CSC 242 Project 2 Writeup

5 examples were tested with algorithms ttEntails and WalkSat: Modus Ponens, Wumpus World, Horn Clause (Unicorn), Liars and Truth Tellers and More Liars and Truth Tellers.

Run all commands in Project\_2 folder

To run Modus Ponens example:

1. javac ModusPonensKB.java
2. java ModusPonensKB

To run Wumpus World example:

1. javac WumpusWorldKB.java
2. java WumpusWorldKB

To run Unicorn example:

1. javac UnicornKB.java
2. java UnicornKB

To run Liars and Truth Tellers example:

For Part a

1. javac LiarsAndTruthKBa.java
2. java LiarsAndTruthKBa

For Part b

1. javac LiarsAndTruthKBb.java
2. java LiarsAndTruthKBb

To run More Liars and Truth Tellers example:

1. javac MoreLiarsAndTruthKB.java
2. java MoreLiarsAndTruthKB

The output is the result of implementation of both algorithms ttEntails and WalkSat, followed by the Knowledge Base.

# Part I: Basic Model Checking

The pseudocode in the book (Fig. 7.10) was used as a reference for the implementation.

Basic model checking is implemented in Prover\_.java class that implements the Prover interface. There are three functions in the class: **ttEntails, ttCheckAll, plTrue** (for Sentence)**, plTrue** (for KB) and **entails**. ttEntails calls ttCheckAll, which searches through the truth table for models that are true for the KB and returns true if all corresponding models are also true for Sentence. ttCheckAll contains most of the implementation of model checking and is responsible for actual computation of entailment. It calls plTrue to check if a KB holds within a model.

Note that plTrue has two implementations, for Sentence and KB, that return true if either the Sentence or KB holds within a model.

The given ArraySet data structure was used to store all the symbols in the KB.

# Part II: Advanced Propositional Inference

The pseudocode in the book (Fig 7.18) was used to implement Advanced Propositional Inference.

The implementation of advanced propositional inference algorithm can be found in Solver\_.java class, which implements the interface Solver.java. Solver\_.java contains two static functions: **walkSat** and **getFalseClause**. walkSat returns a satisfiable model for KB, and if none exists returns null. getFalseClause is called in walkSat so that we can return a random clause from the knowledge base that is false under the model. This is a requirement for the walkSat algorithm.

Because of the probabilistic nature of walkSat, we use Math.random() to generate a random real number between 0 and 1 and then compare it with the value of p (Solver\_.java line 23), which is 0.5 usually for this algorithm. This means that each part of the conditional will be executed about half of the time in the 10,000 iterations, which is the value that we pass in as a parameter and is stored in max\_flips.

Because walkSat returns *any* satisfiable model we must add the negation of the conclusion that we’re testing to the KB and then use it. This is because if we’re checking if p entails q, we should check if a satisfiable model exists for p and not q. If so then every model of p is not a model of q. So if walkSat returns null that means that the conclusion is entailed and if it returns a model then the conclusion is not entailed.

# Other work

Sentence\_.java implements the interface Sentence.java and contains the function **isSatisfiedBy** which returns true if the Sentence is satisfied by the model (passed in as a parameter) and false otherwise. The neat thing is that we have already been given code for Model.satisfies in Model.java, so we just use that in isSatisfiedBy.

Model\_.java implements Model.java and contains the following functions: **set, get, satisfies** (for KB), **satisfies** (for Sentence), **dump** and **flipSymbol**. A HashMap was also added as an instance variable. We use it to map a Boolean value to each symbol. set is used to set a Boolean value to a Symbol. get is used to get the Boolean value of the Symbol that is passed in as a parameter. Satisfies iterates over all the clauses in a KB and if it finds a clause that is not satisfied by the model, it returns false. If it doesn’t find any clause that is not satisfied by the model, it returns true.

dump prints out all the mappings from Symbols to Booleans.

flipSymbol is used in the walkSat algorithm. It takes in a clause and an index as parameters and flips the truth mapping of the Symbol at that index.

subList was implemented in ArraySet.java to get the rest of the elements in the list because ttEntails requires us to separate the first element from the rest of the elements of ArraySet<Symbol> symbol although technically sets are unordered.

# List of All Functions Implemented

## In Prover\_.java

ttEntails

ttCheckAll

plTrue (for Sentence)

plTrue (for KB) and entails

## In Solver\_.java

walkSat

getFalseClause

## In Model\_.java

isSatisfiedBy

set

get

satisfies (for KB)

satisfies (for Sentence)

dump

flipSymbol

## In ArraySet.java

rest

# Output

## Modus Ponens

Results from ttEntails

Whether P, P implies Q entails Q: true

Results from walkSat

Whether P, P implies Q entails Q: true

Knowledge base:

P

(IMPLIES P Q)

(NOT Q)

## Wumpus World

Checking with ttEntails:

Whether p12 is entailed: false

Checking with walkSat:

Whether p12 is entailed: false

Knowledge Base:

(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

## Horned Clause (Unicorn)

Proof using ttEntails:

Whether unicorn is mythical: false

Whether unicorn is magical: true

Whether unicorn is horned: true

Proof using WalkSat:

Whether it is mythical: false

Whether it is magical: true

Whether it is horned: true

Knowledge Base:

(IMPLIES Mythical (NOT Mortal))

(IMPLIES (NOT Mythical) (AND Mortal Mammal))

(IMPLIES (OR (NOT Mortal) Mammal) Horned)

(IMPLIES Horned Magical)

## Liars and Truth Tellers Part a

Results from ttEntails

Whether Amy is truthful: false

Whether Bob is truthful: false

Whether Cal is truthful: false

Results from walkSat

Whether Amy is truthful: false

Whether Bob is truthful: false

Whether Cal is truthful: false

Knowledge base:

(IMPLIES Amy (AND Cal Amy))

(IMPLIES Bob (NOT Cal))

(IMPLIES Cal (AND Bob (NOT Amy)))

## Liars and Truth Tellers Part b

Results from ttEntails

Whether Amy is truthful: false

Whether Bob is truthful: false

Whether Cal is truthful: false

Results from walkSat

Whether Amy is truthful: false

Whether Bob is truthful: false

Whether Cal is truthful: false

Knowledge base:

(IMPLIES Amy (NOT Cal))

(IMPLIES Bob (AND Amy Cal))

(IMPLIES Cal Bob)

## More Liars and Truth Tellers

Results from ttEntails

Whether Hal is truthful: false

Whether Kay is truthful: false

Whether Eli is truthful: false

Whether Jay is truthful: false

Whether Bob is truthful: false

Whether Dee is truthful: false

Whether Gli is truthful: false

Whether Ida is truthful: false

Whether Amy is truthful: false

Whether Lee is truthful: false

Whether Cal is truthful: false

Whether Fay is truthful: false

Results from walkSat

Whether Hal is truthful: false

Whether Kay is truthful: false

Whether Eli is truthful: false

Whether Jay is truthful: false

Whether Bob is truthful: false

Whether Dee is truthful: false

Whether Gli is truthful: false

Whether Ida is truthful: false

Whether Amy is truthful: false

Whether Lee is truthful: false

Whether Cal is truthful: false

Whether Fay is truthful: false

Knowledge base:

(IMPLIES Amy (AND Hal Ida))

(IMPLIES Bob (AND Amy Lee))

(IMPLIES Cal (AND Bob Gli))

(IMPLIES Dee (AND Eli Lee))

(IMPLIES Eli (AND Cal Hal))

(IMPLIES Fay (AND Dee Ida))

(IMPLIES Gli (AND (NOT Eli) (NOT Jay)))

(IMPLIES Hal (AND (NOT Fay) (NOT Kay)))

(IMPLIES Ida (AND (NOT Gli) (NOT Kay)))

(IMPLIES Jay (AND (NOT Amy) (NOT Cal)))

(IMPLIES Kay (AND (NOT Dee) (NOT Fay)))

(IMPLIES Lee (AND (NOT Bob) (NOT Jay)))