A data symmetry reduction technique for temporal-epistemic logic

Mika Cohen[1], Mads Dam[2], Alessio Lomuscio[1], and Hongyang Qu[3]

[1] Department of Computing, Imperial College London, UK
[2] Access Linnaeus Center, Royal Institute of Technology, Sweden
[3] Oxford University Computing Laboratory, UK

Abstract. We present a data symmetry reduction approach for model checking temporal-epistemic logic. The technique abstracts the epistemic indistinguishably relation for the knowledge operators, and is shown to preserve temporal-epistemic formulae. We show a method for statically detecting data symmetry in an ISPL program, the input to the temporal-epistemic model checker MCMAS. The experiments we report show an exponential saving in verification time and space while verifying security properties of the NSPK protocol.

1 Introduction

Abstraction by data symmetry reduction [1] is one of the techniques put forward to tackle the state-explosion problem in model checking reactive systems. While the effectiveness of the methodology is well understood in the context of temporal logics, this is not the case for richer logics. Specifically, no analysis has been conducted so far in the context of temporal-epistemic logic [2]. This seems unsatisfactory as efficient symbolic checkers for epistemic languages have been put forward recently [3–5] and the usefulness of temporal-epistemic specifications demonstrated in a number of application-critical scenarios including web-services [6], automatic fault-detection [7], and security [8].

The models for the applications above display large numbers of initial states, often resulting from randomisation of data parameters (such as nonces and messages), exacerbating further the problem of checking large models. In such models, as it is known, a group of computation paths may well be the same up to renaming the data parameters in question. While in pure temporal logic we can safely collect representatives on these traces and conduct our checks only on these, this is not immediately possible in the presence of temporal-epistemic specifications. In fact, as we recall below, in these frameworks the epistemic operators are defined analysing all possible global states in which the local component for the agent in question is the same even if these belong to different computation paths. Because of this, simply collapsing traces would make some epistemic specifications satisfied in the abstract model even if they were not in the concrete one.

In this paper we show that an alternative methodology solving this problem may be defined. Specifically, we show how we can still reduce the number of

initial states to be considered by using an "abstracted" version of the epistemic relations for agents in the system. We show this reduction is sound and complete for temporal-epistemic specifications, in the sense that no false positives or false negatives are found in the reduced model. We also show how to compute the abstract epistemic relations efficiently, and how to statically detect data symmetry in the input to the temporal-epistemic model checker MCMAS [5] by means of scalarset annotations [1]. The experiments we report on a prototype extension to MCMAS show an exponential reduction in time and space for the verification of the security protocol NSPK.

Related work. *Data symmetry reduction* is a known abstraction technique aiming to collapse states that are equivalent up to a renaming of data, thereby yielding a bisimilar quotient model [1]. There has been no attempt in the literature to extend data symmetry reduction from temporal logic to epistemic logic. Indeed, while abstraction for temporal properties is a well-established research area, abstraction for epistemic properties has only recently begun to receive some attention. In [9, 10], Kripke models for epistemic logic are abstracted by approximating the epistemic relations. However, the models are not computationally grounded, which hampers concrete applications [11]. In [12], computationally grounded systems are abstracted by collapsing local states and actions of agents.

Closer to our contribution, [13] gives a technique for *component symmetry reduction* [14, 15] not too dissimilar from the *data* symmetry reduction technique in this paper. Indeed, Theorem 1 in this paper has a close analogue in [13], although the semantics for the epistemic modality is abstracted there into a counterpart semantics [16]. Beyond this, the main contribution in this paper is to address abstraction and symmetry detection in terms of concrete models represented in the MCMAS model checker. Specifically, we introduce a symbolic extension of MCMAS on which a syntactic criteria can be given that guarantees data symmetry, and we compute the abstract semantics without quantifying over permutations. This allows significantly improved savings in relation to [13].

Overview of paper. The rest of the paper is organised as follows. In section 2 we review the interpreted systems framework, the temporal-epistemic logic CTLK, and the model checker MCMAS. In Section 3 we present the data symmetry reduction technique for CTLK properties of interpreted systems. In Section 4 we show how to detect data symmetry in an interpreted system description in the input language to MCMAS. In Section 5 we report on experimental results for a prototype extension to MCMAS. Finally, Section 6 concludes.

2 Interpreted systems, CTLK, and MCMAS

We model multi-agent systems in the mainstream *interpreted systems* framework [2] and express system requirements in the temporal-epistemic logic CTLK [17]; this section summarises the basic definitions. More details can be found in [2]. We also describe MCMAS [5], a model checker for the verification of CTLK properties of interpreted systems.

Interpreted systems. Consider a set $Ag = \{1...n\}$ of agents. For each agent i, assume a non-empty set L_i of local states that agent i can be in, and a non-empty set ACT_i of actions that agent i can perform. Assume also a non-empty set L_{Env} of states for the environment and a non-empty set ACT_{Env} of actions for the environment. Let $S = L_1 \times \cdots \times L_n \times L_{Env}$ be the set of all possible global states and $ACT = ACT_1 \times \cdots \times ACT_n \times ACT_{Env}$ the set of all possible joint actions. For each agent i assume a local protocol $P_i : L_i \longrightarrow 2^{ACT_i}$ selecting actions depending on the local state of i, and a local evolution function $t_i : L_i \times ACT \longrightarrow L_i$ specifying how agent i evolves from one local state to another depending on its action, the actions of the other agents, and the action of the environment. Analogously, assume an environment protocol $P_{Env} : L_{Env} \longrightarrow 2^{ACT_{Env}}$, and an environment evolution function $t_{Env} : L_{Env} \times ACT \longrightarrow L_{Env}$. Let $P = \langle P_1, \cdots, P_n, P_{Env} \rangle$ be the joint protocol and $t = \langle t_1, \cdots, t_n, t_{Env} \rangle$ be the joint evolution function. Finally, consider a non-empty set $I_0 \subseteq S$ of initial states, and an evaluation function $V : A \longrightarrow 2^S$ for some non-empty set A of propositional atoms.

Definition 1 (Interpreted system). *An interpreted system is a tuple $\mathcal{I} = \langle S, ACT, P, t, I_0, V \rangle$ with a set S of possible global states, a set ACT of possible joint actions, a joint protocol P, a joint evolution function t, a set I_0 of initial states, and an evaluation function V.*

For any global state $g = \langle l_1, \ldots, l_n, l_{Env} \rangle \in S$, we write g_i for the local state l_i of agent i in g, and g_{Env} for the environment state l_{Env} in g.

The local protocols and the local evolution functions together determine how the system of agents proceeds from one global state to the next. The global transition relation $R \subseteq S \times S$ is such that $\langle g, g' \rangle \in R$ if and only if there exists $\overline{a} = \langle a_1, \ldots, a_n, a_{Env} \rangle \in ACT$ such that for all $i \in Ag \cup \{Env\}$, $t_i(\overline{a}, g_i) = g'_i$ and $a_i \in P_i(g_i)$. We assume throughout the paper that the global transition relation R is serial, i.e., for every $g \in S$, there is $g' \in S$ such that gRg'.

A path in $\mathcal{I}$ is an infinite sequence $g^0, g^1, \ldots$ of global states in S such that each pair of adjacent states forms a transition, i.e., $g^j R g^{j+1}$ for all j. The set G of reachable states in $\mathcal{I}$ contains all global states $g \in S$ for which there is a path $g^0, g^1, \ldots, g, \ldots$ starting from some $g^0 \in I_0$.

Intuitively, the local state g_i contains all the information available to agent i: if $g_i = g'_i$ then global state g could, for all agent i can tell, be global state g'. This observation can be used to employ a knowledge modality defined on the relation given by the equality on the local components [2]:

Definition 2 (Epistemic relation). *The epistemic indistinguishability relation $\sim_i \subseteq G \times G$ for agent i is such that $g \sim_i g'$ iff $g_i = g'_i$.*

Computation Tree Logic of Knowledge. We consider specifications in the temporal-epistemic logic CTLK which extends CTL with epistemic modalities.

Definition 3 (CTLK). *Assume a set $Ag = \{1..n\}$ of agents i and a non-empty set A of propositional atoms p. CTLK formulae are defined by the expression:*

$$\phi ::= p \mid \neg\phi \mid \phi \wedge \phi \mid K_i\phi \mid EX\phi \mid EG\phi \mid E(\phi U \phi)$$

The knowledge modality K_i is read "Agent i knows that", the quantifier E is read "For some computation path" and the temporal operators X, G and U are read "In the next state", "Always" and "Until" respectively. We assume customary abbreviations: $\overline{K}_i$ encodes the diamond epistemic modality $\neg K_i \neg$; $AG\,\phi$ represents $\neg E(true\,U\,\neg\phi)$ ("For all paths, always ϕ"); $AF\,\phi$ abbreviates $\neg EG\,\neg\phi$ ("For all paths, eventually ϕ").

Satisfaction of the language above with respect to interpreted systems is defined as standard. Specifically, the knowledge modality K_i is evaluated on an interpreted system $\mathcal{I}$ on a point g by means of Definition 2 as follows:

- $(\mathcal{I}, g) \models K_i\phi$ iff for all g' such that $g \sim_i g'$ we have that $(\mathcal{I}, g') \models \phi$.

The CTL modalities are interpreted on the serial paths generated by the global transition relation R: $(\mathcal{I}, g) \models EX\phi$ iff for some path $g^0, g^1, \ldots$ in $\mathcal{I}$ such that $g = g^0$, we have $(\mathcal{I}, g^1) \models \phi$; $(\mathcal{I}, g) \models EG\phi$ iff for some path $g^0, g^1, \ldots$ in $\mathcal{I}$ such that $g = g^0$, we have $(\mathcal{I}, g^i) \models \phi$ for all $i \geq 0$; $(\mathcal{I}, g) \models E(\phi U \phi')$ iff for some path $g^0, g^1, \ldots$ in $\mathcal{I}$ such that $g = g^0$, there is a natural number i such that $(\mathcal{I}, g^i) \models \phi'$ and $(\mathcal{I}, g^j) \models \phi$ for all $0 \leq j < i$.

We write $[[\phi]]$ for the extension of formula ϕ in $\mathcal{I}$, i.e., the set of reachable states $g \in G$ such that $(\mathcal{I}, g) \models \phi$. We say that formula ϕ is true in system $\mathcal{I}$, written $\mathcal{I} \models \phi$, iff $I_0 \subseteq [[\phi]]$.

MCMAS. The tool MCMAS is a symbolic model checker for the verification of CTLK properties of interpreted systems [5]. We illustrate its input language, the *Interpreted Systems Programming Language* (ISPL), with the bit-transmission protocol, a standard example in temporal-epistemic logic [2].

Example 1 (Bit-transmission protocol [2]). A sender and a receiver communicate over a lossy channel. Their goal is to transmit a bit value $b \in \{0, 1\}$ from the sender to the receiver in such a way that the sender will know that the receiver knows the value of b. The sender sends the bit value and continues to do so until it receives an acknowledgement of receipt. The receiver waits until it receives a bit value and then sends an acknowledgement and re-sends it indefinitely.

The protocol can be modelled as an interpreted system $\mathcal{I}$ with a sender agent S, a receiver agent R and the environment Env as the unreliable channel. We would like to verify that once the sender receives the acknowledgement, the bit held by the receiver is the same as the bit held by the sender, that the receiver knows this, and that the sender knows that the receiver knows this:

$$AG\,(\texttt{recack} \longrightarrow K_S\,K_R\texttt{agree}) \tag{1}$$

where K_S and K_R are the epistemic modalities for the sender and receiver agents respectively, and the propositional atom `agree` holds at a global state g if the variable `bit` in the receiver agrees with the variable `bit` in the sender, i.e., $g_R(\texttt{bit}) = g_S(\texttt{bit})$, and the propositional atom `reckack` holds when the sender has received an acknowledgement. The code in Fig.1 describes the system $\mathcal{I}$ and the CTLK specification (1) in ISPL. The code should be straightforward to understand in view of the above. We refer to [5] for more details.

```
Agent S
Vars
 bit: {0,1};
 rec_ack: {true,false};
Actions = {send__0,send__1,null};
Protocol
 rec_ack=false and bit=0: {send__0};
 rec_ack=false and bit=1: {send__1};
 rec_ack=true: {null};
end Protocol
Evolution
 rec_ack=true if
   R.Action=sendack and
   Env.Action=transmit;
end Evolution
end Agent

Agent Env
Vars:
 state: {ok, error};
end Vars
Actions = {transmit,drop};
Protocol:
 state=ok: {transmit};
 state=error: {drop}
end Protocol
-- Evolution omitted
end Agent
```

```
Agent R
Vars
 bit: {0, 1};
 rec_bit: {true, false};
-- Actions, Protocol omitted
Evolution
 bit=0 and rec_bit=true if
   S.Action=send__0 and
   Env.Action=transmit;
 bit=1 and rec_bit=true if
   S.Action=send__1 and
   Env.Action=transmit;
end Evolution
end Agent

InitStates
 S.rec_ack=false and
 R.rec_bit=false;
end InitStates

Evaluation
 agree if S.bit=R.bit;
 recack if S.rec_ack=true;
end Evaluation

Formulae
 AG recack -> K(S,K(R,agree))
end Formulae
```

Fig. 1. Bit-transmission protocol in ISPL

We briefly describe the features of basic ISPL needed to follow the discussion below (in Section 4); Fig.1 can be consulted for an example. An ISPL programs σ reflects the structure of the defined interpreted system $\mathcal{I}(\sigma)$, with one program section for each agent, and for each agent section four subsections containing, respectively:

- Local variable and domain definitions of the form *X:* $\{d_0, \ldots, d_n\}$.
- Action declarations listing the actions such as `send__0`, `send__1` available to the agent.
- Local protocol specifications of the form *lcond* `:` $\{a_0, \ldots, a_n\}$ where the a_i are actions and *lcond* is a boolean combination of (domain correct) *local equalities* of the form $X = X'$ or $X = d$.
- Local evolution function specifications of the form *assign* `if` *acond* where *assign* is a conjunction of local equalities and *acond* is a boolean combination of atoms i`.Action =` a, where a is an action declared in the agent named i.

In addition, an ISPL program provides an initial state condition and truth conditions for atomic propositions, all in the form of *global state conditions* built from (domain correct) equalities of the form $i.X = j.Y$ or $i.X = d$, where X and Y are local variables of agents i and j respectively.

3 A data symmetry reduction technique

In this section we present a data symmetry reduction technique for CTLK properties of interpreted systems. Subsection 3.1 extends the notion of data symme-

try [1] to interpreted systems; Subsection 3.2 establishes the reduction result; Subsection 3.3 shows how to compute the reduced epistemic relations.

3.1 Data symmetry

We assume that local states are built from variables (as they are in ISPL programs). In detail, an interpreted system $\mathcal{I}$ is given together with a set Var_i of local variables for every agent $i \in Ag \cup \{Env\}$, where each $X \in Var_i$ is associated with a non-empty set D_X, the data domain of X. A local state $l \in L_i$ of agent i is a type respecting assignment to the variables in Var_i, i.e., $l(X) \in D_X$ for every $X \in Var_i$. We write $\mathcal{D}$ for the collection of all domains.

Following [1] we mark domains as either *ordered* or *unordered*.[4] Informally, a system is expected to treat all data from the same unordered domain in a symmetric fashion: Every permutation of data from such a domain should preserve the behaviours of the system.

Definition 4 (Domain permutation). *A domain permutation is a family $\pi = \{\pi^D\}_{D\in\mathcal{D}}$ of bijections $\pi^D : D \longrightarrow D$ that only change values in unordered domains, i.e., if D is ordered, then $\pi^D(d) = d$, for $d \in D$.*

The domain permutation π naturally defines a bijection on the local states L_i of agent i and a bijection on the global states S by point-wise application on data elements inside the states. In detail, for each $l \in L_i$, $\pi(l) \in L_i$ is defined by $\pi(l)(X) = \pi^{D_X}(l(X))$ for local variable $X \in Var_i$; For each global state $g \in S$, $\pi(g) = \langle \pi(g_1), ..., \pi(g_n), \pi(g_{Env})\rangle$.

Definition 5 (Data symmetry). *A set $\Delta \subseteq S$ of states is data symmetric iff $g \in \Delta$ iff $\pi(g) \in \Delta$ for all domain permutations π. A relation $\Delta \subseteq S \times S$ between states is data symmetric iff $\langle g, g'\rangle \in \Delta$ iff $\langle \pi(g), \pi(g')\rangle \in \Delta$ for all domain permutations π. The system $\mathcal{I}$ is data symmetric iff the induced global transition relation R, the set I_0 of initial states, and each extension $V(p)$ of a propositional atom p are data symmetric.*

Example 2. Consider the protocol model $\mathcal{I}$ in Example 1 and mark the bit domain $\{0, 1\}$ as an unordered domain. Two domain permutations are possible: the identity ι leaving all values unchanged, and the transposition $flip$ such that $flip^{\{0,1\}}(0) = 1$ and $flip^{\{0,1\}}(1) = 0$. It can be checked that both ι and $flip$ preserve the global transition relation, the set of initial states, and the extension of the propositional atom `agree`. Therefore the system $\mathcal{I}$ is data symmetric.

Lemma 1. *If system $\mathcal{I}$ is data symmetric, then so is the set G of reachable states, each epistemic relation $\sim_i$, and any formula extension $[[\phi]]$.*

Proof. (Sketch) G is data symmetric: Since I_0 and R are data symmetric, $\sim_i$ is data symmetric: Assume $g \sim_i g'$; then $g_i = g'_i$ and $g, g' \in G$. From the former, $\pi(g_i) = \pi(g'_i)$, i.e., $\pi(g)_i = \pi(g')_i$. But, since G is data symmetric,

[4] Unordered domains are called *scalarsets* in [1].

$\pi(g), \pi(g') \in G$. Thus, $\pi(g) \sim_i \pi(g')$. $[[\phi]]$ is data symmetric: By induction on ϕ we can show that $(\mathcal{I}, g) \models \phi$ iff $(\mathcal{I}, \pi(g)) \models \phi$. For the base step note that the extension $V(p)$ of an atomic proposition is data symmetric. Induction step, epistemic modalities: Since $\sim_i$ is data symmetric. Induction step, temporal modalities: Since the global transition relation R is data symmetric.

Given a data symmetric system $\mathcal{I}$, the global states $g, g' \in S$ are said to be data symmetric, written $g \equiv g'$, if and only if, $\pi(g) = g'$ for some domain permutation π. The equivalence class $[g]$ of global state g with respect to $\equiv$ is called the *orbit of* g. Analogously, the local states $l, l' \in L_i$ are said to be data symmetric, $l \equiv l'$, if and only if, $\pi(l) = l'$ for some domain permutation π.

3.2 Data symmetry reduction

In this section we present a technique that exploits data symmetries to reduce the number of initial states in interpreted systems.

Let $\mathcal{I}$ be a data symmetric interpreted system. An *abstraction of* $\mathcal{I}$ is an interpreted system $\mathcal{I}^A = \langle S, ACT, P, t, I'_0, V \rangle$ where $I'_0 \subseteq I_0$ is minimal such that $I_0 = \{[g] \ : \ g \in I'_0\}$. Thus, the abstract system has a single representative initial state g for each orbit $[g]$ of symmetric initial states in I_0.

Example 3. Consider the data symmetric system $\mathcal{I}$ from Example 2. There are eight initial states in I_0 reflecting the eight possible joint assignments to the variable `bit` in the sender/receiver and the variable `state` in the environment. We can form an abstraction $\mathcal{I}^A = \langle S, ACT, P, t, I'_0, V \rangle$ of $\mathcal{I}$ such that I'_0 contains only four initial states and in each of these `S.bit=1`. Observe that $\mathcal{I}^A \models K_{\mathtt{R}}\,\mathtt{agree} \vee K_{\mathtt{R}}\,\neg\mathtt{agree}$, i.e., in the abstract system initially the receiver knows whether its variable agrees with the senders variable. This follows from the fact that $\mathtt{agree} \leftrightarrow \mathtt{R.bit = 1}$ holds at all reachable states in $\mathcal{I}^A$. By contrast, $\mathcal{I} \not\models K_{\mathtt{R}}\,\mathtt{agree} \vee K_{\mathtt{R}}\,\neg\mathtt{agree}$.

As the example illustrates temporal-epistemic formulae are not preserved from the abstraction $\mathcal{I}^A$ to the original system $\mathcal{I}$ (or vice versa). However, we show that we can make formulae invariant between the original system and the abstract system by abstracting the satisfaction relation.

Definition 6 (Abstract epistemic relation). *The abstract epistemic indistinguishability relation* $\sim_i^A \subseteq G \times G$ *for agent* i *is such that* $g \sim_i^A g'$ *iff* $g_i \equiv g'_i$.

In other words, data symmetric local states are indistinguishable under the abstract epistemic relation $\sim_i^A$. In the abstract semantics the knowledge modality K_i for agent i is defined by the abstract epistemic relation $\sim_i^A$ for agent i.

Definition 7 (Abstract satisfaction). *Abstract satisfaction of* ϕ *at* g *in* $\mathcal{I}$, *written* $(\mathcal{I}, g) \models_A \phi$, *is defined inductively by:*

- *Non-epistemic cases are the same as for standard satisfaction (Section 2)*
- $(\mathcal{I}, g) \models_A K_i\phi$ *iff* $(\mathcal{I}, g') \models_A \phi$ *for all* $g' \in G$ *such that* $g \sim_i^A g'$

Example 4. Continuing Example 3, the abstract semantics avoids the unintended validity, i.e., $\mathcal{I}^A \not\models_A K_{\mathtt{R}}\,\mathtt{agree} \vee K_{\mathtt{R}}\,\neg\mathtt{agree}$. Pick an initial state $g \in I'_0$ in which $g_S(\mathtt{bit}) = g_R(\mathtt{bit}) = 1$. Then, $g' = \langle g_S, flip(g_R), g_{Env}\rangle \in I'_0$ and $g \sim^A_R g'$, since $g_R \equiv flip(g_R)$. However, the atom `agree` has different truth values in g and g'.

Standard satisfaction on a data symmetric interpreted system $\mathcal{I}$ is equivalent to abstract satisfaction on the abstract system $\mathcal{I}^A$.

Theorem 1 (Reduction). *$\mathcal{I} \models \phi$ iff $\mathcal{I}^A \models_A \phi$, assuming $\mathcal{I}$ is data symmetric.*

Proof. (Sketch) By Lemma 1, G is data symmetric, and so $G = \{\pi(g) \mid \text{any } \pi, g \in G^A\}$, where G and G^A are the sets of reachable states in $\mathcal{I}$ and $\mathcal{I}^A$ respectively. Therefore, we can evaluate the epistemic modality in $\mathcal{I}$ by scanning the reduced space G' and apply agent permutations "on the fly", expanding each state g' into its equivalence class $[g']$. So, $(\mathcal{I}, g) \models K_i\phi$ iff $\forall g' \in G^A : \forall \pi : g \sim_i \pi(g') \Rightarrow (\mathcal{I}, \pi(g')) \models \phi$. By Lemma 1, $[[\phi]]$ is data symmetric, and so we can replace the test of the property ϕ at $\pi(g')$ with the test of ϕ at g', and so obtain: $(\mathcal{I}, g) \models K_i\phi$ iff $\forall g' \in G^A : \forall \pi : g \sim_i \pi(g') \Rightarrow (\mathcal{I}, g') \models \phi$. In other words, $(\mathcal{I}, g) \models K_i\phi$ iff $\forall g' \in G^A : g_i \equiv g'_i \Rightarrow (\mathcal{I}, g') \models \phi$. By induction over ϕ, therefore, we obtain: $(\mathcal{I}, g) \models \phi$ iff $(\mathcal{I}^A, g) \models_A \phi$, for all $g \in G^A$. The theorem follows from this, since $[[\phi]]$ is data symmetric by Lemma 1.

3.3 Computing the abstract epistemic relations

Computing the abstract epistemic relations may seem expensive when there is a large number of domain permutations. However, as we show below, we can compute the abstract epistemic relations without applying any domain permutation at all; two local states l and l' are data symmetric if l and l' satisfy the same equalities between variables with the same unordered domain, and in addition each variable with an ordered domain has the same value in l and l'.

Proposition 1 (Equivalence check). *For any $l, l \in L_i$, $l \equiv l'$, if and only if, for all variables $X, Y \in Var_i$ with the same unordered domain $D_X = D_Y$,*

1. *$l(X) = l(Y)$ iff $l'(X) = l'(Y)$*

and for all variables $X \in Var_i$ with an ordered domain D_X,

2. *$l(X) = l'(X)$*

Proof. Pick two local states $l, l' \in L_i$. For each domain $D \in \mathcal{D}$, define the relation $\pi^D = \{\langle l(X), l'(X)\rangle | X \in Var_i, D_X = D\}$. Condition (1) holds iff for each unordered domain D, π^D is functional and injective, i.e., can be extended to a bijection on D. Condition (2) holds iff for each ordered domain D, π^D preserves values, i.e., can be extended to the identity on D.

For symbolic model checkers, Proposition 1 provides a constructive way of computing a boolean formula encoding of the abstract epistemic relations.

4 Data Symmetry Detection

In this section we establish a static test on ISPL programs that establishes whether a given interpreted system is data symmetric, and so amenable to reduction. A natural check [1] is to verify whether or not the program explicitly distinguishes between different values from an unordered domain.

Definition 8 (Symbolic program). *Assume an ISPL program σ and a partition of the variables' domains in σ into ordered and unordered. The program σ, or a program section of σ, is* symbolic *if ground values from an unordered domain appear only in domain definitions.*

For the concept of symbolic ISPL program to be useful, the actions in it need to carry parameters from unordered domains.

Example 5. Consider the program in Fig.1 with the bit-domain marked as unordered; the program is not symbolic. Intuitively, the atomic actions `send__0` and `send__1` could be replaced by one action and a parameter.

4.1 Extended ISPL

We extend ISPL with structured actions that explicitly carry parameters from a specified domain; the parameter can be indicated symbolically by a variable.

```
Agent S
-- Vars, Evolution as in Fig.1
Actions = {send(?bit),null};
Protocol
 rec_ack=false: {send(bit)};
 rec_ack=true: {null};
end Protocol
end Agent
-- Agent Env, InitStates,
-- Evaluation as in Fig.1
```

```
Agent R
-- Vars, Actions, Protocol
-- as in Fig.1
Evolution
 bit=?bit and rec_bit=true if
   S.Action=send(?bit) and
   Env.Action=transmit;
end Evolution
end Agent
```

Fig. 2. Bit-transmission protocol in extended ISPL

The syntax of the extended version of ISPL is as follows; Fig.2 can be consulted for an example. Local variables X of agents are declared as in basic ISPL. Each entry in the list of actions has the form $a(?X)$, where X is a local variable.[5] Intuitively, the *macro-variable* $?X$ represents an arbitrary value of the domain of X. A *term* t is either a local variable X, a macro variable $?X$, or a ground element d (drawn from some domain). A *parametric action* is an expression of the form $a(t)$ where the operation a and the argument term t have identical domains. The syntax for protocol sections and evolution sections are given in

[5] For ease of presentation we restrict attention to the unary case.

the same way as for basic ISPL but using the above definitions of *term* and *parametric action*. The initial states condition and the evaluation section are the same as in basic ISPL (i.e., no macro variables are allowed in equalities).

Intuitively, local variables such as `bit` are evaluated and bound at time of execution in the expected fashion. For instance, in the protocol of S, Fig.2, the parametric action `send(bit)` represents both `send__0` and `send__1` depending on how the term `bit` is instantiated. By contrast, the macro-variable `?bit` in the protocol of R represents an arbitrary bit value.

$$\{a_1(?X_1), \ldots, a_n(?X_n)\} \rightarrow \{a_1(d_{1,1}), \ldots, a_1(d_{1,m_1}), \ldots, a_n(d_{n,1}), \ldots, a_n(d_{n,m_n})\} \quad (2)$$

$$lcond : actions(\overline{x}, \overline{y}) \rightarrow \bigwedge_{\overline{d_1}, \overline{d_2}} \overline{x} = \overline{d_1} \texttt{ and } lcond : actions(\overline{d_1}, \overline{d_2}) \quad (3)$$

$$assign(\overline{y}) \texttt{ if } acond(\overline{x}, \overline{y}) \rightarrow \bigwedge_{\overline{d_1}, \overline{d_2}} assign(\overline{d_2}) \texttt{ if } acond(\overline{d_1}, \overline{d_2}) \quad (4)$$

Fig. 3. Extended ISPL Translation Rules

Translation into basic ISPL Extended ISPL programs are expanded into basic ones by means of the rewrite rules given in Fig. 3. As an illustration, the (symbolic) program in Fig.2 translates to the (non-symbolic) program in Fig.1. Action lists are expanded according to (2) where $\{d_{i,1}, \ldots, d_{i,m_i}\}$ is the domain of $?X_i$. Protocol and evolution rules are expanded to sets of rules (denoted as conjunctions) where $\overline{x}$ is the list of local variables and $\overline{y}$ is the list of macro variables occurring in the rule under consideration, and where we assume that substitutions respect domain assignments. Thus, each macro-variable is replaced during the translation by the elements from its domain.

4.2 Detection theorem

We show that symbolic programs in extended ISPL define data symmetric systems. We assume throughout an extended ISPL program Σ which translates into a basic ISPL program σ, and we write $\mathcal{I}(\Sigma)$ for the interpreted system $\mathcal{I}(\sigma)$ defined by σ. We assume domains are divided into ordered and unordered.

Observe that a domain permutation π defines a substitution of code fragments of σ. In particular, $\pi(a\_\_d) = a\_\_\pi(d)$ for atomic actions $a\_\_d$, and $\pi(x = d) = (x = \pi(d))$ for local equalities $(x = d)$. Continuing, π lifts to a bijection on the set ACT of joint actions by component-wise application: $\pi(\overline{a}) = \langle \pi(a_1), \ldots, \pi(a_n), \pi(a_E) \rangle$.

According to the following lemma, if program Σ is symbolic then agent i is "syntactically data symmetric" in the sense that the set of protocol rules and the set of evolution rules in the translation σ are closed under domain permutations.

Lemma 2 (Syntactic agent closure). *Assume agent i is symbolic in Σ.*

1. *If Δ is a rule i's protocol in σ, then so is $\pi(\Delta)$.*
2. *If Δ is a rule in i's evolution in σ, then so is $\pi(\Delta)$.*

Proof. (Sketch) (1): By rewrite rule (3) of Fig. 3, there is a "source" protocol entry $lcond : actions(\overline{x}, \overline{y})$ in agent i in Σ such that Δ is the rule $(\overline{x} = \overline{d_1}$ `and` $lcond : actions(\overline{d_1}, \overline{d_2}))$, for some $\overline{d_1}, \overline{d_2}$. By rewrite rule (3) again, $(\overline{x} = \pi(\overline{d_1})$ `and` $lcond : actions(\pi(\overline{d_1}), \pi(\overline{d_2}))$ is a protocol rule in agent i in σ. But, this rule is precisely $\pi(\Delta)$, since both $lcond$ and $actions(\overline{x}, \overline{y})$ are symbolic. (2): By rewrite rule (4) of Fig. 3, there is an evolution entry $assign(\overline{y})$ `if` $acond(\overline{x}, \overline{y})$ in agent i in Σ such that Δ is the rule $(assign(\overline{d_2})$ `if` $acond(\overline{d_1}, \overline{d_2}))$, for some $\overline{d_1}, \overline{d_2}$. By rewrite rule (4) again, $(assign(\pi(\overline{d_2}))$ `if` $acond(\pi(\overline{d_1}), \pi(\overline{d_2})))$ is also an evolution rule in agent i in σ. But, this rule is $\pi(\Delta)$, since both $assign(\overline{y})$ and $acond(\overline{x}, \overline{y})$ are symbolic.

It follows that agents are "semantically data symmetric" as defined below.

Definition 9 (Symmetric agent). *Agent i is data symmetric in $\mathcal{I}(\Sigma)$ iff*

1. *$a \in P_i(l)$ iff $\pi(a) \in P_i(\pi(l))$*
2. *$t_i(\overline{a}, l) = l'$ iff $t_i(\pi(\overline{a}), \pi(l)) = \pi(l')$.*

Lemma 3. *If agent i is symbolic in Σ, agent i is data symmetric in $\mathcal{I}(\Sigma)$.*

Proof. (Sketch) The local protocol P_i is data symmetric: Assume $a \in P_i(l)$. By the semantics of basic ISPL, there is a protocol rule $lcond : actions$ in agent i in the translation σ such that $a \in actions$ and l satisfies $lcond$, and so $\pi(a) \in \pi(actions)$ and $\pi(l) \models \pi(lcond)$. But, by Lemma 2.1, $\pi(lcond : actions)$ is also a protocol rule in agent i in σ, and so $\pi(a) \in P_i(\pi(l))$. We conclude that $a \in P_i(l)$ implies $\pi(a) \in P_i(\pi(l))$. The converse implication follows by applying π^{-1}. The local evolution function t_i is data symmetric: Assume $t_i(\overline{a}, l) = l'$. By the semantics of basic ISPL, there is an evolution entry $assign$ `if` $acond$ in agent i in σ such that $\overline{a}$ satisfies $acond$ and $l\,[assign]\,l'$, using Floyd–Hoare logic style notation for local state updates. It follows that $\pi(\overline{a})$ satisfies $\pi(acond)$ and $\pi(l)\,[\pi(assign)]\,\pi(l')$. But, by Lemma 2.2, $\pi(assign$ `if` $acond)$ is an evolution rule in agent i in σ, and so $t_i(\pi(\overline{a}), \pi(l)) = \pi(l')$. We conclude that $t_i(\overline{a}, l) = l'$ implies $t_i(\pi(\overline{a}), \pi(l)) = \pi(l')$. The converse implication follows by applying π^{-1}.

If agents are data symmetric then so is the induced global transition relation.

Lemma 4. *If each agent is data symmetric in $\mathcal{I}(\Sigma)$, then so R.*

Proof. By definition of R, $\langle g, g' \rangle \in R$ iff there is $\overline{a}$ such that $\langle g_i, g'_i \rangle \in t_i(\overline{a})$ and $a_i \in P_i(g_i)$ for $i \in Ag \cup \{E\}$. Since i is data symmetric, this is equivalent to $\langle \pi(g_i), \pi(g'_i) \rangle \in t_i(\pi(\overline{a}))$ and $\pi(a_i) \in P_i(\pi(g_i))$ for $i \in Ag \cup \{E\}$. In turn this is equivalent to $\langle \pi(g)_i, \pi(g')_i \rangle \in t_i(\pi(\overline{a}))$ and $\pi(a_i) \in P_i(\pi(g)_i)$ for $i \in Ag \cup \{E\}$. By definition of R, this is equivalent to $\langle \pi(g), \pi(g') \rangle \in R$.

We reach the symmetry detection result stating that every symbolic program in the extended ISPL defines a data symmetric interpreted system.

Theorem 2 (Detection). *If extended ISPL program Σ is symbolic then interpreted system $\mathcal{I}(\Sigma)$ is data symmetric.*

Proof. (i) R is data symmetric: By Lemmas 3 and 4. (ii) I_0 is data symmetric: The initial states condition `cond` in the basic ISPL translation σ of Σ is symbolic, and so g satisfies `cond` iff $\pi(g)$ satisfies `cond`. (iii) $V(p)$ is data symmetric: Shown as (ii).

5 Implementation and experiments

In this section we describe a prototype extension to MCMAS implementing the data symmetry reduction presented above and report on its performance for a well-known security protocol.

Implementation. The prototype extension takes as input an extended ISPL program in which some domains are marked as unordered, checks that the supplied program is data symmetric (using Detection Theorem 2), compiles it to basic ISPL (using the translation in Section 4), reduces the initial states (as described below) and, finally, checks the supplied CTLK specifications against the abstract semantics (using Proposition 1).

To reduce the initial states, the prototype constructs the symbolic representation of a set S' which contains exactly one representative state for each orbit class, i.e., $S' \subseteq S$ is minimal such that $S = \{[s] : s \in S'\}$, where S is the set of possible global states for the supplied program. Roughly, the symbolic representation of S' is a disjunction of assignments, one assignment for each possible pattern of identities between variables with unordered domains. In detail, let $\mathcal{V}$ be the set of variables with unordered domains, and let $\Delta \subseteq 2^{\mathcal{V}}$ be a partition of $\mathcal{V}$ such that each block $\delta \in \Delta$ contains only variables that share the same domain. Intuitively, the partition Δ represents a pattern of identities between variables in $\mathcal{V}$: $X = Y$ if and only if X and Y belong to the same block $\delta \in \Delta$. For every such partition Δ, the prototype selects an assignment to variables in $\mathcal{V}$ that "agrees" with Δ; The symbolic representation of S' is the disjunction of all such assignments:

$$sym_{S'} = \bigvee_{\Delta} \bigwedge_{\delta \in \Delta, X \in \delta} X = d(\delta) \tag{5}$$

where $d(\delta)$ is a value from the domain shared by variables in block δ; the value $d(\delta)$ is different for different blocks $\delta \in \Delta$ from the same partition.

A symbolic representation of the reduced set I'_0 of initial states can then be obtained by conjuncting $sym_{S'}$ with the initial states condition sym_{I_0} in the supplied program. To optimise the construction, the prototype distributes sym_{I_0} over the disjunction in $sym_{S'}$:

$$sym_{I'_0} = \bigvee_{\Delta} (sym_{I_0} \wedge \bigwedge_{\delta \in \Delta, X \in \delta} X = d(\delta)) \tag{6}$$

i.e., it conjuncts with sym_{I_0} "on the fly" as $sym_{S'}$ is being constructed. As a further optimisation, the prototype generates only some of the possible variable partitions Δ. In particular, if $(X = Y) \wedge sym_{I_0}$ is empty for two variables $X, Y \in \mathcal{V}$, the prototype excludes variable partitions Δ that have a subset δ containing both X and Y.

The prototype computes the symbolic representation of the extension $[[K_i\phi]]$ of an epistemic formula $K_i\phi$ with respect to the abstract satisfaction relation as follows:

$$sym_{[[K_i\phi]]} = sym_G \wedge \neg \bigvee_{\Delta} (sym_\Delta \wedge PreImage(sym_{[[\neg\phi]]} \wedge sym_\Delta, \approx))$$

where sym_G is the symbolic representation of the set of reachable states; Δ ranges over partitions of the set of agent i's local variables with unordered domains; sym_Δ expresses that variables in agent i agree with Δ, i.e., the conjunction of equalities $X = Y$ and inequalities $X \neq Z$ for X, Y belonging to the same block in Δ and X, Z belonging to different blocks in Δ; $sym_{[[\neg\phi]]}$ is the symbolic representation of the extension $[[\neg\phi]]$; $\approx$ relates global states with identical values for the variables in agent i with ordered domains, i.e., $\approx$ is the condition $\bigwedge_X X = prim(X)$ where X ranges over agent i's variables with ordered domains.

NSPK. To evaluate the performance of the technique we tested the prototype on the Needham-Schroeder Public Key protocol (NSPK), a standard example in the security literature [18]. The NSPK protocol involves a number of A–agents and B–agents; each A–agent starts with a nonce (*unique, unpredictable number*) Na, and each B–agent starts with a nonce Nb. We considered the following CAPSL [19] authentication goal for the protocol:

$$Knows\ B : Knows\ A : agree\ A : B : Na, Nb \tag{7}$$

stating that when a protocol session between an A–agent and a B–agent ends, the agents share the nonces Na and Nb, the A–agent knows this and the B–agent knows that the A–agent knows this.[6]

To verify the CAPSL goal (7) we modelled the NSPK protocol as an extended ISPL program with N agents, some of them A–agents and others B–agents. In addition, we modelled a Dolev-Yao attacker [21] in the environment agent. The intruder and all agents start with a unique, non-deterministic nonce value – a value for Na in the case of A–agents and value for Nb in the case of B–agents. Thus we assumed a domain of $N + 1$ nonces; one nonce for the intruder and one for each agent.[7] We marked the domain of nonces as unordered. Finally, we translated the CAPSL goal (7) into the following CTLK formula:

$$\bigwedge_{i:B} AG\,(i.\texttt{Step} = 3 \rightarrow K_i \bigvee_{j:A} (agree(i,j) \wedge K_j \bigvee_{i:B} agree(i,j))) \tag{8}$$

[6] The goal was derived manually in [20].

[7] It would be reasonable to provide the intruder with more than just one initial nonce; the reduction would then yield even bigger savings.

where $i : B$ ranges over B–agents, and $j : A$ ranges over A–agents, and $agree(i, j)$ states that agents i and j agree on the protocol variables Na, Nb, A and B, i.e., $i.Na = j.Na$, $i.Nb = j.Nb$, etc. The specification (8) states that whenever a B–agent i has completed all three protocol steps, the agent i knows that *some* A–agent j agrees with i, and agent i knows that this agent j knows that *some* B–agent i agrees with j.

Experiments Table 1 shows the total verification time (including the time it takes to reduce the initial states) in seconds and the number of reachable states for CTLK specification (8) and different number of participating agents. The experiments ran on a 2 GHz Intel machine with 2GB of memory running Linux. Each run was given a time limit of 24 hours. For this experiment we observed an exponential reduction in both time and space in the number N of agents. Specifically, the state space is reduced by the factor $(N + 1)!$, while the reduction in verification time is more irregular given that MCMAS is a symbolic model checker. We can expect even greater savings for security protocols that involve more than just one unique, unpredictable data value (nonce, session key, password, etc.) per agent.

Table 1. Verification results for NSPK

Agents	Without reduction		With reduction	
	States	Time	States	Time
3	1 536	3	64	1
4	11 400	28	95	4
5	651 600	7 716	905	9
6	–	> 86 400	12 256	24
7	–	> 86 400	21 989	91

6 Conclusions

We presented a data symmetry reduction technique for temporal-epistemic logic in the mainstream interpreted systems framework. The technique uses an abstract satisfaction relation in the reduced system; this was shown to make the reduction sound and complete, i.e., there are no false positives or false negatives in the reduced system. To facilitate the detection of data symmetric systems, i.e., systems amenable to reduction, we extended the interpreted systems programming language (ISPL) with parametric actions. We showed that symbolic programs in the extended ISPL define data symmetric systems. Experiments on the NSPK security protocol show an exponential reduction in verification time and state space for temporal-epistemic security goals.

The reduction technique in this paper reduces initial states only. However, we emphasize that for some applications, such as the security protocol model

considered in this paper, collapsing data symmetric initial states alone yields the same reduced state space as collapsing all data symmetric states, i.e., it yields the quotient model with respect to the orbit relation.

References

1. Ip, C.N., Dill, D.L.: Better verification through symmetry. Form. Methods Syst. Des. **9**(1-2) (1996) 41–75
2. Fagin, R., Halpern, J.Y., Vardi, M.Y., Moses, Y.: Reasoning about knowledge. MIT Press, Cambridge, MA, USA (1995)
3. Gammie, P., van der Meyden, R.: MCK: Model checking the logic of knowledge. In: Proc. CAV'04. Volume 3114 of LNCS., Springer-Verlag (2004) 479–483
4. Nabialek, W., Niewiadomski, A., Penczek, W., Pólrola, A., Szreter, M.: VerICS 2004: A model checker for real time and multi-agent systems. In: Proc. CS&P'04, Humboldt University (2004) 88–99
5. Lomuscio, A., Qu, H., Raimondi, F.: MCMAS: A model checker for multi-agent systems. In: Proc. CAV'09, Springer Verlag (2009.)
6. Lomuscio, A., Qu, H., Solanki, M.: Towards verifying contract regulated service composition. In: Proc. ICWS '08, IEEE Computer Society (2008) 254–261
7. Ezekiel, J., Lomuscio, A.: Combining fault injection and model checking to verify fault tolerance in multi-agent systems. In: Proc. AAMAS'09. (2009) To appear.
8. van der Meyden, R., Su, K.: Symbolic model checking the knowledge of the dining cryptographers. In: Proc. CSFW '04, Washington, DC, USA, IEEE Computer Society (2004) 280
9. Dechesne, F., Orzan, S., Wang, Y.: Refinement of kripke models for dynamics. In: Proc. ICTAC'08, Springer (2008) 111–125
10. Enea, C., Dima, C.: Abstractions of multi-agent systems. In: Proc. CEEMAS'07. (2007) 11–21
11. Wooldridge, M.: Computationally grounded theories of agency. In: Proc. ICMAS'00. IEEE Press (2000) 13–22
12. Cohen, M., Dam, M., Lomuscio, A., Russo, F.: Abstraction in model checking multi-agent systems. In: Proc. AAMAS'09. (2009) To appear.
13. Cohen, M., Dam, M., Lomuscio, A., Qu, H.: A symmetry reduction technique for model checking temporal epistemic logic. In: Proc. IJCAI'09. (2009) To appear.
14. Clarke, E.M., Enders, R., Filkorn, T., Jha, S.: Exploiting symmetry in temporal logic model checking. Form. Methods Syst. Des. **9**(1-2) (1996) 77–104
15. Emerson, E.A., Sistla, A.P.: Symmetry and model checking. Form. Methods Syst. Des. **9**(1-2) (1996) 105–131
16. Lewis, D.: Counterpart theory and quantified modal logic. Journal of Philosophy **65** (1968) 113–126
17. van der Meyden, R., Wong, K.S.: Complete axiomatizations for reasoning about knowledge and branching time. Studia Logica **75**(1) (2003) 93–123
18. Needham, R.M., Schroeder, M.D.: Using encryption for authentication in large networks of computers. Commun. ACM **21**(12) (1978) 993–999
19. Denker, G., Millen, J.: Capsl integrated protocol environment. In: Proc. DISCEX'00, pp 207-221, IEEE Computer Society (2000) 207–221
20. Burrows, M., Abadi, M., Needham, R.: A logic of authentication. ACM Trans. Comput. Syst. **8**(1) (1990) 18–36
21. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Transactions on Information Theory **29**(2) (1983) 198–208