Project 2: Automated Reasoning

Xuejian Ye

October 22, 2018

Table of Contents

1. Project Summary:

This project is made up of two inference methods of program which check entailment in propositional logic

a. language: python3

b. Individual work

c. How to run project:
Run the code in terminal and it will show all the result.

d. Project achievement:
1) Basic model part: completed first **five questions**
2) DPLL part: completed first **four questions**

2. Propositional logic representation

In project, I prefer to use tree to represent sentence of propositional logic because I think binary parse is easier to be constructed by binary tree with clear priorities of each logical symbol and it is more convenient to evaluate the value of the sentence.

2.1 Basic model checking:

This part aim to construct and use the truth table to check whether the model of knowledge base is also the model of the sentence (we want to check). I used **tt_entails**, **tt_check_all** and **pl_true** functions to achieve this function. ([1])

- **Tt_entails function:** find all propositional symbols and check whether knowledge base entails a specific sentence
- **Tt_check_all function:** assign value to every propositional symbol and return the result after every propositional symbol has been assigned with a specific value.
- **pl_true function:** check whether the model makes sentences true.

we return True when the knowledge base is false.

2.2 Advanced Propositional Inference

In this part, I decide to use DPLL algorithm for checking satisfiability. DPLL algorithm use contradiction method to prove a entail b which is a huge different part from basic model checking and add (not b) into KB and checking this model, if we can' find any model that makes the sentence true, we can conclude that a entails b.

It starts from a ground CNF formula (a set of ground clauses and I have to convert sentences to CNF and use they directly), then try to build an assignment that verifies the formula (decides whether the set of clauses is unsatisfiable or not, if the set of clauses is satisfiable the algorithm outputs an assignment that satisfies the set of clauses) and the assignment is built using a backtracking mechanism.

In my code I used **dpll, find_pure_symbol** and **find_unit_clause** functions to achieve this function. ([1])

- When we find any clause in this sentence is false, we could return the False directly.
- We will assign a pure symbol a specific value which will make it closer to find the true model.
- If there is a symbol has not been assigned as a specific value in one clause, we could assign a determined value to this symbol which lead to the clause true.

If there is no any pure symbol in clause, this algorithm will assign every possible value to propositional symbol to check the value of model until find the true model or check there is no true model. But anyway, DPLL improves the efficiency of computation for the most part.

2.2.1 Weakness of this part:

Because DPLL is staring from ground CNF formula, and it can only process with sentence in conjunctive normal form (clauses). So I have to convert sentences to CNF manually and after that using they directly.

3. Result

There are two reasons to show the result is false (The sentence entailed from KB or program can't tell this result).So I check both alpha and not alpha, in order to find the special case when the program can't tell whether the knowledge base can entail the sentence we want to check.

3.1 Result of questions by basic model checking

2.1 Q1

```
s_KB=['P',
      '( P => Q )']
s_check=['Q']

model_check(s_KB,s_check)
```

executed in 7ms, finished 00:26:15 2018-10-22

```
Q : True
```

Mean: {P, P => Q} |= Q.

2.2 Q2

```
s_KB=['( ! P11 )',
      '( B11 <=> ( P12 | P21 ) )',
      '( B21 <=> ( P11 | P22 | P31 ) )',
      '( ! B11 )',
      'B21',]
s_check=['P12']

model_check(s_KB,s_check)
```

executed in 7ms, finished 00:26:15 2018-10-22

```
P12 : False
```

Mean: There is not pit at [1,2].

2.3 Q3

```
s_KB=[
    '( MY => IM )',
    '( ( ! MY ) => ( ( ! IM ) & MA ) )',
    '( ( IM | MA ) => HO )',
    '( HO => MA )']

s_check1=['MY']
s_check2=['MA']
s_check3=['HO']

model_check(s_KB,s_check1)
model_check(s_KB,s_check2)
model_check(s_KB,s_check3)
```

executed in 5ms, finished 00:26:15 2018-10-22

```
MY : not sure
MA : True
HO : True
```

Mean: We can't tell whether the unicorn is mythical, but we know that it is magical and horned.

2.4.1 (a)

```
s_KB=['( A <=> ( C & A ) )',
      '( B <=> ( ! C ) )',
      '( C <=> ( B | ( ! A ) ) )']

s_check1=['A']
s_check2=['B']
s_check3=['C']

model_check(s_KB,s_check1)
model_check(s_KB,s_check2)
model_check(s_KB,s_check3)
```

executed in 6ms, finished 00:26:15 2018-10-22

```
A : False
B : False
C : True
```

Mean: Amy and Bob are liars, while Cal is a truth-teller.

2.4.2 (b)

```
s_KB=['( A <=> ( ! C ) )',
      '( B <=> ( A & C ) )',
      '( C <=> B )']

s_check1=['A']
s_check2=['B']
s_check3=['C']

model_check(s_KB,s_check1)
model_check(s_KB,s_check2)
model_check(s_KB,s_check3)
```

executed in 5ms, finished 00:26:15 2018-10-22

```
A : True
B : False
C : False
```

Mean: Amy is a truth teller, while Bob and Cal are liars.

2.5 Q5

```
s_KB=['( A <=> ( H & I ) )',
      '( B <=> ( A & L ) )',
      '( C <=> ( B & G ) )',
      '( D <=> ( E & L ) )',
      '( E <=> ( C & H ) )',
      '( F <=> ( D & I ) )',
      '( G <=> ( ( ! E ) & ( ! J ) ) )',
      '( H <=> ( ( ! F ) & ( ! K ) ) )',
      '( I <=> ( ( ! G ) & ( ! K ) ) )',
      '( J <=> ( ( ! A ) & ( ! C ) ) )',
      '( K <=> ( ( ! D ) & ( ! F ) ) )',
      '( L <=> ( ( ! B ) & ( ! J ) ) )']

s_check=['A','B','C','D','E','F','G','H','I','J','K','L']
for i in range(len(s_check)):
    a=[s_check[i]]
    model_check(s_KB , a)
```

executed in 1.20s, finished 00:26:16 2018-10-22

```
A : False
B : False
C : False
D : False
E : False
F : False
G : False
H : False
I : False
J : True
K : True
L : False
```

Mean: Only Jay and Kay are truth-tellers

3.2 Result of questions by DPLL

2.1 Q1

```
s_KB=['P',
      '( ( ! P ) | Q )']

s_check=['Q']

#print('Question1:')
model_check(s_KB,s_check)
print('\n')
```

executed in 4ms, finished 00:35:33 2018-10-22

```
Q : True
```

2.2 Q2

```
s_KB=['( ! P11 )',
    '( ( ! B11 ) | P12 | P21 )',
    '( ( ! P12 ) | B11 )',
    '( ( ! P21 ) | B11 )',
    '( ( ! B21 ) | P11 | P22 | P31 )',
    '( ( ! P11 ) | B21 )',
    '( ( ! P22 ) | B21 )',
    '( ( ! P31 ) | B21 )',
    '( ! B11 )',
    'B21']

s_check=['P12']

#print('Question2:')
model_check(s_KB,s_check)
print('\n')
```

executed in 4ms, finished 00:35:33 2018-10-22

```
P12 : False
```

2.3 Q3

```
s_KB=['( ( ! MY ) | IM )',
    '( ( ! HO ) | MA )',
    '( MY | ( ! IM ) )',
    '( MY | MA )',
    '( ( ! IM ) | HO )',
    '( ( ! MA ) | HO )']

s_check1=['MY']
s_check2=['MA']
s_check3=['HO']

#print('Question3:')
model_check(s_KB,s_check1)
model_check(s_KB,s_check2)
model_check(s_KB,s_check3)
print('\n')
```

executed in 7ms, finished 00:35:33 2018-10-22

```
MY : not sure
MA : True
HO : True
```

2.4.1 (a)

```
s_KB=['( ( ! A ) | C )',
    '( ( ! B ) | ( ! C ) )',
    '( C | B )',
    '( ( ( ! C ) | ( ! A ) ) | B )',
    '( ( ! B ) | C )',
    '( A | C )']

s_check1=['A']
s_check2=['B']
s_check3=['C']

#print('Question4(a):')
model_check(s_KB,s_check1)
model_check(s_KB,s_check2)
model_check(s_KB,s_check3)
print('\n')
```

executed in 10ms, finished 00:35:33 2018-10-22

```
A : False
B : False
C : True
```

2.4.2 (b)

```
s_KB=['( ( ! A ) | ( ! C ) )',
    '( C | A )',
    '( ( ! B ) | A )',
    '( ( ! B ) | C )',
    '( ( ( ! A ) | ( ! C ) ) | B )',
    '( ( ! C ) | B )',
    '( ( ! B ) | C )']

s_check1=['A']
s_check2=['B']
s_check3=['C']

#print('Question4(b):')
model_check(s_KB,s_check1)
model_check(s_KB,s_check2)
model_check(s_KB,s_check3)
print('\n')
```

executed in 10ms, finished 00:35:33 2018-10-22

```
A : True
B : False
C : False
```

4. Reference:

[1]. http://aima.cs.berkeley.edu/python/logic.html