John Polimeni: Collaborators Luke Gerstner, Isaac Wong

CSC 242

Writeup Project 2

Ferguson

**Automated Reasoning**

**Classes:**

For this project, we attempted to complete it without the aid of the code provided to us by Professor Ferguson. But after a lack of progress we decided to use the template that he gave us. From the template we used the classes, Biconditional, BinaryCompoundSentence, BinarConnective, Compound Sentence, Conjunction, Disjunction, Implication, Model, Negation, Sentence, Symbol, SymbolTable, UnaryCompound Sentence, and UnuaryConnective. In addition to those classes, we modified his code in classes that he gave us. So, we modified, Clause, KB, ModesPonensKB, and WumpusWorldKB. Therefore, the classes that we created were MyModel, DoorsOfEnlightenmentKB, HornClausesKB, LiarsAndTruthTellersKB, LiarsAndTruthTellersKB2, and MoreLiarsAndTruthTellersKB.

**Part 1: Basic Model Checking:**

The goal of part one of the project was to implement a truth table enumeration method that was described to us in AIMA page 248. In order to accomplish this, we first had to create a myModel class. This class used the interface given to us by Professor Ferguson. In addition to the methods that were implemented by the interface, we also added a clone method that creates a clone of the given model, which is necessary for the algorithm to work. The myModel class uses a hash map to store sentences and Booleans. In the myModel class there are getters and setters, also there is a satisfies method which returns a Boolean value for weather of not the given model has any false Booleans in it, if so it returns false. After creating the myModel class, we implemented the truth table enumeration methods in the KB class. There are two methods that we created in order to make the algorithm operational. The first method is called TTEntails, which takes in a knowledge base and a sentence we call alpha. This method creates two array lists of the symbols of the knowledge base and the sentence alpha. Also in the method is a for loop that inputs symbols from the alpha list into the KBList. After this a recursive call is made to the TTCheckALL method, which takes in a kb, a sentence called alpha, the KBList from the previous method and a model. This method Goes through all of the symbols and determines if they are satisfied by the model and setting a Boolean value in the hash table. The value of this Boolean depends on if it was satisfied or not The method works as such, if the symbols list is empty then the program tests if each sentence in kb satisfies the model, and if so then it returns the result of alpha in the isSatisfied method. If the symbols don’t satisfy the model then it returns true.

If the symbols list isn’t empty then a sentence variable called P is created and that takes the first symbol from the list and an array list called rest is created that stores the rest of the symbols in the list. Additionally, two copies of the models are created, one with P set to true (called M1) and one where P is set to false (Called M2), which are used in the recursive calls. There are two recursive calls, one that uses the copy with M1, and one with M2. After all of the recursive calls are done in TTCheckAll, a Boolean value will be retuned and that is the truth value of the statement. This form of inference is sound and complete because it works for all kb and alpha always terminates. Additionally, the number of models in kb is 2^n, therefore the time complexity is also O(2^n).

**Part 2: Advanced Propositional Inference:**

The second part of this project was to pick another technique for propositional inference, and we chose the resolution based theorem prover. The resolution theorem prover converts the knowledge base and negation of alpha into CNF then resolution is applied and after resolution has finished one of two outcomes will be the result. Either there are no new clauses that can be added, which means KB doesn’t entail alpha, or two clauses resolve and leave us with the empty case, which means KB entails alpha.

The PLReslove method takes in a knowledge base and an alpha, then it converts the kb to clauses using the given CNFConverter class and it converts and negates alpha in the same way. The method begins with a while loop that runs while its true. Then the clauses are put into an iterator because this makes them able to easily be used in a loop. The next loop iterates through all values of the clauses, and within this loop another iterator is created and then used in an additional loop that iterates through all its elements. The purpose of these two loops is to compare all of the clauses. Within these nested for loops we check to see if the two clauses equal each other, and if they are not equal then a new array set is created by calling the method PLResolve. This method returns a set of all possible clauses obtained by resolving the two inputs. After PLResolve is called the new set of resolvents is checked to see if its empty, if it is empty, then true is returned and it can be asserted that kb entails alpha, otherwise the resolvents are added to the clause set. After the while loop is executed and all clauses are compared, then either false will be returned if the resolve method didn’t resolve any new clauses. If new clauses were found then they are added and the loop is executed again. Resolution is sound and complete, and the runtime for a resolution is exponential, therefore it is slower than the part one.

**Sample Problems:**

All the sample problems are inputted in the same way, first they are added using the intern function, then they are converted into sentences and added to the knowledge base. Then in the main method we call either TTEntails, or PLResolution with the parameters being the class we are in as the knowledge base and the symbol that we want to solve as the alpha.

Modus Ponens

P

(IMPLIES P Q)

<Solving by Model Checking>

Name Q

Does {P, P => Q} entail Q: true

<Solving by Resolution>

resolutions resolved: 9

Does {P, P => Q} entail Q: true

Wumpus World

(NOT P1,1)

(IFF B1,1 (OR P1,2 P2,1))

(IFF B2,1 (OR P1,2 (OR P2,2 P3,1)))

(NOT B1,1)

B2,1

<Solving by Model Checking>

Name P1,2

There is a pit in [1,2]: false

<Solving by Resolution>

resolutions resolved: 55530

There is a pit in [1,2]: false

Horn Clause

(IMPLIES M (NOT MO))

(IMPLIES (NOT M) (AND MO MAM))

(IMPLIES (OR (NOT MO) MAM) H)

(IMPLIES H MA)

<Solving by Model Checking>

Name M

We can prove that the unicorn is mythical: false

<Solving by Model Checking>

Name MA

We can prove that the unicorn is magical: true

<Solving by Model Checking>

Name H

We can prove that the unicorn is horned: true

<Solving by Resolution>

resolutions resolved: 248

We can prove that the unicorn is mythical: false

<Solving by Resolution>

resolutions resolved: 356

We can prove that the unicorn is magical: true

<Solving by Resolution>

resolutions resolved: 405

We can prove that the unicorn is horned: true

**Extra Credit:**

Liars and Truth Tellers

(IFF A (AND A C))

(IFF B (NOT C))

(IFF C (OR B (NOT A)))

<Solving by Model Checking>

Name A

Amy is truthful: false

<Solving by Model Checking>

Name B

Bob is truthful: false

<Solving by Model Checking>

Name C

Cal is truthful: true

<Solving by Resolution>

resolutions resolved: 3580

Amy is truthful: false

<Solving by Resolution>

resolutions resolved: 7141

Bob is truthful: false

<Solving by Resolution>

resolutions resolved: 7206

Cal is truthful: true

Part 2

(IFF A (NOT C))

(IFF B (AND A C))

(IFF C B)

<Solving by Model Checking>

Name A

Amy is truthful: true

<Solving by Model Checking>

Name B

Bob is truthful: false

<Solving by Model Checking>

Name C

Cal is truthful: false

<Solving by Resolution>

resolutions resolved: 245

Amy is truthful: true

<Solving by Resolution>

resolutions resolved: 6532

Bob is truthful: false

<Solving by Resolution>

resolutions resolved: 12828

Cal is truthful: false

More Liars and Truth Tellers (resolution did not produce an answer, the runtime was too long)

(IFF A (AND H I))

(IFF B (AND A L))

(IFF C (AND B G))

(IFF D (AND E L))

(IFF E (AND C H))

(IFF F (AND D I))

(IFF G (AND (NOT E) (NOT J)))

(IFF H (AND (NOT F) (NOT K)))

(IFF I (AND (NOT G) (NOT K)))

(IFF J (AND (NOT A) (NOT C)))

(IFF K (AND (NOT D) (NOT F)))

(IFF L (AND (NOT B) (NOT J)))

<Solving by Model Checking>

Name A

Amy is truthful: false

<Solving by Model Checking>

Name B

Bob is truthful: false

<Solving by Model Checking>

Name C

Cal is truthful: false

<Solving by Model Checking>

Name D

Dee is truthful: false

<Solving by Model Checking>

Name E

Eli is truthful: false

<Solving by Model Checking>

Name F

Fay is truthful: false

<Solving by Model Checking>

Name G

Gil is truthful: false

<Solving by Model Checking>

Name H

Hal is truthful: false

<Solving by Model Checking>

Name I

Ida is truthful: false

<Solving by Model Checking>

Name J

Jay is truthful: true

<Solving by Model Checking>

Name K

Kay is truthful: true

<Solving by Model Checking>

Name L

Lee is truthful: false

Doors of Enlightenment (resolution did not produce an answer, the runtime was too long)

<Solving by Model Checking>

Name W

The philosopher should choose door W: false

<Solving by Model Checking>

Name X

The philosopher should choose door X: true

<Solving by Model Checking>

Name Y

The philosopher should choose door Y: false

<Solving by Model Checking>

Name Z

The philosopher should choose door Z: false

**Improvements:**

There are multiple improvements that we could add to make our project better. For example, our part two is not very efficient so by implementing a better algorithm that has a better runtime. Additionally, one feature that I wanted to implement was user input and user generated sentences and knowledgebases. I feel by adding that it would make our program much more practical and useful.