Temporal Logic bot with reflexivity and transitivity as constraints

Bachelor's Project Thesis

Thalia Najjar, s3614794, r.najjar@student.rug.nl,
Supervisor: prof. dr. L.C. Verbrugge

Abstract: A Twitter bot that regularly publishes new tautologies in Temporal Logic (TL) was implemented in this study. The design of the bot relies on exhaustively generating TL formulas before testing them with the reflexive and transitive constraints using a semantic tableau system. Various shortcuts have been implemented to enhance the system's performance. Its efficiency is assessed with regards to its soundness and completeness, its run time relatively to the complexity of the formulas, as well as by comparing it to a pre-existing temporal logic bot. The system appeared to be sound but not complete and has a consistent average run time per formula for formulas having four connectives and less.

1 Introduction

A fact that is true today might not have always been true in the past and might not always be true in the future. Expressing a single fact at two different moments in time requires the use of two different propositions in Propositional Logic or First-Order Logic, but not in Temporal Logic. Temporal Logic is a form of Modal Logic that integrates the concept of time. A single proposition can be evaluated at different moments in time, among which the present, once in the past, once in the future, always in the past or always in the future (Goranko and Rumberg, 2020).

In this research, a temporal logic bot will be designed with the reflexive and transitive restrictions. The bot will induce new formulas all the while regularly posting tautologies on the social networking service *Twitter.*

1.1 Temporal Logic

Introduced in the late 1950s by Arthur Prior, Tense Logic (TL), later also called Temporal Logic, is a system of rules that expresses and reasons about propositions with regards to time. Troubled by philosophical matters related to time, namely predestination and free will, Prior put forward Tense Logic (Øhrstrøm and Hasle, 1993). Since then, Prior's TL proposal has been revised and its applications have broadened as it is now used for the manipulation of time-dependent data, database management or reasoning in artificial intelligence. The embodiment of TL is a collection of worlds related to one another with respect to time. In order to formalize the new reasoning instigated by Tense Logic, four new temporal operators were established, in addition to the wonted truth-functional operators (Goranko and Rumberg, 2020):

- F (or ⟨F⟩) at some time in the future
- P (or ⟨P⟩) at some time in the past
- G (or [F]) at all future times
- H (or [P]) at all past times

At first glance, it seems like temporal logic has introduced four new tenses in logic. However, in actual fact, combining temporal connectives yields additional tenses that logical formulas may make use of (Øhrstrøm and Hasle, 1993). On this account, temporal logic has introduced fifteen new tenses in logic rather than four, as assessed by Hamblin and Prior (Goranko and Rumberg, 2020).

For temporal logic, Kripke semantics are followed. In Kripke models, the set of all worlds is referred to as W. When working with TL, the relations between the different worlds is primordial as they unveil their temporal links. For all w_1, $w_2 \in W$, if w_1 is related to w_2 ($w_1 R w_2$), then w_1 is in the past of w_2, and w_2 is in w_1's future.

Additionally, the Tense Logic considered in this research is paired with the reflexive and transitive restrictions. Time is a concept that is hard to bound. The reflexive relation (ρ) discloses the fact that every world is related to itself. Implementing this world restriction entails that the past extends up until the present moment and the future starts now. Hence, for all $w \in W$, wRw holds.

Similarly, time can be thought of as a linear progression of discrete moments. A moment that is in the past of the present's past will also be in the past of the current instant. The transitive restriction (τ) captures this property. It is a relationship between three worlds defined as: for all w_1, w_2, $w_3 \in W$, if w_1 is related to w_2 (w_1Rw_2) and w_2 is related to w_3 (w_2Rw_3), then w_1 is related to w_3 (w_1Rw_3) (Priest, 2008).

1.2 Tableau rules for TL

The tableau method, invented in 1955 by Evert Willem Beth, is often used in logic to check whether a certain formula or inference is valid. It relies on setting up a tree structure that is sequentially developed as the tree's formulas are worked out.

The goal of a semantic tableau is, for sentences, to try and refute the original sentence by untangling its negation or, in the case of an inference, to try and prove that its premises may be true while the conclusion is false. In this research, the focus will be placed on formulas rather than inferences. If the negation of a sentence is satisfiable, then the sentence itself is not valid. However, if a negation cannot be satisfied, then it means that the original formula is valid.

When a branch in a tableau features two contradicting formulas, it is inconsistent and is said to be *closed*. A branch that does not contain any inconsistencies is *open*. Additionally, a branch is said to be *complete* when all rules than can be applied to it have already been applied (Nour, 2002). If a branch is open and complete, then the negation of the original formula is satisfiable, which means the formula is not a tautology and this specific branch serves as a counter-example. However, if all branches of a tableau close, then the negation of the formula is not satisfiable which implies the original formula has been proved. From this follows, as mentioned by Smullyan (1995) in the refinement of Beth's work, "[...] every formula provable by the tableau method must be a tautology".

Tableau rules may only be applied on the main connective of a formula. They follow pre-established rules derived from the semantics of the connective at hand. For the connectives that are not specific to Tense Logic, namely the negation ($\neg$), the conjunction ($\wedge$), the disjunction ($\vee$), the implication ($\supset$) and the bi-implication ($\leftrightarrow$) connectives, the tableau rules are the same as the usual tableau rules that comply with Kripke's semantics (Priest, 2008). However, new tableau rules were created for the newly introduced connectives.

First, the tableau rules for connectives G and H are:

$$\begin{tikzcd}
GA,i & HA,i \\
irj \arrow[d] & jri \arrow[d] \\
A,j & A,j
\end{tikzcd}$$

These two rules should be applied to *all j* such that the relation *irj* or *jri* appears on the branch respectively for *GA,i* and *HA,i*.

Additionally, the tableau rules for connectives F and P are:

$$\begin{tikzcd}
FA,i \arrow[d] & PA,i \arrow[d] \\
irj & jri \\
A,j & A,j
\end{tikzcd}$$

For both *FA,i* and *PA,i*'s rules, the *j* used is new. It should be introduced by these rule applications through the *irj* or *jri* relation respectively for connectives F and P.

Nonetheless, the above mentioned tableau rules are not the only new rules in TL. Four additional rules need to be defined, which reflect the semantics of the negated temporal connectives:

$$\begin{tikzcd}
\neg GA,i \arrow[d] & \neg HA,i \arrow[d] & \neg FA,i \arrow[d] & \neg PA,i \arrow[d] \\
F\neg A,i & P\neg A,i & G\neg A,i & H\neg A,i
\end{tikzcd}$$

By further application of tableau rules and in an attempt to simplify the tableau's unfolding, the four rules above may be applied using the following shortcuts:

$$\begin{array}{cccc} \neg GA,i & \neg HA,i & \neg FA,i & \neg PA,i \\ \downarrow & \downarrow & irj & jri \\ irj & jri & \downarrow & \downarrow \\ \neg A,j & \neg A,j & \neg A,j & \neg A,j \end{array}$$

Similarly to previous rules, $\neg GA,i$ and $\neg HA,i$ introduce a *new* j with its corresponding world relation, while $\neg FA,i$ and $\neg PA,i$ are applied to *all* j such that *irj* and *jri* respectively appear on the considered branch (Priest, 2008).

Moreover, the temporal logic discussed in this paper has been paired with the reflexive (ρ) and transitive (τ) constraints. Each constraint has a corresponding tableau rule. On the one hand, the tableau rule for ρ is, for all i on the branch:

$$\begin{array}{c} \bullet \\ \downarrow \\ iri \end{array}$$

On the other hand, the tableau rule for τ is, for all *irj* and *jrk* on the branch:

$$\begin{array}{c} irj \\ jrk \\ \downarrow \\ irk \end{array}$$

1.3 Research question

Within the framework of the Bachelor's Project, a tense logic bot is designed. Its aim is to generate new tautologies to be posted on the *Twitter* social media. The design of the logic bot relies on testing tense logic formulas with the reflexive and transitive constraints, using a semantic tableau system.

The research question for this Bachelor Project thus is: *How much of an efficient temporal logic twitter bot can we set up using semantic tableaux, with reflexivity and transitivity as constraints?*
In particular,

- *Can the bot generate a sound and complete logic?*
- *How does the complexity of the formulas affect its efficiency?*
- *How does it perform compared to another temporal logic bot?*

The programmed system will be evaluated by investigating whether it satisfies the soundness and completeness properties of logic, and by testing its efficiency with regards to the complexity of the formulas. Additionally, the system will be compared to a pre-existing logic bot (de Vries, 2018). All of these examinations aim at testing the usability and the efficacy of the set up logic bot.

1.4 Soundness and completeness in Kripke's models

In logic, two properties attest the fact that validities are provable, which are the soundness and completeness properties. The soundness theorem is described, for a finite set of premises Σ, as: "If $\Sigma \vdash$ A, then $\Sigma \vDash$ A". That is, a formula A that is provable from a set of sentences Σ must also be a logical consequence of Σ. Strictly speaking, the soundness theorem asserts that a wrong inference may not be proved.

The completeness theorem is the converse of the soundness theorem. Its definition is, for a finite set of premises Σ: "If $\Sigma \vDash$ A, then $\Sigma \vdash$ A". Thus, if a formula A is a logical consequence of the set of formulas Σ, then it must be provable from Σ (Priest, 2008). Strictly speaking, the completeness theorem claims that any correct inference can be proved.

1.5 Twitter bot

Launched in 2006, the social networking service *Twitter* is an American microblogging system that allows users to publish posts called *tweets*. A single tweet can hold a maximum of 280 characters (Rigolin, 2018).

Twitter offers an open application interface that allows users to pair the social media application to another application. This research's logic bot will be programmed using the class-based, object-oriented programming language *Java* before connecting it to the *Twitter* service.

On *Twitter*, a logic bot that regularly publishes tautologies in classical propositional logic already exists, namely the `@mathslogicbot` account. The project at hand aims at building a similar logic bot for another type of logic, temporal logic.

2 Methods

2.1 Programming language

The semantic tableau system is designed using the class-based, object-oriented programming language

Java. *Java* allows for a well planned and structured code through the use of classes. The object-oriented approach is convenient as it allows for a practical partitioning of the tableau elements for an efficient system. *Java* is a multi-threaded programming language, which means it allows for the parallel run of different tasks. Nevertheless, like any programming language, *Java* has drawbacks, namely that it is memory-consuming and runs slower than other programming languages such as C or C++.

Designing a temporal logic bot that induces tautologies calls for both extensive understanding of human-like processing as well as adequate proficiency in the programming language at hand as to implement efficient computing techniques.

2.2 Formula Factory

Before checking temporal logic formulas using the semantic tableau method, an algorithm meant to generate these logical formulas is set up. The aim is to start by generating the least complex formulas and to sequentially increase their complexities by having more and more connectives. When the length of the generated formulas reaches the 280 character limit actuated by *Twitter*, the formula factory is terminated.

The formulas are stored in a tree structure. The root of the tree is the main connective of the formula and the leaves of the tree are the individual atoms used to build the formula.

In order to generate the formulas, three atoms are made available to use, namely a, b, and c. The number of atoms available is set to three in an attempt to control the number of formulas generated while allowing for some variability.

As mentioned above, the program progressively generates more and more complex formulas. For that, previously created formulas are used when generating a formula with a higher complexity. That is, first of all, all atomic formulas are generated (simply a, b, and c). Then, all possible formulas that only have one connective are created. For that, the system loops over the set of all connectives $\{\neg, G, H, F, P, \wedge, \vee, \supset, \leftrightarrow\}$ and combines them in every possible way with the previously created less complex formulas, namely the atomic formulas. Next, all TL formulas that have two connectives are produced by undergoing the same process. Formulas with zero and one connectives are loaded from files. All formula combinations yielding a formula with two connectives are built. The process is repeated over and over again by sequentially increasing the complexity of the formulas, looping over all connectives and assembling the required combinations. For example, in the case of formula complexity three, unary connectives are only combined with formulas that have complexity two. Binary connectives combine formulas that have complexity zero with others that have complexity two as well as two formulas with complexity one, and so on for all formula complexities. The pseudocode of the main function can be found in Appendix A.1, and the general structure of the formula factory is depicted in Figure 2.1.

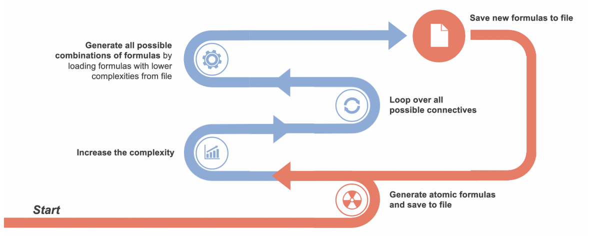
Save new formulas to file
Generate all possible combinations of formulas by loading formulas with lower complexities from file
Loop over all possible connectives
Increase the complexity
Generate atomic formulas and save to file
Start

Figure 2.1: Schematic representation of the formula factory implementation

If the commutative laws hold with the connective at hand, a formula equivalent to an other already generated one is not produced. The reason for this is that the aim of the bot is to create new distinct tautologies, which is why duplicates are undesirable for the system. All binary connectives except the implication are commutative. On that account, the production of formulas that have as main connective an implication has a slightly different implementation.

External files are used to save and monitor the logical formulas being dealt with. One `.ser` file per formula complexity is used to write all generated temporal logic formulas with this complexity. Once the formula factory creates all new formulas with a given formula complexity, they are written in the appropriate `.ser` file before moving on to the next formula complexity. All of these formulas are then serially checked by the semantic tableau solver.

2.3 Tableau solver

As to try and obtain an efficient program, checking the validity of formulas using semantic tableaux should take advantage of both human-like procedures and additional shortcuts.

The tableau is stored in a tree structure. Its nodes may either be formula nodes or relation nodes. When first setting up a tableau, the evaluated formula is negated. This negated sentence constitutes the tableau's initial list. In an attempt to reduce the number of rules applied, double negations are simplified as soon as possible. Thus, the initial list is simplified if necessary and is set to be true in world 0. It is then used as the tableau tree's root and is added to the queue of rules to be applied. At first, the tree is only formed by two nodes: the simplified initial list and the reflexive relation *0r0*. As propositions are worked out, new nodes are added to the tree. Every time a formula node is added, double negations are simplified whenever they are present as to find contradictions more easily. Shortcuts are thus taken advantage of to reduce the run time of the system. Simplifications do not alter the soundness and completeness of the logic at hand, which is why they may be applied. The pseudocode of the main function of the Tableau Solver can be found in Appendix A.2.

When solving a semantic tableau, tableau rules may be applied in any order, but not all application orders yield the same length and complexity of tableaux. Therefore, as the aim of this system is to maximize efficiency while minimizing the run time, a priority queue is implemented. All rule application priorities are displayed in Table 2.1. They have been determined based on how straightforward their rule applications and resulting tableaux are. For example, rule applications that do not split a branch (e.g.: $\wedge$, $\neg\vee$ and $\neg\supset$) have priority over rule applications that do split a branch (e.g.: $\vee$, $\neg\wedge$ and $\supset$). Thus, whenever a rule is applied, its resulting formulas are added to this queue depending on the priorities of their operations. All unapplied rules need to be in the priority queue. Every time a new rule is applied, it is removed from the priority queue. Literals are excluded from it. All tableau rules are defined within the relevant connective class.

There are four rule applications that have been implemented differently, namely the rules for G, H, $\neg F$ and $\neg P$. All rules featured in Table 2.1 are applied once, after which their formula is never considered again. However, not all rule applications operate in the same way in temporal logic. The four rule applications G, H, $\neg F$ and $\neg P$ are exemptions to this general one-application principle. These operations are universal, as they are to be carried out on as many relations as applicable. Their implementation thus applies the following procedure: if a relation node is added, its branch is traversed to see if such a universal rule that may be applied to the relation at hand is featured on the branch. If it is, the tableau rule is applied on the newly added relation. However, it may also happen that the universal rule is added to a branch after a relation node that is in its scope. In order to account for it, the universal tableau rules are to be evaluated by looking for relevant relation nodes on their branch as soon as they are added to it.

Priority	Rules
1	$\neg\neg$
2	$\wedge$, $\neg\vee$ and $\neg\supset$
3	F, P, $\neg G$ and $\neg H$
4	$\vee$, $\supset$ and $\neg\wedge$
5	$\leftrightarrow$ and $\neg\leftrightarrow$

Table 2.1: Priority of the tableau rules to be applied

At every unfolding of a rule, new formulas are to be added to the tree. In order to avoid having an ineffective lengthy tableau, a node on which the splitting of a branch is not dependent is only added if it is not already featured on its branch. However, if the splitting of a branch is dependent on this node addition, not adding a duplicated node would result in erroneous results. Therefore, in this case, the identical node is not dropped. Nevertheless, splitting rules are also optimized. Before applying a splitting rule, the two resulting sub-branches are compared. A branch is then only separated in two if its sub-branches are different. Therefore, purposeless branch splittings are eluded.

Whenever a new formula node is added to the tableau, its branch is immediately traversed to check whether its negation already appears on it. This is done to close branches as soon as possible and is also how a human would solve a tableau.

If the negation of the formula is present on the branch, the considered branch must close. A branch closing is reflected in the system by the deletion of all nodes that are specific to it. The information stored in this branch is not relevant to the tableau solver anymore. Keeping it would be inefficient. Thus, when a contradiction is detected, the root of the considered branch is searched for by traveling upwards through the nodes. When it is found, the branch featuring the contradiction is deleted from the tableau tree. If a contradiction is detected when adding the last node of an operation application, then the branch deletion is immediately executed. Otherwise, if the rule application is not complete yet, adding new nodes afterwards from the same operation application may still alter the status of the tableau, which is why the branch is not erased right away but rather at the end of the current rule application. To better illustrate this issue, take for instance the application of a splitting rule. If the node of the left sub-branch is first added and a contradiction is found, deleting the left child's branch results in the erasure of the parent branch as well, even though there still is a right sub-branch that has not been added yet. Such case figures cause invalid conclusions, which is why branch deletions are not applied instantly, but rather at the end of each rule application.

If all of the tableau's nodes are erased, it means that the tableau is closed, which entails that the evaluated formula is a tautology.

Whenever a new relation node is added to the tableau, constraints are to be reviewed as not to skip any new world relation. The reflexivity restriction is applied every time a new world is introduced. Similarly, the transitivity restriction is applied as soon as its introducing pattern is detected. Additionally, the G, H, $\neg F$ and $\neg P$ connectives are to be reviewed at every relation node addition, as discussed above.

Upon the creation of the tableau, a limit on the number of formulas that may be applied as well as on the number of worlds that may be introduced are determined. The defined limits are set up to prevent the creation of infinite tableaux. Both of them depend on the number of connectives that compose the evaluated sentence. These maximal values are both set to be three times the complexity of the studied formula. The limits have been established based on code efficiency, *Java* capacities and logic principles. That is, as to get an efficient system, infinite tableaux should be detected and stopped as early as possible. *Java* also has capacity limits on the length of stacks that it may handle, which makes it compelling to restrict unbounded tableaux. A middle ground thus had to be found to balance between the prevention of infinite tableaux and the successful detection of tautologies with long tableaux. It came to be a limit of three times the number of connectives in the initial sentence for both the number of worlds introduced and the number of formulas applied.

There are three case figures by which the tableau checking can be terminated.

- The tableau tree is empty as a result of the closing of all of its branches. The tableau is closed, the evaluated formula is a tautology
- The priority queue is empty. All rules that may be applied have been applied but the tableau still has an open branch. This open branch is also complete, and the evaluated formula was therefore not a tautology
- The number of rules applied matches or exceeds the set limit. The tableau is still open but not complete. The formula is either not a tautology, or it requires an extensive number of rule application in order to close

2.4 Twitter posting

After being checked by the tableau solver, formulas that are detected as tautologies are stored in an Array List while waiting to be posted to the *Twitter* social media. The connection between *Java* and *Twitter* is established using a *Java* library for the *Twitter API* called *Twitter4J*. For demonstration purposes, the system is initially programmed to post one tautology every twenty minutes of terminal activity. However, it will be modified to a longer time interval of once every 6 hours upon its hosting on the online platform *Heroku*. As discussed earlier, *Java* is a multi-threaded programming language. It thus enables the parallel run of different tasks. By that means, the formula factory paired with the tautology checking is ran at the same time as the *Twitter* posting.

2.5 Experiment

Once the logic bot is completed, its performance is tested to assess the efficiency of the program and answer the defined research question. On the one hand, the soundness and completeness properties of the set up logic are evaluated. The proofs are worked out in section 3.2. On the other hand, the bot's computational features are analyzed. The run times of the two main parts of the system, namely the formula factory and the tableau solver, are measured independently for different formula complexities. In addition, the used memory storage as well as the system's performance compared to a pre-existing temporal tableau solver are also investigated.

For the evaluation of the system's run time, as mentioned earlier, the formula factory and the tableau solver are studied separately. For formulas with complexities that are lower than four, the run time for each and every formula is measured and then averaged per formula complexity. The number of generated formulas grows exponentially as the formula complexity increases. The current system is ran on a personal device, on which the program only runs when the device is in use. In these circumstances, the extensive production and solving of formulas with complexities that are equal to or higher than four requires a few days per formula complexity. For that reason, not all possible formulas are tested in these cases. Instead, for formulas with complexities four and five, only part of the possible formulas are considered. Nevertheless, to account for their variability and complexity, more and more formulas are tested as the formula complexity increases. For that, the custom number of tested formulas is dependent on the considered formula complexity. The formulas tested are derived from all possible connective and lower complexity formula combinations in an attempt to assess a larger range of rule applications rather than only testing similar formulas. Here again, their run times are measured separately for each formula then averaged per formula complexity. The number of tested formulas for formula complexities zero to five are 3, 42, 840, 21273, 66182 and 265595 respectively. For this purpose, available pre-built *Java* methods are used to accurately measure the run time.

With the aim of maximizing the efficiency of the formula factory, previously generated formulas are saved in serializable files. The memory usage of these serialized formulas is studied by analysis of their resulting files.

Moreover, the performance of the designed bot is also assessed by comparing it to a pre-existing temporal logic tableau solver. This bot has been designed by de Vries (2018). For that comparison, fifteen arbitrary formulas are determined and tested on both systems for formula complexities ranging between 1 and 6. The run time of both tableau solvers are measured and averaged per formula complexity using pre-built formulas. O. de Vries' solver automatically outputs the time needed to compute the solving, which is the value that is used for comparison. The extensive list of tested formulas is shown in Appendix B.1 and B.2. A combination of both tautologies and non-tautologies are tested for every considered formula complexity. The connectives that constitute the formulas are varied as to test and compare a wider range of operations.

3 Results

3.1 The *Twitter* account

The outcome of the designed logic bot can be observed on the *Twitter* account @tenselogicbot.

3.2 Soundness and completeness

The soundness and completeness theorems relevant to Kripke's models have been proved in Priest (2008). In this section, the soundness and completeness properties relevant to this project's characteristics, namely the temporal connectives as well as the accompanying constraints, will be discussed.

3.2.1 Soundness

Definition 3.1 (Faithful).
Let $I = \langle W, R, v\rangle$ be any modal interpretation and let b be any branch of a TL tableau.
Then I is *faithful* to b iff there is a map f from the natural numbers to W such that:

- For every node D, i on b, D is true at $f(i)$ in I;
- If irj is on b, $f(i)Rf(j)$ in I.

We say that f shows I to be *faithful* to b.

Lemma 3.1 (Soundness lemma).
Let b be any branch of a tableau, and let $I = \langle W, R, v\rangle$ be any modal interpretation where R is reflexive and transitive.
If I is faithful to branch b, and a TL tableau rule is applied to b, then that rule produces at least one extension b' such that I is faithful to b'.

Proof:
Suppose that f shows I to be faithful to b.
For every TL tableau rule and for both considered constraint rules (ρ, τ), it will be proven that f shows I to be faithful to at least one extension b' resulting from the rule applications.

Suppose GA, i occurs on b, and the tableau rule of G is applied. This results in the extended branch b'. Since I is faithful to b, every formula on b is true, including GA that is true at $f(i)$. For all i and j, and for every relation irj on b, $f(i)Rf(j)$. For every irj occurring on b, the branch b is extended with A, j. A is true at $f(j)$ by the semantics of G. Hence, I is faithful to branch b', extended from b.

Suppose HA, i occurs on b, and the tableau rule of H is applied. This results in the extended branch b'. Since I is faithful to b, every formula on b is true, including HA that is true at $f(i)$. For all j and i, and for every relation jri on b, $f(j)Rf(i)$. For every jri occurring on b, the branch b is extended with A, j. A is true at $f(j)$ by the semantics of H. Hence, I is faithful to branch b', extended from b.

Suppose FA, i occurs on b, and the tableau rule of F is applied. This results in the extended branch b'. Since I is faithful to b, every formula on b is true, including FA that is true at $f(i)$. Thus, for some $w \in W$, $f(i)Rw$, and A is true at w. Let f' be the same as f except that $f'(j) = w$. Therefore $f'(i)Rf'(j)$ and A is true in $f'(j)$. Hence, I is faithful to branch b', extended from b.

Suppose PA, i occurs on b, and the tableau rule of P is applied. This results in the extended branch b'. Since I is faithful to b, every formula on b is true, including PA that is true at $f(i)$. Thus, for some $w \in W$, $wRf(i)$, and A is true at w. Let f' be the same as f except that $f'(j) = w$. Therefore $f'(j)Rf'(i)$ and A is true in $f'(j)$. Hence, I is faithful to branch b', extended from b.

Rule for reflexivity ρ: Suppose that i occurs on b, and the constraint rule of ρ is applied. The branch b is extended and results in the extended branch b' on which iri occurs. Since I is faithful to b, $f(i) \in W$ and thus $f(i)Rf(i)$ by the semantics of reflexivity. Hence, I is faithful to branch b', extended from b.

Rule for transitivity τ: Suppose that irj and jrk occur on b, and the constraint rule of τ is applied. This results in the extended branch b'. Since I is faithful to b, every formula on b is true, including irj and jrk, and thus $f(i)Rf(j)$ and $f(j)Rf(k)$. The branch b is extended with irk and $f(i)Rf(k)$ by the semantics of transitivity. Hence, I is faithful to branch b', extended from b.

Theorem 3.1 (Soundness theorem).
For a finite set of rules Σ,
If $\Sigma \vdash_{K^t_{\rho\tau}} A$, then $\Sigma \vDash_{K^t_{\rho\tau}} A$

Proof:
The soundness theorem is proved by contraposition using the soundness lemma.
Suppose $\Sigma \nvDash_{K^t_{\rho\tau}}$ A. Then there is an interpretation $I = \langle W, R, v\rangle$ in which R is reflexive and transitive that makes every premise in Σ true, and A false at some world, $w \in W$. Let f be any function such that $f(0) = w$. This shows I to be faithful to the initial list.

By repeatedly applying the Soundness lemma, we find a resulting extended branch b such that I is faithful to every initial section of it. If b is closed, then two contradicting formulas occur on some of its initial section. Given that I is faithful to b, this is impossible, which means that the tableau is open. Hence, $\Sigma \nvdash_{K^t_{\rho\tau}}$ A.
Thus, we conclude that if $\Sigma \vdash_{K^t_{\rho\tau}}$ A, then $\Sigma \vDash_{K^t_{\rho\tau}}$ A, which entails the logic is sound.

3.2.2 Completeness

Definition 3.2 (Induced interpretation). Let b be any open complete branch of a TL tableau. We say that an interpretation $I = \langle W, R, v\rangle$ is an interpretation induced by b iff:

- $W = \{w_i : i \text{ occurs on } b\}$;
- $w_iRw_j \in R$ iff irj occurs on b;
- If p, i occurs on b, then $v_{w_i}(p) = 1$;
  if $\neg p, i$ occurs on b, then $v_{w_i}(p) = 0$; otherwise $v_{w_i}(p)$ can be anything that one likes (either 0 or 1)

Lemma 3.2 (Completeness lemma).
Let b be any open complete branch of a TL tableau. Let $I = \langle W, R, v\rangle$ be an interpretation induced by b.

Then, for all formulas D and for all i, the following holds:

If D, i is on b, then D is true at w_i

If $\neg D, i$ is on b, then D is false at w_i

Proof:

For every TL tableau rule and for both considered constraint rules (ρ, τ), the completeness lemma will be proven by induction on the complexity of D.

Suppose GA, i occurs on b. Since b is complete, then for all j such that irj occurs on b, A, j also occurs on b. By induction hypothesis, for all w_i such that $w_i R w_j$, A is true in j. Hence, GA is true at w_i, as required.

Suppose HA, i occurs on b. Since b is complete, then for all j such that jri occurs on b, A, j also occurs on b. By induction hypothesis, for all w_i such that $w_j R w_i$, A is true in j. Hence, HA is true at w_i, as required.

Suppose FA, i occurs on b. Since b is complete, then there exists a j such that irj and A, j occur on b. By induction hypothesis, $w_i R w_j$ and A is true in j. Hence, FA is true at w_i, as required.

Suppose PA, i occurs on b. Since b is complete, then there exists a j such that jri and A, j occur on b. By induction hypothesis, $w_j R w_i$ and A is true in j. Hence, PA is true at w_i, as required.

Rule for reflexivity ρ: For every $w_i \in W$, iri occurs on b by the semantics of reflexivity.

Rule for transitivity τ: For w_i, w_i, $w_i \in W$, suppose that $w_i R w_j$ and $w_j R w_k$ hold. Then irj and jrk occur on b, which means by the semantics of transitivity that irk occurs on b. Hence, $w_i R w_k$, as required.

Theorem 3.2 (Completeness theorem)**.**
For a finite set of rules Σ:
If $\Sigma \vDash_{K^t_{\rho\tau}} A$, then $\Sigma \vdash_{K^t_{\rho\tau}} A$

Proof:

The completeness theorem is proved by contraposition using the completeness lemma.

Suppose $\Sigma \nvdash_{K^t_{\rho\tau}}$ A.

Let $I = \langle W, R, v\rangle$ be an interpretation induced by an open branch b, in which R is reflexive and transitive. Then, by the completeness lemma, I makes every premise in Σ true at w_0, and A false at w_0. Hence, $\Sigma \nvDash_{K^t_{\rho\tau}}$ A.

However, the completeness lemma can only be applied to open and complete branches. The tableau checking implemented in this research has, in some cases, a branch b that is open but not complete while the tableau is complete. In these cases, the completeness lemma cannot be applied to their branches and the completeness theorem does not hold. Consequently, the implemented logic is not complete.

3.3 Run time of the system

The plotted results of the system's average observed run time as a function of formula complexity are displayed in Figure 3.1, both for the production of formulas and their tableau solving. As it can be seen in the graph, as the formula complexity increases from zero to three, the run time needed for the system to complete both analysed tasks decreases. However, the opposite behavior is observed as the formula complexity increases from three to five since the run times of both tasks increases as well. For the tableau solving, formulas with complexities one to four have approximately the same average run time which fluctuates around 0.2 ms.

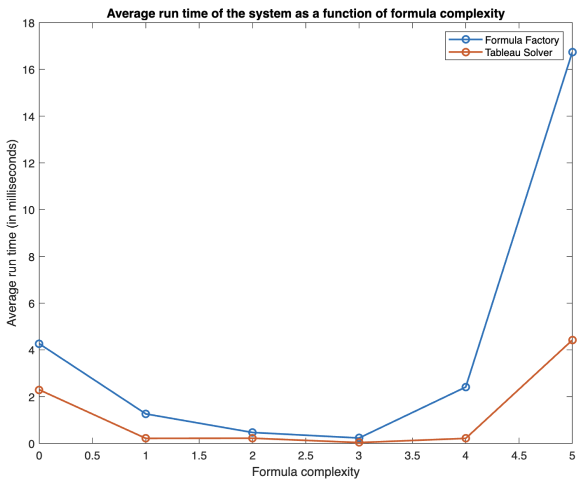

Figure 3.1: Graph showing the average run time per tested formula as a function of formula complexity, both for the formula factory and the tableau solver

3.4 Memory usage of the system

By letting the system run, new serializable files are being generated and saved to the local directory. For the first five formula complexities, the memory usage of the obtained serialized files is monitored. The resulting memory usage is shown in Table 3.1. It can be seen that the memory usage of each file as a whole is increasing exponentially as formula complexity increases. However, the average memory usage of each individual formula is about the same for most complexities, between 110 and 119 bytes, except for formulas that do not have any connective, for which the average memory usage per formula is 188.7 bytes.

Formula complexity	Number of formulas	Total memory usage	Average memory usage (bytes)
0	3	566 bytes	188.7
1	42	5 KB	119
2	840	98 KB	116.7
3	21273	2.5 MB	117.5
4	601860	66.3 MB	110.2

Table 3.1: Obtained memory usage of the saved generated formulas for formula complexities zero to four

3.5 Comparison with another temporal logic bot

Figure 3.2 displays the average run times obtained when running the same formulas on both O. de Vries and the current bot's systems. The run time obtained with this research's bot is consistently lower than the pre-existing bot's run times across all formula complexities.

In other respects, the tableau solvers' obtained run times were plotted separately for tautological and non-tautological tested formulas. The resulting plots are shown in Figures 3.3 and 3.4 respectively. Figure 3.3 reveals that the pre-existing logic bot solves tautologies' tableaux faster than the new designed bot. However, this research's bot is more robust when it comes to non-tautological formulas, as shown Figure 3.4.

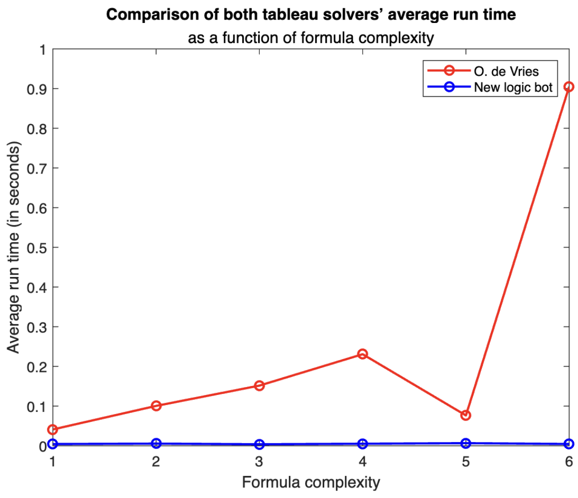

Figure 3.2: Graph showing the comparison of the average tableau solvers run time per tested formula as a function of formula complexity for both considered systems

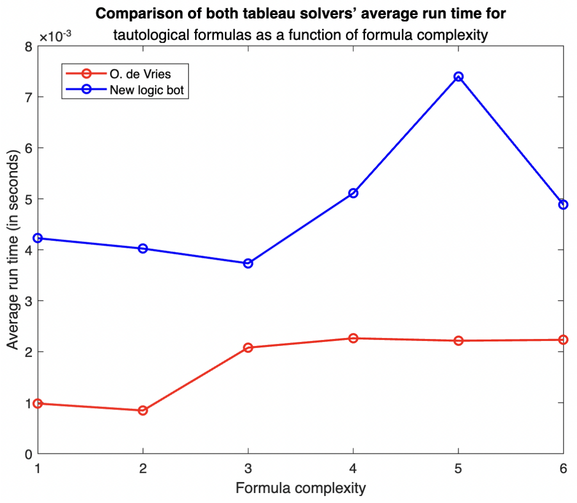

Figure 3.3: Graph showing the comparison of the average tableau solvers run time per tested tautology as a function of formula complexity for both considered systems

4 Conclusion

4.1 Discussion

In this research, a *Twitter* logic bot that regularly publishes tautologies in TL was designed. It makes use of the semantic tableau method. Based on the obtained results, the set up logic appeared to be sound but not complete. This defect is related to the implementation of the stopping condition meant to prevent infinite tableaux. For some tautologies, working out their tableau resembles working out an infinite tableau, only it eventually closes. In the design of this bot, the cutting off of branches

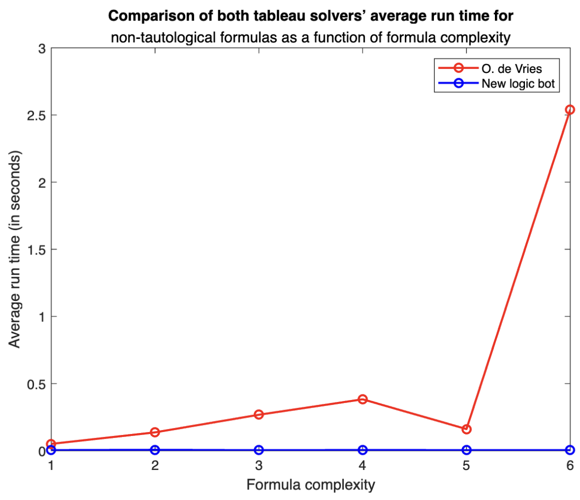

Figure 3.4: Graph showing the comparison of the average tableau solvers run time per tested non tautological formula as a function of formula complexity for both considered systems

does not take into account such case figures. Hence, some open but not complete branches are disregarded in a tableau that is considered complete. Still, infinite tableaux branches are also open but not complete. Implementing a limit on the number of formulas applied solves more problems than those it creates in the set up system. For that reason, cutting off branches is kept to the detriment of detecting some tautologies. As a result, the system at hand is not complete. Nonetheless, the tableaux of formulas that are not tautologies do not close. On that account, the designed system is sound.

The results obtained for the run time of the formula factory are that the average time needed to generate a single formula decreases as the number of connectives in the formula increases from zero to three. Generating formulas with complexity zero follows an exclusive process, as it does not build up on any pre-generated formula. For all other formula complexities, the generation of a single formula in the designed system is executed in one unique step, which is:

- For formulas that have as main connective a unary operator, the combination of the unary connective with a sub-formula retrieved from a serialized file
- For formulas that have as main operator a binary connective, the combination of the binary connective with two sub-formulas loaded from serialized files

The only element that varies in the formula factory from one formula complexity to another is the number of connectives that the loaded formulas have, not the number of steps needed to generate the current formula.

The number of generated formulas increases exponentially as the formula complexity increases. Therefore, the loading of a formula file with formula complexity n generates less formulas than the loading of the formula file with formula complexity $(n + 1)$. As a result, as the formula complexity increases, the number of newly generated formulas per single file loading increases. Since all formula production require the same number of steps, generating more formulas from a single file loading is what yields a decreased average production time.

Still, the run time increases as formula complexity increases from three to five. This observation is due to how the testing was performed. As explained in Section 2.5, for formula complexities zero to three, all possible TL formulas are included in the testing. However, after formula complexity three, only part of the possible TL formulas are considered. The way they have been selected is by only generating a portion of all possible formulas for each file loading combination. As a result of the control on the amount of formulas tested, the complete potential of every file loading is not exploited, which means that more file loading results in less formulas generated. This in turn decreases the average measured run time, although the formula generation requires the same single step as for lower complexities.

The results obtained for the tableau solver run time revealed a similar pattern as for the formula factory. The run time of the solver decreases as formula complexity increases from zero to three but then increases as formula complexity increases from three to five. Here again, the relation observed between the run time and formulas with complexities three to five is due to the way the testing has been performed. When generating formulas, the system always loops over all connectives in the same order, namely $\{\neg, G, H, F, P, \wedge, \vee, \supset, \leftrightarrow\}$. Hence, in all serialized files, the first fragment of formulas all have as main connective unary connectives. For formula complexities four and five, only part of the possible

TL formulas are tested. The tested formulas were collected by taking the first few generated formulas from all possible connective and lower complexity formula combinations. As a result, tested formulas inadvertently are predominantly constructed with temporal connectives. Temporal connectives are prone to generate infinite tableaux, which have a higher run time than non-infinite tableaux, yielding the observed results.
Formulas with complexities one to four have about the same average run time. Regardless of testing induced effects, the tableau solver seems to handle increasing complexities rather consistently.

The observed memory usage's exponential growth is related to the exponential growth of the number of formulas rather than to the increasing complexity of the formulas themselves. This can be deduced from the average memory usage per formula that remains more or less constant for all formula complexities. Nonetheless, formulas with complexity zero appear to be an outlier to this consistent average memory usage. This is due to the fact that a whole file is used to only store three formulas. For all higher order formulas, the memory usage of the empty file alone is averaged over a larger number of formulas.
The exponential growth of the memory usage is an issue for the continuous run of the program. That is, up until now, the reached serialized formula complexity is four. This file requires a memory usage equal to 66.3 megabytes. As this value increases exponentially with the formula complexity increase, formula complexity six will probably require a memory usage of 18 gigabytes and formula complexity eight, 7 tetrabytes. All resources in our current scope (local device/ online hosting application/ Raspberry Pi) have memory limits that will be reached early on. Nonetheless, the amount of formulas generated with formula complexities four and lower is already enough to keep the logic bot running on *Twitter* for years.

The current bot was also compared to a pre-existing logic bot (de Vries, 2018). The observed run time differences between the two tableau solvers are likely caused in part by the many dissimilarities of the two bots. The primary relevant difference is the fact that both bots have different aims. The tableau solver set up by de Vries (2018) aims at producing as output a human-like tableau, whereas this project only provides as output whether or not the tested formula is a tautology. As a result, the former has to follow all customary tableau steps while the latter has as objective to cut down as many steps as possible. This major difference induces various implementation differences that impact the run time of the system. For instance, the omission of duplicated nodes or branches additions in this designed bot helps reduce the complexity of the tableau. Also, the pre-existing bot provides as output the printed tableau tree. This additional action requires some non-negligible execution time.

Besides the different goals of the tableau solvers, the constraints that apply to the considered TL also differ, which alters the tautological property of some formulas. That is, adding restrictions to TL affects the modal interpretation which may either result in simpler tableaux in some cases, or more cluttered ones in others. Both compared systems are transitive, but the current bot is also reflexive, while the pre-existing bot is dense and non-branching towards the future and past. In some cases, these constraint differences yield different tableaux in the two systems for similar formulas, which also affects the run time.

Concerning the implementation of the systems, a first notable difference to discuss is the different programming languages used. *Java*, used in this research, is a compiled language while *Python*, used in de Vries (2018)'s research, is an interpreted language. This entails that *Java* has a shorter execution time than *Python*. Additionally, the implementation of the tableau itself in both codes is different. In the pre-existing bot, the tableau tree is represented in a list, while this research stores the tableau in a tree structure. A list structure allows for a more efficient tableau search as well as the storing of a larger number of formulas.

Apart from the system's differences, the performed testing experiment may have also interfered with the obtained results. For each formula complexity considered, fifteen formulas were tested. This amount is minimal, and tested formulas do not account for all of TL's formula variability. The testing of both systems should be performed multiple times on a wider range of formulas for more accurate results. Furthermore, the tested formulas were half methodically and half randomly selected. It was made sure that all different connectives would be featured in each list and that both tautologies and non tautologies would be represented. This formula monitoring may have also altered the results.

Looking at the run times obtained for tautological and non-tautological TL formulas separately in both systems, tautologies are solved more quickly in the pre-existing bot, while the tableau of formulas that are not tautologies are executed faster in the current bot. Regarding the tested tautologies, the pre-existing bot appears to be more efficient in their solving. Although, an important point to note it that the formulas represented on both traces are not identical. The differing restrictions applied to the considered TL, as discussed above, tweaks the tautological property of some formulas, which may interfere with the observed results.
The difference observed for non-tautological formulas may be due to the fact that the pre-existing logic bot takes care to provide a counter model for each formula that is not a tautology, which increases its run time.

The aim of this project was to create an efficient temporal logic *Twitter* bot. Overall, the goal has been achieved. Nevertheless, there are some temporal logic tautologies within the character limit of *Twitter* that are not detected due to an implemented stopping condition. The designed formula factory seems to be efficient for all formulas, regardless of the number of connectives they have. Yet, storing the generated formulas in serializable files will early on create memory issues. The tableau solver of the system seems to perform rather well and consistently for all formula complexities. Nonetheless, infinite tableaux remain a weakness in the system, both for its resulting inflated run time and its stopping condition that prevents the detection of some tautologies.

4.2 Future improvements

Building up on this system's drawbacks, finding a solution to solve the logic's completeness issue is a path that could be explored. For that, one idea would be to use as the tableau's data structure a list rather than a tree in the programming of this bot. Lists can handle more objects than a tree structure and they relate them more easily to one another.
Nevertheless, this change alone will probably not solve the completeness issue. Some tautologies may have extremely long tableaux that eventually close. Implementing a data structure with more capacity than the current one will not suffice to overcome this issue. Therefore, another path to explore would be the detection of patterns and unwritten rules of infinitely long tableaux that close. This pattern identification will not only help overcome the completeness problem, but will also help enhance the system's efficiency if it was to be generalized to any pattern, not only infinite tableaux tautology patterns, as it would reduce the system's run time.

Similarly, another improvement that could boost the system's performance is the detection of tautologies based on previously detected tautologies. For instance, if sentence S is a tautology, then $(S \vee P)$ will, for any TL sentence P, also be a tautology. Therefore, in an attempt to optimize the efficiency of the tableau, detected tautologies could be stored in a well planned way to later simplify other tableau solving.

The designed bot can be improved in various ways, but the last one that will be discussed here is taking parenthesis simplification into account. The current bot only handles one connective at a time. If it is presented a formula such as $(a \wedge a \wedge a)$, it is dealt with as $((a \wedge a) \wedge a)$. It thus requires the application of two rules in the tableau solving, whereas both applied rules are the same and both connectives have the same priority in the formula. That being said, both of these rules could have been applied at the same time. Taking this into account in the implementation could enhance the system's efficiency and decrease its run time.

Regarding the memory usage issue, an interesting route to take would be to find another way to store the TL formulas. Formulas build up on one another as the complexity increases, so rather than serializing each and every generated formulas, storing a code of some sort that indicates what the formulas are or how to generate them would be less memory consuming than serializing every generated formula, which would help overcome the memory problem.

References

de Vries, O. (2018). A Tableau Prover for Non-Branching Transitive Temporal Logic with Density as Constraint.

Goranko, V. and Rumberg, A. (2020). Temporal Logic. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, summer 2020 edition.

Nour, A. (2002). The Tableau Method for a Logical System Based on a Finite Poset. *Journal of Applied Non-Classical Logics*, 12(1):43–62.

Øhrstrøm, P. and Hasle, P. (1993). AN Prior's rediscovery of tense logic. *Erkenntnis*, 39(1):23–50.

Priest, G. (2008). *An introduction to non-classical logic: From if to is*. Cambridge University Press.

Rigolin, V. H. (2018). What is Twitter? How Do I Get Started? Why Should I Become a User? *Journal of the American Society of Echocardiography*, 31(3):A31–A32.

Smullyan, R. M. (1995). *First-order logic*. Courier Corporation.

A Appendix

Functions

Algorithm A.1 Formula Factory main function

```
resetNewGeneratedFormulasArray()
generateAtomicFormulas()
$running \Leftarrow true$
$complexity \Leftarrow 0$
while running do
  resetNewGeneratedFormulasArray()
  for $connective$ in $allConnectives$ do
    $connective$.generateFormulas(complexity)
  end for
  $complexity \Leftarrow complexity + 1$
  if $newlyGeneratedFormulas$ is empty then
    $running \Leftarrow false$
  else
    serializeFormulas()
  end if
end while
```

Algorithm A.2 Tableau Solver main function

```
$initialList \Leftarrow$ negatedFormula($formulaEvaluated$)
$initialList \Leftarrow initialList$.simplify()
$formulaPriorityQueue$.add($initialList$)
$nbOfFormulasApplied \Leftarrow 0$
$maxNbOfFormulasApplied \Leftarrow 3 * evaluatedFormulaComplexity$
$maxNbOfWorlds \Leftarrow 3 * evaluatedFormulaComplexity$
addReflexivity(0, tableau, tableauRoot)
while $tablea$ is $open$ do
  $formulaToBeWorkedOut \Leftarrow formulaPriorityQueue$.poll()
  $formulaToBeWorkedOut$.applyRule(tableau)
  if $formulaPriorityQueue$ is empty or $nbOfFormulasApplied \geq maxNbrOfFormulasApplied$
  then
    break
  end if
end while
```

B Appendix

Formula complexity	Tested formula	Pre-existing TL bot run time (in s)	New TL bot run time (in s)	Tautology or not
1	Ga	0.28340275	0.0061674	No
	Fa	0.000765	0.0044476	No
	Ha	0.3135175	0.0042859	No
	Pa	0.0069245	0.0041044	No
	$\neg a$	0.000681042	0.0041138	No
	$(a \wedge a)$	0.000508084	0.0040778	No
	$(a \wedge b)$	0.001256708	0.0040464	No
	$(a \vee a)$	0.000918417	0.0044007	No
	$(a \vee b)$	0.001708084	0.0047259	No
	$(a \supset a)$	0.000880166	0.0039921	Tautology
	$(a \supset b)$	0.000885542	0.0041164	No
	$(a \leftrightarrow a)$	0.001333875	0.0046232	Tautology
	$(a \leftrightarrow b)$	0.000907542	0.0041504	No
	$(b \supset a)$	0.002824875	0.0041662	No
	$(c \leftrightarrow c)$	0.000735084	0.0040619	Tautology
2	GGa	0.399813416	0.0235236	No
	HFa	0.211463708	0.0039147	No
	$F(a \supset a)$	0.000486209	0.003716	No/ Tautology
	$(a \supset Fa)$	0.000983833	0.0054645	No/ Tautology
	$H(a \leftrightarrow a)$	0.001301334	0.0036482	Tautology
	$(a \supset (b \supset c))$	0.001215958	0.0034926	No
	$((a \supset a) \vee a)$	0.000705792	0.0035933	Tautology
	$(a \vee \neg a)$	0.0008525	0.0040385	Tautology
	$(Hb \leftrightarrow a)$	0.882078458	0.0039676	No
	$((c \leftrightarrow b) \wedge a)$	0.001420166	0.0041176	No
	$((a \leftrightarrow b) \leftrightarrow a)$	0.002986792	0.0046537	No
	$\neg(a \supset b)$	0.002296292	0.0037432	No
	$(Ga \supset a)$	0.000880125	0.0038377	No/ Tautology
	FGa	0.000640792	0.0069486	No
	$(a \supset (b \supset b))$	0.00052725	0.0038592	Tautology
3	$(Fa \leftrightarrow Fa)$	0.001011208	0.0038923	Tautology
	$(Ga \supset Fa)$	0.001014	0.003623	No/ Tautology
	$\neg(a \wedge \neg a)$	0.004735	0.0040787	Tautology
	$((a \supset b) \vee (a \supset a))$	0.001095042	0.0038554	Tautology
	$(a \supset \neg Gb)$	0.000953542	0.0033967	No
	$((a \wedge b) \supset Fa)$	0.000836	0.0037709	No/ Tautology
	$G(a \supset (a \supset a))$	0.000570917	0.0035061	Tautology
	$((a \vee \neg a) \leftrightarrow a)$	0.00123075	0.0039924	No
	$G(a \vee (a \supset a))$	0.001186125	0.0038382	Tautology
	$(Ga \vee \neg a)$	0.107455042	0.0035095	No
	$(a \supset (Pa \vee a))$	0.005035875	0.0036246	Tautology
	$GPFa$	0.425878833	0.0035605	No
	$(a \leftrightarrow PFa)$	1.655997	0.0042457	No
	$(GPa \supset a)$	0.000709708	0.0036515	No
	$(a \leftrightarrow (Fa \vee b))$	0.217635209	0.0046622	No

Table B.1: Extensive list of tested formulas for the comparison of the two considered TL tableau solvers

Formula complexity	Tested formula	Pre-existing TL bot run time (in s)	New TL bot run time (in s)	Tautology or not
4	$(Fa \supset F(a \vee b))$	0.002173	0.0046645	Tautology
	$(a \supset PFFa)$	0.000718333	0.0047242	No/ Tautology
	$(Fa \supset PFa)$	0.211987958	0.0045287	No/ Tautology
	$(Ga \supset ((a \leftrightarrow b) \vee a))$	0.0007638340001	0.0048043	No
	$((a \leftrightarrow b) \vee F(a \leftrightarrow a))$	0.001166	0.005562	No/ Tautology
	$(((a \supset b) \wedge b) \vee (a \supset a))$	0.000553417	0.0047099	Tautology
	$((b \supset c) \vee ((b \supset a) \supset b)))$	0.000788542	0.0048435	Tautology
	$((a \leftrightarrow (b \leftrightarrow (c \leftrightarrow a))) \wedge a)$	0.0007945409999	0.0065805	No
	$(GGHa \supset a)$	0.001297041	0.0050846	No/ Tautology
	$G((a \leftrightarrow b) \vee Fa)$	1.5295665	0.0047819	No
	$((a \supset a) \wedge (b \supset (a \supset b)))$	0.0082965	0.0060501	Tautology
	$(a \vee (b \vee (c \vee \neg a)))$	0.00106675	0.0045642	Tautology
	$(Fa \leftrightarrow HFa)$	1.278789125	0.0056075	No/ Tautology
	$(\neg a \leftrightarrow \neg(a \wedge a))$	0.0006995000001	0.0058758	Tautology
	$(Pb \leftrightarrow (a \supset (c \supset b)))$	0.421957208	0.0052486	No
5	$((a \wedge Pa) \supset HFa)$	0.001693667	0.0284675	Tautology
	$((Fa \supset a) \vee (a \supset Fa))$	0.0014005	0.0094563	Tautology
	$((a \leftrightarrow (a \vee a)) \wedge (b \supset (b \wedge b)))$	0.0006356670001	0.0087552	Tautology
	$(HF(a \supset b) \vee \neg a)$	0.8383	0.0041991	No
	$\neg((a \leftrightarrow \neg a) \wedge (c \leftrightarrow b))$	0.001207625	0.003889	Tautology
	$((a \supset \neg a) \vee (c \supset (b \supset c)))$	0.001230875	0.005305	Tautology
	$((a \vee Pa) \supset FPa)$	0.176562375	0.0045275	No/ Tautology
	$(FPGHa \supset a)$	0.099983458	0.0042798	No/ Tautology
	$((a \supset HFa) \vee \neg a)$	0.006190917	0.0043958	Tautology
	$(a \leftrightarrow (b \leftrightarrow (c \leftrightarrow (\neg a \leftrightarrow b)))$	0.001237417	0.0054202	No
	$((a \supset (a \vee b)) \wedge (Gb \supset b))$	0.0006836670002	0.0039735	No/ Tautology
	$((b \vee \neg b) \supset (b \vee \neg b))$	0.004628834	0.0038597	Tautology
	$((a \wedge a) \leftrightarrow ((c \vee (b \vee a)) \wedge a))$	0.0007324580001	0.0044997	Tautology
	$((a \supset (\neg a \supset a)) \leftrightarrow (b \vee c))$	0.002900167	0.0039172	No
	$(a \vee (((Fa \supset a) \leftrightarrow c) \vee a))$	0.009698917	0.0043318	No
6	$((a \wedge \neg a) \vee \neg(a \wedge \neg a))$	0.000552709	0.0038286	Tautology
	$((a \supset (\neg a \supset a)) \leftrightarrow (b \leftrightarrow Hc))$	0.0077755	0.0049203	No
	$(a \vee (b \vee ((a \wedge Fa) \vee \neg a)))$	0.000863458	0.0036696	Tautology
	$(Ga \supset G(Fa \wedge Pa))$	0.735242875	0.0079102	No/ Tautology
	$(G(a \leftrightarrow b) \supset (Ga \leftrightarrow Gb))$	0.007246333	0.0056469	Tautology
	$(((a \leftrightarrow Ga) \vee (a \supset a)) \supset (b \wedge b))$	0.033017042	0.0043376	No
	$((a \leftrightarrow (a \leftrightarrow a)) \wedge \neg(a \wedge \neg a))$	0.00122675	0.0039502	No
	$\neg((b \wedge c) \leftrightarrow (\neg b \wedge \neg c))$	0.00137975	0.0040879	No
	$(a \vee (Fa \wedge (G(b \leftrightarrow c) \vee a)))$	1.857521333	0.0039533	No
	$(FGa \supset FGFa)$	0.001526583	0.0046507	Tautology
	$((a \supset (b \wedge a)) \wedge ((c \leftrightarrow b) \vee \neg b))$	0.001221958	0.0043341	No
	$(F(a \vee a) \supset (a \vee GFa))$	3.09420275	0.0040904	No
	$(G(c \supset c) \supset ((c \supset c) \wedge c))$	0.001147417	0.0038456	No
	$((Fa \wedge Fb) \supset F(a \wedge b))$	7.832815417	0.0046952	No
	$(\neg\neg a \supset (a \vee \neg\neg a))$	0.00097625	0.0035851	Tautology

Table B.2: Continuation of the extensive list of tested formulas for the comparison of the two considered TL tableau solvers