university of groningen / faculty of science and engineering

I know exactly what I know (and so do you): constructing an epistemic logic bot in S5 with one or more agents

Bachelor's Project Thesis

Darima Budazhapova, s3410609, d.budazhapova@student.rug.nl,
Supervisor: Prof. Dr. L.C Verbrugge

Abstract: Epistemic logic is a field of modal logic that deals with formalizing knowledge of agents. Using epistemic operators K and M, as well as Kripke's possible world semantics, it allows us to explicitly reason about what individual agents know or don't know via agent-specific accessibility relations between states. S5 is a subsystem of epistemic logic that is suited to reasoning with non-human agents, e.g. servers in a network. This project implements a simple Twitter bot that continuously generates, tests, and posts tautologies in epistemic S5 logic under the reflexivity, transitivity, and euclidicity constraints. Analysis of the bot's solving time and peak memory usage with regards to length and modal depth of the formulas has yielded moderately positive correlations with modal depth showing the most influence (Spearman's $\rho = 0.61$ with peak RAM usage).

1 Introduction

Epistemic logic is a field of modal logic concerned with formalizing knowledge of agents. Reasoning about what agents know and what they consider possible has applications in many fields of computer science, such as cybersecurity and multi-agent systems. The works of G.H. von Wright and Jaakko Hintikka in the 1950s and 60s are considered to be seminal for the field and this project draws strongly on Meyer & van der Hoek's *Epistemic Logic for AI and Computer Science* [1] as well as study material for the course *Logical Aspects of Multi-Agent Systems* [2] for theoretical grounding.

1.1 Epistemic logic system $\mathbf{S5_{(m)}}$

$\mathbf{S5_{(m)}}$ is a subsystem of epistemic logic which consists of system $\mathrm{K}_{(\mathrm{m})}$ and three additional axioms. Subscript m denotes the number of agents; it will be omitted unless a specific instance is being discussed. Boolean truth valuations are used in this project.

Kripke's model in S5

Model (denoted $\mathbb{M}$) is the formal representation of the world-state. $\mathbb{M}$ is a tuple $< S, \pi, R_1, ..., R_m >$ where:

- S is a non-empty set of states. States act as "alternate universes" to one another, representing what is possible, though only one of the states can represent the true disposition of things at any time;
- π is the truth assignment for all propositional atoms in those states;
- R_i is an accessibility relation between states in the set S for an agent i. In this notation, sR_it means "for agent i state s can access state t". While truth valuations of propositional atoms in states of the model are fixed (e.g., φ in state s would be true regardless of agent), connections between those states are individual agent-based, i.e. dependent on the agent's information (see explanation of *epistemic alternatives* below). All R_i in S5 are subject to a special set of constraints called the **equivalence relation** $\mathcal{EQ}$:

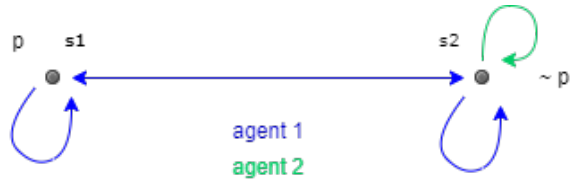


Figure 1.1: Knowledge operator application example on states s_1 and s_2

- R_i is reflexive: for all $s \in S, sR_is$;
- R_i is transitive: for all $s, t, u \in S$, if sR_it and tR_iu, then sR_iu;
- R_i is euclidian: for all $s, t, u \in S$, if sR_it and sR_iu, then tR_iu;
- from the above it also follows that R_i is symmetric, i.e. for all $s, t \in S$, if sR_it, then tR_is.

Operators K and M

S5 uses all the connectives of standard propositional logic ($\neg, \wedge, \vee, \rightarrow$, and $\leftrightarrow$) with the addition of epistemic operators $\mathbf{K_i\varphi}$ ('agent i knows φ') and $\mathbf{M_i\varphi}$ ('agent i considers φ possible'). For $\mathrm{K_i}\varphi$ to be true, φ must be true in all the states accessible from this one (which includes the current state in S5). For $\mathrm{M_i}\varphi$ to be true, φ in one of those worlds is enough. They are reminiscent of $\Box$ and $\diamond$ symbols.

Using a simple example in Figure 1.1, Agent 1 sends a postcard to Agent 2 (p stands for "postcard delivered"). Agent 1, located in state *s1* does not know whether the postcard has been delivered ($\neg Kp$). Agent 2 in state *s2* has not received the postcard ($K_2\neg p$ with the state *s2* on the right being the true one), but until they communicate and resolve the matter, for Agent 1 both states with p and $\neg p$ remain possible, i.e., states *s1* and *s2* in the Figure 1.1 are epistemic alternatives [3] for Agent 1.

Accessibility relations (cont.)

States s and t are called **epistemic alternatives** if an agent cannot distinguish between them, as far as that agent's information allows. Only epistemic alternatives are accessible to one another and the web of accessibility relations is individual to every agent in the model and dependent on their knowledge.

From the relational constraints above it follows that any state can access itself and that any state that can access a different state would also be able to access any state the other one is related to and vice versa. Thus each relational 'set' forms a network of interconnected states where each and every new state is able to access any and all of the states already in the set and they are able to access the new state in return.

World <M, s>

The agents are not directly depicted in S5-models while their knowledge can be inferred from the model representations. Moreover, any statements about the knowledge or beliefs of agents can only be evaluated at a specific state (e.g., in Figure 1.1 the statement K_2p correct in the state s_2 on the right). To represent that, this project uses the term **world** (or pointed Kripke model) <M, s>: it consists of a Kripke model $\mathbb{M}$ and a distinguished ('focal') state s. This allows us to make pronouncements about the state of the agents' knowledge at that focal state: in Figure 1.1 $< \mathbb{M}, s_1 > \models \neg K_1p \wedge \neg K_1\neg p$ and $< \mathbb{M}, s_2 > \models K_2\neg p$.

Axioms of S5

Axioms from system $\mathrm{K_{(m)}}$:

- **A1** all tautologies from propositional logic
- **A2** $(K_i\phi \wedge K_i(\phi \rightarrow \psi)) \rightarrow K_i\psi$
  Alternatively, $K_i(\phi \rightarrow \psi) \rightarrow (K_i\phi \rightarrow K_i\psi)$
- **R1** $\phi, \phi \rightarrow \psi \Rightarrow \psi$ (*Modus Ponens*)
- **R2** $\phi \Rightarrow K_i\phi$ (*Necessitation.* also see discussion on *logical omniscience* complaint). **Not** the same as $\phi \rightarrow K_i\phi$ (which is wrong)!

New axioms of S5:

- **A3** $K_i\phi \rightarrow \phi$
- **A4** $K_i\phi \rightarrow K_iK_i\phi$
- **A5** $\neg K_i\phi \rightarrow K_i\neg K_i\phi$

Axioms A4 and especially A5 may seem contentious when applied to humans, but they are appropriate and valuable for discussion of non-human agents.

Tableau rules

The formulas in epistemic logic S5 are checked using tableaux by applying tableau rules [4] similar to those of modal logic with some amendments for epistemic operators and relational constraints respectively:

$$\begin{array}{cccc} K_i\phi, s & & & \neg M_i\phi, s \\ sR_it & \neg K_i\phi, s & M_i\phi, s & sR_it \\ \downarrow & \downarrow & \downarrow & \downarrow \\ \phi, t & sR_it & sR_it & \neg\phi, t \\ & \neg\phi, t & \phi, t & \end{array}$$

Note on accessibility relations in these rules:

- For the rule for K_i and $\neg M_i$, it is applied for all states t already appearing on the branch and all t discovered later;
- For the rule $\neg K_i$ and M_i, t must be a new state for that branch.

$$\begin{array}{cccc} \bullet & sR_it & sR_it & sR_it \\ \downarrow & tR_iu & sR_iu & \downarrow \\ sR_is & \downarrow & \downarrow & tR_is \\ & sR_iu & tR_iu & \end{array}$$

Soundness and completeness

$\Sigma \vdash_{S5} \phi$ means 'φ is *provable* in S5' — in this case a closed tableau can be constructed from the set of premises Σ and the negation of the conclusion φ using the rules and axioms of S5 (*syntactic consequence*).
$\Sigma \models_{S5} \phi$ means 'φ is *entailed*' (*semantic consequence*), i.e. a set of interpretations that makes all statements in Σ true, also makes φ true.
Axiom system S is called **sound with respect to** class of Kripke models $\mathbb{M}$ if $S \vdash \phi \Rightarrow \mathbb{M} \models \phi$.
Axiom system S is called **complete with respect to M** if $\mathbb{M} \models \phi \Rightarrow S \vdash \phi$.

The soundness and completeness of the tableau system for S5 have been proved in [4] for the single-agent case ($S5_{(1)}$), which can be adapted for multiple agents.

Formula length and modal depth

The *length* ($|\phi|$) of a formula and *modal depth* ($d(\phi)$) follow the inductive definition based on [5]:

$\|p\|$	=	1
$\|\neg\phi\|$	=	$\|\phi\| + 1$
$\|(\phi \wedge \psi)\|$	=	$\|\phi\| + \|\psi\| + 1$
$K_i\phi$	=	$\|\phi + 1\|$

Table 1.1: Definition of length of epistemic formulas

$d(p)$	=	0
$d(\neg\phi)$	=	$d(\phi)$
$d(\phi \wedge \psi)$	=	$maxd(\phi), d(\psi)$
$d(K_i\phi)$	=	$1 + d(\phi)$

Table 1.2: Definition of model depth of epistemic formulas

In definitions from Tables 1.1 and 1.2 we take $\wedge$ to stand for any binary connective and K_i to mean any epistemic operator.

1.2 Research question

The goal of the project is to build a bot that regularly publishes tautologies on the microblogging platform Twitter, after verifying that they are indeed tautologies in the S5 framework of epistemic logic. Therefore, the bot will be evaluated on the following criteria:

- Can the bot formulate tautologies in S5?
- How do the length and depth of the formulas affect the time- and memory-usage of the tableau solver? Which one has more influence?

1.3 Twitter bot

The maximum length of a message (*tweet*) at the time of this project's development was 280 Unicode characters, which sets the limit for the length of a single formula. To accommodate this limit, the maximum formula length for the formula generator is set to 150 elements (not including white spaces or brackets).

2 Methods

The project is written in programming language *Python* due to the author's familiarity and the convenience of many available library packages.

The overview of the operation of the bot is as follows: the **formula generator** module creates a random epistemic logic formula of a given complexity, checks it against saved previously generated formulas for duplicates and passes it on to the **tableau solver**. The tableau solver module checks whether the formula is a tautology and if it is, the **bot** module posts it on Twitter.

Some notes on the terms used in this section:

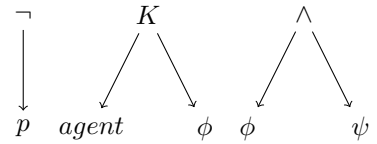
- **atoms** are denoted as letters of the Latin alphabet (*a, b, c*);
- **agents** are identified by numbers (*1, 2*);
- **states** are also identified by numbers; although they are not visible in the generated formulas, they must be kept track of during solving to check whether a given formula is a tautology;
- **connectives** from here on mean connectives of propositional logic ($\neg, \wedge, \vee, \rightarrow, \leftrightarrow$) as well as epistemic operators (K_i, M_i), unless otherwise stated, when discussing elements of a formula;
- **formula tree** — epistemic logic formula represented in a tree form.

2.1 Model

Model is a custom class that contains the formula that is being investigated, the atoms of the formula, the states (along with truth valuations for the atoms in those states), the agents and their recorded accessibility relations in a single object. A model-class object is referred to as **world**. Its main purpose is keeping track of the model's conditions discovered while solving the tableau for a given formula. When during solving a tableau splits, this is reflected in two world-objects, each representing its own branch of the tableau.

2.2 Formula generator

All formulas are first assembled in the formula generator. The generator receives the basic parameters (desired formula length, as well as how many atoms and agents it can use while generating a formula) and randomly selects connectives, atoms, and agents to construct a well-formed formula in the language of S5. The formulas are arranged as **binary trees** (using anytree package) with connectives serving as parent nodes to the children atoms/(sub)formulas (see example below). A node contains information about its type (agent, atom, or operator), "name", priority tier (see Section 2.3), ID number, etc.



The rule for K is also applicable for M; the *agent* node (left child) is always terminal. Binary connectives $\vee, \rightarrow, \leftrightarrow$ use the same tree rule as $\wedge$. *Formula* stands for either an atom or a branch (i.e., subformula). Negations share the nodes with the connectives they are applied to, unless an atom is being negated; in that case, negation is located in its own node (as in the first example). This modular approach allows the generator to avoid parsing expressions in brackets and to build formulas of any length and complexity by simply growing the "tree". The root (top node) of any tree or branch holds the main connective of that (sub)formula, which is where the tableau solver of the project can start investigating.

```
IMP/None
├── b/None
└── NEG_BI_IMP/None
    ├── K/None
    │   ├── 1/None
    │   └── NEG_OR/None
    │       ├── BI_IMP/None
    │       │   ├── a/None
    │       │   └── b/None
    │       └── b/None
    └── b/None
end formula-builder output
```

Figure 2.1: Example of a generated formula of length 7.

Figure 2.1 demonstrates a sample of a tree that translates to line-format formula $b \rightarrow \neg(b \leftrightarrow K_1\neg(b \vee (a \leftrightarrow b)))$. "None" after the atom/operator name signifies that the formula is not currently attached to any specific state.

The formula generator starts building outwards from the "leaves" upwards: inner subformulas first, then connectives "grow" on top of them. It starts with some number of atoms (between 1/4 and 2/3 of desired formula length), then it selects a con-

nective/operator to apply to existing nodes. The contents of the tree nodes (connectives, atoms, or agents) are chosen randomly and the formula grows until it reaches the desired complexity.

If the generator detects too many subformulas for the remaining number of connectives to be chosen, it limits the choice to non-negated binary connectives. Unfortunately, the variability in the number of starting atoms was added very late into the project for the purpose of achieving a range of modal depths, which resulted in a large number of "duds" — generated formulas that are not unified under a single main connective. These "duds" are therefore excluded from analysis.

Once we have generated a well-formed logical formula of a given complexity, we compare it to previously generated ones recorded in a file. If no match exists, it is passed on to the tableau solver. If it matches a previously generated formula, it is discarded. If no such formula is detected, the new formula is recorded into a file with other formulas of that length.

Writing to file

Conversion of generated formulas to file and retrieving them for comparison will be done via the built-in JSON exporter/importer function of the `anytree` package. There are three types of files generated by the algorithm: formula storage (which contains all previously generated well-formed formulas of that length), tautology storage (where only tautologies are recorded for later posting), and solver data (which records length and depth of formulas, whether it is a tautology, as well as the time taken to solve it and the peak memory usage of the solver).

2.3 Tableau solver

Once a newly generated formula arrives, the purpose of the tableau solver is to find whether the negation of the generated formula has a model (i.e., whether there are circumstances under which the original formula is false and therefore not a tautology). To discover that, the tableau solver adds a negation on top of the main connective (root node), marks the (new) root node as existing in state 0, and investigates the newly negated formula. The order of resolving connectives (their **priority tier**) is as follows:

1	$\neg, \neg\neg$
2	$\wedge, \neg\vee, \neg\rightarrow$
3	$K, \neg M$
4	$\neg K, M$
5	$\vee, \neg\wedge, \rightarrow, \leftrightarrow, \neg\leftrightarrow$

Table 2.1: Priority order of connectives and operators.

These priority tiers are built-in properties of connective nodes. Other custom node attributes include **state**, i.e., which state the formula is true in, as well as type of node (atom, agent, or connective). Agents and atoms are marked with priority 0 so that they're "resolved", i.e. their truth valuation recorded in the appropriate state by the model-object, as soon as they appear in the list of nodes available for solving.

Since the formula under investigation is shaped like a tree, expanding the tableau functionally means resolving (and deleting) connectives from the root node down: e.g., formula $a\wedge(b\vee\neg a)$ gets resolved into a, $b\vee\neg a$ (where a and $\vee$ are roots of their respective subformulas). It resolves the tree into smaller and smaller subformulas until the solver arrives at atoms (terminal nodes). Then the solver can mark the atoms' truth valuation at its model representation for future reference. Hence, "building the tableau" actually involves dismantling the formula tree and filling out the model object. A tableau branch closes when a new truth valuation of an atom contradicts a previous conclusion recorded in the model.

The tableau is solved depth-first, as a human logician would. When the tableau branches, the main tree is put on hold while the solver works on the "left branch" — in actuality a copy of the current world, formula tree included. Should that branch close, the copy is erased and the solver continues with the "right branch" using the main world. A complete and still open branch proves that the formula was not a tautology, hence no further solving necessary. The world of that branch then contains a countermodel for the original formula.

K and $\neg M$ operators

These tier-3 operators are special cases that can never be completely resolved — they might need to be applied again to newly discovered accessibility relations. Therefore, while being "solved" for the first time, a copy of that entire subformula is moved to a **sidebar**, so that it can be assessed again for a new state. Checking in with the sidebar every time a new accessibility relation appears also ensures the algorithm doesn't get stuck in a loop, resolving the same operator over and over.

Representing the equivalence relation $\mathcal{EQ}$

As described in the Section 1.1, the S5 language employs a special combination of relational constraints called *equivalence relation* $\mathcal{EQ}$. Taken together, reflexivity, transitivity, and euclidicity produce a system where if for agent 1 state 0 can access states 1, 2, 3 and state 3 accesses state 4, then state 4 can also access 0, 1, 2, 3 and vice versa. All states that share any relation in common (for the same agent) have direct access to each other as well as themselves.

Consequently, in this project the accessibility relations are implemented as undirectional "grab bag" sets of states that are recorded as properties of agents. For any newly discovered relation that involves one of the states already in the set, the new state gets added to the set. So in the example above, agent 1 would have had the set $< 0, 1, 2, 3 >$ already recorded, which with the addition of state 4 becomes the set $< 0, 1, 2, 3, 4 >$ where any of of these states can access themselves as well as any other state in the set.

Algorithmic example — excerpt from the tableau solver

The pseudocode example here is a part of the tableau solver: specifically the method for resolving tier-4 operators, M and $\neg K$. The root node (`operator`) of this subformula is the epistemic operator (M or $\neg K$), its agent represented as its left child. The variable `home_state` refers to the ID number of `operator`'s current state, i.e., in which state this epistemic formula is located in the tableau.

As explained in Section 1.1, the rule for diamond-like operators requires the solver to generate a `new state` and record a new accessibility relation for the `agent` in question between the `home state` and `new state` using the Model-class methods `add state` and `add relation`.

Algorithm 2.1 Solve diamond-like epistemic operator

```
1:  function SOLVE M OR ¬K(operator, world)
2:      home_state ← operator.state
3:      agent ← operator.left_child
4:      register new state in the world
5:      register new accessibility relation between
    home_state and new_state for agent
6:      current_set ← all states in equivalence
    relation with home_state and new_state for
    agent
7:      if operator is ¬K then
8:          inject negation into inner formula
9:      end if
10:     pass on operator's state to its children
11:     remove epistemic operator node from the
    tree
12:     function TRIGGER SIDEBAR(agent,
    home_state, new_state, world)
13:         if new accessibility relation formed for
    box-like formula with matching agent then
14:             solve box-like formula
15:         end if
16:     end function
17: end function
```

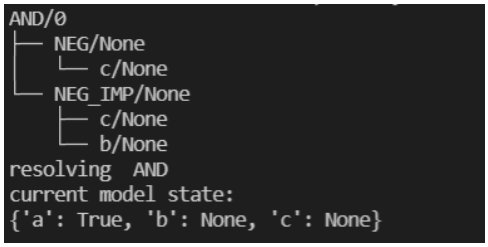
If the epistemic operator in question is $\neg K$, then a negation needs to be inserted in the tree between the parent operator and its `right child` (next top node in the subformula). The method `insert neg node` makes the appropriate modifications to the subtree: generating a negation node for atoms, modifying the `right child` if it is an operator (e.g., $\vee$ node become $\neg\vee$, an already negated operator $\neg \rightarrow$ becomes nodes $\neg\neg$ and $\rightarrow$).

Model-class methods `confer state` and `remove epist op` respectively assign the `new state id` number to the subformula φ (i.e., record that φ is true in the new state as opposed to the one $K_1\phi$ was in) and uncouple and delete the root epistemic operator node.

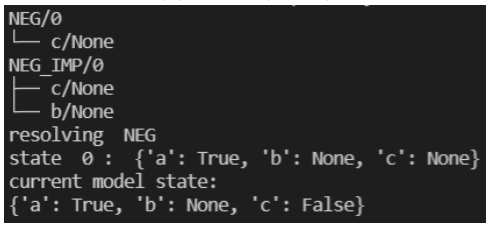
Finally, since resolving a diamondlike operator opened up a new accessibility relationship for this agent, `trigger sidebar` method checks whether there are any previously expanded tier-3 epistemic

operators (K or $\neg M$) that might need to be applied to the new state.

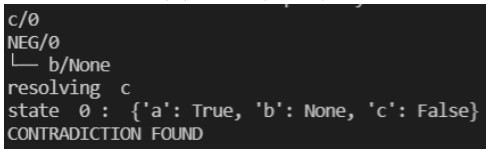
Simple formula example



(a) $\neg c \wedge \neg(c \rightarrow b)$



(b) $\neg c$, $\neg(c \rightarrow b)$



(c) c, $\neg b$

Figure 2.2: Tableau solving progression for formula $\neg c \wedge \neg(c \rightarrow b)$

Figure 2.2 shows the steps of the tableau solver's work on a simple example formula $\neg c \wedge \neg(c \rightarrow b)$. Initially (Figure 2.2a) only one connective ($\wedge$) in state 0 (`AND/0`) is available for resolving, so that is what the tableau solver works on and passes the parent node's state onto the children. Next, in Figure 2.2b the formula has two available branches: $\neg c$ and $\neg(c \rightarrow b)$, both also in state 0, so they are sorted according to the priority order from Section 2.3. `NEG/0` is resolved by marking its child-atom c as 'false' in state 0 and connective `NEG_IMP` (negation of implication) is resolved into c and $\neg b$ (Figure 2.2c). Here the tableau solver runs into a contradiction, as the available subformula claims that c is `True` in state 0 when it has already been recorded as `False` previously on this branch. Therefore, the branch closes and, since there are no other branches, the inverse of the original formula $\neg c \wedge \neg(c \rightarrow b)$ has been proven a to be a tautology.

Dealing with infinite branches

A tableau in the language of S5 can unfold into an infinite branch if a box-like formula contains a diamond-like formula for the same agent (using states within the same equivalence relation). To prevent the solver from getting stuck in an infinite loop, a counter of repeat resolutions was implemented for diamond-like epistemic operators. If it detects the same M or $\neg K$ node be resolved thrice on the same branch, the branch is judged infinite — therefore, open and complete.

Measuring the resource usage

The bot measures the time and maximum RAM usage of the tableau solver with the help of Python native packages `time` and `tracemalloc`. The module `time` is used to measure CPU time — the time it takes for CPU to solve the formula, as opposed to "wall time", which is how much real time has passed between the start and the end of a process. CPU time was chosen for measurement so that the data gathered is not contaminated by outside factors, e.g. the computer's resources being occupied by third party processes.

2.4 Twitter publisher

An auxiliary script converts formula tree structures to single-line human-readable string format. The Twitter bot uses the Python library tweepy to generate and post formulas proved to be tautologies. The publisher bot loads a list of generated tautologies of random length (3 to 150 elements) and posts one on @epistemic_botS5 every 4 hours. If the number of already prepared tautologies is running low (less than 3), it runs the generator and solver modules several times until they provide the queue with several more ready tautologies. The library used for triggering and running the code at specific intervals is the APScheduler package.

2.5 Code repository

The code for the entire project is available on GitHub at

https://github.com/budazhapova/epistemic-bot-s5. The repository contains two branches: `master` with the working code version of the Twitter bot that operates from a Raspberry Pi and `data`, which is set to generate and solve formulas in bulk and contains the CSV file with the acquired data as well as the JSON files with the generated formulas.

3 Results and discussion

The system as designed and implemented can indeed generate an unlimited number of epistemic logic formulas for a potentially unlimited number of propositional atoms and agents (the working version of the generator, however, is limited to three of each for practical reasons). It tests the generated formulas using the tableau method.

To obtain data for this analysis, a maximum of 200 formulas of lengths 3-15 were generated excluding duplicates. Three was chosen as a starting point because that is the length of a shortest possible tautology (e.g., $p \to p$). Naturally, in smaller length batches it is impossible to generate so many non-duplicated formulas, which is why the total number of formulas of that length or depth is included in the "total" columns. Within any specific length batch there is some amount of variability with regards to depth: most of all formulas have modal depth of 3-4 with maximum depth of 8.

length	depth	total	time	memory
3	1	4	0.00781	5121
4	1-2	27	0.01563	4545
5	1-4	153	0.01562	6205
6	1-5	180	0.01562	7600
7	1-6	200	0.01562	10146
8	2-6	200	0.01562	8938
9	2-7	200	0.03125	12786
10	2-7	200	0.03125	13600
11	2-7	200	0.03125	15719
12	3-7	200	0.03125	17287
13	2-8	200	0.03125	18516
14	3-8	200	0.04688	27240
15	3-8	200	0.04688	24050

Table 3.1: Median CPU solving time (sec) and peak memory (bytes) by formula length

depth	length	total	time	memory
1	3-7	38	0.01562	4367
2	4-13	340	0.01562	6666
3	5-15	652	0.01562	11100
4	5-15	626	0.03125	18086
5	6-15	356	0.04688	24318
6	7-15	107	0.06250	34532
7	9-15	37	0.2344	55765
8	13-15	8	0.20312	67494

Table 3.2: Median CPU solving time (sec) and peak memory (bytes) by modal depth

	CPU time	peak memory
length	0.373	0.489
depth	0.505	0.609

Table 3.3: Spearman's ρ (rank-order) correlation between factors and resources used.

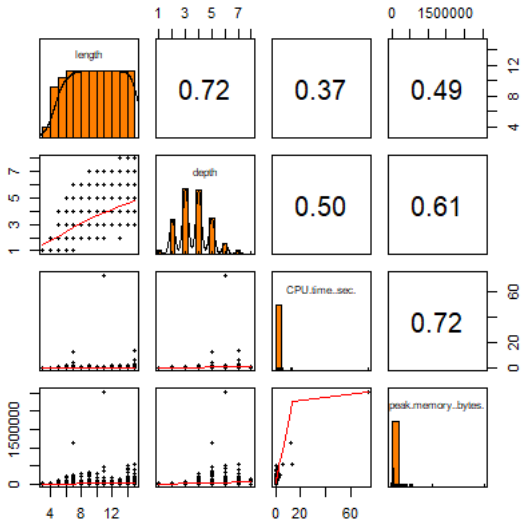


Figure 3.1: Scatterplot matrix of relations between formula types and resource usage of the tableau solver. Numerical coefficients denote Spearman's ρ between factors.

As we can see in Tables 3.1 and 3.2, length of a formula does not hold a lot of influence over time required for solving it. Peak memory (maximum RAM occupied during solving) increases more linearly both with the length and the modal depth —

the longer the formula, the more nodes it contains; the more depth it has, the more operators the solver needs to take care of. Extra memory is allocated for branching as the solver retains a copy of both the model object and the formula tree for each branch of the tableau. CPU time remains comparatively stable with little variance, as can be seen both in Tables 3.1 and 3.2 and Figure 3.1.

The strongest correlation between the features of an investigated formula and the resources usage of the solver is found for modal depth and peak memory usage (Spearman's $\rho = 0.61$). The RAM usage increases almost linearly with the modal depth with the exception of depth 8 for which only 8 formulas were generated. The correlation results show moderately positive relationships between formula length, modal depth, solving time, and peak RAM usage (see Figure 3.1), showing that there is indeed a positive link between input variables (length, depth) and the solver's resource usage, but it is not strong enough to name either of the formula features a definitive influence.

Overall, it must be concluded that formula depth has a stronger effect on the resource demands of the solver.

4 Conclusion

For this project an autonomously operating algorithm for generating, solving, and publishing tautologies in epistemic logic in the language of $S5_{(m)}$ was designed and implemented. The system consists of 3 main modules: the formula generator, the tableau solver, and the Twitter publisher. The formula generator creates random formulas for a given formula length using a variable number of propositional atoms. The tableau solver tests the generated formulas using the semantic tableau method to find out whether they are tautologies. Generated and proven tautologies are then published on Twitter with the interval of 4 hours between posts.

The formula generator uses random selection for constructing epistemic formulas in the language of S5. The formulas can be of any desired length, though for purpose of fitting into a single Twitter post the generator's upper bound is set to 150 elements. Depending on the length, a formula can use up to 3 atoms (*a, b, c*) and 3 agents (*1, 2, 3*), with the number of atoms and agents used increasing with the length. The algorithm can choose between a number of propositional logic connectives ($\neg, \wedge, \vee, \rightarrow, \leftrightarrow$) and epistemic operators (K, M) as well as their negations. The generator links the subformulas with arbitrarily chosen connectives, resulting in a randomly constructed formula of desired length and random modal depth.

If the generated formula is well-formed, it is checked against previously generated formulas on file and, if novel, is passed on to the tableau solver.

The tableau solver investigates the formula to determine whether it is a tautology using tableau rules and Kripke's possible worlds semantics. If the formula is proven to be a tautology, it can be queued for posting on the bot's Twitter account.

The tableau solver cannot always complete branches due to the presence of potentially infinite loops. It is possible that some of them could close with time and that it does not detect some tautologies because of it. Therefore, the algorithm cannot be called complete, i.e. it may not provide a closed tableau for the negations of all formulas that are entailed in S5. However, all formulas that are provable (under the restriction of declaring infinite branches open and complete) are indeed entailed — the set of truth valuations can be read from the model object after or at any point during solving.

The generated formula, tautology or not, is recorded into a JSON file housing all previously generated formulas of that length for future comparison. Additionally, new tautologies are placed into a queue for publishing on Twitter. The queue holds several previously tested formulas and pops one whenever the time comes to publish a new post. When the queue is close to being depleted, it calls the formula generator and tableau solver to provide it with new tautologies. This ensures stable posting intervals due to ready availability of tautologies to publish.

Data analysis has shown that while both the solving (CPU) time and peak memory usage of the tableau solver are moderately influenced by formula length and modal depth, it is not a notably strong relationship. Between the length and depth, modal depth appears to be the stronger affecting factor with Spearman's ρ of 0.61 correlation to RAM usage. However, these measurements are strongly affected by the hardware used during the project. While the algorithm is very lightweight, especially in comparison to resource-demanding Ma-

chine Learning projects, a weaker machine could show more of a relationship between complexity of a formula and the resource usage of the solver. For example, the Raspberry Pi on which the Twitter publisher is located takes noticeably longer to solve formulas it generates. The length of time required, however, makes it less than suitable for gathering data.

The variance in the data is most likely due to the influence of branching tableaux for memory and epistemic operators for solving time. When a tableau splits into two branches, each branch is represented by its own Model-class object (world), which contains not only the formula tree and its associated sidebar, but also all the supplementary model information, such as: the number of atoms and agents involved, accessiblity relation sets discovered, total number of nodes generated for the tree, whether any of the diamond-like operators were resolved more than once (for the infinite branch stopper), etc. Naturally, each branching event drastically increases the amount of RAM involved; thus, multiple branching events would account for the outliers in the peak memory row of the scatterplot matrix in Figure 3.1. Similarly, it is likely that the presence of epistemic operators in the formulas is responsible for the higher datapoints and outliers in the CPU time measurements. Resolving box-like operators requires the solver to check all the relevant accessibility relations for the agent in question and expand the inner formula for all appropriate relations found. Diamond-like epistemic operators resolve the formula into a new state *and* are required to check with the previously expanded box-like formulas and resolve those, if there are any where it resolution is appropriate. Therefore, while resolving epistemic operators does not necessarily create such a demand for memory resources, the time required scales up with the number of agents, states, as well as previously recorded relation sets.

Issues and potential improvements

The random nature of the generator has positive and negative sides. The positive is that it can generate a formula of any length as desired without having to first "build up" to it. The negative is the consequence of the positive: since the user has no control over the process after specifying the desired length, which produces more and more duplicates as more formulas are generated and recorded. The system has little mid-process guidance and the heuristics that are implemented to prevent generation of split trees (formulas that don't unify under a main connective) often fail after the last amendment to vary the number of generated atoms (in order to ensure some variation in modal depth per formula length). The resulting "duds" are entirely excluded from analysis as they are not well-formed formulas and cannot be evaluated.

Related to the issue of generating unwanted duplicates is the problem of impossibility of early closure of branches. Since a branch needs to work its formulas down to propositional atoms before it can detect a contradiction, early closure by comparing subformulas more complex than single propositional atoms and their negations: meaning that, say, $p \vee q$ and $\neg(p \vee q)$ is not recognized as a contradiction and that branch would not close until one is discovered among the truth valuations of the atoms. A modification that would allow to store previously discovered formulas for comparison in parallel with the solving would shave off some effort and time at the expense of memory.

Another potential avenue for improvement would be to adapt the tableau solver for breadth-first search or adopt a kind of guidance algorithm to determine which branch to investigate first. The algorithm in its current form traverses the tableau branches one at a time, when the fastest way to detect a non-tautology would be to follow the branch that is most likely to remain open. While such an approach would be less computationally and memory-intensive than a breadth-first search (which would process all the branches simultaneously), it would require fairly complicated heuristics in order to replicate the intuition of an experienced human logician. And neither of these improvements would necessarily improve the investigation of tautologies — after all, for a formula to be proven a tautology, all branches of the tableau for the formula's negation need to be investigated, so the order in which it is done makes no difference.

This project makes no attempt to optimize the resource usage, so another potential improvement would be to make it more memory-efficient.

And last, but not least, are the potential expansions into other subtypes of epistemic logic: for example, with some tweaks to the handling of rela-

tions this project could be adapted for use on doxastic logic KD45 (another type of epistemic logic that deals with belief instead of knowledge). An extension for dynamic epistemic logic that would allow the algorithm to deal with evolving knowledge or an extension for common ("everyone knows") and implicit knowledge, which formalizes situations where a fact can be deduced from other facts known separately by different members of a group are among the plausible systems that can be built on top of this project.

References

[1] Meyer, John-Jules Charles and van der Hoek, Wiebe, *Epistemic logic for AI and computer science*, ser. Cambridge tracts in theoretical computer science. Cambridge University Press, vol. 41.

[2] Verbrugge, L.C., "BOK-project: Logics for artificial intelligence."

[3] W. van der Hoek and L. Verbrugge, "Epistemic logic: a survey," in *Game Theory and Applications, vol. 8*, L. Petrosjan and V. Mazalov, Eds. Nova Science Publishers, pp. 53–94.

[4] G. Priest, *An Introduction to Non-Classical Logic: From If to Is*. Cambridge University Press, google-Books-ID: rMXVbmAw3YwC.

[5] van Ditmarsch, Hans, Halpern, Joseph Y., van der Hoek, Wiebe, and Kooi, Barteld, "An introduction to logics of knowledge and belief," in *Handbook of epistemic logic*. College Publications, pp. 1–52.