Project2:Automated Reasoning

CSC442:Introduction to AI

Ziyi Kou

Collaboration: Ziyi Kou & Ziqiu Wu

1 Part One: Basic Model Checking

Given a set of propositional logic sentences, the basic idea of part one is just tree-searching all the condition that every symbol is true or false which finally formulates a truth table to check.

1.1 Overview

Propositional logic sentence is expressed by knowledge representation language and represents some assertion about the world. It is usually composed by a set of different symbols connected with several connectives or just a single symbol.

Taking the knowledge base that contains one or several sentences as input, the agent need to return an action which is often also a symbol or a sentence.

In part one, we firstly construct a parser to parse the propositional logic representation to the computer language format. Then the algorithm builds a truth table and checks if query is true when knowledge base is true. If so, we could prove knowledge base entails the query sentence. Figure 1 is the overview graph for the whole procedure.

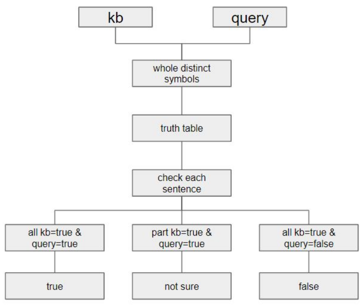


Figure 1: overview of the part1

1.2 Algorithm

There are two main methods used in part1.

The first one is constructing model by generating values for each distinct symbol extracted from both knowledge base and query sentence. There is a dictionary structure in the model to store the value for each symbol. For every step, the model gets the symbol popped from the symbol list and sets its value true and false respectively until there is no more symbol in the symbol list, which means all symbols have been assigned a value. Figure 2 illustrates the procedure when there are two symbols in the whole problem.

The second one is checking if the query is true or false based on the truth

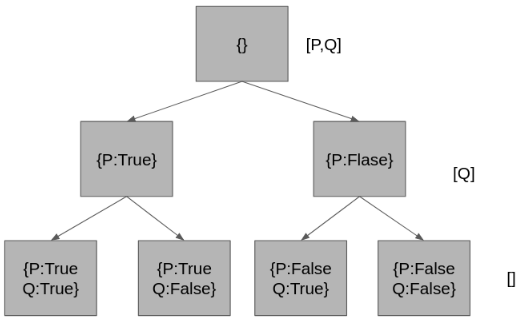


Figure 2: the procedure of generating model

table generated before. Since the input sentences are propositional logic, we need firstly convert them into computer language, splitting each sentence recursively until single symbol that we could assign values. Then by returning recursively we could get values for each sentence. Figure 3 is an example for sentence $B_{1,1} <=> (P_{1,2} \vee P_{2,2} \vee P_{3,1})$.

When all the sentences in knowledge base are true, if the query is always

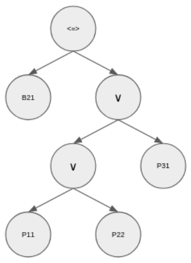


Figure 3: split the sentence and assign the values of symbols

true in each possible model, we say knowledge base entails the query. When all

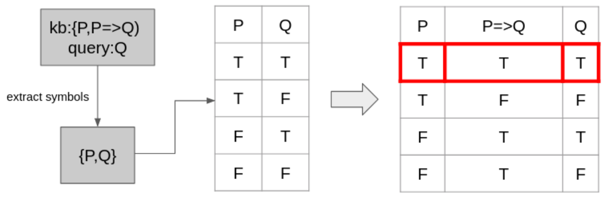


Figure 4: procedure for problem 1

the sentences in knowledge base are true but some query are true some are not in different possible worlds, we are not sure about the query. Finally, when the value of knowledge base is false in some model, we just return true since a false knowledge base could always infer a true result. Figure 4 is the whole procedure for problem 1.

1.3 Functions

There are several classes to express different variables and build the truth table. Table 1 is an overview description for different classes.

Class	Description
Symbol	The class for symbols like 'P' 'Q' or 'Stench' which stands for a proposition that can be true or false.
Operator	The class representing five connectives used in the project.
Sentence	The class representing propositional logic sentences composed of one or more symbols with connectives.
Model	The class containing the basic functions for "truth-table enumeration method".
Basic	The class containing the implementation of "truth-table enumeration method" and testing procedure and result of first 6 test samples.

Table 1: description of used classes

For every class there are several functions to contribute to the whole algorithm. We describe each of them in Table 2,3,4,5,6.

	Symbol
name:String	Define the name of the symbol, like 'P' or 'Q'. It is often defined when the class is constructed.
value:String	Assign the corresponding value(true or false) to the symbol. It often happens when the model checks the sentence. Before that, the value is often none.

Table 2: description of class Symbol

	Operator
name:String	Define the name of the symbol, like 'P' or 'Q'. It is often defined when the class is constructed.
value:String	Assign the corresponding value(true or false) to the symbol. It often happens when the model checks the sentence. Before that, the value is often none.

Table 3: description of class Operator, all the functions are static which is easy for the use

	Sentence
value:String	The value for the sentence which is a single symbol or some connected with connectives.
preProcess():String	Preprocess the value of input sentence, deleting space inside and return processed value.
getSymbols():Set	Split the sentence and return all the distinct symbols from it.

Table 4: description of class Sentence

	Model
map:Map	Save the values(true or false) for different symbols. The number of symbols increase with the depth of recursion.
assign():Model	Put new symbol and its value into the map, returning the model itself.
clone():Model	Deepcopy the original model to a new model that has different address in the memory. Note that the model's map is also a complex variable, we need also copy its each value to a new map.
satisfyKB():Boolean	Check if all the sentence in knowledge base is true corresponding to values in the specific model. If one is false, then the returning is false.
singleSatisfy(): Boolean	With the value of sentence as input, we split the value with designed priority of connectives recursively until there only exists single symbol. Then we assign values for them from the model and return step by step and finally the value of whole sentence. In each recursion process, we could define it as following: • check if the input is just one single symbol, if so, return the boolean value of it corresponding to the values in model. If not, go to next step. • check if there exists useless brackets like "$(A \vee B)$". If so, delete all brackets and go next. • check if the input value contains valid brackets. If so, we firstly only consider connectives outside brackets. After collecting them, we extract the least priority connective and split the sentence by it. Then we recurs the two split value with a corresponding operator connecting two values. • if there is no valid brackets, we collect all the connectives and select the least priority connective to split as same as the step above.

Table 5: description of class Model

	Basic
tt_entails():Boolean	Collect all the symbols in knowledge base and query to a set. Then call tt_check_all function to check if knowledge base entails query.
tt_check_all() :Boolean	Build each model recursively by assigning symbol into it until where is no more remaining symbols in the list. Then We check knowledge base and query. If and only if in all possible world, when knowledge base is true, query is true. We could return true for this method.
main()	Test first 6 problem below to check if queries could be entailed by the knowledge. We firstly transform each symbol in the problems to 'A-Z'. Then put them into program to check their value. If the result is true, we return true. If the result is false, we add a "¬" in front of the query to check again. If the vresult is still false, we cannot make sure about the result, else we return false.

Table 6: description of class Basic

Problem1	Problem2	Problem3	Problem4	Problem5	Problem6
• Q: True	• C: False	• mythical:Not Sure • magical:True • horned:True	(a) • Amy:False • Bob:False • Cal:True (b) • Amy:True • Bob:False • Cal:False	symbol:person' name with same letter • J:True • K:True • Others: False	(a) • X:True • Y:not sure • Z:not sure • W:not sure (b) • X:True • Y:not sure • Z:not sure • W:not sure

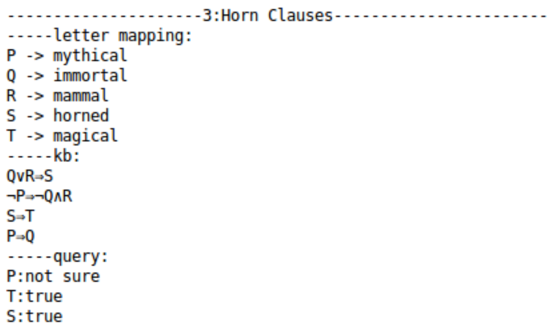
Table 7: results for first 6 problems in basic part. For problem 6, For each x in A, B, C, D, E, F, G, and H we assume x means "x is a knight" and ¬x means "x is a knave", For each x in X Y Z W we assume x means "x is a good door" and ¬x means "x is a bad door"

1.4 Experiments

we test first 6 problems in the project document. For each problem, we transform their language or symbols to 'A-Z' symbol that is easy to program. Then we put all sentence to the functions and get the result.

Take problem 3 as an example shown in Figure 5. We transform the words like 'mythical' to single letter represented the name of symbols. Then we use these symbols to build knowledge base and query. Finally we put them into program and output value for each symbol.

With the same method for all 6 problems, we could get their results shown



```
---------------------3:Horn Clauses-----------------------
-----letter mapping:
P -> mythical
Q -> immortal
R -> mammal
S -> horned
T -> magical
-----kb:
Q∨R⇒S
¬P⇒¬Q∧R
S⇒T
P⇒Q
-----query:
P:not sure
T:true
S:true
```

Figure 5: problem3

in Table 7 above. Actually, the results of part one and part 2 are same, the difference is their methods and discuss which one is more effective.

2 Part Two: Advanced Propositional Inference

In second part, we use resolution rules to solve same problems. We need to convert sentences to CNF Sentence and convert query with a not connective. We use resolution rules to resolve pair of symbols. If we finally could resolve all the symbols and get an empty sentence, we could prove that the original query is true.

2.1 Overview

A sentence expressed as a conjunction of clauses is said to be in conjunctive normal form or CNF. A CNF sentence is an conjunction of "$\vee$" "$\wedge$" and "$\neg$".

For sentences in knowledge base and query, we firstly convert them into CNF sentence, and then use a method to each two of CNF sentence to get a new CNF by resolution rules. Figure 6 shows an overview procedure for part2.

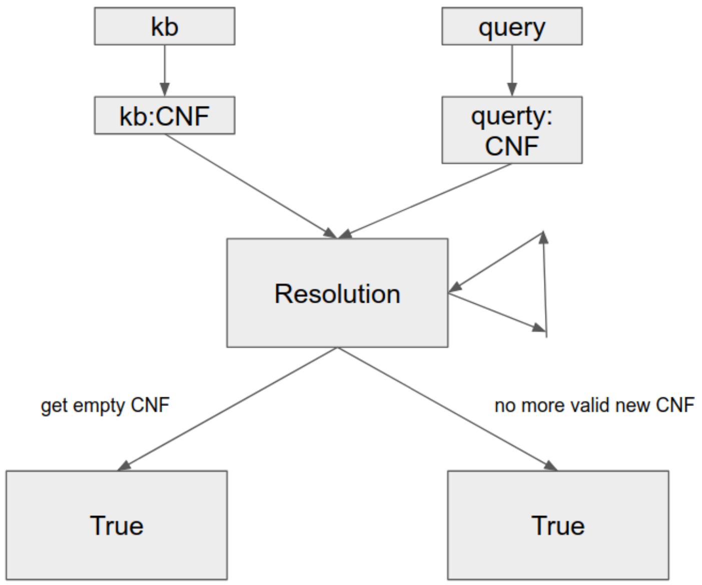


Figure 6: overview of the part2

2.2 Algorithm

In part two, there are also two main methods. The first one is the main resolution function. We convert sentences in knowledge base and query to CNF and put them together. For each two sentence we resolve a new sentence from them and put into a new CNF set it is not empty, otherwise if we get an empty CNF sentence, we could just determine the original query is true. Figure 7 shows this procedure.

The second method is the detail of doing the resolution given two CNF sentences. We firstly extract symbols or symbols with ”not” operator from all sentences and put them into two sets. If there is one symbol in one set and a ”not” operator with this symbol in another set. We record it. If there is only one pair and not all the parent CNFs and result CNF have connectives, we return the result CNF, otherwise we turn a CNF with None value. Figure 8 shows this procedure.

Take problem 1 as an example. From original knowledge base and query, we can convert the sentence to CNF ”P” and ”$\neg P \vee Q$”. Then we use resolution method shown in Figure 9.

2.3 Functions

The classes for part2 are similar with part1. Table 8 is an overview of the whole classes.

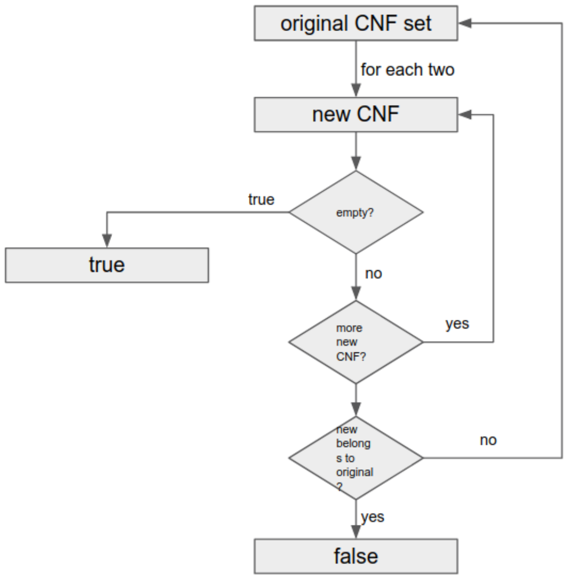


Figure 7: procedure for the whole resolution method

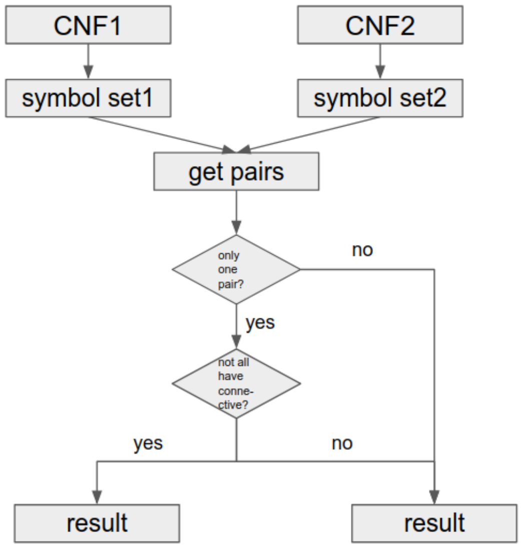
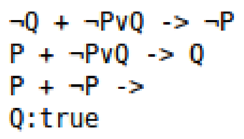


Figure 8: procedure for the single resolution method



```
¬Q + ¬PvQ -> ¬P
P + ¬PvQ -> Q
P + ¬P ->
Q:true
```

Figure 9: procedure for the single resolution method

Class	Description
Clause	The class is similar with the class "Sentence" in part1. It stores an CDF sentence as its values and has some function to do the single resolution method.
Advance	The class containing the implementation of PL_Resolution method and testing procedure and results of first 6 test samples.

Table 8: description of classes in part2

For each class, we have some functions described in Table9 and Table 10

2.4 Experiments

We have done experiments on first 6 problems. And the results are same as part 1. Figure 10 is an example as problem 2.

3 Future words

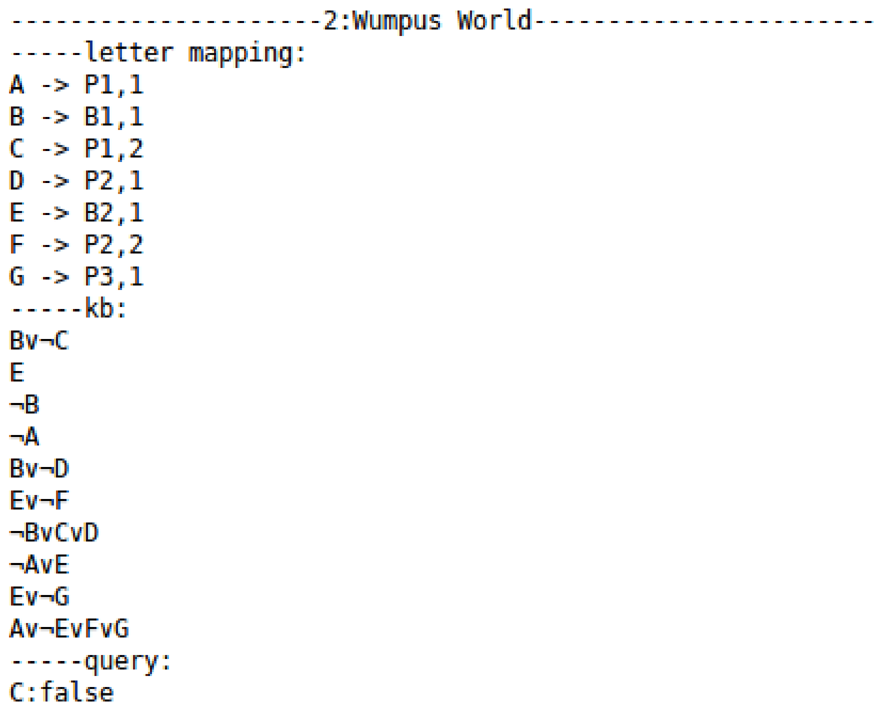
I think we could make the truth table better. When doing the recursion, every possible world are connected with "and". So if one is false, we need not to calculate others.

	Clause
value:String	The value for CNF sentence.
resort():String	Preprocess the value of input sentence, resort the symbols from 'A' to 'Z' since we need to compare the symbols in different pairs.
intersect():Clause	We define two sets contained symbols from two CNF, detecting symbols with same value but with or without "¬" connective. If there is only one pair in two sets and not all the three CNF contains "∨", we return the result CNF, otherwise we just return None which is skipped in the resolution method.

Table 9: description of class Sentence

	Advance
PL_Resolution() :boolean	For both CNFs in knowledge base and query we put them in a same set and calculate the result CNF from each of two until an empty CNF show up or all the new CNFs belongs to the old CNFs.
main()	We test first 6 problems with same method, displaying procedure and results for each problem. With each problem, if the result is true, we reutnr true. If the result is false, we exchange the query with or without a "¬" connective and test again. If the result is stil false, we return "not sure" otherwise we return "true".

Table 10: description of class Sentence



```
--------------------2:Wumpus World----------------------
-----letter mapping:
A -> P1,1
B -> B1,1
C -> P1,2
D -> P2,1
E -> B2,1
F -> P2,2
G -> P3,1
-----kb:
Bv¬C
E
¬B
¬A
Bv¬D
Ev¬F
¬BvCvD
¬AvE
Ev¬G
Av¬EvFvG
-----query:
C:false
```

Figure 10: problem 2