Formal Methods for High-Assurance Software Engineering

HomeWork Assignment 03

Shahin Roozkhosh
shahin@bu.edu
U01670169

September 2020

Problem 1. Provide a detailed comparison of the tableaux method and the resolution method.

Solution. Even though this question asks only for the second exercise on page 19, I did both the second one and third one out of curiosity and enjoyment.
As the hint suggests, let $X = \{x_1, \cdots x_k\}$ be a set of $k \geq 1$ variables. We define 2^k distinct wff's $\phi_1, \cdots, \phi_{2^k}$ where each ϕ_j is a disjunction containing x_i or $\neg x_i$ for every $1 \leq i \leq k$.
The conjunction ψ of these wff's, i.e., $\psi = \bigwedge\{\phi_1, \cdots \phi_{2^k}\}$ is not satisfiable. Clearly, ψ is a *CNF* and The conjunction ψ of these wff's, i.e., $\psi = \bigwedge\{\phi_1, \cdots \phi_{2^k}\}$ is not satisfiable. Here's why it's not satisfiable:

From Propositional logic laws we know transferring the non-CNF formula

$$(x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \cdots \vee (x_n \wedge y_n)$$

into CNF, produces a CNF with 2^n clauses:

$$\begin{aligned}
(x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \cdots \vee (x_k \wedge y_k) \equiv \\
(x_1 \vee x_2 \vee \cdots \vee x_k) \wedge \\
(y_1 \vee x_2 \vee \cdots \vee x_k) \wedge \\
(x_1 \vee y_2 \vee \cdots \vee x_k) \wedge \\
(y_1 \vee y_2 \vee \cdots \vee x_k) \wedge \\
\vdots \\
(y_1 \vee y_2 \vee \cdots \vee y_k)
\end{aligned} \tag{1}$$

Now in the formula above let $x_i \equiv \neg y_i$, and let line j in equation (2) as ϕ_j. the original formula translates to:

$$(x_1 \wedge \neg x_1) \vee (x_2 \wedge \neg x_2) \vee \cdots \vee (x_n \wedge \neg x_n)$$

which is obviously unsatisfiable; so as the equivalent CNF form of it, i.e, $\psi = \bigwedge\{\phi_1, \cdots \phi_{2^k}\}$ where each ϕ_j is a disjunction containing x_i or $\neg x_i$ for every $1 \leq i \leq k$.

Back to our original problem, here is our strategy: First, we show that a closed tableau for ψ contains at least $k!$ **distinct** paths. Then we examine the proof length for proving the unsatisfiability of the same wff, $\psi = \bigwedge\{\phi_1, \cdots \phi_{2^k}\}$ using the *resolution* method and conclude that the proof length using the *tableaux* is strictly higher that the proof length using *resolution* method. Especially for the large numbers of k, this difference exacerbates.

Analysis of the proof length of $(\psi \vdash_{tab} \perp)$ using *tableaux* method:
To show unsatisfiability of a wff using the *tableaux* method, we need to construct a closed tableau. We know that a tableau T is a closed tableau if all its paths are closed. Where, by the definition a path from the root to a leaf in tableau T is a closed path if it includes both a wff x_i and its negation $\neg x_i$. The intuition here is on the number of operators. First let's look at the tableaux expansion rules:

EXPANSION RULES

$$\frac{\varphi \wedge \psi}{\begin{array}{l}\varphi \\ \psi\end{array}}$$

$$\frac{\neg(\varphi \wedge \psi)}{\neg\varphi \mid \neg\psi}$$

$$\frac{\varphi \vee \psi}{\varphi \mid \psi}$$

$$\frac{\neg(\varphi \vee \psi)}{\begin{array}{l}\neg\varphi \\ \neg\psi\end{array}}$$

$$\frac{\varphi \rightarrow \psi}{\neg\varphi \mid \psi}$$

$$\frac{\neg(\varphi \rightarrow \psi)}{\begin{array}{l}\varphi \\ \neg\psi\end{array}}$$

$$\frac{\neg\neg\varphi}{\varphi}$$

There are two important points here: First, applying each rule only omits at most one single operand in $\{\wedge, \vee, \rightarrow\}$ (except $\neg$). Second, none of the expansion rules transform any operand in $\{\wedge, \vee, \rightarrow\}$ to another operand in $\{\wedge, \vee, \rightarrow\}$. With this simple observation now let's reconsider the shape of our formula ψ:

$$\begin{aligned} \psi \equiv \\ (x_1 \vee x_2 \vee \cdots \vee x_k)\wedge \\ (y_1 \vee x_2 \vee \cdots \vee x_k)\wedge \\ (x_1 \vee y_2 \vee \cdots \vee x_k)\wedge \\ (y_1 \vee y_2 \vee \cdots \vee x_k)\wedge \\ \vdots \\ (y_1 \vee y_2 \vee \cdots \vee y_k) \end{aligned} \tag{2}$$

We know we have to apply expansion rules in a way that at the end of the day, we can reach to leafs of the form p and $\neg p$ with no operands in $\{\wedge, \vee, \rightarrow\}$. Even in the most efficient way of using tableaux expansions i.e. “expand conjunctions as much as possible before disjunctions”, starting from and ϕ_i in the formula, there are at $k-1$ $\vee$ operands at the beginning. Each application of expansion rules in the best case, helps us to get rid of **only one** $\vee$ **operand** yet resulting in a branch. So the intuition is in order to get rid of all $\vee$ operands in each ϕ_i, would lead to at least k branches). At each level in the resulting sub-trees by applying “expand conjunctions as much as possible before disjunctions” optimization, we still have wff’s with $k-2$ $\vee$s. Hence the whole argument repeats this time resulting in $k-1$ branches and so on. Hence there will be at the very least $k!$ branches using tableaux.

Analysis of the proof length of $(\psi \vdash_{res} \perp)$ using *resolution method* method:

In order to use resolution, the good news is that ψ is already in a CNF form. Luck! Next, we write down ψ as a set of clauses, the initial knowledge base: $\{\phi_1, \phi_2, ..., \phi_{2^n}\}$. By Put down every clause in the knowledge base first(2^n steps here), then apply resolution repeatedly. Here the most intuitive algorithm (not necessarily optimal) is to match ϕ_i two by two and apply the only resolution rule on them. This is possible because we math pairs in a way that there is at-least one atom p in one of them, while $\neg p$ is in the other one. Following this method at each level we get rid of half of wff’s and since at the beginning there are 2^k wff’s $\{\phi_1, \cdots \phi_{2^k}\}$, at each level we apply the resolution rule 2^{j-1} starting from $j = k$ in the first level. Hence there will be at-least $2^{1+2+3+...+k-1}$ = $2^{k(k-1)/2}$ branches.

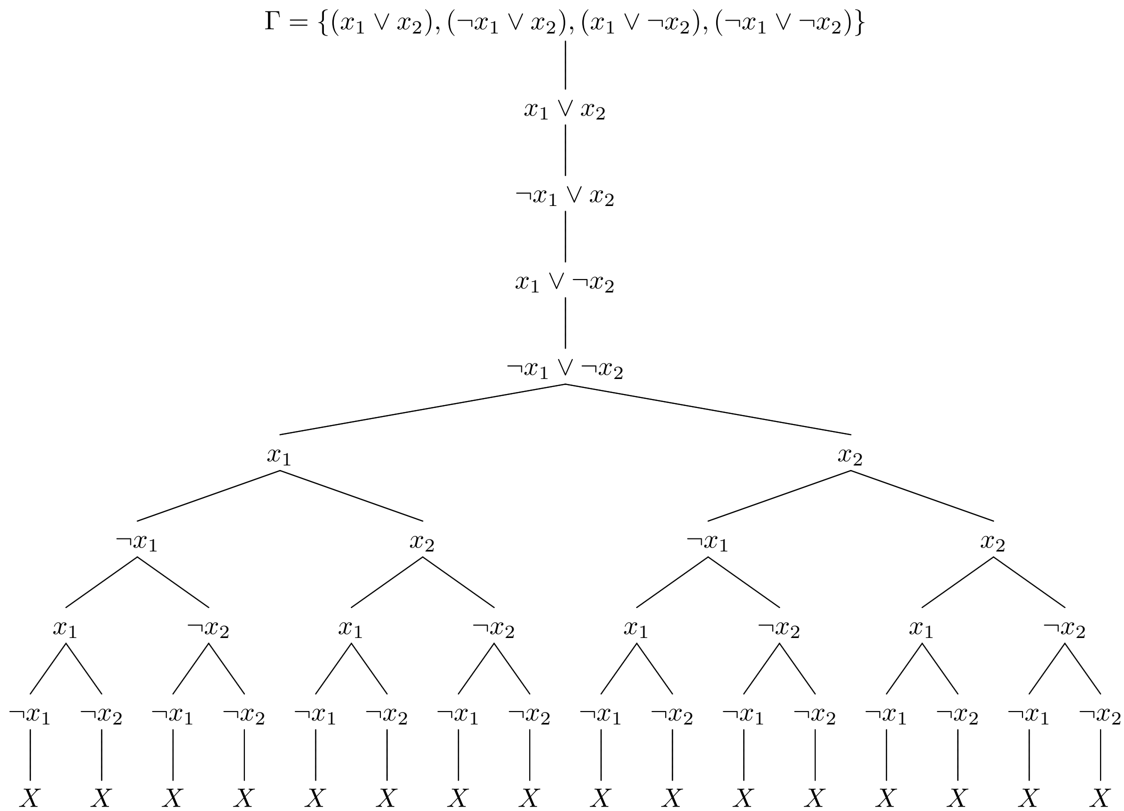
Discussion: Now the argument is straightforward. The algorithm we proposed for *resolution method* always beat the best-case scenario for the *tableaux method*. Because $k! >>> k^2 2^k$ for large ks Reference: Slide 08 page 19 by Prof.Kfoury.

Minimal example:

Letting $k = 4$

tableau Method:

$\psi = (x_1 \vee x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2)$



resolution Method:

Knowledge Base: $\Gamma = \{(x_1 \vee x_2), (\neg x_1 \vee x_2), (x_1 \vee \neg x_2), (\neg x_1 \vee \neg x_2)$

1	$x_1 \vee x_2$	
2	$\neg x_1 \vee x_2$	
3	$x_1 \vee \neg x_2$	
4	$\neg x_1 \vee \neg x_2$	
5	x_2	resolve 1,2
6	$\neg x_2$	resolve 3,4
7	$\perp$	resolve 5,6

Problem 2. Fact: satisfiability of ϕ can be determined by first checking if $ROBDD(\phi)$ is equal to the $ROBDD$ with a single terminal label "0", in which case ϕ is unsatisfiable, otherwise ....
(a) Fill in the missing part in preceding statement. (b) Determine if ϕ is satisfiable and construct a satisfying assignment. (c) Determine if ϕ is satisfiable and count the number of satisfying assignments.

Solution.

(a):
Otherwise there would have exists another terminal, labeled "1" where if we consider $ROBDD(\phi)$ as a DAG, there exists at least one path from the root to this "1" terminal.

(b):
Determining satisfiability is answered in (a).
Continuing argument in (a): ϕ is satisfiable if and only if there exists a terminal, labeled "1" where if we consider $ROBDD(\phi)$ as a DAG, there exists at least one path from the root to this "1". In this case in the corresponding path(s) we can form a satisfying assignment $\{A_{p_1}, A_{p_2}, \cdots, A_{p_n}\}$ where n is the number of atoms in ϕ. Here's how A_{p_i} is defined: By traversing from root to the terminal "1", **w.r.t a fixed order of atoms**, if the edge corresponding to that atom p_i is a dashed edge, let $A_{p_i} = \neg p$, otherwise, if it's a solid edge, $A_{p_i} = p$.

(c):
Determining satisfiability is answered in (a).
Following argument is relative to a fixed ordering of the variables in $ROBDD(\phi)$ which by $ROBDD(\phi)$ iss uniquely defined DAG.

The number of satisfying assignments is **exactly** equal to the number of **distinct** paths from root to the terminal "1" in the $ROBDD(\phi)DAG$. To be more specific, this is how we can count these assignments: Let $ROBDD(\phi)$ be a directed acyclic graph named G. We form DAG G' where all directed edges are simple reversed. this is doable in $O(|\phi|)$. Now, satisfying assignments corresponds to paths from terminal "1" to the root. Without loss of generality, we can assume imposing order on variables from root to terminal "1" is $p_1, p_2, \cdots, p_n$ where p_i is an atom. Then in the G', let the number of edges from the terminal to *some* subset of nodes, namely S_n in the level of p_n is e_n. Similarly we can define

e_i as the number of edges from the subset of vertices S_i (that are reachable from S_{i+1}) to e_i. Then the total number of satisfying assignments will be:

$$\prod_{i=1}^{n} e_i$$

Problem 5.

Solution. `https://github.com/ro0zkhosh/CS511/blob/master/HW3/scheduling.py`

Problem 6.

Solution. `https://github.com/ro0zkhosh/CS511/blob/master/HW3/clique.py`