

AxiMo: automated axiomatic reasoning for information update

Simon Richards^{1,2}

School of Electronics and Computer Science, University of Southampton

Mehrnoosh Sadrzadeh³

Laboratoire PPS, Université Denis Diderot- Paris 7, and Oxford University Computing Laboratory

Abstract

We present an algorithm for proving epistemic properties of dynamic scenarios in multi-agent systems and an implementation of it as the C^{++} program `AxiMo`. The program consists of a rewrite system and a recursive reasoner, and we prove that it is sound with regard to the algebraic axiomatics of dynamic epistemic logic. We study the termination and complexity of the program and show its applicability by proving properties of honest and dishonest versions of a coin toss scenario and the muddy children puzzle.

Keywords: Automation, Rewrite system, Algebraic axiomatics, Information update, Dynamic epistemic logic.

1 Introduction

One of the applications of modal logic is reasoning about information of agents in multi-agent systems in the context of epistemic logics [7]. This field of application has been extended to *update* of information of agents in the context of dynamic epistemic logics [1,3,5]. In these applications, one reasons about information of *interacting* agents who communicate with each other and get their information updated as a result. Dynamic epistemic logic (DEL) reasons about information of these agents, the communication actions between them, and the changes induced to the information by the actions. One of the novelties of DEL is its ability to reason about honest as well as dishonest agents and their actions. It does so by using a modality that stands for a *possibly wrong belief*, this is the belief that is caused by the cheating and lying actions of dishonest agents.

Kripke structures provide relational semantics for modal logics. One advantage of these models, other than being intuitive, is that their frame conditions directly give rise to axioms

¹ Support from EPSRC grant EP/D000033/1 at Southampton University is acknowledged by both authors.

² Email: sgr104@ecs.soton.ac.uk

³ Email: mehrs@comlab.ox.ac.uk

of a Hilbert-style proof system, e.g. a transitive frame gives rise to axiom 4: the so called *positive introspection* axiom of knowledge. Although semantics of DEL is based on Kripke structures, not all of its modal axioms are obtained from its frame conditions. This is because the semantics of DEL involves higher level operations that act on the Kripke structures themselves. But if one thinks in the spirit of Stone duality and moves to an algebraic semantics, the axioms corresponding to these operations can be treated on the same level as the axioms corresponding to the modalities, that is as operations on the base algebra. As a result, one directly obtains an axiomatics to reason about information flow in multi-agent systems. It was this line of thought that led to the algebraic axiomatics of DEL, referred to as an *epistemic system* [9,2]. *Epistemic systems* subsume the existing relational models of DEL, but are more general in the sense that they do not rely on a Boolean setting and model epistemic and dynamic modalities as adjoint pairs.

The sequent calculus presented in [2,9] lacks a cut-elimination theorem, thus this paper presents, for the first time, an algorithm for proving properties of interactive multi-agent scenarios encoded in *epistemic systems* and an implementation of this algorithm as the program `AXIMO` [8] written in C^{++} . The input language of `AXIMO` is a restriction of the terms of an *epistemic system* and its algorithm implements a rewrite system and a reasoner. The rewrite system is based on the axioms of the algebra and reduces an input inequality to a set of atomic ones. The reasoner uses recursion over the assumptions of the scenario and completes the proof of the input property. The termination of the program follows from the finite number of inequalities generated by the rewrite engine and termination of the finite number of recursive calls. We prove that the program is sound with regard to *epistemic systems*. As test cases, we present and analyze the proofs of properties of honest and dishonest versions of a coin toss scenario and the muddy children puzzle.

`AXIMO` came out of the Masters' project of the first author in Computer Software Engineering, under supervision of the second author. We need to compare its efficiency with that of `DEMO` [6], which is a model checker based on the Kripke semantics of DEL. We believe that the domain of application of `AXIMO` can be extended to proving properties of security protocols (both classical and quantum), by modularly adding their relevant axioms (e.g. hashes and signatures, non-local quantum co-relations), thus re-use the existing dynamic epistemic machinery underneath.

2 The Algebra

We recall the definition of the algebraic semantics of dynamic epistemic logic from previous work [2,9]. It consists of a triple $(M, Q, \{app_A\}_{A \in \mathcal{A}})$ where $Q = (Q, \vee, \bullet, \tau, \top, \perp)$ is a quantale, $M = (M, \vee, \top, \perp)$ is its right module via the action $- \cdot - : M \times Q \rightarrow M$, and $\{app_A\}_{A \in \mathcal{A}}$ is a family of endomorphisms $app_A = (app_A^M : M \rightarrow M, app_A^Q : Q \rightarrow Q)$ satisfying

$$\tau \leq app_A^Q(\tau) \quad (1)$$

$$app_A^Q(q \bullet q') \leq app_A^Q(q) \bullet app_A^Q(q') \quad (2)$$

$$app_A^M(m \cdot q) \leq app_A^M(m) \cdot app_A^Q(q) \quad (3)$$

We define the stabilizer of Q in M as follows

$$Stab(Q) = \{m \in M \mid \forall q \in Q, m \cdot q \leq m\}$$

For each action of Q we define a kernel in M as follows

$$\forall q \in Q, \ker(q) = \{m \in M \mid m \cdot q = \perp\}$$

We recall the axioms for the action of Q on M

$$\begin{aligned} (\bigvee_i m_i) \cdot q &= \bigvee_i (m_i \cdot q), & m \cdot (\bigvee_i q_i) &= \bigvee_i (m \cdot q_i) \\ m \cdot (q \bullet q') &= (m \cdot q) \cdot q', & m \cdot \tau &= m \end{aligned}$$

Since the action preserves all joins of M , it has a Galois right adjoint on M denoted by $- \cdot q \dashv [q]-$ and defined as

$$m \cdot q \leq m' \quad \text{iff} \quad m \leq [q]m'$$

The app_A endomorphisms are join preserving on M and Q , hence they also have Galois right adjoints denoted by $app_A^M(-) \dashv \Box_A^M-$ and $app_A^Q(-) \dashv \Box_A^Q-$ and defined as

$$\begin{aligned} app_A^M(m) \leq m' & \quad \text{iff} \quad m \leq \Box_A^M m' \\ app_A^Q(q) \leq q' & \quad \text{iff} \quad q \leq \Box_A^Q q' \end{aligned}$$

2.1 Interpretation of the Algebra

Elements of the module are interpreted as logical propositions and the partial order of the module as the logical entailment between propositions. The elements of the quantale are interpreted as communication actions and the join on the quantale stands for the non-deterministic choice of actions. Hence the order of the quantale is the order of non-determinism. The composition of quantale \bullet is the sequential composition of actions. The action of the quantale on the module $m \cdot q$ is the update of information in m by action q . Its right adjoint is indeed the *weakest precondition* of Hoare logic and the *dynamic modality* of PDL and DEL with the following reading

- $[q]m$ is all the propositions that become true after applying action q to proposition m . We read it as ‘after doing action q proposition m holds’.

Each app_A map denotes the *appearance* of agent A as follows

- $app_A^M(m)$ is all the propositions that *appear* to agent A as true where as in reality proposition m is true. We read it as ‘the *appearance* of agent A of proposition m ’.
- $app_A^Q(q)$ is all the actions that *appear* to agent A as happening where as in reality action q is happening. We read it as ‘the *appearance* of agent A of action q ’.

The Galois right adjoint in each case is the epistemic modality denoting *not necessarily truthful knowledge* or in a context where no wrong knowledge is allowed *possibly wrong belief*:

- We read $\Box_A^M m$ as ‘agent A knows that proposition m holds’.
- We read $\Box_A^Q q$ as ‘agent A knows that action q is happening’.

The set $Stab(Q)$ is interpreted as the set of *facts*. These are elements of the module that are stable under any update. The reason for stability is that our actions are epistemic and do not change the facts of the world. If a fact is true before an action, it will stay true afterwards.

The kernel of each action is its *co-precondition* or *co-content*, that is the set of states to which it *cannot* apply.

Example 2.1 Consider a coin tossing scenario with two agents 1 and 2, where agent 1 tosses a coin and covers it. None of the agents know on what face the coin has landed. This scenario is encoded in an epistemic system $(M, Q, \{app_A\}_{A \in \mathcal{A}})$ with states $s_H, s_T \in M$, agents $1, 2 \in \mathcal{A}$, and facts $f_H, f_T \in Stab(Q)$. State s_H is the state in which the coin has landed heads and fact f_H is the fact saying ‘the coin is heads’. We thus have $s_H \leq f_H$ and similarly $s_T \leq f_T$ for the state in which the coin is tails and its corresponding fact. Since both of the agents are uncertain about the face of the coin, we have $app_1(s_H) = app_1(s_T) = s_H \vee s_T$ and similarly for 2. Suppose now that 1 uncovers the coin and publicly announces that it is heads. This is the action $a_H \in Q$ that appears as it is to all the agents since it is public $app_1(a_H) = app_2(a_H) = a_H$. Since a_H is the announcement of heads, it cannot apply to the states that satisfy tails, that is $f_T \leq ker(a_H)$. We want to prove that after the announcement of heads, agent 2’s uncertainty gets waived and he gets to know the fact that the coin is heads, that is $s_H \leq [a_H] \Box_2 f_H$. By dynamic adjunction this inequality holds iff we have $s_H \cdot a_H \leq \Box_2 f_H$. By epistemic adjunction this is iff $app_2(s_H \cdot a_H) \leq f_H$, by axiom (3) of epistemic systems it is enough to prove $app_2(s_H) \cdot app_2(a_H) \leq f_H$. Now we replace the $app_2(s_H)$ and $app_2(a_H)$ with their assumed values and obtain $(s_H \vee s_T) \cdot a_H \leq f_H$, which is equivalent to $(s_H \cdot a_H) \vee (s_T \cdot a_H) \leq f_H$ by join preservation of the action of Q on M . By definition of disjunction, we have to prove $s_H \cdot a_H \leq f_H$ and $s_T \cdot a_H \leq f_H$. The first one follows from assumption $s_H \leq f_H$, order preservation of update $s_H \cdot a_H \leq f_H \cdot a_H$, and stability of facts $f_H \cdot a_H \leq f_H$. The second follows since $s_T \leq f_T \leq ker(a_H)$ and thus $s_T \cdot a_H = \perp \leq f_H$.

In a similar fashion, if agent 1 does a lying action \bar{a}_H and announces heads when he sees tails, we prove that agent 1 acquires *wrong* knowledge, that is $s_T \leq [\bar{a}_H] \Box_2 f_H$ by assuming $app_1(\bar{a}_H) = \bar{a}_H$ where as $app_2(\bar{a}_H) = a_H$ and $ker(\bar{a}_H) = f_H$.

3 The Program

3.1 Input/Output

Assumptions of a scenario are either read from a text input file⁴ or asked from the user and assigned interactively. These assumptions include, state variables $s \in At_{St}$, action variables $a \in At_{Ac}$, and fact variables $f \in At_{Ft}$, atomic inequalities about which states satisfy which facts $s \leq f$, appearance to agents of states and actions, which are sets of states and actions respectively, and kernels of actions. Appearances have the following form

$$app_A(s) = \{s_1, \dots, s_n\}, \quad app_A(a) = \{a_1, \dots, a_n\}$$

Kernels are generated via the following syntax

$$k ::= f \mid \Box_A f \mid k * k$$

The inequality to be verified about the scenario is of the form $l \leq r$ and is generated via the following syntax

⁴ We do not yet have a general format for the input file.

$ \begin{aligned} l &::= s \mid l.a \mid l_1 * l_2 \\ r &::= f \mid \Delta r \mid r_1 * r_2 \\ a &::= a_1 \vee a_2 \end{aligned} $	$ \begin{aligned} * &::= \wedge \mid \vee \\ \Delta &::= \square_A \mid [a] \end{aligned} $
--	--

The language presented here is a slight variation of what is used by the program where states, facts, and actions are enumerated, dynamic modality $[a]$ is denoted by $d(i)$ and epistemic modality \square_A by $e(j)$. `AXIMO` processes the input inequality and the assumptions and returns a ‘passed’ or ‘failed to prove’ answer. While computing, the computation steps are broadcasted to the console, so that e.g. the user can find out what was problematic in case of a ‘failed to prove’ answer.

3.2 Algorithm and Complexity

After verifying that the input inequality is of the correct form, the program proceeds by processing the input inequality. This is done via two sets of rewrite rules: on inequalities and on the left and right hand side expressions of inequalities. The rules for inequality rewriting are as follows

Inequality rewriting rules:	
$l_1 \wedge l_2 \leq r_1 \vee r_2 \rightsquigarrow$	$ \begin{cases} l_1 \leq r_1 & \text{or} \\ l_1 \leq r_2 & \text{or} \\ l_2 \leq r_1 & \text{or} \\ l_2 \leq r_2 & . \end{cases} \quad (1) $
$l_1 \vee l_2 \leq r_1 \wedge r_2 \rightsquigarrow$	$ \begin{cases} l_1 \leq r_1 & \text{and} \\ l_1 \leq r_2 & \text{and} \\ l_2 \leq r_1 & \text{and} \\ l_2 \leq r_2 & . \end{cases} \quad (2) $
$l \leq \Delta r \rightsquigarrow$	$ \begin{cases} l \cdot a \leq r & \Delta = [a] \\ app_A(l) \leq r & \Delta = \square_A \end{cases} \quad (3) $

The rules for expression rewriting are as follows

Expression rewriting rules:	
$\Delta(r_1 * r_2) \rightsquigarrow$	$(\Delta r_1) * (\Delta r_2) \quad (4)$
$(l_1 * l_2) \cdot a_1 \cdots a_n \rightsquigarrow$	$(l_1 \cdot a_1 \cdots a_n) * (l_2 \cdot a_1 \cdots a_n) \quad (5)$
$l \cdot (a_1 \vee a_2) \rightsquigarrow$	$(l \cdot a_1) \vee (l \cdot a_2) \quad (6)$
$[a_1 \vee a_2] r \rightsquigarrow$	$[a_1] r \vee [a_2] r \quad (7)$
$app_A(l_1 * l_2) \rightsquigarrow$	$app_A(l_1) * app_A(l_2) \quad (8)$
$app_A(l \cdot a_1 \cdots a_n) \rightsquigarrow$	$app_A(l) \cdot app_A(a_1) \cdots app_A(a_n) \quad (9)$

The appearances of atomic states and actions and the kernels of atomic actions are rewritten to their inputted value via the following 3 expression rewriting rules

Rewriting using the input:

$$app_A(s) \rightsquigarrow s_1 \vee \cdots \vee s_n \quad (10)$$

$$app_A(a) \rightsquigarrow a_1 \vee \cdots \vee a_n \quad (11)$$

$$ker(a) \rightsquigarrow k \quad (12)$$

The decision procedure has two steps: (I) elimination of the $*$ and Δ connectives from the input inequality using the above rules, the resulting inequalities are written to either the ‘and’ or the ‘or’ list, (II) recursive elimination of the inequalities of these lists using the assumptions, i.e. facts satisfies by states and kernels of actions. A ‘passed’ or ‘failed to prove’ message is returned based on the fullness of the lists. This procedure is briefed below

(I) Eliminate $*$ and Δ connectives

- (i) repeat until all $*$ ’s are eliminated, write to ‘and’ - ‘or’ lists
 - (a) eliminate the $*$ ’s outside scope of a Δ or $- \cdot a$ by 1,2
 - (b) push the $*$ ’s inside scope of a Δ or $- \cdot a$ to outside by 4-7
- (ii) eliminate the Δ ’s by 3
- (iii) push the $*$ ’s and $- \cdot a$ ’s inside scope of app_A to outside b 8,9
- (iv) eliminate app_A by repeating: 10,11; do (i).(a); write to ‘or’ list

At the end of this step all the inequalities will be of the following atomic form

$$s \cdot a_1 \cdots a_n \leq f$$

(II) Eliminate inequalities

- For each inequality in the ‘and’ list do
 - (i) if $s \leq f$ in assumptions then eliminate,
 - (ii) else repeat

{	recursively call $s \leq ker(a_1)$
	recursively call $s \cdot a_1 \leq ker(a_2)$
	...
	recursively call $s \cdot a_1 \cdots a_{n-1} \leq ker(a_n)$
- If the ‘and’ list is empty return ‘passed’.
- else repeat (II) for ‘or’ list, return ‘passed’ after 1st elimination,
- If nothing gets eliminated from ‘or’ list, return ‘failed to prove’.

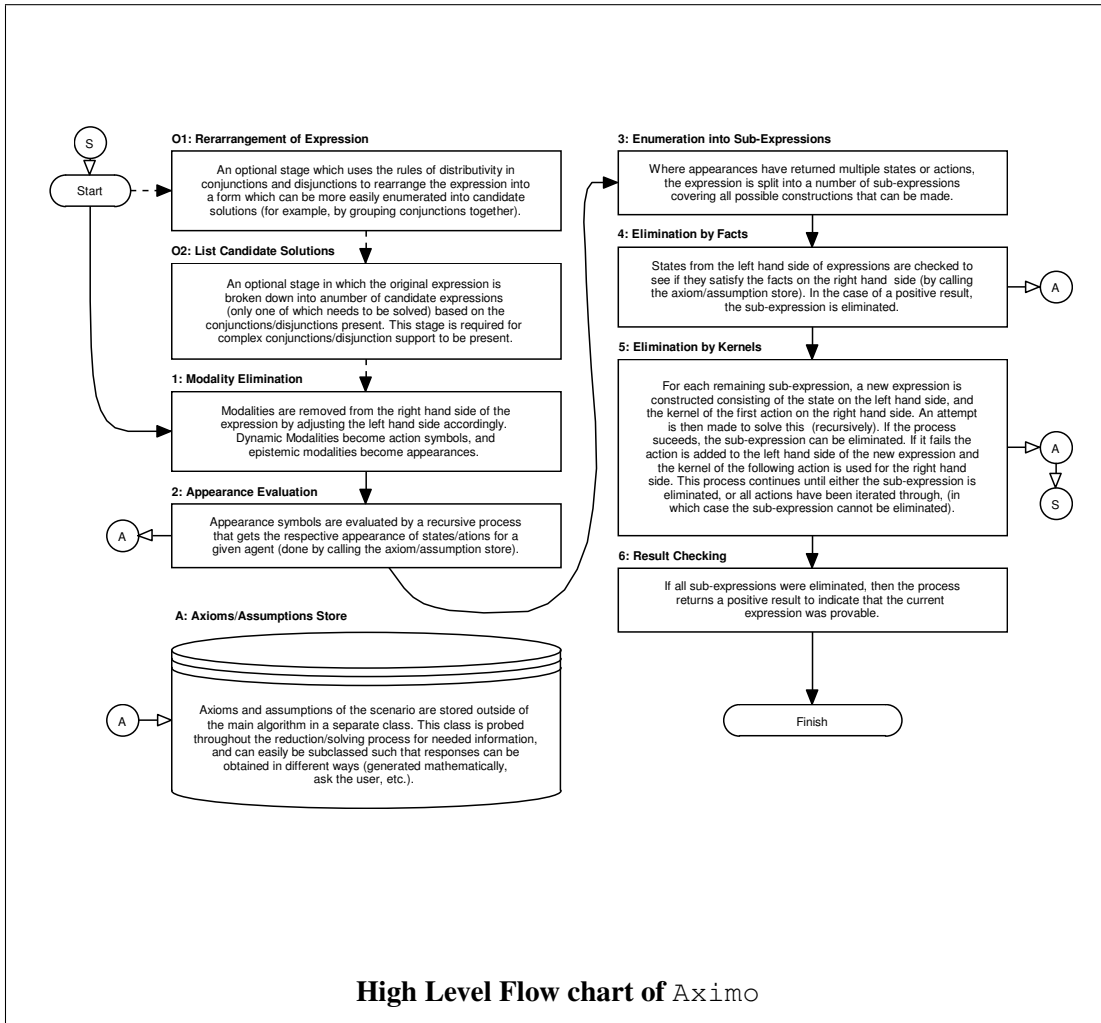
A simple calculation provides us with the maximum number of atomic inequalities that are generated from an input inequality by following the above procedure. This is equal to

$$W \times W' \times W''$$

where $W = Disj \times Conj$ for $Disj$ the number of disjuncts in the input inequality and $Conj$ the number of conjuncts of input inequality. For k maximum cardinality of the appearance sets of the states and actions, n the number of epistemic modalities of the input inequality and m the number of actions plus dynamic modalities of the input inequality, we

have $W' = (k^n)^{(m+1)}$. For n' the maximum number of epistemic modalities of the kernels of actions of the generated atomic inequalities, we have $W'' = \sum_{i=0}^m (k^{n'})^i$ as the sum of the number of inequalities generated by recursion for the kernel of actions. Since dynamic modalities are not allowed in the kernel of actions, the recursive calls are all terminating. As a result, the procedure described above is terminating and will always provide the user with an answer. A ‘failed to prove’ answer means that the program could not prove that the input inequality follows from the assumptions provided by the user.

Example 3.1 In a scenario where either no one cheats or no one suspects the cheating actions, the number of inequalities in W' and W'' reduce to k^n and $(k^{n'}) \times m$ respectively. This is because in either case the sets of appearances of actions of the scenario is a singleton. If the cheating action is suspected by at most w alternative actions by all the agents, these numbers become k^{2n} and $w^{m \times n'}$, respectively.



3.3 Data Structures

Internally, inequalities are stored as a pair of classes each representing a list, for symbols on the left and right hand sides of the inequality, respectively. These are pointed to

from an encapsulating class which is used to provide inequality-wide functionality and handle the inequality from a single pointer. By using two separate list classes, modifications and function calls can be made on each side of the inequality independently from the other, reducing difficulty of implementation. The solving process uses a list of lists to keep track of its progress. Each item in the first list represents the various candidate solutions, with the entry itself being a list to all the sub-inequalities present in that particular candidate. It is worth noting that the size of these lists can be quite dynamic, expanding as new candidates/sub-inequalities are created, and shrinking as sub-inequalities are successfully eliminated.

3.4 Semantics

Given an epistemic system $\mathcal{E} = (M, Q, \{app_A\}_{A \in \mathcal{A}})$, we interpret the expressions of Aximo via the following pair of maps

$$\llbracket - \rrbracket_M : \mathcal{L} \rightarrow M, \quad \llbracket - \rrbracket_Q : \mathcal{L}_a \rightarrow Q$$

where \mathcal{L} is the set of expressions generated from the syntax of Aximo and \mathcal{L}_a is the action-only expressions generated by the syntax of actions. These maps are defined in a routine fashion by induction as follows for $x \in \{l, r\}$

$$\begin{aligned} \llbracket s \rrbracket_M &= s, & \llbracket a \rrbracket_Q &= a, & \llbracket f \rrbracket_M &= f \\ \llbracket [a] r \rrbracket_M &= \llbracket [a]_Q \rrbracket \llbracket r \rrbracket_M, & \llbracket \square_A r \rrbracket_M &= \square_A \llbracket r \rrbracket_M \\ \llbracket l \cdot a_1 \cdots a_n \rrbracket_M &= \llbracket l \rrbracket_M \cdot \llbracket a_1 \rrbracket_Q \bullet \cdots \bullet \llbracket a_n \rrbracket_Q \\ \llbracket x_1 * x_2 \rrbracket_M &= \llbracket x_1 \rrbracket_M * \llbracket x_2 \rrbracket_M, & \llbracket a_1 \vee a_2 \rrbracket_Q &= \llbracket a_1 \rrbracket_Q \vee \llbracket a_2 \rrbracket_Q \end{aligned}$$

Theorem 3.2 Soundness. *For a set \mathcal{S} of assumption inequalities, an epistemic system \mathcal{E} in which \mathcal{S} holds, and a pair of interpretation maps $(\llbracket - \rrbracket_M, \llbracket - \rrbracket_Q)$, we have that if Aximo returns a ‘passed’ answer to an inequality then the inequality holds in \mathcal{E} .*

Proof. Assume Aximo has returned ‘passed’ to an inequality $l \leq r$ generated by its syntax, we show that $\llbracket l \rrbracket_M \leq \llbracket r \rrbracket_M$. The proof has two steps: (1) Aximo ’s rewrite rules are truth preserving, and (2) Aximo ’s procedure on atomic inequalities is sound, that is if it passes $s \cdot a_1 \cdots a_n \leq f$ then $\llbracket s \cdot a_1 \cdots a_n \rrbracket_M \leq \llbracket f \rrbracket_M$. The former is straightforward, the latter is done by induction on the number of actions in an atomic inequality. In the induction base case there are no actions and Aximo passes $s \leq f$ if it is in \mathcal{S} . This implies $\llbracket s \rrbracket_M \leq \llbracket f \rrbracket_M$ since inequalities of \mathcal{S} also hold in \mathcal{E} , and by stability of facts and definition of interpretation maps it follows that $\llbracket s \cdot a_1 \cdots a_n \rrbracket_M \leq \llbracket f \rrbracket_M$. For the induction step, assume Aximo has passed $s \cdot a_1 \cdots a_k \leq f$, we need to show $\llbracket s \cdot a_1 \cdots a_k \rrbracket_M \leq \llbracket f \rrbracket_M$. By the antecedent either $s \leq f$ is passed or else any of $s \leq \ker(a_1)$ or $s \cdot a_i \leq \ker(a_{i+1})$ for some $1 \leq i \leq k-1$ are passed. The consequence follows from the first case by the base case of induction and from the second case by the induction hypothesis, for example, if $s \leq \ker(a_1)$ is passed then $\llbracket s \rrbracket_M \leq \llbracket \ker(a_1) \rrbracket_M$ holds, thus by definition of kernel we have $\llbracket s \rrbracket_M \cdot \llbracket a_1 \rrbracket_M = \perp$ and it follows that $\llbracket \perp \cdot a_2 \cdots a_k \rrbracket_M = \perp \leq \llbracket f \rrbracket_M$. \square

Remark 3.3 Since ‘failed to prove’ in AximO is defined by not being ‘passed’, the contrapositive of the above theorem says that for a set of assumptions \mathcal{S} , if the interpretation of an inequality generated by syntax of AximO does not hold in an epistemic system in which \mathcal{S} holds, then AximO returns a ‘failed to prove’ answer to it.

Remark 3.4 State and action Kripke structures, in the sense of [1], are formed from the assumption inequalities in \mathcal{S} : the states of the state Kripke structure are the set of states mentioned in \mathcal{S} and the accessibility relations between them are obtained by taking the union of products of images of the appearance maps with the singleton set of the state under consideration. A similar method on actions and their appearances provides us with a Kripke structure for actions. Applying the construction detailed in [2,9] provides us with a *Boolean* epistemic system that trivially satisfies \mathcal{S} .

4 Test Cases

The first test case is our simple coin toss example with cheating and lying actions. The second test case is the milestone puzzle of muddy children and a versions of it with lying children.

Coin-Toss:

Recall the coin toss scenario from example 2.1. Initially, each agent thinks either the coin is heads f_H or tails f_T . In the case of an honest announcement of heads by agent 1, the program rewrites $s_H \leq [a_H]\Box_2 f_H$ as follows

$$\begin{aligned} s_H \leq [a_H]\Box_2 f_H &\rightsquigarrow^3 s_H \cdot a_H \leq \Box_2 f_H \rightsquigarrow^3 \text{app}_2(s_H \cdot a_H) \leq f_H \rightsquigarrow^9 \\ \text{app}_2(s_H) \cdot \text{app}_2(a_H) &\leq f_H \rightsquigarrow^{10} (s_H \vee s_T) \cdot \text{app}_2(a_H) \leq f_H \rightsquigarrow^{11} \\ (s_H \vee s_T) \cdot a_H &\leq f_H \rightsquigarrow^5 s_H \cdot a_H \vee s_T \cdot a_H \leq f_H \rightsquigarrow^2 \begin{cases} s_H \cdot a_H \leq f_H & \text{and} \\ s_T \cdot a_H \leq f_H \end{cases} \end{aligned}$$

At this point we have two inequalities in the ‘and’ list. The program eliminates the first one since $s_H \leq f_H$ is in the assumptions of scenario. For the second one $s_T \leq f_H$ is not in the assumptions so the program recurses on $s_T \leq \ker(a_H)$. This gets re-written to $s_T \leq f_T$ by rule (12) and since $s_T \leq f_T$ is an assumption of the scenario, it gets eliminated from the list. So the program folds back and eliminates $s_T \leq \ker(a_H)$ from the ‘and’ list, now the list is empty and it returns ‘passed’.

The lying announcement is dealt with similarly. The output screen for the inequality $s_T \leq [\bar{a}_H]\Box_A f_H$ is presented below, where s_H, s_T, f_H, f_T are represented respectively by $s(0), s(1), f(0), f(1)$, the lying action is $a(2)$ and the honest announcement action is $a(0)$. The epistemic modalities \Box_2 is denoted by $e(2)$ and the dynamic modality $[\bar{a}_H]$ by $d(2)$.

```

Enter expression: s(0)(<= d(2)e(1)f(0)
Solving- s(1)(<= d(2)e(1)f(0)
Rearrangement/Optimisation- s(1)(<= d(2)e(1)f(0)
Candidate Solutions- s(1)(<= d(2)e(1)f(0)
Attempting to Solve Candidate- s(1)(<= d(2)e(1)f(0)
Candidate Enumerated- s(1)(<= d(2)e(1)f(0)
Dynamic Modalities Removed- s(1)a(2)(<= e(1)f(0)
Epistemic Modalities Removed- app(1|s(1))app(1|a(2))(<= f(0)
Appearances Evaluated- s(0,1)a(2)(<= f(0)
Further Enumeration- s(0)a(2)(<= f(0), s(1)a(2)(<= f(0)
Parts Remaining After Elimination by Axioms- s(1)a(2)(<= f(0)
Parts Remaining After Elimination by Known Solution- s(1)a(2)(<= f(0)
Performing Elimination by Action Kernels Trying- s(1)(<= kernel(a(2))
-as- s(1)(<= f(1) - Solving s(1)(<= f(1)
- Rearrangement/Optimisation- s(1)(<= f(1)
- Candidate Solutions- s(1)(<= f(1)
- Attempting to Solve Candidate- s(1)(<= f(1)
- Candidate Enumerated- s(1)(<= f(1)
- Dynamic Modalities Removed- s(1)(<= f(1)
- Epistemic Modalities Removed- s(1)(<= f(1)
- Appearances Evaluated- s(1)(<= f(1)
- Further Enumeration- s(1)(<= f(1)

- Parts Remaining After Elimination by Axioms- - *none*
--Expression Passed--
Parts Remaining After Elimination by Action Kernels- *none*
--Expression Passed--
Try another expression? Enter Y/N:
    
```

A sample output screen of Aximo

In the above screen, the assumptions are read from an input file. An alternative would be to ask them from the user interactively, for instance the program asks ‘does $s(0) \leq f(0)$ and if the user enters ‘yes’, it eliminates it. Similarly, the program asks the user ‘what is the appearance of state 0 to agent 1 and rewrites the appearance expression to the disjunction of the values entered by the user, which are separated by commas.

Complexity wise, our calculations show that to verify nested knowledge properties like $s \leq [a'] \square_i \cdots \square_j f_H$ for i, j ranging over the agents present in a coin toss scenario and a' any action, we obtain a better complexity bound as shown below:

Agents	App. of actions	No. cases
honest	singleton	2^n
cheating with no suspicion	singleton	2^n
cheating with suspicion	$ app_i(a') = w$	$2^n \times w$

Muddy children with lying:

The puzzle goes like this: n children are playing in the mud and $k \geq 1$ of them have dirty foreheads. Each child sees other children’s foreheads but cannot see his own. Their father announces ‘at least one of you is dirty’, and asks ‘do you know if you are dirty?’. The children look around and simultaneously reply no! We prove that after $k - 1$ rounds of no answers, all the dirty children get to know that they are dirty. The children are denoted by numbers $\mathcal{A} = \{i \mid 1 \leq i \leq n\}$. The states are denoted by s_β for $\beta \subseteq \mathcal{A}$ where each s_β represents the state in which the children in β are dirty and the children not in β are clean. The appearance to each child of each state is $app_i(s_\beta) = s_{\beta \cup \{i\}} \vee s_{\beta \setminus \{i\}}$, that is the choice of two states: in one he is dirty and in another one he is clean. The facts are

$\{f_\emptyset\} \cup \{f(i) \mid 1 \leq i \leq n\} \cup \{f'(i) \mid 1 \leq i \leq n\}$, where f_\emptyset stands for 'no child is dirty', $f(i)$ stands for 'child i is dirty', and $f'(i)$ stands for 'child i is clean'. Each state satisfies its corresponding fact, that is $s_\beta \leq f(i)$ for all $i \in \beta$ and $s_\beta \leq f'(i)$ for all $i \notin \beta$. The actions include father's original announcement a and the children's simultaneous no answers (all encoded in the same action) a' . These actions are public announcements, so their appearances to each child is identity, that is $app_i(a) = a$ and $app_i(a') = a'$. The kernel of a is f_\emptyset . The kernel of the no answers a' is $\bigvee_{i=1}^n \Box_i f(i)$, that is the state in which some child knows that he is dirty. We input the following to the program for $1 \leq i \leq k$

$$s_{\{1,2,\dots,k\}} \leq [a] \underbrace{[a'] \cdots [a']}_{k-1} \Box_i f(i)$$

The program does the following rewriting

$$\begin{aligned}
 s_{\{1,2,\dots,k\}} &\leq [a] \underbrace{[a'] \cdots [a']}_{k-1} \Box_i f(i) \rightsquigarrow^3 s_{\{1,2,\dots,k\}} \cdot a \leq \underbrace{[a'] \cdots [a']}_{k-1} \Box_i f(i) \rightsquigarrow^3 \\
 s_{\{1,2,\dots,k\}} \cdot a \cdot a' &\leq \underbrace{[a'] \cdots [a']}_{k-2} \Box_i f(i) \rightsquigarrow^3 \cdots \rightsquigarrow^3 s_{\{1,2,\dots,k\}} \cdot a \cdot \underbrace{a' \cdots a'}_{k-1} \leq \Box_i f(i) \rightsquigarrow^3 \\
 app_i(s_{\{1,2,\dots,k\}} \cdot a \cdot \underbrace{a' \cdots a'}_{k-1}) &\leq f(i) \rightsquigarrow^9 app_i(s_{\{1,2,\dots,k\}}) \cdot app_i(a) \underbrace{app_i(a') \cdots app_i(a')}_{k-1} \leq f(i) \\
 &\rightsquigarrow^{10} (s_{\{1,2,\dots,k\}} \vee s_{\{1,2,\dots,k\} \setminus \{i\}}) \cdot app_i(a) \underbrace{app_i(a') \cdots app_i(a')}_{k-1} \leq f(i) \rightsquigarrow^{11} \cdots \rightsquigarrow^{11} \\
 (s_{\{1,2,\dots,k\}} \vee s_{\{1,2,\dots,k\} \setminus \{i\}}) \cdot a \underbrace{a' \cdots a'}_{k-1} &\leq f(i) \rightsquigarrow^5 \\
 s_{\{1,2,\dots,k\}} \cdot a \underbrace{a' \cdots a'}_{k-1} \vee s_{\{1,2,\dots,k\} \setminus \{i\}} \cdot a \underbrace{a' \cdots a'}_{k-1} &\leq f(i) \rightsquigarrow^2 \left\{ \begin{array}{l} s_{\{1,2,\dots,k\}} \cdot a \underbrace{a' \cdots a'}_{k-1} \leq f(i) \text{ and} \\ s_{\{1,2,\dots,k\} \setminus \{i\}} \cdot a \underbrace{a' \cdots a'}_{k-1} \leq f(i) \end{array} \right.
 \end{aligned}$$

The first inequality of the list gets eliminated since $s_{\{1,2,\dots,k\}} \leq f(i)$ is in the assumptions. For the second inequality the program does $k-1$ recursive calls with the kernels of actions, and only the last call, that is $s_{\{1,2,\dots,k\} \setminus \{i\}} \cdot a \underbrace{a' \cdots a'}_{k-2} \leq ker(a')$ gets eliminated from the list and results in a 'passed' answer.

For a lying version, assume after $k-1$ rounds of no answers, the dirty children lie in round k by still announcing that they do not know that they are dirty, as a result the clean children get confused and wrongly think that they are dirty. The inequality to be verified is

$$s_{\{1,2,\dots,k\}} \leq [a] \underbrace{[a'] \cdots [a']}_{k-1} [a''] \Box_j f'(j)$$

where $k+1 \leq j \leq n$ and a'' is the lying action of dirty children with appearance identity to the dirty children but an honest no answer to the clean children, i.e. $app_{j'}(a'') = a'$. The proof is very similar to the honest case. For a demo with encoded assumptions for a number of muddy children scenarios see the webpage of `AXIMO` [8], under **Muddy Children Demonstration Version** in the **Download** section.

5 Challenges and Future Work

Theoretical challenges. (1) The decision procedure can be optimized by finding invertible versions of our non-invertible rules, e.g. 1,2 4, and by turning the expression rewriting rules into inequality rewriting ones. (2) The language can be made more expressive by adding negation and its corresponding rules, e.g. those of a Boolean or a Heyting algebra. (3) The complexity can be improved by cutting down on recursive calls to kernels and instead use stability theorems, for instance, the one developed in previous work [4], which expresses stability under update with any action that has a positive content.

Practical challenges. (1) In order to stop `AXIMO` from looping indefinitely, we only allow epistemic modalities in the kernels of actions. A natural generalization would be to relax this and instead use loop checking. (2) There are overlaps between the sub-inequalities generated from by the rewrite system, in order to avoid repetition, we aim to make the program memorize those. (3) The main algorithm in `AXIMO` relies heavily on list manipulation and contains a lot of dynamic memory allocation, making it more suitable to a language such as Digital Mars D which supports list handling and garbage collection at the compiler/language specification level, thus improving overall performance, e.g. porting the algorithm to D would not reduce its accessibility to developers.

Comparison. `DEMO` is a model checker [6] based on the underlying Kripke semantics of DEL. Its input is the initial kripke structure of the scenario and the kripke structures of the actions involved. Its main task is computing the *update product* of these structures, as introduced in [1], and then browsing it to model check a dynamic epistemic property. We defer a formal comparison of `AXIMO` and `DEMO` to future work and only hint to the fact that since `AXIMO` is based on a non-Boolean propositional setting and moreover has an interactive mode of entering assumptions, it stores less information about states than `DEMO`.

Acknowledgement

We thank Samson Abramsky, Corina Cîrstea, Vincent Danos and Ross Duncan for invaluable discussions, comments, and questions.

References

- [1] A. Baltag and L.S. Moss, ‘Logics for epistemic programs’, *Synthese* **139**, 2004.
- [2] A. Baltag, B. Coecke and M. Sadrzadeh, ‘Epistemic actions as resources’, *Journal of Logic and Computation* **17(3)**, 555-585, 2007.
- [3] J. van Benthem, ‘One is a Lonely Number’, Technical Report PP-2002-27, ILLC, Amsterdam, 2002, to appear in P. Kopke, ed., *Colloquium Logicum*, Munster, 2001, AMS Publications.
- [4] C. Cîrstea and M. Sadrzadeh, ‘Coalgebraic Epistemic Update without Change of Model’, *Lecture Notes in Computer Science* **4624**, pp. 158-172, June 2007.
- [5] W. van Der Hoek and M. Wooldridge, ‘Time, Knowledge, and Cooperation: Alternating-Time Temporal Epistemic Logic’, *COORDINATION* 2002.
- [6] Jan van Eijck, CWI, Amsterdam <http://homepages.cwi.nl/~jve/demo/DEMO.pdf>.
- [7] R. Fagin, J. Y. Halpern, Y. Moses and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.
- [8] S. Richards and M. Sadrzadeh, `AXIMO`, downloadable from <http://www.charcoalfeathers.net/research/projects/aximo>, August 2007.
- [9] M. Sadrzadeh, ‘Actions and Resources in Epistemic Logic’, Ph.D. Thesis, University of Quebec at Montreal, 2005, www.ecs.soton.ac.uk/~ms6/all.pdf.

6 Appendix

