Writeup

Jiayi Qi
netID: jqi8

Part 1. Basic Model Checking

A. Representation of Propositional Language

In this program, I use a set of class objects to represent both connectives and objects.

1. Unary Classes

Class **Obj**:

Represent **variable**. This kind of class object contains one attributes, the Boolean value.

This kind of class object have a **set** function with one argument, **b**. This function will set the Boolean value with **b**.

Class **Negation**:

Represent **negation** (¬). This kind of class object contains two attributes, one is the Boolean value and the other is the Obj class object. For example, assuming that negation object ***neg*** represents ¬A. Its Boolean value can be True or False. And its Obj class object is A.

This kind of class object have an **update** function, which can update the Boolean value when the **update** function is called.

2. Binary Classes

Class **And**:

Represent **AND** (∧). This kind of class object contains three attributes, one for the Boolean value and the other two are the two **Obj** objects connected by AND connectives.

This kind of class object have an **update** function, which can update the Boolean value when the **update** function is called.

Class **Or**:

Represent **OR** (∨). This kind of class object contains three attributes, one for the Boolean value and the other two are the two **Obj** objects connected by OR connectives.

This kind of class object have an **update** function, which can update the Boolean value when the **update** function is called.

Class **Arrow**:

Represent **implication** (⇒). This kind of class object contains three attributes, one for the Boolean value and the other two are the two **Obj** objects connected by implication connectives.

This kind of class object have an **update** function, which can update the Boolean value when the **update** function is called.

Class **Arrow2**:

Represent **equivalence** (⟺). This kind of class object contains three attributes, one for the Boolean value and the other two are the two **Obj** objects connected by equivalence connectives.

This kind of class object have an **update** function, which can update the Boolean value when the **update** function is called.

B. Functions

1. arrow(a, b)

   Compute whether an implication statement is true or not and return the result. This function is called when assigning a Boolean value to Arrow object.

2. arrow2(a, b)

   Compute whether an equivalence statement is true or not and return the result. This function is called when assigning a Boolean value to Arrow2 object.

3. TT_entails(KB, a, variables)

   This function will get all the Obj objects in both **KB** and **a**. And then return the value of **TTcheckall** function. Argument **variables** is a list contains all the **Obj** objects in the original **KB** and **Alpha** in a specific order. This argument serves as a global variable which is used for set values to all the **Obj** when a model is generated. The model is a list contains several Boolean values. The order in model is the order in **variables**. This means **model[index]** is the Boolean value of **variables[index]**.

4. TTcheckall(KB, a, symbols, model, variables)

   This function is to generate the truth table (every possible model). Argument **symbols** is a list contains the unassigned **Obj** variables. So when this list is empty, a model is generated. Then I check whether the **KB** is true or not using current argument **model** using **PL_true** function. If **KB** is true, then return the Boolean value of **a** determined by **PL_true** function. If not, it will return true.

   If **symbols** list is not empty, then I take out the first **Obj** variables, **x**, in **symbols** list. And a new **symbols list** is generated by removing the first **Obj** variables. And then I create two new models, one for x=True and the other for x=False. And call **TTcheckall** function for both models. Then return the conjunction result of these two function result.

5. PL_true(a, model, variables):

   This function is for applying **model** with **a**, and return the Boolean value of **a**. Note: In both KB and alpha, the last element is a conjunction of the whole set. So in **PL_true** function, I can simply check the last element to determine **KB or alpha** is true or not.

C. Problems

1. Modus Ponens

   Simply use this model checking algorithm (truth table). **Result is true, which means Modus Ponens is true.**

2. Wumpus World (simple)

   Use this model checking algorithm to determine whether the knowledge base entails **P1,2**. If the algorithm return true, then **P1,2** is true. **Result is false, which means P1,2 is false.**

3. Horn clause

   Use this model checking algorithm to determine whether the knowledge base entails **mythical**. If the algorithm returns true, then **mythical** is true. This is also the case for both **magical** and **horned**.

   Results:

   The unicorn is not mythical (False).

   The unicorn is magical (True).

   The unicorn is horned (True).

4. Liars and Truth Tellers

   To solve this problem, I need to check the entailment from **KB** to a model. **KB** is the equivalences between the speaker and his words. If **KB** entails a model, then the model is the truthfulness of each.

   Results:

   (a) A=False, B=False, C=True

   (b) A=True, B=False, C=False

D. Screenshot

```
qijiayideMacBook-Pro:project_2 qijiayi$ python3 basic.py
Modus Ponens

{P,P=>Q}|=Q:
True


Wumpus World (Simple)

P12:
False


Horn clause

Unicorn is mythical:
False
Unicorn is magical:
True
Unicorn is horned:
True


Liars and Truth-tellers (a):

A,B,C:
[False, False, True]
Liars and Truth-tellers (b):

A,B,C:
[True, False, False]
```

Part 2. Resolution-based theorem prover

A. Representation of CNF

A CNF clause is represented by a list, each element is a variable. For example, clause $A \vee \neg B \vee C$ is represented as [A,-B,C]. For the uniqueness and representing

the complementarity, every different variables are assigned to a different number. Then the complementarity is represented as the sum equals to zero. In this case, knowledge base is a list, which every element is a CNF clause list.

B. Functions

1. pl_resolution(KB, nota):

   This function is to take out all the CNF clauses. And keep doing a loop. In every loops, it will generate a new clause from every pairs of clauses using **pl_resolve** function. **KB** is the knowledge base, **nota** is the negation of **alpha**.If the new clause is empty, it means a contradiction is generated. So **KB** entails **alpha**. If not, add the new clause in the list of new clause. If there is no new clause is generated, then return false. If not, add the new clause set to the clause set and start another loop.

   Note: Because the function checks the existence of a clause, so the order matters. So before adding a clause to any set, I will sort the clause to make it unique.

2. pl_resolve(list1, list2)

   This function returns the set of all possible clauses obtained by resolving **list1**, **list2**. Arguments **list1** and **list2** are both a single CNF clause.

C. Problems

1. Modus Ponens

   Simply use this algorithm. Use {P, P=>Q} to prove Q. **Result is true, which means Modus Ponens is true.**

2. Wumpus World(Simple)

   Simply use this algorithm. Use the knowledge base to prove P1,2. **Result is false, which means P1,2 is false.**

3. Horn clauses

   Simply use this algorithm. Use the knowledge base to prove mythical. So do the cases for magical, horned.

   Results:

   Mythical: False
   Magical: True
   Horned: True

4. Liars and Truth-tellers

   Simply use this algorithm. Use the knowledge base to determine A. If it returns true, then A is true. And do the same thing with B and C. Then we get the result.

Results:

(a)A=False, B=False, C=True

(b)A=True, B=False, C=False

D. Screenshot

```
qijiayideMacBook-Pro:project_2 qijiayi$ python3 advanced.py
Modus Ponens:
{P,P=>Q}|=Q:
True

Wumpus World(Simple)
P12:
False

Horn Clause
Mythical:
False

Magical:
True

horned:
True
Liars and Truth-tellers (a)
A:
False

B:
False

C:
True
Liars and Truth-tellers (b)
A:
True

B:
False

C:
False
```