Comparing State-Representations for DEL Model Checking

Gregor Behnke Malvin Gattinger Haitian Wang Avijeet Ghosh

ILLC, University of Amsterdam
Amsterdam, The Netherlands

Chennai Mathematical Institute
Chennai, India

g.behnke@uva.nl malvin@w4eg.eu h.wang12@uva.nl avi.ghosh23@gmail.com

Model checking with the standard Kripke models used in (Dynamic) Epistemic Logic leads to scalability issues. Hence alternative representations have been developed, in particular symbolic structures based on Binary Decision Diagrams (BDDs) and succinct models based on mental programs. While symbolic structures have been shown to perform well in practice, their theoretical complexity was not known so far. On the other hand, for succinct models model checking is known to be PSPACE-complete, but no implementations are available.

We close this gap and directly relate the two representations. We show that model checking DEL on symbolic structures encoded with BDDs is also PSPACE-complete. In fact, already model checking Epistemic Logic without dynamics is PSPACE-complete on symbolic structures. We also provide direct translations between BDDs and mental programs. Both translations yield exponential outputs. For the translation from mental programs to BDDs we show that no small translation exists. For the other direction we conjecture the same.

1 Introduction

Reasoning about knowledge and its representation has become an important line of study in Artificial Intelligence, most importantly in Epistemic Planning [5]. A widely used logical framework to reason about the knowledge of multiple intelligent agents is Epistemic Logic, a version of Modal Logic [8]. Its standard semantics uses Kripke models with possible worlds to represent what agents know. Moreover, knowledge can change. For example in a card game, a player may announce its card to a subset of players privately. Then the knowledge of all players changes: some players come to know the card, and other players may still observe that such a private announcement is made. To formalise and reason about such knowledge updates, a widely used framework is Dynamic Epistemic Logic (DEL) [4, 13], of which the simplest version is Public Announcement Logic (PAL) [25, 23].

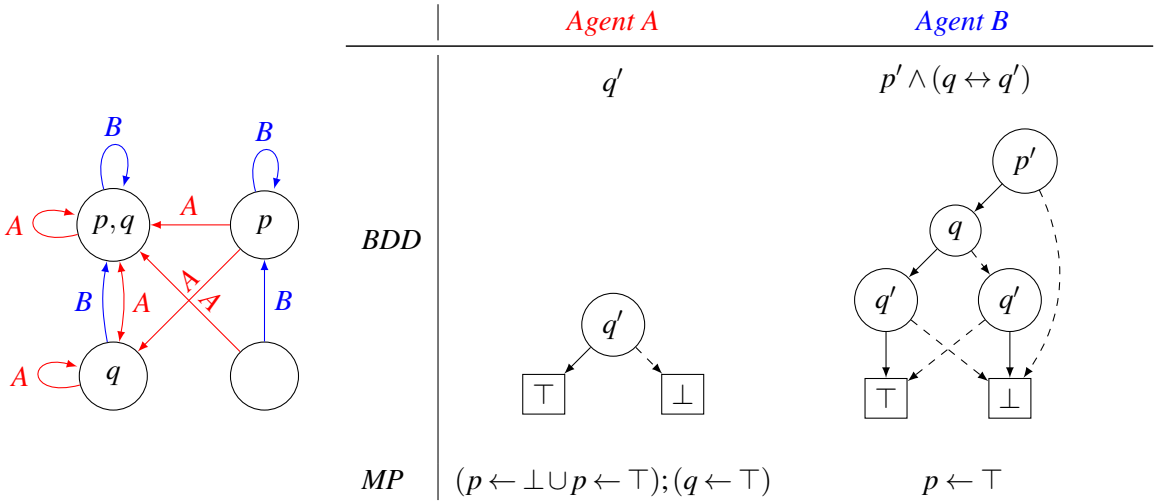
One way to make DEL and PAL practically useful is *model checking*: given a model and a formula, decide whether the formula holds. It is known that on Kripke models the computational complexity of this task for PAL is in P and for DEL is in PSPACE [1]. Model-checkers for PAL have been used in verifying protocols with security applications like the Russian card problem [14, 22]. Moreover, DEL model checkers like SMCDEL [17] can be used in combination with large language models (LLMs) in order to improve the explainability and reasoning ability compared to plain LLMs [26].

Model checking has scalability issues, because Kripke models enumerate all possible situations. For example, every possible way to distribute cards among players needs a possible world, leading to an exponentially large model. Different techniques have been developed to represent Kripke models more compactly, to make epistemic model-checking easier. Here we focus on two such works for DEL:

- Symbolic structures based on Binary Decision Diagrams (BDDs) [7, 17, 24]. This approach uses the famous data structure from [9] and is similar to techniques for temporal logics [10].
- Succinct models based on mental programs (MPs) [12], a variant of PDL [16], and similar to regular languages. In particular these programs can encode the epistemic accessibility relations.

A. Bjorndahl (Ed.): Theoretical Aspects of
Rationality and Knowledge 2025 (TARK 2025)
EPTCS ??, 2025, pp. 1–18, doi:10.4204/EPTCS.??.??

Example 1.1. *To illustrate the different representations of Kripke models, consider the model below. It has four worlds and two agents. Agent A considers all q-worlds possible and knows nothing else, whereas Agent B only considers p-worlds possible and knows the value of q. Next to the model we show how these relations are encoded by BDDs and by mental programs. We note that one relation results in a relatively small BDD, while the other is more suitable to be encoded as a short mental programs.*



Both approaches lead to a new model-checking problem, because when moving from one representation to another, the complexity of model checking can differ. Interestingly, for the succinct representation theoretical results about the model checking complexity are available, but not yet for the symbolic structures. With the present article we close this gap. We summarise the known and new results in Table 1. Note that when using symbolic structures, model checking EL, PAL and DEL all become equally hard. We discuss why this happens after we introduce knowledge structures in Definition 2.1.

Representation	Kripke models	Symbolic Structures	Succinct models
EL-S5	in P [15]	PSPACE: Theorem 2.11	-
PAL-S5	in P [13]	PSPACE: Theorem 2.11	-
DEL-S5	PSPACE [20]	PSPACE: Theorem 2.17	-
EL-K	in P [15]	PSPACE: Theorem 2.16	PSPACE [2]
PAL-K	in P [8]	PSPACE: Theorem 2.16	PSPACE [12]
DEL-K	PSPACE [1]	PSPACE: Theorem 2.16	PSPACE [12]

Table 1: Model checking complexity results. We write "PSPACE" for PSPACE-complete here.

While the EL and PAL rows in Table 1 show an easier complexity for Kripke models than for symbolic structures, the key difference lies in input size: symbolic structures can represent Kripke models exponentially more succinctly. Thus, polynomial-time algorithms for Kripke models may become exponential when measured against symbolic input. On the other hand, the DEL rows show that symbolic structures do not lead to an additional complexity blowup. The results suggest that one should not translate Kripke models to other representations, but work directly with the symbolic or succinct representation, i.e. a given system description should be formalized directly into a structure with BDDs or mental programs, to avoid ever using exponential memory that would be needed by a Kripke model.

The symbolic structures have been implemented and benchmarks show that they outperform standard Kripke models, but the succinct representations have not been implemented, with the exception of [21].

To better understand the relation between the different encodings of relations, we provide translations from BDDs to mental programs and vice versa, and study the translation complexity.

In the remainder of this section we define the languages. In section 2 we prove that symbolic model checking is PSPACE-complete, first for PAL-S5, then for DEL-K. In section 3 we recall mental programs used in succinct DEL, and in section 4 we provide translations between BDDs and mental programs. We conclude with comments about multi-pointed models and ongoing implementation work in section 5.

Definition 1.2. *Throughout this article we work with finite vocabularies V that come with an order. For any vocabulary V we define* Boolean formulas $\beta \in \mathscr{L}_B(V)$ *by* $\beta ::= \top \mid \bot \mid p \mid \neg\beta \mid \beta \wedge \beta$ *where* $p \in V$. *A* state *is a subset of the vocabulary $s \subseteq V$. That is, we identify states with the atomic propositions that are true at them. The Boolean semantics are defined as usual and denoted by $s \vDash \beta$.*

To work with both PAL and DEL, we will use a general language definition with a parameter set D.

Definition 1.3. *For any vocabulary V and any set D the language $\mathscr{L}_D(V)$ is given by $\varphi ::= \top \mid \bot \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_i\varphi \mid [d]\varphi$ where $p \in V$, $i \in A$ and $d \in D$. Concretely, $\mathscr{L}_{\mathsf{EL}}$ is given by $D = \varnothing$, $\mathscr{L}_{\mathsf{PAL}}$ is given by $D = \{!\varphi \mid \varphi \in \mathscr{L}_{\mathsf{PAL}}\}$, and $\mathscr{L}_{\mathsf{DEL}}$ is given by letting D be the set of events from Definition 2.3.*

The simpler logic we consider is Public Announcement Logic (PAL), where the only actions are truthful public announcements, i.e. D is the set of all formulas, used as announcements. More general actions include private announcements or factual (also called ontic) changes. In DEL these are usually modeled by letting D be the set of all pointed action models [4, 13], but here we will mostly use the transformers from [7]. Yet another option are the succinct event models from [12]. All these languages come with a mutual but well-founded recursion: elements of D are made using formulas, and formulas use elements of D. We refer to [13, Chapter 6] for details.

The following definition will be relevant to determine the input size of model checking problems.

Definition 1.4. *We define the length $|\cdot| \colon \mathscr{L}_{\mathsf{PAL}}(V) \cup \mathscr{L}_{\mathsf{DEL}}(V) \to \mathbb{N}$ of formulas as follows*

$$\begin{array}{llllllllll} |\top| & := & 1 & |\neg\varphi| & := & |\varphi|+1 & |[!\psi]\varphi| & := & |\psi|+|\varphi|+1 \\ |\bot| & := & 1 & |\varphi_1 \wedge \varphi_2| & := & |\varphi_1|+|\varphi_2|+1 & |[\mathscr{X},x]\varphi| & := & |\mathscr{X}|+|\varphi|+1 \\ |p| & := & 1 & |K_i\varphi| & := & |\varphi|+1 & & & \end{array}$$

where $|\mathscr{X}|$ denotes the size of a transformer as in Definition 2.4 below.

2 Symbolic Structures with BDDs

We recall the definition and main features of the knowledge and belief structures from [7, 17]. In general we assume that elements of $\mathscr{L}_B$ are represented using Binary Decision Diagrams (BDDs) from [9].

Definition 2.1. *A* knowledge structure *is a tuple $\mathscr{F} = (V, \theta, O)$ where V is a finite set called* vocabulary*, $\theta \in \mathscr{L}_B(V)$ is the* state law *and $O_i \subseteq V$ for each $i \in A$ are the* observables*. A* state *of $\mathscr{F}$ is a $s \subseteq V$ such that $s \vDash \theta$. A* pointed *knowledge structure is a tuple $(\mathscr{F}, s)$ where s is a state of $\mathscr{F}$.*

Knowledge structures encode the knowledge of agents (that in Kripke models is represented by a relation) with a set of observables. Each $O_i \subseteq V$ encodes a relation $R_i \subseteq \mathscr{P}(V) \times \mathscr{P}(V)$ given by $R_i st \iff s \cap O_i = O_i \cap t$. Note that this is always an equivalence, matching S5 Kripke models. Hence the corresponding frameworks are called EL-S5, PAL-S5 and DEL-S5. The more general belief structures (for EL-K, PAL-K and DEL-K) can encode arbitrary relations using formulas over a double vocabulary. In addition to this, note that in Table 1, even though model checking for PAL and EL were much easier than DEL (P for PAL,EL whereas PSPACE for DEL) using Kripke models, when it comes to model

checking using symbolic structures, all three of them becomes equally hard (PSPACE). This is because all possible states in a Kripke model are listed distinctly when the model is represented in a general Kripke structure. Therefore, going over all of them does not cost any computational resource beyond the size of the Kripke model itself. But when represented using symbolic structures, only the set of propositions V is part of the input. Hence in order to iterate through indistinguishable possibilities of an agent (for example for checking formulas with $\hat{K}_i$ modalities), all possible (at most $2^{|V|}$) states that come out of V have to be traversed and checked. Since such $\hat{K}_i$ modalities are present in EL, PAL as well as DEL, model checking all three of them becomes equally hard.

Definition 2.2. *A* belief structure *is a tuple $\mathscr{F} = (V, \theta, \Omega)$ where V and θ are as in Definition 2.1 and for each agent i we have an* observation law $\Omega_i \in \mathscr{L}_B(V \cup V')$ *where V' denotes a fresh copy of V.*

The intuition behind Ω_i is that it is true at a pair of states iff the states are related. That is, a Boolean formula $\Omega_i \in \mathscr{L}_B(V \cup V')$ encodes the relation $R_i \subseteq \mathscr{P}(V) \times \mathscr{P}(V)$ given by $R_i st :\iff s \cup t' \vDash \Omega_i$. For example, if $\Omega_{\text{Alice}} = p \wedge q'$ then from any state where p is true Alice will consider any other state where q is true possible. In particular we will have a loop at the state $\{p,q\}$ because $\{p,q\} \cup \{p,q\}' \vDash p \wedge q'$. This encoding is also widely used for temporal model checking [10, Section 8.3].

Analogous to how symbolic structures encode Kripke models, transformers encode action models. To keep track of old propositional values before factual change, we introduce another set of fresh variables, denoted by the operation $(\cdot)^\circ$. Just as V' is a fresh copy of V, V_-° is a fresh copy of V_-.

Definition 2.3. *A* transformer *is a tuple $\mathscr{X} = (V^+, \theta^+, V_-, \theta_-, \Omega_i^+)$ where V^+ is such that $V \cap V^+ = \varnothing$, $\theta^+ \in \mathscr{L}_{\mathsf{DEL}}(V \cup V^+)$ is called the* event law*, $V_- \subseteq V$ is called the* modified subset*, $\theta_- : V_- \to \mathscr{L}_B(V \cup V^+)$ is called the* change law *and $\Omega_i^+ \in \mathscr{L}_B(V \cup V')$ for each i. An* event *is a pair $(\mathscr{X}, x)$ where $x \subseteq V^+$. To update $(\mathscr{F}, s)$ with (χ, x), let $\mathscr{F} \times \chi := (V^{new}, \theta^{new}, \Omega_i^{new})$ where $V^{new} = V \cup V^+ \cup V_-^\circ$,*

$$\theta^{new} = [V_- \mapsto V_-^\circ](\theta \wedge \|\theta^+\|_{\mathscr{F}}) \wedge \bigwedge_{q \in V^-} (q \leftrightarrow [V_- \mapsto V_-^\circ](\theta_-(q)))$$

where $[p \mapsto \psi]\varphi$ denotes a substitution, $\Omega_i^{new} = ([V_- \mapsto V_-^\circ][(V_-)' \mapsto (V_-^\circ)']\Omega_i) \wedge \Omega_i^+$, and lastly the new actual state is $s^x := (s \setminus V_-) \cup (s \cap V_-)^\circ \cup x \cup \{p \in V_- | s \cup x \vDash \theta_-(p)\}$.

Since our goal is a complexity study of model-checking algorithms, we need to be precise about the size of all structures we take as inputs. We define the size of transformers as follows.

Definition 2.4. *Given a transformer $\mathscr{X} = (V^+, \theta^+, V_-, \theta_-, \Omega_i^+)$, we define its size by mutual recursion with Definition 1.4 as $|\mathscr{X}| := |V^+| + |\theta^+| + |V_-| + \sum_{p \in V_-} |\theta_-(p)| + \sum_{i \in I} |\Omega_i^+|$.*

Also Definition 2.3 above and the following two definitions are mutually recursive.

Definition 2.5. *The semantics for $\mathscr{L}_{\mathsf{PAL}}$ and $\mathscr{L}_{\mathsf{DEL}}$ on knowledge and belief structures are as follows. We omit the standard Boolean cases.*

1. *For knowledge structures: $(\mathscr{F}, s) \vDash K_i\varphi$ iff for all states t of $\mathscr{F}$, if $s \cap O_i = t \cap O_i$, then $(\mathscr{F}, t) \vDash \varphi$.*
   For belief structures: $(\mathscr{F}, s) \vDash K_i\varphi$ iff for all states t of $\mathscr{F}$, if $s \cup t' \vDash \Omega_i$, then $(\mathscr{F}, t) \vDash \varphi$.
2. *$(\mathscr{F}, s) \vDash [!\psi]\varphi$ iff $(\mathscr{F}, s) \vDash \psi$ implies $(\mathscr{F}^\psi, s) \vDash \varphi$ where $\mathscr{F}^\psi := (V, \theta \wedge \|\psi\|_{\mathscr{F}}, O)$.*
3. *$(\mathscr{F}, s) \vDash [\mathscr{X}, x]\varphi$ iff $(\mathscr{F}, s) \vDash [x \sqsubseteq V^+]\theta^+$ implies $(\mathscr{F} \times \mathscr{X}, s^x) \vDash \varphi$.*

Definition 2.6. *For any structure $\mathscr{F}$ and any formula $\varphi \in \mathscr{L}_{PAL}(V) \cup \mathscr{L}_{DEL}(V)$ we define its* local Boolean translation *$\|\varphi\|_{\mathscr{F}}$ as follows.*

$$\|\top\|_{\mathscr{F}} := \top \qquad \|\bot\|_{\mathscr{F}} := \bot \qquad \|p\|_{\mathscr{F}} := p \qquad \|\neg\psi\|_{\mathscr{F}} := \neg\|\psi\|_{\mathscr{F}} \qquad \|\psi_1 \wedge \psi_2\|_{\mathscr{F}} := \|\psi_1\|_{\mathscr{F}} \wedge \|\psi_2\|_{\mathscr{F}}$$

If $\mathscr{F}$ is a knowledge structure, let $\|K_i\psi\|_{\mathscr{F}} := \forall(V \setminus O_i)(\theta \to \|\psi\|_{\mathscr{F}})$. If $\mathscr{F}$ is a belief structure, let $\|K_i\psi\|_{\mathscr{F}} := \forall V'(\theta' \to (\Omega_i \to (\|\psi\|_{\mathscr{F}})'))$. Let $\|[\psi]\xi\|_{\mathscr{F}} := \|\psi\|_{\mathscr{F}} \to \|\xi\|_{\mathscr{F}^\psi}$, where $\mathscr{F}^\psi := (V, \theta \wedge \|\psi\|_{\mathscr{F}}, O)$. Lastly, let $\|[\mathscr{X}, x]\varphi\|_{\mathscr{F}} := \|[x \sqsubseteq V^+]\theta^+\|_{\mathscr{F}} \to [V_-^\circ \mapsto V_-][x \sqsubseteq V^+][V_- \mapsto \theta_-(V_-)]\|\varphi\|_{\mathscr{F}\times\mathscr{X}}$ where $F \times \mathscr{X}$ is from Definition 2.3.

The above Boolean translation is equivalent, i.e. $\mathscr{F}, s \vDash \varphi$ iff $s \vDash \|\varphi\|_{\mathscr{F}}$. The SMCDEL implementation [19] uses the translation, as it is faster than using Definition 2.5. For any Kripke model there exists an equivalent symbolic structure and vice versa, and similarly for action models and transformers [17].

There are now six different model checking problems: The three languages $\mathscr{L}_{\mathsf{EL}}$, $\mathscr{L}_{\mathsf{PAL}}$ and $\mathscr{L}_{\mathsf{DEL}}$ can each be interpreted on knowledge structures (S5) and on belief structures (K). We first consider the easiest case PAL-S5 and then the most general case DEL-K. Just like general Kripke models are more general than those for S5, the Ω_i can express anything that O_i can do, i.e. every knowledge structure can be seen as a belief structure, replacing O_i, with the formula $\Omega_i := \bigwedge_{p \in O_i}(p \leftrightarrow p')$. This needs roughly twice as much memory, resulting in no significant increase of the model-checking input size.

2.1 Symbolic Model Checking PAL-S5

To define the model checking problem for symbolic structures we need to say how exactly the input is given. For V and O_i this is obvious, but for θ we stress that we assume a BDD and not a formula.

Definition 2.7. *The* symbolic model checking task for PAL for S5 *is the following. Given a knowledge structure $\mathscr{F} = (V, \theta, O_i)$ where θ is a Boolean function encoded as a BDD, an actual state s and a formula $\varphi \in \mathscr{L}_{\mathsf{PAL}}(V)$ which may contain dynamic modalities, decide whether $\mathscr{F}, s \vDash \varphi$. The input size is $|V| + |\theta| + |O_i| + |\varphi|$ where $|\theta|$ is the node count of the BDD and $|\varphi|$ is the length of the formula.*

We will show that this problem is PSPACE complete. The following theorem shows that the problem is already PSPACE-hard for EL (i.e. without announcements). This means that the symbolic representation matters, because model checking EL on standard Kripke models is in P [15], but it becomes harder when moving from Kripke models to symbolic structures.

Theorem 2.8. *Model checking $\mathscr{L}_{\mathsf{EL}}$ on knowledge structures is PSPACE-hard.*

Proof. We reduce the evaluation of a Quantified Boolean Formula (QBF) to model checking $\mathscr{L}_{\mathsf{EL}}$ on a knowledge structure as follows. Before going further into QBFs, let us introduce a notation. For a set of propositions $P = \{p_1, p_2, \ldots, p_k\}$, the notation $\forall P \equiv \forall p_1 \forall p_2 \ldots \forall p_k$. Same is true for $\exists P$.

Take any QBF (wlog. in prenex form) $\psi = \forall P_1 \exists P_2 \ldots \forall P_{n-1} \exists P_n \varphi$ where φ is a Boolean formula (possibly in CNF) over $P_1 \cup \ldots \cup P_n$. Note that there are no free variables in ψ. Let $V = \bigcup_i P_i$ and let $\theta = \top$. Let the set of agents be $\{1, \ldots, n\}$ and let $O_i := V \setminus P_i$ for each i. Let $\mathscr{F} = (V, \theta, O_i)$ and let $s = \varnothing$. Let $\widehat{K}_i\varphi := \neg K_i \neg \varphi$. We now have the following equivalences:

$$
\begin{array}{lcll}
\psi \text{ is QBF-true} & \iff & \vDash \forall P_1 \exists P_2 \ldots \forall P_{n-1} \exists P_n \varphi & \text{QBF truth definition} \\
& \iff & s \vDash \forall P_1 \exists P_2 \ldots \forall P_{n-1} \exists P_n \varphi & \text{no free variables} \\
& \iff & s \vDash \forall (V \setminus O_1) \exists (V \setminus O_2) \ldots \forall (V \setminus O_{n-1}) \exists (V \setminus O_n) \varphi & \text{by definition of } O_i \\
& \iff & \mathscr{F}, s \vDash K_1 \widehat{K}_2 \ldots K_{n-1} \widehat{K}_n \varphi & \text{by Definition 2.5}
\end{array}
$$

The last formula is of the same length as the given QBF. Hence we have a polynomial reduction from QBF-truth to model checking knowledge structures. □

In short, it is hard to do model-checking in polynomial time on symbolic structures since we need to go over every possible valuation on V while evaluating $K_i\psi$, unlike in Kripke models where these valuations are enumerated explicitly in the model and thus part of the input size.

Given that $\mathscr{L}_{\mathsf{EL}}$ is a fragment of $\mathscr{L}_{\mathsf{PAL}}$ and $\mathscr{L}_{\mathsf{DEL}}$ we have the following corollary.

Corollary 2.9. *Model checking $\mathscr{L}_{\mathsf{PAL}}$ and $\mathscr{L}_{\mathsf{DEL}}$ on knowledge structures is PSPACE-hard.*

Having shown hardness, we now turn to membership. For simplicity we focus on PAL in this section. We will define an algorithm and then show its correctness and its memory usage. Note that to stay in PSPACE we cannot use the common PAL reduction axioms (see [13, Def 4.53]) as they would blow up the size of formulas by making copies of the announced formulas. For example, consider $[!K_ip]K_ip$. The reduction axiom would give us the following equivalences:

$$[![!K_ip]K_ip]K_ip \equiv ([!K_ip]K_ip) \to K_i[![!K_ip]K_ip]p \equiv ((K_ip) \to K_i[!K_ip]p) \to K_i[!((K_ip) \to K_i[!K_ip]p)]p$$

Here the last formula which would then be model checked is huge in size with respect to the initial $[![!K_ip]K_ip]K_ip$. In general, rewriting $[![![!\ldots[![!K_ip]]K_ip]K_ip]\ldots]K_ip$ leads to a formula with 2^n copies of K_ip and thus needing exponential space in terms of the original formula length.

Our Algorithm 1 check takes as inputs a knowledge structure $\mathscr{F}$, a PAL formula φ and a list of formulas L. For the model checking problem L is initially empty and populated by the algorithm itself.

Algorithm 1: check

Input: knowledge structure $\mathscr{F} = (V, \theta, O)$, list of formulas $L = [\ell_0, \ldots, \ell_k]$, state $s \subseteq V$ of $\mathscr{F} \times \ell_0 \times \cdots \times \ell_k$, formula φ

Precondition: s is a state of $\mathscr{F}$ (i.e. $s \vDash \theta$) and L can be announced on $(\mathscr{F}, s)$.

Output: true or false

```
1  switch φ do
2      case p do return p ∈ s ;
3      case ¬φ do return NOT check(F, L, s, φ) ;
4      case φ1 ∧ φ2 do return (check(F, L, s, φ1) AND check(F, L, s, φ2)) ;
5      case K_i φ1 do
6          foreach t ⊆ V do
7              if t ⊨ θ AND t ∩ O_i = s ∩ O_i then
8                  stillExists := true // did t ''survive'' the announcements in L?
9                  foreach j ∈ [0, ..., k] do
10                     if NOT check(F, [ℓ_0, ..., ℓ_{j-1}], t, ℓ_j) then stillExists := false ;
11                 if stillExists AND NOT check(F, [ℓ_0, ... ℓ_k], t, φ1) then
12                     return false
13         return true
14     case [!ψ]φ1 do
15         if check(F, L, s, ψ) then return check(F, L++[ψ], s, φ1) else return true ;
```

The algorithm proceeds by recursion on the formula φ. There are two challenges we need to take care of so that the algorithm takes only polynomial space:

- Firstly, we cannot compute the BDDs of announcements as in Definition 2.6, because short formulas may have BDDs of exponential size. Hence we track announcements in the list L.
- When evaluating formulas of the form $K_i\varphi$, we must not compute the exponentially large set of all states. But we can iterate over them instead, without ever storing the full set.

To make both solutions compatible with each other, when evaluating $K_i\varphi$ we check in Line 9 whether a state still exists after the sequence of announcements in L, using recursive calls. This technique is similar

to the model checking algorithm for succinct DEL in [1]. It is also similar to the context-dependent semantics in [27], where the announcements are stored as a list of formulas and the model update is not triggered until $K_i\varphi$ formulas are evaluated. We now formally state that check is correct.

Lemma 2.10. *Given a pointed knowledge structure $\mathscr{F}$, a state s of $\mathscr{F}$, a list of* PAL*-S5 formulas $L = [\ell_0, \ldots, \ell_k]$ and a* PAL*-S5 formula φ, we have $\mathscr{F}^{\ell_0 \ldots \ell_k}, s \vDash \varphi$ iff* $\mathsf{check}(\mathscr{F}, L, s, \varphi)$ *returns* true*.*

Proof sketch. By induction on the size of inputs φ, $\mathscr{F}$ and L. The difficult case is for $K_i\varphi_1$. It iterates over all valuations (line 6) checking whether they agree on what i observes (line 7) and recursively verifies φ_1 as per Definition 2.5. The survival of the valuation by the sequence of announcements L is checked in line 9. For a detailed proof, see the appendix.

Next comes our main result for this subsection.

Theorem 2.11. *Model checking $\mathscr{L}_{\mathsf{PAL}}, \mathscr{L}_{\mathsf{EL}}$ on knowledge structures is PSPACE-complete.*

Proof. Hardness follows from Corollary 2.9. By Lemma 2.10 it remains to show that algorithm 1 only takes space polynomial in the size of the input. Membership for $\mathscr{L}_{\mathsf{EL}}$ follows as a special case of $\mathscr{L}_{\mathsf{PAL}}$. At any instance of a call to $\mathsf{check}(\mathscr{F}, L, s, \varphi)$, exactly one switch case matches. Every recursive call taken in $\mathsf{check}(\mathscr{F}, L, s, \varphi)$ is of size no more than the size of the current input. Moreover, in the call stack, there are at most linear many recursive calls left to evaluate at any given instance. In each call, the space used is polynomial. The most intensive is the case for $K_i\varphi_1$. We iterate over every valuation $t \subseteq V$, which needs memory in the size of $|V|$. Note that we never store the whole list of states nor results. □

We note that the model checking algorithm performing well *in practice* (and used in SMCDEL [19]) is *not* in PSPACE. To see this, take a Boolean formula for which the BDD has to be large, for example consider the vocabulary $V = \{p_1, \ldots p_{2n}\}$ and the formula $\varphi := (p_1 \wedge p_{n+1}) \vee \ldots \vee (p_n \wedge p_{n+n})$ which has length in $\mathscr{O}(n)$. Suppose then we model check $[!\varphi]p$ on a knowledge structure $\mathscr{F}$ with state law $\theta = \top$ using the SMCDEL algorithm. This means we update the state law (stored as a BDD) from $\top$ to $\theta \wedge \|\varphi\|_{\mathscr{F}}$. As discussed in [9, p. 681] the BDD of this formula needs 2^{n+1} many nodes. Hence the resulting knowledge structure needs exponential space and this is not possible in PSPACE.

In summary, there is a trade-off between time and space here: Algorithm 1 will need exponential time to evaluate K_i, whereas the BDD-based method may need exponential space for $[!\varphi]$.

2.2 Symbolic Model Checking DEL-K

We now generalize the setting from the previous section in two aspects. First, we switch from S5 to K by using belief structures instead of knowledge structures. Second, we move from public announcements to general events. We already discussed that DEL can be seen as an extension of PAL in the introduction. We now illustrate how the move from S5 to K, and the additional feature of factual change are dealt with symbolically. The difference between S5 and K is captured by $O_i \subseteq V$ and $\Omega_i \in \mathscr{L}_B(V \cup V^+)$: O_i always encodes an equivalence relation for a hard notion of knowledge, while Ω_i can encode arbitrary relations to describe belief. Factual change is captured by V_- and θ_- in the transformers from Definition 2.3, describing which propositions are changed and how [17, Section 2.8].

Theorem 2.12. *Model checking $\mathscr{L}_{\mathsf{EL}}$ on belief structures is PSPACE-hard.*

Proof. Similar to the proof of Theorem 2.8, but instead of O_i let Ω_i be the BDD of $\bigwedge_{p \in P_i}(p \leftrightarrow p')$. This BDD has size linear in $|V|$. Hence we reduce QBF truth to model checking $\mathscr{L}_{\mathsf{EL}}$ on belief structures. □

Again, given that $\mathscr{L}_{\mathsf{EL}}$ is a fragment of $\mathscr{L}_{\mathsf{PAL}}$ and $\mathscr{L}_{\mathsf{DEL}}$ we have the following corollary.

Corollary 2.13. *Model checking $\mathscr{L}_{\mathsf{PAL}}$ and $\mathscr{L}_{\mathsf{DEL}}$ on belief structures is PSPACE-hard.*

Algorithm 2: checkDELK

Input : Knowledge structure $\mathscr{F} = (V, \theta, \Omega)$, list of events $L = [(\mathscr{X}_0, x_0), \ldots, (\mathscr{X}_k, x_k)]$, state $s \subseteq V$, formula $\varphi \in \mathscr{L}_{DEL}(V)$

Precondition: s is a state of $\mathscr{F}$ (i.e. $s \vDash \theta$) and L can be executed on $(\mathscr{F}, s)$.

Output: true or false

switch φ **do**
  case p **do**
    if $L = []$ **then**
      return $p \in s$
    `// Compute the final state`
    foreach $j \in [0, \ldots, k]$ **do**
      $s := (s \setminus V_{j,-}) \cup \{q \in V_{j,-} \mid \mathsf{checkDELK}(\mathscr{F}, [(\mathscr{X}, x_0), \ldots, (\mathscr{X}, x_j)], s, [x_j^+ \sqsubseteq V_j^+]\theta_j^-(q))\}$
    return $p \in s$
  case $\neg\varphi$ **do**
    return NOT $\mathsf{checkDELK}(\mathscr{F}, L, s, \varphi)$
  case $\varphi_1 \wedge \varphi_2$ **do**
    return ($\mathsf{checkDELK}(\mathscr{F}, L, s, \varphi_1)$ AND $\mathsf{checkDELK}(\mathscr{F}, L, s, \varphi_2)$)
  case $K_i\psi$ **do**
    foreach $t \subseteq V$, $t_0^+ \subseteq V_0^+, \ldots, t_k^+ \subseteq V_k^+$ **do**
      if $t \vDash \theta$ **then**
        if $s \cup t' \vDash \Omega_i$ *AND* $x_1^+ \cup t_1^{+\prime} \vDash \Omega_{0,i}^+$ *AND* $\ldots$, $x_k^+ \cup t_k^{+\prime} \vDash \Omega_{k,i}^+$ **then**
          `accessible` := true `// did` t `"survive" the actions in` L`?`
          foreach $j \in [0, \ldots, k]$ **do**
            if *not* $\mathsf{checkDELK}(\mathscr{F}, [(\mathscr{X}_0, t_0^+), \ldots, (\mathscr{X}_{j-1}, t_{j-1}^+)], t, [t_j^+ \sqsubseteq V_j^+]\theta_j^+)$ **then**
              `accessible` := false
          if *accessible* **then**
            if *not* $\mathsf{checkDELK}(\mathscr{F}, [(\mathscr{X}_0, t_0^+), \ldots, (\mathscr{X}_k, t_k^+)], t, \psi)$ **then**
              return false
    return true
  case $[\mathscr{X}, x]\psi$ **do**
    if $\mathsf{checkDELK}(\mathscr{F}, L, s, [x \sqsubseteq V^+]\theta^+)$ `// checking the precondition`
    then
      return ($\mathsf{checkDELK}(\mathscr{F}, L{+}{+}[(\mathscr{X}, x)], s, \psi)$)
    return true

Similar to the previous section about PAL-S5, we prove membership by defining an algorithm. Also here the challenge is that we cannot compute the potentially too large BDD of the new state law after updating. Fortunately, also a similar solution works: the list L will now not just contain a list of public announcements, but a list of transformers. Then to check $\mathscr{F}, s \vDash [\mathscr{X}, x]\varphi$ we add that transformer to L.

Later, when we check $K_i\varphi$, we iterate "on the fly" over the states t that may be the result of the sequence of transformers, including the additional atoms introduced and taking into account the factual change.

Lemma 2.14. *Given a pointed knowledge structure $\mathscr{F} = \langle V, \theta, O_i \rangle, s \subseteq V$ and a* DEL *formula φ, we have $\mathscr{F}, s \vDash \varphi$ iff* checkDELK$(\mathscr{F}, [], s, \varphi)$ *returns* true.

Proof sketch. Similar to Lemma 2.10, see appendix for the full proof.

Theorem 2.15. checkDELK$(\mathscr{F}, [], s, \varphi)$ *takes at most polynomial space with respect to size of input.*

Proof. An argument analogous to that for Theorem 2.11 shows that algorithm 2 runs using space at most polynomial in size of the input. □

Together we now get our main result for DEL-K and as a result PAL-K and EL-K as well.

Theorem 2.16. *Model checking $\mathscr{L}_{\mathsf{EL}}, \mathscr{L}_{\mathsf{PAL}}$ and $\mathscr{L}_{\mathsf{DEL}}$ on belief structures is PSPACE-complete.*

Proof. Using algorithm 2, which is correct by Lemma 2.14 it is correct and uses polynomial space by Theorem 2.15, we have membership. By Corollary 2.13, we have hardness. □

Moreover, since DEL-S5 is a special case of DEL-K we have the following.

Theorem 2.17. *Model checking $\mathscr{L}_{\mathsf{DEL}}$ on knowledge structures is PSPACE-complete.*

Proof sketch. Just like in Theorem 2.12, for membership use the equivalent $\Omega_i = \bigwedge_{p \in O_i} p \leftrightarrow p'$ (for the belief structure) and $\Omega_i^+ = \bigwedge_{p^+ \in O_i^+} p^+ \leftrightarrow p^{+\prime}$ (for any transformer). The resulting belief structure is at most twice the size of the given knowledge structures. The hardness result is from Theorem 2.8.

3 Succinct Models with Mental Programs

An alternative to the symbolic structures used in the previous section is the framework of *Succinct* DEL as presented by [12]. It is based on a version of (Propositional Dynamic Logic) PDL, also called Dynamic Logic of Propositional Assignment (DLPA) [3].

Definition 3.1. *The language of* mental programs *over a vocabulary V is defined by*

$$\pi ::= p \leftarrow \top \mid p \leftarrow \bot \mid \beta? \mid \pi \cup \pi \mid \pi;\pi \mid \pi \cap \pi$$

where $p \in V$ and β is a Boolean formula over V. Let Π_V denote the set of all mental programs over V. The length of a mental program is defined as follows.

$$\begin{array}{llllll} |p \leftarrow \top| & := & 1 & |\beta?| & := & |\beta| \\ |p \leftarrow \bot| & := & 1 & |\pi_1 \cup \pi_2| & := & |\pi_1| + |\pi_2| \end{array} \qquad \begin{array}{lll} |\pi_1;\pi_2| & := & |\pi_1| + |\pi_2| \\ |\pi_1 \cap \pi_2| & := & |\pi_1| + |\pi_2| \end{array}$$

In the rest of the article we are mostly interested in how a single relation over the set of states $\mathscr{P}(V)$ can be encoded. Hence we omit further details about succinct DEL and refer to [12] for how Kripke models can be encoded using one mental program for each agent, and how actions and events can be encoded using mental programs as well.

Definition 3.2. *Two states $s,t \subseteq V$ are related by a mental program π (written as $s \xrightarrow{\pi} t$) as follows:*

$$\begin{array}{ll} s \xrightarrow{p \leftarrow \top} t & :\Longleftrightarrow\ t = s \cup \{p\} \\ s \xrightarrow{p \leftarrow \bot} t & :\Longleftrightarrow\ t = s \setminus \{p\} \\ s \xrightarrow{\beta?} t & :\Longleftrightarrow\ s = t \text{ and } s \vDash \beta \end{array} \qquad \begin{array}{ll} s \xrightarrow{\pi_1 \cup \pi_2} t & :\Longleftrightarrow\ s \xrightarrow{\pi_1} t \text{ or } s \xrightarrow{\pi_2} t \\ s \xrightarrow{\pi_1;\pi_2} t & :\Longleftrightarrow\ \exists u \subseteq V :\ s \xrightarrow{\pi_1} u \text{ and } u \xrightarrow{\pi_2} t \\ s \xrightarrow{\pi_1 \cap \pi_2} t & :\Longleftrightarrow\ s \xrightarrow{\pi_1} t \text{ and } s \xrightarrow{\pi_2} t \end{array}$$

We define $R_\pi := \{(s,t) \in V \times V \mid s \xrightarrow{\pi} t\}$.

Example 3.3. *From the state $\{p,q\}$ we can reach the state $\{p\}$ using the mental program $q \leftarrow \bot$. From the state $\{p,q\}$ we can reach the states $\{p\}$ and $\{q\}$ using the mental program $(p \leftarrow \bot) \cup (q \leftarrow \bot)$.*

Other versions of mental programs [12] also include the general assignment $p \leftarrow \beta$ and inverse π^{-1}. However, these operators do not add expressivity, as shown in [18]. Also the following shows that mental programs as defined above are complete in the sense that they can encode all relations.

Definition 3.4. *For any $x \subseteq y \subseteq V$, let $\mathsf{of}(x,y) := \bigwedge_{p\in x} p \wedge \bigwedge_{p\in y\setminus x} \neg p$. For any set $x \subseteq V$ we define $\mathsf{change}(x) := ;_{p\in x}((p \leftarrow \top) \cup (p \leftarrow \bot))$, and for any state $s \subseteq V$ we define $\mathsf{goto}(s,V) := (;_{p\in s}(p \leftarrow \top))\ ;(;_{p\in V\setminus x}(p \leftarrow \bot))$.*

We illustrate Definition 3.4 with some examples. First, $x = \{p\}$ and $y = \{p,q,r\}$ then we have $\mathsf{of}(x,y) = p \wedge \neg q \wedge \neg r$. Second, if $x = \{q,r\}$ then $\mathsf{change}(x) = (q \leftarrow \top \cup q \leftarrow \bot); (r \leftarrow \top \cup r \leftarrow \bot)$. Third, if $V = \{p,q,r\}$ then $\mathsf{goto}(\{p\},V) = p \leftarrow \top; q \leftarrow \bot; r \leftarrow \bot$.

Lemma 3.5. *For any relation $R \subseteq V \times V$ there is a mental program π such that $R_\pi = R$.*

Proof. Let $\pi := \bigcup_{(x,y)\in R}(\mathsf{of}(x,V)?;\mathsf{goto}(y))$ and apply Definition 3.2. □

In [12] it is shown how Kripke models can be encoded with mental programs and that model checking DEL on such *succinct* models is in PSPACE. This is shown not by giving a PSPACE algorithm, but an alternating Turing machine algorithm and then using the fact that PSPACE = APTIME [11]. This means the algorithm given cannot easily be translated to an actual implementation. The authors suggest that "a model checking procedure for our succinct language may use BDD techniques of [6]" [12, p. 130].

4 Comparison and Translations

To compare BDDs and mental programs, we present a list of examples in Table 2. Each row stands for a class of relations or an operation on relations, and shows how it can be represented or executed, in the different representations. Notably, observing more propositions, and thus going from the total relation (in row 2) to the identity (in row 5), leads to a shorter mental program, but to a longer Boolean formula.

	Mental program π	Boolean function Ω	Observable O
empty relation	$?\bot$	$\bot$	n/a
total relation	$\mathsf{change}(V)$	$\top$	$\varnothing$
only observe p	$\mathsf{change}(V \setminus \{p\})$	$p \leftrightarrow p'$	$\{p\}$
only observe $\{p,q\}$	$\mathsf{change}(V \setminus \{p,q\})$	$(p \leftrightarrow p') \wedge (q \leftrightarrow q')$	$\{p,q\}$
identity relation	$?\top$	$\bigwedge_{p\in V}(p \leftrightarrow p')$	V
single edge $s \to t$	$?\mathsf{of}(s,V);\mathsf{change}(V);\mathsf{of}(t,V)$	$\mathsf{of}(s,V) \wedge \mathsf{of}(t',V')$	n/a
complement	n/a	Given β, use $\neg\beta$	n/a
inverse	See translation in [18].	Given β, swap p with p' etc.	no change
composition	Given π_1 and π_2, use $\pi_1;\pi_2$	See Def. 4.1	n/a
intersection	Given π_1 and π_2, use $\pi_1 \cap \pi_2$	Given β_1 and β_2, use $\beta_1 \wedge \beta_2$	union

Table 2: Examples of relations and operations.

The "n/a" entries in Table 2 mean that this operation cannot be defined (in general) with the given representation. For example, given a mental program we cannot easily define one for the complement (Row 7). All mentioned operations can be done with Boolean functions, but with observational variables (last column) three operations are impossible because they yield non-equivalence relations.

4.1 Translating Mental Programs to BDDs

Given a mental program we now want to encode the same relation over subsets of V using a Boolean function. We give the following definition using Boolean formulas on the right side, but it also provides a mapping of mental programs to BDDs by reading all connectives on the right side as BDD operations. Recall that Π_V is the set of mental programs over V and $\mathscr{L}_B(V \cup V')$ is the set of Boolean formulas over the double vocabulary where the prime makes a fresh copy of each atomic proposition letter.

Definition 4.1. *We now define a function* $\mathsf{tr} : \Pi_V \to \mathscr{L}_B(V \cup V')$.

$$\begin{array}{ll} \mathsf{tr}(p \leftarrow \top) := p' \wedge \bigwedge_{p \neq q \in V}(q \leftrightarrow q') & \mathsf{tr}(\pi_1 \cup \pi_2) := \mathsf{tr}(\pi_1) \vee \mathsf{tr}(\pi_2) \\ \mathsf{tr}(p \leftarrow \bot) := \neg p' \wedge \bigwedge_{p \neq q \in V}(q \leftrightarrow q') & \mathsf{tr}(\pi_1 ; \pi_2) := [V'' \mapsto V'](\exists V'(\mathsf{tr}(\pi_1) \wedge \mathsf{tr}(\pi_2)')) \\ \mathsf{tr}(\beta ?) := \beta \wedge \bigwedge_{p \in V}(p \leftrightarrow p') & \mathsf{tr}(\pi_1 \cap \pi_2) := \mathsf{tr}(\pi_1) \wedge \mathsf{tr}(\pi_2) \end{array}$$

The left cases of Definition 4.1 are easy, they mostly consist of the "do not change anything else" conjunctions. Among the right three cases, the one standing out is the composition $\pi_1 ; \pi_2$. Here we write $[\cdot \mapsto \cdot]\varphi$ for simultaneous substitution of atoms in φ. For example, $[\{p', q'\} \mapsto \{p, q\}](p' \wedge q'') = (p \wedge q'')$. Strictly speaking in $[A \mapsto B]\varphi$ both A and B are ordered lists and we use the implicit bijection between them as the substitution function [17, Def. 1.0.3]. We note that $\mathsf{tr}(\pi_2)'$ is $[V \mapsto V'][V' \mapsto V'']\mathsf{tr}(\pi_2)$, i.e. this changes the formula from the vocabulary $V \cup V'$ to the vocabulary $V' \cup V''$. The Boolean quantification $\exists V'$ then eliminates all single-primed variables and lastly the outermost substitution ensures that the resulting formula is over the vocabulary $V \cup V'$ as we want. We stress that $\exists$ is only expensive when done syntactically. As a BDD operation it in fact deletes variables and nodes from a BDD [9].

Theorem 4.2. *The translation from Def. 4.1 is correct: for any* $s, t \subseteq V$, *we have* $s \xrightarrow{\pi} t$ *iff* $(s \cup t') \vDash \mathsf{tr}(\pi)$.

Proof sketch. By induction on the structure of π and applying Definition 3.2. See appendix for proof.

Note that already translating $p \leftarrow \top$ needs $\mathscr{O}(|V|)$ many nodes in the BDD. This illustrates the high length of formulas resulting from this translation. However, it is not a problem specific to the particular translation given above, but applies to any correct translation, as the following theorem states.

Theorem 4.3. *For any translation* tr' *from mental programs to BDDs there exists a mental program* π *such that* $\mathsf{tr}'(\pi)$ *has size exponential in* $|\pi|$.

Proof sketch. Take any vocabulary $V = \{p_1, \ldots p_{2n}\}$ with this ordering fixed. Let $\pi := \beta ?$ where β is the Boolean formula $(p_1 \wedge p_{n+1}) \vee \ldots \vee (p_n \wedge p_{n+n})$ from page 7. We show that the BDD corresponding to the mental program π has at least 2^{n+1} many nodes. For the details, see the appendix.

4.2 Translating BDDs to Mental Programs

We write $(t_0 \dashleftarrow (p_i) \rightarrow t_1)$ to denote a node in a BDD that is labelled with variable p and has an else-edge (dotted) pointing to node t_0 and a then-edge (solid) pointing to node t_1. Note that p_i may come from V or from V', and in the latter case we write p'_i. For the leaves, we just write $\top$ and $\bot$.

Definition 4.4. *We define a function* $\tau : BDD_{(V \cup V')} \times [V \cup V'] \to \Pi_V$ *where* $[V \cup V']$ *is the set of all lists with elements from* $V \cup V'$. *We distinguish different cases for the given BDD* Ω.

$$\tau(\bot, L) := ?\bot \qquad \tau(\top, []) := ?\top \qquad \tau(\top, [p_k, .., p_n]) := ((p_k \leftarrow \bot) \cup (p_k \leftarrow \top)) ; \tau(\top, [p_{k+1}, .., p_n])$$

$$\tau(\Omega = (t_0 \dashleftarrow (p_i) \rightarrow t_1), [p_k, .., p_n]) := \begin{cases} (?\neg p_k ; \tau(t_0, [p_k, .., p_n])) \cup (?p_k ; \tau(t_1, [p_k, .., p_n])) & \text{if } i = k \\ ((p_k \leftarrow \bot) \cup (p_k \leftarrow \top)) ; \tau(\Omega, [p_{k+1}, .., p_n]) & \text{otherwise} \end{cases}$$

$$\tau(\Omega = (t_0 \dashleftarrow (p'_i) \rightarrow t_1), [p_k, .., p_n]) := \begin{cases} (p_k \leftarrow \bot ; \tau(t_0, [p_{k+1}, .., p_n])) \\ \cup (p_k \leftarrow \top ; \tau(t_1, [p_{k+1}, .., p_n])) & \text{if } i = k \\ ((p_k \leftarrow \bot) \cup (p_k \leftarrow \top)) ; \tau(\Omega, [p_{k+1}, .., p_n]) & \text{otherwise} \end{cases}$$

Lastly, define $\tau_0 : BDD_{V \cup V'} \to \Pi_V$ *by* $\tau_0(\Omega) := \tau(\Omega, [p_0, \ldots, p_n])$.

Both "otherwise" cases above are about variables $i > k$ not mentioned by BDD. If the BDD does not mention a unprimed variable, this simply means the relation does not depend on that variable in the starting state, so the mental program also does not have to mention it. On the other hand, not mentioning a primed variable in the BDD means we must allow the mental program to change that variable arbitrarily.

Theorem 4.5. *The translation τ always terminates.*

Proof. We observe that at each recursive step either the size of the BDD Ω or the length of the list L strictly decreases, while the other size stays the same. □

Theorem 4.6. *The translation τ_o from Definition 4.4 is correct. That is, given a vocabulary V, for any $s,t \subseteq V$, we have $s \xrightarrow{\tau_0(\Omega)} t$ iff $(s \cup t') \vDash \Omega$.*

Proof sketch. For any BDD Ω, let Ω^* be its unraveling to a tree. Note that we have $\tau(\Omega) = \tau(\Omega^*)$. Hence for the proof we assume wlog. that β is a tree and proceed by induction over the tree structure. For all details, see the appendix.

We now consider the length of the mental programs resulting from the translation. Unfortunately, given a BDD for a vocabulary V with $|V| = n$ the output of τ will always have at least length $\mathcal{O}(2^n)$. The result may be simplified using equivalences such as $p \leftarrow \top; p? \equiv p \leftarrow \top$ and $p \leftarrow \bot; p? \equiv \bot?$. But even including such simplifications we believe that there is no better translation in the following sense.

Conjecture 4.7. *For any translation τ' from BDDs to mental programs that is correct in the sense of Theorem 4.6, there exists a BDD β such that $\tau'(\beta)$ has a length exponential in the number of nodes of β.*

As an example, take the BDD encoding the relation $R = \{(s,s) \mid |s| = \lceil \frac{|V|}{2} \rceil\}$, i.e. the identity restricted to the states where exactly half of the vocabulary is true. The BDD encoding R is a $|V| \cdot |V|$ grid. We can find a mental program encoding this relation using (the proof of) Lemma 3.5, but the size of the mental program is $2^{|V|/2} \cdot |V|$. It is not clear whether there exists a shorter mental program that encodes R. Essentially the mental program needs to count, and at the same time preserve the valuation. But as there are no additional variables the only way to count without forgetting is to try all valuations. Thus we believe that any mental program π encoding R must have size exponential in the size of the *BDD*.

5 Conclusions and Future Work

In this paper, we showed that the symbolic model-checking tasks for EL, PAL and DEL on both knowledge structures and belief structures are all PSPACE-complete. In addition we compared how the same relations can be encoded in belief structures with BDDs on one hand, and in succinct models with mental programs (a regular language like syntax) on the other hand. That is, we provided translations from mental programs to BDDs, and back. We have also proven that any such translation from mental programs to BDDs will lead to an exponential blowup in size, and we conjecture the same for the other direction from BDDs to mental programs.

An aspect we did not consider here but that is relevant for epistemic planning are multi-pointed models and actions. Concerning implementations, a PAL model checker using mental programs was implemented in [21]. In parallel to the theoretical work presented here we have improved the performance of this code and implemented the translations from section 4. We plan to benchmark all methods and eventually merge them into SMCDEL in the future.

References

[1] Guillaume Aucher & François Schwarzentruber (2013): *On the Complexity of Dynamic Epistemic Logic*. In Burkhard C. Schipper, editor: *Proceedings of the 14th Conference on Theoretical Aspects of Rationality and Knowledge (TARK 2013), Chennai, India, January 7-9, 2013*, pp. 19–28. Available at `http://www.tark.org/proceedings/tark_jan7_13/p19-aucher.pdf`.

[2] Philippe Balbiani, Andreas Herzig, François Schwarzentruber & Nicolas Troquard (2014): *DL-PA and DCL-PC: model checking and satisfiability problem are indeed in PSPACE*. *CoRR* abs/1411.7825. arXiv:1411.7825.

[3] Philippe Balbiani, Andreas Herzig & Nicolas Troquard (2013): *Dynamic Logic of Propositional Assignments: A Well-Behaved Variant of PDL*. In: *28th Annual ACM/IEEE Symposium on Logic in Computer Science*, pp. 143–152, doi:10.1109/LICS.2013.20.

[4] Alexandru Baltag, Lawrence S. Moss & Slawomir Solecki (1998): *The logic of public announcements, common knowledge, and private suspicions*. In Itzhak Gilboa, editor: *Theoretical Aspects of Rationality and Knowledge (TARK 1998)*, TARK 1998, pp. 43–56. Available at `https://dl.acm.org/citation.cfm?id=645876.671885`.

[5] Vaishak Belle, Thomas Bolander, Andreas Herzig & Bernhard Nebel (2023): *Epistemic planning: Perspectives on the special issue*. *Artif. Intell.* 316, p. 103842, doi:10.1016/J.ARTINT.2022.103842.

[6] Johan van Benthem, Jan van Eijck, Malvin Gattinger & Kaile Su (2015): *Symbolic Model Checking for Dynamic Epistemic Logic*. In Wiebe van der Hoek, Wesley H. Holliday & Wen-fang Wang, editors: *Proceedings of the 5th International Workshop on Logic, Rationality, and Interaction (LORI 2015)*, pp. 366–378, doi:10.1007/978-3-662-48561-3_30.

[7] Johan van Benthem, Jan van Eijck, Malvin Gattinger & Kaile Su (2018): *Symbolic Model Checking for Dynamic Epistemic Logic – S5 and Beyond*. *Journal of Logic and Computation (JLC)* 28(2), pp. 367–402, doi:10.1093/logcom/exx038.

[8] Patrick Blackburn, Maarten de Rijke & Yde Venema (2001): *Modal Logic*. *Cambridge Tracts in Theoretical Computer Science* 53, Cambridge University Press, doi:10.1017/CBO9781107050884.

[9] Randal E. Bryant (1986): *Graph-Based Algorithms for Boolean Function Manipulation*. *IEEE Transactions on Computers* C-35(8), pp. 677–691, doi:10.1109/TC.1986.1676819.

[10] Sagar Chaki & Arie Gurfinkel (2018): *BDD-Based Symbolic Model Checking*. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith & Roderick Bloem, editors: *Handbook of Model Checking*, Springer Nature, pp. 219–245, doi:10.1007/978-3-319-10575-8_8.

[11] Ashok K. Chandra, Dexter C. Kozen & Larry J. Stockmeyer (1981): *Alternation*. *Journal of the ACM* 28(1), pp. 114—-133, doi:10.1145/322234.322243.

[12] Tristan Charrier & François Schwarzentruber (2017): *A Succinct Language for Dynamic Epistemic Logic*. In S. Das, E. Durfee, K. Larson & M. Winikoff, editors: *Proceedings of the 16th Conference on Autonomous Agents and Multiagent Systems*, AAMAS '17, International Foundation for Autonomous Agents and Multiagent Systems, pp. 123–131. Available at `https://www.ifaamas.org/Proceedings/aamas2017/pdfs/p123.pdf`.

[13] Hans van Ditmarsch, Wiebe van der Hoek & Barteld Kooi (2007): *Dynamic Epistemic Logic*. Springer, doi:10.1007/978-1-4020-5839-4.

[14] Hans van Ditmarsch, Wiebe van der Hoek, Ron van der Meyden & Ji Ruan (2006): *Model Checking Russian Cards*. *Electronic Notes in Theoretical Computer Science* 149(2), pp. 105–123, doi:10.1016/j.entcs.2005.07.029.

[15] Ronald Fagin, Joseph Y. Halpern, Yoram Moses & Moshe Y. Vardi (1995): *Reasoning About Knowledge*. MIT Press, doi:10.7551/MITPRESS/5803.001.0001.

[16] Michael J. Fischer & Richard E. Ladner (1979): *Propositional dynamic logic of regular programs*. *Journal of Computer and System Sciences* 18(2), pp. 194–211, doi:10.1016/0022-0000(79)90046-1.

[17] Malvin Gattinger (2018): *New Directions in Model Checking Dynamic Epistemic Logic*. Ph.D. thesis, University of Amsterdam. Available at `https://malv.in/phdthesis`.

[18] Malvin Gattinger (2020): *Towards Symbolic and Succinct Perspective Shifts*, doi:10.5281/zenodo.4767546. Presentation at the EpiP workshop at ICAPS 2020.

[19] Malvin Gattinger, Stefano Volpe, Bas Laarakker & Daniel Miedema (2024): *SMCDEL — A symbolic model checker for Dynamic Epistemic Logic*, doi:10.5281/zenodo.11061438. Available at `https://github.com/jrclogic/SMCDEL`. Version 1.3.0.

[20] Ronald de Haan & Iris van de Pol (2021): *On the Computational Complexity of Model Checking for Dynamic Epistemic Logic with S5 Models*. Journal of Applied Logics 8(3), pp. 621–658. Available at `http://www.collegepublications.co.uk/downloads/ifcolog00045.pdf#page=10`.

[21] Maickel Hartlief (2020): *Making Model Checking Scalable: Implementing Succinct Kripke Models for Public Announcement Logic*. Available at `https://fse.studenttheses.ub.rug.nl/23607/`. BSc thesis, Groningen. Code avaliable at `https://github.com/Maickel-Hartlief/SucExpModelCheckers`.

[22] Esteban Landerreche & David Fernández-Duque (2017): *A case study in almost-perfect security for unconditionally secure communication*. Designs, Codes and Cryptography 83(1), pp. 145–168, doi:10.1007/s10623-016-0210-y.

[23] Carsten Lutz (2006): *Complexity and succinctness of public announcement logic*. In Hideyuki Nakashima, Michael P. Wellman, Gerhard Weiss & Peter Stone, editors: *5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), Hakodate, Japan, May 8-12, 2006*, ACM, pp. 137–143, doi:10.1145/1160633.1160657.

[24] Daniel Miedema & Malvin Gattinger (2023): *Exploiting Asymmetry in Logic Puzzles: Using ZDDs for Symbolic Model Checking Dynamic Epistemic Logic*. In Rineke Verbrugge, editor: *Theoretical Aspects of Rationality and Knowledge 2023 (TARK 2023)*, pp. 407—-420, doi:10.4204/EPTCS.379.32.

[25] Jan Plaza (2007): *Logics of public communications*. Synthese 158(2), pp. 165–179, doi:10.1007/S11229-007-9168-7.

[26] Weizhi Tang & Vaishak Belle (2024): *ToM-LM: Delegating Theory of Mind Reasoning to External Symbolic Executors in Large Language Models*. In Tarek R. Besold, Artur d'Avila Garcez, Ernesto Jimenez-Ruiz, Roberto Confalonieri, Pranava Madhyastha & Benedikt Wagner, editors: *Neural-Symbolic Learning and Reasoning*, Springer Nature Switzerland, pp. 245–257, doi:10.1007/978-3-031-71170-1_20.

[27] Yanjing Wang & Qinxiang Cao (2013): *On axiomatizations of public announcement logic*. Synthese 190, pp. 103–134, doi:10.1007/s11229-012-0233-5.

A Proofs

Lemma 2.10. *Given a pointed knowledge structure $\mathscr{F}$, a state s of $\mathscr{F}$, a list of* PAL-*S5 formulas $L = [\ell_0, \ldots, \ell_k]$ and a* PAL-*S5 formula φ, we have $\mathscr{F}^{\ell_0 \ldots \ell_k}, s \vDash \varphi$ iff* $\mathsf{check}(\mathscr{F}, L, s, \varphi)$ *returns* true.

Proof. By induction on the input size and case distinction on φ. We only show the interesting cases for knowledge and for announcements.

Case $\varphi = K_i\varphi_1$. We have:

$$\begin{aligned}\mathscr{F}^{\ell_0 \ldots \ell_k}, s \vDash K_i\varphi_1 &\text{ iff } \forall t \subseteq V : s \cap O_i = t \cap O_i \text{ and } t \text{ survives in } \mathscr{F}^{\ell_0 \ldots \ell_k} \text{ implies } \mathscr{F}^{\ell_0 \ldots \ell_k}, t \vDash \varphi_1 \\ &\text{iff } \forall t \subseteq V : s \cap O_i = t \cap O_i \text{ and } \bigwedge_{i=0}^{k} \mathscr{F}^{\ell_0 \ldots \ell_{i-1}}, t \vDash \ell_i \text{ implies } \mathscr{F}^{\ell_0 \ldots \ell_k}, t \vDash \varphi_1\end{aligned}$$

Note that, assuming for a $t \subseteq V$, $s \cap O_i = t \cap O_i$, $\bigwedge_{i=0}^{k} \mathscr{F}^{\ell_0 \ldots \ell_{i-1}}, t \vDash \ell_i$ if and only if `stillExists`$=$ true after the execution of the loop in line 9. This is so by using IH as for any $i \in [1, |L| - 1]$, $|\mathscr{F}| + |\ell_0| + \ldots + \ell_{i-1} + \ell_i < |\mathscr{F}| + \sum_{j=0}^{k} |\ell_j| + |\varphi_1| + 1$. Line 7 checks whether $s \cap O_i = t \cap O_i$. Hence

$$\begin{aligned}&\mathscr{F}^{\ell_0 \ldots \ell_k}, s \vDash K_i\varphi_1 \\ &\text{iff } \forall t \subseteq V : \text{line 7 and } \texttt{stillExists} \text{ is true after loop 9 implies } \mathscr{F}^{\ell_0 \ldots \ell_k}, t \vDash \varphi_1 \\ &\text{iff } \forall t \subseteq V : \text{line 7 and } \texttt{stillExists} \text{ is true after loop 9 implies } \mathsf{check}(\mathscr{F}[\ell_0 \ldots \ell_k], t, \varphi_1) \neq \mathsf{false}\end{aligned}$$

Case $\varphi = [!\psi]\varphi_1$. We have:

$$\begin{aligned}\mathscr{F}^{\ell_0 \ldots \ell_k}, s \vDash [!\psi]\varphi_1 &\text{ iff } \mathscr{F}^{\ell_0 \ldots \ell_k}, s \vDash \psi \text{ implies } \mathscr{F}^{\ell_0 \ldots \ell_k \psi}, s \vDash \varphi_1 \\ &\text{iff } \mathsf{check}(\mathscr{F}, L, s, \psi) = \mathsf{true} \text{ implies } \mathsf{check}(\mathscr{F}, L{+}{+}[\psi], s, \varphi_1) = \mathsf{true}, \text{by IH}\end{aligned}$$

Note that we make two recursive calls here. We can apply the IH to both because the input size strictly decreases: $|\mathscr{F}| + \sum_{i=0}^{k} |\ell_i| + |\psi| < |\mathscr{F}| + \sum_{i=0}^{k} |\ell_i| + |\psi| + |\varphi_1| + 1$ and $|\mathscr{F}| + \sum_{i=0}^{k} |\ell_i| + |\psi| + |\varphi_1| < |\mathscr{F}| + \sum_{i=0}^{k} |\ell_i| + |\psi| + |\varphi_1| + 1$.

The $\mathsf{check}(\mathscr{F}, L, s, \psi)$ is evaluated in line 15. If it is false, then $\mathsf{check}(\mathscr{F}, [\ell_0, \ldots, \ell_k], s, [!\psi]\varphi_1) = \mathsf{true}$ by line 15. Hence, $\mathscr{F}^{\ell_0 \ldots \ell_k}, s \vDash [!\psi]\varphi_1$ iff $\mathsf{check}(\mathscr{F}, [\ell_0, \ldots, \ell_1], s, [!\psi]\varphi_1) = \mathsf{true}$. □

Lemma 2.14. *Given a pointed knowledge structure $\mathscr{F} = \langle V, \theta, O_i \rangle, s \subseteq V$ and a* DEL *formula φ, we have $\mathscr{F}, s \vDash \varphi$ iff* $\mathsf{checkDELK}(\mathscr{F}, [], s, \varphi)$ *returns* true.

Proof. We show the more general claim that for any list $L = [(\mathscr{X}_0, x_0), \ldots, (\mathscr{X}_k, x_k)]$ of events we have $(\mathscr{F}, s) \otimes (\mathscr{X}_0, x_0) \otimes \ldots \otimes (\mathscr{X}_k, x_k) \vDash \varphi$ iff $\mathsf{checkDELK}(\mathscr{F}, L, s, \varphi)$ returns true. The proof is by induction on φ, and we omit the easy cases for negation and conjunction.

Case $\varphi = p$. We have the following equivalences:

$$\begin{aligned}&\mathscr{F}^{\ell_0 \ldots \ell_k}, s \vDash p \\ &\text{iff } p \in s^k = (s^{k-1} \setminus V_{k,-}) \cup (s^{k-1} \cap V_{k,-})^{\circ} \cup x^k \cup \{p \in V_{k,-} \mid s^{k-1} \cup x_k \vDash \theta_{k,-}(p)\}, \text{by Definition 2.3} \\ &\text{iff } p \in s^{k,\prime} = (s^{k-1} \setminus V_{k,-}) \cup \{q \in V_{k,-} \mid \mathsf{checkDELK}(\mathscr{F}, [], s, [x_j \sqsubseteq V_k^{+}]\theta_{k,-}(q))\}, \text{because } p \in V \\ &\text{iff } \mathsf{checkDELK}(\mathscr{F}, L, s, p) = \mathsf{true}, \text{by definition of checkDELK}\end{aligned}$$

Note the difference between s^k and $s^{k,'}$ here. $s^{k,'}$ removes the copy variables and the event state (of which variables are from the new event vocabulary) while the last part both represent the modified propositions whose precondition was true in the previous state. However, by assumption $p \in V$, therefore we can simply remove these two parts.

Case $\varphi = K_i\psi$. We have the following equivalences:

$\mathscr{F}^{\ell_0\dots\ell_k}, s \vDash K_i\psi$

iff for all t in the final model $s^{final} \cup t' \vDash \Omega_i^{final}$ implies $\mathscr{F}^{final}, t \vDash \psi$

iff $\forall t \subseteq V$ s.t. $t \vDash \theta, t_0^+ \subseteq V_0^+, \dots, t_k^+ \subseteq V_k^+$:

$s \cup t' \vDash \Omega_i, x_1^+ \cup t_1^+ \vDash \Omega_{0,i}^+, .., x_k^+ \cup t_k^+ \vDash \Omega_{k_i}$ and t survives implies $\mathscr{F}^{\ell_0\dots\ell_k}, t \vDash \psi$

iff $\forall t \subseteq V$ s.t. $t \vDash \theta, t_0^+ \subseteq V_0^+, \dots, t_k^+ \subseteq V_k^+$:

$$s \cup t' \vDash \Omega_i, x_1^+ \cup t_1^+ \vDash \Omega_{0,i}^+, .., x_k^+ \cup t_k^+ \vDash \Omega_{k_i} \text{ and } \bigwedge_{i=0}^{k} \mathscr{F}^{\ell_0\dots\ell_{i-1}}, t \vDash [t_i^+ \sqsubseteq V_i^+]\theta_i^+ \text{ implies } \mathscr{F}^{\ell_0\dots\ell_k}, t \vDash \psi$$

iff $\mathsf{checkDELK}(\mathscr{F}, L, s, K_i\psi) = \mathsf{true}$

Case $\varphi = [\mathscr{X}, x]\psi$. We have the following equivalences:

$\mathscr{F}^{\ell_0\dots\ell_k}, s \vDash [\mathscr{X}, x]\psi$

iff if $\mathscr{F}^{\ell_0\dots\ell_k}, s \vDash [x \sqsubseteq V^+]\theta^+$ then $(\mathscr{F}^{\ell_0\dots\ell_k} \times \mathscr{X}, s^x) \vDash \psi$

iff if $\mathsf{checkDELK}(\mathscr{F}, L, s, [x \sqsubseteq V^+]\theta^+) = \mathsf{true}$ then $\mathsf{check}(\mathscr{F}, L{+}{+}[(\mathscr{X}, x)], s, \psi) = \mathsf{true}$, by IH $\square$

Theorem 4.2. *The translation from Def. 4.1 is correct: for any $s, t \subseteq V$, we have $s \xrightarrow{\pi} t$ iff $(s \cup t') \vDash \mathsf{tr}(\pi)$.*

Proof. By induction on the structure of π and applying Definition 3.2.

- $s \xrightarrow{p \leftarrow \top} t \iff s \cup t' \vDash p' \wedge \bigwedge_{p \neq q \in V}(q \leftrightarrow q')$, since $p \in t$ and other variables remain the same.
- $s \xrightarrow{p \leftarrow \bot} t \iff s \cup t' \vDash \neg p' \wedge \bigwedge_{p \neq q \in V}(q \leftrightarrow q')$, since $p \notin t$ and other variables remain the same.
- $s \xrightarrow{\beta?} t \iff s = t \wedge s \vDash \beta \iff s \cup t' \vDash \beta \wedge \bigwedge_{p \in V}(p \leftrightarrow p')$, since β must be true in $s = t$.
- $s \xrightarrow{p \cup q} t \iff s \xrightarrow{p} t$ or $s \xrightarrow{q} t \iff s \cup t' \vDash tr(p)$ or $s \cup t' \vDash tr(q) \iff s \cup t' \vDash tr(p) \vee tr(q)$.
- $s \xrightarrow{\pi_1;\pi_2} t$
$$\begin{aligned}
&\iff \exists u \subseteq V : s \xrightarrow{\pi_1} u \text{ and } u \xrightarrow{\pi_2} t \\
&\iff \exists u \subseteq V : s \cup u' \vDash \mathsf{tr}(\pi_1) \text{ and } u \cup t' \vDash \mathsf{tr}(\pi_2) \\
&\iff \exists u \subseteq V : s \cup u' \vDash \mathsf{tr}(\pi_1) \text{ and } u' \cup t'' \vDash [V \mapsto V', V' \mapsto V'']\mathsf{tr}(\pi_2) \\
&\iff \exists u \subseteq V : s \cup u' \cup t'' \vDash \mathsf{tr}(\pi_1) \wedge [V \mapsto V', V' \mapsto V'']\mathsf{tr}(\pi_2) \\
&\iff s \cup t'' \vDash \exists V'(\mathsf{tr}(\pi_1) \wedge [V \mapsto V', V' \mapsto V'']\mathsf{tr}(\pi_2)) \\
&\iff s \cup t' \vDash [V'' \mapsto V'](\exists V'(\mathsf{tr}(\pi_1) \wedge [V \mapsto V'][V' \mapsto V'']\mathsf{tr}(\pi_2)))
\end{aligned}$$
- $s \xrightarrow{p \cap q} t \iff s \xrightarrow{p} t$ and $s \xrightarrow{q} t \iff s \cup t' \vDash tr(p)$ and $s \cup t' \vDash tr(q) \iff s \cup t' \vDash tr(p) \wedge tr(q)$ $\square$

Theorem 4.3. *For any translation tr' from mental programs to BDDs there exists a mental program π such that $\mathsf{tr}'(\pi)$ has size exponential in $|\pi|$.*

Proof. Take a vocabulary $V = \{p_1, \ldots p_{2n}\}$ with this ordering fixed. Consider again the formula $\beta := (p_1 \wedge p_{n+1}) \vee \ldots \vee (p_n \wedge p_{n+n})$ from page 7. Let $\pi := \beta?$. Then π also has length in $\mathcal{O}(n)$. From Definition 16, $\mathsf{tr}(\pi) = \beta \wedge \bigwedge_{p \in V}(p \leftrightarrow p')$. We show that the BDDs corresponding to this boolean formula has at least 2^{n+1} many nodes. First, as discussed in [9, p. 681], the BDD that corresponds to the β formula has at least 2^{n+1} many nodes.

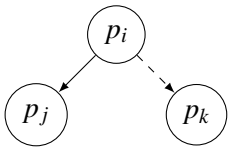


Figure 1: Node p_i in β's BDD.

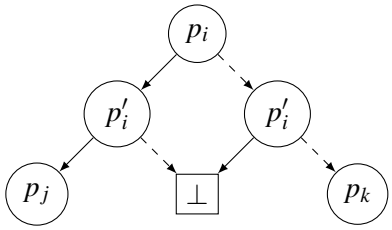


Figure 2: Node p_i in $\beta \wedge \bigwedge_{p \in V}(p \leftrightarrow p')$'s BDD.

Now we examine the size of the BDDs that corresponds to $\beta \wedge \bigwedge_{p \in V}(p \leftrightarrow p')$. Since none of the copy variables p'_i appear in β, it will not cause any subtrees to be merged. Instead, as shown in Figure 2, for each node p_i that appears in $\beta's$ BDD in Figure 1, two more nodes of p'_i are added with the solid and dashed edges encoding the $(p_i \leftrightarrow p'_i)$ sub-formula in π. Therefore, the BDD for π has at least 2^{n+1} many nodes. □

Theorem 4.6. *The translation τ_o from Definition 4.4 is correct. That is, given a vocabulary V, for any $s,t \subseteq V$, we have $s \xrightarrow{\tau_0(\Omega)} t$ iff $(s \cup t') \vDash \Omega$.*

Proof. For any BDD Ω, let Ω^* be its unraveling to a tree. Note that we have $\tau(\Omega) = \tau(\Omega^*)$. Hence for the proof we assume wlog. that β is a tree.

We proceed by induction over the structure of BDDs. In the case when the BDD is a single node $\bot$, the mental program is $?\bot$, which represents the empty relations. When the BDD is a single node $\top$, the mental program is $((p_0 \leftarrow \bot) \cup (p_0 \leftarrow \top)); \ldots; ((p_n \leftarrow \bot) \cup (p_n \leftarrow \top)); ?\top$, which represents the complete relations. The translation is correct for both of these base cases.

The trickier case is when the BDD has at least three nodes (including $\top$ and $\bot$). We proceed by an inner induction on the length of the vocabulary. The base case is when the vocabulary is empty and the only BDDs that can be constructed are just $\bot$ and $\top$nodes. As shown above, the translation is correct for both cases.

For the induction step we are adding one more variable p_0 at the beginning of the vocabulary V. We have the inductive hypothesis that for all t_i that can be constructed with the vocabulary $V = [p_1, .., p_k]$, we have $s \xrightarrow{\tau(t_0,V)} t \iff s \cup t' \vDash t_0$.

We prove that the translation remains correct for the extended vocabulary $V_1 := \{p_0\} \cup V$ by showing that both t_0 under V and $\tau(t_0, V)$ encode the same relations.

Given the new vocabulary, the unraveled BDDs that can be constructed all have the shape as shown in Figure 3, where the t_is are subtrees which can be seen as the complete trees constructed in the original vocabulary V. We can construct the corresponding relations in Figure 4. Specifically, after extending the vocabulary, we make a copy of the states in the original vocabulary V and add p_0 to each of these copies to indicate that p_0 is true in these states. The states in which p_0 is false are put on the bottom level in the

figure while the states in which p_0 is true are on the top level. We construct the corresponding relations as follows:

1. Construct the relations corresponding to t_1 in the original vocabulary V on the bottom level.
2. Construct the relations corresponding to t_2 in V on the bottom level and then shift the source states of the relations from the bottom level to the corresponding states (i.e. those that differ only by p_0.) on the top level. In Figure 4 these relations are depicted as the dotted top-to-bottom arrows.
3. Construct the relations corresponding to t_3 in V on the bottom level and then shift the target states of the relations from the bottom level to the corresponding states on the top level. In Figure 4 these relations are depicted as the dotted bottom-to-top arrows.
4. Construct the relations corresponding to t_4 in V on the bottom level and then shift both the source and target states of the relations from the bottom level to the corresponding states on the top level.

Each branch t_i in Figure 3 corresponds to part of the relation in Figure 4.

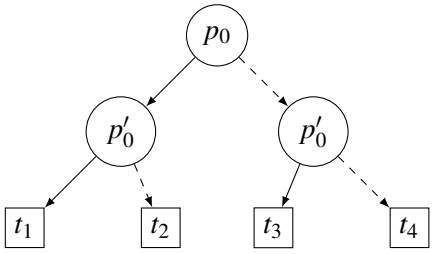


Figure 3: The BDD after p_0 is added.

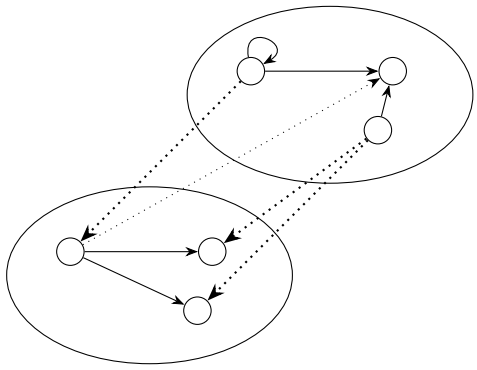


Figure 4: The corresponding Kripke model.

From inductive hypothesis we know that the translations are correct in the original vocabulary, which means that π_i and t_i encode the same relations in the original vocabulary V. Assume $\tau(t_i,V) = \pi_i$, then $\tau(t_0,V_1) = ?\neg p_0;((p_0 \leftarrow \bot;\pi_1)\cup(p_0 \leftarrow \top;\pi_2))\cup ?p_0;((p_0 \leftarrow \bot;\pi_3)\cup(p_0 \leftarrow \top;\pi_4)) = (?\neg p_0;\pi_1)\cup(?\neg p_0;(p_0 \leftarrow \top);\pi_2)\cup(?p_0;(p_0 \leftarrow \bot);\pi_3)\cup(?p_0;\pi_4)$. Similar to the reasoning in the previous part, we can see that this represents the same relation in Figure 4, e.g. the relations corresponding to π_2 go from bottom to top. Therefore, the BDD t_0 and the mental program $\tau(t_0,V_1)$ encode the same relation in the extended vocabulary $\{p_0\}\cup V$. This concludes the proof that the translation is correct. □