Simon Barer 301356804 sbarer@sfu.ca CMPT310

Assignment 2

Read-in File

My program starts by outlining its purpose (backward chaining) and requesting the name of the file containing the rules. This file is then stripped of its extraneous characters, and the rules/atoms are populated into the kb list of the program.

A while loop then commences, prompting the user for a query, and returning its validity (and all steps in the solution) until the user types ‘quit’, or the program is terminated manually.

Solve Function

The solve function takes in the query, goals, as its only argument. The first operation in the function is a check that will return True if the length of the first element in goals is 0, which indicates that a Fact has been found. The reason that it is the length of the first element of goals being zero, and not the number of elements in goals being zero is because an empty list could indicate that no new rules have been found, which would mean True is always returned. However, by using python’s pop and insert functions, when a goal is found, goals will have [[ ]], which is a list with one element, whose length is zero. In other words, when a rule is found, an empty element will be inserted into the goals list, which can then be used as a check to return True.

If this first check is not entered (goals contains valid, unproven elements), the new target query will be the first rule popped from the goals list. At this point, a boolean check is set to false. This boolean’s function is to ensure that all atoms in a given clause are true. For instance, if a^b->c, the boolean will only be true if a and b are true, but false if only one, or neither, of a and b are true.

The program then enters a nested for loop that parses each atom within the target rule, and compares it with the rules in kb to see if there are any matches. Upon finding a match, it inserts the corresponding rule into goals, and then solves for that rule. This will happen recursively until a Fact is found, at which point the function will return and the boolean check set to true. If no fact is found, the boolean check will remain false for that atom, and the loop will return False immediately. This is because, with regards to compound rules, there is no reason to check the remaining atoms in that rule (if a^b->c, and a is False, then the truth of b doesn’t matter).

Printing Diagnostics

After a query is entered by the user, the program informs the user what is being evaluated at each step, until it arrives at either a Fact or False, then bubbles up to show how from that baseline finding, the corresponding rule can be proven or disproven. This transpires until all possibilities are parsed, and which point the program will indicate whether the original query can be proven to be True or False given the inputted kb.

Limitations

Because this chaining program only takes in a query argument, it is unable to keep a running tracker of each atom that has been proven. As a result, it is unable to aggregate disparate rules/atoms together to prove something. For instance, if p->q and q->p are given as rules, the program would enter an infinite loop, as it is unable to know that p/q has already been evaluated. A solution to this problem would be to maintain a list or hash table/dictionary that stores information on what atoms have been evaluated, and use that as a reference check to ensure no infinite loops are entered.