Department of Computer Science
Technical University of Cluj-Napoca

Introduction to Artificial Intelligence

Adrian Groza, Radu Razvan Slavescu and Anca Marginean

Contents

Lab 7

First Order Logic

It might be tedious to model reasonably large problems in Propositional Logic (PL from now on), as you need to write a huge amount of propositions to encode the available knowledge. First Order Logic (FOL in short) offers you some more operators, which allow a more compact way of writting the same knowledge.

Learning objectives for this week are:

1. To get familiar with encoding knowledge in FOL for Prover9
2. To see how you can deal with ordered sets in FOL, without using the set of natural numbers $\mathbb{N}$

One drawback of Propositional Logic it that it does not allow you to write general enough sentences. For example, you cannot simply write something like "If there is a pit in square (i, j), then you can feel breeze in square (i, j)". If both i and j are in the range 1..4, Propositional Logic will force you to write 16 sentences of the type $P23 \rightarrow B23$. This makes the knowledge base to grow rapidly and to become unmanageable.

First Order Logic addresses this by allowing the $\forall$ (`forall`) quantifier. All the PL sentences in the example above could now be collapsed into just one FOL proposition:

$$\forall\ i, j : pit(i, j) \rightarrow breeze(i, j)$$

Here, *pit* and *breeze* are two predicates, each of arity 2, which are true iff the room number (i, j) contains a pit or a breeze respectively. The arguments i and j are called *bound* variables. They are assumed to belong to a known domain (for example, here the domain is the set $\{1, 2, 3, 4\}$ for each of them).

7.1 First Example: Socrates in First Order Logic

We will formalize once again the knowledge about Socrates in Lab #5, but this time, we'll do it in FOL. Please recall that:

1. Socrates is a human.
2. All humans are mortal.

Our target is to prove that Socrates is a mortal.

First, we need to write down the constants involved, which reffer particular objects in the domain of discourse. We have here just one person, namely Socrates, so we use all in all one constant: `Socrates`.

Then, we need to identify the predicates we might need. One of them would be predicate `human(x)`, which is true iff the object x is a human being. Another predicate which we will need is `mortal(x)`, which is true iff its argument x is mortal.

In both cases, we assume the domain of value for x is known. For our example, this could be the set of all creatures who are alive. Hence, `human(Socrates)` is true, but `human(Viserion)` [1] or `human(Incitatus)` [2] are false. Similarly, `mortal(Socrates)` is true (as we will prove soon), while `mortal(Zeus)` is arguably false.

Given these, the Prover9 implementation is given in listing 7.1.

Listing 7.1: Socrates KB in FOL.

```
set(binary_resolution).
set(print_gen).

% 1. Socrates is a human.
% 2. All humans are mortal.
% 3. Prove that Socrates is a mortal.

% constant: Socrates
% predicate human(x): x is a human
% predicate mortal(x): x is mortal


formulas(assumptions).
 human(Sokrates).
 human(x) -> mortal(x).
end_of_list.

formulas(goals).
 mortal(Sokrates).
end_of_list.
```

Exercise 7.1 *Run the example in listing 7.1, which is provided in file* `socratesFOL.in`*. Look at the output and try to understand the transformations performed by Prover9 over the input before the search for a proof begins. Carefully identify the original form and the transformed form for implications.*

Now focus on the PROOF section and follow, step by step, the produced proof. Try to understand the meaning of the following marks: `assumption, clausify, goal, resolve, deny`*.*

There are two conventions for naming the constants and variables involved in the predicates. The default convention states that the names of the variables in clauses start with (lower case) 'u' through 'z'. But if you set the flag called `prolog_style_variables`, they will start with (upper case) 'A' through 'Z', as in Prolog. Setting the flag is achieved by adding the following line to your input file: `set(prolog_style_variables).`

We will use the Prolog convention once, in the next example. Then, for the rest of this lab, we will stick with the former convention.

7.2 FDR goes to war in FOL

We try now to represent a slightly larger knowledge base in FOL: the statements about FDR in lab #5. We know that:

[1] One of the dragons of Daenerys Targaryen from The Game of Thrones

[2] The horse of the Roman emperor Caligula, whom he appointed a priest

1. FDR is a politician.
2. A politician could be an isolationist or an interventionist (maybe both!).
3. An interventionist would declare war.
4. If US is under attack, even an isolationist would declare war.
5. US is under attack.

We employ the following set of constants and predicates:

- constant: `fdr` (meaning FDR)
- constant: `country_us` (meaning US)
- predicate: `under_attack(x)` (meaning country x is under attack)
- predicate: `politician(x)` (meaning x is a politician)
- predicate: `interventionist(x)` (meaning x is an interventionist)
- predicate: `isolationist(x)` (meaning x is an isolationist)
- predicate: `declare_war(x)` (meaning x will declare war)

The Prover9 implementation is given in listing 7.2.

Listing 7.2: FDR declares war - FOL version.

```
%
% FDR is a politician.
% A politician could be an isolationist or an interventionist (maybe both!).
% An interventionist would declare war.
% If US is under attack, even an isolationist would declare war.
% US is under atack.
%

%
% constant: fdr (meaning FDR)
% constant: country_us (meaning US)
% predicate: under_attack(x) (meaning country x is under attack)
% predicate: politician(x)  (meaning x is a politician)
% predicate: interventionist(x) (meaning x is an interventionist)
% predicate: isolationist(x) (meaning x is an isolationist)
% predicate: declare_war(x) (meaning x will declare war)
%



formulas(assumptions).
 under_attack(country_us).
 politician(fdr).
 politician(x) -> isolationist(x) | interventionist(x).
 interventionist(x) -> declare_war(x).
 isolationist(x) & under_attack(country_us) -> declare_war(x).
end_of_list.

formulas(goals).
 declare_war(fdr).
end_of_list.
```

Exercise 7.2 *Run the example in the file* ***fdr-warFOL.in****, which contains the example in listing 7.2 and repeat the tasks specified in exercise 7.1.*

In listing 7.3 you can see the same problem modeled using the backward implication operator (which is similar to the `:-` symbol in Prolog). Please note the name of variables and constants, which follow the Prolog convention.

Listing 7.3: FDR declares war - FOL version with backward implications.

```
%
% FDR is a politician.
% A politician could be an isolationist or an interventionist (maybe both!).
% An interventionist would declare war.
% If US is under attack, even an isolationist would declare war.
% US is under atack.
%


%
% constant: fdr (meaning FDR)
% constant: us (meaning US)
% predicate: under_attack(x) (meaning country x is under attack)
% predicate: politician(x)  (meaning x is a politician)
% predicate: interventionist(x) (meaning x is an interventionist)
% predicate: isolationist(x) (meaning x is an isolationist)
% predicate: declare_war(x) (meaning x will declare war)
%


formulas(assumptions).
 under_attack(us).
 politician(fdr).
 (isolationist(P) | interventionist(P)) <- politician(P).
 declare_war(X) <- interventionist(X).
 declare_war(X) <- isolationist(X) & under_attack(us) .
end_of_list.

formulas(goals).
 declare_war(fdr).
end_of_list.
```

Now, listing 7.4 shows you how some predefined operators of Prover9 could be redefined. We did this with 3 operators, in order to make them look similar with their Prolog counterparts.

For instance, the line

```
redeclare(backward_implication, "COLON_MINUS").
```

means the usual operator for the backward implication, i.e., $\leftarrow$, is replaced by the string `"COLON_MINUS"`, which resembles the `:-` construction in Prolog (simply redefining $\leftarrow$ as `:-` failed as the colon character is used in Prover9 for the cons operator). Now, instead of writting `a(x) <- b(x)`, we'll have to write `a(x) COLON_MINUS b(x)`.

In line with this, the disjunction operator (`|`) was replaced "SEMICOLON" and the conjunction operator (`&`) became "COMMA".

Listing 7.4: FDR declares war - Prolog-style syntax.

```
%
% FDR is a politician.
% A politician could be an isolationist or an interventionist (maybe both!).
% An interventionist would declare war.
```

```
% If US is under attack, even an isolationist would declare war.
% US is under atack.
%


%
% constant: fdr (meaning FDR)
% constant: us (meaning US)
% predicate: under_attack(x) (meaning country x is under attack)
% predicate: politician(x)  (meaning x is a politician)
% predicate: interventionist(x) (meaning x is an interventionist)
% predicate: isolationist(x) (meaning x is an isolationist)
% predicate: declare_war(x) (meaning x will declare war)
%

set(prolog_style_variables).
redeclare(backward_implication, "COLON_MINUS").
redeclare(disjunction, "SEMICOLON").
redeclare(conjunction, "COMMA").


formulas(assumptions).
 under_attack(us).
 politician(fdr).
 (isolationist(P) SEMICOLON interventionist(P)) COLON_MINUS politician(P).
 declare_war(X) COLON_MINUS interventionist(X).
 declare_war(X) COLON_MINUS isolationist(X) COMMA under_attack(us) .
end_of_list.

formulas(goals).
 declare_war(fdr).
end_of_list.
```

Exercise 7.3 *Run the example in the file* `fdr-warFOL-Prolog.in`*, which contains the example in listing 7.4 and make sure you got one proof. Extract the clauses in the* `assumptions` *part, replace the three string-type operators with their Prolog versions (e.g.,* `COLON_MINUS` *with* `:-` *and alike) and save the them in a file called* `fdr.pl`*. Could you load the* `fdr.pl` *file in Prolog and ask if FDR will declare war? Will you get the same answer as in Prover9? Why (not)?*

7.3 Alligator and Beer: finding models in FOL

The problem we will model is a variation of the puzzle widely known as Einstein's riddle. Surely Uncle Google does the best he can to help you find the solution. It's time to ask Mace4 to do the same. For this instance of the problem, the latter is probably faster than the former (both implementation and running time included).

There are five houses in a row: a, b, c, d and e. Each has a different color (amber, beige, cyan, denim, emerald) and an owner of different nationality. The owners have differrent cars, pets and preffer different drinks. We know that:

1. The Austrian lives in the amber house.
2. Cider is drunk in the middle house.
3. The Belgian owns the bulldog.
4. The Czech lives in the first house on the left.

5. The man who owns a Dacia lives next to the man with the eagle.
6. The Czech lives next to the denim house.
7. The Bugatti owner has a cat.
8. The person in the beige house drives a Cadillac.
9. Advocaat is drunk in the cyan house.
10. The Dane drinks eiswein.
11. The Estonian drives an Edonis.
12. The Cadillac is always parked in front of the house next to the one with a donkey.
13. The cyan house is immediately to the right of the emerald house.
14. The Aston Martin owner drinks daiquiri.

We want to know:

- Where is the alligator?
- Who drinks beer?

Exercise 7.4 *Can you answer the two questions?*

We want to model this problem in FOL, then to ask Mace4 to find a model. To this end, we will use 5 constants, namely {`a, b, c, d, e`} to denote each of the five houses. For the other information, we are going to use several predicates of arity one. Hence, `amber(x)` will mean that house `x` has color amber (where `x` is either `a, b, c, d` or `e`), while `austrian(x)` will mean the person in house `x` is of Austrian nationality.

We must model the common sense knowledge about this problem (e.g., to teach Prover9 what "neighbor" means), then to tell that each house has at least one color, but no more. In the first attempt, we will do this without using numbers, just FOL constructs.

Then, we will encode all clues using the predicates mentioned above.

Eventually, we will save everyting in the file `alligatorBeer1.in` and call Mace4 with command `mace4 -c -f alligatorBeer1.in | interpformat`. The model produced should give us the solution.

Exercise 7.5 *Using a predicate `differentFrom(x,y)`, write down enough sentences to express the idea that `a,b,c,d,e` are distinct from one another. You should come up with a couple of clauses like `differentFrom(a,b)`. Then, write a clause to say the "differentFrom" relation is symmetrical.*

Exercise 7.6 *Using a predicate `rightneighbor(x,y)`, write down enough sentences to express the fact that "b is immediately to the right of a" for every pair in `a,b,c,d,e`. You should come up with a couple of clauses like `rightneighbor(a,b)`. Then, write clauses to say that "it is not the case that `rightneighbor(a,b)`". Complete the rest of the exercises and try to figure out whether the clause `rightneighbor(a,b)` is necessary.*

Exercise 7.7 *Define a predicate `neighbor(x,y)` which says that you are the neighbor of someone either if you live just to his right or he lives just to your right (i.e., you live just to his left).*

Exercise 7.8 *Write down clauses to express that each house has at least one nationality, pet, drink, color, car. E.g.,* `austrian(x) | belgian(x) | czech(x) | dane(x) | estonian(x).`

Exercise 7.9 *Specify that each property applies to at most one house. How can you do that?*

Exercise 7.10 *Now try to model the rest of the clues. You may want to write down clauses like* `austrian(x)` $\leftrightarrow$ `amber(x).` *Call Mace4 to generate the models (hopefully there is only one of them) and explore them in order to extract the required answers.*

7.4 Solutions to exercises

Hint for exercise 7.3: think about Horn clauses.

Solution for exercise 7.4: The Estonian. The Czech.

Hint for exercise 7.9: try to model this information in the following manner: "if we have two houses x and y and both are inhabited by an Austrian, then x and y must be the same house".

Solution for exercise 7.5 - 7.10.
See file `alligatorBeer1.in`.

Bibliography