CMPT 310 Assignment 3 Report

Joshua Campbell

301266191

In this assignment, I constructed a backwards chaining program that reads in a list of rules and a set of queries supplied by the user and then attempts to solve those queries using the supplied rules.

Backwards Chaining Implementation:

The backwards chaining algorithm is a recursive algorithm that starts with a set of queries supplied by the user and a set of rules that it will attempt to use in order to prove the queries. It tries to accomplish this goal by iterating over the supplied rules in order find a rule such that the query is the result of the rule. If a rule is found where the result of the rule is the query, the body of that rule is then added to the set of queries to solve and the program will attempt to solve the updated set of queries. If the program cannot find a rule for one of the queries, the failed chain of rules is then outputted, the most recently used rule is removed from the current chain of reasoning, and a new rule is attempted provided one exists. This will continue until the set of all (updated) queries has been exhausted (the queries have been solved) or until there are no more rules that can be used to solve the queries (there is no solution for the user entered queries).

Features of the program:

At startup, this program will read in a set of rules from a rule file and it will then prompt the user to enter queries. The user can enter a set of queries that the program will attempt to solve. If the program encounters a query that it cannot solve (possibly the body of a rule used to prove another query), the program will back track and it will output a diagnostic showing the failed proof chain. If the program finds a solution to the set of queries, it will output two solutions. The first solution (Solution Chain with duplicates removed) shows all of the individual rules that were used to solve the original query. The second solution (Unaltered Solution Chain) shows the exact chain of rules that the program used to solve the original query. The main difference between these two is that the second solution shows how the algorithm reuses rules in order to prove the bodies of other rules/queries that were used in generating the solution. If the program fails to find a solution, the first and second solution will both list “no solution” as their output.

Limitations of the program:

One of the main limitations of this program is that it is unable to handle rules that prove each other such as “p => q and q => p” or “p ∧ q => p”. The reason for this is that these types rules can cause infinite recursion where the algorithm endlessly attempts to prove the result of one rule using the other (or in the case of “p ∧ q => p”, it will attempt to use itself to prove p recursively). One potential solution to this problem would be to treat the solution chain as graph of rules and then check if sets of rules are generating cycles. If a cycle is found, the program could abort from that chain of reasoning.

Program Output Explanation:

If the program encounters a failed solution chain or a solution chain, the output (for example b^c => a, j => b, j, f^d => c), will show that the first rule used to attempt to solve the query (in this case the query is “a”) was b^c => a. The next rule used was j => b which was used to try and solve the “b” part of previous rule. This continues until either all queries/rule bodies are solved (a solution chain) or until the output cannot find a rule to satisfy the current query/atom of the body of a rule (a failed solution chain). If “Solution Chain” and “Unaltered Solution Chain” is output with rules, this means the program found a solution for the original set of queries. If “Solution Chain” and “Unaltered Solution Chain” is output with “no solution”, then this means the program failed to find a solution for the set of queries.

Program Examples (running on supplied rules.txt file):

Successful Solution:

Enter a query: a

Would you like to add another query to the set? (yes/no): yes

Enter a query: q

Would you like to add another query to the set? (yes/no): no

Query: a, q,

Rules: p, p => q, d, q^j => g, f => e, f^d => c, d^g => c, e^d => c, j => b, b^c => a, j,

---

Failed Solution Chain: b^c => a, j => b, j, f^d => c

Solution Rules (duplicates removed): b^c => a, j => b, d^g => c, d, q^j => g, j, p => q, p

Unaltered Solution Chain: b^c => a, j => b, j, d^g => c, d, q^j => g, p => q, p, j, p => q, p

Unsuccessful Solution:

Enter a query: e

Would you like to add another query to the set? (yes/no): no

Query: e,

Rules: p, p => q, d, q^j => g, f => e, f^d => c, d^g => c, e^d => c, j => b, b^c => a, j,

---

Failed Solution Chain: f => e

Solution Rules (duplicates removed): no solution

Unaltered Solution Chain: no solution