CMPT 310 - Artificial Intelligence Survey

Assignment 2

Due date: October 22, 2018 — J.P. Delgrande
10 marks — October 3, 2018

Important Note: Students must work individually on this, and other CMPT 310, 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!

Assume that a knowledge base consists of a set of rules of the form $a_1 \wedge \cdots \wedge a_n \Rightarrow a$ where $n \geq 0$, a is an atom, and each a_i is an atom. For a rule $r = (a_1 \wedge \cdots \wedge a_n \Rightarrow a)$, define $head(r) = a$ and $body(r) = \{a_1, \ldots, a_n\}$. If $i = 0$ then the rule is a fact and is just written as a. Here is an example of a set of rules:

p
$p \Rightarrow q$
$q \wedge s \wedge t \Rightarrow r$
$p \wedge u \Rightarrow s$
$q \wedge h \Rightarrow r$
t
$t \Rightarrow h$
$v \Rightarrow u$

You are to implement, using Python 3, a backward chaining algorithm for propositional rules. Your program will be called with a file name as an argument on the command line; this file will contain a set of rules. Your program will then successively read a query, given as an atom, and output whether or not that atom is a logical consequence of the set of rules.

Backward Chaining Procedure: Here is pseudo code that gives an outline of the basic algorithm. The argument to *solve* is a list of goals that have yet to be solved. R is the set of rules. If the user's query is q, then *solve* will initially be called with argument (q).

```
solve(goals):
    if goals = () then return(succeed)
    let a := first(goals)
    let goals := rest(goals)
    for each r ∈ R where head(r) = a
        if solve(append(body(r), goals)) = succeed
        return(succeed)
    return(fail)
```

General Notes:

1. Your program will be given the name of a file containing rules on the command line. It should then process a set of queries as entered by the user, where a query will be a single atom.

2. A rule will be represented as a list, where a rule

   $$a_1 \wedge \cdots \wedge a_n \Rightarrow a$$

   will be represented as the list

   $$(a,\ a_1,\ \ldots,\ an)$$

   So the head of the rule comes first, followed by the body. The example set of rules can be represented as follows:

   (p)

   (q, p)

   (r, q, s, t)

   etc.

3. You must run your program on the test cases that will be provided. In addition, if necessary, submit any other test cases that you feel provide further evidence that your program is working as expected.

4. Your program should output helpful, readable, diagnostics, indicating for example how an atom was derived or when a chain of reasoning fails or succeeds.

5. External documentation should briefly describe what you have done, as well as discuss features and limitations of your program. For example you might want to consider how your program would behave if it had the rule $p \wedge q \Rightarrow p$ or the pair of rules, $q \Rightarrow p$ and $p \Rightarrow q$. (You are not expected to necessarily handle cases such as these, but you are expected to know about them and to document them.)