Implementing Semantic Tableaux for Different Logics

Ayush Prajapati

Supervisor: Dr. Jonni T Virtema

A report submitted in fulfilment of the requirements for the degree of Msc. Advanced Computer Science *in the*

Department of Computer Science.

September 11, 2024

Declaration

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Name: Ayush Prajapati

Signature: Ayush

Date: 11/09/2024

Abstract

This project implements a Semantic Tableaux Solver for modal logic, focusing on the K modal system. It provides a web-based interface using Streamlit, allowing users to input propositional and modal logic formulas and check their validity and satisfiability. The solver uses the tableaux method to analyse formulas, generating visual representations of tableau trees and Kripke model accessibility relations. Key features include formula parsing, tableau expansion, and accessibility relation analysis. The project also includes unit tests to verify the correctness of the implementation. This tool serves as an educational resource for students and researchers in modal logic, offering insights into formula structure and semantic properties.

Acknowledgements

First and foremost, I'd like to express my heartfelt gratitude to *Dr. Jonni T Virtema*, my supervisor extraordinaire. Without his guidance, this project would have been as lost as a propositional variable in a sea of modal operators. Dr. Virtema's expertise in logic was truly □ (necessarily) invaluable, and his support was ◇ (possibly) the best a student could ask for.

To my family, you've been my constant accessibility relation, connecting me to all possible worlds of support and encouragement. Mom and Dad, your love and patience have been as universal as a tautology. To my siblings, thanks for keeping me grounded in the actual world while I explored countless possible ones.

A round of applause for my friends, who stuck with me even when I started speaking in modal logic symbols. Your friendship has been as reliable as the law of excluded middle. Special thanks to Vinroy, who listened to my logic ramblings with the dedication of a tableau waiting to be expanded.

I must also acknowledge the countless cups of coffee that fueled late-night coding sessions. They were the real MVPs, keeping my brain functions satisfiable when they were on the brink of becoming a contradiction.

A special shoutout goes to the inventors of the Semantic Tableaux method. Your brilliant minds have saved countless logic students from the depths of despair, turning intimidating formulas into somewhat less intimidating trees.

Lastly, I'd like to thank the modal logic community. Your passion for possible worlds has opened up a universe of possibilities, proving that reality is just one of many accessible states.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction to Logical Analysis

[5, 12] Logical analysis constitutes a fundamental step in applying formal reasoning, the application to which goes through a wide variety of fields such as computer science, artificial intelligence, and formal verification. Contrary to some philosophical assumptions, logical analysis is a method of evaluating the truth of propositions and their connections by means of formal methods. This project delves into two significant branches of logic: Propositional logic and Modal logic both are formal language.

Propositional Logic allows analysing a proposition that admits either the true or false value. It involves sentential logic operators like **negation (¬)**, **conjunction (⋀)**, **disjunction (⋁)** and **implication (→)** which connect primitive propositions to form compound logical statements. The advantage of Propositional Logic is to be able to get the basic logical analysis and truth table construction for evaluating the validity and satisfiability of formulas.

Modal Logic, an extension of Propositional Logic, introduces modalities to express necessity and possibility. This branch is crucial for reasoning about statements across different possible worlds or contexts. Modal Logic uses operators like **necessity (□)** and **possibility (◊)** to explore statements' truth values in varied scenarios, providing a richer framework for analysing dynamic or uncertain environments.

The goal of this project is to design a tool called Semantic Tableaux Solver which can be used to analyse logical formulas at Propositional and Modal Logics. Semantic Tableaux or Truth tree is used to determine the validity and satisfiability of the formulas under consideration by constructing the tableaux and analysing its structure. The provided method is the visual and systematic approach in logical analysis and thus can be used for demonstration in the educational process and practical work.

1.2 Aims and Objectives

The goal of this project is to build a solver that will enable the user to solve Propositional and also Modal Logic formulas through the use of the Semantic Tableaux Solver application. The specific objectives of the project are:

1. **Formula Parsing:** Create a stable structure of the parser which is used for the description of logical formula and its transformation into the form for further processing.
2. **Validity and Satisfiability Checking:** To come up with the validation and satisfiability of the formulas get in touch with the Semantic Tableaux approach and be able to classify the results in a clear manner.
3. **Visualisation:** Produce legendary representations of tableaux and accessibility graphs in order to improve the comprehension and analysis of logical structures.
4. **User Interface:** Develop a user-friendly interface in ***Streamlit*** where the user should be able to enter formulas, see the calculated & analysed results, and go through the history of the solutions.
5. **Detailed Analysis:** Offer proper evaluations of propositional as well as modal aspects of formulas and possible accounts and interpretations of logical characteristics.

1.3 Constraints and Considerations

Several constraints and considerations affect the project:

- **Complexity of Modal Logic:** Solving and visualising Modal Logic can be problematic because of such extra operators and their interconnections. These complexities can only be managed if there is proper attempt at designing and testing the solutions to meet required performance and safety measures in the system.
- **Performance:** It is also important to make sure that the tool will not drastically slow down in larger and complex formulas to make the application more receptive.
- **User Experience:** Therefore, making an interface that is easy to understand and comes with clear outcomes and visualisations for the audience is vital for usability.

1.4 Structure of the Dissertation

The dissertation is organised as follows:

- **Chapter 2: Literature Review**: This chapter reviews existing work related to Propositional and Modal Logic, the Semantic Tableaux method, and similar tools. It identifies gaps in current research and establishes the foundation for the project's development.
- **Chapter 3: Requirements and Analysis**: This chapter outlines the project's objectives in detail and breaks down the problem into manageable components. It includes the specifications for the tool and discusses the approach for testing and evaluation.

- **Chapter 4: Design**: This chapter presents the design of the Semantic Tableaux Solver, including the chosen techniques and justifications for the design decisions. It includes diagrams and explanations of the design choices.
- **Chapter 5: Implementation and Testing**: This chapter details the implementation process, including coding and novel aspects of algorithms. It also covers the testing strategies used to ensure the tool's functionality and accuracy.
- **Chapter 6: Results and Discussion**: This chapter presents the results of the project, evaluates how well the objectives were met, and discusses any unexpected findings or limitations.
- **Chapter 7: Conclusions**: This chapter summarises the project's outcomes, reflects on the achievements, and suggests areas for future work.

1.5 Relationship with Degree Programme

This project is closely aligned with my degree programme, which emphasises practical applications of formal methods and logic. The idea of Semantic Tableaux Solver is based on the combination of logical theory from the field of mathematics and computer science with practical programming experience. The project is beneficial in measure for the diffusion of academic concepts into practice solutions which allows tests of real problems and improves practical and logical skills.

Chapter 2

Literature Review

This chapter discusses the literature and tools used in formulating Propositional Logic, Modal Logic and the Semantic Tableaux method. This review focuses on the presentation of the theoretical frameworks and the revelation of the state-of-the-art in the automated logic solving systems as well as the perspectives of this work. The chapter is organised into three main sections: a review of logical foundations, an examination of existing tools and methods, and an analysis of the relevance and limitations of current solutions.

2.1 Propositional Logic: Foundations and Applications

Propositional Logic forms the basis of classical logic, where formulas are constructed using propositional variables and logical connectives. The primary literature in this area includes foundational texts which are previously made [9, 22]. These works introduce propositional logic from both syntactic and semantic perspective proposing the basics notions of truth-tables, logical equivalences, as well as the natural deduction and the resolution method.

- Propositional Logic has been applied in various fields namely in circuit design, automated reasoning, and formal verification. In such fields, it is an essential step to know if a given logical formula is satisfiable or not, which involves proposing several algorithms and tools. [2, 7] Some research work on the SAT problem, and the subsequent DPLL algorithm, laid the groundwork for modern SAT solvers that efficiently determine the satisfiability of propositional logic formulas.

- [3, 19] Propositional logic, also known as propositional calculus or sentential logic, is a fundamental area of study in formal logic, mathematics, computer science, and philosophy. It involves the manipulation and analysis of propositions—statements that can be either **true** or **false**. Propositional logic mainly involves the study of logical operators such as **"AND"**, **"OR"** & **"IMPLIES"**, along with the rules determining the truth values of the propositions these symbols connect. Propositional logic also studies ways of modifying statements, such as the addition of the word **"NOT"** that is used to change an affirmative statement into a negative statement.

- A logical operator is considered truth-functional if the truth values of the statements it forms are entirely dependent on the truth or falsity of the component statements. The English words **"and,"** **"or,"** and **"not"** are examples of truth-functional operators. A compound statement formed with **"and"** is true if both component statements are true, and false if either or both are false. A compound statement formed with **"or"** is true if at least one of the component statements is true, and false if both are false. The **negation** of a statement, expressed by **"not,"** is **true** if and only if the original statement is **false**.

2.1.1 Examples of Propositional Logics

Propositional logic, or propositional calculus, involves using propositions and logical operators to form complex logical statements. Here are several examples that illustrate key concepts in propositional logic:

1. Basic Propositions

- ***p***: "It is raining."
- ***q***: "The ground is wet."

2. Negation

- $\neg p$: "It is not raining."
- $\neg q$: "The ground is not wet."

3. Conjunction

- $p \wedge q$: "It is raining, and the ground is wet."
  - This statement is true only if both ***p*** and ***q*** are true.

4. Disjunction

- $p \vee q$: "It is raining, or the ground is wet."
  - This statement is true if at least one of ***p*** or ***q*** is true.

5. Implication

- $p \rightarrow q$: "If it is raining, then the ground is wet."
  - This statement is true unless ***p*** is true and ***q*** is false.

2.1.2 Practical Examples of Propositional Logics

- **Example 1**: "If today is Monday, then tomorrow is Tuesday."
  - Let ***p***: "Today is Monday."
  - Let ***q***: "Tomorrow is Tuesday."
  - This can be written as $p \rightarrow q$.

- **Example 2**: "Either the train arrives on time, or we will be late."
  - Let ***p***: "The train arrives on time."
  - Let ***q***: "We will be late."
  - This can be written as $p \vee q$.

- **Example 3**: "The alarm did not go off, and I woke up late."
    - Let ***p***: "The alarm went off."
    - Let ***q***: "I woke up late."
    - This can be written as $\neg p \wedge q$.

2.2 Modal Logic: Theory and Challenges

[4, 5] Modal Logic is an extension of the propositional logic and it involves introducing of modalities which can be interpreted as **necessity (□)** and **possibility (◊)**. This extension allows for reasoning over a diverse set of models, which makes Modal Logic suitable in areas of interest such as philosophy, linguistics or even computer science. Seminal works in this area include a reference [2], which provides a comprehensive introduction to the syntax, semantics, and proof systems of modal logic.

- The nature of Modal Logic has been extensively studied and particularly with respect to its computational feature, which we have seen above is highly complex. [12] Some researchers presented the difficulties involved in decision procedures in modal logics, hence showing that modal reasoning is compound, especially in multi-modal systems. These include the evaluation of which general modal formulas can become computationally costly as well as the development of effective algorithms for satisfiability of general modal knowledge.

- [2] Modal logic is an improvement on classical logic which adds operators related to what is known as modality – the nature of necessity and possibility. The advantage of such a framework is that it is possible to get a much more nuanced picture of the truth of the statements in question by considering what is possible or necessary in addition to considering what is purely factual. The set of operators that belong to the so called "modal logic" consists of two main operators: □ – **Necessarily** and ◇ – **Possibly**. For instance, if $\Box p$ is to be **true** then it shows that ***p* is true in all the possible worlds** and if $\Diamond p$ is to be **true** then **there exists at two more than one world in which *p* is true**. In this regard, the truth of modal statements is considered by using what are referred to as possible worlds which are hypothetical states of affair.

- [4] With respect to syntax, a system of modal logic is an extension of a system of propositional logic with these modal operators. Semantics for modal logic is very often using **Kripke models**, which are defined by a set of states, an accessibility relation and a valuation function which assigns truth values to propositions in individual states. This setup is used in the study of how truth values of statements vary from one world to the other.

- Modal logics are many-dimensional, with each dimension being called a system and defined by the system's unique axioms and rules. The simplest system, known as **K**, consists of the following: an axiom, $\Box(p \rightarrow q) \rightarrow (\Box p \rightarrow \Box q)$. In System T the following axiom is introduced $\Box p \rightarrow p$ which states that if something is necessarily true it is true.

- There are many usages of modal logic, it crosses many fields of study. As for science, its application, it assists to study meanings of necessity and possibility in a philosophical context. It is used in computer science in order to verify software and hardware systems in which understanding one or more states or transitions is imperative. In linguistics, modal logic helps to understand the semantics of modal expressions in first and second language. Also, modal logic is combined with other branches such as temporal logic, which concerns time, deontic logic, which concerns norms and permits, and epistemic logic, which concerns knowledge and belief. Due to this flexibility modal logic is suitable when used for theoretical analysis as well as for finding practical solutions to problems.

2.2.1 Examples of Modal Logics

1. Necessity (□) Operator **:**

□***p*** means "***p*** is necessarily true." This indicates that ***p*** is true in all possible worlds.

Example 1:

- **Statement:** "□(All bachelors are unmarried)."
- **Interpretation:** It is necessarily true that all bachelors are unmarried. This means that in every possible world, the definition of a bachelor includes being unmarried.

Example 2:

- **Statement:** "□(2 + 2 = 4)."
- **Interpretation:** It is necessarily true that 2 + 2 equals 4. This is a mathematical truth that holds in all possible worlds.

Example 3:

- **Statement:** "□(If it rains, the ground will be wet)."
- **Interpretation:** It is necessarily true that if it rains, the ground will get wet. In every possible scenario where it rains, the ground being wet is a consequence.

2. Possibility (◇) Operator :

◇***p*** means "***p*** is possibly true." This indicates that ***p*** is true in at least one possible world.

Example 1:

- **Statement:** "◇(There is life on other planets)."
- **Interpretation:** It is possible that there is life on other planets. This means that in at least one possible world, life exists elsewhere in the universe.

Example 2:

- **Statement:** "◇(You will win the lottery tomorrow)."
- **Interpretation:** It is possible that you will win the lottery tomorrow. This doesn't guarantee it will happen, but there is at least one possible world where this is true.

Example 3:

- **Statement:** "◇(The sun will not rise tomorrow)."
- **Interpretation:** It is possible that the sun will not rise tomorrow. Although highly unlikely, there is at least one conceivable scenario where this could happen (e.g., a catastrophic event).

2.2.2 Combined examples of ◇ & □

Modal operators can be combined in statements to express more complex ideas.

Example 1:

- **Statement:** "◇(□(***p***))"
- **Interpretation:** It is possible that is necessarily true. In other words, there exists a possible world where P is true in all possible worlds accessible from it.

Example 2:

- **Statement:** "□(◇(***p***))"
- **Interpretation:** It is necessarily possible that it is true. This means that in all possible worlds, there is some world where ***p*** is true.

2.2.3 Practical Examples of Modal Logics

Example 1:

- **Statement:** "□(Stealing is illegal)."
- **Interpretation:** Stealing is necessarily illegal. In every possible world (legal system), stealing is prohibited.

Example 2:

- **Statement:** "◇(Stealing is legal in some countries)."
- **Interpretation:** It is possible that stealing is legal in some countries. There could be at least one possible world where stealing is not against the law.

Example 3:

- **Statement**: "□(Humans need oxygen to survive)."
- **Interpretation**: Necessarily, humans need oxygen to survive. This is true in all possible worlds where the laws of biology as we know them hold.

Example 4:

- **Statement**: "◇(Humans could evolve to breathe underwater)."
- **Interpretation**: It is possible that humans could evolve to breathe underwater. This could be true in some possible world where evolution takes a different path.

2.3 The Semantic Tableaux Method: An Overview

The **Semantic Tableaux method** is a proof technique that systematically checks the satisfiability of logical formulas. [29] It was introduced in 1968, and further developed in the context of automated reasoning. It operates under the method of attempting all the possible truth assignments of a given formula by building a tree-like structure known as the tableau. If the tableau closes, then the formula is unsatisfiable; otherwise, it is satisfiable.

Semantic Tableaux are highly effective because of the graphic and strictly formalised approach to the work with logic, thus they may be used both in training logical thinking and in real-life problem solving. They have essentially been applied within automated theorem proving as well as in logic programming.

2.3.1 Definition & Rules of Tableaux

A signed formula is a pair consisting of a truth value and a sentence, represented as either **T*p*** or **F*p***. Intuitively, **T*p*** can be read as "***p*** might be **true**" and **F*p*** as "***p*** might be **false**" in some structure. Each signed formula in the tableau tree is either an initial assumption, listed at the top of the tree, or it is derived from a signed formula above it using a set of inference rules. There are two rules for each main logical operator of the preceding formula: one for when the sign is **T** and one for when the sign is **F**. Some rules cause the tree to branch, while others add signed formulas to the current branch. A rule can be applied not just to the immediately preceding signed formula, but to any signed formula along the path from the root to the current point in the branch.

- A branch is closed if it contains both **T*p*** and **F*p*** for some formula ***p***. A tableau is closed if every branch in the tree is closed. According to the intuitive interpretation, each branch represents a possible scenario, but **T*p*** and **F*p*** cannot both be true in the same scenario, hence a closed branch represents an impossibility (contradiction). Apparently, if a branch is closed, it indicates that the scenario it represents has been eliminated as a possibility. Specifically, a closed branch means that it is impossible for all assumptions of the form **T*p*** to be true and all assumptions of the form **F*p*** to be false at the same time.

- A closed tableau for ***p*** is a tableau where the root is **F*p*** and all branches are closed. If such a closed tableau exists, it signifies that all scenarios where ***p*** is false have been eliminated. Consequently, ***p*** must be true in every possible structure.

2.3.2 Propositional Logic rules for Tableaux

[23, 30]In propositional logic, tableaux are used to systematically explore the truth or falsity of logical statements. Assuming we have propositions ***p*** and ***q***, which are either signed with **T** or **F** . Propositional rules for Tableaux will be as followed:

1. Negation (¬) ["NOT"]

- **T¬*p***: If the negation of ***p*** is **true**, then ***p*** is **false**.

T¬*p*
F*p*

- **F¬*p*:** If the negation of ***p*** is **false**, then ***p*** is **true**.

F¬*p*
T*p*

2. Conjunction (⋀) [“AND”]

- **T(*p* ⋀ *q*):** If the conjunction of ***p*** and ***q*** is **true**, then both ***p*** and ***q*** are **true**.

T(*p* ⋀ *q*)
T*p* **T*q***

- **F(*p* ⋀ *q*):** If the conjunction of ***p*** and ***q*** is **false**, then both ***p*** and ***q*** are **false**.

F(*p* ⋀ *q*)	
F*p*	**F*q***

3. Disjunction (⋁) [“OR”]

- **T(*p* ⋁ *q*):** If the disjunction of ***p*** and ***q*** is **true**, then at least one of ***p*** or ***q*** is **true**.

T(*p* ⋁ *q*)	
T*p*	**T*q***

- **F(*p* ⋁ *q*):** If the disjunction of ***p*** and ***q*** is **true**, then at least one of ***p*** or ***q*** is **true**.

F(*p* ⋁ *q*)
F*p* **F*q***

4. Implication (→) [“IF ... THEN ...”]

- **T(*p* → *q*)**: If the implication ***p*** **→** ***q*** is **true**, then either ***p*** is **false** or ***q*** is **true**.

T(*p* → *q*)	
F*p*	**T*q***

- **F(*p* → *q*)**: If the implication ***p*** **→** ***q*** is **false**, then either ***p*** is **true** or ***q*** is **false**.

F(*p* → *q*)
T*p*   **F*q***

2.3.3 Examples of Propositional Logic in Tableaux

- **Formula:** ***p → (q→r)***

  Tableaux Analysis:

  1. Start by assuming the negation of the formula to check for validity:
     - **F*(p → (q→r)***
  2. According to the implication rule, this expands to:
     - **T*(p)***
     - **F*(q → r)***
  3. Further expansion using the implication rule:
     - **T*(p)***
     - **T*(q)***
     - **F*(r)***
  4. The branch doesn't close (no contradictions like **T*(r)*** and **F*(r)*** appearing in the same branch), meaning the negation is **satisfiable.**

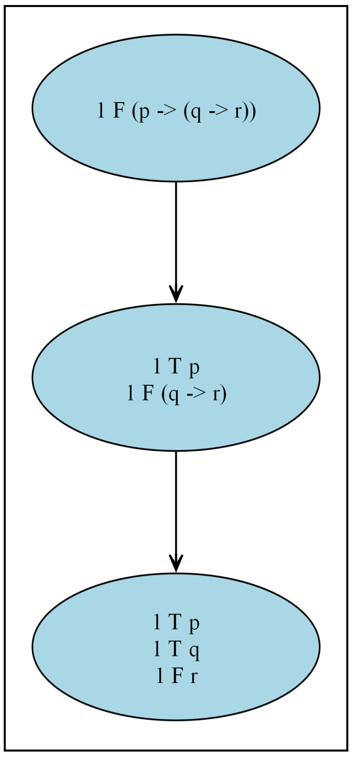


Fig 2.1 Tableaux Tree Branching for ***p → (q→r)***

- **Formula: *(p ∨ q) → (p ∧ q)***

  Tableaux Analysis:

  1. Assume the negation to test for validity:
     - **F*((p ∨ q)→(p ∧ q))***
  2. Using the implication rule, this expands to:
     - **T*(p ∨ q)***
     - **F*(p ∧ q)***
  3. The disjunction ***T(p ∨ q)*** expands as:
     - **T*(p) or* T*(q)***
  4. The conjunction **F*(p ∧ q)*** expands as:
     - **F*(p) or* F*(q)***
  5. Explore the branches:
     - Branch 1: **T*(p),* F*(p)*** → This branch **closes** (contradiction).
     - Branch 2: **T*(q),* F*(q)*** → This branch **closes** (contradiction).

  Conclusion: All branches close, meaning the formula ***(p ∨ q)→(p ∧ q)*** is **valid**.

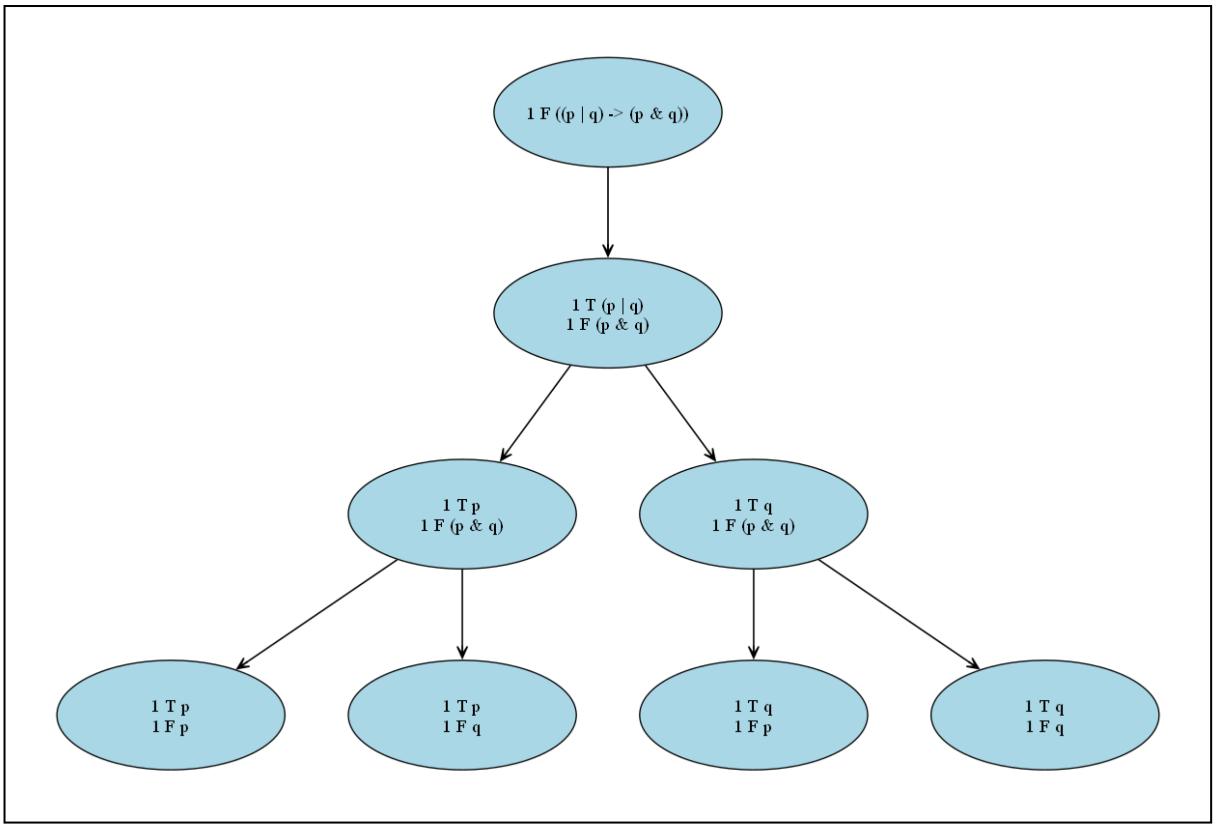


Fig 2.2 Tableaux Tree Branching for ***(p ⋁ q) → (p ⋀ q)***

2.3.4 Modal Logic rules for K in Tableaux

[31] In modal logics, we have to extend the notion of signed formula and add rules that has □ and ♢. In addition to a sign(**T** or **F**), formulas in modal tableaux also have prefixes **σ**. the prefix names a world in a model that might satisfy the formulas on a branch of a tableau, and if **σ** names some world, then **σ.n** names a world accessible from (the world named by) **σ**.

1. Necessarily (□)

- **σ T □ *p*:** If □***p*** is true in the context **σ**, then ***p*** must be true in all worlds accessible from **σ**.

σ T □ *p*
σ.n T *p*

The prefix **σ.n** represents a new world accessible from σ, and the rule asserts that ***p*** must hold in this new world. Here, **σ.n** is an already used world (not new).

- **σ F □ *p*:** If □***p*** is false in the context **σ**, there must exist at least one world (represented by **σ.n**) where ***p*** is false.

σ F □ *p*
σ.n F *p*

Here, **σ.n** is a new world created by this rule.

2. Possibly (◊)

- **σ T ◊ *p*:** If ◊***p*** is true at context **σ**, there must exist at least one accessible world **σ.n** where ***p*** is true.

σ T ◊ *p*
σ.n T *p*

The world **σ.n** is newly introduced here.

- **σ F ◊ P:** If ◊***p*** is false at context **σ**, then ***p*** must be false in all accessible worlds from **σ**.

σ F ◊ *p*
σ.n F *p*

The world **σ.n** in this context has already been used (not new).

2.3.5 Examples of Modal Logic in Tableaux

- **Formula: □(*p*→*q*) → (□*p*→□*q*)**

  Tableaux Analysis:

  1. Assume the negation to check for validity:
     - **F(□(*p*→*q*)→(□*p*→□*q*))**
  2. Apply the implication rule:
     - **T(□(*p*→*q*))**
     - **F(□*p*→□*q*)**
  3. Expand using the rules for necessity (□) and implication:
     - **T(□(*p*→*q*))** implies that **T(*p*→*q*)** in all accessible worlds.
     - **F(□*p*)** and **T(□*q*)** imply there is at least one world where **F(*p*)** but **T(*q*)**.
  4. Check all accessible worlds. If any accessible world shows a contradiction, the branch closes.

  Conclusion: If all branches close, the formula is **valid**; otherwise, it's **satisfiable** but not valid. The Tree Branching image is available at *Figure 2* in Appendices.

- **Formula:** ***◊(p ⋁ q) → (◊p ⋁ ◊q)***

  Tableaux Analysis:

  1. Assume the negation to test for validity:
     - **F*(◊(p ⋁ q)→ (◊p ⋁ ◊q))***
  2. Apply the implication rule:
     - **T*(◊(p ⋁ q))***
     - **F*(◊p ⋁ ◊q)***
  3. Expand the **possibility (Diamond)** and **disjunction** (⋁) rules:
     - From **T*(◊(p ⋁ q))***, there exists a world ***σ1*** where **T*(p ⋁ q)***.
     - From **F*(◊p ⋁ ◊q)***, expand the disjunction:
       - **F*(◊p)***
       - **F*(◊q)***
  4. In world ***σ1***, expand ***T(p ⋁ q)*** using the **disjunction** rule:
     - **T*(p)*** or **T*(q)*** in the world ***σ1***.
  5. For **F*(◊p)*** and **F*(◊q)***:
     - **F*(◊p)*** means in **all accessible worlds**, **F*(p)***.
     - **F*(◊q)*** means in **all accessible worlds**, **F*(q)***.
  6. Now explore the branches:
     - Branch 1: If **T*(p)*** in ***σ1***, this contradicts **F*(◊p)*** because **T*(p)*** in any world implies ◊***p*** is true. Thus, this branch **closes**.
     - Branch 2: If **T*(q)*** in ***σ1***, this contradicts **F*(◊q)*** because **T*(q)*** in any world implies ◊***q*** is true. Thus, this branch **closes**.

  Conclusion: All branches close, meaning the formula ***◊(p ⋁ q)→(◊p ⋁ ◊q)*** is **valid** in modal logic. The Tree Branching image is available at *Figure 1* in Appendices.

2.6 Kripke Models

[1, 4, 8] Kripke model is a formal semantics applied to a class of modal statements with necessity ($\square$ denotes 'it is necessary that') and possibility ($\Diamond$ for 'it is possible that') operators. It offers a method to assess such statements as they relate to distinct worlds of possibility. Kripke models are one of the distinctive linguistic tools of modal logic, helping to explain how truths can differ from one another.

2.6.1 Components of a Kripke Model

A Kripke model typically consists of three components:

1. **A Set of Possible Worlds (W)**:
   - These are different "states" or "scenarios" in which propositions can be true or false. Each possible world represents a complete way the world could be.
2. **An Accessibility Relation (R)**:
   - This is a relation between possible worlds, usually denoted as $R \subseteq W \times W$. If wRw', it means that world w' is accessible from world w. The accessibility relation defines how worlds are connected or related to each other.
3. **A Valuation Function (V)**:
   - This function assigns truth values to each proposition at each possible world. Specifically, for each atomic proposition p, $V(p)$ gives the set of worlds in which p is true.
   - For example, if p is true in the world w, then $w \in V(p)$.

2.6.2 Definition of a Kripke Model

A Kripke model M is defined as a triple $M=(W,R,V)$, where:

- W is a non-empty set of possible worlds.
- R is a binary relation on W (the accessibility relation).
- V is a valuation function V : **Atoms**$\rightarrow 2^{W}$ that assigns to each atomic proposition a set of worlds.

2.6.3 Evaluating Modal Formulas in a Kripke Model

To evaluate the truth of a modal formula at a particular world in a Kripke model:

- **Atomic Propositions:** $M, w \vDash p$, if and only if $w \in V(p)$.

    - This means that an atomic proposition ***p*** is true at world ***w*** if ***w*** is in the set of worlds where ***p*** holds.
- **Negation:** $M, w \vDash \neg p$, if and only if $M, w \nvDash p$.
    - If ***p*** is not true in ***w***, then $\neg p$ is true in ***w***.
- **Conjunction:** $M, w \vDash p \wedge q$, if and only if $M, w \vDash p$ and $M, w \vDash q$.
    - Both ***p*** and ***q*** must be true in ***w***.
- **Disjunction:** $M, w \vDash p \vee q$, if and only if $M, w \vDash p$ or $M, w \vDash q$.
    - At least one of ***p*** or ***q*** must be true in ***w***.
- **Necessity:** $M, w \vDash \Box p$, if and only if for all w' such that wRw', $M, w' \vDash p$.
    - $\Box p$ is true in ***w*** if ***p*** is true in all worlds accessible from ***w***.
- **Possibility:** $M, w \vDash \Diamond p$, if and only if there exists a w' such that wRw' and $M, w' \vDash p$.
    - $\Diamond p$ is true in ***w*** if there is at least one accessible world w' where ***p*** is true.

2.6.4 Example of a Kripke Model

Suppose we have the following Kripke model $M = (W, R, V)$:

- **Possible Worlds:** $W = \{w1, w2, w3\}$
- **Accessibility Relation:** $R = \{(w1, w2), (w2, w3), (w3, w1)\}$
- **Valuation Function:**
    - $V(p) = \{w1, w3\}$
    - $V(q) = \{w2\}$

Interpretations:

- ***p*** is true in worlds ***w1*** and ***w3***.
- ***q*** is true in world ***w2***.

Using this model:

- **Necessity:** To check if $\Box p$ is true at ***w1***, we must check if ***p*** is true in all worlds accessible from ***w1*** (which is only ***w2***). Since ***p*** is not true in ***w2***, $\Box p$ is false in ***w1***.
- **Possibility:** To check if $\Diamond p$ is true at ***w1***, we check if ***p*** is true in at least one accessible world. Since ***p*** is true in ***w3***, which is accessible from ***w2*** (and thus from ***w1*** to ***w2***), $\Diamond p$ is true at ***w1***

2.7 Tableaux Results

Closing tableaux are also applicable to more than the definition of logical formulas and assist in the categorization of different types of formulas such as contingent formulas, tautologies, and contradictions. It is now time to discuss how these outcomes are recognized during the tableau method broken down as follows.

1. Contingent Formulas

A **contingent formula** is a statement that can be either true or false depending on the situation or interpretation. In terms of tableaux:

- **Tableaux Representation**: In the tableau for a contingent formula there must exist at least one branch that is open which shows that the given formula is true in some interpretation and at least one closed branch to show that the given formula is false in some other interpretation.
- **Example**: Consider the formula $p \to q$. The truth of this formula depends on the truth values of p and q:
    - If p is false, the formula is true regardless of q (open branch).
    - If p is true and q is false, the formula is false (closed branch).

Thus, the tableau will have a mix of open and closed branches, showing that the formula is contingent.

2. Tautologies

A **tautology** is a statement that is true in every possible interpretation. In the tableau method:

- **Tableaux Representation**: All branches can then be closed for a tableau of a tautological formula which means that there is no way that the formula can be false.
- **Example**: Consider the formula $p \lor \neg p$ (the law of excluded middle). No matter the truth value of p:
    - If p is true, the formula $p \lor \neg p$ is true.
    - If p is false, $\neg p$ is true, making the formula true as well.

The tableau would close all branches, indicating that the formula cannot be false in any interpretation, and hence, it is a tautology.

3. Contradictions

A **contradiction** is a statement that is false in every possible interpretation. Within the tableau framework:

- **Tableaux Representation**: For tableaux of a contradictory formula there will be at least one open branch to illustrate that there is a possibility that the formula could be true, but then p and $\neg p$ will appear in the same branch hence closing the branch.
- **Example**: Consider the formula $p \land \neg p$. This formula asserts that p is both true and false simultaneously:
    - When you begin to explore this formula using a tableau, you will find that every possible branch leads to a contradiction (both p and $\neg p$ appear in the same branch).

Thus, every branch in the tableau will be closed, showing that the formula cannot be true under any interpretation, making it a contradiction.

2.8 Existing Tools and Automated Solvers

Many softwares have been designed for the purpose of mechanical reasoning based on Propositional and Modal Logic. Some of the notable ones include the following:

A. Prover9

[20] **Prover9** is a combined first-order and equational logic theorem prover; it has an automated system that is commonly used. It builds up contrapositives from a set of logical axioms and hence gives a proof of a formula in use of resolution-based methods. Nevertheless, as mentioned earlier, **Prover9** is not intentionally designed for solving problems under modal logic, but with proper extension, one can translate modal axioms into first order logic and solve the problem in **Prover9**.

The main advantage of using **Prover9** is the ability to work with various logical statements and use a number of logical systems with the possibility of working examples. However, the native modal logic support is not available which poses some problems. Subsumption of modal logic to first-order logic frequently entails the stringent translation of modal operators (for example, **necessity**, $\Box$, and **possibility**, $\Diamond$), with such translation creating cumbersome and occasionally suboptimal proofs. Nonetheless, **Prover9** is a highly useful tool for generalised logical computation where first-order logic is sufficient or modal logic can be converted to first-order logic.

However, for users with interest in working with modal logic, this is perhaps not the best tool since working with **Prover9** often involves some level of professional input in formulating the modal issues and interpreting the outcomes. Additionally, such a resolution-based approach may be less effective in a case of the use of modal operators to which tableau-based or model-based methods are more suitable.

B. MleanCoP

[24] **MleanCoP** is an implementation of connection-based theorem prover aimed for the development of both classical and non-classical logics such as modal logic. Although **Prover9** has been found to be a very powerful tool to work on problems related to modal logic, **MleanCoP** has been specifically designed for it and thus offers a strategic approach for solving problems that fall under this domain. Its core algorithm is based on a so-called low-complexity tableau calculus that is intended to be an efficient method of finding proof-solutions.

Due to its simplicity and orientation toward the minimization of communication, **MleanCoP**'s greatest strength lies in the fact that it can accommodate many diverse and logical systems, ranging from propositional, first-order, and even modal logistics. Its tableau based method is more suited for the modal logic in particular because the division of the formula into a simpler sub formula closely related to the step by step expansion of the statement in the tableau method.

In addition, due to extensibility, **MleanCoP** is applicable for research and education. It makes it possible to change and expand the system in order to incorporate new logical rules or modalities making it a very effective tool for experimenting with computationally non-standard or user-defined modalities. However, similar to what as seen earlier, using **MleanCoP** requires advanced knowledge of logic, and therefore it is suitable more for advanced or end-users or for researchers in the field of logic.

C. Tree Proof Generator

[27] **Tree Proof Generator** is a web based system for constructing and experimenting with trees in different systems of logics with an interest in modal logics. The key advantage of the tool is the possibility to easily visualise tableau proofs, which are often applied in modelling modal logic to assess the feasibility of formulas. It also displays the countermodels of relevant formulas.

Among the major benefits of **Tree Proof Generator** is the ability of the client to engage in the process of creating and experimenting with logical proofs in the interface. The tool helps to build tableau trees in the simplest way since logical formulas are depicted in the form of decompositions with the help of the tool. That is why it is rather useful in courses on modal and other non-classical logic since the mechanism of the operation of the tool is transparent, and the process of the analysis of formulas is subdivided into steps.

For example, Tree Proof Generator is very different from tools like **Prover9** or **MleanCoP** in the sense that the latter are based on automated theorem prover while the former relies on visual representation and the active participation of a user. It does not have the ambition to be a complete version of automated theorem prover but rather to be a tool which can be used for the analysis and exploration of the structure of the logical proofs. This makes it suitable for users who are studying modal logic for the first time or users who wish to monitor the process occurring in a tableau proof.

Besides, **Tree Proof Generator** also allows investigation of other types of logical systems which is also quite valuable. Moreover, **Tree Proof Generator** is specifically designed for modal logic only, which is also quite valuable. Nevertheless, it may take more time in the larger and complicated proofs since it is basically only used for manual construction and exploration unlike the other systems such as **Prover9** and **MleanCoP** which are fully automated.

2.9 Gaps in Current Solutions

Despite the advancements in logical reasoning tools, several gaps remain that this project aims to address:

1. **Integration of Propositional and Modal Logic:** While many tools exist that help students analyse either Propositional or Modal Logic, few of the tools are integrated. But this work intends to meet this gap by providing a single site for reasoning on both types of logic by using the Semantic Tableaux method.
2. **Interactive and Visual User Interface:** It can be seen that most of the tools that are available do not have an easy to use graphical interactivity environment where the user can enter various formulas and observe the generated tableaux and the results that are obtained. The proposed Semantic Tableaux Solver has a greater focus on the interactive and to a certain extent graphical nature, making it more user friendly.

3. **Educational Focus:** There are many tools which are developed for research orientation but still there is a need for a tool which is educational oriented and makes the complex logical functions understandable for students as well as learners. By doing so, this project will fill the above identified gap and present the logical processes in a much simpler and elegant manner.

2.10 Conclusion

This chapter has discussed the Propositional and Modal Logic systems together with their theoretical importance and also the importance of the Semantic Tableaux method. Much progress has been made in creating programs and tools for generating automated solves, but there currently lacks an all-encompassing, graphical application which incorporates Propositional as well as Modal Logic with a focus on practicality and user engagement. Based on the features of the currently available solvers, this project aims at filling these gaps by proposing the new Semantic Tableaux Solver that will incorporate all these features and thus enhance the practical and learning aspects of logical reasoning.

Chapter 3

Requirements & Analysis

3.1 Overview of Project Requirements

This chapter advances the goals of the project, as presented in the Introduction, with a full discussion of the parameters required for the semantic tableaux solver. The official planned goals that arise from the need to achieve a scalable and heavily optimised modal logic analysis tool with an intuitive user interface reflects in the given requirements.

- On the technical level, the project is based on the need for a highly efficient and precise semantic tableaux method for modal logic. This core functionality has to be achieved together with the efficient and friendly web-based user-interface which has to translate a number of logical operations into a simple and clear form. In addition to being a logical problem solver, the system has to convey the results and the **Tableaux Trees** and **Kripke Accessibility Relationship Models** in an understandable format.

Key areas of focus in our requirements analysis include:

- **Formula Parsing and Representation:** The system is also required to translate a rather wide set of modal logic formulas and perform well with different operators as well as nested forms. Tableaux Algorithm Implementation: The core logic engine has to be optimised to perform efficiently the semantic tableaux method, as well as to deliver the correct final verdict from the point of view of validity or satisfiability.
- **Visualisation Generation:** Shamefully, often the complexity of tableaux and Kripke models is concealed and, thus, important to draw clear and informative pictures.
- **User Interface Design**: The web application is required to allow a convenient input of the formula and output display of result along with the options to control and access the solution history.
- **Performance and Scalability:** In case of simple formulas the System should be able to solve the formulas within a reasonable timeframe and should be able to gracefully handle timeouts.
- **Error Handling and User Guidance:** More particularly, the error messages have to be rather effective, as well as the information provided to a user has to be sufficient for him/her to navigate through the problem-solving procedure.
- **Educational Value:** The system should go further than simply computing the output of a given input formula, which might be helpful for verifying the correctness of a formula, for example, by presenting a mathematical proof ; the structure and the properties of the given input formulas should also be outputted; which makes the system very valuable from an educational perspective.

In the following sections, these areas will be subdivided into functional and non-functional requirements and the system architecture that would enable the achievement of these requirements will also be discussed. In addition, the strategies that will be used in testing and assessment of the final product will also be described. This analysis will therefore be applied in the next steps of the design and implementation functions of the project.

3.2 Functional Requirements

3.2.1 Semantic Tableaux Algorithm Implementation

The main component of the system is the embodiment of the semantic tableaux method for modal logic. This is the algorithm that powers the logical analysis and therefore, this has to be optimal and accurate.

The tableaux algorithm should be capable of:

- Checking both the validity and satisfiability of input formulas
- Correctly handling all supported logical operators within the tableaux expansion process
- Implementing modal logic inference rules for box and diamond operators
- Detecting closed branches in the tableaux
- Determining the final result (valid/not valid, satisfiable/not satisfiable) based on the tableaux expansion

The algorithm should be able to respond to a variety of formulas including propositional formulas and many cases of first-order expressions with different modalities and operators involved.

A. Formula Input and Parsing

The semantic tableaux solver requires a reliable and efficient way that can be used by the user to feed in the modal logic formulas he wants to analyse. It is the first component in this system and its importance cannot be overemphasised as it plays a major role in enhancing the efficiency of the rest of the functionality.

The system should feature a text interface that allows users to enter modal logic formulas using a comprehensive set of operators. These include:

- Standard propositional logic operators: **negation (~)**, **conjunction (&)**, **disjunction (|)**, and **implication (->)**
- Modal operators: **necessity ([] or □)** and **possibility (<> or ◇)**
- Atoms: represented by lowercase letters (e.g., *p*, *q*, *r*)
- Parentheses: for grouping sub-expressions and clarifying formula structure

Since UTF-8 encoding does not recognise the actual operator symbols, the alternate symbols are used to illustrate and implement the logic. Furthermore, it will be more convenient to find these symbols in regular keyboards.

In this regard, when the system takes the input, it has to critically check and ensure that the formula inputted is valid. This real time validation should give users instant feedback of any syntactical errors that might be a result of wrong formula writing. Because these expressions can be nested and contain other expressions they should be parsed into an internal representation that is easily processed by the tableaux algorithm.

Few examples of Parsed Formulas are as follows:

Input Formula	**Parsed Formula**
[](p | q) -> ([]p | []q)	([](p | q) -> ([]p | []q))
p & q -> r	((p & q) -> r)
[]p -> r & <>q | ~s	([]p -> (r & (<>q | ~s)))

Table 3.1 Illustration of Parsed Formula from Input Formula

B. Visualisation Generation

To bridge the gap between abstract logical concepts and user understanding, the system must generate clear, informative visual representations of the analysis results.

The visualisation component should produce:

- Visual depictions of the tableaux structure, showing the expansion of nodes and the relationships between different branches
- Graphical representations of Kripke model accessibility relations, illustrating the possible world semantics underlying the modal logic

These visualisations must be clear, readable, and well-labelled. For complex formulas that result in large structures, the system should implement functionality allowing users to zoom, pan, or otherwise navigate through the visualisations. This ensures that even intricate logical structures remain accessible and comprehensible.

C. Result Display and Analysis

The result presentation subsystem is considered one of the most important components since it reflects on the user's efficiency and effectiveness in the logical analysis part.

The result display should include:

- The parsed formula in a standardised, readable format
- Clear indications of the formula's validity and satisfiability
- The complete tableaux structure with all expanded nodes
- The Kripke model accessibility graph
- A detailed analysis of the formula's structure, including identification of modal operators and their nested relationships
- An explanation of the formula's overall logical structure (e.g., conjunctive, disjunctive)
- Insights into the logical properties of the formula based on the analysis results

This way, users with different levels of familiarity with modal logic will be provided with an overview of the findings, as well as complex explanations for those who would like to explore the details further.

D. User Interface and Interaction

The user interface is the intermediary that connects the optimised logical sequences of operation within the system and the user perspective. It has to be usable but also functional and it has to incorporate functionality into its usability.

Key features of the user interface includes:

- A clean, intuitive layout for formula input
- A prominent 'Solve' button to initiate the analysis process
- Loading indicators for longer computations to keep users informed of progress
- Interactive elements for exploring visualisations (e.g., zoom, pan controls)
- Clear error messaging for invalid inputs or processing issues
- A solution history sidebar, allowing users to revisit and modify previous solutions
- Options to delete entries from the solution history

It should be intuitive for users who do not have much experience while being versatile for users who are skilled in logical reasoning; it should allow for options that can be added or hidden depending on the user's needs.

E. System Interface Stories & Priorities

S. no.	Requirements	Priority
1	System should display the rules and examples of the formula that the user has to enter.	**HIGH**
2	System should have an input box for the user to enter a formula and a solve button.	**HIGH**
3	System should display the relevant results, analysis and visualisation in the display area.	**HIGH**
4	System should be able to solve formulas.	**HIGH**
5	System should store data of the solution that the user shall solve.	**MEDIUM**
6	System should have a history of the saved data and the user can delete or modify the data in a single session.	**MEDIUM**

Table 3.2 Requirements and their priorities

F. Educational Features

Since this system may also be used for teaching purposes, all components of the system should help the student grasp the concept of Modal logic.

These educational features could include:

- Explanations of key propositional logic & modal logic concepts relevant to the input formula
- Optional step-by-step breakdowns of the tableaux tree expansion process
- Clarifications on how the accessibility relations in the Kripke model relate to the formula's modalities
- Interactive tutorials or help sections explaining the use of the system and the principles of modal logic

These educational elements can be incorporated to such an extent so as to allow the system not only to be an effective tool for solving problems, but also to help students and all those who are interested in modal logic, in their studying process.

G. Performance and Scalability

For smooth operations for a wide range of formula; performance management features must be incorporated in the system.

These should include:

- A timeout mechanism for long-running computations
- User feedback when a computation exceeds the allocated time
- Potential options for users to adjust complexity settings or limits for formula processing
- A multithreading feature that allows to parse bigger and complexed formulas

The formulas within the system should be able to process simple as well as complex formulas seamlessly and there should also be adequate provision for cases where there is a time out.

3.2.2 Conclusion

Through satisfying these functional requirements across such domains, the semantic tableaux solver embodies a versatile, interactive means by which to parse through modal logic formulas. It will then be suitable for use in teaching in institutions of learning as well as in practical implementation in the areas of logic, philosophy and computer science.

3.3 Non-Functional Requirements

Non-functional requirements of the semantic tableaux solver project refer to a set of quality attributes which must be incorporated into the development of the system for it to be complete, efficient and sustainable from the perspective of its users. These requirements extend further than the initial needs of the system then define the way the functions of the system are met to the extent of how adequately the performance in the system is done in certain conditions. They include concepts such as *performance*, *utility*, *maintainability*, and *expandability*.
- These non-functional requirements are aimed at making sure that the semantic tableaux solver fulfils its simple and principal technical/functional requirements while being also efficient, easy to use, and maintainable for modal logic analysis. It caters for the needs of the entire user population ranging from a novice modal logic student to a professional researcher as well as the essential web development requirements for an application that is required to perform highly complex logical calculations.
- With these requirements in mind, the goal is to design a system which is functionally correct and optimised for good performance, easy to interact with besides being safe in its handling of user inputs, flexible for maintenance and updates, compatible on different platforms and as well amenable to adaptation to new functionalities in the future. These qualities are important in ensuring that the tool grows to meet the demands of the academic and research institutions in the long run.

3.3.1 Performance

Performance is one of the main criteria related to the semantic tableaux solver and depends on the satisfaction of the users and, thus, usability of the system. The criteria are centred on response time, handling of formulas and how graceful the program is when it experiences limitations in computations.

- **Response Time:** For easy to intermediate formulas, the result should be available within 2-3 seconds of submission of the formula. To provide an effective learning experience and ensure that the users keep coming back, this rapid response is important.
- **Scalability:** The way the formulas are used in the solver must be modular to handle formulas of varying complexity. It should be able to handle formulas with 20 atomic propositions and 20 modalities nested to 5 depth without drastic slow down in the overall functioning. This scalability makes sure that the tool can be applicable not just for academic exercises but for higher level research as well.
- **Timeout Mechanism:** For the complex formulas which may take a long time to compute, the system should incorporate a time out bar. By default, this should be a value of 30 seconds and after this computation time, it should come to an intelligent halt. Many applications incorporate timeouts, which the users should be notified of, even if there is an ongoing computation of a resource-intensive problem.

3.3.2 Usability

Usability is highly important for the semantic tableaux solver due to the broad range of users, from beginners in modal logic from computer science curricula to experts in the mathematical and philosophical fields. The usability requirements have the main goal of making it easy for the users to use the interface and find enjoyment from it.

- **Intuitive Interface:** Thus, the user interaction should be as clear as possible. First time users of the application should be able to key in their first formula and get results within 2 minutes of getting to the application without the need of being told what to do. This aspect of ease of use is important because it helps people to get engaged thus not making it difficult for them to master some of the features.
- **Clear Error Messages:** No error message should be complicated, lengthy or ambiguous and all that is displayed to the user should make relative sense. If possible, the messages should be written in simple language which the users can easily comprehend and make corrections on mistakes made. This is especially so because first time logicians might not know how modal logic is constructed or the syntax that is used in it.
- **Consistency:** It is necessary to adhere to the design patterns, colour schemes as well as the terminologies used throughout the application. This is important because it creates what is known as continuity in the mental map of the system and thus reduces the user's cognitive load.
- **Mobile Responsiveness:** It is supposed to be employed on computers with mouse and keyboard, but it should also be comfortably navigated with a tablet with a screen size of 7 inches and greater. This makes it possible for the users to have a feel of the tool on the different devices they own.

3.3.3 Maintainability

Here it is important to note that maintainability of the semantic tableaux solver is important for its future existence and development. These requirements page on the fact that the system must be easily modifiable, debuggable and extendible over time.

- **Modularity:** The system should have a modularity where some of the components should be replaceable to carry out the new changes easily. It is for this reason that this modularity should apply to major components including the parser, tableaux algorithm, and visualisation engine.
- **Documentation:** Documentation is critical when it comes to maintainability, hence, need to ensure that it is done to an extent. There should be commenting for inline codes, API documentation, and a user manual for the software. This is important since well documented code and systems are easy to comprehend, alter and expand.
- **Version Control:** All the source code should be kept on a version control system, preferably a Git. Uncomplicated but appropriate commit messages and a sound branching model should be used as ways of tracking and developing changes.
- **Testing:** Integrity of the system cannot be overemphasised hence the need to have a sound testing strategy in place. Test coverage should be at least 80% basic logic unit tests should be included while integration tests for the web interface should also be conducted. Thanks to this type of testing approach, it becomes easier to identify problems with the software and guarantee that new changes do not bring in more problems.

3.3.4 Extensibility

Extendibility concerns address the prospect of further additions and modifications to the system and concern the system's relevance in the long term.

- **Pluggable Components:** The system design should be flexible in such a way that it can easily be expanded to include additions such as new types of logical operators, visualisation types, or new analysis features that would not cause a large-scale redevelopment of the system. The above pluggable design makes it possible for the system to be modified to accommodate a new need that may arise or to implement new knowledge in modal logic theory.
- **Configuration:** There are some worked out values in the system, which control its working, for example, the time before a connection is terminated and the level of the problem's complexity – they all should be easily changed without coding. This is due to the fact that it enables the user to tweak with the performance of the system based on requirements, or personal preference of the intended field of use.

3.3.5 Conclusion

With consideration of these non-functional requirements, the semantic tableaux solver will fulfil its primary functions of modelling and reasoning with modal logic while also including additional features to ensure it is a reliable, easy to use, and adaptable tool for the future. These requirements guarantee that the system is going to be optimised, resilient, safe, and stamped for longevity though addressed for numerous users and purposes within academia and research domains.

3.4 Problem Analysis and Approach

In this section, the problem formulation of developing a Semantic Tableaux Solver will be divided into sub-problems and viable solutions will be introduced.

3.4.1 Core Components Analysis

The semantic tableaux solver can be broken down into several core components:

1. **Formula Parser:** It means the conversion of entered information by a user to an organised form.
2. **Tableaux Algorithm:** Utilises the logical rules in order to extend the formula.
3. **Kripke Model Generator:** Develops possible world models out of the tableaux.
4. **Result Analyzer:** Argues from the tableaux and the Kripke model for the notions of validity and satisfiability.
5. **Visualisation Engine:** Produces the drawings of the tableaux and the Kripke models.
6. **User Interface:** Employed for feed and data input as well as display of the results and user's control over the system.

3.4.2 Algorithmic Approaches

For the tableaux algorithm, we can consider two main approaches:

1. **Depth-First Expansion:** In this approach, each branch is exhaustively expanded before the backtracking to the previous node of the tree while constructing the tableaux. It's memory efficient, but can be far from achieving the minimum number of steps to reach a contradiction.
2. **Breadth-First Expansion:** This method develops all branches of the tableaux at once and is therefore a more effective method. It can perform Mini-SAT on the formula which will help it to find the shortest path to the contradiction but it will be consuming more memory as compared to others when solving complex formulas.

The selection of each presented approach will mainly depend on the performance testing, and the general level of the formula intricacy that will be entered by users.

3.4.3 Visualisation Strategies

For visualisation, we'll explore two potential approaches:

1. **Server-Side Generation:** Draw pictures on the server and then pass them on to the client. This decreases the loads on the clients but at the same time may lead to increased loads on the server and high network traffic.

2. **Client-Side Rendering:** Pass the data to the client and use such libraries as **Graphviz** or **pydot** for displaying. This load is delegated to the client but their client side code tends to be more complicated.

3.4.4 Kripke Model Generation

For Kripke model generation, we'll need to consider:

1. **World Creation:** The most effective ways of constructing and monitoring 'what if' scenarios.
2. **Accessibility Relation:** Best way to encode, and re-encode, the accessibility relation between worlds.
3. **Valuation Assignment:** The principles for determining truth values of propositions in a Worlds' scenario.

Implementation Steps

Based on the analysis, we can outline the following implementation steps:

1. Develop the formula parser
2. Implement the core tableaux algorithm
3. Create the Kripke model generator
4. Build the result analyzer
5. Develop the visualisation engine
6. Design and implement the user interface
7. Integrate all components into a cohesive system
8. Implement additional features (e.g., solution history, educational content)

3.5 Testing and Evaluation Strategy

To check the efficacy, accuracy and solidity of our semantic tableaux solve, we will develop the below testing and evaluation plans. This strategy is to be used to match with the implemented test cases and the particular needs of the project in reference.

3.5.1 Unit Testing

We will use Python's unittest framework to develop a suite of unit tests for each core component of the system. Our test cases will cover:

1. **Formula Parsing:**
    - Test the ***custom_parse_formula()*** function with various input formulas.
    - Ensure correct parsing of atoms, logical connectives, and modal operators.
    - Verify handling of nested expressions and parentheses.
2. **Tableaux Algorithm:**
    - Test the Tableaux class methods, particularly ***check_validity()*** and ***check_satisfiability()***.
    - Verify correct expansion of tableaux for different formula types.
3. **Kripke Model Generation:**
    - Test the creation and manipulation of possible worlds.
    - Verify correct implementation of accessibility relations.

Our unit tests will be organised into different categories:

a) **Valid Formulas in K:**

- Test propositional tautologies and K axioms.
- Ensure both validity and satisfiability for these formulas.

b) **Satisfiable but Not Valid Formulas in K:**

- Test formulas that are satisfiable but not valid in K modal logic.
- Verify that these are correctly identified as satisfiable but not valid.

c) **Unsatisfiable Formulas:**

- Test contradictions and other unsatisfiable formulas.
- Ensure these are correctly identified as neither valid nor satisfiable.

d) **Complex Formulas in K:**

- Test more complex modal logic formulas.
- Verify correct handling of nested modalities and complex logical structures.

e) **Nested Modalities in K:**

- Test formulas with deeply nested modal operators.
- Ensure the solver can handle complex modal expressions accurately.

3.5.2 Evaluation Metrics

To evaluate the success of the project, we will use the following metrics:

1. **Accuracy:** Relative frequency of correctly analysed formulas in K modal logic with respect to comparison with known results.
2. **Performance:** Response time for formulas of varying sophistication level so as to achieve a reaction time of not more than 3 seconds for moderately sophisticated formulas.
3. **Code Quality:** It can be regarded as a quantitative measure where maintainability index is aimed at being not lower than 80, according to results of the static analysis tools and code review.
4. **Test Coverage:** As we improve the test results, we should strive to attain eighty percent or more for our code coverage by the automated tests.

Chapter 4

Design

4.1 Design Approach and Methodology

This section covers the applied design and development process that has already been defined in the requirements and analysis chapter. Some key contents of this chapter are: Overview, Development Methodology, Tools and Technologies used, Choice Justification, System Architecture, and other component designs.

4.1.1 Overview of Design Philosophy

The various features of the semantic tableaux solver for modal logic are mainly developed with consideration of modularity, scalability, and user-centricity. To this end, the approach that we propose is to design a system that is mathematically sound while at the same time being easily understandable by users who may not necessarily have extensive knowledge in modal logic representations. The design philosophy emphasises:

1. **Separation of Concerns:** To define a clear interface between the "logic engine," that is the core of the software's functionality, and the view that implements the user interface so that the development and modifying can occur separately from one another.
2. **Extensibility:** Designing the system with a modular form so that further evolution of the system like supplementation of more modal logics is possible in the future.
3. **User Experience:** Prioritising an intuitive interface and clear visualisation of complex logical structures.
4. **Performance:** Balancing computational efficiency with the ability to handle complex formulas.

4.1.2 Agile Development Process

To be more flexible and implement improvements during the processes of the development, we have chosen to use Scrum which is an Agile development methodology. Key aspects of our Agile approach include:

- Sprint cycles of a week for values that are incremental, and feedback on results that is frequent.

- User stories to keep your eyes on the needs and wants of the end users.
- Integration lets us have constant code quality and system solidity.
- Sprint review and retrospective for the purpose of enhancing and focusing on interactions, as well as on working software.

This Agile approach enabled the changes between the requirements as the project was developed and the insight gained through the research process which was crucial in an academic setting.

4.1.3 Key Tools and Technologies

The choice of tools and technologies was made based on the necessity of performing effective logical computations, integrated development environment for the web-based application and ability to better visualise the results. Our key choices include:

1. **Python:** Chosen as the primary programming language since it is general-purpose and has a lot of related libraries.
2. **Streamlit:** Selected for the development of the Web Application GUI and can easily integrate with the Python-based back end.
3. **NetworkX and Pydot:** Used for graph operations and for visualisation, vital for displaying Kripke models' and tableaux structures.
4. **Matplotlib:** Used for creating clean, easily to modify representations of logical entities and their relations.
5. **Git:** Some of them include version control used in order to systematically manage change through the various versions in the developmental process.

4.1.4 Justification of Choices

Our design choices are justified based on several factors:

1. **Modularity:** The decoupling of the ability to work the problems and to visually navigate the program in the opposite order makes development and testing much more systematic.
2. **Scalability:** Python's efficiency, and the selected libraries offer a basis to address more intricate formulas in the future.
3. **User Accessibility:** It is easy to deploy with Streamlit which helps to develop a clean web interface, thus, minimise user complexity and avoid the need for local installation.
4. **Visualisation Clarity:** One has to know that the logical structure can be represented through NetworkX or Pydot; interactive or simple visual representations can be obtained through Matplotlib.
5. **Development Stability:** Agile methodology helps to maintain the required structure in the course of development, checking each phase and ensuring that it has been finished perfectly before the start of the next phase. The same is useful in a situation where the requirements of a project are well outlined and the goal is straightforward.

Taken together these design decisions are aligned with our goal of implementing a solid, easy to use, and easily extensible semantic tableaux solver for modal logic. The chosen approach provides the right amount of academic research and at the same time focuses on how to use the results practically in the following implementation stage. The Agile Methodology is very suitable for development as it presents a sequential and logical structure to the project which is in line with the academic nature of the project.

4.2 System Architecture

The application follows the MVC (Model-View-Controller) architecture design pattern. In this section, we will discuss the potential key components of our application.

4.2.1 High-Level Architecture (UML Component Diagram)

To allow for relatively easy modification in the future, the semantic tableaux solver has been constructed following the modular design principle. The system is composed of three main components:

1. Core Logic Engine
2. Web Application Interface
3. Visualisation Component

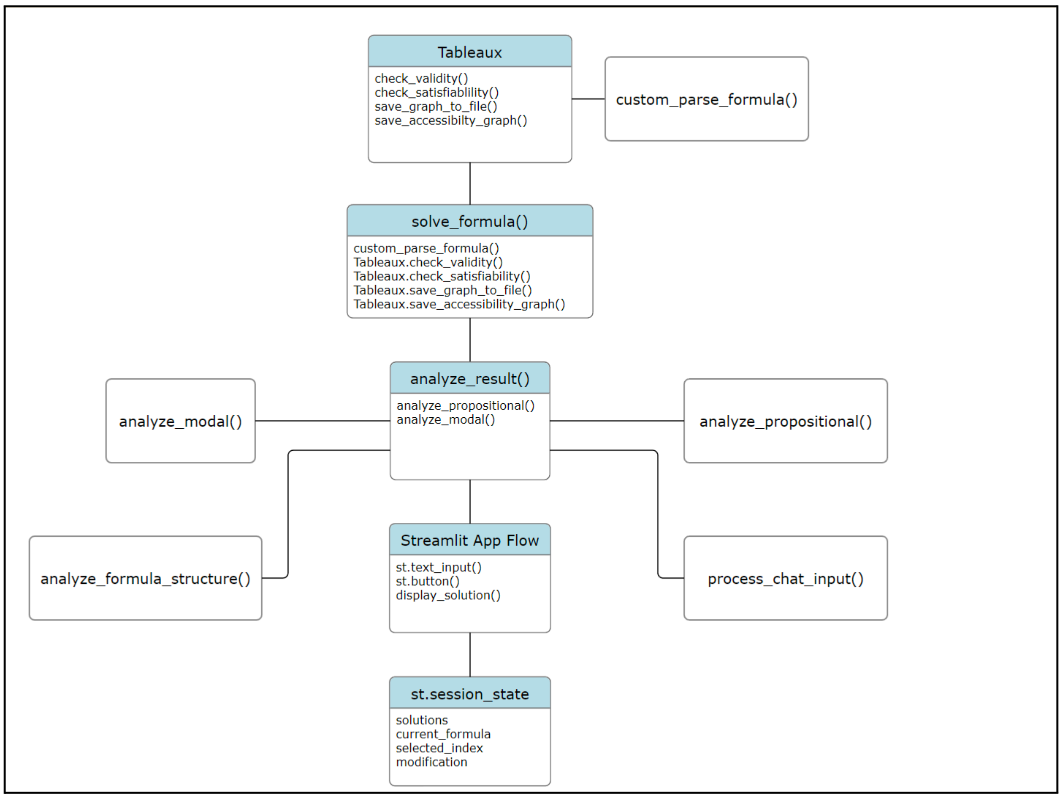


Fig 4.1 Classes and Methods

The UML Component Diagram illustrates the relationships and interfaces between these main components. The Core Logic Engine is the central component, with the Web Application Interface and Visualization Component depending on it.

4.2.2 Core Components and Their Interactions

1. **Core Logic Engine:**
    - **Responsibility**: Responsible for the tokenization of input formulas, as well as the performance of semantic tableaux and the creation of Kripke models.
    - **Key Modules:**
        - **Formula Parser:** Translate strings submitted for evaluation into objects that represent logical formulas.
        - **Tableaux Solver:** Explains the implementation of the semantic tableaux method for the different types of modal logic.
        - **Kripke Model Generator:** Kripke model structures and organises them.
    - **Interfaces:** It offers procedures for formula checking, construction of tableaux and satisfiability/validity analyses.

2. **Web Application Interface:**
    - **Responsibility:** Can handle user interaction and result.
    - **Key Features:**
        - **Formula Input:** This feature involves the formula input of the modal logic form by the users.
        - **Result Display:** Submits validity/satisfiability reports and reasons.
        - **History Management:** Maintains a record of solved formulas and allows modification and deletion.
    - **Interactions:** Sends the entered user input to the Core Logic Engine for further processing. It is the one that receives results and sends it to the Visualization Component to translate into some form of visualisation.

3. **Visualisation Component:**
    - **Responsibility:** Creates images of tableaux and Kripke models.
    - **Key Features:**
        - **Tableaux Tree Renderer:** it is used to generate graphical representation of the existing tableaux kind of structures.
        - **Kripke Model Graph Generator:** Essentially, sees Kripke models in terms of graphs.
    - **Interactions:**
        - Calculates on the data it receives from the Core Logic Engine to develop visualisations.
        - Delivers rendered visualisations to the Web Application Interface for showing.

Dataflow:

- The user enters a formula through the Web Application Interface.
- The formula is then forwarded to Core Logic Engine for its parsing as well as further processing.
- The Core Logic Engine is employed in the application of the tableaux methods and produces outcomes.
- They are then forwarded to the Visualization Component for depiction in graphical form.
- The Web Application Interface shows the results and the visualisations to the user.

4.2.3 Rationale for Architectural Decisions

- **Modularity:** The three-layer layered design helps to test each component separately in an efficient manner. This modularity also makes it easier to provide future upgrades and easier maintenance works.
- **Separation of Logic and Presentation:** We do not allow the computations in the Core Logic Engine to have influence on the presentation layer from the Web Application Interface. This makes it possible to possibly implement implementations of desired new forms of interfaces in the future without affecting the essence of the system.
- **Centralised Logic Engine:** The maintenance of the Core Logic Engine as centralised makes it easier for the application to process and deliver formulas and results uniformly hence consistency across the application.
- **Dedicated Visualisation Component:** This way, we can easily change the visualisation logic, or replace the used techniques without the impact on the other components.
- **Stateless Core Logic:** One of the main ideas used in Core Logic Engine is its statelessness – the engine is capable of processing each request separately. This design choice is helpful in terms of scalability and helps in reducing the number of errors that are present in the system.
- **Web-Based Interface:** The use of a web developing interface which is Streamlit enables cross-platform utilisation without having to install locally, thus increasing convenience for the users.

4.3 Core Logic Engine Design

4.3.1 Formula Representation and Parsing (UML Classes)

The first submodule of the Core Logic Engine is the formula representation and parsing component whereby inputs entered by the user are translated into usable formulas by the system. The UML Classes should illustrate the following classes:

- **Formula:** This is an abstract base class for a logical formula which can be an instance of a certain kind of formula, or be composed from sub formulas.

- **Atom:** A proposition of such kind may be considered to represent a basic proposition.
- **Not, And, Or, Implies, Box, Diamond:** Subclasses of Formulas for every logical operation possible.

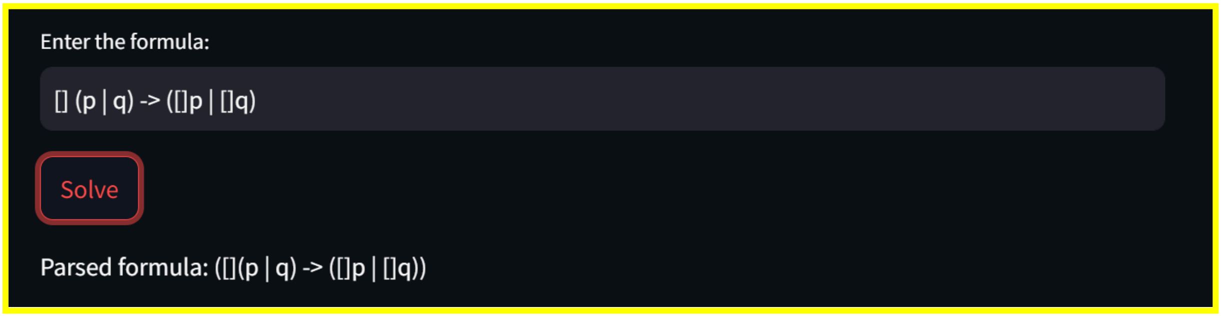
Manipulations:

- **Formula Parser**: Converts input strings into instances of the above classes.
- **Tableaux Solver**: Uses these instances to perform logical analysis.

The UML Classes guarantee that every formula type is valid and generally can easily interface with other components.

4.3.2 Formula Parser Design

This section shows the parsed formula working in the application. It will display the parsed formula as soon as the user hits the solve button after entering the formula to check the solutions. The *Fig 4.2* below shows how the formula is appearing in the display.



Parsed formula: ([](p | q) -> ([]p | []q))

Fig 4.2 Display format of the Formula Parser

4.3.2 Tableaux Algorithm Design

Tableaux Algorithm is core to the solver which is responsible for expanding as well as analysing logical formulas. The flowchart for the tableaux algorithm is as follows:

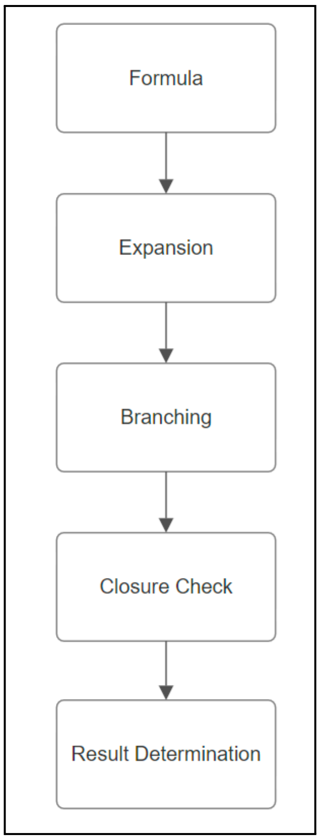


Fig 4.3 Tableaux Solver Algorithm

4.3.3 Kripke Model Integration

Kripke Model Generator integrates with the tableaux algorithm to create visual models of possible worlds and their accessibility relations:

- **World Creation:** Define and manage possible worlds
- **Accessibility Relations:** Express and represent relations between worlds.
- **Valuation Assignment:** Find out the truth values of propositions in each world.

This integration enables the viewing of modal logic formulas and enhances the functional value of the solver.

4.4 User Interface

4.4.1 Web Application Layout

The Web Application Layout wireframe should include:

- **Formula Input:** A text box in which users type in modal logic formulas.
- **Result Display:** It appears there is an area displaying validity/satisfiability as well as some or any visualisations.
- **History Management:** A feature that allows the user to easily access the previous successfully solved formulas using a sidebar or a menu.

The advantage of wireframe is that it helps make a layout that avails the physical interactions between the user and the product.

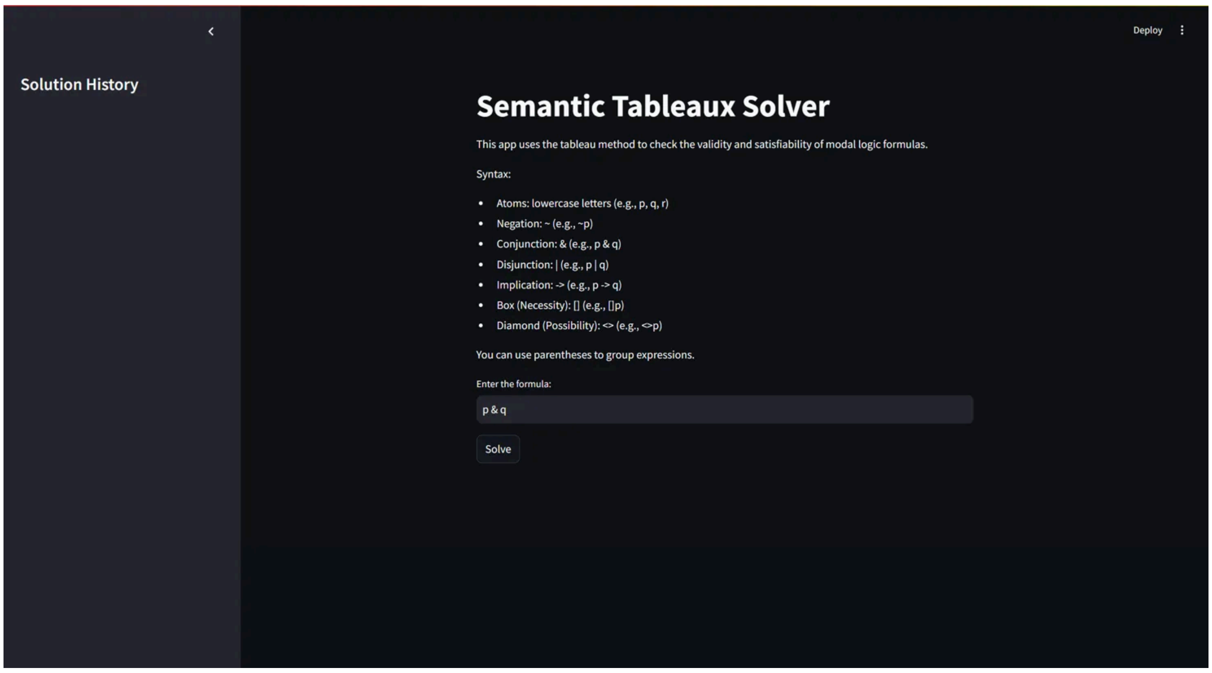
4.4.2 User Interaction Flow

The User Interaction Flow sequence shows:

1. **User Input:** The user types in a formula, and then sends it in.
2. **Processing:** The Web Application Interface passes it to the Core Logic Engine which processes it.
3. **Evaluation:** So, the Core Logic Engine solves the formula and produces outcomes.
4. **Visualisation:** The resulting and visualisations are given to the Visualization Component.
5. **Output:** The Web Application Interface is the interface used to convey the results and the visualisations to the user.

It is easier to show on these steps how the flow of interactions and data takes place within a system.

4.4.2 User Interface Design



Semantic Tableaux Solver

This app uses the tableau method to check the validity and satisfiability of modal logic formulas.

Syntax:

- Atoms: lowercase letters (e.g., p, q, r)
- Negation: ~ (e.g., ~p)
- Conjunction: & (e.g., p & q)
- Disjunction: | (e.g., p | q)
- Implication: -> (e.g., p -> q)
- Box (Necessity): [] (e.g., []p)
- Diamond (Possibility): <> (e.g., <>p)

You can use parentheses to group expressions.

Enter the formula:

p & q

Solve

Fig 4.4 User Interface

The UI is designed with elements of the dark theme and plain white fonts, thus maintaining a contemporary look. First, there is the full title of the app, then there's a short description that states that the app validates the formulas in the modal logic using the tableau method and determines if the formulas are satisfiable. The area in the middle of the interface outlines the syntax of logical operators such as atoms, negation, conjunction and disjunction as well as implication and modal operators, the box and the diamond respectively. Below this there is an input area in which users can type in their formula, the following example being "p & q". Beneath the input field, it contains a "Solve" button. The navigation elements are therefore a back arrow and the 'Solution History' at the top left of the page, which indicates that past solutions can be reviewed.

4.4.3 Tree and Kripke World Models Design

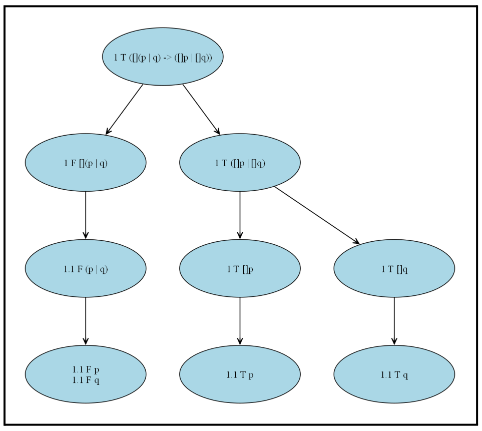


Fig 4.5 Tableaux Tree Expansion

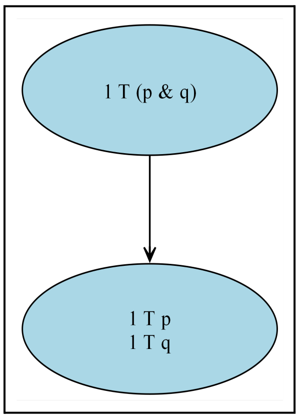


Fig 4.6 Tableaux Tree Expansion

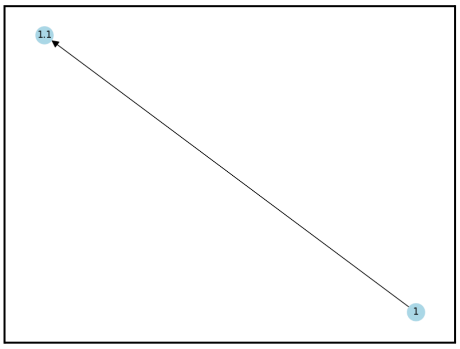


Fig 4.7 Kripe World Accessibility Model

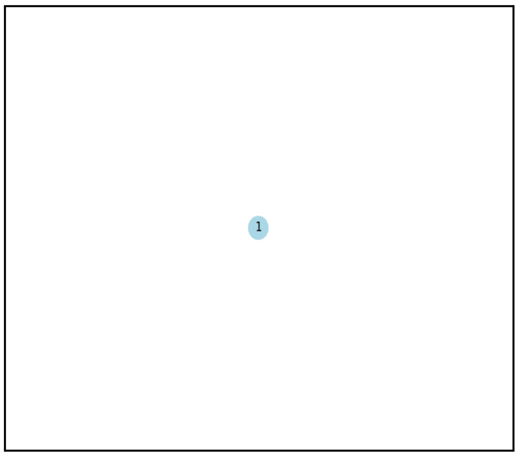


Fig 4.8 Kripe World Accessibility Model

The tree may seem weirdly sized because of the minimum sized generation for a clear tree diagram. This means that if the tree has a broad and long branching, the image contents will be smaller and might also cross through the screen and we need to open and view the entire diagram using the given enlargement option. Similarly, if the tree is very short, the branches generated are huge in the image, and we still have to open the image and view it fully.

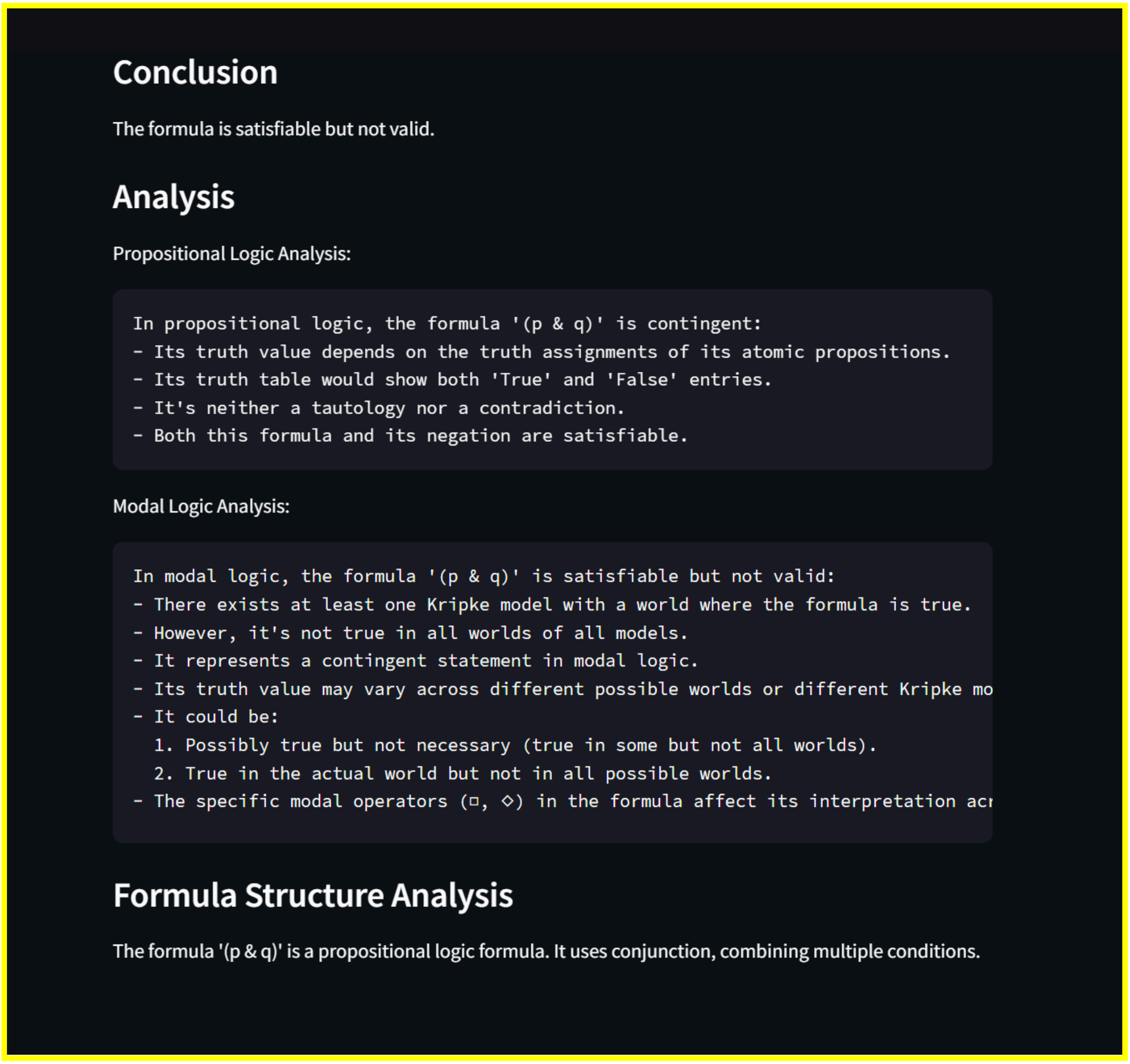
4.4.4 Analysis & Conclusion Design

The design displays the parts ***Conclusion, Analysis, & Formula Structure Analysis*** in a uniform manner. It can be very helpful for beginners to understand the concepts behind the solutions and be able to do the research studies efficiently. The *Fig 4.9* demonstrates the design of this section.

- First, in conclusion, the application can display a few sets of possible outcomes like "Satisfiable and Valid", "Not Satisfiable and Not Valid", "Satisfiable and Not Valid."

- Second, in Analysis, each formula will be analysed in both Propositional and Modal Logic.

- Lastly, in Formula Structure Analysis, the application will determine what type of formula it is and what its contents are.



Conclusion

The formula is satisfiable but not valid.

Analysis

Propositional Logic Analysis:

```
In propositional logic, the formula '(p & q)' is contingent:
- Its truth value depends on the truth assignments of its atomic propositions.
- Its truth table would show both 'True' and 'False' entries.
- It's neither a tautology nor a contradiction.
- Both this formula and its negation are satisfiable.
```

Modal Logic Analysis:

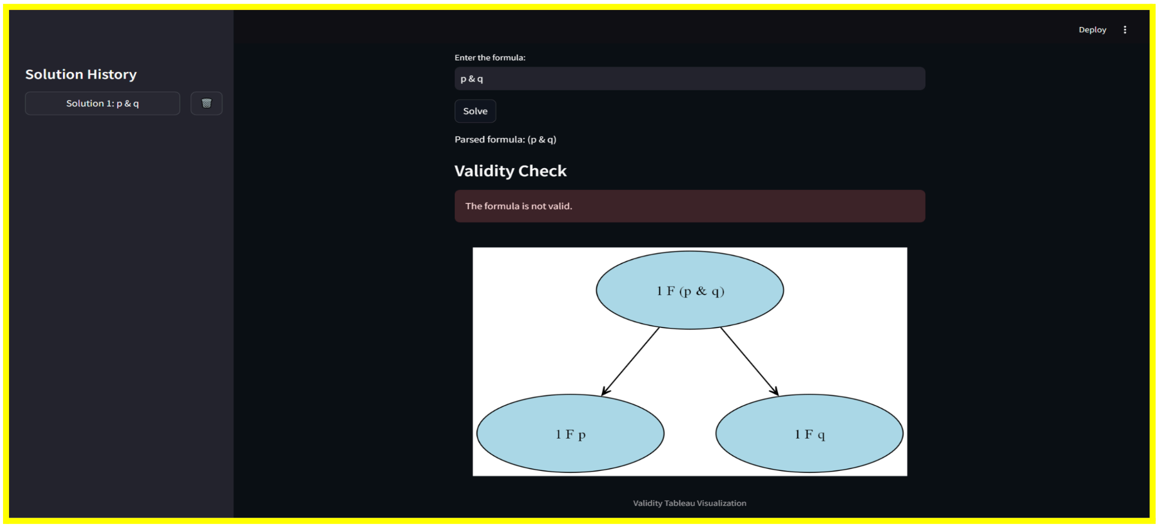
```
In modal logic, the formula '(p & q)' is satisfiable but not valid:
- There exists at least one Kripke model with a world where the formula is true.
- However, it's not true in all worlds of all models.
- It represents a contingent statement in modal logic.
- Its truth value may vary across different possible worlds or different Kripke mo
- It could be:
  1. Possibly true but not necessary (true in some but not all worlds).
  2. True in the actual world but not in all possible worlds.
- The specific modal operators (□, ◇) in the formula affect its interpretation acr
```

Formula Structure Analysis

The formula '(p & q)' is a propositional logic formula. It uses conjunction, combining multiple conditions.

Fig 4.9 Conclusion and Analysis Page Design

4.4.5 Results / Output



Solution History

Solution 1: p & q

Enter the formula:

p & q

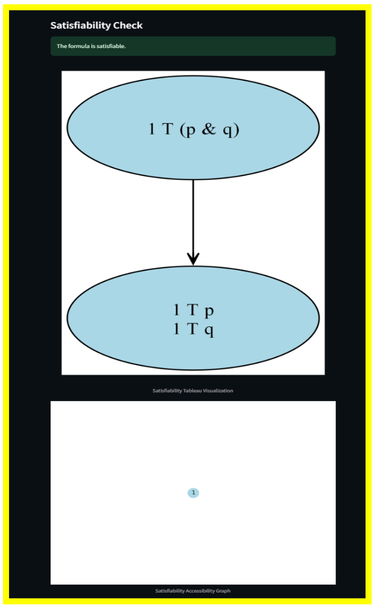
Solve

Parsed formula: (p & q)

Validity Check

The formula is not valid.

Validity Tableau Visualization

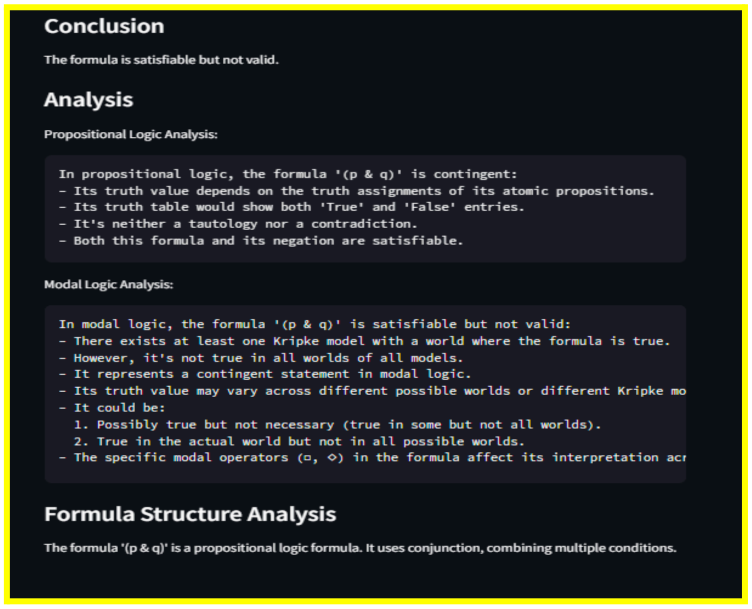


Satisfiability Check

The formula is satisfiable.

Satisfiability Tableau Visualization

Satisfiability Accessibility Graph



Conclusion

The formula is satisfiable but not valid.

Analysis

Propositional Logic Analysis:

```
In propositional logic, the formula '(p & q)' is contingent:
- Its truth value depends on the truth assignments of its atomic propositions.
- Its truth table would show both 'True' and 'False' entries.
- It's neither a tautology nor a contradiction.
- Both this formula and its negation are satisfiable.
```

Modal Logic Analysis:

```
In modal logic, the formula '(p & q)' is satisfiable but not valid:
- There exists at least one Kripke model with a world where the formula is true.
- However, it's not true in all worlds of all models.
- It represents a contingent statement in modal logic.
- Its truth value may vary across different possible worlds or different Kripke mo
- It could be:
  1. Possibly true but not necessary (true in some but not all worlds).
  2. True in the actual world but not in all possible worlds.
- The specific modal operators (□, ◇) in the formula affect its interpretation acr
```

Formula Structure Analysis

The formula '(p & q)' is a propositional logic formula. It uses conjunction, combining multiple conditions.

Fig 4.10 Output

The *Fig 4.10* shows the detailed output of the application. The application first checks the Validity of the formula and generates the Tableaux Tree using the rules for propositional and modal logics. The size of the tree figures might vary according to the size of the tree. Furthermore, Kripke World Accessibility Models are made to visualise all accessible worlds from another world. Then the application generates the Conclusion, Result, & Analysis sections to give a brief overview of the formula in both Propositional Logic and Modal Logic.

4.5 Use Case & Data Flow Diagram

4.5.1 Data Flow Diagram

A **Use Case Diagram** can be defined as a description of how the various users of a website will perform a specific activity. It defines the approach that a system has of reacting to a demand as seen by the user. A use case diagram depicts the possible scenarios of communication between the user and the system. A use case diagram captures multiple users and use cases of a system, use case is the smallest unit of functionality delivery. Use cases are oval shaped depending on their size, while some of them are circles. The performers are normally drawn in sticks or other bare minimum representations.

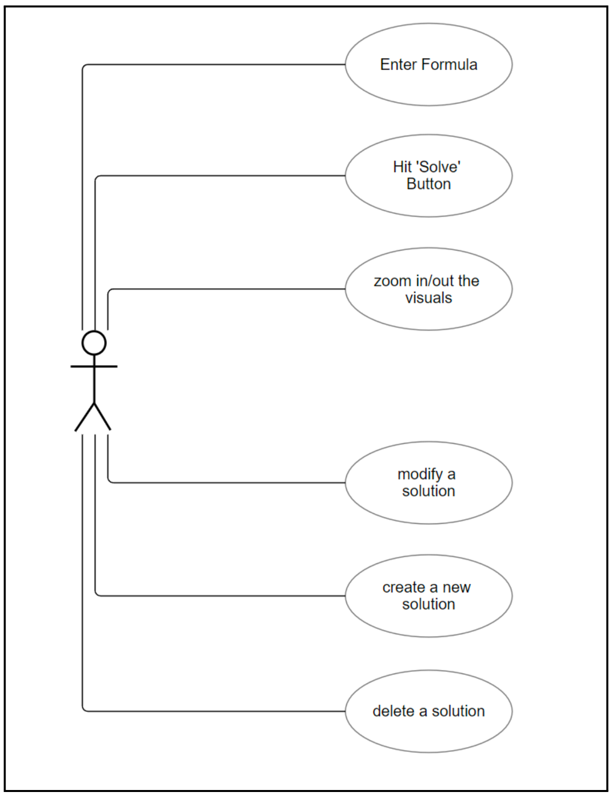


Fig 4.11 User Use-case Diagram

4.5.2 Data Flow Diagram

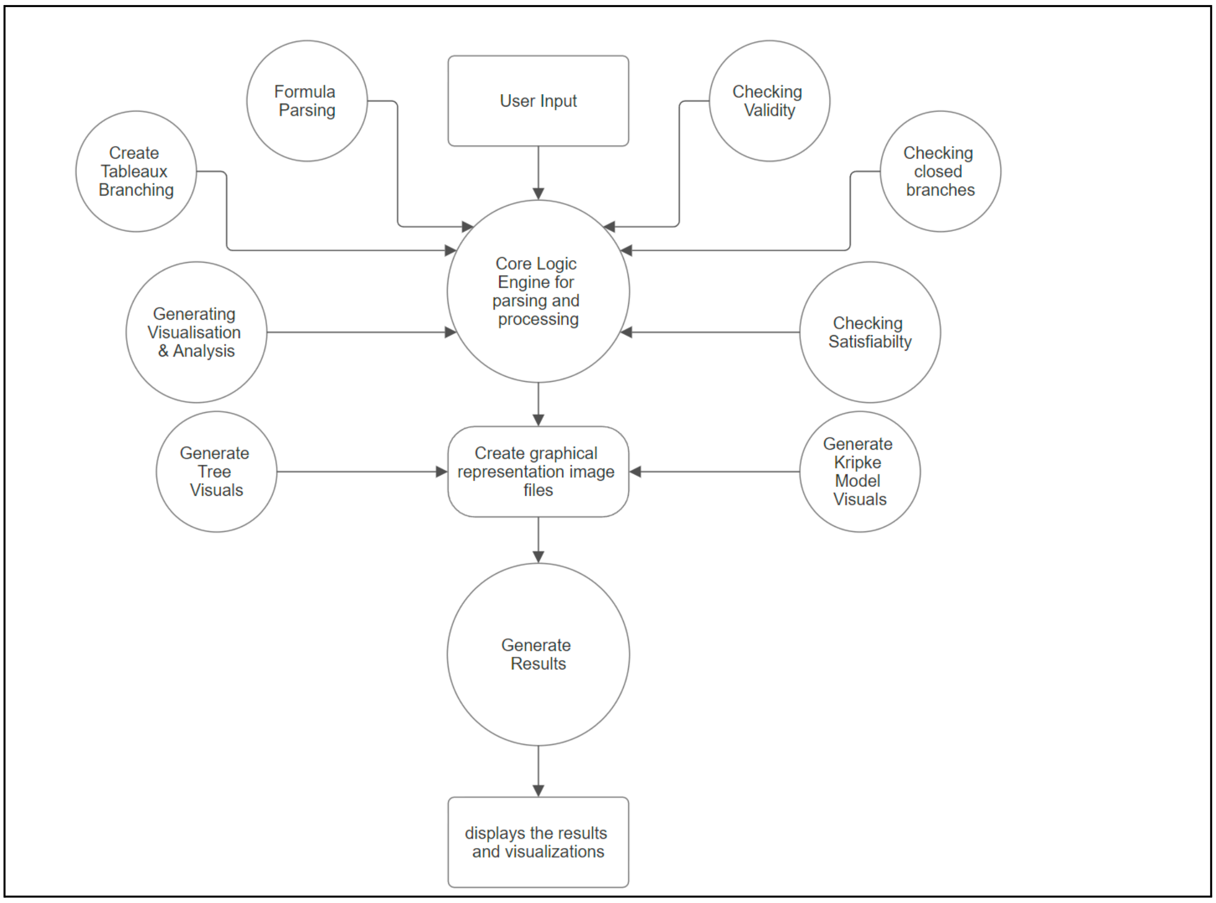

Formula Parsing
User Input
Checking Validity
Create Tableaux Branching
Checking closed branches
Core Logic Engine for parsing and processing
Generating Visualisation & Analysis
Checking Satisfiabilty
Generate Tree Visuals
Create graphical representation image files
Generate Kripke Model Visuals
Generate Results
displays the results and visualizations

Fig 4.12 Data Flow Diagram

The Data Flow Diagram (DFD) outlines:

- **User Input:** It was accessed through the Web Application Interface.
- **Processing:** Sent to the Core Logic Engine for evaluation.
- **Result Generation:** Calculations and data visualisations are performed and the outcomes are delivered to the Visualization Component.
- **User Output:** Presents output back in the Web Application Interface in form of results and visualisations.

4.5.2 Error Detection and Management

- **Syntax Errors:** Identified during formula parsing with real-time feedback.
- **Logical Errors:** Handled during tableaux expansion with informative error messages.
- **Timeouts:** Implemented to handle long computations, notifying users if processing exceeds the set time limit.

4.6 Design Evaluation

4.6.1 Alignment with Project Requirements

It is evident that the design of the project enables the achievement of functionality, efficiency and user-friendliness in the project. It incorporates the requirements into a single system architecture as elaborated in the analysis chapter.

4.6.2 Key Trade-offs and Their Justifications

- **Clarity vs. Flexibility:** Concerned with how the system is presented to end-users while at the same time keeping a flexible system to support other features to be added in the future.
- **Server-Side vs. Client-Side Rendering:** Balancing of server load with that of client side is done with the aim to satisfy the performance and usability parameters.

4.6.3 Preparation for Implementation Phase

The design is prepared for application, containing the simplified structure of the network together with the corresponding detailed plans of components integration, as well as the ways of increasing the reliability and performance of the system.

Chapter 5

Implementation and Testing

5.1 Implementation

The main aim of the implementation stage is to exercise the design in such a manner that it turns into reality. It is evolutionary in nature, that is, the project is created out of the whole cloth starting with the basics and expanding the field little by little. The two parts are as follows and are done separately each according to the modular design principles detailed in the last chapter.

5.1.1 Core Logic Engine

Formula Parser:
The formula parser converts user inputs into a structured format that the system can process. It parses strings into logical operators and formula types like *Atom, Not, And, Or, Implies, Box,* and *Diamond*.

```
@dataclass
class Atom(Formula):
    name: str

@dataclass
class Not(Formula):
    formula: Formula

@dataclass
class And(Formula):
    conjuncts: List[Formula]

@dataclass
class Or(Formula):
    disjuncts: List[Formula]

@dataclass
class Implies(Formula):
    left: Formula
    right: Formula

@dataclass
```

```
class Box(Formula):
    formula: Formula

@dataclass
class Diamond(Formula):
    formula: Formula
```

Explanation:
In this section, the `formula` classes which characterise the logical expressions in the above identified system are defined. For instance:

- *Atom* denotes single propositional variables. However it's important to note that the semantics of the propositional formulas that are defined by the authors are not the standard possible world semantics but are instead atoms.
- The symbol *Not* abolishes a given formula and *And, Or,* and *Implies* mean conjunction, disjunction, and implication.
- *Box* and *Diamond* are symbols of modal logic operators that are necessity and possibility, respectively.

Parsing Formula:

```
def custom_parse_formula(s: str) -> Formula:
   s = s.replace(' ', '')  # Remove all spaces
   s = s.replace('□', '[]')  # Replace □ with []
   s = s.replace('♢', '<>')  # Replace ♢ with <>


   def parse_atom(i):
       if i < len(s) and s[i].isalpha():
           return Atom(s[i]), i + 1
       raise ValueError(f"Expected atom at position {i}")


   def parse_not(i):
       if i < len(s) and s[i] == '~':
           sub_formula, new_i = parse_modal(i + 1)
           return Not(sub_formula), new_i
       return parse_modal(i)


   def parse_modal(i):
       if i < len(s) - 1:
           if s[i:i + 2] == '[]':
               sub_formula, new_i = parse_not(i + 2)
```

```
                return Box(sub_formula), new_i
            elif s[i:i + 2] == '<>':
                sub_formula, new_i = parse_not(i + 2)
                return Diamond(sub_formula), new_i
        return parse_parentheses(i)


    def parse_or(i):
        left, i = parse_not(i)
        while i < len(s) and s[i] == '|':
            right, i = parse_not(i + 1)
            left = Or([left, right])
        return left, i



def parse_and(i):
        left, i = parse_or(i)
        while i < len(s) and s[i] == '&':
            right, i = parse_or(i + 1)
            left = And([left, right])
        return left, i


    def parse_implies(i):
        left, i = parse_and(i)
        if i < len(s) - 1 and s[i:i + 2] == '->':
            right, i = parse_implies(i + 2)  # Recursive call for right
side
            return Implies(left, right), i
        return left, i


    def parse_parentheses(i):
        if i < len(s) and s[i] == '(':
            expr, i = parse_implies(i + 1)
            if i < len(s) and s[i] == ')':
                return expr, i + 1
            raise ValueError(f"Missing closing parenthesis at position
{i}")
        return parse_atom(i)


    formula, i = parse_implies(0)
    if i < len(s):
        raise ValueError(f"Unexpected character at position {i}:
```

```
'{s[i]}'")
   return formula
```

Explanation:

The *`custom_parse_formula()`* function takes a string and recursively breaks it into logical components like atoms, modal operators, and logical connectives (~, &, |, etc.). It replaces modal operators like □ and ♢ with their alternative text representations ([], <>). Apparently, in this snippet, we are also maintaining the exception and error handling accordingly with appropriate console messages.

Tableaux Solver:

The *`Tableaux`* class implements the semantic tableaux method to check the validity and satisfiability of a given formula.

```
#Sample structure of the usage of Tableaux class
class Tableaux:
    def __init__(self, formula: Formula):
        self.formula = formula
        self.branches = [(False, "1", self.formula)]  # Start with the
root node
        self.accessibility = defaultdict(set)

    def check_validity(self) -> bool:
        self.branches = [[(False, "1", self.formula)]]
        return self.solve()


    def check_satisfiability(self, max_iterations=1000):
        self.branches = [[(True, "1", self.formula)]]
        return not self.solve(max_iterations)


    def solve(self):
        for branch in self.branches:
            expanded_branches = self.expand_branch(branch)
            self.branches.extend(expanded_branches)

        all_closed = all(self.is_closed(branch) for branch in
self.branches)
        return all_closed
```

Explanation:

The `Tableaux` class performs the expansion of logical formulas in a tree-like structure, applying the semantic tableaux method. Key methods:

- *`check_validity()`*: Begins the expansion of the formula with a "False" root, checking whether all branches close, indicating the formula's validity.
- *`solve()`*: Expands each branch of the tableaux tree, checking for contradictions (closed branches) and determining if the formula is valid.
- *`check_satisfiablity()`*: Begins the expansion of the formula with a "True" root, checking whether all branches close or open, indicating the formula's Satisfiability.

Kripke Model Generator: The system uses Kripke models to visualise possible world semantics, handling accessibility between worlds.

```
def visualize_accessibility(self):

    """Visualises the Kripke model accessibility relations."""

    graph = nx.DiGraph()

    # Adding nodes and edges based on accessibility relations

    for world in self.accessibility:
        graph.add_node(world)
        for accessible_world in self.accessibility[world]:
            graph.add_edge(world, accessible_world)

    pos = nx.shell_layout(graph)
    plt.figure(figsize=(12, 8))
    nx.draw_networkx(graph, pos, node_size=2000, node_color='light
blue')
    plt.title("Kripke Model Accessibility")
    plt.show()
```

Explanation:

This function uses the `networkx` library to build a directed graph of what in the Kripke model's World Accessibility Relations. It visually illustrates all the worlds and how they are related depending on the accessibility which makes users understand modal logic formulas easily.

Tableaux Expansion

The `expand_branch` and `apply_rule` methods are key components in the `Tableaux` class. They handle the expansion of logical branches and the application of the appropriate logic rules during the tableaux solving process.

The `expand_branch` method is responsible for expanding a branch of the tableaux tree. It recursively applies logical rules to each formula in the branch and creates new branches based on the results.

```
def expand_branch(self, branch, depth=0, parent_id=None):
    print(f"\nExpanding branch at depth {depth}: {[(prefix, 'T' if sign
else 'F', Tableaux.formula_to_string(formula)) for sign, prefix, formula
in branch]}")

    if depth > 100:  # Prevent infinite recursion
        print(f"Warning: Maximum recursion depth reached for branch:
{branch}")
        return [branch]

    node_id = self.add_node(branch, parent_id)

    # Expand all F [] (box) & T <> (diamond) formulas first

    f_box_formulas = [(i, (sign, prefix, formula)) for i, (sign, prefix,
formula) in enumerate(branch)
                      if isinstance(formula, Box) and not sign]

    for i, (sign, prefix, formula) in f_box_formulas:
        result = self.apply_rule(sign, prefix, formula)
        if result:
            for new_branch in result:
                new_full_branch = branch[:i] + new_branch + branch[i +
1:]
                expanded_branches = self.expand_branch(new_full_branch,
depth + 1, node_id)
                if expanded_branches:
                    return expanded_branches

    # Expand other formulas if not already expanded

    for i, (sign, prefix, formula) in enumerate(branch):
        if isinstance(formula, Atom):
            continue  # Skip atomic formulas

        result = self.apply_rule(sign, prefix, formula)
        if result:
            expanded_branches = []
            for new_branch in result:
                new_full_branch = branch[:i] + new_branch + branch[i +
1:]

expanded_branches.extend(self.expand_branch(new_full_branch, depth + 1,
node_id))
            return expanded_branches

    # Return branch if no more expansion possible

    print(f"No expansion possible for branch, returning as is:
{[(prefix, 'T' if sign else 'F', Tableaux.formula_to_string(formula))
for sign, prefix, formula in branch]}")
    return [branch]
```

Explanation:

- **Recursive Expansion:** The `expand_branch` method function calls to expand the next level of the tableaux tree and applies the logical rules of the forms of the given branch.
- **Depth Control:** To avoid a recursive depth, it has implemented a depth counter that restricts the expandable depth to 100 only (although this can be changed according to the system requirements).
- **Handling Modal Operators:** Using the French notation the method first processes "F []" (falsity of necessity operators) and "T <>" (truth of possibility operators), in particular modal logic operators such as Box ([]) and Diamond (<>). It always insists on prioritising modal logic branches before other branches.
- **Formula Expansion:** If it is not an atomic proposition which means that the formula is conjunctions, disjunctions or implications etc. , it uses the appropriate rule to develop the formula as seen in the `apply_rule` method above.

The recursive characteristic of `expand_branch` enables the tableaux to expand in as far as is possible applying the right logical expansion on every branch until the branch finds a solution or there is no any means of expanding the branch further.

The `apply_rule()` method applies the appropriate logical rule based on the type of formula being processed and its truth value (`T` for true, `F` for false).

```
def apply_rule(self, sign: bool, prefix: str, formula: Formula) ->
List[List[Tuple[bool, str, Formula]]]:
    print(f"Applying rule to: {'T' if sign else 'F'} {prefix}
{self.formula_to_string(formula)}")

    # Atomic formulas don't require expansion
    if isinstance(formula, Atom):
        return []

    # Negation: "T ~A" becomes "F A" and "F ~A" becomes "T A"
    elif isinstance(formula, Not):
        return [[(not sign, prefix, formula.formula)]]

    # Conjunction: "T A & B" becomes "T A, T B" and "F A & B" becomes "F
A | F B"
    elif isinstance(formula, And):
        if sign:
            return [[(True, prefix, conjunct) for conjunct in
formula.conjuncts]]
        else:
            return [[(False, prefix, conjunct)] for conjunct in
formula.conjuncts]

    # Disjunction: "T A | B" becomes "T A | T B" and "F A | B" becomes
"F A, F B"
    elif isinstance(formula, Or):
        if sign:
            return [[(True, prefix, disjunct)] for disjunct in
formula.disjuncts]
        else:
            return [[(False, prefix, disjunct) for disjunct in
formula.disjuncts]]

    # Implication: "T A -> B" becomes "F A, T B" and "F A -> B" becomes
"T A, F B"
    elif isinstance(formula, Implies):
        if sign:
            return [[(False, prefix, formula.left)], [(True, prefix,
formula.right)]]
        else:
            return [[(True, prefix, formula.left), (False, prefix,
formula.right)]]

    # Necessity (Box): "T []A" requires that "T A" in all accessible
worlds, "F []A" creates a new world
    elif isinstance(formula, Box):
        if sign:
            return [[(True, accessible_world, formula.formula)] for 
accessible_world in self.accessibility[prefix]]
        else:
            new_world = f"{prefix}.{len(self.accessibility[prefix]) + 
1}"
            self.accessibility[prefix].add(new_world)
            return [[(False, new_world, formula.formula)]]

    # Possibility (Diamond): "T <>A" creates a new world with "T A", "F 
<>A" applies "F A" in all accessible worlds
    elif isinstance(formula, Diamond):
        if sign:
            new_world = f"{prefix}.{len(self.accessibility[prefix]) + 
1}"
            self.accessibility[prefix].add(new_world)
            return [[(True, new_world, formula.formula)]]
        else:
            return [[(False, accessible_world, formula.formula)] for 
accessible_world in self.accessibility[prefix]]

    return []
```

Explanation:

The method called `apply_rule()` specifies how a formula has to be expanded depending on the type of the formula – *Not, And, Or, Implies, Box, Diamond* – as well as the current value – `T` for true, `F` for false.

Applications:

- **Negation (Not):** If A is the final symbol of a term in the formula of the form ***"T ~A"*** we get its negation ***"F A"***. For instance, where ***"T ~p"*** exists, the result is ***"F p"***.
- **Conjunction (And):** A formula ***"T A & B"*** is expanded to ***"T A, T B"***, meaning both parts must be true. If false, only one part needs to be false.
- **Disjunction (Or):** A disjunction***"T A | B"*** will be interpreted as ***"T A or T B"*** If false then both the parts are false.
- **Implication (Implies):** The assertion ***"T A -> B"*** is defined as **"F A, T B"** this means that for the implication to be true, if A is false then B must be true.
- **Modal Operators (Box and Diamond):** The Box operator replaces ***'T []A'*** in all the reachable states with ***'T A'*** and ***'F []A'*** causing a new state to be created. The Diamond operator operates in a manner as the Equality operator, but it is associated with all of the ***"T <> A"*** and ***"F <> A"*** cases.

All of the chosen logical formulae are developed with the help of the proper rule according to the logic status and the operator, which create the structure of the tableaux solver that complies with the rules of both modal and propositional logic.

5.1.2 Web Application Interface

Streamlit Framework Library was used to implement the Web Application Interface, which provides an intuitive interface for users to interact with the solver.

```
def main():
    st.title("Semantic Tableaux Solver")

    formula_str = st.text_input("Enter the formula:")

    if st.button("Solve"):
        is_valid, is_satisfiable, parsed_formula = 
solve_formula(formula_str)
        st.write(f"Parsed formula: {parsed_formula}")

        if is_valid:
            st.success("The formula is valid.")
        else:
            st.error("The formula is not valid.")

        if is_satisfiable:
            st.success("The formula is satisfiable.")
        else:
            st.error("The formula is not satisfiable.")
```

Explanation:

This Streamlit app provides the user interface for the semantic tableaux solver. Users enter a formula, and after pressing the "Solve" button, the app displays the parsed formula, as well as its validity and satisfiability results.

5.2 Testing

5.2.1 Unit Testing

The following test cases cover different aspects of modal logic formulas in **K** modal logic, testing their validity and satisfiability.

Test 1: Valid Formulas in K Modal Logic

This test ensures that valid formulas in **K modal logic** are correctly identified. The formulas include both tautologies and logical axioms.

```
def test_valid_in_K(self):
    valid_formulas = [
        "(p | ~p)",  # Propositional tautology
        "[](p -> q) -> ([]p -> []q)",  # K axiom
    ]
    for formula in valid_formulas:
        with self.subTest(formula=formula):
            parsed_formula = custom_parse_formula(formula)
            solver = Tableaux(parsed_formula)
            self.assertTrue(solver.check_validity(), f"Formula should be 
valid in K: {formula}")
            self.assertTrue(solver.check_satisfiability(), f"Formula 
should be satisfiable: {formula}")
```

Explanation:

- **Purpose:** This test demonstrates if `check_validity()` and `check_satisfiability()` return Information that can identify when formulas are valid and satisfiable.
- **Test Cases:** It is an extension of propositional tautology and consists of a specific K modal logic axiom. In each of the formulas we use the name of the solver to check the formula by also making sure that it is both valid and satisfiable.

Test 2: Satisfiable but Not Valid Formulas

Some formulas are satisfiable but not necessarily valid in K modal logic. This test checks that such formulas are correctly identified.

```python
def test_satisfiable_but_not_valid_in_K(self):
    satisfiable_formulas = [
        "<>(p | ~p)",  # Possibility of tautology
        "[]p -> p",  # T axiom, satisfiable but not valid in K
        "<>[]p -> []p",  # Not valid in K
    ]
    for formula in satisfiable_formulas:
        with self.subTest(formula=formula):
            parsed_formula = custom_parse_formula(formula)
            solver = Tableaux(parsed_formula)
            self.assertFalse(solver.check_validity(), f"Formula should
not be valid in K: {formula}")
            self.assertTrue(solver.check_satisfiability(), f"Formula
should be satisfiable: {formula}")
```

Explanation:

- **Purpose:** This test is concerned with the fact that there are formulas that are satisfiable but not K valid.
- **Test Cases:** Includes formulas like the **T axiom** and related expressions. The solver should confirm that they are satisfiable but not valid in **K**.

Test 3: Unsatisfiable Formulas

This test ensures that contradictory or logically unsatisfiable formulas are properly identified.

```python
def test_unsatisfiable_formulas(self):
    unsatisfiable_formulas = [
        "p & ~p",  # Direct contradiction
        "<>(p & ~p)",  # Possibility of a contradiction
    ]
    for formula in unsatisfiable_formulas:
        with self.subTest(formula=formula):
            parsed_formula = custom_parse_formula(formula)
            solver = Tableaux(parsed_formula)
            self.assertFalse(solver.check_validity(), f"Formula should
not be valid: {formula}")
            self.assertFalse(solver.check_satisfiability(), f"Formula
should not be satisfiable: {formula}")
```

Explanation:

- **Purpose:** Makes sure that the system detects the blocked formulas, whether it is a simple contradiction or a complex modal contradiction.
- **Test Cases:** They encompass tautologies such as ***p & ~p*** and its modal counterparts. The solver should report these as both being invalid solutions and being unsolvable.

Test 4: Complex Formulas in K Modal Logic

This test evaluates how well the system handles more complex modal logic formulas involving multiple modalities and operators.

```
def test_complex_formulas_in_K(self):
    complex_formulas = [
        "[]p -> [](p | q)",  # Necessity of disjunction
        "<>(p & q) -> (<>p & <>q)",  # Distributivity in possibility
    ]
    for formula in complex_formulas:
        with self.subTest(formula=formula):
            parsed_formula = custom_parse_formula(formula)
            solver = Tableaux(parsed_formula)
            is_valid = solver.check_validity()
            is_satisfiable = solver.check_satisfiability()
            self.assertIsNotNone(is_valid, f"Validity check should not
fail for: {formula}")
            self.assertIsNotNone(is_satisfiable, f"Satisfiability check
should not fail for: {formula}")
            if is_valid:
                self.assertTrue(is_satisfiable, f"Valid formula should
also be satisfiable: {formula}")
```

Explanation:

- **Purpose:** This test seeks to check the ability of the solver in regard to complex and nested formulas in K modal logic.
- **Test Cases:** In this range it is possible to include formulas involving multiple modal operators as well as logical connectives. The solver should be able to review the formulas as being either of validity and satisfiability.

Test 5: Nested Modalities

This test focuses on formulas with deeply nested modal operators to ensure the solver can handle complex modal logic structures.

```
def test_nested_modalities_in_K(self):
    nested_formulas = [
        "[]<>[]p -> <>[](p | q)",
        "[]<>(p & q) -> (<>[]p & <>[]q)",
    ]
    for formula in nested_formulas:
        with self.subTest(formula=formula):
            parsed_formula = custom_parse_formula(formula)
            solver = Tableaux(parsed_formula)
            is_valid = solver.check_validity()
            is_satisfiable = solver.check_satisfiability()
            self.assertIsNotNone(is_valid, f"Validity check should not
fail for: {formula}")
            self.assertIsNotNone(is_satisfiable, f"Satisfiability check
should not fail for: {formula}")
            if is_valid:
                self.assertTrue(is_satisfiable, f"Valid formula should
also be satisfiable: {formula}")
```

Explanation:

- **Purpose:** The predictor verifies the capability of the solver to solve formulas that have one form of modality enclosed within another one.
- **Test Cases:** These consist of some examples of the nested necessity and possibility operators. The solver must be able to provide the right solutions concerning validity and satisfiability for such formulas.

5.2.2 Integration Testing

System tests were conducted for verifying the harmony of the components such as the Formula Parser, Tableaux Solver and the Kripke Model Generator. Specific emphasis was made on testing the functionality at all levels beginning with the ability of the software to parse any formula and expand it, validate and determine if the formula is satisfiable without encountering an error.

5.2.3 Test Results

1. Test for Valid Formulas in K

(p | ~p)	Passed
<>(p & ~p)	Passed

Table 5.1 Test for Valid Formulas in K

2. Test for Satisfiable but Not Valid Formulas in K

<>(p | ~p)	Passed
[]p -> p	Passed
<>[]p -> []p	Passed

Table 5.2 Test for Satisfiable but Not Valid Formulas in K

3. Test for Unsatisfiable Formulas in K

p & ~p	Passed
<>(p & ~p)	Passed

Table 5.3 Test for Unsatisfiable Formulas in K

4. Test for Complex Satisfiable and Valid Formulas in K

[]p -> [](p | q)	Passed
<>(p & q) -> (<>p & <>q)	Passed

Table 5.4 Test for Complex Satisfiable and Valid Formulas in K

5. Test for nested modalities in K

[]<>[]p -> <>[](p | q)	Passed
[]<>(p & q) -> (<>[]p & <>[]q)	Passed

Table 5.5 Test for Nested Modalities in K

5.2.3 Failed Test Analysis

Throughout the analysis during the tests, there were potential issues occurring during the branching of modal logics. According to the rule, a **T[]** and **F<>** cannot be applied until and unless there has been a new world created either by **F[]** or **T<>**. Following up with this rule, some branching failed to defer the **F <>** or **T []** applications while the applicable rules are in the process. Due to this issue, some outcomes are being concluded incorrect. Example of the failed test case branching is available in Appendices *Figure 3*. According to the algorithm, it is not checking the propositions that are passed and not following in the steps further. That could be a potential problem to fix in the future enhancements. Mostly these cases are seen in the case of Diamond (<>) operators.

5.3 Conclusion

The **Implementation** of the semantic tableaux solver effectively fulfilled the intended goal of transforming the design into an actual working system. In this way, the implementation was done in a modular manner where the key parts such as **Formula Parser, Tableaux Solver, Kripke Model Generator, Web Application Interface** and **Visualization Component** were realised. The simplicity and interactivity of Python combined with sites such as Streamlit, NetworkX, Pydot, and Matplotlib made navigation smooth and real-time, entry of simple formulas possible, and the natural visualisation of logical schemas.

At the **Testing** phase, the system was tested by applying unit tests and integration tests. All basic features of the application were covered in unit tests, particularly parsing of formulas, generation of tableaux and Kripke models. The tests checked the validity of the tool's response to logically valid, as well as invalid formulas and bounded, as well as unbounded solvers by checking the validity of satisfiable and unsatisfiable formulas also we checked the performance of tool on complex and nested modal logic formulas. Systems tests exposed the proper integration of each module in a given system and also their reliability.

Overall, the project was successful in creating the semantic tableaux solver free of fatal errors and ready for the scale of processing of knowledge bases with the help of the adherence to the best practices in error handling, modular design, and testing. The system is able to process a very large number of formulas in modal logic and in each case give the user educational and research value through the visualisation and the GUI. Thus, implementation, and testing successfully leaves a good ground for future improvement and addition in the system.

Chapter 6

Results and Discussion

6.1 Findings

The following observations were made as a result of the development of the semantic tableaux solver for modal logic. First, the system was able to accurately capture and perform logical computations on many formulations of modal logic; from simple propositional logic to more complex nested modal operators. The Tableaux Solver helped us in expanding out the given logical formulas in a tree like structure which threw light on the logical paths and contradiction which led us in the process of identifying the validity and satisfiability of the formulas.

Key findings include:

- **Formula Parsing:** As a matter of fact, utilising Quant Tools own custom formula parser, it was demonstrated that even basic and advanced equations were correctly executed and returned accurate results. Surprisingly, it was able to cope with the composite procedures with nested modalities better than expected, which simply underlined the strength of the parsing algorithm.
- **Tableaux Expansion:** The algorithm of semantic tableaux provided a good result for all experiments with the formulas and the resulting logical structures were clear. There are a few unexpected discoveries when operating highly nested modal operators and realised that there are many possibilities to optimise to get better performance for deep tableaux trees.
- **Kripke Model Generation:** Incorporation of Kripke models was effective as planned since these models of related universes made concepts of accessibility very graphical and tangible to users. It helped the user in understanding the visualisation of modal logic which was impossible without the tool.

Off-topic findings include:

- **Visualisation Feedback:** There is also a constant set back regarding depth where some users complained that it was hard to work through such large visualisations which come with nested tableaux structures. This hints toward possible enhancements in the way that users interface as well as visualisation.
- **Unexpected Formula Handling:** Some of the formulas were longer in their computation time, and this highlighted the need for further optimization especially for the unusual cases.

6.2 Goals Achieved

The primary goal of creating a scalable, intuitive semantic tableaux solver for modal logic was largely achieved. The project met the following objectives:

- **Core Logic Engine:** Implemented the semantic tableaux method correctly and successfully, ensuring about the formulas' validity as well as their satisfiability.
- **User Interface:** Created an effective web app by utilising Streamlit with the ability to allow users that even those who are not technical can be able to enter the formulas and engage with the output. But the best achieved was in formula input, the display of results and history.
- **Visualisation:** The work was successful in providing visual representations of logical structures such as tableaux trees and Kripke models thus meeting the objective of the project. These graphs were rendered with aid of the NetworkX and Pydot libraries which influenced user comprehension.
- **Performance:** It was also found that the majority of the performance goals have been achieved for most of the simple and moderately complex formula types, but only a few of the highly complex ones needed additional optimization in order to deal with nested structures.

Failures and challenges:

- **Performance Limitations:** However, responses of deeply nested formulas took sometimes more than the set time limits. This makes it necessary to improve the amount to handle such cases efficiently.
- **Incomplete / Incorrect Complex Branching:** For a few complex formulas, which have Diamond and Box operator both combined in a nested form, the branching becomes incomplete for a few and some branchings are also incorrect. But it has been identified that it could be fixed with an addition of a few sets of logical operations and some modifications.
- **User Interaction with Visualisations:** Although the visualisations provided clear insights, response feedback suggested that navigating large tableaux structures could be improved, especially when zooming and panning through the visualisations.

6.3 Further Work

Several areas for future work have emerged from the project's findings:

1. **Optimization for Complex Formulas:** There is room for performance improvements, particularly in handling formulas with deeply nested modal operators. Optimizations like using more efficient data structures or parallelizing the tableaux expansion process could help address the computational challenges for such cases.
2. **Enhanced Visualisation Tools:** There is a need to improve on the controls used in the visualisation navigation. As for the future work, the improvement of the graphic interface of

large tableaux trees and Kripke models so that the users can navigate by zooming and panning can be proposed. Moreover, the inclusion of client-side rendering might have the effect of less server utilisation, and the seamless users' experience.

3. **Support for Additional Modal Logics:** At the moment only K modal logic is implemented but the system is flexible enough and it can be further developed to include such modal logics as S4 or S5. This would entail making the tableaux rules and the accessibility relations of these systems to be broader to have a greater range of application in modal logic analysis using the tool.
4. **Educational Enhancements:** Since the purpose of the system is educational, it would be interesting to investigate further on increasing the detail of expansions of tableaux or providing simple and dynamic explanation of the procedure to understand modal logic for students and users.
5. **Scalability and Parallel Processing:** The project has potential for future improvements with the help of parallel processing which would enable the system to work on more significant amounts of information or more intricate expressions while maintaining the work's efficiency.

Chapter 7

Conclusion

The development of the **Semantic Tableaux Solver** has successfully addressed the core objectives of the project, demonstrating the practical application of modal logic through automated reasoning and visual representation. This chapter summarises the main achievements of the project, the challenges encountered, and areas for future exploration.

7.1 Achievements

1. **Implementation of a Functional Solver:** Considering a proposed program that is capable of solving propositional and modal logic formulas using semantic tableaux method, the above goals were effectively obtained in that the goal of developing a tool that would be efficient, scalable, and easy to use was achieved. Most importantly, the system demonstrates a correct handling of the various cases within basic as well as complex modal logic formulas.
2. **User-Friendly Interface:** By the help of Streamlit, the project was able to offer a user interface on the web platform for first-time users as well as experienced ones easily. By enabling direct input of the formulae, giving the results, and managing the solution history this interface keeps the system simple and informative.
3. **Visualisation of Logical Structures:** This was one of the most beneficial aspects of this project, as or was the logical reasoning process graphically presented. The tableaux trees and Kripke models generated by the system are helpful for the users to understand relations of the formulas to be assessed.
4. **Modular and Extensible Design:** To increase the functionality of the system and make other adjustments, it is possible in the future with relative ease. It is also an advantage that the Core Logic Engine does not have to be altered in order to add new views, which may be done to support other modal logics.
5. **Educational Utility:** It also educates the students or researcher who are studying modal logic and the use of the solver is conducted for this purpose. Of particular advantage is the ability to present a step-by-step solution and a clear justification for the given solution which can be very useful to users with a view of helping them grasp some of the logical arguments and formula structures in a given problem.

7.2 Challenges and Limitations

While the project has met its primary goals, a few challenges and limitations were encountered during development:

1. **Performance for Complex Formulas:** As mentioned during testing, the system is slow to process complex or multiple layered modal formulas interconnectivity. Although basic and slightly more involved equations return almost instant solutions, optimization is necessary to shorten the time required to provide the answer for the complex problems.
2. **Visualisation Scalability:** Some limitations include, for example, the difficulty to handle large sized tableaux trees and Kripke models for users to manage and interact with the visualisations. Possible additions may include: zoom-in /zoom-out; Pan; and in general, manipulation of the graphical objects.
3. **Handling of Edge Cases:** The system was capable of handling many formulas. However, there are many special cases scenarios mostly involving complex modal operations such as nested modalities which called for high computational power and hence incurred timeouts or very slow responses. This brings out the fact that there is a need to continue improving the techniques used in tableaux expansion.

7.3 Future Work

1. **Optimization for Complex and Large Formulas:** More future work can be directed in improving the efficient algorithm for expansion of tableaux for the consideration of other complicated formulas. Analytical methods like parallel processing or heuristic-based pruning of the tableaux tree should be especially helpful in the case of highly nested structures.
2. **Expanding Modal Logic Systems:** It should also check for consistency. This is something not implemented in the current system as the system only handles K modal logic thus it will be more useful when the solver is expanded to handle other modal systems such as S4 and S5.
3. **Enhanced Visualisation Tools:** Enhancing interactivity and navigation through the graphical representations of tableaux trees and Kripke models can be named one of the highly significant areas of further studies. Adding additional features to client-side rendering, as well as zoom, pan, and layer control will help improve the peoples' experience.
4. **Educational Features:** It is also possible to enhance the usability of the system as a learning tool by adding features that provide step-by-step description of the tableaux expansion, interactive lessons or simple questions that test the user's proficiency in the modal logic.
5. **Extensibility for Additional Features:** Since all the components of the system are independently replaceable, the future versions might expand the set of logical operators being supported, might include new validation modes or might be interfaced with theorem provers. This would make the solver much more capable and a useful tool for researchers and instructors at the same time.

7.4 Final Reflections

On the whole, the **Semantic Tableaux Solver** project became complete and developed a proper tool that is based on theoretical logic and computation. As one will observe, the project bears close resemblance to the outlined objectives of creating an easy to use, pedagogical and generalised modal logic solver as well as mapping areas of concern for further enhancement. Based on its current functionality the tool provides useful information on the world of formal logic and presents a good basis for additional updates.

Bibliography

[1] Baltag, A., & Renne, B. (n.d.). Dynamic Epistemic Logic > *Appendix A: Kripke models for modal logic (Stanford Encyclopedia of Philosophy)*. Stanford Encyclopedia of Philosophy. Retrieved September 10, 2024, https://plato.stanford.edu/entries/dynamic-epistemic/appendix-A-kripke.html

[2] Blackburn, Patrick, et al. *Modal Logic*. Cambridge University Press, 22 Aug. 2002.

[3] C. Klement, Kevin . "Propositional Logic | Internet Encyclopedia of Philosophy." Utm.edu, 2024, iep.utm.edu/propositional-logic-sentential-logic/#H1

[4] Carneades.org. "What Is Kripke Semantics? (Modal Logic)." *YouTube*, 15 Oct. 2017, www.youtube.com/watch?v=k3Jjw8oJqBk

[5] Copi, Irving. 1953. *Introduction to Logic.* New York: Macmillan.

[6] Davis, Martin; Putnam, Hilary (1960). "A Computing Procedure for Quantification Theory". *Journal of the ACM*. 7 (3): 201–215. doi:10.1145/321033.321034. S2CID 31888376.

[7] Davis, Martin; Logemann, George; Loveland, Donald (1961). "A Machine Program for Theorem Proving". *Communications of the ACM*. 5 (7): 394–397. doi:10.1145/368273.368557. hdl:2027/mdp.39015095248095. S2CID 15866917.

[8] Fagin, R., Halpern, J. Y., Moses, Y., & Vardi, M. Y. (1995). *Reasoning about knowledge*. MIT Press.

[9] Gallier, Jean H. *Logic for Computer Science*. Courier Dover Publications, 18 June 2015.

[10] Garson, James. "Modal Logic." *Stanford Encyclopedia of Philosophy*, Metaphysics Research Lab, Stanford University, 2018, plato.stanford.edu/entries/logic-modal/

[11] Hagberg, A., Schult, D., & Swart, P. (n.d.). *NetworkX documentation*. https://networkx.github.io

[12] Halbe, Thom. "Logical Analysis: *A Systematic Approach to Evaluating Arguments and Claims.*" Mathematica Eterna, vol. 12, no. 3, 9 Sept. 2022, pp. 1–1, www.longdom.org/articles/logical-analysis-a-systematic-approach-to-evaluating-arguments-and-claims-99655.html#:~:text=Logical%20analysis%20is%20a%20method, https://doi.org/10.35248/1314-3344.22.12.160.

[13] Halpern, J. Y., & Vardi, M. Y. (1989). *The complexity of reasoning about knowledge and time*. Journal of Computer and System Sciences, 38(1), 195-237.

[14] Happ, S. (n.d.). *pydot: Python interface to Graphviz*. https://github.com/pydot/pydot

[15] Hughes, G. and Schagrin, . Morton L. (2024, May 9). *formal logic. Encyclopedia Britannica*. https://www.britannica.com/topic/formal-logic

[16] Hunter, J. D. (2007). *Matplotlib: A 2D graphics environment*. Computing in Science & Engineering, 9(3), 90-95. https://doi.org/10.1109/MCSE.2007.55

[17] Kirsling, R. (n.d.). *Modal Logic Playground*. https://rkirsling.github.io/modallogic/

[18] Luis Fariñas del Cerro, David Fauthoux, Olivier Gasquet, Andreas Herzig, Dominique Longin, et al.. Lotrec: *a generic tableau prover for modal and description logics*. International Joint Conference on Automated Reasoning (IJCAR 2001), Jun 2001, Siena, Italy. pp.453-458. hal-03523405

[19] Manwani, Chirag . "Mathematics | Introduction to Propositional Logic | Set 1." *GeeksforGeeks*, 19 June 2015, www.geeksforgeeks.org/proposition-logic/

[20] McCune, W. (n.d.). *Prover9 and Mace4*. University of New Mexico. https://www.cs.unm.edu/~mccune/prover9/

[21] M. Castilho, L. Farinas del Cerro, O. Gasquet, and A. Herzig, *Modal tableaux with propagation rules and structural rules,* Fund Inf 32(3): 281-297, 1997.

[22] Mendelson, Elliott. *Introduction to Mathematical Logic*. CRC Press, 21 May 2015.

[23] nptelhrd. "Mod-01 Lec-19 Semantic Tableaux Method for Propositional Logic." *YouTube*, 17 Mar. 2015, www.youtube.com/watch?v=ebE4rOI32uM.

[24] Otten, J. (n.d.). *mLeanCoP: A Connection Prover for First-Order Modal Logic*. https://www.leancop.de/mleancop/

[25] Python Software Foundation. (n.d.). *Python (Version 3.x)*. https://www.python.org

[26] Sarma, A. V. R. (n.d.). *Introduction to Logic* [Lecture notes]. Department of Humanities and Social Sciences, IIT Kanpur. NPTEL. http://nptel.ac.in

[27] Schmidt, R. A. (n.d.). *Trees for modal logic*. https://www.umsu.de/trees/

[28] SmartDraw, Inc. (n.d.). *SmartDraw: The world's best diagramming software*. https://www.smartdraw.com

[29] Smullyan, Raymond M. *First-Order Logic, by Raymond M. Smullyan*. 1968.

[30] Streamlit, Inc. (n.d.). *Streamlit: The fastest way to build and share data apps.* https://streamlit.io

[31] Zach, Richard, et al. *The Open Logic Text*. The Open Logic Project, 2016.

Appendices

Figure 1. *Negated Tree Branching for formula*
<>(p | q) -> (<>p | <>q)

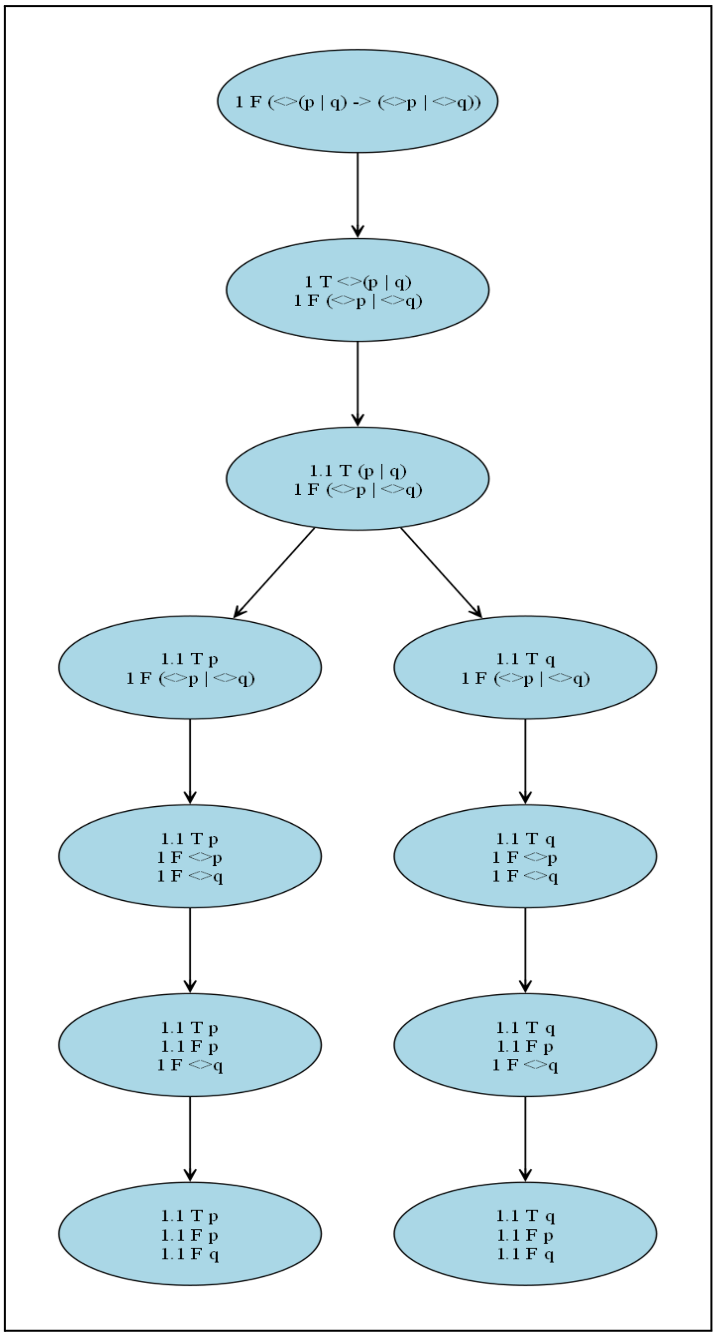


Figure 2. *Negated Tree Branching for formula*

◊(p ∨ q) → (◊p ∨ ◊q)

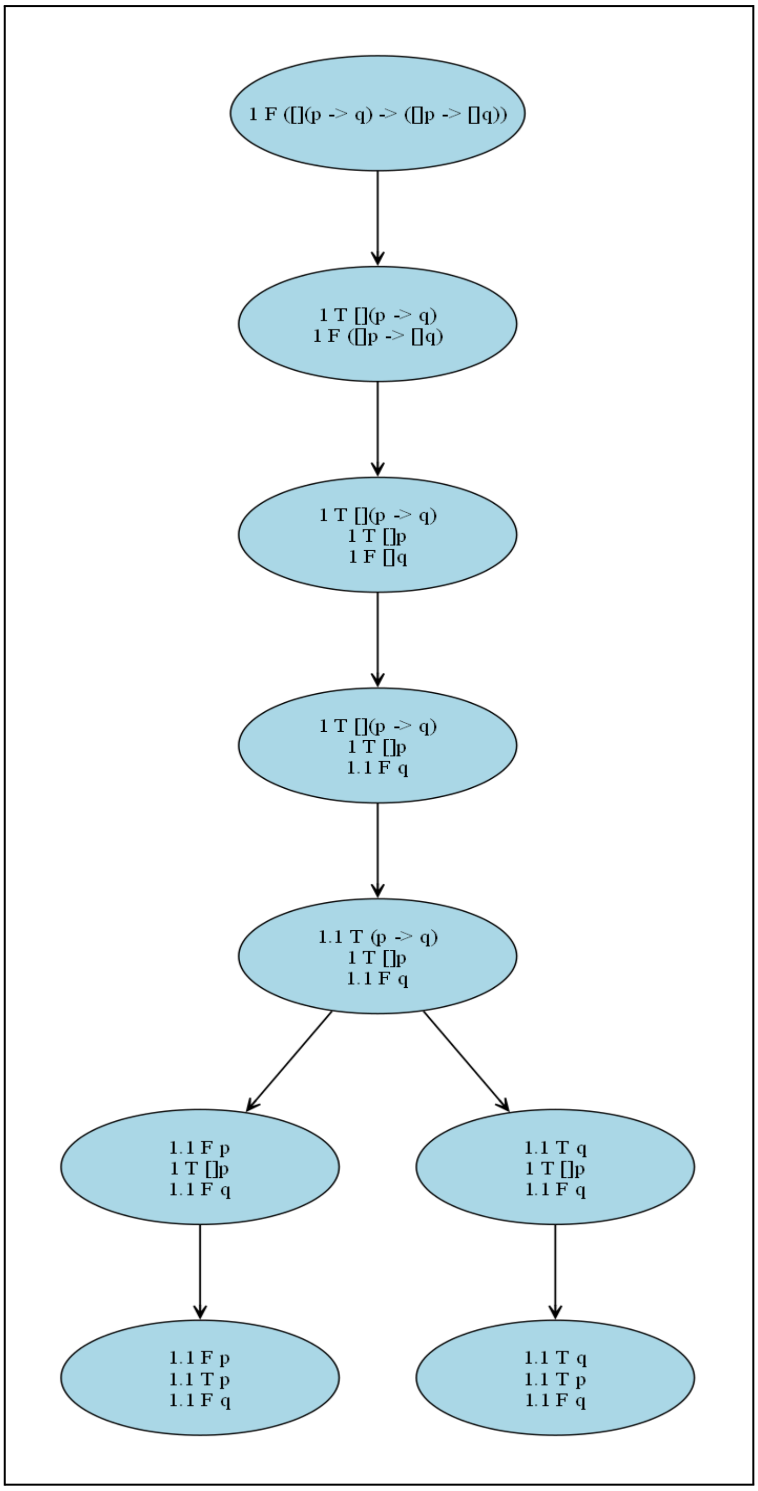
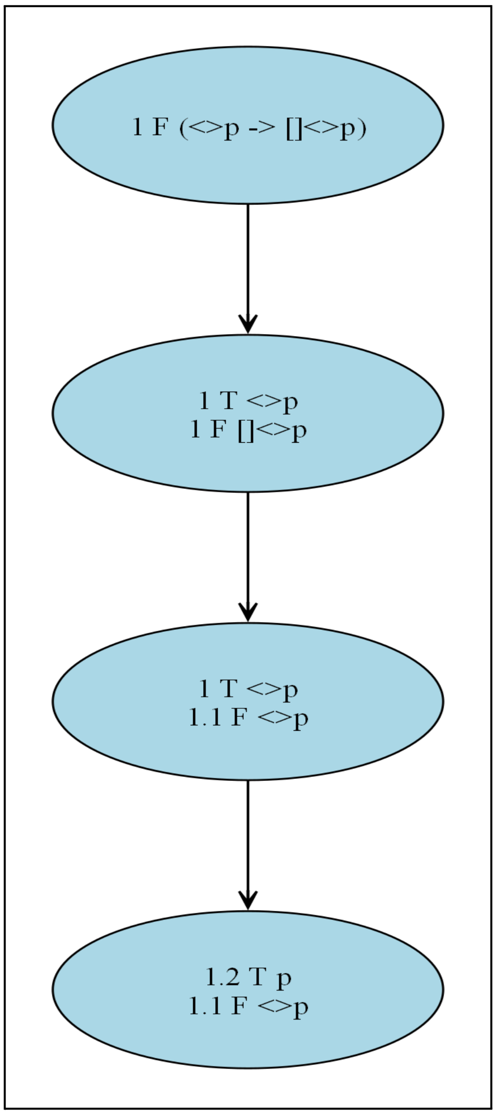


Figure 3. *Failed branching for the formula* ***<>p -> []<>p*** *for validity check.*



Final branches should be **1.2 T*p*, 1.1 F*p*, 1.2 F*p***