

Computational Modeling of Modal Formulas

Joseph Thaidigsman

May 4, 2018

1 Introduction

For this paper, I chose to write a program that decides the satisfiability of any given modal formula, and if satisfiable, visualizes some number of example models that satisfy it. I imagined that formulating algorithms to accomplish this task would be a process rich with interesting insights into the structure of modal logic, and fortunately, this proves to be the case.

Thus, this paper begins with an overview of the development process, justifying the steps that lead into the formulation of the algorithm and proving its correctness. After doing so, I will detail the extensions to the program that I have made so as to make it more useful on a practical level, for modeling actual arguments, briefly highlighting some of its potential applications.

The source code of the software I developed, along with instructions for how to download and use it, is [hosted on GitHub](#) for reference and further use.

2 A Brief Overview of Necessary Concepts

Before getting into the details of the development, it's pertinent to review a few key features of Modal Logic that allow a program such as this one to evaluate satisfiability of formulas. To begin, consider the following two theorems:

Theorem 1 (Benthem in [3]). *Basic modal logic has the finite modal property: every satisfiable modal formula has a finite model.*

Theorem 2 (Benthem in [3]). *Modal Logic has the effective finite model property.*

In proving the latter of the two through the use of the former, Benthem remarks that this is sufficient to decide SAT for modal formulas: by unraveling a model in

which φ holds at the root via bisimulation, there need be only a finite depth of expansion to decide satisfiability for φ in the model — namely, the modal depth of φ . He provides an upper bound of $\text{len}(\varphi)^{md(\varphi)+1}$ to the size of the unraveled tree, and thus, through enumeration of all possible trees of that length, there exists an algorithm to determine the satisfiability of any modal formula.

Remark 3. This enumeration approach is intractable for any large input and thus unsuitable for practical usage.

Consider some modal formula φ of modal depth $md(\varphi) = 10$, length $\text{len}(\varphi) = 30$. From the bound given by Benthem, we can compute the upper bound of the size of this model, coming from a formula of reasonable size: $30^{11} \approx 1.7 \cdot 10^{16}$. A modern processor, running at 4 GhZ (or four billion operations per second) would therefore have to spend at least 4.4 million seconds per tree being considered!¹

Thus, it proves necessary to take a more precise approach than simple enumeration-and-search. Let us now turn our attention to the construction of a *Semantic Tableaux*, commonly used in modern automated deduction[3]. Though the algorithmic construction of a Tableaux is not exactly what is implemented within the program, the concepts within provide the backbone of the structure of my software. Constructing a tableaux for a formula φ (the process of which I hereby detail informally for brevity's sake) involves recursively moving from formula to subformula, syntactically breaking down formulas into smaller cases, seeking to create a model in which φ holds true at the root, until the tableaux becomes *closed*, proving its validity as no counter-example is possible, or necessarily remains *open*.

Definition 4 (D'Agostino in [2]). A *closed* branch of a tableaux is one in which an impossibility condition (e.g. a contradiction like $p \wedge \neg p$) appears. A *closed tableaux* is therefore a tableaux in which all branches are closed. In contrast, an *open* branch is one in which it is possible to provide a satisfying, non-contradictory assignment.

A formal description of the process of tableaux construction is out of scope for this paper, but the key concept of recursively breaking down a formula by way of its syntactic structure until a satisfying assignment or contradiction is found is key to my program's algorithmic process.

¹In fact, this assumes only one operation would be performed per world, which is impossible. Thus, the real time would be much greater than this.

3 Formulating the Software

In constructing this program, I sought to develop a tool for quickly determining the satisfiability of a modal formula, and if satisfiable, to visualize some of its satisfying models in a useful way. For a satisfiable formula, bisimulations and filtration can provide a more clear bound on which models differ in a meaningful semantic way, but attempting to fine-tune an algorithm to select such models that remain invariant under bisimulation proves to be an exponentially more difficult problem than merely deeming a formula satisfiable or not.

Thus, while it remains necessary to determine the satisfiability of a given formula correctly in all cases, we can relax our constraints for the models we choose to visualize, turning instead to a more intuitive sense of usefulness. To do so quickly, without having to struggle with high-constant exponential bounds for the running time of my program, I sought to emulate something similar to the graph approach to iterative satisfiability-checking detailed by Paulo and Sheila Veloso[4][5] and by Marcin Cuber[1].

As a quick overview for accuracy's sake, before getting to the conceptually interesting parts of the algorithm: my program begins by parsing through a given modal formula, reconstructing it in prefix form (e.g. $(p \wedge q)$ is expressed instead as (\wedge, p, q)).² After recursively rewriting the formula in a more strict prefix form, it becomes far easier to analyze and perform a syntactic decomposition, as the scope of each operator/connector is easily determined by observing the first item in a tuple.

After parsing formula φ , I construct a *modal graph*, a representation of a possible model that may satisfy it, and initialize it with a single world. Within this world, I store a list of unprocessed formulas to handle, which initially is comprised solely by φ . In determining satisfiability of the formula, I defined four different classifications of subformula actions in accordance to their structure, each of which I treat differently³. These four are named as follows:

- **In-place Decomposition (IPD):** An in-place decomposition is an action that simplifies a subformula by breaking an uncompleted subformula within a world into two smaller subformulas within that world, each of which must be satisfiable within it for the world to remain open.

²This parser is based on one written by Marcin Cuber's open-source modal logic syntax analyzer, which can be found [here](#), though I altered it significantly for efficiency reasons.

³This is similar to how Cuber classifies his formulas[1], though his implementation does not assign multiple actions to certain forms for a more useful visualization

- **Graph-duplication Decomposition (GDD):** A graph-duplication decomposition is an action that simplifies a subformula by creating a new, identical copy of the model, differing only in the world in which the GDD formula triggered the duplication. In the initial copy of the graph, only one piece of that subformula remains, and in the new copy, the other piece lies.
- **World-forming Decomposition (WFD):** A world-forming decomposition is one in which a subformula from w is simplified by forming a new world w' within the same modal model, adding an edge s.t. wRw' , and passing the WFD subformula from the initial world without its outermost modality to the new world's unprocessed formulas.
- **Enforcement Decomposition (EFD):** An enforcement decomposition is a decomposition that simplifies a subformula by creating a *child rule*. If a world $w \in W$ has a child rule \mathcal{I} , then $\forall w' \in W : wRw'$, the invariant asserts that $M, w' \models \mathcal{I}$.

I will hereby associate ten different syntactic forms of subformulas⁴ with their classifications and the actions that the program takes when faced with each:

IPD	GDD	WFD	EFD
$(\varphi \wedge \psi)$	$(\varphi \vee \psi)$	$\neg \Box \varphi$	$\Box \varphi$
$\neg(\varphi \rightarrow \psi)$	$\neg(\varphi \wedge \psi)$	$\Diamond \varphi$	$\neg \Diamond \phi$
$\neg(\varphi \vee \psi)$	$(\varphi \rightarrow \psi)$		

Each of those forms are decomposed into simpler parts as follows, performing their corresponding action as specified on the prior page during their decomposition:

IPD	IPD'	IPD''	GDD	GDD'	GDD''
$(\varphi \wedge \psi)$	φ	ψ	$(\varphi \vee \psi)$	φ	ψ
$\neg(\varphi \rightarrow \psi)$	φ	$\neg \psi$	$\neg(\varphi \wedge \psi)$	$\neg \varphi$	$\neg \psi$
$\neg(\varphi \vee \psi)$	$\neg \varphi$	$\neg \psi$	$(\varphi \rightarrow \psi)$	$\neg \varphi$	ψ

WFD	WFD'	EFD	EFD'
$\neg \Box \varphi$	$\neg \varphi$	$\neg \Diamond \varphi$	$\neg \varphi$
$\Diamond \varphi$	φ	$\Box \varphi$	φ

⁴Through reductions, this could be a smaller list, but the motivation for additional explicit cases becomes more clear in observing which forms generally call for additional considerations for useful visualizations, a distinction lost upon further reductions.

Additionally, any double-negatives are simplified throughout. Through iterative repetition of this algorithm on all worlds in all models generated, eventually all worlds in all graphs will track only literals or their negations. When a literal or negated literal is processed as a subformula, it is added to a list of necessarily true atomic propositions or necessarily false ones, respectively, and removed from the list of uncompleted tracked subformulas.

The concepts gleaned from the previous digression into semantic tableaux analysis are precisely relevant here, as the program checks with each new value assignment of a literal whether any contradictions within a world are formed. If they are, then the graph is discarded from consideration. If all graphs close, then the program concludes that the input formula is unsatisfiable, analogous to the closing of all branches within a tableaux. Note that this is treated in the opposite manner as it would be in a semantic tableaux: the semantic tableaux seeks to close all branches that serve as counter-examples to a formula to prove its validity, whereas this algorithm seeks non-contradictory models to visualize satisfying models. If a modal graph finishes processing all tracked subformulas in all worlds, assigning all variables without conflict, then it represents a satisfying model for the input formula, and it can conclude that the formula is satisfiable. By using this algorithm to search for satisfying models of the negation of a formula, if it returns that it is unsatisfiable, this proves the formula is valid in much the same way as a semantic tableaux does.

I will now show the correctness of this algorithm.

Remark 5. There exist only four semantically different formulas of the eleven classified syntactic forms when accounting for recursive decomposition of negations.

Observe that by distributing through the negations that occur within each syntactic form within each classification, we can reduce each into each other. Let us pick $\varphi \wedge \psi$ arbitrarily for IPD types and show the reduction of the others to it: $\neg(\varphi \rightarrow \psi)$, by propositional logic using DeMorgan's laws, is equivalent to $(\varphi \wedge \neg\psi)$. Note that $\neg\psi$ is also a well-formed formula that ψ could represent, and thus this form is equivalent to the form $(\varphi \wedge \psi)$. Also, $\neg(\varphi \vee \psi)$ is equivalent by propositional logic to $(\neg\varphi \wedge \neg\psi)$ and it too is therefore of the form $(\varphi \wedge \psi)$.

The same holds for GDD reductions: $\neg(\varphi \wedge \psi)$ is equivalent to $(\neg\varphi \vee \neg\psi)$, which is of the form $(\phi \vee \psi)$, and thus is semantically equivalent. $(\varphi \rightarrow \psi)$ is equivalent to $(\neg\varphi \vee \psi)$, which is again of the form $(\varphi \vee \psi)$ and semantically equivalent.

For WFD, using DUAL, we can express $\neg\Box\varphi$ as $\Diamond\neg\varphi$, in which φ may very well stand for $\neg\varphi$, and thus it is semantically equivalent to $\Diamond\varphi$; for EFD, the same holds for translating $\neg\Diamond$ into $\Box\neg$.

Theorem 6. *The Computational Modeling Algorithm hitherto described will always find a satisfying model for a formula ϕ if ϕ is satisfiable, else it will determine it is unsatisfiable.*

Proof: Here, we induct on $\text{len}(\phi)$, the length of the formula, defining length as the count of literals and their negations that appear within the formula.

Base Case: $\text{len}(\phi) == 1$. In this case, the algorithm will assign to the starting world a value for the one literal specified within ϕ , setting it to true if it appears as a positive literal within ϕ , else letting it be false if its negation appears. If the opposite assignment had already been set within the world, it will correctly return that the formula is unsatisfiable.

Inductive Hypothesis: Assume all formulas of $\text{len} < k$ have their satisfiability correctly determined by the program.

Inductive Step: Show that a formula ϕ with $\text{len}(\phi) = k$ has its satisfiability correctly determined.

Here, there exist four cases, the four actions that the algorithm can take based on the syntactic form of ϕ .

IPD As shown in the prior remark, as a corollary stemming from basic propositional logic rules, all forms within this category are semantically reducable to $(\varphi \wedge \psi)$. Thus, consider the action taken for a ϕ of that form, reducing it without changing semantic meaning or satisfiability if it is not of that form. The algorithm will break this formula into two subformulas, φ and ψ , which it will then recursively decompose. Since ϕ has length k , and conjunction syntax ensures that φ and ψ are non-zero, these are now two subformulas of $\text{len} < k$. By the inductive hypothesis, when the formula processes these, their satisfiability will be determined correctly, and the algorithm will return that the formula is satisfiable iff both subformulas are satisfiable, else it will return that the formula is unsatisfiable, which is correct.

GDD As before, we can consider only the case $(\varphi \vee \psi)$. Here, the algorithm will create a duplicate graph, altering the subformulas tracked in the original world to consider only φ and having the other graph only consider ψ at that world. It will then return that the formula is satisfiable iff at least one of the two is satisfiable, which matches the truth functionality of \vee , and since each subformula has $\text{len} < k$, this will return the correct value

WFD Again as before, we need only consider the $\Diamond\varphi$ case. Here, a new world w' is

created, and the formula is removed from the current world w , instead determining if φ is satisfiable after adding wRw' . Since $M, w \models \Diamond\varphi$ iff $\exists w'$ where $wRw' \wedge M, w' \models \varphi$, if the algorithm returns that the subformula in w' is satisfiable, then it will correctly return that w is satisfiable. Though φ may have the same length as ϕ , there may only be a finite number of modalities in any input, and eventually there will be either a literal, which the base case proves is handled correctly, or there will be a term in GDD or IPD form, which will reduce the size of each subformula and return correctly as proven. Thus, WFD forms have their satisfiability correctly determined.

EFD Finally, we consider ϕ being handled at w of form $\Box\varphi$. In this case, the algorithm enforces that $\forall w' \in W : wRw', M, w' \models \varphi$. By the same reasoning as the last case, these satisfiabilities will be determined correctly, and by the semantic truth definition of \Box , this will correctly return the proper satisfiability determination for ϕ .

Thus, all cases have been covered and the algorithm has been determined to function correctly for all inputs ϕ of all lengths. This concludes the proof.

As a final note before moving on, in addition to the core algorithm outlined so far, I added numerous different actions to some syntactic forms in order to improve the usefulness of the visualization. Though I won't detail all of them, here are two examples that highlight an interesting divide between being simply “technically correct” and being actually useful:

From the defined functionality, whenever a formula of the form $\Diamond\varphi$ is handled, a new world is created. While this leads to correct results, this often doesn't result in “realistic” models that occur from modeling a real-life scenario. For instance, the formula $\Diamond p \wedge \Diamond q$ would never result in a graph with two worlds, one world with $p \wedge q$ true, and the other that has an edge to it to make the formula true — it would only ever display a model with three worlds, with p and q being true in different worlds. While this isn't incorrect, it limits the usefulness of the visualizer. Thus, upon evaluating such a formula, if the world already has a child or more, it will copy the graph and try to make the subformula true at one of its already-existing children, and if that proves satisfiable, then it will also display that model along with the one that would be generated from the original specification.

Another brief example is a modification to evaluating $\varphi \rightarrow \psi$: along with checking one graph with subformula $\neg\varphi$ and the other with ψ , I add another graph that checks the satisfiability of $\varphi \wedge \psi$. Regardless of its result, it won't affect correctness, but if

it is satisfiable, it will display the model alongside the others, and this has massively helped its usefulness in modeling statements — frequently people will assume that the antecedent is true in natural language when asserting an implication, and this allows the program to visualize that and better represent human language.

4 Extensions and Applications

After completing the core program, I added support for adding frame constraints when evaluating satisfiability: a user can specify any combination of reflexivity, transitivity, or symmetry to enforce that all generated models must have. This vastly expands the usefulness of the program, as it can now be used to test validity and satisfiability of statements in systems other than just K, with axiomatic extensions to the language being implicitly enforced through the edge properties of the modal graph. To see this in practice, observe the following three figures, which show how this software can be used to get a useful intuition for why certain axioms hold on frames with specific properties:

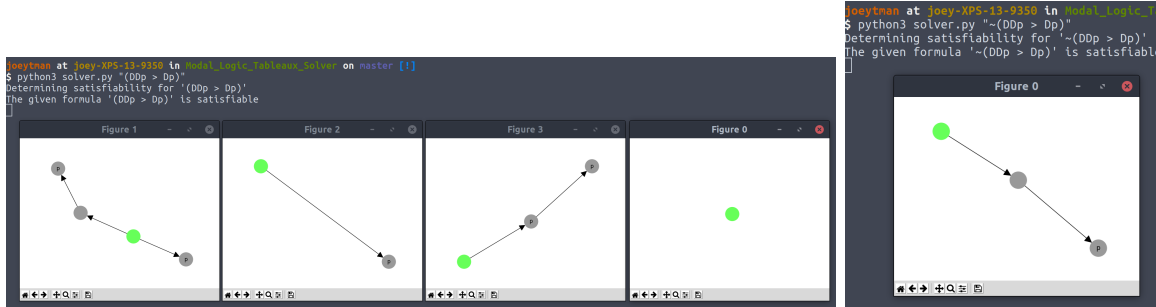


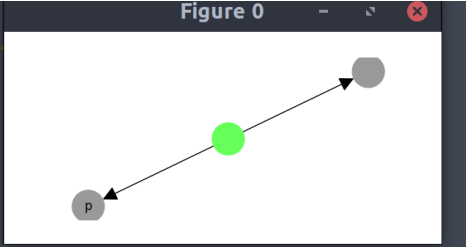
Figure 1: Visualizing a few ways in which both $(\Diamond\Diamond p \rightarrow \Diamond p)$ and $\neg(\Diamond\Diamond p \rightarrow \Diamond p)$ are satisfiable on unrestricted frames

```
joeytman at joey-XPS-13-9350 in Modal_Logic_Tableaux_Solver on master [!]  
$ python3 solver.py "~(DDp > Dp)" reflexive transitive  
Determining satisfiability for '~(DDp > Dp)' on frames that are: reflexive transitive  
The given formula '~(DDp > Dp)' is unsatisfiable
```

Figure 2: When enforcing frame restrictions that make R transitive and reflexive, $\neg(\Diamond\Diamond p \rightarrow \Diamond p)$ is unsatisfiable. This suffices to prove that $(\Diamond\Diamond p \rightarrow \Diamond p)$ on such frames.

As another example, if you were unknowledgeable of what constitutes the validity of the 5 axiom, you can keep adding constraints until its negation is unsatisfiable, like it is on euclidean (reflexive + transitive + symmetric) frames:

```
joeytman at joey-XPS-13-9350 in Modal_Logic_Tableaux
$ python3 solver.py "~(<p > []<p)"
Determining satisfiability for '~(<p > []<p)'
The given formula '~(<p > []<p)' is satisfiable
[]
```



```
joeytman at joey-XPS-13-9350 in Modal_Logic_Tableaux_Solver on master
$ python3 solver.py "~(<p > []<p)" reflexive symmetric transitive
Determining satisfiability for '~(<p > []<p)' on frames that are: reflexive symmetric transitive
The given formula '~(<p > []<p)' is unsatisfiable
```

5 Concluding Remarks and Reflection

Obviously, this is just a small sample of the possible uses of the program. It's been wonderfully interesting to use to visualize arguments made by politicians and political speakers⁵, and it was also fascinating to build as it required a deep engagement with the structure of the modal language and its permissiveness in having SAT decidable.

I plan to continue developing the program until I'm able to add the other frame properties I'd like to, and I'm considering adding support for non-normal modal logics (I would love to add the ability to visualize counterfactuals, especially, but this is a substantially more difficult problem).

As it stands, though, it was fantastic developing this program to reinforce the knowledge that I've gained from the class over the past few months (even if I had no clue what I was getting myself in to when I decided to develop this — a similar program was the Master's Thesis of one of the sources I cite within this paper [1], which made me realize that I vastly underestimated the difficulty of developing such a program). Even still, I have no regrets about tackling it, and I'm proud of having implemented the semester's knowledge in such a concrete way, and hopefully others online will find it useful — there's a depressing lack of existing open-source software that relates to modal logic, and that's a real tragedy.

⁵An addition to the paper I had wanted to include, but it would have taken far too many pages to do it justice to be able to fit within the scope of the essay. I'll be updating the main page of the GitHub repository with a writeup on that in the coming weeks, as I have it mostly finished, so if this paper was interesting to you, be sure to look back and check it out!

References

- [1] Marcin Cuber. “Propositional Modal Logic Using Tableaux Methodology For Theorem Proving”. MA thesis. University College, London, 2016.
- [2] Marcello D’Agostino et al. *Handbook of tableau methods*. Springer Science & Business Media, 2013.
- [3] Johan Van Benthem et al. *Modal logic for open minds*. Center for the Study of Language and Information Stanford, 2010.
- [4] Paulo AS Veloso and Sheila RM Veloso. “On Graph refutation for relational inclusions”. In: *arXiv preprint arXiv:1203.6159* (2012).
- [5] Paulo AS Veloso, Sheila RM Veloso, and Mario RF Benevides. “On a Graph Approach to Modal Logics”. In: *Electronic Notes in Theoretical Computer Science* 305 (2014), pp. 123–139.