear3.pl -h
The following options are displayed:

```
-B <file>     declare the file containing the background knowledge
-c <value>    make a <value> cross validation
-d <value>    stop the search of a new clause whenever <value>%
              of the remaining positive examples are covered
-e <file>     build extensional examples from intentional ones
              contained in <file>.int (option -b is mandatory)
-f <file>     learn from examples contained in <file>.int and
              background knowledge contained in <file>.bkg
-l            only consider linked clauses in hypotheses
-w <value>    declare the width of the beam (3 by default)
-p <value>    accept that p% of the negative examples are not compatible
              with the hypothesis
-v <value>    verbosity level (0 (default), 1, 2 or 3)
```

You can use these options to make finer learning sessions. As an illustration
`topLear3 -f father  -c 10  -w 5  -v 2`
runs a learning session from the examples contained in the file *father.pl* and

the background knowledge contained in the file *father.bkg*. The learning process consists in a 10 cross validation, each step lauching a beam search with a beam of lenght 5. The level of verbosity is set to 2.

In the *programs* directory you have several documented python 3.2 scripts that you can use to make specific learning sessions.

References

1. Wielemaker, J.: An overview of the SWI-Prolog programming environment. In Mesnard, F., Serebenik, A., eds.: Proceedings of the 13th International Workshop on Logic Programming Environments, Heverlee, Belgium, Katholieke Universiteit Leuven (december 2003) 1–16 CW 371.
2. Bouthinon, D., Soldano, H., Ventos, V.: Concept learning from (very) ambiguous examples. In Perner, P., ed.: MLDM. Volume 5632 of Lecture Notes in Computer Science., Springer (2009) 465–478