Propositional Logic Project Report

Jing Zhang

Collaborator: Fengxiang Lan

March 6, 2018

1. Introduction to the Algorithms

1.1TT-Entails

The Truth Table is a general algorithm to identify entailment in propositional logic, shown in the Figure 1 (Figure 7.10 from the textbook AIMA). The algorithm recursively enumerates all possible assignments to symbols in finite space.

This algorithm is implemented directly based on the definition of entailment, which guarantees its soundness. And it works for any KB and alpha (the query sentence), which means it is also complete. Also it's not hard to find that this algorithm always terminates since there are only finitely many models to examine.

1.2 PL-Resolution

A resolution algorithm is a series of inference procedures based on resolution and utilizing the principle of proof by contradiction in the meanwhile. To prove KB$|=\alpha$, we prove that $(KB \wedge \neg \alpha)$ can't be satisfied instead. This means proving by contradiction.

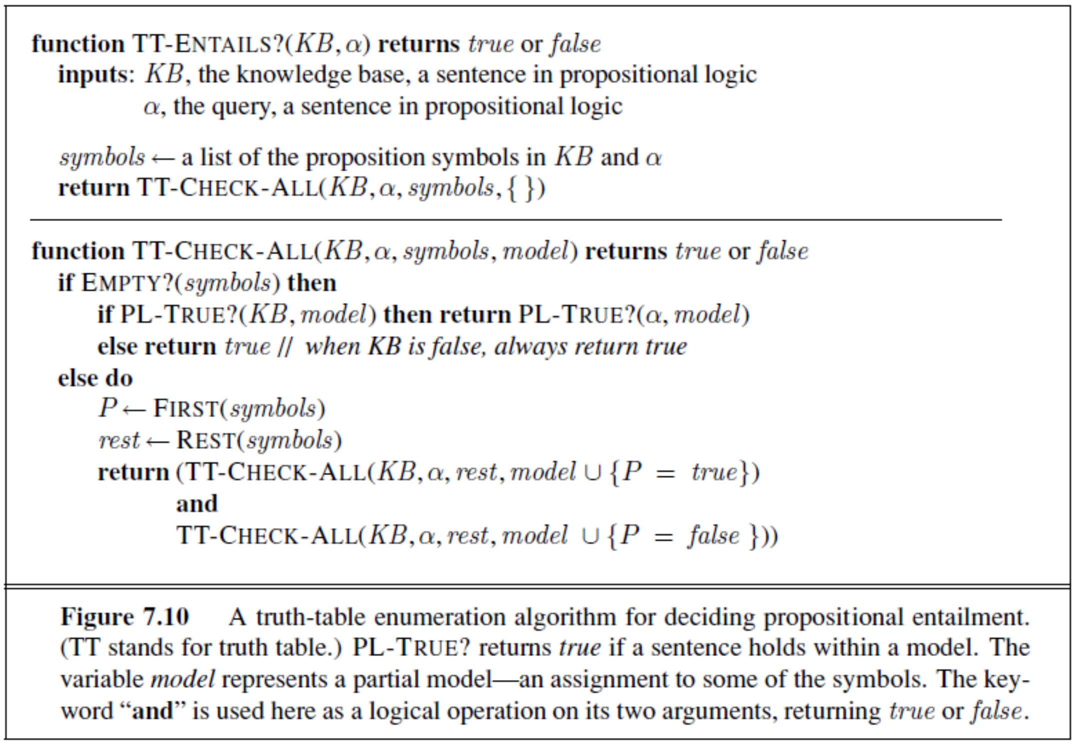
The resolution algorithm we implemented is shown in Figure 2 (Figure 7.12 and Figure 7.13 from the textbook AIMA).

The resolution algorithm we implemented is shown in Figure 2 (Figure 7.12 and Figure 7.13 from the textbook AIMA).

Firstly, we need to converted $(KB \wedge \neg \alpha)$ into CNF. Then we apply the resolution rule to the resulting clause. Each pair that contains complementary literals is resolved and in that way we get a new clause, which is added to the set if it is not already present, or in other words, if it is new. The process continues until one of two things happens:

(1) There are no more clauses that can be added, in which case KB does not entail alpha;

(2) Two clauses resolve to yield the empty clause, in which case KB entails alpha.



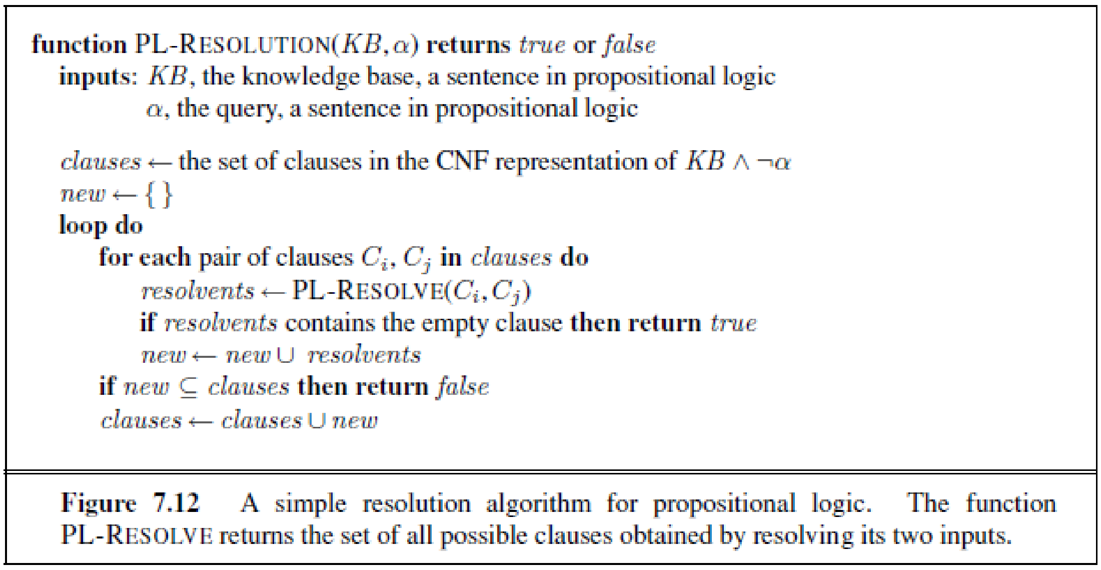
```
function TT-ENTAILS?($KB, \alpha$) returns true or false
  inputs: $KB$, the knowledge base, a sentence in propositional logic
          $\alpha$, the query, a sentence in propositional logic

  symbols ← a list of the proposition symbols in $KB$ and $\alpha$
  return TT-CHECK-ALL($KB, \alpha$, symbols, { })
```

```
function TT-CHECK-ALL($KB, \alpha$, symbols, model) returns true or false
  if EMPTY?(symbols) then
     if PL-TRUE?($KB$, model) then return PL-TRUE?($\alpha$, model)
     else return true // when KB is false, always return true
  else do
     $P$ ← FIRST(symbols)
     rest ← REST(symbols)
     return (TT-CHECK-ALL($KB, \alpha$, rest, model $\cup \{P = true\}$)
            and
            TT-CHECK-ALL($KB, \alpha$, rest, model $\cup \{P = false\}$))
```

Figure 7.10 A truth-table enumeration algorithm for deciding propositional entailment. (TT stands for truth table.) PL-TRUE? returns *true* if a sentence holds within a model. The variable *model* represents a partial model—an assignment to some of the symbols. The keyword "**and**" is used here as a logical operation on its two arguments, returning *true* or *false*.

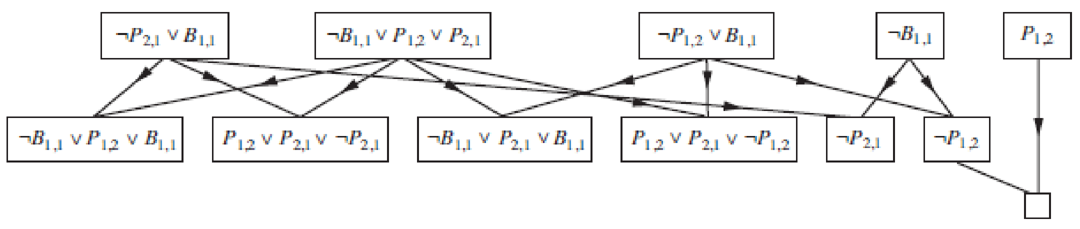
Figure 1 TT-Entail Algorithm



```
function PL-RESOLUTION($KB, \alpha$) returns true or false
  inputs: $KB$, the knowledge base, a sentence in propositional logic
          $\alpha$, the query, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  new ← { }
  loop do
     for each pair of clauses $C_i, C_j$ in clauses do
        resolvents ← PL-RESOLVE($C_i, C_j$)
        if resolvents contains the empty clause then return true
        new ← new ∪ resolvents
     if new ⊆ clauses then return false
     clauses ← clauses ∪ new
```

Figure 7.12 A simple resolution algorithm for propositional logic. The function PL-RESOLVE returns the set of all possible clauses obtained by resolving its two inputs.



$\neg P_{2,1} \vee B_{1,1}$
$\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}$
$\neg P_{1,2} \vee B_{1,1}$
$\neg B_{1,1}$
$P_{1,2}$
$\neg B_{1,1} \vee P_{1,2} \vee B_{1,1}$
$P_{1,2} \vee P_{2,1} \vee \neg P_{2,1}$
$\neg B_{1,1} \vee P_{2,1} \vee B_{1,1}$
$P_{1,2} \vee P_{2,1} \vee \neg P_{1,2}$
$\neg P_{2,1}$
$\neg P_{1,2}$

Figure 7.13 Partial application of PL-RESOLUTION to a simple inference in the wumpus world. $\neg P_{1,2}$ is shown to follow from the first four clauses in the top row.

Figure 2 PL-Resolution

2. Use Our Project to Solve Problems.

2.1 Sample Problem Solved

We have formulated and solved the following sample problems:

(1) Modus Ponens;
(2) Wumpus World (Simple);
(3) Horn Clauses;
(4) Liars and Truth-tellers;
Specially, we write the question(a) and (b) of this sample problem in a same file, by implement two constructor function (different augments to distinguish).
(5) More Liars and Truth-tellers;
(6) The Doors of Enlightenment.

If you want see any results of those sample problems, please go to the path proj2\src\pl\examples.

2.2 To Solve a New Problem

If you want use our project to test a new problem, create a new file and follow the following steps:

1) To build a KB, you need to define the constructor function first. All the new KBs in sample examples, are extension of the class KB. In the constructor function, you need instantiate the symbols, and use the logical notation functions, for example Negation(), to describe the clauses. Then use the function add() to add it to the KB. Take the sample problem Modus Ponens as an example, the constructor should be:

```
public ModusPonensKB() {
            super();
            Symbol p = intern("P");
            Symbol q = intern("Q");
            add(p);
            add(new Implication(p, q));
 }
```

2) In the main function, you need to instantiate the KB, a TT model, a Resolution model and the variable you want to check. Take the sample problem Modus Ponens as an example:

```
ModusPonensKB MPKB = new ModusPonensKB();
Sentence alpha = new Symbol("Q");
TT tt = new TT();
Resolution resolution = new Resolution();
```

3) Call the TTENTAIL(,) function, and the PLresolution(,) function to see the results. Take the sample problem Modus Ponens as an example:

   Boolean resultTT = tt.TTENTAIL(MPKB, alpha);
   Boolean resultRE = resolution.PLresolution(MPKB, alpha);

3. Files Lists

3.1TTModel.java

The Model.java just offers the interface, so we write the TTModel.java to implement the functions in Model.java. Besides the set(), get(), satisfies(KB), satisfies(sentence) and dump(), we also implemented a copy_model() function which is used to copy TTModel objects. That's because, when we are building the truth table, we'll modify the object.

3.2TT.java

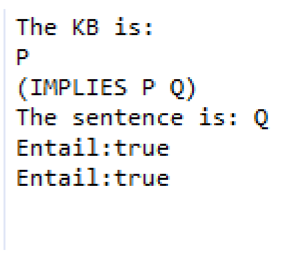
We implement the algorithm described in Figure 1 in this TT.java, with the two functions: TTENTAIL() and TTCheckAll(). To call the TTENTAIL() function, you need two augments, KB and a propostional logic sentence. To solve a query problem, you won't need to call the TTCheckAll() function by yourself.

3.3Resolution.java

We implemented the PL-Resolution algorithm shown in Figure 2 in Resolution.java with utilization of the pl.cnf.* to converter the KB into the CNF mode. To call the PLresolution(), you will need two augments, KB and the query sentence, both in propositional logic.

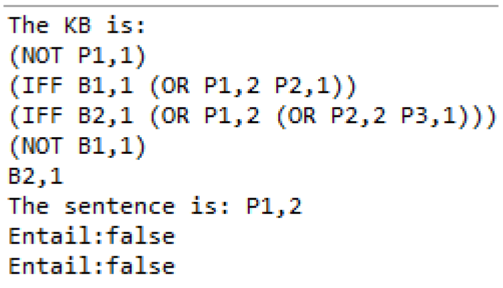
4. Results

4.1Modus Ponens



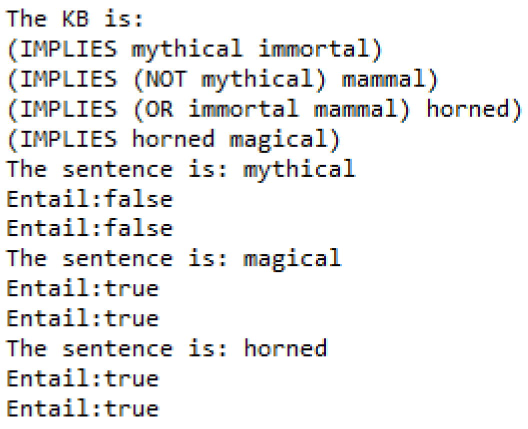
```
The KB is:
P
(IMPLIES P Q)
The sentence is: Q
Entail:true
Entail:true
```

4.2 Wumpus World (Simple)



```
The KB is:
(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
The sentence is: P1,2
Entail:false
Entail:false
```

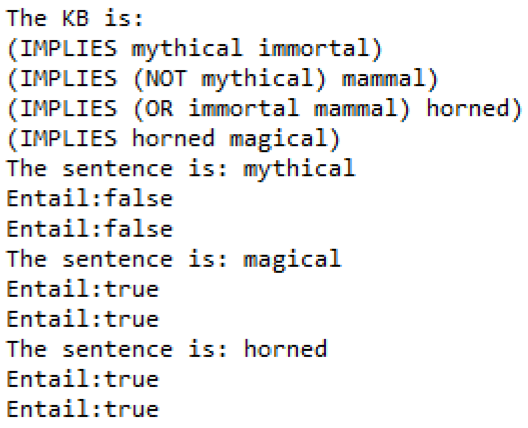
4.3Horn Clauses



```
The KB is:
(IMPLIES mythical immortal)
(IMPLIES (NOT mythical) mammal)
(IMPLIES (OR immortal mammal) horned)
(IMPLIES horned magical)
The sentence is: mythical
Entail:false
Entail:false
The sentence is: magical
Entail:true
Entail:true
The sentence is: horned
Entail:true
Entail:true
```

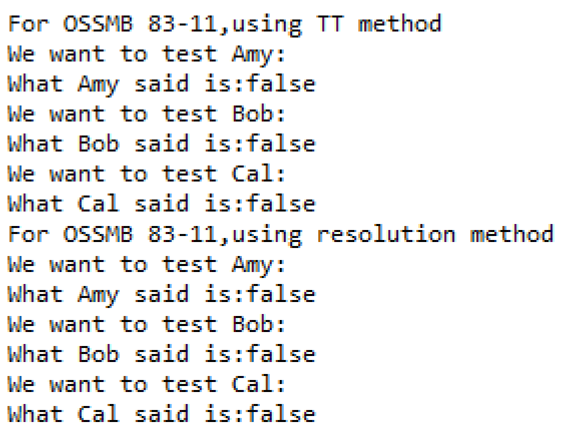
4.4Liars and Truth-tellers

Question (a):



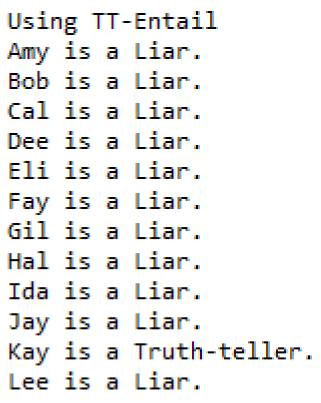
```
The KB is:
(IMPLIES mythical immortal)
(IMPLIES (NOT mythical) mammal)
(IMPLIES (OR immortal mammal) horned)
(IMPLIES horned magical)
The sentence is: mythical
Entail:false
Entail:false
The sentence is: magical
Entail:true
Entail:true
The sentence is: horned
Entail:true
Entail:true
```

Question (b):

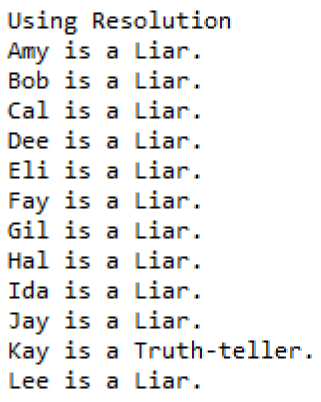


```
For OSSMB 83-11,using TT method
We want to test Amy:
What Amy said is:false
We want to test Bob:
What Bob said is:false
We want to test Cal:
What Cal said is:false
For OSSMB 83-11,using resolution method
We want to test Amy:
What Amy said is:false
We want to test Bob:
What Bob said is:false
We want to test Cal:
What Cal said is:false
```

4.5More Liars and Truth-tellers

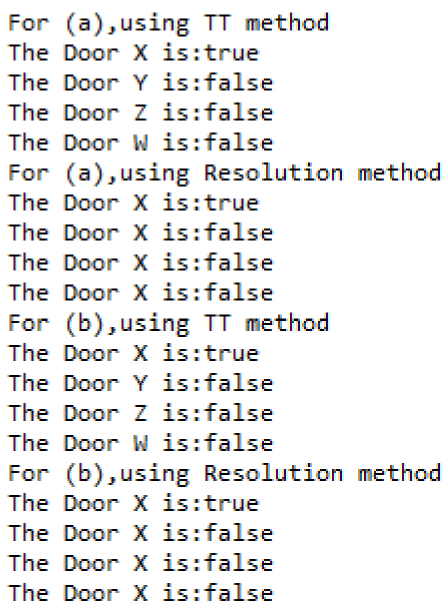


```
Using TT-Entail
Amy is a Liar.
Bob is a Liar.
Cal is a Liar.
Dee is a Liar.
Eli is a Liar.
Fay is a Liar.
Gil is a Liar.
Hal is a Liar.
Ida is a Liar.
Jay is a Liar.
Kay is a Truth-teller.
Lee is a Liar.
```



```
Using Resolution
Amy is a Liar.
Bob is a Liar.
Cal is a Liar.
Dee is a Liar.
Eli is a Liar.
Fay is a Liar.
Gil is a Liar.
Hal is a Liar.
Ida is a Liar.
Jay is a Liar.
Kay is a Truth-teller.
Lee is a Liar.
```

4.6The Doors of Enlightenment



```
For (a),using TT method
The Door X is:true
The Door Y is:false
The Door Z is:false
The Door W is:false
For (a),using Resolution method
The Door X is:true
The Door X is:false
The Door X is:false
The Door X is:false
For (b),using TT method
The Door X is:true
The Door Y is:false
The Door Z is:false
The Door W is:false
For (b),using Resolution method
The Door X is:true
The Door X is:false
The Door X is:false
The Door X is:false
```

5. reference

[1]. Artificial Intelligence: A Modern Method 3rd Edition. Stuart J.Russell, Peter Norvig