Automated Reasoning

Jiupeng Zhang, October 2018

Part I: Basic Model Checking

- **Program Design**

Two versions of model checking were implemented in our program: one is to build a concrete truth-table for querying, the other is to build complete ***models*** recursively for satisfaction checking, the former one is intuitive and debugging friendly, and the latter solution is concise and easy understanding. These algorithms were encapsulated as strategy pattern components, makes them handy to call and test:

PLAlgorithms.Entailment.*ModelChecking*.entails(kb, α);
PLAlgorithms.Entailment.*RecursiveModelChecking*.entails(kb, α);
PLAlgorithms.Entailment.*Resolution*.entails(kb, α);

Another interesting part of our design is the ***Sentence*** hierarchy, to represent a nested structure, we designed two classes to store both atomic and complex sentences. However, differences (e.g. combination order and element count) between connective 'and', 'or', 'implication' and 'bidirectional implication' should be noticed. Thus, we set connective $\wedge$ and $\vee$ as multivariate operators, $\Rightarrow$ and $\Leftrightarrow$ as binary operators, and $\neg$ as a unary operator. As *Figure 1* shows below, a ***ComplexSentence*** aggregates by a connective object and an array of ***Atomic / ComplexSentence***, which are subclasses of ***Sentence***. The pre-defined binding relationships between subclauses help us handle sentences much easier.

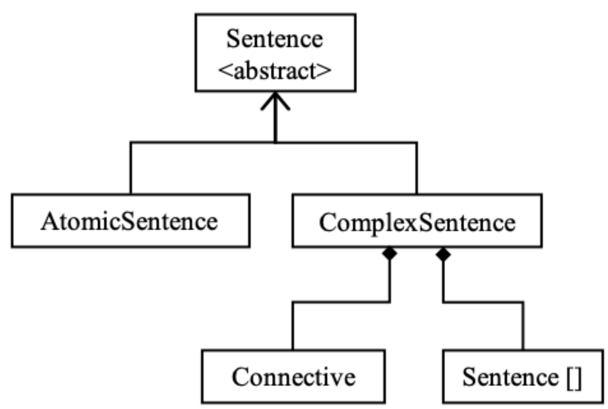

Figure 1: the design of sentence hierarchy

Besides, the ***ComplexSentence*** implements under the ***HashSet*** data structure which has a natural and elegant property of deduplication, and this helps eliminate redundant clauses automatically. For an example, we will get a simplified sentence '$A \wedge B$' if '$A \wedge A \wedge B \wedge B \wedge B$' is given, and if we do negate on a sentence, we will switch the symbol instead of directly add a $\neg$ to it (e.g. it returns 'A' instead of '$\neg\neg A$' if we negate a '$\neg A$'). In fact, these elimination strategies save a considerable computing time.

- **Concepts and Algorithms**

Figure 2 is the flow chart for ***ModelChecking***, as we can see, this diagram shows the recursive implementation of ModelChecking.

In this algorithm, we check values in our ***Knowledge Base** (kb)* until we created models / all symbols appeared in a specific problem, after that, we check if the ***alpha*** sentences satisfy the model. If a check is not passed, we directly return result false, if no conflict were found after iteration, it returns true.

Another version of model checking is to draw a concrete truth-table by extracting symbols from a knowledge base and alpha sentences, and then build models for these symbols by loops. In the non-recursive implementation, we can easily record all unsatisfied lines even though the result is defined. It is clear to locate all unsatisfied symbol-value assignments to helps us debugging.

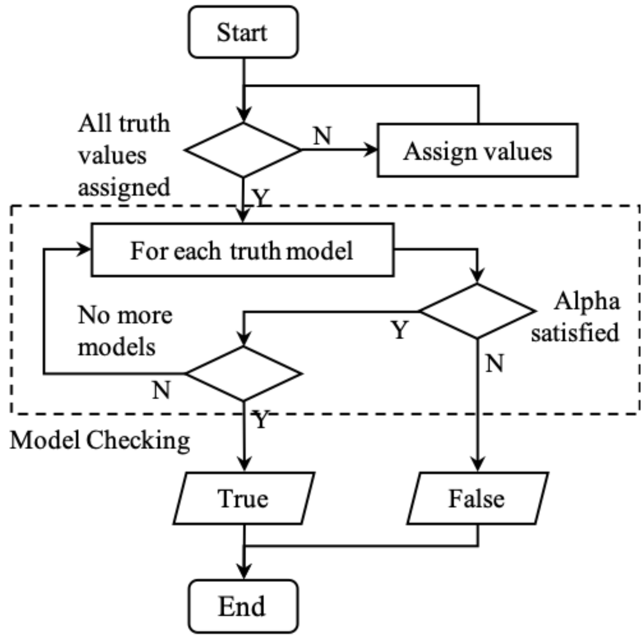

Figure 2: model checking

- **Implementation**

  To run the 'entails' method, we need to build a propositional knowledge base (a.k.a. ***PLKnowledgeBase / kb***) first, and then insert sentences via connective methods (***AND***, ***OR***, ***NOT***, ***IMPLIES***, ***BI_IMPLIES***) into ***kb***.

KB: {at11, ¬s11, s11 ⇔ (w21 ∨ w12)}

PLKnowledgeBase *knowledgeBase* = **new PLKnowledgeBase()**;
knowledgeBase.insert(*at11*);
knowledgeBase.insert(***NOT***(*s11*));
knowledgeBase.insert(***BI_IMPLIES***(*s11*, **OR**(*w21*, *w12*)));

Then, we run the method by given alpha sentences (if more than one sentences are given, we will conjunctive them by ∧), after execution, a Boolean result will be returned, indicates if ***kb*** entails ***alpha***.

We have tested sample 1 to 6 and the corresponded knowledge base, alpha, and results from the program are shown below:

*Note: To avoid taking up too many pages, I listed valid row (for True results) or invalid row (for False results) only, instead of printing the entire truth tables. Specifically, the **Satisfied Models** indicates the satisfaction lines (where kb[current model] ⇒ α) in a truth table, and the **Unsatisfied Models** records the invalid lines in the truth table that will makes the function returns False.*

Sample #1 - Modus Ponens:

(Continued on the next page…)

KB: $\{P,\ P \Rightarrow Q\}$ **Alpha**: Q

ModelChecking.entails(**KnowledgeBases**.***modusPonensKnowledgeBase***(), Q);

Truth Table:
[P, Q, P, (P ==> Q), Q]

F, F, F, T, F;
F, T, F, T, T;
T, F, T, F, F;
T, T, T, T, T; ← satisfied model

Satisfied Models:
1. $P \wedge Q \wedge (P \Rightarrow Q) \wedge Q$

Result: True

Sample #2 - Wumpus World (Simple):

KB: $\{\neg P11,\ B11 \Leftrightarrow (P12 \vee P21),\ B21 \Leftrightarrow (P11 \vee P22 \vee P31),\ \neg B11,\ B21\}$ **Alpha**: P12

ModelChecking.entails(**KnowledgeBases**.***wumpusWorldKnowledgeBase***(), P12);

<u>Un</u>satisfied Models:
1. $\neg P11 \wedge \neg B11 \wedge \neg P12 \wedge \neg P21 \wedge B21 \wedge \neg P22 \wedge P31 \wedge (B11 \Leftrightarrow (P12 \vee P21)) \wedge (B21 \Leftrightarrow (P11 \vee P22 \vee P31)) \wedge \neg B11 \wedge B21 \wedge \neg P12$
2. $\neg P11 \wedge \neg B11 \wedge \neg P12 \wedge \neg P21 \wedge B21 \wedge P22 \wedge \neg P31 \wedge (B11 \Leftrightarrow (P12 \vee P21)) \wedge (B21 \Leftrightarrow (P11 \vee P22 \vee P31)) \wedge \neg B11 \wedge B21 \wedge \neg P12$
3. $\neg P11 \wedge \neg B11 \wedge \neg P12 \wedge \neg P21 \wedge B21 \wedge P22 \wedge P31 \wedge (B11 \Leftrightarrow (P12 \vee P21)) \wedge (B21 \Leftrightarrow (P11 \vee P22 \vee P31)) \wedge \neg B11 \wedge B21 \wedge \neg P12$

Result: False

Sample #3a - Horn Clauses*:

This problem is slightly different from previous ones, it asks whether a proposition is provable. To solve the question, we need to check truth value for both *Alpha* and ¬*Alpha*, and below is the reason:

In the model checking algorithm, a *False* value will be returned if unsatisfied models were found (because this indicates that *KB* doesn't entail *Alpha*). However, a boundary situation is that, when we feed a sentence with unmentioned symbols in *KB*, constraints are lacked, for reason that no models will satisfy both *Alpha* and ¬*Alpha* in this incomplete condition (*e.g.* $kb \in \{a\}$, $\alpha \in \{b\}$), the algorithm will result *False*, too.

(Continued on the next page…)

To sum up, to test if a sentence is provable, we must test for sentences and their opposites with the same knowledge base to show that a KB is complete. Only in this way can we assume a sentence is provable and the return value is valid.

KB: {mythical ⇒ immortal, ¬mythical ⇒ (¬immortal ∧ mammal), (immortal ∨ mammal) ⇒ horned, horned ⇒ magical} **Alpha**: {mythical, ¬mythical}

ModelChecking.entails(**KnowledgeBases**.***hornClausesKnowledgeBase***(), mythical);

Unsatisfied Models:

1. **[check if mythical]** ¬mythical ∧ ¬immortal ∧ mammal ∧ horned ∧ magical ∧ (mythical ⇒ immortal) ∧ (¬mythical ⇒ (¬immortal ∧ mammal)) ∧ ((immortal ∨ mammal) ⇒ horned) ∧ (horned ⇒ magical) ∧ ¬mythical
2. **[check if ¬mythical]** mythical ∧ immortal ∧ ¬mammal ∧ horned ∧ magical ∧ (mythical ==> immortal) ∧ (¬mythical ==> (¬immortal ∧ mammal)) ∧ ((immortal ∨ mammal) ==> horned) ∧ (horned ==> magical) ∧ mythical
3. **[check if ¬mythical]** mythical ∧ immortal ∧ mammal ∧ horned ∧ magical ∧ (mythical ==> immortal) ∧ (¬mythical ==> (¬immortal ∧ mammal)) ∧ ((immortal ∨ mammal) ==> horned) ∧ (horned ==> magical) ∧ mythical

Result: False, False

The scenario shows that 'mythical' is not entailed in the current KB. In other words, we cannot prove the unicorn is mythical.

To simplify trace text, ***provable tests*** will be omitted but marked with * on samples titles if required.

Sample #3b - Horn Clauses*:

KB: {mythical ⇒ immortal, ¬mythical ⇒ (¬immortal ∧ mammal), (immortal ∨ mammal) ⇒ horned, horned ⇒ magical} **Alpha**: magical (provable test omitted)

ModelChecking.entails(**KnowledgeBases**.***hornClausesKnowledgeBase***(), magical);

Satisfied Models:

1. ¬mythical ∧ ¬immortal ∧ mammal ∧ horned ∧ magical ∧ (mythical ⇒ immortal) ∧ (¬mythical ⇒ (¬immortal ∧ mammal)) ∧ ((immortal ∨ mammal) ⇒ horned) ∧ (horned ⇒ magical) ∧ magical
2. mythical ∧ immortal ∧ ¬mammal ∧ horned ∧ magical ∧ (mythical ⇒ immortal) ∧ (¬mythical ⇒ (¬immortal ∧ mammal)) ∧ ((immortal ∨ mammal) ⇒ horned) ∧ (horned ⇒ magical) ∧ magical
3. mythical ∧ immortal ∧ mammal ∧ horned ∧ magical ∧ (mythical ⇒ immortal) ∧ (¬mythical ⇒ (¬immortal ∧ mammal)) ∧ ((immortal ∨ mammal) ⇒ horned) ∧ (horned ⇒ magical) ∧ magical

Result: True

Sample #3c - Horn Clauses*:

KB: {mythical ⇒ immortal, ¬mythical ⇒ (¬immortal ∧ mammal), (immortal ∨ mammal) ⇒ horned, horned ⇒ magical} **Alpha**: horned (provable test omitted)

ModelChecking.entails(**KnowledgeBases**.***hornClausesKnowledgeBase***(), horned);

Satisfied Models:

1. ¬mythical ∧ ¬immortal ∧ mammal ∧ horned ∧ magical ∧ (mythical ⇒ immortal) ∧ (¬mythical ⇒ (¬immortal ∧ mammal)) ∧ ((immortal ∨ mammal) ⇒ horned) ∧ (horned ⇒ magical) ∧ horned
2. mythical ∧ immortal ∧ ¬mammal ∧ horned ∧ magical ∧ (mythical ⇒ immortal) ∧ (¬mythical ⇒ (¬immortal ∧ mammal)) ∧ ((immortal ∨ mammal) ⇒ horned) ∧ (horned ⇒ magical) ∧ horned
3. mythical ∧ immortal ∧ mammal ∧ horned ∧ magical ∧ (mythical ⇒ immortal) ∧ (¬mythical ⇒ (¬immortal ∧ mammal)) ∧ ((immortal ∨ mammal) ⇒ horned) ∧ (horned ⇒ magical) ∧ horned

Result: True

Conclusion for entire sample #3:

- We <u>cannot</u> prove that the unicorn is mythical.
- We can prove that the unicorn is magical.
- We can prove that the unicorn is horned.

Sample #4a - Liars and Truth-tellers:

KB: {Amy ⇔ (Amy ∧ Cal), Bob ⇔ ¬Cal, Cal ⇔ (Bob ∨ ¬Amy)} **Alpha**: {Amy, Bob, Cal}

ModelChecking.entails(**KnowledgeBases**.***liarsAndTruthTellers1KnowledgeBase***(), Amy);
ModelChecking.entails(**KnowledgeBases**.***liarsAndTruthTellers1KnowledgeBase***(), Bob);
ModelChecking.entails(**KnowledgeBases**.***liarsAndTruthTellers1KnowledgeBase***(), Cal);

Un/satisfied Models:

1. **[check Amy]** ¬Amy ∧ Cal ∧ ¬Bob ∧ (Amy <=> (Amy ∧ Cal)) ∧ (Bob <=> ¬Cal) ∧ (Cal <=> (Bob ∨ ¬Amy)) ∧ ¬Amy
2. **[check Bob]** ¬Amy ∧ Cal ∧ ¬Bob ∧ (Amy <=> (Amy ∧ Cal)) ∧ (Bob <=> ¬Cal) ∧ (Cal <=> (Bob ∨ ¬Amy)) ∧ ¬Bob
3. **[check Cal]** ¬Amy ∧ Cal ∧ ¬Bob ∧ (Amy <=> (Amy ∧ Cal)) ∧ (Bob <=> ¬Cal) ∧ (Cal <=> (Bob ∨ ¬Amy)) ∧ Cal

Result: False, False, True

Conclusion: Amy is a liar, Bob is a liar, Cal is that truth-teller.

Sample #4b - Liars and Truth-tellers:

KB: {Amy ⇔ ¬Cal, Bob ⇔ (Amy ∧ Cal), Cal ⇔ Bob} **Alpha**: {Amy, Bob, Cal}

ModelChecking.entails(**KnowledgeBases**.***liarsAndTruthTellers2KnowledgeBase***(), Amy);
ModelChecking.entails(**KnowledgeBases**.***liarsAndTruthTellers2KnowledgeBase***(), Bob);
ModelChecking.entails(**KnowledgeBases**.***liarsAndTruthTellers2KnowledgeBase***(), Cal);

Satisfied Models:

1. **[check Amy]** Amy ∧ ¬Cal ∧ ¬Bob ∧ (Amy <=> ¬Cal) ∧ (Bob <=> (Amy ∧ Cal)) ∧ (Cal <=> Bob) ∧ Amy
2. **[check Bob]** Amy ∧ ¬Cal ∧ ¬Bob ∧ (Amy <=> ¬Cal) ∧ (Bob <=> (Amy ∧ Cal)) ∧ (Cal <=> Bob) ∧ ¬Bob
3. **[check Cal]** Amy ∧ ¬Cal ∧ ¬Bob ∧ (Amy <=> ¬Cal) ∧ (Bob <=> (Amy ∧ Cal)) ∧ (Cal <=> Bob) ∧ ¬Cal

Result: True, False, False

Conclusion: Amy is the truth-teller, and the others are liars.

Sample #5 - More Liars and Truth-tellers:

KB: {Amy ⇔ (Hal ∧ Ida), Bob ⇔ (Amy ∧ Lee), Cal ⇔ (Bob ∧ Gil), Dee ⇔ (Eli ∧ Lee), Eli ⇔ (Cal ∧ Hal), Fay ⇔ (Dee ∧ Ida), Gil ⇔ (¬Eli ∧ ¬Jay), Hal ⇔ (¬Fay ∧ ¬Kay), Ida ⇔ (¬Gil ∧ ¬Kay), Jay ⇔ (¬Amy ∧ ¬Cal), Kay ⇔ (¬Dee ∧ ¬Fay), Lee ⇔ (¬Bob ∧ ¬Jay)}
Alpha: ¬Amy, ¬Bob, ¬Cal, ¬Dee, ¬Eli, ¬Fay, ¬Gil, ¬Hal, ¬Ida, Jay, Kay, ¬Lee

---

ModelChecking.entails(**KnowledgeBases**.***liarsAndTruthTellers3KnowledgeBase***(), NOT(Amy), NOT(Bob), NOT(Cal), NOT(Dee), NOT(Eli), NOT(Fay), NOT(Gil), NOT(Hal), NOT(Ida), Jay, Kay, NOT(Lee));

Satisfied Models:

1. ¬Amy ∧ ¬Hal ∧ ¬Ida ∧ ¬Bob ∧ ¬Lee ∧ ¬Cal ∧ ¬Gil ∧ ¬Dee ∧ ¬Eli ∧ ¬Fay ∧ Jay ∧ Kay ∧ (Amy ⇔ (Hal ∧ Ida)) ∧ (Bob ⇔ (Amy ∧ Lee)) ∧ (Cal ⇔ (Bob ∧ Gil)) ∧ (Dee ⇔ (Eli ∧ Lee)) ∧ (Eli ⇔ (Cal ∧ Hal)) ∧ (Fay ⇔ (Dee ∧ Ida)) ∧ (Gil ⇔ (¬Eli ∧ ¬Jay)) ∧ (Hal ⇔ (¬Fay ∧ ¬Kay)) ∧ (Ida ⇔ (¬Gil ∧ ¬Kay)) ∧ (Jay ⇔ (¬Amy ∧ ¬Cal)) ∧ (Kay ⇔ (¬Dee ∧ ¬Fay)) ∧ (Lee ⇔ (¬Bob ∧ ¬Jay)) ∧ (¬Amy ∧ ¬Bob ∧ ¬Cal ∧ ¬Dee ∧ ¬Eli ∧ ¬Fay ∧ ¬Gil ∧ ¬Hal ∧ ¬Ida ∧ Jay ∧ Kay ∧ ¬Lee)

Result: True

Conclusion: I combined the tests and results to make it more intuitive to get the result that only Jay and Kay are truth-tellers, and the others are all liars.

Sample #6a - The Doors of Enlightenment*:

KB: {a ⇔ x, b ⇔ (y ∨ z), c ⇔ (a ∧ b), d ⇔ (x ∧ y), e ⇔ (x ∧ z), f ⇔ ((d ∧ ¬e) ∨ (¬d ∧ e)), g ⇔ (c ⇒ f), h ⇔ ((g ∧ h) ⇒ a), (x ∨ y ∨ z ∨ w)} **Alpha**: {x, ¬x, y, ¬y, z, ¬z, w, ¬w}

ModelChecking.entails(**KnowledgeBases**.***doorsOfEnlightenment1KnowledgeBase***(), x);
ModelChecking.entails(**KnowledgeBases**.***doorsOfEnlightenment1KnowledgeBase***(),NOT(x));
ModelChecking.entails(**KnowledgeBases**.***doorsOfEnlightenment1KnowledgeBase***(), y);
ModelChecking.entails(**KnowledgeBases**.***doorsOfEnlightenment1KnowledgeBase***(),NOT(y));
ModelChecking.entails(**KnowledgeBases**.***doorsOfEnlightenment1KnowledgeBase***(), z);
ModelChecking.entails(**KnowledgeBases**.***doorsOfEnlightenment1KnowledgeBase***(),NOT(z));
ModelChecking.entails(**KnowledgeBases**.***doorsOfEnlightenment1KnowledgeBase***(), w);
ModelChecking.entails(**KnowledgeBases**.***doorsOfEnlightenment1KnowledgeBase***(),NOT(w));

Satisfied Models for {x}:

Model #1:[A ∧ X ∧ ¬B ∧ ¬Y ∧ ¬Z ∧ ¬C ∧ ¬D ∧ ¬E ∧ ¬F ∧ G ∧ H ∧ ¬W ∧ (A <=> X) ∧ (B <=> (Y ∨ Z)) ∧ (C <=> (A ∧ B)) ∧ (D <=> (X ∧ Y)) ∧ (E <=> (X ∧ Z)) ∧ (F <=> ((D ∧ ¬E) ∨ (¬D ∧ E))) ∧ (G <=> (C ==> F)) ∧ (H <=> ((G ∧ H) ==> A)) ∧ (X ∨ Y ∨ Z ∨ W) ∧ X]

Model #2:[A ∧ X ∧ ¬B ∧ ¬Y ∧ ¬Z ∧ ¬C ∧ ¬D ∧ ¬E ∧ ¬F ∧ G ∧ H ∧ W ∧ (A <=> X) ∧ (B <=> (Y ∨ Z)) ∧ (C <=> (A ∧ B)) ∧ (D <=> (X ∧ Y)) ∧ (E <=> (X ∧ Z)) ∧ (F <=> ((D ∧ ¬E) ∨ (¬D ∧ E))) ∧ (G <=> (C ==> F)) ∧ (H <=> ((G ∧ H) ==> A)) ∧ (X ∨ Y ∨ Z ∨ W) ∧ X]

Model #3:[A ∧ X ∧ B ∧ ¬Y ∧ Z ∧ C ∧ ¬D ∧ E ∧ F ∧ G ∧ H ∧ ¬W ∧ (A <=> X) ∧ (B <=> (Y ∨ Z)) ∧ (C <=> (A ∧ B)) ∧ (D <=> (X ∧ Y)) ∧ (E <=> (X ∧ Z)) ∧ (F <=> ((D ∧ ¬E) ∨ (¬D ∧ E))) ∧ (G <=> (C ==> F)) ∧ (H <=> ((G ∧ H) ==> A)) ∧ (X ∨ Y ∨ Z ∨ W) ∧ X]

Model #4:[A ∧ X ∧ B ∧ ¬Y ∧ Z ∧ C ∧ ¬D ∧ E ∧ F ∧ G ∧ H ∧ W ∧ (A <=> X) ∧ (B <=> (Y ∨ Z)) ∧ (C <=> (A ∧ B)) ∧ (D <=> (X ∧ Y)) ∧ (E <=> (X ∧ Z)) ∧ (F <=> ((D ∧ ¬E) ∨ (¬D ∧ E))) ∧ (G <=> (C ==> F)) ∧ (H <=> ((G ∧ H) ==> A)) ∧ (X ∨ Y ∨ Z ∨ W) ∧ X]

Model #5:[A ∧ X ∧ B ∧ Y ∧ ¬Z ∧ C ∧ D ∧ ¬E ∧ F ∧ G ∧ H ∧ ¬W ∧ (A <=> X) ∧ (B <=> (Y ∨ Z)) ∧ (C <=> (A ∧ B)) ∧ (D <=> (X ∧ Y)) ∧ (E <=> (X ∧ Z)) ∧ (F <=> ((D ∧ ¬E) ∨ (¬D ∧ E))) ∧ (G <=> (C ==> F)) ∧ (H <=> ((G ∧ H) ==> A)) ∧ (X ∨ Y ∨ Z ∨ W) ∧ X]

Model #6:[A ∧ X ∧ B ∧ Y ∧ ¬Z ∧ C ∧ D ∧ ¬E ∧ F ∧ G ∧ H ∧ W ∧ (A <=> X) ∧ (B <=> (Y ∨ Z)) ∧ (C <=> (A ∧ B)) ∧ (D <=> (X ∧ Y)) ∧ (E <=> (X ∧ Z)) ∧ (F <=> ((D ∧ ¬E) ∨ (¬D ∧ E))) ∧ (G <=> (C ==> F)) ∧ (H <=> ((G ∧ H) ==> A)) ∧ (X ∨ Y ∨ Z ∨ W) ∧ X]

Model #7:[A ∧ X ∧ B ∧ Y ∧ Z ∧ C ∧ D ∧ E ∧ ¬F ∧ ¬G ∧ H ∧ ¬W ∧ (A <=> X) ∧ (B <=> (Y ∨ Z)) ∧ (C <=> (A ∧ B)) ∧ (D <=> (X ∧ Y)) ∧ (E <=> (X ∧ Z)) ∧ (F <=> ((D ∧ ¬E) ∨ (¬D ∧ E))) ∧ (G <=> (C ==> F)) ∧ (H <=> ((G ∧ H) ==> A)) ∧ (X ∨ Y ∨ Z ∨ W) ∧ X]

Model #8:[A ∧ X ∧ B ∧ Y ∧ Z ∧ C ∧ D ∧ E ∧ ¬F ∧ ¬G ∧ H ∧ W ∧ (A <=> X) ∧ (B <=> (Y ∨ Z)) ∧ (C <=> (A ∧ B)) ∧ (D <=> (X ∧ Y)) ∧ (E <=> (X ∧ Z)) ∧ (F <=> ((D ∧ ¬E) ∨ (¬D ∧ E))) ∧ (G <=> (C ==> F)) ∧ (H <=> ((G ∧ H) ==> A)) ∧ (X ∨ Y ∨ Z ∨ W) ∧ X]

Unsatisfied Models for {y}:

False L3078:[A ∧ X ∧ ¬B ∧ ¬Y ∧ ¬Z ∧ ¬C ∧ ¬D ∧ ¬E ∧ ¬F ∧ G ∧ H ∧ ¬W ∧ (A <=> X) ∧ (B <=> (Y ∨ Z)) ∧ (C <=> (A ∧ B)) ∧ (D <=> (X ∧ Y)) ∧ (E <=> (X ∧ Z)) ∧ (F <=> ((D ∧ ¬E) ∨ (¬D ∧ E))) ∧ (G <=> (C ==> F)) ∧ (H <=> ((G ∧ H) ==> A)) ∧ (X ∨ Y ∨ Z ∨ W) ∧ ¬Y]

False L3079:[A ∧ X ∧ ¬B ∧ ¬Y ∧ ¬Z ∧ ¬C ∧ ¬D ∧ ¬E ∧ ¬F ∧ G ∧ H ∧ W ∧ (A <=> X) ∧ (B <=> (Y ∨ Z)) ∧ (C <=> (A ∧ B)) ∧ (D <=> (X ∧ Y)) ∧ (E <=> (X ∧ Z)) ∧ (F <=> ((D ∧ ¬E) ∨ (¬D ∧ E))) ∧ (G <=> (C ==> F)) ∧ (H <=> ((G ∧ H) ==> A)) ∧ (X ∨ Y ∨ Z ∨ W) ∧ ¬Y]

False L3806:[A ∧ X ∧ B ∧ ¬Y ∧ Z ∧ C ∧ ¬D ∧ E ∧ F ∧ G ∧ H ∧ ¬W ∧ (A <=> X) ∧ (B <=> (Y ∨ Z)) ∧ (C <=> (A ∧ B)) ∧ (D <=> (X ∧ Y)) ∧ (E <=> (X ∧ Z)) ∧ (F <=> ((D ∧ ¬E) ∨ (¬D ∧ E))) ∧ (G <=> (C ==> F)) ∧ (H <=> ((G ∧ H) ==> A)) ∧ (X ∨ Y ∨ Z ∨ W) ∧ ¬Y]

False L3807:[A ∧ X ∧ B ∧ ¬Y ∧ Z ∧ C ∧ ¬D ∧ E ∧ F ∧ G ∧ H ∧ W ∧ (A <=> X) ∧ (B <=> (Y ∨ Z)) ∧ (C <=> (A ∧ B)) ∧ (D <=> (X ∧ Y)) ∧ (E <=> (X ∧ Z)) ∧ (F <=> ((D ∧ ¬E) ∨ (¬D ∧ E))) ∧ (G <=> (C ==> F)) ∧ (H <=> ((G ∧ H) ==> A)) ∧ (X ∨ Y ∨ Z ∨ W) ∧ ¬Y]

Unsatisfied Models for {z}:

False L3078:[A ∧ X ∧ ¬B ∧ ¬Y ∧ ¬Z ∧ ¬C ∧ ¬D ∧ ¬E ∧ ¬F ∧ G ∧ H ∧ ¬W ∧ (A <=> X) ∧ (B <=> (Y ∨ Z)) ∧ (C <=> (A ∧ B)) ∧ (D <=> (X ∧ Y)) ∧ (E <=> (X ∧ Z)) ∧ (F <=> ((D ∧ ¬E) ∨ (¬D ∧ E))) ∧ (G <=> (C ==> F)) ∧ (H <=> ((G ∧ H) ==> A)) ∧ (X ∨ Y ∨ Z ∨ W) ∧ ¬Z]
False L3079:[A ∧ X ∧ ¬B ∧ ¬Y ∧ ¬Z ∧ ¬C ∧ ¬D ∧ ¬E ∧ ¬F ∧ G ∧ H ∧ W ∧ (A <=> X) ∧ (B <=> (Y ∨ Z)) ∧ (C <=> (A ∧ B)) ∧ (D <=> (X ∧ Y)) ∧ (E <=> (X ∧ Z)) ∧ (F <=> ((D ∧ ¬E) ∨ (¬D ∧ E))) ∧ (G <=> (C ==> F)) ∧ (H <=> ((G ∧ H) ==> A)) ∧ (X ∨ Y ∨ Z ∨ W) ∧ ¬Z]
False L3950:[A ∧ X ∧ B ∧ Y ∧ ¬Z ∧ C ∧ D ∧ ¬E ∧ F ∧ G ∧ H ∧ ¬W ∧ (A <=> X) ∧ (B <=> (Y ∨ Z)) ∧ (C <=> (A ∧ B)) ∧ (D <=> (X ∧ Y)) ∧ (E <=> (X ∧ Z)) ∧ (F <=> ((D ∧ ¬E) ∨ (¬D ∧ E))) ∧ (G <=> (C ==> F)) ∧ (H <=> ((G ∧ H) ==> A)) ∧ (X ∨ Y ∨ Z ∨ W) ∧ ¬Z]
False L3951:[A ∧ X ∧ B ∧ Y ∧ ¬Z ∧ C ∧ D ∧ ¬E ∧ F ∧ G ∧ H ∧ W ∧ (A <=> X) ∧ (B <=> (Y ∨ Z)) ∧ (C <=> (A ∧ B)) ∧ (D <=> (X ∧ Y)) ∧ (E <=> (X ∧ Z)) ∧ (F <=> ((D ∧ ¬E) ∨ (¬D ∧ E))) ∧ (G <=> (C ==> F)) ∧ (H <=> ((G ∧ H) ==> A)) ∧ (X ∨ Y ∨ Z ∨ W) ∧ ¬Z]

Unsatisfied Models for {w}:
False L3078:[A ∧ X ∧ ¬B ∧ ¬Y ∧ ¬Z ∧ ¬C ∧ ¬D ∧ ¬E ∧ ¬F ∧ G ∧ H ∧ ¬W ∧ (A <=> X) ∧ (B <=> (Y ∨ Z)) ∧ (C <=> (A ∧ B)) ∧ (D <=> (X ∧ Y)) ∧ (E <=> (X ∧ Z)) ∧ (F <=> ((D ∧ ¬E) ∨ (¬D ∧ E))) ∧ (G <=> (C ==> F)) ∧ (H <=> ((G ∧ H) ==> A)) ∧ (X ∨ Y ∨ Z ∨ W) ∧ ¬W]
False L3806:[A ∧ X ∧ B ∧ ¬Y ∧ Z ∧ C ∧ ¬D ∧ E ∧ F ∧ G ∧ H ∧ ¬W ∧ (A <=> X) ∧ (B <=> (Y ∨ Z)) ∧ (C <=> (A ∧ B)) ∧ (D <=> (X ∧ Y)) ∧ (E <=> (X ∧ Z)) ∧ (F <=> ((D ∧ ¬E) ∨ (¬D ∧ E))) ∧ (G <=> (C ==> F)) ∧ (H <=> ((G ∧ H) ==> A)) ∧ (X ∨ Y ∨ Z ∨ W) ∧ ¬W]
False L3950:[A ∧ X ∧ B ∧ Y ∧ ¬Z ∧ C ∧ D ∧ ¬E ∧ F ∧ G ∧ H ∧ ¬W ∧ (A <=> X) ∧ (B <=> (Y ∨ Z)) ∧ (C <=> (A ∧ B)) ∧ (D <=> (X ∧ Y)) ∧ (E <=> (X ∧ Z)) ∧ (F <=> ((D ∧ ¬E) ∨ (¬D ∧ E))) ∧ (G <=> (C ==> F)) ∧ (H <=> ((G ∧ H) ==> A)) ∧ (X ∨ Y ∨ Z ∨ W) ∧ ¬W]
False L4082:[A ∧ X ∧ B ∧ Y ∧ Z ∧ C ∧ D ∧ E ∧ ¬F ∧ ¬G ∧ H ∧ ¬W ∧ (A <=> X) ∧ (B <=> (Y ∨ Z)) ∧ (C <=> (A ∧ B)) ∧ (D <=> (X ∧ Y)) ∧ (E <=> (X ∧ Z)) ∧ (F <=> ((D ∧ ¬E) ∨ (¬D ∧ E))) ∧ (G <=> (C ==> F)) ∧ (H <=> ((G ∧ H) ==> A)) ∧ (X ∨ Y ∨ Z ∨ W) ∧ ¬W]

Result:
True for {x}, False for {y}, False for {z}, False for {w}
False for {¬x}, False for {¬y}, False for {¬z}, False for {¬w}

Conclusion: Only x is provable, so the philosopher should choose the door x.

Sample #6b - The Doors of Enlightenment*:

KB: {a ⇔ x, c ⇔ a, g ∨ c, h ⇔ ((g∧h) ⇒ a)}   **Alpha**: {x, ¬x, y, ¬y, z, ¬z, w, ¬w}

ModelChecking.entails(**KnowledgeBases**.***doorsOfEnlightenment2KnowledgeBase***(), x);
ModelChecking.entails(**KnowledgeBases**.***doorsOfEnlightenment2KnowledgeBase***(),NOT(x));
ModelChecking.entails(**KnowledgeBases**.***doorsOfEnlightenment2KnowledgeBase***(), y);
ModelChecking.entails(**KnowledgeBases**.***doorsOfEnlightenment2KnowledgeBase***(),NOT(y));
ModelChecking.entails(**KnowledgeBases**.***doorsOfEnlightenment2KnowledgeBase***(), z);
ModelChecking.entails(**KnowledgeBases**.***doorsOfEnlightenment2KnowledgeBase***(),NOT(z));
ModelChecking.entails(**KnowledgeBases**.***doorsOfEnlightenment2KnowledgeBase***(), w);
ModelChecking.entails(**KnowledgeBases**.***doorsOfEnlightenment2KnowledgeBase***(),NOT(w));

Satisfied Models:
1. **[check X]** A ∧ X ∧ C ∧ ¬G ∧ H ∧ (A <=> X) ∧ (C <=> A) ∧ (G ∨ C) ∧ (H <=> ((G ∧ H) ==> A)) ∧ X
2. **[check X]** A ∧ X ∧ C ∧ G ∧ H ∧ (A <=> X) ∧ (C <=> A) ∧ (G ∨ C) ∧ (H <=> ((G ∧ H) ==> A)) ∧ X
3. **[all other checks are unsatisfied, so we omitted these]**

Result:
True for {x}, False for {y}, False for {z}, False for {w}
False for {¬x}, False for {¬y}, False for {¬z}, False for {¬w}

Conclusion: The result is the same as sample #6a indicates that the philosopher had heard enough to make a decision.

Part II: Advanced Propositional Inference

- **Program Design**

  We implemented the resolution-based theorem prover in our project.

  Our implementation is based on the algorithm given in the textbook. However, several improvements are made. The flowchart of our core module shows in *Figure 3*.

  First, it validates if the *KB* and *Alpha sentences* are both in conjunctive normal form, if the check not passed, a converter will work to convert these sentences into CNF for future processing.

  The CNF converter will handle raw sentences in the following order:

  1. Convert bidirectional implications '⇔'.
  2. Eliminate unidirectional implications '⇒'.
  3. Move negatives '¬' inside parentheses.
  4. Remove conjunctive connectives '∧'

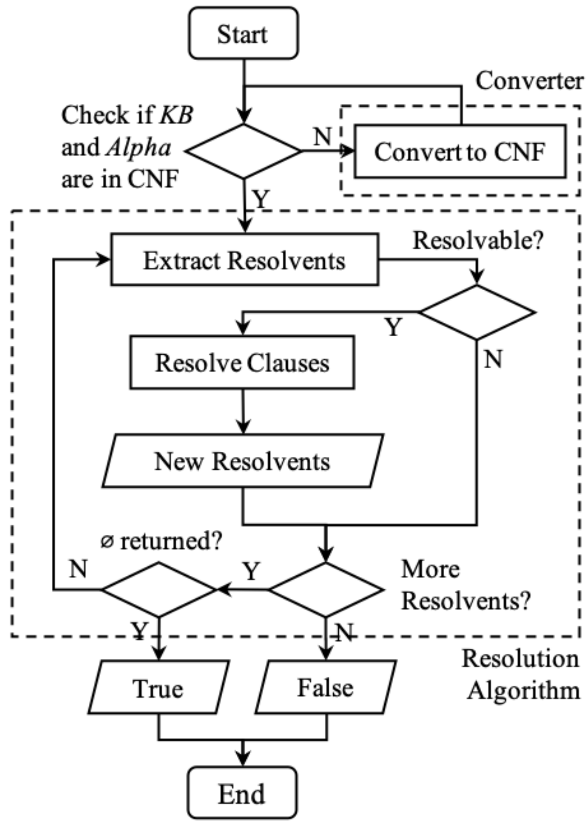

Figure 3: Resolution

After CNF validation, starts our main algorithm, we add all converted *KB* sentences and negated *Alpha* sentences into a set called resolvents. Then, for each clause pairs in that set, we check if they can be resolved. If a pair is able to be resolved, it will create new sentences (we will split the conjunctive connectives into separate sentences, so the outcome might be more than one sentence). After that, we check the returned set (named *newResolvents* in our program). If that set contains an empty clause (means we derived into a contradiction that returns a ∅ clause), we skip the loop and directly return result True. Otherwise, we keep tracking the size of generated resolvents, if nothing changed in an iteration, we will return a False, for reason that no new logic represents were created to prove us getting a contradiction.

The main difference we made in our implementation is that we skip useless resolvents created which saves a considerable amount of time. We ignore lengthy clauses in the case when our new production is larger or equal than the maximum of the unresolved sentences because our target is to resolve clauses until we find a contradiction. The derived and skipped sentences are actually implied by our previous resolvents, so it does not matter if we didn't append them to current resolvents. Therefore, the only thing we need to ensure is to keep all the clauses in resolvents combinable (that is, we should never ignore a sentence in our resolvents). Based on the above conditions, we can choose clauses with good properties (shorter in length) for out new-generation resolvents.

- **Implementation & Performance**

Here are standard outputs from program part 2 (partial contents are omitted):

Sample #1 - Modus Ponens:

KB: [P, (P ==> Q)] **α:** [Q] **Converted CNF:** P (Q ∨ ¬P) **Resolution Trace:** resolve P, P resolve P, (Q ∨ ¬P) add new [Q] resolve ¬Q, P resolve ¬Q, (Q ∨ ¬P)	add new [¬P] resolve (Q ∨ ¬P), P add new [Q] resolve (Q ∨ ¬P), (Q ∨ ¬P) resolve P, P resolve P, ¬Q resolve P, Q resolve P, (Q ∨ ¬P) add new [Q] resolve P, ¬P **Is Q true? true**

Sample #2 - Wumpus World (Simple):

KB: [¬P11, (B11 <=> (P12 ∨ P21)), (B21 <=> (P11 ∨ P22 ∨ P31)), ¬B11, B21] **α:** [P12] **Converted CNF:** ¬B11 B21 ¬P11 (P12 ∨ ¬B11 ∨ P21) (B21 ∨ ¬P11) (B21 ∨ ¬P22) (¬P21 ∨ B11) (¬P12 ∨ B11) (¬B21 ∨ P11 ∨ P22 ∨ P31) (B21 ∨ ¬P31) **Resolution Trace:**	resolve ¬P12, ¬B11 resolve ¬P12, B21 resolve ¬P12, ¬P11 resolve ¬P12, (P12 ∨ ¬B11 ∨ P21) add new [(¬B11 ∨ P21)] resolve ¬P12, (B21 ∨ ¬P11) resolve ¬P12, (B21 ∨ ¬P22) resolve ¬P12, (¬P21 ∨ B11) …… resolve (B21 ∨ ¬P31), ¬P11 resolve (B21 ∨ ¬P31), (P12 ∨ ¬B11 ∨ P21) resolve (B21 ∨ ¬P31), (P22 ∨ P31) resolve (B21 ∨ ¬P31), (¬B21 ∨ P11 ∨ P22 ∨ P31) resolve (B21 ∨ ¬P31), (B21 ∨ ¬P31) **Is P12 true? false**

Sample #3a - Horn Clauses:

Can we prove that unicorn is mythical? **Converted CNF:** (magical ∨ ¬horned) (horned ∨ ¬immortal) (¬mythical ∨ immortal) (mythical ∨ ¬immortal) (mammal ∨ mythical) (horned ∨ ¬mammal) **Resolution Trace:** resolve (magical ∨ ¬horned), (magical ∨ ¬horned) resolve (magical ∨ ¬horned), (horned ∨ ¬immortal) resolve (magical ∨ ¬horned), (¬mythical ∨ immortal) resolve (magical ∨ ¬horned), (mythical ∨ ¬immortal) resolve (magical ∨ ¬horned), (mammal ∨ mythical) resolve (magical ∨ ¬horned), (horned ∨ ¬mammal)	resolve (horned ∨ ¬immortal), (magical ∨ ¬horned) resolve (horned ∨ ¬immortal), (horned ∨ ¬immortal) resolve (horned ∨ ¬immortal), (¬mythical ∨ immortal) resolve (horned ∨ ¬immortal), (mythical ∨ ¬immortal) resolve (horned ∨ ¬immortal), (mammal ∨ mythical) …… add new [magical] resolve horned, (horned ∨ ¬immortal) resolve horned, magical resolve horned, (¬mythical ∨ immortal) resolve horned, (mythical ∨ ¬immortal) resolve horned, immortal resolve horned, (mammal ∨ mythical) resolve horned, mythical resolve horned, (horned ∨ ¬mammal) resolve horned, horned **false**

Sample #3b - Horn Clauses:

Can we prove that unicorn is magical? **Converted CNF:** (magical ∨ ¬horned) (horned ∨ ¬immortal) (¬mythical ∨ immortal) (mythical ∨ ¬immortal) (mammal ∨ mythical) (horned ∨ ¬mammal) **Resolution Trace:** resolve (magical ∨ ¬horned), (magical ∨ ¬horned) resolve (magical ∨ ¬horned), (horned ∨ ¬immortal) resolve (magical ∨ ¬horned), (¬mythical ∨ immortal) resolve (magical ∨ ¬horned), (mythical ∨ ¬immortal) resolve (magical ∨ ¬horned), (mammal ∨ mythical)	resolve (magical ∨ ¬horned), (horned ∨ ¬mammal) …… resolve ¬magical, (horned ∨ ¬mammal) resolve ¬magical, ¬immortal resolve ¬magical, ¬mammal resolve ¬mythical, (magical ∨ ¬horned) resolve ¬mythical, (horned ∨ ¬immortal) resolve ¬mythical, (¬mythical ∨ immortal) resolve ¬mythical, (mythical ∨ ¬immortal) add new [¬immortal] resolve ¬mythical, ¬magical resolve ¬mythical, ¬mythical resolve ¬mythical, (mammal ∨ mythical) add new [mammal] resolve ¬mythical, ¬horned resolve ¬mythical, mythical **true** **Is unicorn magical? true**

Sample #3c - Horn Clauses:

Can we prove that unicorn is horned? **Converted CNF:** (magical ∨ ¬horned) (horned ∨ ¬immortal) (¬mythical ∨ immortal) (mythical ∨ ¬immortal) (mammal ∨ mythical) (horned ∨ ¬mammal) **Resolution Trace:** resolve (magical ∨ ¬horned), (magical ∨ ¬horned) resolve (magical ∨ ¬horned), (horned ∨ ¬immortal) resolve (magical ∨ ¬horned), (¬mythical ∨ immortal)	resolve (magical ∨ ¬horned), (mythical ∨ ¬immortal) resolve (magical ∨ ¬horned), (mammal ∨ mythical) resolve (magical ∨ ¬horned), (horned ∨ ¬mammal) …… add new [¬immortal] resolve ¬mythical, ¬mythical resolve ¬mythical, (mammal ∨ mythical) add new [mammal] resolve ¬mythical, ¬horned resolve ¬mythical, mythical **true** **Is unicorn horned? true**

Sample #4a - Liars and Truth-tellers:

Converted CNF: (¬Cal ∨ ¬Amy ∨ Amy) (¬Amy ∨ Amy) (¬Amy ∨ Cal) (Bob ∨ Cal) (¬Cal ∨ ¬Bob) (Bob ∨ ¬Cal ∨ ¬Amy) (¬Bob ∨ Cal) (Amy ∨ Cal) **Resolution Trace:** resolve (¬Cal ∨ ¬Amy ∨ Amy), (¬Cal ∨ ¬Amy ∨ Amy) resolve (¬Cal ∨ ¬Amy ∨ Amy), (¬Amy ∨ Amy) resolve (¬Cal ∨ ¬Amy ∨ Amy), (¬Amy ∨ Cal) resolve (¬Cal ∨ ¬Amy ∨ Amy), (Bob ∨ Cal)	resolve (¬Cal ∨ ¬Amy ∨ Amy), (¬Cal ∨ ¬Bob) resolve (¬Cal ∨ ¬Amy ∨ Amy), (Bob ∨ ¬Cal ∨ ¬Amy) resolve (¬Cal ∨ ¬Amy ∨ Amy), (¬Bob ∨ Cal) …… resolve ¬Cal, (Bob ∨ ¬Cal ∨ ¬Amy) resolve ¬Cal, (Bob ∨ ¬Amy) resolve ¬Cal, (¬Bob ∨ Cal) add new [¬Bob] resolve ¬Cal, Amy resolve ¬Cal, (Amy ∨ Cal) add new [Amy] resolve ¬Cal, ¬Bob resolve ¬Cal, Cal **Is Amy a truth-teller? false** **Is Bob a truth-teller? false** **Is Cal a truth-teller? true**

Sample #4b - Liars and Truth-tellers:

Converted CNF:
(¬Amy ∨ ¬Cal)
(¬Amy ∨ Bob ∨ ¬Cal)
(Bob ∨ ¬Cal)
(¬Bob ∨ Cal)
(Amy ∨ Cal)
(Amy ∨ ¬Bob)

Resolution Trace:
resolve (¬Amy ∨ ¬Cal), (¬Amy ∨ ¬Cal)
resolve (¬Amy ∨ ¬Cal), (¬Amy ∨ Bob ∨ ¬Cal)
resolve (¬Amy ∨ ¬Cal), (Bob ∨ ¬Cal)
resolve (¬Amy ∨ ¬Cal), (¬Bob ∨ Cal)
resolve (¬Amy ∨ ¬Cal), (Amy ∨ Cal)
resolve (¬Amy ∨ ¬Cal), (Amy ∨ ¬Bob)
resolve ¬Amy, (¬Amy ∨ ¬Cal)
resolve ¬Amy, (¬Amy ∨ Bob ∨ ¬Cal)
resolve ¬Amy, (Bob ∨ ¬Cal)
resolve ¬Amy, (¬Bob ∨ Cal)
resolve ¬Amy, (Amy ∨ Cal)
add new [Cal]
……
resolve (Amy ∨ Cal), ¬Cal
add new [Amy]
resolve (Amy ∨ Cal), (¬Amy ∨ Bob ∨ ¬Cal)
resolve (Amy ∨ Cal), (Bob ∨ ¬Cal)
resolve (Amy ∨ Cal), (¬Bob ∨ Cal)
resolve (Amy ∨ Cal), ¬Bob
resolve (Amy ∨ Cal), Amy
resolve (Amy ∨ Cal), (Amy ∨ Cal)
resolve (Amy ∨ Cal), (Amy ∨ ¬Bob)
resolve (Amy ∨ ¬Bob), (¬Amy ∨ ¬Cal)
resolve (Amy ∨ ¬Bob), ¬Cal
resolve (Amy ∨ ¬Bob), (¬Amy ∨ Bob ∨ ¬Cal)
resolve (Amy ∨ ¬Bob), (Bob ∨ ¬Cal)
resolve (Amy ∨ ¬Bob), (¬Bob ∨ Cal)
resolve (Amy ∨ ¬Bob), ¬Bob
resolve (Amy ∨ ¬Bob), Amy
resolve (Amy ∨ ¬Bob), (Amy ∨ Cal)
resolve (Amy ∨ ¬Bob), (Amy ∨ ¬Bob)

Is Amy a truth-teller? tree
Is Bob a truth-teller? false
Is Cal a truth-teller? false

Sample #5 - More Liars and Truth-tellers:

Converted CNF:
(Kay ∨ Hal ∨ Fay)
(Kay ∨ Ida ∨ Gil)
(Hal ∨ ¬Amy)
(¬Eli ∨ Dee ∨ ¬Lee)
(¬Fay ∨ Dee)
(¬Dee ∨ Eli)
(¬Cal ∨ Gil)
(¬Dee ∨ ¬Ida ∨ Fay)
(¬Hal ∨ ¬Ida ∨ Amy)
(¬Dee ∨ ¬Kay)
(Jay ∨ Bob ∨ Lee)
(¬Hal ∨ Eli ∨ ¬Cal)
(¬Dee ∨ Lee)
(¬Kay ∨ ¬Hal)
(¬Gil ∨ ¬Ida)
(Amy ∨ ¬Bob)
(¬Jay ∨ ¬Cal)
(Lee ∨ ¬Bob)
(¬Hal ∨ ¬Fay)
(¬Gil ∨ ¬Bob ∨ Cal)
(¬Kay ∨ ¬Ida)
(Kay ∨ Dee ∨ Fay)
(¬Eli ∨ Cal)
(Eli ∨ Jay ∨ Gil)
(¬Lee ∨ ¬Bob)
(¬Kay ∨ ¬Fay)
(¬Amy ∨ Ida)
(¬Eli ∨ Hal)

Resolution Trace:
resolve (Kay ∨ Hal ∨ Fay), (Kay ∨ Hal ∨ Fay)
resolve (Kay ∨ Hal ∨ Fay), (Kay ∨ Ida ∨ Gil)
resolve (Kay ∨ Hal ∨ Fay), (Hal ∨ ¬Amy)
resolve (Kay ∨ Hal ∨ Fay), (¬Eli ∨ Dee ∨ ¬Lee)
resolve (Kay ∨ Hal ∨ Fay), (¬Fay ∨ Dee)
resolve (Kay ∨ Hal ∨ Fay), (¬Dee ∨ Eli)
resolve (Kay ∨ Hal ∨ Fay), (¬Cal ∨ Gil)
……
resolve (Kay ∨ Gil), (¬Hal ∨ ¬Ida ∨ Amy)

(¬Jay ∨ ¬Lee) (¬Fay ∨ Ida) (Bob ∨ ¬Amy ∨ ¬Lee) (Bob ∨ ¬Cal) (Jay ∨ Amy ∨ Cal) (¬Jay ∨ ¬Amy) (¬Eli ∨ ¬Gil) (¬Jay ∨ ¬Gil)	resolve (Kay ∨ Gil), (¬Eli ∨ Cal) resolve (Kay ∨ Gil), (Jay ∨ Lee) resolve (Kay ∨ Gil), (¬Lee ∨ ¬Bob) resolve (Kay ∨ Gil), (¬Amy ∨ Ida) resolve (Kay ∨ Gil), (Kay ∨ Gil) **Is Jay a truth-teller? true** **Is Kay a truth-teller? true** **(All other results are false)**

Sample #6a - The Doors of Enlightenment

Can we prove if X is a good door? **Converted CNF:** (A ∨ ¬C) (B ∨ ¬C) H (¬D ∨ E ∨ F) (¬E ∨ D ∨ F) (D ∨ E ∨ ¬F) (¬F ∨ G) (H ∨ ¬A) (¬B ∨ Y ∨ Z) (C ∨ G) (¬D ∨ ¬E ∨ ¬F) (G ∨ H) (¬X ∨ D ∨ ¬Y) (¬X ∨ E ∨ ¬Z) (W ∨ X ∨ Y ∨ Z) (¬D ∨ D ∨ ¬F) (¬C ∨ F ∨ ¬G) (A ∨ ¬H ∨ ¬G) (¬E ∨ E ∨ ¬F) (¬E ∨ Z) (B ∨ ¬Z) (¬D ∨ X) (¬D ∨ Y) (¬E ∨ X) (¬B ∨ C ∨ ¬A) (B ∨ ¬Y)	(X ∨ ¬A) (A ∨ ¬X) **Resolution Trace:** resolve (A ∨ ¬C), (A ∨ ¬C) resolve (A ∨ ¬C), (B ∨ ¬C) resolve (A ∨ ¬C), H resolve (A ∨ ¬C), (¬D ∨ E ∨ F) …… resolve (A ∨ ¬X), (¬D ∨ Y) resolve (A ∨ ¬X), (¬E ∨ X) resolve (A ∨ ¬X), (¬B ∨ C ∨ ¬A) resolve (A ∨ ¬X), (B ∨ ¬Y) resolve (A ∨ ¬X), (X ∨ ¬A) resolve (A ∨ ¬X), (A ∨ ¬X) …… **true** **Is X a good door? true** **Can we prove if Y is a good door?** …… **false** **Can we prove if Z is a good door?** …… **false** **Can we prove if W is a good door?** …… **false**

Sample #6b - The Doors of Enlightenment

Can we prove if X is a good door? **Converted CNF:** $(A \vee \neg C)$ $(C \vee \neg A)$ $(G \vee H)$ $(A \vee \neg H \vee \neg G)$ $(C \vee G)$ H $(X \vee \neg A)$ $(A \vee \neg X)$ $(H \vee \neg A)$ **Resolution Trace:** resolve $(A \vee \neg C)$, $(A \vee \neg C)$ resolve $(A \vee \neg C)$, $(C \vee \neg A)$ resolve $(A \vee \neg C)$, $(G \vee H)$ resolve $(A \vee \neg C)$, $(A \vee \neg H \vee \neg G)$ resolve $(A \vee \neg C)$, $(C \vee G)$ resolve $(A \vee \neg C)$, H resolve $(A \vee \neg C)$, $(X \vee \neg A)$ resolve $(A \vee \neg C)$, $(A \vee \neg X)$ resolve $(A \vee \neg C)$, $(H \vee \neg A)$ resolve $(C \vee \neg A)$, $(A \vee \neg C)$ resolve $(C \vee \neg A)$, $(C \vee \neg A)$ resolve $(C \vee \neg A)$, $(G \vee H)$ resolve $(C \vee \neg A)$, $(A \vee \neg H \vee \neg G)$ resolve $(C \vee \neg A)$, $(C \vee G)$ resolve $(C \vee \neg A)$, H	resolve $(C \vee \neg A)$, $(X \vee \neg A)$ resolve $(C \vee \neg A)$, $(A \vee \neg X)$ resolve $(C \vee \neg A)$, $(H \vee \neg A)$ resolve $\neg X$, $(A \vee \neg C)$ resolve $\neg X$, $(C \vee \neg A)$ …… resolve $(A \vee \neg G)$, $(C \vee G)$ resolve $(A \vee \neg G)$, H resolve $(A \vee \neg G)$, $(X \vee \neg A)$ resolve $(A \vee \neg G)$, $(A \vee \neg X)$ resolve $(A \vee \neg G)$, $(H \vee \neg A)$ resolve $(A \vee \neg G)$, $(A \vee \neg G)$ …… **true** **Is X a good door? true** **Can we prove if Y is a good door?** …… **false** **Can we prove if Z is a good door?** …… **false** **Can we prove if W is a good door?** …… **false**

Execution time:

Sample #1	Sample #2	Sample #3a	Sample #3b	Sample #3c
≈0ms	4ms	6ms	6ms	9ms
Sample #4a	**Sample #4b**	**Sample #5**	**Sample #6a**	**Sample #6b**
7ms	8ms	199ms	93ms	32ms