CMPT 411/721 - Knowledge Representation and Reasoning

Assignment 2

Due date: October 25, 2019

J.P. Delgrande
October 7, 2019

Important Note: Students must work individually on this and other CMPT 411/721 assignments. You may not discuss the specific questions in this assignment, nor their solutions with any other student. You may not provide or use any solution, in whole or in part, to or by another student.

You are encouraged to discuss the general concepts involved in the questions in the context of completely different problems. If you are in doubt as to what constitutes acceptable discussion, please ask!

Implementing the Fitting Operator [10 marks]

This question involves generalising forward chaining (bottom-up) Horn derivations to implement the Fitting Operator, as covered in class.

Recall that we had the following from class: A knowledge base KB consists of a set of rules of the form $a \Leftarrow a_1, \ldots, a_n$ where $n \geq 0$, a is an atom, and each a_i is either of the form p or $\sim p$, where p is an atom. By maintaining a set of conclusions C, and adding an atom p to C when p is known to be solved and $\sim p$ when p is known to be insoluble, the forward-chaining procedure can be extended to handle negation as failure. So, now we can add atoms of the form $\sim p$ to the set C of consequences, where $\sim p$ means that p cannot be proven. Here is the procedure:

A Bottom-up Procedure:

$C := \{\}$;
repeat
    either
        choose $r \in KB$ such that
            r is '$a \Leftarrow a_1, \ldots, a_m$' and
            $a_i \in C$ for all i, and $a \notin C$;
        $C := C \cup \{a\}$
    or
        choose a such that for every rule $a \Leftarrow a_1, \ldots, a_m$
            either for some a_i we have $\sim a_i \in C$
            or some $a_i = \sim g$ and $g \in C$
        $C := C \cup \{\sim a\}$
until no more choices

For full marks, your program should run in linear time, relative to the size of the knowledge base and your documentation should contain an argument as to why your program runs in linear time. See questions in the Brachman and Levesque text for Chapters 5 and 6 as to how this might be done. Note that if your program doesn't run in linear time, it is still possible to get an "A" on the assignment. In either case, your documentation should contain a section on the running time of your algorithm.

Example:

$p \Leftarrow q, \sim r$

$p \Leftarrow s$

$q \Leftarrow \sim s$

$r \Leftarrow \sim t$

t

$s \Leftarrow w$

The following is a sequence of atoms added to C:

$t, \sim r, \sim w, \sim s, q, p.$

Test your program on the preceding example, as well as the next one:

$a \Leftarrow b$
$b \Leftarrow \sim h$
$c \Leftarrow d, e$
$e \Leftarrow$
$d \Leftarrow f, \sim b$
$f \Leftarrow \sim g, \sim h, \sim j$
$f \Leftarrow j$
$g \Leftarrow \sim j$
$h \Leftarrow e$
$i \Leftarrow \sim k$
$k \Leftarrow \sim i$

Notes for the implementation:

1. Please implement your program in Python. (Other languages are acceptable, but you must first get an ok from the TA.)

2. You need to hand in a copy of everything: program listing, documentation, and test results (see below).

3. You should write a command-line program for the assignment.

4. You need to have a file called README.TXT. This file contains the technical information necessary to execute your program. Your README.TXT file should at least contain the following:

   - An example command-line to run your program.
   - Anything else one needs to know in order to test your code.

5. You will need to pass a file name as a parameter to your program (e.g. testcase1.txt). The file you pass to your program will contain the rules.

6. You must run your program for each test case file and submit your results. In addition you may submit your own test cases to provide further evidence that your program is working as expected.

7. Your program should accept the following syntax for the rules file.

   Each rule in the file is specified by a triple:
   $[h\ [p_1\ ...\ p_n]\ [n_1\ ...\ n_m]]$

   - h is an atom which is the head of the rule
   - A list containing p_1, ..., p_n as positive atoms
   - A list containing n_1, ..., n_m as negative atoms
   - All white-space characters (space, tab, newline) are ignored
   - Anything after a '#' character to the end of the line is ignored (i.e. use # for comments)
   - The following is an example set of rules which could be stored in a text file:

```
# here are some example rules:

[a [b1 b2] [c c10]]   # i.e. a <- b1, b2, not c, not c10
[h [a b c] []]        # i.e. h <- a, b, c
[g [] [e f]]          # i.e. g <- not e, not f
[f [] []]             # i.e. f <-
```

8. Your program should output literals as they are derived. Ideally you will also print the reason why a literal was derived.