university of groningen / faculty of science and engineering

Efficiency of doxastic logic bot for serial transitive euclidean models

Bachelor's Project Thesis

P.B.C. Barho, S3788040, p.b.c.barho@student.rug.nl,
Supervisor: prof. dr. L.C. Verbrugge

Abstract: Doxastic logic is a modal logic concerned with the beliefs of agents. The doxastic logic KD45 is based on serial transitive euclidean models. This study is concerned with the performance of a bot, that was designed to post newly generated tautologies of KD45 on Twitter. The performance of the bot, depending on the complexity of the formulas, was evaluated with regard to its run time and RAM usage. Whilst the average run time per formula of the tableau solver was rather consistent, the RAM usage differed significantly with increasing numbers of connectives and modal depth. The limitations, such as a time-out to avoid infinite branches, were discussed to determine possible improvements in the implementation. These improvements include, e.g., a pattern checker, a dynamic number of agents, as well as an adapted approach to the data analysis.

1 Introduction

Everyone has knowledge or beliefs about some facts. What do I believe in? What do I already know to be true? When considering artificial agents, one has to formalize the notions of knowledge and belief in some way, for example, to treat them formally in philosophical debates. What does it mean if an agent believes something? And how can we formalize the knowledge or belief of different agents?

By using the notions of necessity and possibility of modal logic, one can express the notions of knowledge and belief of several agents using epistemic logic [1]. It can be used to formalize philosophical dilemmas and has a wide range of applications in multi-agent systems.

This research is concerned with doxastic logic, a modal logic that studies the belief of agents. A Twitter bot will be designed that posts newly generated tautologies of the doxastic logic $\mathrm{KD45}_n$ (where n stands for the number of agents). By analyzing the solving mechanism of the bot, its performance depending on the complexity of the formulas can be evaluated.

1.1 Epistemic logic

Epistemic logic is a modal logic that is concerned with the notions of knowledge and the belief of agents. It is based on the work of the Finnish philosopher Hintikka, who was the first person to reason about knowledge and belief using modal logic in 1962 [2]. While the knowledge of an agent is always true, belief need not be true. Belief is based on the incomplete knowledge base of an agent. The logic concerning belief is called *doxastic logic*.

The notions of necessity and possibility of modal logic can be used in an epistemic manner to form the modal operators of belief (B_a) and possibility (M_a) [1]:

- The belief operator B_a stands for the belief of agent a. The sentence $B_1\varphi$ stands for "Agent 1 believes that φ".
- The possibility operator M_a describes that agent a deems something to be possible. The sentence $\neg B_1 \neg \varphi$ can be expressed by $M_1\varphi$ which means "Agent 1 deems φ possible".

To formalize the semantics of doxastic logic, Kripke models are used. A model ($\mathbb{M}$) consists out of

- a non-empty set of states (S),
- a truth assignment (π) for the propositional atoms in those states,
- the accessibility relations (R_a) between the states (of agent a).

A Kripke world (w) consists of a model $\mathbb{M}$ and a state in S. If an accessibility relation exists for agent a from state i to state j in world ($\mathbb{M}$, i), it means that agent a considers world ($\mathbb{M}$, j) *possible*. Those worlds, such as j, are *epistemic alternatives* [1].

The logic KD45$_n$ is the most common logic used for belief (where n stands for the number of agents). It contains the following constraints on the accessibility relations (for $a \leq n$) [3]:

- R$_a$ is *transitive* if $\forall$i, j, k $\in$ S: (i, j) $\in$ R$_a$ and (j, k) $\in$ R$_a$ $\Rightarrow$ (i, k) $\in$ R$_a$.
- R$_a$ is *euclidean* if $\forall$i, j, k $\in$ S: (i, j) $\in$ R$_a$ and (i, k) $\in$ R$_a$ $\Rightarrow$ (j, k) $\in$ R$_a$.
- R$_a$ is *serial* if $\forall$i $\in$ S $\exists$ j $\in$ S (i, j) $\in$ R$_a$.

1.1.1 Axioms

The rules and axioms for the system KD45$_n$ are the following [1]:

- **R1:** $\dfrac{\varphi \quad \varphi \rightarrow \psi}{\psi}$ *(Modus ponens)*
- **R2:** $\dfrac{\varphi}{B_a\psi}$ *(Necessitation)*
- **A1:** All instances of propositional tautologies.
- **A2:** $(B_a\varphi \wedge B_a(\varphi \rightarrow \psi)) \rightarrow B_a\psi$
- **D:** $\neg B_a(\bot)$
- **A4:** $B_a\varphi \rightarrow B_aB_a\varphi$
- **A5:** $\neg B_a\varphi \rightarrow B_a\neg B_a\varphi$

The rules R1 and R2 denote that if their premises are proven to be true, their conclusion is proven to be true as well. Axiom A1 also applies to propositional tautologies involving epistemic formulas, such as B_ap $\rightarrow B_a$p. Axioms A2 denotes that the belief of an agent a is closed under logical consequence. Axiom D describes that an agent does not believe in contradictions, indicating that the knowledge base is not inconsistent. Moreover, axiom A4 means that an agent believes that he believes something (known as positive introspection). Lastly, axiom A5 denotes that an agent believes that he does not believe something (known as negative introspection) [1].

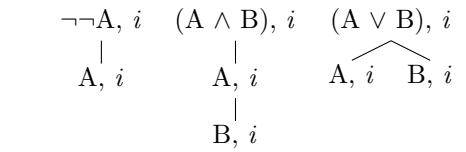
1.2 Tableaux

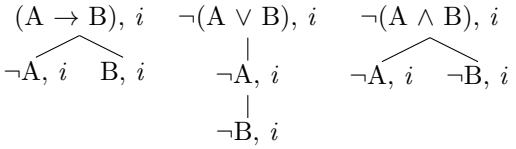
Semantic tableaux are a method of evaluating the validity of formulas in logic. The general idea is that the logical formula gets detangled step-by-step based on specific tableau rules [4]. The tableau rules depend on the corresponding logic and the truth tables of its connectives. In the case of KD45$_n$, they will be described in Section 1.2.1. If an inference has premises, the tableau starts with the main branch containing the premises and its negated conclusion and the state *0*. In the case of this research, it only focuses on singular formulas: The tableau starts with a negation of the formula for which we want to test validity and state *0*. If the application of tableau rules results in a contradiction on a branch in the state, the branch *closes*. If all the branches of a tableau *close*, the tableau *closes* and the formula is valid. This is because the negation of the starting formula is not satisfiable. If all tableau rules which apply to a branch have been applied, the branch is *complete*. If a tableau contains a *complete* and *open* branch (so it did not close), the formula is not a tautology [4].

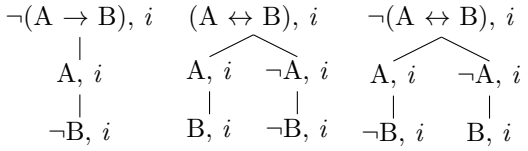
1.2.1 Tableau rules

A tableau rule is always applied to the main connective of a formula. If several tableau rules can be applied, the order of application is not important but can minimize the time of evaluation, as explained in Section 2.3. The tableau rules used in this research are based on the truth tables of each connective as well as the axioms of KD45$_n$ [4, 1].

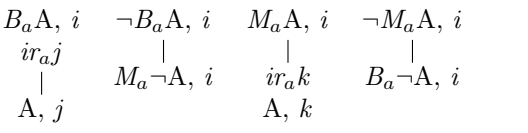
The tableau rules for the propositional logic operators negation ($\neg$), conjunction ($\wedge$), disjunction ($\vee$), implication ($\rightarrow$) and biconditional ($\leftrightarrow$) are the following, where $i \in \mathbb{N}$:







The doxastic logic operators belief (B_a) and possibility (M_a) are the following:

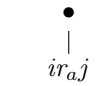


The rule for the belief operator (B_a) should be applied to all states j, such that the relation ir_aj appears on the branch. The rule for the possibility operator (M_a) should introduce a new state k on the branch.

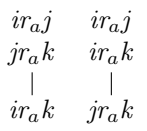
1.2.2 Relational constraints

Additionally to the tableau rules for operators, the tableau rules for the relational constraints have to be considered.

For the serial constraint, it applies to all states i on the tableau, that there exists another state j:



Here, j should be a new state. For the transitive and euclidean constraints, respectively, the tableau rules are as follows:



1.3 Twitter bots

Twitter is an online social network and microblogging service that was launched in 2006, based on the idea of Jack Dorsey [5]. Users of the platform can publish posts (so-called *tweets*) which can consist of a maximum of 280 characters [6].

Because of its popularity and a growing number of users, it has transformed into an information publishing venue which can be used by anyone for purposes such as advertisements, customer service, campaigning, et cetera [7].

Automated programs, called *bots*, can use an open application interface (API) provided by Twitter, to use the platform for automated posts of any kind. The Twitter API limits the number of *tweets* a bot can post in a specific time frame to avoid spamming, but with the limit of 300 tweets per three hours (for the standard API [8]) it still allows for great freedom for Twitter bots.

There exist several logic bots on Twitter that post tautologies of different logics. An example of an existing bot is @mathslogicbot, which posts tautologies of propositional logic [9].

1.4 Research question

The goal of this research is to design a doxastic logic bot that generates formulas and posts tautologies in KD45$_n$ on Twitter. The research question is therefore:

How does a doxastic logic bot, based on serial transitive euclidean models KD45$_n$, perform when solving theorems with increasing complexity?

The efficiency of the logic bot will be evaluated based on the used time for the execution as well as its memory usage.

2 Methodology

The code for this project contains three main components: The formula generator, the tableau solver, and the Twitter publisher. The formula generator generates formulas in the doxastic modal logic KD45$_n$, the tableau solver determines whether the formulas are tautologies by the use of tableaux, and the Twitter publisher posts the tautologies on Twitter.

Each of the components will be described in more detail in the following subsections. As each of the components should be able to run in parallel, the multi-threaded programming language `Java` was chosen. `Java` supports concurrency, moreover, it is understandable and structured through the use of classes. This allows the code to be easily extendable and reused for further research. The source code of this project is available under

`https://github.com/pbc-barho/doxlogicbot` [10].

2.1 Formulas

Formulas are stored in a binary tree structure where its main connective is the root and the children are either complex formulas or propositional atoms. If the main connective is unary (e.g., $\neg$, B_a, or M_a), only the left child is used. With the rest of the connectives, both children are used. Each formula also has a priority (see Table 2.1), which is based on the amount of additional computation it adds when its main connective is being solved. If the main connective is a negation, the priority depends on the main connective of its child (if it is not a propositional atom). As an example, connectives that lead to splitting the branch have the lowest priority, whilst formulas that only lead to additional formulas on the branch have a higher priority. The priority determines in which order the formulas get solved by the tableau solver.

Priority	**Connectives**
1	$\neg\neg$
2	$\neg\vee$, $\wedge$, $\neg\rightarrow$
3	$\neg M_a$, B_a
4	$\neg B_a$, M_a
5	$\neg\wedge$, $\neg\leftrightarrow$, $\leftrightarrow$, $\rightarrow$, $\vee$

Table 2.1: The priorities of formulas based on their main connective, sorted from highest priority (1) to lowest priority (5).

2.2 Formula generator

The formula generator is responsible for generating formulas in KD45$_n$ of varying complexities, which can then be solved by the tableau solver (as described in Section 2.3). The general implementation was inspired by Thalia Najjar's Bachelor thesis of 2021 [11].

The formula generator aims to start generating the least complex formulas and then progressively increasing the complexity by combining all existing formulas with the connectives. The pseudocode of the formula generator is given in Appendix A.1. In Table 2.2, all connectives, propositional atoms, and agents are listed which are used for the formula generation.

Connectives	$\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, B_a, M_a
Propositional atoms	p, q, r
Agents	1, 2, 3

Table 2.2: The connectives, propositional atoms, and agents used by the formula generator to generate formulas of varying complexities.

First, atomic formulas are generated, which are stored in a `ser`-file. Then, the generator goes through all the connectives and combines the atomic formulas with each connective in all possible combinations using each connective's `generateFormulas` method. The newly generated formulas get stored again, and the generator repeats the process by going through the connectives again. This process goes on until no new formulas with a length of below 280 characters get added. This character limit is based on the character limit for Twitter posts.

The length of a formula (len) is based on its printed length including whitespace and brackets. Table 2.3 defines the length of a formula based on its connectives. With each binary connective the length gets increased by five, as it also adds white space, as well as brackets, to the printed formula. Because there are no outer brackets necessary when the main connective of a formula is a binary connective, they do not get printed and the length gets deducted by two.

$$\begin{aligned} \text{len(p)} &= 1 \\ \text{len}(\neg\text{A}) &= \text{len(A)} + 1 \\ \text{len}(\text{A} \otimes \text{B}) &= \text{len(A)} + \text{len(B)} + 5 \\ \text{len}(B_a\text{A}) &= \text{len(A)} + 2 \\ \text{len}(M_a\text{A}) &= \text{len(A)} + 2 \end{aligned}$$

Table 2.3: The length of a formula (len) based on its connectives, where $\otimes$ stands for any binary connective and p refers to a propositional atom.

The modal depth of a formula (md) is used to describe the nesting of the modal operators of a formula. It is defined based on the deepest nested modal operators in a formula, according to Table 2.4 [12].

As the logic bot should not post the same formula twice, duplicated formulas have to be avoided. For

md(p)	=	0
md(¬A)	=	md(A)
md(A ⊗ B)	=	max(md(A), md(B))
md(B_aA)	=	md(A) + 1
md(M_aA)	=	md(A) + 1

Table 2.4: The modal depth of a formula (md) based on its connectives, where ⊗ stands for any binary connective and p refers to a propositional atom [12].

this, two extra steps were implemented. Firstly, before the generated formulas get stored, the generator checks whether the same formula already exists in the `ser`-file. Secondly, formulas with symmetric connectives, such as ∧, ∨, and ↔, are logically equivalent if their right and left formulas are exchanged. Therefore, the formula generation is limited such that formulas with these symmetric connectives that are logically equivalent (e.g., p ∧ q and q ∧ p) are not both generated.

2.3 Tableau solver

Once the formula generator has generated the first formulas, the tableau solver chooses a random formula of the file and begins evaluating it. Since the generated formulas are reused to create formulas of higher complexities, a copy of the formula file is used, which the tableau solver accesses. This way, a formula can be removed from the file once it is evaluated, without influencing the formula generator in any way.

The tableau solver negates the chosen formula and forms a tableau with it using a `Tableau` class. It then solves the tableau depth-first by evaluating each branch after the other. The solving mechanism of a branch is described in Section 2.3.1.

The state of a formula is stored as an integer in the `tableauFormula`-class. The initial formula starts in state 0, and each added state on the tableau increases the integer by one. Agents of a formula are not stored as natural numbers (as explained in Section 1), but as instances of an `agent`-class. This way, the agent-specific relations between states can be stored in the corresponding `agent`-class itself. Each instance of an agent has a name, that corresponds to a natural number (the formula generator is limited to the three agents, 1, 2, and 3, as described in Section 2.2).

If a branch is open and complete, the solver stops as the formula is not a tautology and the solver chooses a new formula to solve. If the branch is closed, it begins to solve the next branch. The branches are stored in a `branches` list, such that the solver can easily remove one and begin solving the next one. Once the `branches` list is `empty`, the solver stops, and the formula gets saved in a `tautologies` file, since an empty list means that all the branches are closed and therefore the tableau is closed.

This `tautologies` file can be accessed by the Twitter publisher which is described in Section 2.4.

2.3.1 Branch solver

Before the branch gets solved, a timer is started. This timer stops the solving of the branch after ten seconds to avoid infinite branches. Once a potentially infinite branch is found, the tableau solver stops as the formula is probably not a tautology. This might lead to a wrong conclusion if the solving of the branch gets stopped even though the branch is not infinite, but it avoids the solver to get stuck in a loop.

The solving of a branch runs in a loop until the branch is either stopped by the timer, closed, or open and complete. The pseudocode of the branch solver can be found in Appendix A.2. Each loop starts by pulling the formula with the highest priority of the `leftOverFormulas` priority queue. This queue sorts the formulas based on their priority as described in Section 2.1. Therefore, always one of the formulas with the highest priority gets dequeued first.

The chosen formula (called `currentFormula`) gets evaluated using its tableau rule by calling its method `applyRule`. Afterward, the method `isClosed` checks whether the tableau is closed by comparing whether a formula and its negation exist in the same state on the branch. This is done by using the `negatedFormulasOnBranch` and the `formulasOnBranch` lists, that contain all the negated and non-negated formulas on the branch (including their states), respectively. The lists also contain the complex formulas on the branch, such that also any complex formula and its negation lead to the closing of the branch.

The application of a tableau rule might lead to additional formulas to be added to

the `leftOverFormulas`, `formulasOnBranch`, or `negatedFormulasOnBranch` lists. If a tableau rule leads to a splitting of the branch, a copy of the current branch is added to the `branches` list of the tableau, and each part of the resulting new formulas is added to the current branch and the copied branch, respectively.

The agent-specific accessibility relations get checked by using an `agents` list and adding new accessibility relations to them based on the constraints of $KD45_n$ (seriality, transitivity, and euclideanicity).

If a tableau rule of a formula can be applied several times, such as the rule for the belief operator, the formula gets stored in an `infFormulas` list. In every iteration of the solver-loop, the branch solver checks whether any of the formulas in the `infFormulas` list can be applied (e.g., when there were new relations added to the branch). If that is the case, they get applied using the `applyNewRelations` method.

After these steps, the loop starts again by choosing a new formula from the `leftOverFormulas` priority queue.

2.4 Twitter publisher

The Twitter publisher is responsible for posting the tautologies on Twitter, as the name suggests. It uses the `Java`library Twitter4J, which can access the Twitter application programming interface and is able to automatically post a tautology every two hours on the Twitter account named "doxlogicbot" [13].

To do so, it accesses the `tautologies` file containing the tautologies which were added by the tableau solver. It chooses the first tautology of the file, removes it, and posts it using the `getString` method. Because the tableau solver solves the generated formulas in a random order, the tautologies which will be posted will be in a random order as well. After posting, the Twitter publisher sleeps for three hours using the `sleep` method of the `Thread` library.

The Twitter publisher runs in parallel with the formula generator and the tableau solver and posts the tautologies independently of them with the use of threads. The code runs indefinitely on a server to ensure that it will eventually post all the tautologies of up to 280 characters.

2.5 Data acquisition

To answer the research question (see Section 1.4), the performance of the logic bot is evaluated for solving theorems of increasing complexity. This is done in two ways: By timing the execution of the tableau solver and by determining the memory usage of the solver. Both measurements will be analyzed per complexity of formulas, which is measured by the modal depth and the number of connectives.

The measurements are conducted on a system with a 2,3 GHz Quad-Core Intel Core i7 processor with 16 GB DDR4 memory operating at 3733 MHz. These specifications have to be taken into account when analyzing the obtained data.

As mentioned before, the tableau solver's run time as well as its RAM usage are analyzed per complexity of the formulas. Therefore, one has to take the percentage of tautologies of the total number of formulas per complexity into account. This percentage will be measured by counting the number of generated formulas, as well as the number of tautologies per complexity.

As the number of generated formulas per complexity grows exponentially, the measurements for all the formulas of higher complexities would take several days, which is not realizable on a personal computer. Therefore, a specific number of formulas, that are randomly generated, is analyzed. The specific number of analyzed formulas can be seen in Tables 2.5 and 2.6.

Nr. of connectives	Nr. of formulas
1	48
2	286
3	286
4	286
5	286
6	286

Table 2.5: The number of tested formulas per complexity categorized by the formulas number of connectives.

3 Results

The performance of the tableau solver was evaluated as mentioned previously in Section 2. The raw

Modal depth	Nr. of formulas
0	184
1	489
2	439
3	241
4	98
5	27
6	3

Table 2.6: The number of tested formulas per complexity categorized by the formulas modal depth.

data and the code used to evaluate the performance can be found on the GitHub repository containing the source code [10].

The average values and standard deviations for the run time of the tableau solver per formula complexity as well as the RAM usage can be found in Appendix B. Tables B.1 and B.2 contain the data per number of connectives for tautologies and non-tautologies, respectively. Tables B.3 and B.4 contain the data per modal depth for tautologies and non-tautologies, respectively.

As the RAM usage should be analyzed per formula complexity, the RAM usage for solving propositional atoms is used as a baseline. This baseline indicates the minimum RAM usage that is required for the tableau solver, without having to apply any tableau rules. Therefore, the difference between the RAM usage for solving propositional atoms and the RAM usage for solving formulas of higher complexities is given in the tables as well.

The tableau solver was stopped after a maximum run time of ten seconds, as mentioned in Section 2.3.1. Therefore, any data points that exceeded a run time of ten seconds were excluded from the averages, as they are outliers. The data points per complexity were plotted in Figures 3.1 and 3.3 to visualize any outliers.

The outliers that are visible in Figures 3.1 and 3.3 were removed using the interquartile range of the data set. How many of the tested formulas per formula complexity were outliers is shown in Tables B.5 and B.6. The code that was used to determine the outliers can be found on GitHub [10]. As the modal depth of 3 only contains 14 data points, the data points that were considered outliers during the analysis of the interquartile range of the number of connectives were removed additionally. The data sets with removed data points can be seen in Figures 3.2 and 3.4.

The number of data points per complexity and whether it is a tautology can be seen in Table B.7 in Appendix B.

Furthermore, the average observed run time including its standard deviation was plotted per formula complexity, which can be seen in Figures 3.5 and 3.6, for the number of connectives and modal depth, respectively. Similarly, the average RAM usage was plotted per formula complexity, which can be seen in Figures 3.7 and 3.8, for the number of connectives and modal depth, respectively.

It can be observed that the average run time per number of connectives stays rather consistent for each formula complexity when taking the standard deviation into account. The average run time and its standard deviation increase for the formulas with six connectives as compared to lower complexities. The RAM usage per number of connectives first increases and then decreases slightly from the complexities of two to four connectives. The standard deviation of the RAM usage per number of connectives stays approximately the same.

Furthermore, the standard deviations of the run time of formulas that are not tautologies are slightly bigger than those of the formulas that are tautologies. Lastly, the number of outliers (as can be seen in Table B.5) increases drastically with an increasing number of connectives. Therefore, the averages of the run time, as well as the RAM usage, are taken from fewer formulas (with increasing complexity).

When considering the average run time per modal depth, it is visible that the run time stays around 7 ms. The standard deviation of the run time for non-tautologies is greater than the standard deviation for tautologies. The standard deviation of the run time for a modal depth of 3 is drastically smaller than for the other modal depths. The standard deviations per modal depth are larger than the standard deviations per number of connectives, indicating great variation between the data points (as can be seen in Figure 3.6). The data with a modal depth of 5 or 6 was not plotted, as the complete data sets timed out (see Table B.6).

The average RAM usage per modal depth stays consistent at around 450 kB. Its standard deviation decreases per modal depth until a modal depth of 3.

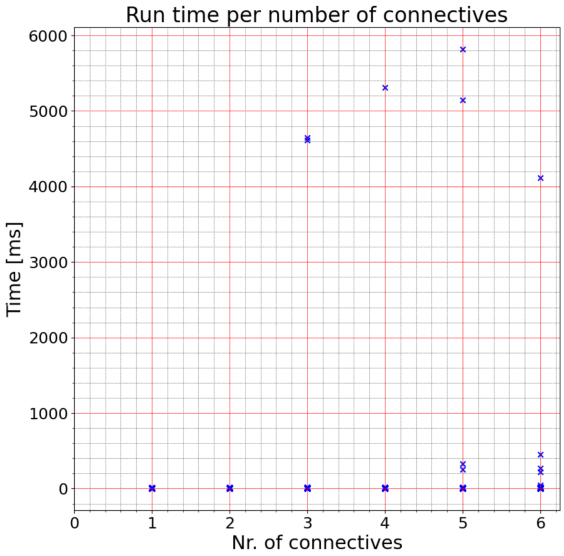


Figure 3.1: Scatter plot of the run time per formula complexity (measured by the number of connectives).

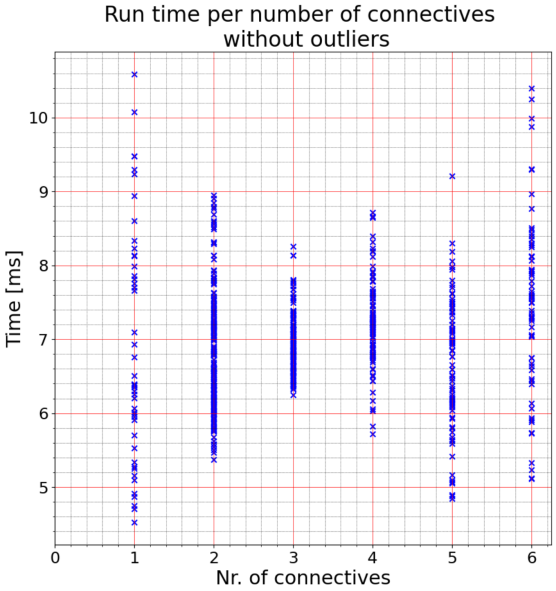


Figure 3.2: Scatter plot of the run time per formula complexity (measured by the number of connectives) with removed outliers.

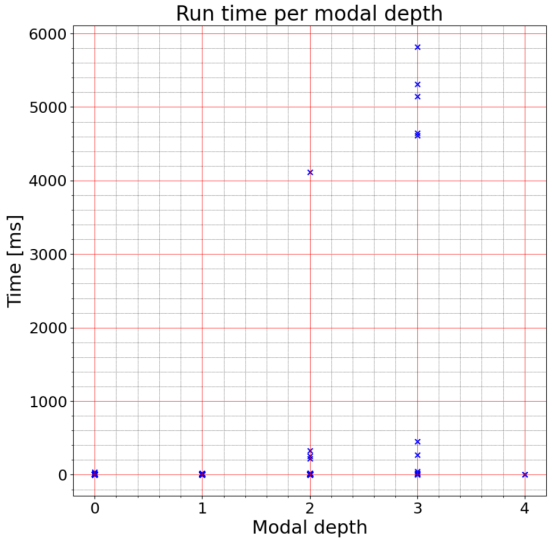


Figure 3.3: Scatter plot of the run time per formula complexity (measured by the modal depth).

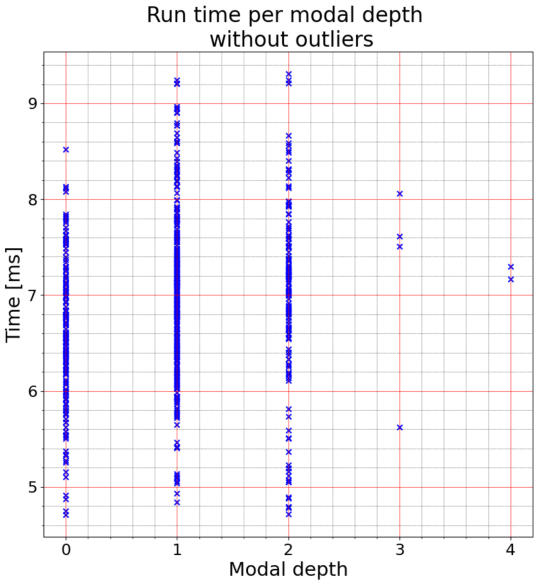


Figure 3.4: Scatter plot of the run time per formula complexity (measured by the modal depth) with removed outliers.

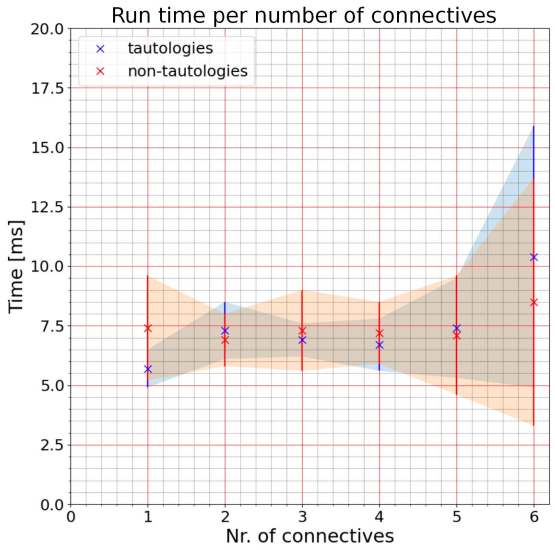


Figure 3.5: Average run time in ms per number of connectives with its standard deviation, plotted for tautologies and non-tautologies.

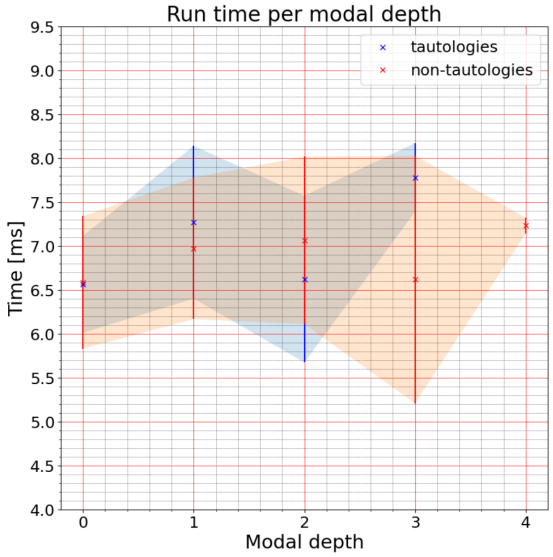


Figure 3.6: Average run time in ms per modal depth with its standard deviation, plotted for tautologies and non-tautologies.

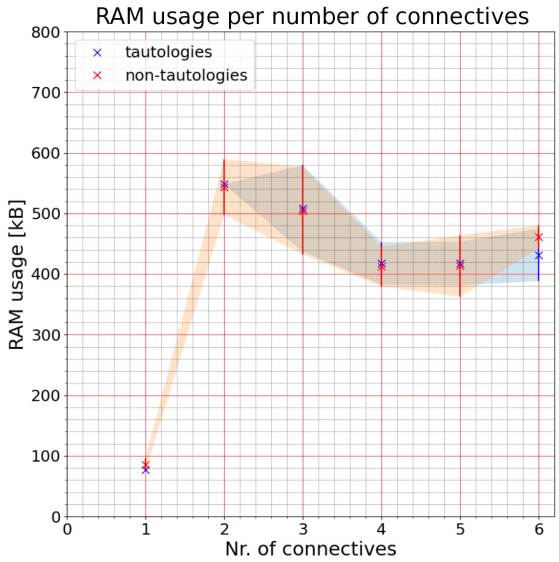


Figure 3.7: Average RAM usage in kB per number of connectives with its standard deviation, plotted for tautologies and non-tautologies.

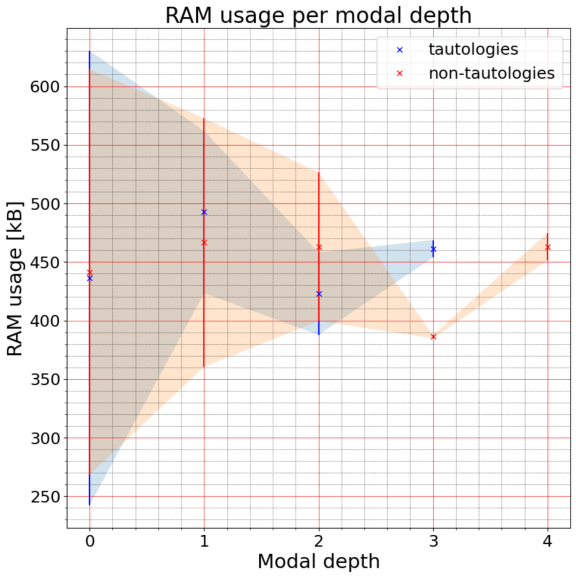


Figure 3.8: Average RAM usage in kB per modal depth with its standard deviation, plotted for tautologies and non-tautologies.

Then it increases slightly again for a modal depth of 4 (as can be seen in Figure 3.8). Similarly to the run time, the standard deviation of the RAM usage per modal depth is slightly smaller for tautologies than for non-tautologies.

Generally, the number of data points per complexity for tautologies is significantly less than that for non-tautologies, as can be seen in Table B.7.

3.1 Twitter Bot

The Twitter bot can be observed on the Twitter account named "doxlogicbot" [7]. An example of a tweet can be seen in Figure C.1 in Appendix C. The GitHub repository that contains the source code also contains the csv-files that contain the raw data of the randomly generated formulas [10].

4 Discussion

This project aimed to design a doxastic logic bot based on serial transitive euclidean models $KD45_n$ and to analyze how it performs when checking the validity of constructed formulas with increasing complexity. The complexity of the formulas was determined in two different ways: as the number of connectives and as the modal depth. The efficiency of the bot was evaluated based on its run time, as well as its memory usage.

4.1 Run time analysis

When the formulas were analyzed per number of connectives, the average run time was consistent up to four connectives and increased for more connectives (see Figure 3.5). The standard deviation for tautologies was slightly smaller than the standard deviation for non-tautologies and they increased from four connectives upwards. This could be explained by the increasing number of rule applications that are necessary to solve the tableaux of formulas with higher complexities. However, considering the consistent average run time, the solver seems to be able to solve formulas with different numbers of connectives rather consistently.

When looking at the analysis of the formulas per modal depth, it is visible that the run time is varying a lot more in comparison to the number of connectives (see Figure 3.6). This might be based on the smaller data sets for modal depths of three to four. When looking at Table B.7, it is visible that there are only very few formulas for those greater modal depths, so the run time of one formula affects the average run time for that complexity tremendously. This would also explain the decreasing standard deviations for the greater modal depths.

Generally, the standard deviations are greater per modal depth than per number of connectives. This could be related to the application of the seriality, transitivity and euclideanicity restrictions. If many restrictions have to be applied or many agents are included in the formula, the solver needs a lot more time, which is in contrast with a formula that closes early.

4.2 RAM usage analysis

The RAM usage first increased significantly from one to two connectives and then decreased slightly again (see Figure 3.7). This steep increase in RAM usage can be linked to the use of formula applications. One connective only requires the application of one rule, where the literals can directly be used to arrive at a conclusion. For two connectives upwards, the intermediate formulas have to be solved, which might explain the increase in RAM usage.

That the RAM usage decreases again slightly from three connectives onwards could be due to the number of formulas that were considered outliers. In Table B.5 it is visible that the number of formulas with two or three connectives that were analyzed was significantly higher than with a higher number of connectives. Therefore, it is likely that the formulas that would have had a greater run time (which would result in a higher average run time) were timed out due to the time limit. Further research would be necessary to exclude that possibility.

The RAM usage analyzed per modal depth (see Figure 3.8) is varying the most out of the measurements. A modal depth of zero can include any number of connectives (without modal operators), so that explains the large standard deviation. A modal depth of zero to one contains the low RAM usage that is measured for one connective (see Figure 3.7), so this increases the standard deviation. Furthermore, the very small standard deviation of the RAM usage for formulas with a modal depth of three to four can, again, be explained by the small

number of formulas with that complexity (see Table B.7).

4.3 General analysis

The small number of data points for some complexities can be explained by the number of generated formulas and the outlier detection. The number of generated formulas per number of connectives was 286, which resulted in an uneven distribution of the numbers of formulas generated per modal depth (see Table 2.5 and 2.6).

The time out that was implemented to avoid infinite branches mostly affected formulas of greater complexities, especially greater modal depths. Therefore, the number of tested formulas decreased significantly (see Tables B.5 and B.6)). That formulas with greater modal depths timed out more often might not necessarily mean that they always led to infinite branches, but the checking and application of restrictions can take a long time if many worlds and agents are involved.

Additionally, the reduced number of formulas affected the outlier detection using the interquartile range. If there are fewer data points, the single measurements affect the range more, which leads to the exclusion of fewer formulas. This is what happened for the data points of a modal depth of three (see Figure 3.3). They were considered outliers when they were analyzed per number of connectives (see Figures 3.1 and 3.2), but not when they were analyzed per modal depth. To allow for a coherent analysis they were still excluded (as described in Section 3), which led to Figure 3.4.

The analysis for the measurements of tautologies is based on fewer formulas than those that are not tautologies (see Table B.7). This might lead to an inaccurate impression of the performance of the tableau solver when comparing tautologies with non-tautologies. Therefore, the difference in the standard deviations of the run time and RAM usage between the tautologies and non-tautologies could be misleading.

Lastly, the formula generator was bound to using maximally three propositional atoms and three agents. Therefore, many formulas, especially those of higher complexities, could not be generated and analyzed. Similarly, the generation of symmetric formulas that are logically equivalent was restricted (as described in Section 2.2) but not completely avoided (e.g., $p \wedge (p \wedge q)$ and $q \wedge (p \wedge p)$ are still included).

Due to the exponential growth of generated formulas, the execution of the formula generator requires to load an increasing number of formulas with lower complexities to generate new ones. Whilst the performance of the formula generator was not evaluated as a part of this research, it did result in issues regarding the memory limitations of the local device and server that the code was run on. A maximal heap memory use was defined to avoid any execution errors, but this might affect the run time of formulas with higher complexities (that were not evaluated as part of this paper). Whilst this does not form an issue for the bot, as the number of tautologies of lower complexities is enough to let the bot run for years, it has to be taken into account. Therefore, due to the length of the formulas that were evaluated as part of this research, the implemented character limit for the Twitter posts was not relevant.

4.4 Possible improvements

When taking all of these aspects into account, several improvements of the project could be implemented. Firstly, the data analysis could take into account the number of formulas per complexity that time out, and include a more even distribution of formulas that are used for the analysis. This way, the averages and standard distributions per complexity would be more comparable to each other as the different percentages of tautologies and the number of formulas per complexity affected the error analysis of the data.

Secondly, a dynamic number of agents and propositional atoms could be used, as the current implementation of the formula generator excludes any formulas with more than three agents or propositional atoms. If the number of agents and propositional atoms would increase according to the number of connectives, all possible formulas could be generated. A similar approach was used for another logic bot [14].

With increasing complexities (especially when measured by the modal depth) an increasing number of formulas timed out due to the ten seconds time limit. Whilst the time limit did avoid solving infinite branches, it also stopped the solving of formulas that might have been solvable. Espe-

cially formulas with greater modal depths timed out due to the extensive checking of applicable restrictions in the code. Therefore, avoiding a timer and implementing another approach to avoid infinite branches would be beneficial. One possibility would be to use a pattern checker to detect any repeating patterns whilst solving a branch. In the case of an infinite branch, the repeating pattern of applied rules could be detected and the solver could be stopped.

Other improvements that are worth mentioning are reusing the results of previously solved formulas, using intermediate strategies regarding the application of restrictions (replicating the solving strategies of human logicians), and optimizing the code through, e.g., using lists instead of serialized files, or using a different data structure than a tree to store the tableaux. These implementations could improve both the memory usage, as well as the run time of the bot, depending on their implementation (which requires further research).

4.5 Further research

One possibility for further research would be to implement another solving strategy, such as a breadth-first search instead of a depth-first search. By using threads several branches could be evaluated simultaneously. This way, the solver could stop after finding one open and complete branch, which might result in faster solving times per formula.

Furthermore, the notion of common beliefs or implicit beliefs could be added to the system. Especially for greater numbers of agents it would be interesting to see how this implementation would affect the logic bot. The beliefs of several agents could be combined and result in new beliefs, even if no agent individually believes that fact [15]. An expansion of this logic bot with other epistemic logic subsystems, e.g., S5, would be a further possibility that could be explored.

Lastly, a comparison to the performance of other logic bots would be an interesting addition to the research. Especially comparing different solving strategies and programming languages could provide an interesting insight into the dependence of the performance of tableau solvers and logic bots in general.

5 Conclusion

This project consisted of implementing a doxastic logic bot based on serial transitive euclidean models $KD45_n$ and analyzing its performance when checking the validity of constructed formulas with increasing complexity. The development of the doxastic logic bot was succesful, but its performance partly depends on the complexity of the formulas, especially when considering the modal depths. However, some complications influenced the performance and analysis of the bot, such as the implemented time limit to avoid infinite branches as well as the limited number of formulas used for the analysis. Several aspects could be improved upon for further research such as using a pattern checker to detect infinite branches, using different solving strategies, and using a dynamic number of agents and propositional atoms to generate formulas.

References

[1] J. Meyer and W. van der Hoek, *Epistemic Logic for AI and Computer Science*, ser. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2004. [Online]. Available: https://books.google.com.pa/books?id=WgXyLRyEO8AC

[2] J. Hintikka, *Knowledge and Belief: An Introduction to the Logic of the Two Notions.* Ithaca: Cornell University Press, 1962.

[3] J. Y. Halpern, "The relationship between knowledge, belief, and certainty," *Annals of Mathematics and Artificial Intelligence*, vol. 4, no. 3-4, p. 301–322, 1991.

[4] G. Priest, *An Introduction to Non-Classical Logic: From If to Is.* Cambridge University Press, 2008.

[5] J. Burgess and N. K. Baym, *Twitter: A Biography.* New York University Press, 2020. [Online]. Available: https://doi.org/10.18574/nyu/9781479841806.001.0001

[6] A. Rosen, "Tweeting made easier," Nov 2017. [Online]. Available: https://blog.twitter.com/official/en_us/topics/product/2017/tweetingmadeeasier.html

[7] Z. Chu, S. Gianvecchio, H. Wang, and S. Jajodia, "Detecting automation of twitter accounts: Are you a human, bot, or cyborg?" *Dependable and Secure Computing, IEEE Transactions on*, vol. 9, pp. 811–824, Nov 2012.

[8] Twitter, "Rate limits: Standard v1.1." [Online]. Available: https://developer.twitter.com/en/docs/twitter-api/v1/rate-limits

[9] M. Scroggs, "Logic Bot," Nov 2014. [Online]. Available: https://twitter.com/mathslogicbot

[10] P. Barho, "Github repository 'doxlogicbot'," 2021. [Online]. Available: https://github.com/pbc-barho/doxlogicbot

[11] T. Najjar, "Temporal logic bot with reflexivity and transitivity as constraints," BSc project, University of Groningen, Jul 2021.

[12] L. A. Nguyen, "Constructing the least models for positive modal logic programs," *Fundamenta Informaticae*, vol. 42, no. 1, p. 29–60, 2000.

[13] P. Barho, "KD45 Doxastic Logic Bot," 2022. [Online]. Available: https://twitter.com/doxlogicbot

[14] G. A. Zumel, "Intuitionistic logic bot," BSc project, University of Groningen, Jul 2021.

[15] R. Verbrugge, "BOK Project, Unit 2," Logics for Artificial Intelligence, 2019.

A Pseudocode

Algorithm A.1: Formula generator main algorithm

```
initializeConnectives();
addAtoms();
running ← true;
count ← 0;
while running do
    oldSize ← generatedFormulas.size();
    addFormulas();
    count ← count + 1;
    if count is 1 then
        tableauSolver.start();
    end
    if generatedFormulas.size() is oldSize then
        break ;                          /* All new formulas are above 280 characters */
    end
end
```

Algorithm A.2: Solving algorithm for branches

```
Result: Branch is closed
leftOverFormulas.add(negatedFormula);
agents ← negatedFormula.getAgents();
worlds ← negatedFormula.getStatus();
startTime ← System.currentTimeMillis();
while leftOverFormulas is not empty and elapsedTime < 10 seconds do
    currentFormula ← leftOverFormulas.poll();
    if currentFormula.getRoot() is not leaf then
        currentFormula.getRoot().applyRule();
    else
        formulasOnBranch.add(currentFormula) ;              /* The formula is atomic */
    end
    if isClosed() then
        return true ;                                        /* The branch is closed */
    end
    checkRelationsPerAgent();
    foreach infFormula ∈ infFormulas do
        infFormula.applyNewRelations()
    end
end
return leftOverFormulas is not empty ; /* The branch is open & complete or infinite */
```

B Tables

Nr. of connectives	Run time [ms]	RAM usage [MB]	RAM usage difference [kB]
1	5.7 ± 0.8	3.14 ± 0.00	76.5 ± 0.0
2	7.3 ± 1.2	3.61 ± 0.00	548.1 ± 0.2
3	6.9 ± 0.7	3.57 ± 0.07	508.0 ± 71.6
4	6.7 ± 1.1	3.48 ± 0.04	417.4 ± 35.0
5	7.3 ± 2.1	3.48 ± 0.04	417.3 ± 36.2
6	10.4 ± 5.5	3.54 ± 0.04	431.5 ± 43.2

Table B.1: The average run time and RAM usage per number of connectives of the tableau solver using depth-first search for tautologies.

Nr. of connectives	Run time [ms]	RAM usage [MB]	RAM usage difference [kB]
0	8.3 ± 7.3	3.06 ± 0.01	-
1	7.4 ± 2.2	3.15 ± 0.01	85.3 ± 10.6
2	6.9 ± 1.1	3.61 ± 0.05	543.5 ± 45.9
3	7.3 ± 1.7	3.57 ± 0.07	504.3 ± 72.4
4	7.2 ± 1.3	3.48 ± 0.04	412.4 ± 32.8
5	7.1 ± 2.5	3.48 ± 0.05	413.1 ± 50.6
6	8.5 ± 5.2	3.52 ± 0.02	460.8 ± 19.0

Table B.2: The average run time and RAM usage per number of connectives of the tableau solver using depth-first search for non-tautologies.

Modal depth	Run time [ms]	RAM usage [MB]	RAM usage difference [kB]
0	6.6 ± 0.6	3.50 ± 0.19	436.3 ± 193.9
1	7.3 ± 0.9	3.56 ± 0.07	492.6 ± 69.2
2	6.6 ± 0.9	3.49 ± 0.04	422.9 ± 35.1
3	7.8 ± 0.4	3.52 ± 0.01	461.3 ± 7.4

Table B.3: The average run time and RAM usage per modal depth of the tableau solver using depth-first search for tautologies.

Modal depth	Run time [ms]	RAM usage [MB]	RAM usage difference [kB]
0	6.6 ± 0.8	3.50 ± 0.17	441.5 ± 173.0
1	7.0 ± 0.8	3.53 ± 0.11	466.6 ± 106.1
2	7.1 ± 1.0	3.53 ± 0.06	463.0 ± 63.4
3	6.6 ± 1.5	3.45 ± 0.01	386.2 ± 1.3
4	7.2 ± 0.1	3.53 ± 0.01	463.0 ± 11.6

Table B.4: The average run time and RAM usage per modal depth of the tableau solver using depth-first search for non-tautologies.

Nr. of connectives	Nr. of formulas	Outliers from the time limit	Outliers based on IQR
1	48	0	2
2	286	68	10
3	286	132	21
4	286	160	21
5	286	197	9
6	286	212	10

Table B.5: The number of formulas per complexity (measured by the number of connectives), the number of formulas that were removed as they reached the time limit, and the number of formulas that were removed based on being outliers.

Modal depth	Nr. of formulas	Outliers from the time limit	Outliers based on IQR
0	184	0	13
1	489	136	25
2	439	280	20
3	241	227	10*
4	98	96	0
5	27	27	0
6	3	3	0

Table B.6: The number of formulas per complexity (measured by the modal depth), the number of formulas that were removed as they reached the time limit, and the number of formulas that were removed based on being outliers. *These outliers were removed based on the IQR analysis in Table B.5.

Complexity		Nr. of formulas	Nr. of tautologies
Nr. of connectives	1	46	6
	2	208	25
	3	133	18
	4	105	11
	5	80	12
	6	64	8
Modal depth	0	171	24
	1	328	40
	2	139	15
	3	4	2
	4	2	0

Table B.7: The number of formulas per complexity (measured by the modal depth), the number of formulas that were removed as they reached the time limit, and the number of formulas that were removed based on being outliers.

C Figures

Figure C.1: Screenshot of an example tweet from the Twitter bot [13].