CSC442 Artificial Intelligence
Project #2

Andrii Osipa
URID: aosipa

October 2017

1 Definitions

The main data structure in this work is a plain string. Reasons for that is that it is easier to write statements and they have more understandable form. Other point is that many operations may be easily preformed with regular expressions and it seems much easier to perform changes on the string compared to class that has class(es) as parameter(s) that has class(es) as parameter(s) ... And one more point is ease of evaluation. On the other hand, notation that is used in this work has some conceptual similarities with java classes posted by Prof.Ferguson.

- Sentence is a string, which must start and end with a bracket.
- Knowledge base(KB) is a list of sentences.
- Clause is a string which must start and end with a bracket.
- CNF is a list of clauses.

All functions assume correct inputs, except some minor cases sometimes, e.g first and last brackets or not required brackets.

1.1 Formulas representation

In current work we introduce the following notation:

- Logical and $\wedge$ is &.
- Logical or $\vee$ is |.
- Logical implication $\rightarrow$ is $->$.
- Logical equivalence $\leftrightarrow$ is $<->$.
- Logical not $\neg$ is $\sim$.
- Logical truth T is 1 and F is 0.

All the special symbols mentioned above are not allowed to be used as a part of variables(constants) names. Everything* else is allowed, except no spaces.

The sentence is a string with no spaces. In a sentence, each operation must be bracketed, except negation which may be or may be not bracketed. E.g. $A \vee B \vee C$ will be $(A|(B|C))$ or $((A|B)|C)$. Same holds for implication, equivalence and logical and.

For clause only first and last brackets required, so it will have a normal view(as we used in real life). Clauses consist only from symbols and $|, \sim$.

Examples: $(\sim P11)$, $(B11 <-> (P12\|P21))$, $(B21 <-> ((P11\|P22)\|P31))$, $(\sim B11), (B21)$.

1.2 Models

Each model is a dictionary with strings as keys and strings '0' or '1' as values which correspond to T and F.

2 Basic Model Checking

2.1 Sentence evaluation

Each sentence is evaluated by substituting values given in a model and then looping through it while there exists a part of form (...), where there are no brackets inside. Basically, those are supposed to be terms like $0|0$, $0|1$, $1|0$, $1|\ 1$, $0->\ 0$, etc. Those we will call simple sentences and will evaluate them using special function and replace them with their value in $\{0, 1\}$. On each step we will also replace negated elements by doing simple substitutions $\sim 0 \to 1$ and $\sim 1 \to 0$; and remove useless brackets, e.g. when we have (0) or (1).

After no parts of the form (...) left means that we just ended with a string '0' or '1', each of which can be translated into int and then corresponding boolean value.

2.2 Simple sentence evaluation

Evaluation for $x * y$, where $x, y \in \{0, 1, \sim 0, \sim 1\}$ and $* \in \{\&, |, ->, <->\}$ is done just by checking if input is equal to particular case for each case. There are only 64 cases, so it is not hard.

2.3 Model Checking

There is a function implemented that will check if a given model satisfies KB, which works in an obvious way using two previously discussed subroutines.

Algorithm from AIMA Figure 7.10 is implemented to check if $KB \models \alpha$, which means that each model of KB is a model of α.

And some utility functions to check if KB is satisfied with the model which will do sentence evaluation procedure for each sentence in KB if the model has assignments for every variable in KB.

Comments about this inference procedure: it will require to loop through all 2^n possible models, where n is a number of variables in KB and α. This is not bad, it's a stable result and it will be quite fast for not big models. Good point is that the result is a proved one(not probabilistic).

3 Advanced Propositional Inference

3.1 Conversion to CNF

Next two methods discussed require KB to be in a CNF form.

I did my own algorithm to transform a sentence into CNF (means I have not looked at what's was done by Prof.Ferguson).

Algorithm for this is recursive and quite simple:

- If we have v, where v is variable then return v.
- If we have $P \wedge Q$ then return $CNF(P) \wedge CNF(Q)$.
- If we have $P \vee Q$ then let $CNF(P) = P_1 \wedge ... \wedge P_k$ and $CNF(Q) = Q_1 \wedge ... \wedge Q_l$. Return $(P_1 \vee Q_1) \wedge ... \wedge (P_1 \vee Q_l) \wedge ... \wedge (P_k \vee Q_l)$.
- If we have $\neg\neg P$ then return $CNF(P)$.
- If we have $\neg(P \wedge Q)$ then return $CNF(\neg P \vee \neg Q)$.
- If we have $\neg(P \vee Q)$ then return $CNF(\neg P \wedge \neg Q)$.
- If we have $P \to Q$ then return $CNF(\neg P \vee Q)$.

- If we have $P \leftrightarrow Q$ then return $CNF((P \rightarrow Q) \wedge (Q \rightarrow P)$.

And these statements were transferred into 13 cases that represent them. The number of cases increased as some cases were split to cases when we have variables or formulas and also as implication and equivalence are not tracked inside a formula so there are some additional cases.

To find all these cases I used regular expressions and some additional functions that 'correct' formulas or check if they can be represented as $x * y$, where $*$ is some operation and both x and y are valid.

The task of converting to CNF is not that easy as their many small problems may occur as formulas may take infinite amounts of different forms. I tested mine mostly on the formulas from examples and also on some other ones.

Note about my implementation: I do not filter resulted list of clauses to make them optimal, e.g. there may occur some like $A|A$ or $B|A| \sim A$ or $[A|B, B, C|A]$, which is equivalent to A, () and $[B, C|A]$. This leads to more clauses generated in resolution algorithm and therefore increases run time. For walksat, this does not really change the situation.

3.2 Walksat Algorithm

I picked this algorithm just because I like it (it also happened that it's simple) and because it's fast enough, compared to two others in this work. Implementation is just from the AIMA book. Default parameters are $p = 0.5$ and limit on flips is 1000. Obviously, limit of 1000 may be improved and it could depend on a number of distinct variables as if we try to find a satisfiable assignment for an unsatisfiable formula with 1 or 2 or 3 or 5 variables then the algorithm will loop through all possible assignments(with some probability, of cause).

Disadvantages of the algorithm: If the sentence is unsatisfiable we will never be 100% that it is.

3.3 Resolution Algorithm

I implemented this algorithm exactly of the reason stated in the end of the previous paragraph.

The basic implementation is exactly from AIMA book. Changes/improvements that I did in the algorithm:

- Doing resolution for $KB+\alpha$ and $KB+\neg\alpha$ in parallel: as we know the resolution is refutation complete, so if the problem does not have a contradiction then it will produce new and new clauses until we have produced every possible clause(in case of a finite model) or infinitely long. Generating all possible clauses is time-consuming as there are $O(n!3^n)$ of them for n variables. And on each step, we will compute up to m^2 new clauses, where m is a number of clauses initially on this step. Doing both tasks in parallel gives us a chance to find an answer without infinite loop if the formula in question is either satisfiable or not.
- There are $O(n!3^n)$ unordered clauses. To reduce this number each new clause will be sorted in alphabetical order(variable names). This will reduce complexity a bit.
- I also added a limit for number steps(when we produce all possible new clauses from given) as this causes also huge problems - computation becomes long and that's not what I am looking for in CSC 442 Assignment.

4 Results

Note, KB in problems(where mentioned) is mentioned not in CNF, despite it was converted to CNF before run if required. Also, KBs are written with normal notation, not the one that will be accepted by the code. Anyway, this document is to be read by humans, not computers. Notation, accepted by the code is in the code file.

4.1 Modus Ponens

- **Model checking.** Just run the algorithm. Done in negligible time, only 4 models to check. Statement is proved.
- **Walksat.** Run twice for $KB = [P, P \rightarrow Q, \neg Q]$ and for $KB = [P, P \rightarrow Q, \neg Q]$. The achieved result(as expected) first one does not find a satisfying assignment, the second run finds one. It worth mentioning that it did 999 flips before concluding that formula is unsatisfiable which is an overkill for a model with 2 variables. Therefore, we conclude that *with high probability* Q is entailed by $KB = [P, P \rightarrow Q]$.

Runtime: negilible for solving satisfiable formula and 1s for unsatisfiable one.

- **Resolution.** Do resolution for $KB = [P, P \to Q, \neg Q]$. We do only one here, as we expect this to be false.

4.2 Wumpus World (Simple)

- **Model checking.** Done in 0.1s, 128 models to check. Result is $\neg P_{12}$.
- **Walksat.** The situation is very similar to the previous case: both problem definition and solution method and runs time.
- **Resolution.** We assume that we do not have any "expected" value for $P_{1,2}$, so we use two-way resolution. Done in negligible time. Number of clauses at the end is 15-16.

4.3 Horn Clauses

Problem formalization is the following: $KB = [Mythical \to Immortal,\ \neg Mythical \to \neg Immortal,\ (Immortal \vee Mammal) \to Horned,\ Horned \to Magical]$. Questions: $Immortal, Mythical, Horned$.

- **Model checking.** "We can not prove anything about α" in this case means that $KB \not\models \alpha$ and $KB \not\models \neg\alpha$. Therefore, we run twice model checking for α and $\neg\alpha$ for every alpha in questions. If results of both runs are negative then we can conclude that we can not prove anything about α.

  Done in negligible time, 32 models to check.
- **Walksat.** Formulation of this problem for walksat: for each α in question it is enough to show that $KB + \alpha$ and $KB + \neg\alpha$ are satisfiable.

  Results: done in $< 0.2s$ and < 36 flips in each case, but this is a matter of chance.
- **Resolution.** We can not solve this problem with resolution as we will not encounter a contradiction for any of the assumptions. (We can say this, after using model checking we proved that we can not prove anything about the formulas in question)

4.4 Liars and Truth-tellers

In this problem, we work under the assumption that everybody is definitely a truth-teller or a liar and this is provable.

Define a problem in the following way: let Amy, Bob and Cal be variables. If the variable has $True$ value then we say that person is a truth-teller, otherwise, the person is a liar.

With this definition of variables, we can now notice that "tells" translates into equivalence relation. So if Person is a truth-teller, so corresponding variable's value is T then the statement must also have value T and same for Fs.

From previous assumptions we can now write given knowledge in a formal logic way:

(a) $KB = [Amy \leftrightarrow (Cal \wedge Amy), Bob \leftrightarrow \neg Cal, Cal \leftrightarrow (Bob \vee \neg Amy)]$.

(b) We treat phrase "Bob is correct" as "What Bob says is truth" which is equivalent to what Bob says.
$KB = [Amy \leftrightarrow \neg Cal,\ Bob \leftrightarrow (Amy \wedge Cal),\ Cal \leftrightarrow (Amy \wedge Cal)]$.

Questions: Amy, Bob, Cal.

- **Model checking.** Run model checking for KB and Amy, KB and Bob, KB and Cal. If the result of a run is positive then we can conclude that person is a truth teller. Otherwise, we conclude that person is a liar, as the person must have one of these two roles.

  Done in negligible time, only 8 models to check.
- **Walksat.** As we work under assumptions stated above it is enough to run walksat for $KB + \alpha$ for each α in question and return what walksat returns.

  This approach is not the most effective one as if it happens that $\alpha = False$ then walksat will work until it performs a maximal number of flips, which is much more then needed to solve the satisfiable case: 999 compared to ¡2(often). And therefore we have runtime either 0s or 0.8s.

- **Resolution.** We assume that we do not have any "expected" value for each formula in question, so we use two-way resolution. Done in 0s - 0.02s and number of clauses in the "latest" list for resolution is 20-40.

4.5 More Liars and Truth-tellers

Same assumptions as in previous problem.

$KB = [Amy \leftrightarrow (Hal \wedge Ida),\ Bob \leftrightarrow (Amy \wedge Lee),\ Cal \leftrightarrow (Bob \wedge Gil),\ Dee \leftrightarrow (Eli \wedge Lee),\ Eli \leftrightarrow (Cal \wedge Hal),\ Fay \leftrightarrow (Dee \wedge Ida),\ Gil \leftrightarrow (\neg Eli \wedge \neg Jay),\ Hal \leftrightarrow (\neg Fay \wedge \neg Kay),\ Ida \leftrightarrow (\neg Gil \wedge \neg Kay),\ Jay \leftrightarrow (\neg Amy \wedge \neg Cal),\ Kay \leftrightarrow (\neg Dee \wedge \neg Fay),\ Lee \leftrightarrow (\neg Bob \wedge \neg Jay)]$.

Questions: $Amy, Bob, Cal, Dee, Eli, Fay, Gil, Hal, Ida, Jay, Kay, Lee$.

- **Model checking.** We run entailment algorithm for KB and every question. Note that no previous knowledge is reused, e.g. when we proved something about Amy we do not assign a value to it and then update KB, despite that could have been done. One of the reasons why this is not done is that this is not captured with definitions of entailment function in AIMA book.

  Elapsed time is 0.12s-0.14s if what we trying to prove is not in fact entailed and 1.3s if it is. Number of models: 4096.

- **Walksat.** As we work under assumptions stated above it is enough to run walksat for $KB + \alpha$ for each α in question and return what walksat returns.

  This approach is not the most effective one as if it happens that $\alpha = False$ then walksat will work until it performs a maximal number of flips, which is much more then needed to solve the satisfiable case: 999 compared to ¡50(often). And therefore we have runtime either around 0.3s or around 5s. We see, that despite this problem is very similar to the previous one, but the number of variables has its effect.

  Also, we do not reuse knowledge gained in previous runs(for previous questions).

- **Resolution.** I did not solve this problem with resolution as it takes too much time as a number of clauses is HUGE, and therefore computation time will be too big for one problem of a CSC 442 assignment.

4.6 Doors of Enlightenment

Let's define variables in a similar way to the previous problems: A, B, C, D, E, F, G, H are defined exactly as variables before; X, Y, Z in the way that if X has truth value it means that door X is good, etc. Also, assume that "Either x or y" is exclusive or (despite it is not always like that in real life).

(a) $KB = [A \leftrightarrow X,\ B \leftrightarrow (Y \vee Z),\ C \leftrightarrow (A \wedge B),\ D \leftrightarrow (X \wedge Y),\ E \leftrightarrow (X \wedge Z),\ F \leftrightarrow ((D \wedge \neg E) \vee (\neg D \wedge E)),\ G \leftrightarrow (C \rightarrow F),\ H \leftrightarrow ((G \wedge H) \rightarrow A)]$

(b) We had "incomplete facts," C said "A and ...are both knights" which means "A and B or C or D or E or F are both knights".

G said "If C is a knight,..." must be completely changed. Basically, this would look like $G \leftrightarrow (C \rightarrow ...)$ in logic. If G has false value then $C \rightarrow ...$ must also have a false value which is possible only when C has true value. If G has true value then $C \rightarrow ...$ must also have a true value and this does not tells us anything about C. Therefore the last fact is $\neg G \rightarrow C$.

$KB = [A \leftrightarrow X,\ H \leftrightarrow ((G \wedge H) \rightarrow A),\ C \leftrightarrow (A \wedge (B \vee C \vee D \vee E \vee F \vee G \vee H)),\ \neg G \rightarrow C]$.

Here we do not assume that each door is definitely good or bad. We assume that possible there may be different models for this problems, so theoretically each door can have each value. (Contradiction to this statement for some door is what we want to prove)

Questions are X, Y, Z.

- **Model checking.** To find out if there is a "good" door we just have to see if any of the X, Y, Z are entailed by KB.

  Model checking finds that X is good in the first case in 0.5s and reject to others within 0.25s. Models checked: 2048.

For (b) part we have only 512 or 1024 models, as only 1 or 2 out of 3 doors are mentioned in KB. Door X is proved to be good in 0.1s (512 models) and two others rejected within 0.15s (1024 models).

- **Walksat.** As from the nature of the problem we try walksat on $KB + \alpha$ and on $KB + \neg\alpha$. The goal is to have one satisfied and other - not. If the goal is achieved we can conclude that α seems to be good; we're not completely sure as unsatisfiability was proved by the algorithm with a limited number of flips and it's probabilistic.

  Runtime to find satisfiability of a satisfiable statement is $< 0.25s$ and to prove unsatisfiability is $4s$. As in this problem, the answer is door (X) only so it takes $> 4s$ to prove that and two other doors are rejected in less than $0.3s$ each as both statements we try to prove are satisfiable.

  For part b results are similar, except that runtime is smaller as we have smaller KB and fewer variables.

- **Resolution.** I did not solve this problem with resolution as it takes too much time as a number of clauses is HUGE, and therefore computation time will be too big for one problem of a CSC 442 assignment.

4.7 Summary

Different methods have different performance, I would say that walk sat is quite a practical one it allows to solve more problems (compared to resolution) in a reasonable time(compared to a theoretical run time of model checking for models with a big number of variables).

Some results are not "the best", which is because of formulas representation and all processing required, but that's a price for readability of the inputs and this point is not critical as assignment does not contain any really huge and hard models.

5 Instructions

Run "python aosipa.py".

This will run examples using all three implemented methods in the following order: entailment, walksat, resolution. For each run of the function for inference, there will be printed statistics such as runtime and other parameters relevant to each function.