MAKING MODEL CHECKING SCALABLE: Implementing Succinct Kripke Models for Public Announcement Logic

Bachelor's Project Thesis

Maickel Hartlief, s3445801, m.hartlief@student.rug.nl
Supervisor: Dr B. R. M. Gattinger

Abstract: Public Announcement Logic (PAL) is an extension of Propositional Logic including agents with knowledge about the world, and announcements that can be made to update the knowledge of these agents. Kripke Models for PAL can be extremely large, and model checkers slow. Aiming for a more efficient alternative, we consider a different representation: Succinct Kripke Models. This representation uses Mental Programs (based on Dynamic logic of propositional assignments) to describe the relations between worlds for agents, and a Boolean formula to denote the set of worlds, instead of listing all worlds and relations explicitly. Model checking is PSPACE-complete for both types of representations, but we still expect a difference in actual performance. To test this, we implemented the following in Haskell: a model checker for explicit Kripke models, a model checker for Succinct models, and an explicit and succinct model for the Muddy Children problem. The size of the succinct model is smaller than that of the explicit model, and this difference becomes larger for a higher number of children. The expectation is that the execution time for checking formulas will not be much larger. The results show that the succinct model checker is slower than the Kripke model checker, and that the difference grows for a higher number of children.

1 Introduction

Computers are already everywhere, and continue to become an even larger part of our lives. Our reliance on them in virtually every field from health-care to communication and from transport to economics is often warranted because of the many benefits it grants. This reliance however is also a great risk, considering that fatal bugs can still occur.

Take for example the Ariane 5 rocket incident from 1996 (Lions, Luebeck, Fauquembergue, Kahn, Kubbat, Levedag, Mazzini, Merle, and O'Halloran, 1996). Part of the code for this rocket was reused from the Ariane 4, and like everything else, thoroughly tested. However, a function of the code was not used at all by the Ariane 5, but was only left in so that no redesign of that program needed to be done. This unchecked bit of code happened to convert a 64-bit floating point into a 16-bit integer and while this worked properly for the Ariane 4, the Ariane 5 had a higher horizontal velocity, causing the variable in that function to overflow. Due to the idea that hardware could fail but software could not, this error caused the system to believe that the hardware was faulty and it shut itself down. The rocket was now soaring through the air without navigation, and consequently exploded along with its 400 million dollar payload.

This and many other instances of system failures make the following question a relevant one for human civilization's reliance on computers. Is there a way to make sure that software systems function properly?

A method of validating the proper functioning of systems is model checking, which is checking whether a certain specification is fulfilled, given a situation. To take the Ariane 5 example, the situation could be the program and the simulation, and the specification the successful launch of the rocket. This can of course take many forms, like a model, simulation, program or other formalizable system. Ironically, model checking can also be done by computers.

Efforts to make these model checkers include several methods of symbolic model checking for Dynamic Epistamic Logic (DEL) by Gattinger (2018).

1.1 Time-Complexity

A problem for model checkers is their computational complexity. The memory required to store a representation of situations, and the time needed to check specifications grow beyond realistically manageable for many types of representations.

1.1.1 PAL model checking

We look at S5 Kripke Models for Public Announcement Logic (PAL). This is the simplest form of DEL, and can besides boolean logic encode agents with knowledge, and truthful announcements that change this knowledge. With this, we are able to model situations with agents and their epistemic states in a changing environment. The precise specifications will be explained in section 2.

A Kripke model represents every possible world explicitly, as well as every agent including a list of all the worlds. The number of worlds can grow exponentially with the number of propositions in the model. Model checking DEL is PSPACE-complete (Aucher and Schwarzentruber, 2013). This means that it requires a polynomial amount of space, even on non-deterministic computers.

1.1.2 Succinct PAL model checking

Usually, a more succinct representation of a model takes longer to model check. However, Charrier and Schwarzentruber (2017) provide a framework that is exponentially more succinct for some models, but instead of the complexity being the expected EXPSPACE-complete, it remains PSPACE-complete. This has to do with the use of Mental Programs (Balbiani, Herzig, Schwarzentruber, and Troquard, 2014)

1.2 Actual Performance

To propose using Succinct models as opposed to explicit Kripke models, we need to make sure the smaller size of the models is worth the difference in actual computation time. Theoretically, this already sounds promising, but PSPACE-complete still has a large range, which means different systems which are all PSPACE-complete can have large differences in actual execution time. It is therefore worthwhile to test the following question: what is the actual performance of a succinct model checker relative to an explicit Kripke model checker?

To do this, we will take the implementation of several model types in the Symbolic Model Checking Dynamic Epistemic Logic (SMCDEL) system by Gattinger (2018) and the explicit model checker DEMO-S5 by van Eijck (2014). We will extend it with the framework for Succinct models proposed by Charrier and Schwarzentruber (2017) to check its actual performance against explicit Kripke model checking.

If we look at how the model checkers work, we can hypothesize about their performance. A large portion of the time model checking larger models is spent looking through the list of worlds and the propositions that hold true in them. Succinct models use a list of propositions and a formula that needs to hold true for every valid state. Instead of looking through a list that grows exponentially for larger models, it simply generates the states it needs by applying a Mental Program on a vocabulary of propositions. There are of course other factors at play, and it can be the case that having to generate almost all states several times takes more time than looking through all of the states several times, but the aim is to make the succinct model checker similar in speed to the explicit Kripke model checker.

2 Background on PAL

This thesis uses Public Announcement Logic. We will go over some background information and notation. Feel free to skip over parts that are already familiar. A far more extensive overview can be found in van Ditmarsch, van der Hoek, and Kooi (2007).

2.1 Propositional Logic

In a propositional logic we can say things about properties of a world. The syntax of propositional logic is given by the following definition:

Definition 1 *Given a vocabulary V, the boolean language $\mathcal{L}_B(V)$ is given by*

$$\varphi ::= \top \mid \bot \mid p \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi$$

where $p \in V$. We represent a boolean assignment (or 'world') as a set of propositions $w \subseteq V$ where all $p \in w$ are true. Furthermore, we specify the semantics of the above grammar:

- $w \models \top$ *always holds*
- $w \models \bot$ *never holds*
- $w \models p$ *holds iff* $p \in w$
- $w \models \neg\varphi$ *holds iff* $w \models \varphi$ *does not hold*
- $w \models \varphi \wedge \psi$ *holds iff* $w \models \varphi$ *and* $w \models \psi$ *hold*
- $w \models \varphi \vee \psi$ *holds iff* $w \models \varphi$ *or* $w \models \psi$ *holds.*

With this, we can for instance say that it is sunny outside (s) and that it is not warm inside ($\neg t$). The vocabulary V of $\mathcal{L}_B(V)$ in this example is $\{s, t\}$ and the assignment is $w = \{s\}$.

The operators $\bot$ and $\vee$ can be defined in terms of the other operators in Definition 1 so are technically not necessary. However, they are explicitly represented in the model checker, so it makes sense to do so here as well. For a computer, being able to use the operator directly instead of having to represent it as a longer formula in terms of the other operators is more efficient.

2.2 Epistemic Logic

Epistemic logic extends Definition 1 with agents that can have knowledge about the world, and can therefore deem several worlds possible.

Definition 2 *We extend the grammar of Definition 1 as such:*

$$\varphi ::= \top \mid \bot \mid p \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid K_i\varphi$$

where $p \in V$ and $i \in I$ (I being the set of agents). We read $K_i\varphi$ as "agent i knows that φ is true".

An epistemic logic can also include common knowledge, but this will be ignored here.

We can extend our example and say an agent Annie (a for short) is inside, so she knows it is not warm inside ($K_a\neg t$). However, the room she is in has no windows, so she does not know whether it is sunny or not sunny outside ($\neg K_a s \wedge \neg K_a \neg s$).

Note that the word 'whether' here implies a lack of knowledge of the actual truth-value of S. We can say whether an agent knows something without saying anything about that proposition's truth-value. To apply this to the example, let us say Annie's roommate Bob (b for short) is outside and Annie knows this. She knows that Bob knows *whether* it is sunny outside, without knowing herself. $K_a(K_b s \vee K_b \neg s)$ and the previously stated $\neg K_a s \wedge \neg K_a \neg s$ do not contradict each other. We define $K_i^? \varphi$ to be a shorthand for $K_i\varphi \vee K_i\neg\varphi$, thus meaning "agent i knows whether φ is true".

Now to link this with the concept of worlds, let us say that the propositions that hold true in world w are $Val(w) \subseteq V$. The set of valuations of all worlds for our previously established vocabulary is $W = \{\emptyset, \{s\}, \{t\}, \{s,t\}\}$. These are all the possible configurations for all propositions in the vocabulary. An issue with this is that there can exist multiple worlds with the same valuation, and this issue will be explained in Section 2.4.1.

For Annie, the worlds with valuation $\emptyset$ and $\{s\}$ are indistinguishable and so are those with $\{t\}$ and $\{s,t\}$, since both have the same truth-value for t and the truth-value of s is not observed. Since the actual world is $\{s\}$, the possible worlds according to Annie are $\emptyset$ and $\{s\}$. Since Bob is outside and does not know whether it is warm inside ($\neg K_b^? t$), but does observe that s, the possible worlds for him are $\{s\}$ and $\{s,t\}$.

2.3 Public Announcement Logic

The epistemic logic from Definition 2 is further expanded with notation for public announcements to get Public Announcement Logic. PAL is the simplest variant of DEL, and the one we will be using here.

Definition 3 *The language $\mathcal{L}_P(V)$ extends the language in Definition 2 as such:*

$$\varphi ::= \top \mid \bot \mid p \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid K_i\varphi \mid [!\psi]\varphi$$

where $p \in V$, $i \in I$, and we read $[!\psi]\varphi$ as "after publicly, truthfully announcing ψ, φ holds."

A public announcement can change the model, therefore $[!\psi]\varphi$ is called a dynamic operator.

We demonstrate this operator in the example: suppose Bob calls up Annie and tells her that it is sunny outside. Assuming Annie actually listens to and believes Bob, and assuming Annie and Bob are the only agents in this model, this could be considered a public announcement. However, if those assumptions are not made due to Annie distrusting Bob, or them having a third housemate called Charles (c for short), then this is not a public announcement.

A public announcement is always truthful, and every agent will hear it and believe it. So to be pedantic, say in this universe the weather is always accurate, and is announced on giant speakers that penetrate even windowless rooms. After this announcement, Annie knows that it is sunny outside and only one possible world remains. We can say this like $[!s](K_a s \wedge K_c s)$, meaning "after publicly announcing that it is sunny outside, Annie and Charles know that it is sunny outside".

2.4 Kripke Models

Now that we have covered how worlds and agents can exist in a given situations, it is time to talk about how these are represented formally. We take the standard definition of a Kripke model:

Definition 4 *A Kripke model is defined by $M = (W, (\rightarrow_a)_{a\in Ag}, Val)$ where:*

- *W is a non-empty set of worlds;*
- *$(\rightarrow_a)_{a\in Ag} \subseteq W \times W$ are epistemic relations for every agent a in Ag. It is represented by a partition of W. This is a set of non-empty subsets containing elements of W such that all elements in W are included in exactly one of those subsets. Every partition defines an equivalence relation, meaning the worlds in the same partition are indistinguishable for the agent;*

- Val *is a valuation function containing a set of propositions $Val(w) \subseteq V$ for every world $w \in W$ where all and only the propositions in $Val(w)$ are true in w.*

We can now represent models formally, but we have no way of determining which of the worlds represents reality. To link this back to our example: there is no way to communicate that it is in fact sunny outside and not warm inside. For this, we use a *pointed model*, which is a model together with the actual world, defined as follows:

Definition 5 *A pointed Kripke model is a tuple $(M, w) = ((Val_w)_{w \in W}, (\rightarrow_a)_{a \in Ag}, w \in W)$ where w is the actual world.*

A slightly different version of a Kripke model is used in Definition 5 than introduced in Definition 4, because it is closer to how the model checker represents models. There are no functional differences between these versions, but the set of worlds and the valuation of each world is combined into 1 set of tuples.

We continue the example of Annie, Bob, and Charles, but more formally now in the form of a pointed Kripke model as defined in Definition 5. To make it more interesting, the room Charles is in does have a window, so $K_c(s \wedge \neg t)$. The situation before the public announcement of the weather can be represented by a Kripke model as

Model 1 $(M, w) = (Val, \{\rightarrow_a, \rightarrow_b, \rightarrow_c\}, w_1)$ *where*

$$\begin{aligned} Val(w_0) &= \emptyset \\ Val(w_1) &= \{s\} \\ Val(w_2) &= \{t\} \\ Val(w_3) &= \{s, t\} \\ \rightarrow_a &= \{\{w_0, w_1\}, \{w_2, w_3\}\} \\ \rightarrow_b &= \{\{w_0, w_2\}, \{w_1, w_3\}\} \\ \rightarrow_c &= \{\{w_0\}, \{w_1\}, \{w_2\}, \{w_3\}\} \end{aligned}$$

$$\begin{tikzcd} w_0 : \emptyset \arrow[r, leftrightarrow, "a"'] \arrow[d, leftrightarrow, "b"] & w_1 : \{s\} \arrow[d, leftrightarrow, "b"'] \\ w_2 : \{t\} \arrow[r, leftrightarrow, "a"] & w_3 : \{s, t\} \end{tikzcd}$$

Figure 2.1: Visual representation of Model 1. All worlds are reflective, so the reflective arrows are left out for clarity.

This model is also represented as a graph in Figure 2.1. Nodes in this graph are worlds, and edges are relations for agents. Since relations are equivalence relations, the edges are bidirectional and there should be edges from every node to itself, but this is left out of the graph for clarity.

2.4.1 Meta-knowledge

A world in a Kripke model is not simply its valuation, but is additionally denoted by a unique integer. To explain the reason for this, we need to understand that knowledge in epistemic logic also models meta-knowledge. This means that an agent can know things about the knowledge of other agents, and those agents can know about that, and so forth. Consider the following formula:

$$s \wedge K_b s \wedge \neg K_b K_c s \wedge \neg K_c s$$

For the sake of clarity we take a simpler version of the example mentioned earlier where the only proposition is whether it is sunny outside, so intuitively the worldspace should be $W = \{\emptyset, \{s\}\}$.

What we say here is that Bob does not know whether Charles knows it is sunny. Bob does know it is sunny, but there are still 2 worlds in which it is sunny that Bob deems possible, namely the one in which Charles knows it is sunny, and the one in which he does not. Because agents can have knowledge about knowledge, propositions are no longer the only thing that distinguish worlds from each other.

This means that in this case the list of valuations is actually $\emptyset, \{p\}, \{p\}$ (also represented in Figure 2.2), but since we want to make every world unique, we use integers, resulting in $W = \{w_0, w_1, w_2\}$ where $Val(w_0) = \emptyset$, $Val(w_1) = \{p\}$, and $Val(w_2) = \{p\}$.

The current representation of our, and in fact any situation is formalized enough for a computer to work with, which allows us to program something that checks the truth of statements in a given situation. A model checker for the (explicit) Kripke models as described in Definition 5 will be compared to a model checker for another representation.

$$\begin{tikzcd} w_0 : \emptyset \arrow[r, leftrightarrow, "c"] & w_1 : \{s\} \arrow[r, leftrightarrow, "b"] & w_2 : \{s\} \end{tikzcd}$$

Figure 2.2: A model in which two different worlds have the same valuation. All worlds are reflective, so the reflective arrows are left out for clarity.

2.5 Succinct Kripke Models

Annie, Bob, and Charlie live in the first apartment of a apartment complex and every apartment has their own thermostat. This means every single apartment can either be warm or not. We represent this with the propositions $t_1, t_2, ..., t_n$ for n apartments where Annie, Bob, and Charlie live in apartment 1. For all 4 worlds listed in Model 1, t_2 can either be warm or not warm, resulting in 8 worlds. For all those worlds, t_3 can be warm or not, making 16 worlds. We can see that the number of worlds grows exponentially for the number of propositions. If the apartment building only holds 10 apartments, an average computer can easily deal with this, but if $n = 100$ or $n = 1000$, this becomes a problem.

The second issue is similar and concerns the agents. Of course, every apartment is inhabited by at least one person. Every agent holds a list of lists containing all worlds exactly once. This means that for the size of the model, we multiply the number of worlds with the number of agents. We need even more worlds with the same valuations in case agents do not know about each other's knowledge. We end up with a space complexity of $\mathcal{O}(a * 2^p)$ where a is the number of agents and p is the number of propositions in a model. Navigating this vast amount of data for big models will be very time consuming for a model checker.

For this and other reasons, Charrier and Schwarzentruber (2017) proposed another representation of situations: one in which worlds are not listed explicitly, and neither are relations between worlds for agents. In these Succinct models, the set of worlds is represented by a boolean formula which holds for only and all worlds in the model that can be made with the vocabulary.

Definition 6 *A pointed Succint Kripke model is a tuple* $(M, s) = (V, \beta_M, (\pi_a)_{a\in Ag}, s \in S)$ *where*

- V *is a finite set of atomic propositions;*
- β_M *is a boolean formula over V;*
- $(\pi_a)_{a\in Ag}$ *is a mental program for each agent;*
- *s is a state in the statespace that is the actual state.*

There are a couple of things to note about Definition 6. First of all, worlds are now called states. This is because here, worlds are not made explicit, so we can not label them with a unique integer. States are only represented by a set of propositions that are true in that state. Another thing is the introduction of mental programs.

2.5.1 Mental Programs

Charrier and Schwarzentruber (2015) propose a representation of knowledge in PAL based on Dynamic Logic of Propositional Assignments (DLPA), which they call mental programs. Given a state, these mental programs can be used to generate all reachable states.

Definition 7 *For any agent and from any given state, reachable states can be calculated by a Mental Program* π *using the following syntax:*

$$\pi ::= p \leftarrow \varphi \mid \varphi? \mid \pi;\pi \mid \pi \cup \pi \mid \pi \cap \pi \mid \pi^{-1}$$

where $p \in V$. *We specify the semantics of the above grammar as:*

- $p \leftarrow \varphi$ *assigns the truth value of* φ *to the proposition p in the current state;*
- $\varphi?$ *tests whether* φ *holds in the current state;*
- $\pi;\pi$ *executes a sequence of mental programs;*
- $\pi \cup \pi$ *executes either mental program;*
- $\pi \cap \pi$ *is the intersection of mental programs;*
- π^{-1} *is the inverse of mental programs.*

Now instead of agents holding a list of every world, every agent has a mental program which does not grow exponentially if more propositions get added to a model.

Considering the version of our example where $V = \{s, w\}$ Annie will have the Mental Program $\pi_a = (s \leftarrow \top) \cup (s \leftarrow \bot)$, which means that from any state, she can reach a state in which s is assigned either true or false, translating to "Annie cannot distinguish between states in which it is sunny and states in which it is not sunny". If we now consider an apartment block with 10 apartments ($V = \{s, w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9, w_{10}\}$), her Mental Program would be

$$\begin{gathered}((s \leftarrow \top) \cup (s \leftarrow \bot));\\ ((w_2 \leftarrow \top) \cup (w_2 \leftarrow \bot));\\ ((w_3 \leftarrow \top) \cup (w_3 \leftarrow \bot));\\ ((w_4 \leftarrow \top) \cup (w_4 \leftarrow \bot));\\ ((w_5 \leftarrow \top) \cup (w_5 \leftarrow \bot));\\ ((w_6 \leftarrow \top) \cup (w_6 \leftarrow \bot));\\ ((w_7 \leftarrow \top) \cup (w_7 \leftarrow \bot));\\ ((w_8 \leftarrow \top) \cup (w_8 \leftarrow \bot));\\ ((w_9 \leftarrow \top) \cup (w_9 \leftarrow \bot));\\ ((w_{10} \leftarrow \top) \cup (w_{10} \leftarrow \bot))\end{gathered}$$

assuming she can only observe t_1. We can observe that her mental program increases in size linearly for every proposition p for which $\neg K_a^? p$ holds. In contrast, Annie's relations $\rightarrow_a$ in the case of a Kripke model increases linearly with every added world, which increases exponentially for every proposition p regardless of her knowledge on it. For the same situation with 10 apartments, $\rightarrow_a$ would be a list of lists of in total $2^{11} = 2048$ worlds.

3 Model Checking

A model checker does the following: given a pointed model (M, s) and a PAL formula φ, output whether $M, s \models \varphi$ holds. Here, (M, s) can be a pointed Kripke Model model as described in Definition 5 with s being the identifier of the actual world, or a pointed succinct model as described in Definition 6 with s being the actual state.

In the following sections, implementations of both an explicit and succinct model checker for PAL will be explained. To save space and maintain focus on the most important parts, not all the code will be given in this thesis. The full program can be found here:

`https://github.com/Maickel-Hartlief/SucExpModelCheckers.git`

3.1 Haskell

The model checkers are implemented in the purely functional programming language Haskell, which has some advantages useful for this particular system. Another big reason for using Haskell is that the system is linked to the Symbolic Model Checking for Dynamic Epistemic Logic (SMCDEL) system developed by Gattinger (2018), which is also implemented in Haskell.

3.2 An Explicit Model-Checker

We start by defining the datatype `Form` that can represent all operators of PAL with the corresponding compoments the operator applies to. This definition can be seen in the code snippet in Figure 3.1. Every operator starts with a keyword, followed by the components it applies to. most operators are recursive. A `Prp` is an `Int` and an `Agent` is a `String`. `[Form]` means the operator applies to a list of formulas.

The next step is to define a `Model`. The implementation of this is shown in Figure 3.2. This definition is directly translated from Definition 5.

The model checking is done by a function that takes a pointed model and a formula and returns a boolean. The implementation of this can be seen in Figure 3.3. It is not necessary to know everything that is going on in that code snippet. One thing worth mentioning however is that the exclamation mark on line 14 is a function that performs the public announcement. It goes through all worlds and agents to delete the worlds that do not satisfy the announced formula. This is very time consuming.

3.3 A Succinct Model-Checker

The succinct model checker uses the same `Form` datatype as for the Kripke model checker (Figure 3.1), but naturally a different `Model` (Figure 3.4) datatype and therefore a different `isTrue` function (Figure 3.5).

The `SuccinctModel` datatype is directly translated from M of Definition 6.

The `sucIsTrue` function mostly works the same, since the model representations mostly differ in how they encode worlds and agent knowledge, which is not used in boolean logic. Apart from the `P` case being able to directly use the current state instead of having to find the actual world in the list of worlds like in Figure 3.3, Only the `Kno` and `Ann` are different.

The algorithm `localstate` In line 13 of Figure 3.3 finds the subset containing the actual world and applies the formula to all worlds of that subset. The algorithm `isStateOf` takes the actual state, generates all reachable states using the mental program of the agent, and applies the formula to all generated states.

The `Ann` cases seem very similar for `isTrue` and `sucIsTrue`, but the implementation of the exclamation mark for succinct and Kripke models are very different. The implementation of the explicit one was explained in Section 3.2. The implementation for succinct models is a great deal simpler: it adds the announced formula to the list of formulas that denote β_M. There is no need to manually delete states, because the states are not explicitly represented. They are generated by the vocabulary and β_M whenever they are needed.

Additionally, succinct Kripke Models use mental programs (Figure 3.6), and a function `reachableFromHere` (Figure 3.7) dedicated to generate states given an actual state and a mental program.

3.4 Translation

In real applications, problems do not come in the form of models. And even if they do, they might come in a different model type than what your model checker works with. To test real performance as accurately as possible, it is relevant to test the speed and accuracy with which problems can be translated to a given model type. Additionally, it helps connecting the inputs for the different model checkers to each others, reducing confounding variables.

Arguably, having a common starting point and translating that to both an explicit Kripke model and a succinct Kripke model would be balanced choice. However, the form of the common starting point is an entirely different research question and outside of the scope of this project.

A two-way translator from succinct to explicit and back was implemented to analyze the structure of the models. It has been included in the result,

```haskell
data Form
  = Top                             -- ^ True Constant
  | Bot                             -- ^ False Constant
  | PrpF Prp                        -- ^ Atomic Proposition
  | Neg Form                        -- ^ Negation
  | Conj [Form]                     -- ^ Conjunction
  | Disj [Form]                     -- ^ Disjunction
  | Impl Form Form                  -- ^ Implication
  | Equi Form Form                  -- ^ Bi-Implication
  | K Agent Form                    -- ^ Knowing that
  | Ck [Agent] Form                 -- ^ Common knowing that
  | Kw Agent Form                   -- ^ Knowing whether
  | PubAnnounce Form Form           -- ^ Public announcement that
    deriving (Eq,Ord,Show)
```

Figure 3.1: ***Form*** **datatype for all operators of PAL, used by both model checkers. Imported from** ***SMCDEL.Language*****. The unused operators (Xor, Forall, Exists, Ck, Ckw, PubAnnounceW, Announce, AnnounceW, and Dia) are exlcluded.**

```haskell
data Model = Mo [(World, [Prp])] [(Agent, [[World]])] deriving (Eq,Ord,Show)
```

Figure 3.2: ***Model*** **datatype for explicit Kripke models**

though few conclusions can be based on it due to the flaws in the Kripke to Succinct translator.

Translating the succinct version to the Kripke one works well. Because of the explicitness of this representation, there is little room for variations in how a particular model is represented.

However, mental programs and succinct models with a more complicated β_M can be expressed in a myriad of ways. Many generalizations were made in the translator from explicit to succinct, which resulted in an unnecessarily long model representation. The succinct model resulting from translating the explicit one was substantially larger than the one generated from scratch. For 5 muddy children, the kripke model is approximately 4 times larger than the succinct one. However, the succinct model translated from the Kripke model is approximately 27 times larger than the Kripke model. The point of using succinct models is to save memory, so this is not a useful model.

4 Benchmarking

4.1 The Muddy Children problem

A classic example of a scenario involving epistemic logic and public announcements introduced by Littlewood (1953) as a logically equivalent story is the Muddy Children problem:

Three children are playing outside, knowing that if they get muddy, there will be consequences. They all want to stay clean while making the others muddy. Some of the children do end up getting mud on their forehead, without their siblings saying anything about it. Now the father enters the stage, announcing "At least one of you has mud on their forehead"[1], after which he prompts "please step forward if you know whether you have mud on your face"[2] over and over. Assuming that the children are perfect logicians and they can see each other's faces but not their own, what will happen?

In the case that only one child is muddy, being a perfect logician, they will immediately figure out that the other children are all not muddy. Therefore, they must be muddy themselves.

In the case that two of the children are muddy, nothing happens after the first iteration of the second announcement [2]. However, the children can deduce from this that at least two children are muddy, because otherwise something would have happened. After the second iteration of announcement [2], the 2 children with mud on their forehead will see one other child with mud on their forehead. They will deduce that they must also have mud on their forehead.

In the case all three children are muddy, after not seeing anything happen after the second iteration of announcement [2], they now know at least three children are muddy using the same reasoning for the previous case. We observe that the number of children that are muddy will be the number of times announcement [2] will need to be made for the muddy children to know their own state. The children without mud on their face also know that they are not muddy as soon as the muddy children step forward, because otherwise the muddy children would not have known their own state yet.

Model 2 is the Kripke model of this problem and Model 3 a succinct model. Figure 4.1 shows a visual representation of the explicit model.

Model 2 *The explicit Kripke model for the three muddy children problem is* $M =$

```haskell
isTrue :: (Model,Int) -> Form -> Bool
isTrue _  Top                    = True
isTrue _  Bot                    = False
isTrue (Mo val _,w) (PrpF p)     = p `elem` unsafeLookup w val
isTrue a (Neg f)                 = not $ isTrue a f
isTrue a (Conj fs)               = all (isTrue a) fs
isTrue a (Disj fs)               = any (isTrue a) fs
isTrue a (Impl f g)              = not (isTrue a f) || isTrue a g
isTrue a (Equi f g)              = isTrue a f == isTrue a g
isTrue (m, w) (K i f) =
  all
    (\w' -> isTrue (m,w') f)
    (localState (m, w) i)
isTrue a (Kw i f)                = isTrue a (Disj [ K i f, K i (Neg f) ])
isTrue (m, w) (PubAnnounce f g)  = not (isTrue (m,w) f) ||
                          isTrue (m ! f, w) g
```

Figure 3.3: *isTrue* **function for explicit model checker. The excluded operators mentioned in Figure 3.1 are also excluded here.**

```haskell
data SuccinctModel = SMo [Proposition] Form [Form] [(Agent, MenProg)] deriving (Eq,Ord,
    Show)
```

Figure 3.4: `SuccinctModel` datatype for succinct Kripke models

$\{Val(w_0), Val(w_1), Val(w_2), Val(w_3), Val(w_4), Val(w_5), Val(w_6), Val(w_7), Val(w_8)\}, \{\rightarrow_{c_1}, \rightarrow_{c_2}, \rightarrow_{c_3}\}$ *where*

$$Val(w_0) = \emptyset$$
$$Val(w_1) = \{m_1\}$$
$$Val(w_2) = \{m_2\}$$
$$Val(w_3) = \{m_3\}$$
$$Val(w_4) = \{m_1, m_2\}$$
$$Val(w_5) = \{m_1, m_3\}$$
$$Val(w_6) = \{m_2, m_3\}$$
$$Val(w_7) = \{m_1, m_2, m_3\}$$
$$\rightarrow_{c_1} = \{\{w_0, w_1\}, \{w_2, w_4\}, \{w_3, w_5\}, \{w_6, w_7\}\}$$
$$\rightarrow_{c_2} = \{\{w_0, w_2\}, \{w_1, w_4\}, \{w_3, w_6\}, \{w_5, w_7\}\}$$
$$\rightarrow_{c_3} = \{\{w_0, w_3\}, \{w_1, w_5\}, \{w_2, w_6\}, \{w_4, w_7\}\}$$

where m_n is the atomic proposition of whether child c_n is muddy.

Model 3 *The Succinct model for the three muddy children problem is $M = (V, \beta_M, (\pi_a)_{a\in Ag})$ where*

$$V = \{m_1, m_2, m_3\}$$
$$\beta_M = \top$$
$$\pi_{c_1} = (m_1 \leftarrow \top) \cup (m_1 \leftarrow \bot)$$
$$\pi_{c_2} = (m_2 \leftarrow \top) \cup (m_2 \leftarrow \bot)$$
$$\pi_{c_3} = (m_3 \leftarrow \top) \cup (m_3 \leftarrow \bot)$$

where m_n is the atomic proposition of whether child c_n is muddy.

As we can see, this is too small of a model to really have a difference between the two representations. There are only three agents and three propositions, and a maximum of three announcements need to be made for all the children to know their own state. The beautiful thing about this problem, however, is its scalability. We can simply replace "three" with n where $n \geq 1$ and say that the number of muddy children is m where $0 \leq m \leq n$. We now have a generalized the Children problem, where a model has n propositions and n agents, and m announcements need to be made for the children to know their own state. With this, we can start testing the performance of the two model checkers.

4.2 Implementation

The models need to be generated, since different values for n or m yield different models. A succinct and explicit version of a function was written that takes n and m and output a model for the muddy children problem with n children of which m are muddy.

The function used for testing the model checkers calls the function that generates the model and publicly announces announcement [1] and then repeats [2] from the problem description in Section 4.1 until all children know their own state. The time needed to run this function is what we are interested in.

Generating these models is included in these results, but the time it takes is negligible. The full implementation can be seen at the URL provided at the start of Section 3, but will not be covered here. The result is in the same form as the examples for three children in Models 2 & 3.

```haskell
sucIsTrue :: (SuccinctModel, State) -> Form -> Bool
sucIsTrue _  Top                          = True
sucIsTrue _  Bot                          = False
sucIsTrue (_ ,s) (PrpF p)                 = p `elem` s
sucIsTrue a (Neg f)                       = not $ sucIsTrue a f
sucIsTrue a (Conj fs)                     = all (sucIsTrue a) fs
sucIsTrue a (Disj fs)                     = any (sucIsTrue a) fs
sucIsTrue a (Impl f g)                    = not (sucIsTrue a f) || sucIsTrue a g
sucIsTrue a (Equi f g)                    = sucIsTrue a f == sucIsTrue a g
sucIsTrue (m@(SMo v _ _ rel), s) (K i f) =
   all
    (\s' -> sucIsTrue (m,s') f)
    (filter (`isStateOf` m) $ reachableFromHere v (unsafeLookup i rel) s)
sucIsTrue a (Kw i f)                      = sucIsTrue a (Disj [ K i f, K i (Neg f) ])
sucIsTrue (m, s) (PubAnnounce f g) = not (sucIsTrue (m, s) f) ||
                                     sucIsTrue(m ! f, s) g
```

Figure 3.5: `sucIsTrue` function for succinct model checker. The excluded operators mentioned in Figure 3.1 are also excluded here.

```haskell
data MenProg =
                Ass Prp Form      -- Assign prop to truthvalue of form
              | Tst Form          -- Test form
              | Seq [MenProg]     -- Execute forms sequencially
              | Cup [MenProg]     -- execute either of the forms
              | Cap [MenProg]     -- intersection of forms
              | Inv MenProg       -- inverse of form
  deriving (Show, Eq, Ord)
```

Figure 3.6: `MenProg` datatype for all operators of mental programs, used by agents in succinct models

4.3 results

The Succinct model checker and the Kripke model checker are both tested on the muddy children problem with a different number of (muddy) children to test their actual performance. These results are shown in Figure 4.2. This graph also shows the computation time of the DEMOS5 sytem by van Eijck (2014), as well as the results of starting with the other of the 2 representations and translating to the different model type before model checking.

The translating can be used as a way of calibrating the two representations. The Kripke and succinct models for the muddy children were created separately from each other, so there might be discrepancies between them that have an unwanted impact on the results. By using both starting models to test both model checkers, these discrepancies can be analysed.

We can see in the graph that DEMOS5 and the Explicit model checker from both a Succinct model and an Kripke model starting points perform similarly. Though there are minor differences, the lines are close to each other, have roughly the same shape and do not seem to diverge. The translator from succinct Models to Kripke models works well, but the other way around does not, which is explained in Section 3.4 and can be seen in Figure 4.2. The lines of the Succinct model checker with translation takes longer than the one without translation, and the difference is larger than the translation itself. The lines also diverge.

5 Conclusions

5.1 Summary

Model checkers are important, but models can suffer from being memory inefficient. More succinct representations of models usually takes longer to model check. However, Charrier and Schwarzentruber (2017) provide a framework for succinct models that is exponentially more succinct, but checking is PSPACE-complete just like Kripke Model checking. Kripke and succinct model checkers were implemented and tested on the muddy children problem to analyze the actual performance of a succinct model checker relative to a Kripke model checker.

5.2 Answers

In Figure 4.2 we can see the performance of several model checkers applied to several representations of models. The lines for DEMOS5 and ones that were translated first are for evaluating the reliability of the data. The results we are interested in are the Succinct model and the Kripke model. We can see that not only is the Succinct model checker slower, the difference in performance

```haskell
reachableFromHere :: [Prp] -> MenProg -> State -> [State]
reachableFromHere _ (Ass p f) s       = if boolIsTrue s f
                                            then [sort $ union [p] s]
                                            else [delete p s]
reachableFromHere _ (Tst f) s         = [ s | boolIsTrue s f ]
reachableFromHere _ (Seq []) s        = [ s ]
reachableFromHere v (Seq (mp:rest)) s = concat [ reachableFromHere v (Seq rest) s'
                                              | s' <- reachableFromHere v mp s ]
reachableFromHere _ (Cup []) _        = []
reachableFromHere v (Cup (mp:rest)) s = nub $    reachableFromHere v mp s
                                             ++ reachableFromHere v (Cup rest) s
reachableFromHere _ (Cap []) _        = []
reachableFromHere v (Cap (mp:rest)) s = reachableFromHere v (Cap rest) s `intersect`
                                        reachableFromHere v mp s
reachableFromHere v (Inv mp)    s = [s' | s' <- allStatesFor v, areConnected v mp s' s]
```

Figure 3.7: ***reachableFromHere*** **function for knowledge in succinct model checker**

$$\begin{tikzcd}
& w_4 : \{m_1, m_2\} \arrow[rr, leftrightarrow, "c_3"'] \arrow[dl, leftrightarrow, "c_1"] \arrow[dd, leftrightarrow, "c_2"] & & w_7 : \{m_1, m_2, m_3\} \arrow[dl, leftrightarrow, "c_1"] \arrow[dd, leftrightarrow, "c_2"] \\
w_2 : \{m_2\} \arrow[rr, leftrightarrow, "c_3"'] \arrow[dd, leftrightarrow, "c_2"] & & w_6 : \{m_2, m_3\} \arrow[dd, leftrightarrow, "c_2"] & \\
& w_1 : \{m_1\} \arrow[rr, leftrightarrow, "c_3"'] \arrow[dl, leftrightarrow, "c_1"] & & w_5 : \{m_1, m_3\} \arrow[dl, leftrightarrow, "c_1"] \\
w_0 : \emptyset \arrow[rr, leftrightarrow, "c_3"'] & & w_3 : \{m_3\} &
\end{tikzcd}$$

Figure 4.1: Visual representation of the Muddy Children problem from Model 2. All worlds are reflective, so the reflective arrows are left out for clarity.

grows for larger inputs. This does not necessarily conflict with the conclusion of Charrier and Schwarzentruber (2017) who stated that the time-complexity of Succinct models and Kripke models is both PSPACE-complete. However, in practice it does mean that when choosing a representation there is still a trade-off between succinctness and time-complexity for some models, even considering the more promising Succinct models.

Important to note, however, is that this does not necessarily apply to all models. Though model checking knowledge is very time-consuming, making a public announcement is faster than for Kripke models. It makes little sense to not check anything involving knowledge after making a public announcement though, so these two will often appear together, just like in the Muddy Children problem.

6 Discussion

6.1 problems

In the following section we will discuss some potential problems with this research that have been considered but not fully solved.

6.1.1 Test problem

Though as discussed in Section 4.1 the Muddy Children problem is very suitable for testing the performances of model checkers, it is only one problem with very few different announcements. Not all operators are used in this problem, meaning that to fully test the performance of the model checker, a more varied set of problems would need to be tested.

6.1.2 Expressiveness

Though presented as being equally expressive, succinct models actually do lack the ability to express a specific type of scenario, where Kripke models have no issue with them. This example concerns meta-knowledge and is discussed in Section 2.4.1. Since succinct models use states which only hold the valuation of itself and no unique integer or other marker, worlds with the same valuation that are still different for agents because of meta-knowledge are one and the same state. In its standard form, this would make succinct models less expressive than Kripke models. However, there is a reasonably straightforward fix with low cost to complexity. This fix is a proposition with no semantic meaning that is true in one of the identical states and false in the other, functioning solely as a way to make these states distinct. This fix was implemented into the translator, but for succinct models that are made from scratch, the user would have to do this manually.

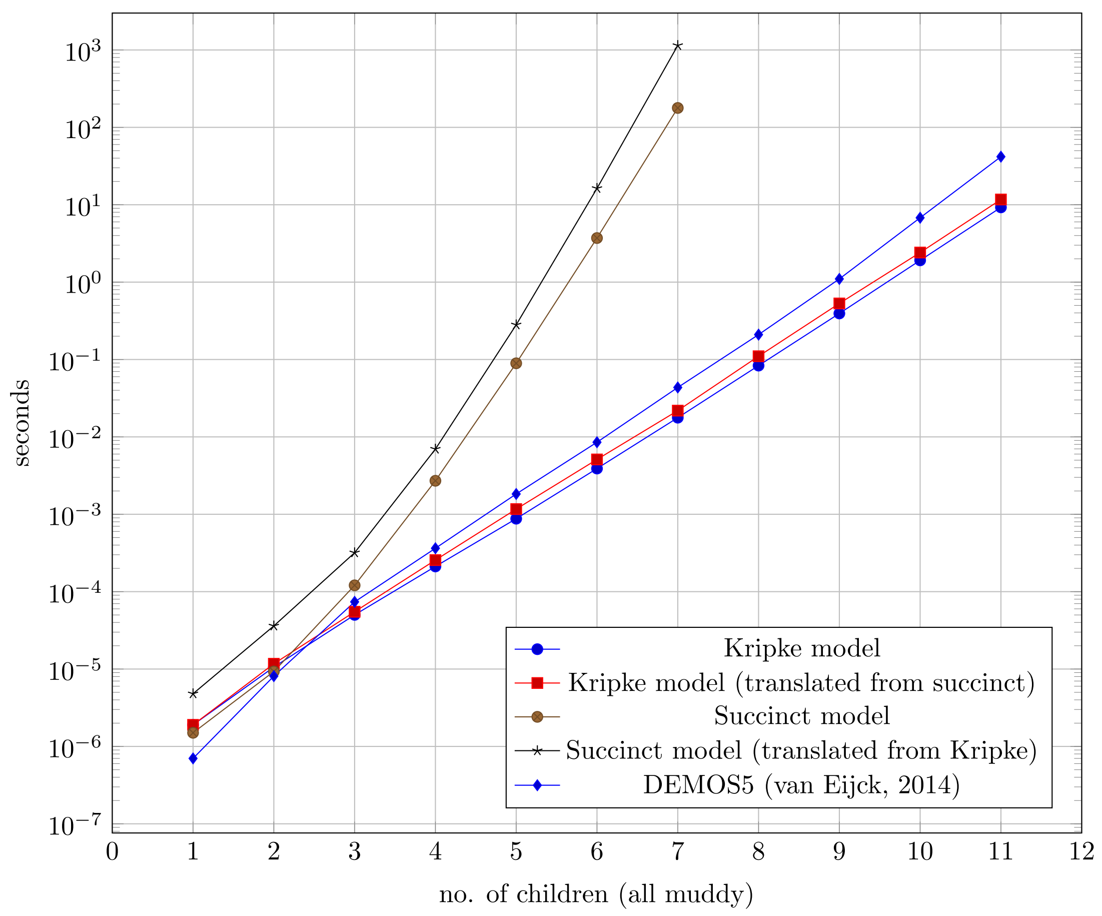

Figure 4.2: Muddy Children benchmark results (logarithmic scale).

6.1.3 Inefficiencies

Programming languages do a lot of things in the background that a programmer rarely sees or even knows about. Haskell is no exception. I am not an expert in the language, and even if I were I could have missed some things. Perhaps I unnecessarily call a general function while a more specific function with lower time-complexity would have sufficed as well.

These factors are kept to a minimum since both model checkers are implemented in the same programming language, and a profiler was used to analyze the time spent in each function and locate possible inefficiencies. However, there might still be some.

6.1.4 Translations

The models for the muddy children were generated completely separately, meaning they weren't written in a common representation and then translated to both model representations. A translator was used as well, but this yielded very large succinct models, defeating their purpose of being more succinct. This means there is no way to guarantee that the implementations of the models are optimal. Luckily the Muddy Children problem is straightforward, but there might still be a more efficient implementation that could make a difference in the results.

6.2 Future Research

There are of course more things to be discovered around this topic. In the following sections, we will go over some ideas that would be interesting and useful to focus on in the future.

6.2.1 Improving the Translators

A functioning but non-optimal translator was implemented for this project, but research into optimal translators could really help out model checkers in both research and applications. Being able to translate models to other types of models is very useful in research about these representations. Comparisons between modeltypes become easier and more reliable, and researchers can more easily use the implementations and results of other research. Additionally, it would be a good tool to expand compatibility of model checkers to other types of models.

6.2.2 DEL

PAL is only the simplest version of DEL. It would be useful to extend the current system to work for and test more operators to extend the language from PAL to full DEL. This would not have to be a completely new model checker. This project can simply be extended upon.

6.2.3 Humans as Model Checkers

Model checker programs are great, but in the past and for the foreseeable future, it is humans who will solve problems and write solutions. Even if computers far outperform humans in model checking, it is still a useful practical and psychological question to ask how humans check models. Additionally, maybe we can learn something from how humans do it and improve computer model checkers, like in many other fields. Apart from figuring out how humans check models, there are some other interesting questions like: how do humans represent models? How do humans translate from a problem description in natural language to a more formal model representation? Is the human model representation static or ever changing, and does it change significantly between different types of problems or different people? Does the way humans parse problems, encode models, and check models teach us something about common mistakes humans make? Are human models more succinct or explicit?

References

Guillaume Aucher and François Schwarzentruber. On the complexity of dynamic epistemic logic. *CoRR*, abs/1310.6406, 2013. URL `http://arxiv.org/abs/1310.6406`.

Philippe Balbiani, Andreas Herzig, François Schwarzentruber, and Nicolas Troquard. DL-PA and DCL-PC: model checking and satisfiability problem are indeed in PSPACE. *CoRR*, abs/1411.7825, 2014. URL `http://arxiv.org/abs/1411.7825`.

Tristan Charrier and François Schwarzentruber. Arbitrary public announcement logic with mental programs. AAMAS '15, page 1471–1479, Richland, SC, 2015. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450334136. URL `http://people.irisa.fr/Francois.Schwarzentruber/publications/AAMAS2015_arbitrary_public_announcements_with_mental_programs.pdf`.

Tristan Charrier and François Schwarzentruber. A succinct language for dynamic epistemic logic. In S. Das, E. Durfee, K. Larson, and M. Winikoff, editors, *16th Conference on Autonomous Agents and Multiagent Systems AAMAS*, pages 123–131, 2017. URL `http://www.aamas2017.org/proceedings/pdfs/p123.pdf`. Long version available at `https://hal.archives-ouvertes.fr/hal-01487001/`.

Malvin Gattinger. *New directions in model checking dynamic epistemic logic*. PhD thesis, University of Amsterdam, 2018. URL `https://dare.uva.nl/search?identifier=74ef40e9-42e7-45e1-b2b8-0cca7bc31a0f`.

Jacques-Louis Lions, Lennart Luebeck, Jean-Luc Fauquembergue, Gilles Kahn, Wolfgang Kubbat, Stefan Levedag, Leonardo Mazzini, Didier Merle, and Colin O'Halloran. Ariane 5 flight 501 failure report by the inquiry board, 1996. URL `http://esamultimedia.esa.int/docs/esa-x-1819eng.pdf`.

John Edensor Littlewood. *A mathematician's miscellany.* London, 1953. doi:https://doi.org/10.1090/S0002-9904-1955-09949-X.

Carsten Lutz. Complexity and succinctness of public announcement logic. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '06, page 137–143, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933034. doi:10.1145/1160633.1160657. URL `https://doi.org/10.1145/1160633.1160657`.

H. van Ditmarsch, W. van der Hoek, and B. Kooi. *Dynamic Epistemic Logic*. Synthese Library. Springer Netherlands, 2007. ISBN 9781402058394. URL `https://books.google.nl/books?id=dKRQPHvOIGQC`.

Jan van Eijck. DEMO-S5. Technical report, CWI, 2014. URL `https://homepages.cwi.nl/~jve/software/demo_s5`.