CSCE 625
Programming Assignment #4

PROGRAM OVERVIEW

The program reso.cpp takes a .kb file as input. The .kb file has to have clauses in CNF form without the or symbol (v) and all the literals separated by space as shown in the example1.kb file in the question. The program then scans the uncommented lines from the .kb file and calls cleanup function to remove redundant spaces, remove duplicate literals and sort each of the clauses so that the literals appear in alphabetical order. After cleanup, the clauses are stored in a vector of strings named “clauses”. This ensures a clause at index i can be accessed in O(1) or constant time. During this time the visited list of clauses is also checked so as to avoid redundant clauses. The visited list of clauses is maintained using a map named “visited” from string to integer. This ensures that we can check if a clause is visited or not in minimal time. There is another vector named “parent” that stores the indices of two parent clauses from whom each of the clauses are generated. The parents for clauses[i] are parent[i].first and parent[i].second and can be accessed in O(1) if index i is known. The ones that are initial clauses or the ones given in the .kb file, have indices -1 and -1 as the two parents. For the intermediate clauses that will be generated, its parents are populated appropriately taking the indices of the clauses that generate them. After reading from the file and storing the clauses, the resolution function is called that checks if two clauses in the list of clauses can be resolved using a function named “resolvable”. If they can indeed be resolved, then a resolution pair is formed by creating an object of a class named “ResPair” that contains the indices of the two clauses that can be resolved and we put this ResPair object into our priority queue. The priority queue is named “candidates”. It has a custom comparison function named “CompareResPair” which sorts the elements in the queue based on the lowest length of the sum of the lengths of the two clauses meaning lowest sum is given highest priority. This is what acts as our heuristic scoring function (generalization of unit-clause preference heuristic). After all the resolvable pairs have been pushed into the queue, we pop the top element from the queue and find the resolvents for that particular resolution pair by calling “findresolvents” function. If we get any resolvent as empty clause, we print success and return true to later call “preprint” function to trace the origin of the empty clause from the resolution of the input clauses and the generated intermediate clauses. On the other hand, if we don’t get an empty clause as one of the generated resolvents, we check if the resolvent is a visited clause; if not, then we add that resolvent to our list of clauses and update the visited list of clauses. We also update the corresponding parent vector at that same index as the clauses vector takes for this new clause being pushed into the clauses list. We continue this until we run out of ResPairs in the priority queue named candidates. If we don’t get an empty clause, we return false and print failure. In both the cases, failure or success, we print the total number of resolutions (iterations of the main loop) and the max queue size.

For any queries please feel free to call or message me.

Abhinandan Aryadipta (Abhi)
UIN: 925001240
Email: aryarockstar@tamu.edu
Ph: 9799857204