Department of Computer Science
Technical University of Cluj-Napoca

Introduction to Artificial Intelligence

Adrian Groza, Radu Razvan Slavescu and Anca Marginean

Contents

Lab 5

Inference in Propositional Logic

This lab will begin presenting how proofs can be build automatically, starting from a set of axioms assumed to hold. Programs which achieve this are called automated theorem provers, or just provers for short. The one we will use to illustrate the idea is called Prover9 [**?**].

Learning objectives for this week are:

1. To see the structure of a Prover9 input file corresponding to a problem
2. To understand the output returned by the prover
3. To be able to explain, step by step, the proof obtained

5.1 Getting started with Prover9 and Mace4

Prover9 searches for proofs; Mace4, for counterexamples. The sentences they operate on could be written in propositional, first-order or equational logic. This first lab is concerned with sentences written solely in propositional logic. Here, we have true or false propositions, like P or Q, but no sentences of type $\forall x P(x)$.

5.1.1 Installing Prover9 and Mace4

Download the current command-line version of the tool (LADR-2009-11A), which is available at https://www.cs.unm.edu/ mccune/mace4/download/LADR-2009-11A.tar.gz. Unpack it, change directory to LADR-2009-11A, then type `make all` and follow the instructions on the screen.

Do not forget to add the `bin` folder to you `PATH`, such that you should be able to start Prover9 by simply typing `prover9` in a terminal, regardless your current directory.

5.2 First Example: Socrates

Let us assume the following hold:

1. If someone's name is Socrates, then he must be a human.
2. Humans are mortal.
3. The guy over there is called Socrates.

Now try to prove that Socrates is a mortal.

The assumptions in this tiny, well known knowledge base could be represented formally using different types of logics. Right now, we employ the simplest one, called Propositional Logic, which comprises variable names for sentences plus a set of logical operators like $\wedge, \vee, \neg, \rightarrow, \leftrightarrow$.

In our example, we'll use the following set of sentences/propositions:

S: The guy over there is called Socrates

H: Socrates is a human

M: Socrates is mortal

The Prover9 input file containing the implementation is presented in Listing **??**.

Listing 5.1: Knowledge on Socrates' mortal nature

```
1  assign(max_seconds, 30).
2  set(binary_resolution).
3  set(print_gen).
4
5  % 1. If someone's name is Socrates, then he must be a human.
6  % 2. Humans are mortal.
7  % 3. The guy over there is called Socrates.
8  % 4. Prove that Socrates is a mortal.
9
10 % S: the guy is called Socrates
11 % H: Socrates is a human
12 % M: Socrates is mortal
13
14
15 formulas(assumptions).
16  S.
17  S -> H.
18  H -> M.
19 end_of_list.
20
21 formulas(goals).
22 M.
23 end_of_list.
```

Exercise 5.1 *Type* `prover9 -f socrates.in` *in order to run Prover9 with* `socrates.in` *as an input file. Redirect the output to* `socrates.out` *and examine it. Have you obtained a proof of your goal? (the text* `THEOREM PROVED` *in the file/on the screen should indicate this).*

5.3 Input File Explained

The input files for Prover9 comprise some distinct parts. In this Section, we will explain the meaning of each part in the input file for the example in Listing **??**.

The first part contains some flags. For example, `assign(max_seconds, 30)` limits the processing time at 30 seconds, while `set(binary_resolution)` allows the use of the binary resolution inference rule (`clear(binary_resolution)` would do the opposite). Writing `set(print_gen)` instructs Prover9 to print all clauses generated while searching for the proof. For now, we don't focus on flags, just assume they are given in each exercise. Further, when we might get to fine tuning them, we'll use at the comprehensive description provided in the All Prover9 Options of the online manual (https://www.cs.unm.edu/ mccune/mace4/manual/2009-11A/).

Comment lines start with the % symbol.

The part between `formulas(assumptions)` and the corresponding `end_of_list` contain the actual knowledge base, i.e., the sentences which are assumed to be true. Sentence S means, as already mentioned, that the guy is called Socrates, while $S \rightarrow H$ says that if you are Socrates,

then you are a human. You can use − for negation, & for logical conjunction and | for logical disjunction. Each sentence (called formula) ends with a dot.

The part between `formulas(goals)` and the corresponding `end_of_list` state the goal the prover must demonstrate, namely M in our case.

5.4 Output File Explained

The output file produced (i.e., `socrates.out`) starts with some information on the running process, a copy of the input and a list of the formulas that are not in clausal form. Then, these clauses are processed according to the algorithm for transforming them into the Clausal Normal Form and we get:

```
4 S.  [assumption].
5 -S | H.  [clausify(1)].
6 -H | M.  [clausify(2)].
7 -M.  [deny(3)].
```

Please remember the equivalence between $P \to Q$ and $\neg P \vee Q$, which is used for instance in line 5. This is what `clausify` does. One should also notice in line 7 the goal has been added in the negated form $\neg M$. This is called *reductio ad absurdum*: if one both accepts the axioms and denies the conclusion, then a contradiction will be inferred. Prover9 actually searches for such a contradiction by repeatedly applying inference rules over the existing clauses till the empty clause is obtained.

The SEARCH section of the output lists all clauses inferred during search for the proof.

The PROOF section shows just those which actually helped in building the demonstration. For the example above:

```
1 S -> H # label(non_clause).  [assumption].
2 H -> M # label(non_clause).  [assumption].
3 M # label(non_clause) # label(goal).  [goal].
4 S.  [assumption].
5 -S | H.  [clausify(1)].
6 -H | M.  [clausify(2)].
7 -M.  [deny(3)].
8 H.  [resolve(5,a,4,a)].
9 -H.  [resolve(7,a,6,b)].
10 $F.  [resolve(9,a,8,a)].
```

Lines 1-4 are the original ones, 5 and 6 are the clausal forms of 1 and 2, while 7 is the goal denial. Line 8 shows how the resolution is applied over clauses 5 and 4 (Prover9 calls this "binary resolution"). Propositional resolution inference rules says that if we have $P \vee Q$ and $\neg Q \vee R$, we can infer $P \vee R$. The first literal in clause 5 ($-S$, hence the index a) and the first literal in clause 4 are "resolved" and the inferred clause is H. If P is absent, this inference rule is called "unit deletion", or, if R is absent, "back unit deletion". Line 10 shows the derived contradiction.

Exercise 5.2 *Add a clause specifying that Socrates is a philosopher (use sentence symbols P for "philosopher"). Run Prover9 again and take a look at the generated clauses and at the produced proof. Is the set of generated clauses different in this case? How about the proof?*

5.5 A more complex example: FDR goes to war

Given:

1. If your name is FDR, then you are a politician.
2. The name of the guy addressing the Congress is FDR.
3. A politician can be isolationist or interventionist [1]
4. If you are an interventionist, then you will declare war.
5. If you are an isolationist and your country is under attack, then you will also declare war
6. The country is under attack.

we intend to prove that war will be declared.

Exercise 5.3 *Implement the knowledge above, save it in the file* ***fdr-war.in*** *and use Prover9 to show that war will be declared. Then, try to prove that war will not be declared. Have you succeeded both times?*

Exercise 5.4 *Let us add now the following piece to our knowledge base: the country is not under attack (* ***-attack****). The knowledge base contains a contradiction now. Let's try to show that war will be declared. Then, try to prove that war will not be declared. Have you succeeded both times?*

As mentioned earlier, Prover9 seeks for a contradiction in the body of clauses which incorporates the initial knowledge base and the negated conclusion. If the initial knowledge base is consistent (with no contradiction), then the only source of contradiction would be the negate conclusion which has just been added. But, if the initial knowledge base already contains a contradiction, then Prover9 will eventually derive it, no matter what the added conclusion might be. So, in this case, everything would be provable. Mace4 can help to detect if there exists at least one model of a knowledge base, which would mean it is not contradictory. For example, if no `-attack` is present in the knowledge base, we can comment out the goal line and do `mace4 -c -f fdr-war.in` and, if we get a model of the knowledge base, then we can be sure it contains no contradiction.

Listing 5.2: A model for FDR knowledge base example

```
interpretation( 2, [number=1, seconds=0], [

        relation(attack, [ 1 ]),

        relation(declare_war, [ 1 ]),

        relation(fdr, [ 1 ]),

        relation(interventionist, [ 0 ]),

        relation(isolationist, [ 1 ]),

        relation(politician, [ 1 ])
]).
```

In the output produced by Mace4, `relation(attack [1])` means that in the model built, `attack` has value TRUE (`[0]` would mean FALSE). The current model number is specified by `number`.

[1] Usually you can't be both, but some politicians can.

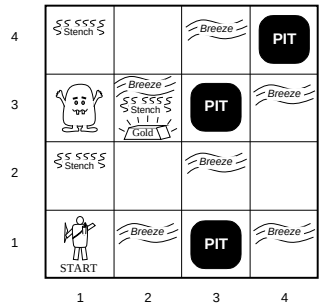

Figure 5.1: Wumpus world.

5.6 Wumpus World in Propositional Logic

A 4x4 grid contains one monster called wumpus and a number of pits [**?**]. Pits must be avoided and so should be the wumpus room as long as this is alive. Squares adjacent to wumpus are smelly, ans so is the square with the wumpus. Squares adjacent to a pit are breezy. Glitter iff gold is in the same square. Shooting kills wumpus if you are facing it. Shooting uses up the only arrow. Grabbing picks up gold if in same square. Releasing drops the gold in same square.

Exercise 5.5 *We will use the chessboard-like notation for this problem. Sentence* $-Sij$ *will mean there is no smell in the square on column* i*, line* j*. Let us assume the agent knows* $-S11$*,* $-S21$ *and* $S12$*. He also knows* $-B11$*,* $-B12$*,* $B21$*. Write down sentences to express "if there is a wumpus in (1,2), then there is a smell in (1,1)". Write down sentences to express "if I can smell something in (1,1), then there is a wumpus in (1,2) or in (2,1)". Can you prove the wumpus is in (1,3)?*

Exercise 5.6 *The agent is in the initial state (corner (1,1)) and perceives no smell and no breeze. No other perceptions are available, so you should drop all of them from your knowledge base. Can you prove now the following squares are safe for the agent to go there: (1,2); (2,1); (2,2)? Let's assume the agent goes in one of the squares he has proven to be safe and gets some perceptions from there. List the whole set of rooms which could be proven to be safe, given the new information available. Repeat the process until no safe room can be added.*

Exercise 5.7 *Write down a rule which says: a room is safe if there is no pit in it and there is no alive wumpus in it. Can you prove room (1,3) is safe, using only percepts obtained from safe rooms? Can you write enough rules for the agent to safely navigate among the whole castle, unti he gets the gold?*

5.7 Solutions to Exercises

Solution to exercise **??** Yes - P is also inferred, even if not relevant. No - the proof remains the same, as only some of the generated sentences are part of it and P is not among them.

Solution to exercise **??** You should be able to automatically proof the sentence "War will be declared", but not the sentence "War will not be declared". The message "SEARCH FAILED" and "Exiting with failure" should be seen on the screen. That means Prover9 has stopped searching and failed to find a proof.

Solution to exercise **??**

Listing 5.3: First wumpus problem

```
1  assign(max_seconds, 30).
2
3  formulas(assumptions).
4  -S11.
5  -S21. %colon 2, line 1
6  S12.
7  -B11.
8  B21.
9  -B12.
10 W11 -> S11.
11 W11 -> S12.
12 W11 -> S21.
13 W12 -> S12.
14 W12 -> S11.
15 W12 -> S22.
16 W12 -> S13.
17 W21 -> S21.
18 W21 -> S11.
19 W21 -> S22.
20 W21 -> S13.
21 W22 -> S22.
22 W22 -> S21.
23 W22 -> S12.
24 W22 -> S23.
25 W22 -> S32.
26 W13 -> S13.
27 W13 -> S14.
28 W13 -> S23.
29 W13 -> S12.
30 W31 -> S31.
31 W31 -> S21.
32 W31 -> S32.
33 W31 -> S41.
34 S12 -> W11 | W22 | W13 | W12.
35 end_of_list.
36
37 formulas(goals).
38 W13.
39 end_of_list.
```

Appendix A

Running latex

LATEX is a typesetting system suitable for producing scientific and mathematical documents. Three starting points are:

- Getting started with LaTeX (David R. Wilkins)
- LaTeX Tutorial (Jeff Clark)
- Very Brief Introduction to Latex by Radu Slavescu

Kile is one latex editor. If installed, just type `kile` in the terminal.
Two introductory tutorials on beamer are [**?**] and [**?**]. The beamer manual is:
Presentation template

Appendix B

Linux support

Becoming familiar with the Linux requires patience. You must have the desire to try and figure things out on your own, rather than having everything done for you. Two starting references are:

- Introduction to Linux A Hands on Guide, Machtelt Garrels
- Brief Synopsis of Linux by Radu Slavescu

Table **??** lists basic commands.

B.1 Assignment guide

This section should contain only code developed by you, without any line re-used from other sources. This section helps me to correctly evaluate your amount of work and results obtained. Including in this section any line of code taken from someone else leads to failure of IS class this year. Failing or forgetting to add your code in this appendix leads to grade 1. Don't remove the above lines.

From your final report, remove the text/examples/algorithms/rules/bibliographic references/etc - keep only your notes. If the documentation does not meet minimum standard for lisability and scientific discourse, it will be classified by the furious teaching assistant as unacceptable and therefore rejected.

Check list:

1. Your original code is included in the Appendix .
2. Your original code and figures are readable.

Table B.1: Quickstart linux commands.

Command	Meaning
pwd	display present working directory
ls	displays the files in the current working directory
cd	change the directoris
chmod +x	set a file as executable
man	read man pages of a command
ssh	connects to a secure shell

3. All the references are added in the Bibliography section.

4. All your figures are referred in text (with command ref), described in the text, and they have relevant caption.

5. The final documentation describes only your project. Don't forget to remove all tutorial lines in the template (like these one).

6. The main algorithm of your tool is formalised in latex in chapter **??**.