Verification of agent knowledge in dynamic access control policies*

Masoud Koleini, Eike Ritter and Mark Ryan

School of Computer Science
The University of Birmingham
Birmingham, B15 2TT, UK

Abstract. We develop a modeling technique based on interpreted systems in order to verify temporal-epistemic properties over access control policies. This approach enables us to detect information flow vulnerabilities in dynamic policies by verifying the knowledge of the agents gained by both reading and reasoning about system information. To overcome the practical limitations of state explosion in model-checking temporal-epistemic properties, we introduce a novel abstraction and refinement technique for temporal-epistemic safety properties in ACTLK (ACTL with knowledge modality K) and a class of interesting properties that does fall in this category.

1 Introduction

Assume a conference paper review system in which all the PC members have access to the number of the papers assigned to each reviewer. Further assume that a PC member Alice can see the list of the papers that are assigned to another PC member and that are not authored by Alice. Then if Alice is the author of a submitted paper, she can find who the reviewer of her paper is by comparing the number of papers assigned to each reviewer (shown by the system) with the number of the assigned papers of that reviewer which she has access to.

The above is an example of a potential information leakage in *content management systems*, which are collaborative environments that allow users to create, store and manage data. They also allow controlling access to the data based on the user roles. In such multi-agent systems, access to the data is regulated by *dynamic access control policies*, which are a class of authorization rules that the permissions for an agent depend on the state of the system and change when agents interact with the system [1–3]. In complicated access control scenarios, there is always a risk that some required properties do not hold in the system. For instance and for a conference paper review system, the following properties need to hold in the policy:

- It should be impossible for the author of a paper to be assigned as the reviewer of his own paper (temporal safety property).
- There must be no way for the author of a paper to find out who is the reviewer of his paper (epistemic safety property).

* The original version of this paper appeared in Lecture Notes in Computer Science (LNCS), Volume 7795, 2013, pp 448–462.

Epistemic properties take *knowledge* of the agents into account. The knowledge can be gained by directly accessing the information, which complies with one of the meanings of the knowledge in ordinary language, that means the agent *sees* the truth. But agent also knows the truth when he indirectly reasons about it [4].

Information flow as a result of reasoning is a critical vulnerability in many collaborative systems like conference paper review systems, social networks and document management systems, and is difficult to detect. The complication of access control policies in multi-agent collaborative frameworks makes finding such weaknesses more difficult using non-automated mechanisms. Moreover, the state of art dynamic access control verification tools are unable to find such properties as they do not handle epistemic property verification in general. Therefore as the *first contribution* of this paper, we propose a policy authorization language and express how to use the *interpreted systems framework* [5] in order to model the related access control system. Using interpreted systems enables us to address misconfiguration in the policy and information disclosure to unauthorized agents by verifying temporal-epistemic properties expressed in the logic CTLK (CTL with knowledge modality K). The knowledge of an agent in our modelling covers both the knowledge gained by reasoning and by reading information when access permission is granted.

The practical limitation of interpreted systems is the state explosion for the systems of medium to large state space. There is also a limited number of research on the automated abstraction and refinement of the models defined in interpreted systems framework. As the *second contribution*, we develop an novel fully automated abstraction and refinement technique for verifying safety properties in ACTLK (which is a subset of CTLK) over an access control system modelled in the framework of interpreted systems. We extend counterexample guided abstraction refinement [6] to cover the counterexamples generated by the verification of temporal-epistemic properties and when the counterexample is tree-like [7]. In this paper, we only discuss the counterexamples with finite length paths, but this approach can be extended to the paths of infinite length using an unfolding mechanism [6]. We use a model-checker for multi-agent systems [8] and build the abstract model in its modelling language. The refinement is guided using the counterexample generated by the model-checker. The counterexample checking algorithm is provably sound and complete. We also introduce an interactive refinement for a class of epistemic properties that does not fall in ACTLK, but can specify interesting security properties.

The reminder of the technical report is organized as follows: Related works are discussed in section 2, interpreted systems are introduced in section 3, formal syntax and semantics of access control policies are provided in section 4, deriving an interpreted system from a policy is described in section 5, abstraction and refinement technique is given in sections 6 and 7. Case studies and experimental results are included in section 8.

2 Related work

In the area of knowledge-based policy verification, Aucher et al. [9] define *privacy policies* in terms of permitted or forbidden knowledge. The dynamic part of their logic deals with sending or broadcasting data. Their approach is limited in modeling knowledge gained by the interaction of the agents in a multi-agent system. RW framework [2] has the most similar approach with ours. The transition system in RW is build over the knowledge of the active coalition of agents. In each state, the knowledge of the coalition is the accumulation of the knowledge obtained by performing actions or sampling system variables in previous transitions together with the initial knowledge. In the other words, knowledge in RW is gained by reading or altering system variables, not by reasoning about them. This is similar to PoliVer [10], which approximates knowledge by readability. Such verification tools are not able to detect information flow as a result of reasoning.

In the field of abstraction and refinement for temporal-epistemic logic, Cohen et al. [11] introduce the theory of simulation relation and existential abstraction for interpreted systems. Their approach is not automated and they have not provided how to refine the abstract model if the property does not hold and the counterexample is spurious. A recent research on abstraction and refinement for interpreted systems is done by Zhou et al. [12]. Although their work is about abstraction and refinement of interpreted systems, their paper is abstract and mainly discusses the technique to build up a tree-like counterexample when verifying ACTLK properties.

3 Background

3.1 Interpreted systems

Fagin et al. [13] introduced interpreted systems as the framework to model multi-agent systems in games scenarios. They introduced a detailed transition system which contains agents, local states and actions. Such a framework enables reasoning about both temporal and epistemic properties of the system. Lomuscio et al [14] have used a variant of interpreted systems to verify ATLK (alternating time temporal logic [15] with knowledge) properties over the interpreted systems. They have also developed a model-checker for interpreted systems called MCMAS [8] which we will use as the model-checking engine in our implementation.

The multi-agent system formalism known as *interpreted systems (IS)* [5, 13] contains a set $\Omega = \{e, 1, \ldots, n\}$ of agents including the *environment* e with the same specification as the other agents. Interpreted systems contain the following elements:

- **Local states:** Each agent in a multi-agent framework has its own local state. The set of local states for the agent i is denoted by L_i. The local state of an agent represents the information the agent has direct access to. The environment can be seen as the agent which is capable of capturing or holding the information that is inaccessible to the other agents. For example, the communication channel in a bit transmission protocol can be modelled as the environment. The set of *global states* is $S = L_e \times L_1 \times \cdots \times L_n$, representing the system at a specific time. The system evolves as a function over the time. We also use the notation of L_i as the function that accepts a set of global states and returns the corresponding set of local states for agent i. For each $s \in S$, $l_i(s)$ denotes the local state of agent i in s.

- **Actions:** State transitions are the result of performing actions by different agents. If $i \in \Omega$, then ACT_i is the set of actions accessible for the agent i. The set of *joint actions* is defined as $ACT = ACT_e \times ACT_1 \times \cdots \times ACT_n$. We also use ACT_i as the function that accepts a joint action and returns the action of agent i.
- **Protocols:** Protocols are defined as mappings from the set of local states to the set of local actions and define the actions each agent can perform according to its local state ($P_i : L_i \rightarrow 2^{ACT_i} \backslash \{\emptyset\}, i \in \Omega$). In general, action performance is non-deterministic.

Definition 1 (Interpreted system). *Let Φ be a set of atomic propositions and $\Omega = \{e, 1, \ldots, n\}$ be a set of agents. An* interpreted system *I is a tuple:*

$$I = \langle (L_i)_{i \in \Omega}, (P_i)_{i \in \Omega}, (ACT_i)_{i \in \Omega}, S_0, \tau, \gamma \rangle$$

where (1) L_i is the set of local states of agent i, and the set of global states is defined as $S = L_e \times L_1 \times \cdots \times L_n$ (2) ACT_i is the set of actions that agent i can perform, and $ACT = ACT_e \times ACT_1 \times \cdots \times ACT_n$ is defined as the set of joint actions (3) $S_0 \subseteq S$ is the set of initial states (4) $\gamma : S \times \Phi \rightarrow \{\top, \bot\}$ is called the interpretation function *(5) $P_i : L_i \rightarrow 2^{ACT_i} \backslash \{\emptyset\}$ is the protocol for agent i (6) $\tau : ACT \times S \rightarrow S$ is called the* partial transition function *with the property that if $\tau(\alpha, s)$ is defined, then for all $i \in \Omega :\ ACT_i(\alpha) \in P_i(l_i(s))$. We also write $s_1 \xrightarrow{\alpha} s_2$ if $\tau(\alpha, s_1) = s_2$.*

Definition 2 (Reachability). *A global state $s \in S$ is* reachable *in the interpreted system I if there exists $s_0 \in S_0$, $s_1, \ldots, s_n \in S$ and $\alpha_1, \ldots, \alpha_n \in ACT$ such that for all $1 \leq i \leq n :\ s_i = \tau(\alpha_i, s_{i-1})$ and $s = s_n$. In this paper, we use G to denote the set of reachable states.*

For an interpreted system I and each agent i we define an epistemic accessibility relation on the global states as follows:

Definition 3 (Epistemic accessibility relation). *Let I be an interpreted system and i be an agent. We define the* Epistemic accessibility relation *for agent i, written $\sim_i$, on the global states of I by $s \sim_i s'$ iff $l_i(s) = l_i(s')$ and s and s' are reachable.*

3.2 CTLK logic

We specify our properties in CTLK [16]. CTL (Computational Tree Logic) is a branching-time temporal logic which has tree-like time model structure and allows quantification over paths, and CTLK adds the epistemic modality K to the CTL. CTLK is defined as follows:

Definition 4. *Let Φ be a set of atomic propositions and Ω be a set of agents. If $p \in \Phi$ and $i \in \Omega$, then CTLK formulae are defined by:*

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

The symbol E is existential path quantifier which means "there exists at least one path"'. Temporal connectives X, G and U mean "neXt state", "all future states (Globally)" and "Until"'. EX, EG and EU provide the adequate set of CTLK connectives. For instance, safety properties defined by $AG(\phi)$ (all future states (Globally)) where A is the universal path quantifier, can be written as $\neg E(\top U \neg\phi)$, or the equivalence for liveness properties $AF(\phi)$ (always for some future state) is $\neg EG(\neg\phi)$. Epistemic connective K_i means "agent i knows that".

Example 1. Consider a conference paper review system. Assume that a_1 is the author of the paper p_1. Then the safety property that says if all the papers are assigned to the reviewers and a_2 is the reviewer of p_1, then a_1 does not know the fact that a_2 is the reviewer of his paper can be defined as: $AG(\mathsf{reviewer}(\mathsf{p}_1, \mathsf{a}_2) \rightarrow \neg K_{\mathsf{a}_1}\mathsf{reviewer}(\mathsf{p}_1, \mathsf{a}_2))$.

In an student information system, the property that states no two students can be assigned as the demonstrator of each other is specified by: $AG(\neg(\mathsf{demonstratorOf}(\mathsf{a}_2, \mathsf{a}_3) \wedge \mathsf{demonstratorOf}(\mathsf{a}_3, \mathsf{a}_2)))$.

Definition 5 (Satisfaction relation). *Let I be an interpreted system, $s \in G$ where G is the set of reachable states and $p \in \Phi$ where Φ is the set of atomic propositions. For any CTLK-formula ϕ, the notation $(I, s) \models \phi$ means ϕ holds at state s in interpreted system I. The relation $\models$ is defined inductively as follows:*

$$
\begin{array}{lll}
(I,s) \models p & \Leftrightarrow & \gamma(s,p) = \top \\
(I,s) \models \neg\phi & \Leftrightarrow & (I,s) \not\models \phi \\
(I,s) \models \phi_1 \vee \phi_2 & \Leftrightarrow & (I,s) \models \phi_1 \textit{ or } (I,s) \models \phi_2 \\
(I,s) \models K_i\phi & \Leftrightarrow & (I,s') \models \phi \textit{ for all } s' \in G \textit{ such that } s \sim_i s' \\
(I,s) \models EX\phi & \Leftrightarrow & \textit{for some } s' \textit{ such that } s \xrightarrow{\alpha} s' : (I,s') \models \phi \\
(I,s) \models EG\phi & \Leftrightarrow & \textit{there exists a path } s_1 \xrightarrow{\alpha} \ldots \textit{ such that } s = s_0 \textit{ and for all}
\end{array}
$$
$$i \geq 0 : (I, s_i) \models \phi$$
$$
(I,s) \models E(\phi_1 U \phi_2) \quad \Leftrightarrow \quad \textit{there exists a path } s_1 \xrightarrow{\alpha} \ldots \textit{ such that } s = s_1, \textit{ there is}
$$
$$\textit{some } i \geq 1 \textit{ such that } (I, s_i) \models \phi_2 \textit{ and for all } j < i \textit{ we have } (I, s_j) \models \phi_1$$

We use the notation $I \models \phi$ if for all $s_0 \in S_0$: $(I, s_0) \models \phi$.

4 Policy syntax

Multi-agent access control systems grant or deny user access to the resources and services depending on the access rights defined in the policy. Access to the resources is divided into *write access*, which when granted, allows updating some system variables (in the context of this work, Boolean variables) and *read access*, that returns the value of some variables when granted. In this section, we present a simple policy syntax to define actions, permissions and evolutions. In the following section, we give semantics of the policy language by constructing an interpreted system from it.

Technical preliminaries Let V be a finite set of variables and $Pred$ a finite set of predicates. The notation $\boldsymbol{v}$ is used to specify a sequence of distinct variables. An *atomic formula* or simply an *atom* is a predicate that is applied to a sequence of variables with the appropriate length. An access control policy is a finite set of rules defined as follows:

$$\begin{array}{ll} L ::= \top \mid \bot \mid w(\boldsymbol{v}) \mid L \vee L \mid L \wedge L \mid L \rightarrow L \mid \neg L \mid \forall v\ [L] \mid \exists v\ [L] & \\ W ::= +w(\boldsymbol{v}) \mid\ -w(\boldsymbol{v}) \mid \forall v.\ W & \\ W_s ::= W \mid W_s, W & \\ A_R ::= \mathsf{id}(\boldsymbol{v}) : \{W_s\} \leftarrow L & \text{Action rule} \\ R_R ::= \mathsf{id}(\boldsymbol{v}) : w(\boldsymbol{u}) \leftarrow L & \text{Read permission rule} \end{array}$$

In the above, $w \in Pred$, and $w(\boldsymbol{v})$ is an atom. L denotes a logical formula over atoms, which is the condition for performing an action or reading information. $\{W_s\}$ is the effect of the action that include the updates. $+w(\boldsymbol{v})$ in the effect means executing the action will set the value of $w(\boldsymbol{v})$ to true and $-w(\boldsymbol{v})$ means setting the value to false. In the case of $\forall v.W$ in the effect, the action updates the signed atom in W for all possible values of v. In the case that an atom appears with different signs in multiple quantifications in the effect (for instance, $w(c,d)$ in $\forall x. + w(c,x), \forall y. - w(y,d)$), then only the sign of the last quantification is considered for the atom. id indicates the identifier of the rule.

Let $a(\boldsymbol{v}) : E \leftarrow L$ be an action rule. The *free variables* of the logical formula L are denoted by $\mathbf{fv}(L)$ and are defined in the standard way. We also define $\mathbf{fv}(E) = \bigcup_{e \in E} \mathbf{fv}(e)$ where $\mathbf{fv}(\pm w(\boldsymbol{x})) = \boldsymbol{x}$ and $\mathbf{fv}(\forall x.W)$=$\mathbf{fv}(W) \backslash x$. We stipulate: $\mathbf{fv}(E) \cup \mathbf{fv}(L) \subseteq \boldsymbol{v}$. If $r(\boldsymbol{v}) : w(\boldsymbol{u}) \leftarrow L$ is a read rule, then $\mathbf{fv}(\boldsymbol{u}) \cup \mathbf{fv}(L) \subseteq \boldsymbol{v}$.

Let Σ be a finite set objects. A *ground atom* is a variable-free atom; i.e. atoms with the variables substituted with the objects in Σ. For instance, if reviewer$\in Pred$ and Bob,Paper$\in \Sigma$, then reviewer(Bob,Paper) is a ground atom. In the context of this paper, we call the ground atoms as (atomic) *propositions*, since they only evaluate to true and false.

An *action* $\alpha : \varepsilon \leftarrow \ell$ contains an identifier α together with the *evolution rule* $\varepsilon \leftarrow \ell$, which is constructed by instantiating all the arguments in an action rule $a(\boldsymbol{v}) : E \leftarrow L$ with the objects in Σ. We refer to the whole action by its identifier α. In an asynchronous multi-agent system, it is crucial to know the agent that performs an action. As the convention and for the rest of this article, we consider the first argument of the action to be the agent performing that action. Therefore, in the action assignReviewer(Alice,Bob,Paper), Alice is the one that assigns Bob as the reviewer of Paper. If α is an action, then $\mathbf{Ag}(\alpha)$ denotes the agent that performs α.

A *read permission* $\rho : p \leftarrow \ell$ is constructed by substituting the arguments in read permission rule $r(\boldsymbol{v}) : w(\boldsymbol{u}) \leftarrow L$ with the objects in Σ. ρ is the identifier, p is the proposition and ℓ is the condition for reading p. As for the actions, we assume the first argument in ρ to be the agent that reads the proposition p, which is denoted by $\mathbf{Ag}(\rho)$.

Definition 6 (Policy). *An* access control policy *is a finite set of actions and read permissions derived by instantiating a set of rules with a finite set of objects.*

5 Building an interpreted system from a policy

In access control systems, we deal with read and write access procedures. Write procedures, which update a set of variables, are contained in interpreted systems as actions. In interpreted systems, a principal knows a fact if it is included in his local state or he can deduce it by applying logical reasoning. In access control systems and in addition to the local information, agents may obtain permission to directly access some resources in the system. This permission may be granted by the system or other agents (delegation of authority). For instance, in a web application users always have access to their own profile, but they cannot access other users' profile unless the permission is granted by the owners. When a read permission to a resource is granted, the resource will become a part of agent's local state. When the permission is denied, it will be removed from agent's directly accessible information. This behaviour is similar to a system which uses dynamically changing local states to model permissions.

Interpreted systems formally contain local states which cannot change during execution of the system. In order to model temporary read permissions, we need to introduce some locally accessible information, which simulates the temporary read access. In this section, we explain how to introduce temporary read permissions when modelling access control systems. Moreover, we model access control systems in asynchronous manner using interpreted systems framework. An interpreted system is *asynchronous* if all joint actions contain at most one non-Λ agent action where Λ denotes no-operation.

Given a policy, we build an access control system based on interpreted systems framework by considering the requirements above. Incorporating temporary read permissions requires introducing some information into the local states. We say the proposition p is local to the agent i if its value only depends on the local state of i. In the other words, for all $s, s' \in S$ where $s \sim_i s'$ we have $\gamma(s,p) = \gamma(s',p)$.

Definition 7 (Local interpretation). *Let L_i be the set of local states of agent i in interpreted system I and Φ_i be the set of local propositions. We define the* local interpretation *for agent i as a function $\gamma_i : L_i \times \Phi_i \to \{\top, \bot\}$ such that $\gamma_i(l,p) = \gamma(s,p)$ where $l_i(s) = l$ for some global state s. We require the set of local propositions to be pairwise disjoint.*

The following lemma provides the theoretical background of modelling knowledge by readability in an interpreted system.

Lemma 1. *Let I be an interpreted system, G the set of reachable states, i an agent, Φ the set of propositions and $p \in \Phi$. Suppose that $p', p'' \in \Phi_i$. If for all $s \in G$:*

$$\textit{if } \gamma_i(l_i(s),p'') = \top \textit{ then } (I,s) \models p \;\Leftrightarrow\; \gamma_i(l_i(s),p') = \top \tag{1}$$

Then we have:

$$\gamma_i(l_i(s),p'') = \top \quad \Rightarrow \quad (I,s) \models K_i p \vee K_i \neg p$$

Proof. We first prove that

$$\gamma_i(l_i(s),p'') = \top \text{ and } (I,s) \models p \quad \Rightarrow \quad (I,s) \models K_i p \tag{2}$$

Let us assume that $\gamma_i(l_i(s), p'') = \top$ and $(I, s) \models p$. By (1) we have $\gamma_i(l_i(s), p') = \top$. Consider any state $s_1 \in G$ such that $s_1 \sim_i s$. By the definition of $\sim_i$, we have $l_i(s_1) = l_i(s)$. Therefore, $\gamma_i(l_i(s_1), p') = \top$ and $\gamma_i(l_i(s_1), p'') = \top$ which implies $(I, s_1) \models p$. Hence, by the definition of K_i we are able to conclude that $(I, s) \models K_i p$. The proof for the second case:

$$\gamma_i(l_i(s), p'') = \top \text{ and } (I, s) \models \neg p \Rightarrow (I, s) \models K_i \neg p \tag{3}$$

is similar to the first proof. Therefore, by (2) and (3) we have $\gamma_i(l_i(s), p'') = \top \Rightarrow (I, s) \models K_i p \vee K_i \neg p$.

To model knowledge by readability, we incorporate all the atomic propositions that appear in the policy into the environment. We call those propositions *policy propositions*. Now for each policy proposition p and for each agent, we introduce two local atomic propositions: p_{read} (p'' in Lemma 1) as the read permission of proposition p, and p_{loc} (p' in Lemma 1) as the local copy of p. We modify the transition function in order to satisfy the following property: for all reachable states, if p_{read} is true (agent has read access to p) in a state, then p_{loc} is assigned the same value as p. This property guarantees agent's knowledge of proposition p whenever his access to p is granted.

Building the interpreted system Given a policy $\mathcal{C}$ with Σ_{Ag} as the set of agents, we build up an interpreted system that models the access control system in the following way:

Let $\Phi_{\mathcal{C}}$ be the set of propositions that appear in $\mathcal{C}$ (policy propositions), and $\mathcal{A}_{\mathcal{C}}$ and $\mathcal{R}_{\mathcal{C}}$ the set of actions and read permissions in $\mathcal{C}$ respectively. For an interpreted system that corresponds to the policy $\mathcal{C}$, the knowledge gained by reading system information need to be incorporated into the local states of the agents.

Procedure 1 adopts Lemma 1 which describes a method to model temporary read permissions. The function INCKNOWLEDGE in procedure 1 accepts $\mathcal{A}_{\mathcal{C}}$, $\mathcal{R}_{\mathcal{C}}$, $\Phi_{\mathcal{C}}$ and Σ_{Ag} as the input. For each agent i in Σ_{Ag}, Procedure 1 generates a set of local propositions Φ_i. The local state of agent i consists of all valuations of Φ_i. For each proposition $p \in \Phi_{\mathcal{C}}$, the set Φ_i contains two propositions $p_{\mathit{loc}}, p_{\mathit{read}}$ where p_{loc} is the copy of p and gets updated whenever p_{read} as the access permission for p is true (refer to Lemma 1 for the details). The procedure modifies the actions and corresponding evolutions in $\mathcal{A}_{\mathcal{C}}$ into the set $\mathcal{A}^u_{\mathcal{C}}$ in order to update the propositions in Φ_i in the appropriate way. For each action and for each agent, if p appears in the effect (if-conditions in lines 12 and 18), then the action will replace with two freshly created actions: one sets p_{read} to true and p_{loc} to the same value as p if the read permission of p evaluates to true in the next state (lines 13 and 19). Otherwise (read permission of p evaluates to false in the next state), p_{read} will set to false and p_{loc} remains unchanged (lines 15 and 21). If p does not appear in the effect (line 24), p_{loc} and p_{read} will only get updated whenever the read permission of p is affected by the action.

Calculating the symbolic transition function: We provide the details for calculating the symbolic transition function we use for traversing over a path in our system. The symbolic transition function accepts a set of states as input and returns the result of performing an action over the states of that set.

Procedure 1 Incorporating read permissions into evolution rules

```
function $\textsc{incKnowledge}(\mathcal{A}_\mathcal{C}, \mathcal{R}_\mathcal{C}, \Phi_\mathcal{C}, \Sigma_{Ag})$
    ▷ Input: $\mathcal{A}_\mathcal{C}$ is the set of actions, $\mathcal{R}_\mathcal{C}$ is the set of read permissions, $\Phi_\mathcal{C}$ the set of policy propositions and $\Sigma_{Ag}$ the set of agents
    ▷ Output: returns the updated set of actions and the set of local propositions
    $\mathcal{A}^u_\mathcal{C} := \mathcal{A}_\mathcal{C}$
    for all $i \in \Sigma_{Ag}$ do
        $\Phi_i := \emptyset$
        for all $p \in \Phi_\mathcal{C}$ do
            determine $r : p \leftarrow \ell_r \in \mathcal{R}_\mathcal{C}$ where $\mathbf{Ag}(r) = i$
            $\Phi_i := \Phi_i \cup \{p_{loc}, p_{read}\}$
            $\hat{\mathcal{A}}^u_\mathcal{C} := \emptyset$
            for all $\alpha : \varepsilon \leftarrow \ell \in \mathcal{A}^u_\mathcal{C}$ do
                if $+p \in \varepsilon$ then
                    construct $\alpha_1 : \varepsilon \cup \{+p_{loc}, +p_{read}\} \leftarrow$
                        $\ell \wedge (\ell_r[\top/v \mid +v \in \varepsilon][\bot/v' \mid -v' \in \varepsilon])$ where $\mathbf{Ag}(\alpha_1) = \mathbf{Ag}(\alpha)$
                    construct $\alpha_2 : \varepsilon \cup \{-p_{read}\} \leftarrow$
                        $\ell \wedge \neg(\ell_r[\top/v \mid +v \in \varepsilon][\bot/v' \mid -v' \in \varepsilon])$ where $\mathbf{Ag}(\alpha_2) = \mathbf{Ag}(\alpha)$
                    $\hat{\mathcal{A}}^u_\mathcal{C} := \hat{\mathcal{A}}^u_\mathcal{C} \cup \{\alpha_1, \alpha_2\}$
                else if $-p \in \varepsilon$ then
                    construct $\alpha_1 : \varepsilon \cup \{-p_{loc}, +p_{read}\} \leftarrow$
                        $\ell \wedge (\ell_r[\top/v \mid +v \in \varepsilon][\bot/v' \mid -v' \in \varepsilon])$ where $\mathbf{Ag}(\alpha_1) = \mathbf{Ag}(\alpha)$
                    construct $\alpha_2 : \varepsilon \cup \{-p_{read}\} \leftarrow$
                        $\ell \wedge \neg(\ell_r[\top/v \mid +v \in \varepsilon][\bot/v' \mid -v' \in \varepsilon])$ where $\mathbf{Ag}(\alpha_2) = \mathbf{Ag}(\alpha)$
                    $\hat{\mathcal{A}}^u_\mathcal{C} := \hat{\mathcal{A}}^u_\mathcal{C} \cup \{\alpha_1, \alpha_2\}$
                else
                    if for all $q \in \mathbf{fv}(\ell_r) : +q \notin \varepsilon$ and $-q \notin \varepsilon$ then
                        $\hat{\mathcal{A}}^u_\mathcal{C} := \hat{\mathcal{A}}^u_\mathcal{C} \cup \{\alpha\}$
                    else
                        construct $\alpha_1 : \varepsilon \cup \{+p_{loc}, +p_{read}\} \leftarrow \ell \wedge$
                            $(\ell_r[\top/v \mid +v \in \varepsilon][\bot/v' \mid -v' \in \varepsilon]) \wedge p$ where $\mathbf{Ag}(\alpha_1) = \mathbf{Ag}(\alpha)$
                        construct $\alpha_2 : \varepsilon \cup \{-p_{loc}, +p_{read}\} \leftarrow \ell \wedge$
                            $(\ell_r[\top/v \mid +v \in \varepsilon][\bot/v' \mid -v' \in \varepsilon]) \wedge \neg p$ where $\mathbf{Ag}(\alpha_2) = \mathbf{Ag}(\alpha)$
                        construct $\alpha_3 : \varepsilon \cup \{-p_{read}\} \leftarrow \ell \wedge$
                            $\neg(\ell_r[\top/v \mid +v \in \varepsilon][\bot/v' \mid -v' \in \varepsilon])$ where $\mathbf{Ag}(\alpha_3) = \mathbf{Ag}(\alpha)$
                        $\hat{\mathcal{A}}^u_\mathcal{C} := \hat{\mathcal{A}}^u_\mathcal{C} \cup \{\alpha_1, \alpha_2, \alpha_3\}$
                    end if
                end if
            end for
            $\mathcal{A}^u_\mathcal{C} := \hat{\mathcal{A}}^u_\mathcal{C}$
        end for
    end for
    return $\{\Phi_i \mid i \in \Sigma_{Ag}\}, \mathcal{A}^u_\mathcal{C}$
end function
```

As a convention, we use $s[p \mapsto m]$ where $s \in S$ to denote the state that is like s except that it maps the proposition p to the value m. Let $st \subseteq S$ be a set of states. When performing the action $\alpha : \varepsilon \leftarrow \ell$ in the states of st, the transition is only performed in the states that satisfy the permission ℓ. In the resulting states, the propositions that do not appear in ε remain the same as in the states that the transition begins. Therefore, we define:

$$\Theta_\alpha(st) = \Big\{ s[p \mapsto \top \mid +p \in \varepsilon][p \mapsto \bot \mid -p \in \varepsilon] \;\Big|\; s \in st, (I, s) \models \ell \Big\}$$

Definition 8 (Derived interpreted system). *Let $\mathcal{C}$ be a policy with Σ_{Ag} as the set of agents, $\Phi_\mathcal{C}$ the set of policy propositions, and $\mathcal{A}^u_\mathcal{C}$ and Φ_i, $i \in \Sigma_{Ag}$ derived from procedure 1. Let $\Omega = \{e\} \cup \Sigma_{Ag}$ and $\Phi = \bigcup_{i \in \Omega} \Phi_i$ where $\Phi_e = \Phi_\mathcal{C}$. Then the interpreted system derived from policy $\mathcal{C}$ is:*

$$I_\mathcal{C} = \langle (L_i)_{i \in \Omega}, (P_i)_{i \in \Omega}, (ACT_i)_{i \in \Omega}, S_0, \tau, \gamma \rangle$$

where

1. *L_i is the set of local states of agent i, where each local state is a valuation of the propositions in Φ_i. The set of global states is defined as $S = L_e \times L_1 \times \cdots \times L_n$*
2. *$ACT_i = \{\alpha \in \mathcal{A}^u_\mathcal{C} \mid \boldsymbol{Ag}(\alpha) = i\} \cup \{\Lambda\}$ where Λ denotes* no operation*, and a joint action is a $|\Omega|$-tuple such that at most one of the elements is non-Λ (asynchronous interpreted system). For simplicity,* we denote a joint action with its non-Λ element
3. *$S_0 \subseteq S$ is the set of initial states*
4. *γ is the interpretation function over S and Φ. If $p \in \Phi_i$ then we have $\gamma(s, p) = \gamma_i(l_i(s), p)$*
5. *P_i is the protocol for agent i where for all $l \in L_i$: $P_i(l) = ACT_i$*
6. *τ is the transition function that is defined as follows: if α is a joint action (or simply, an action) and $s \in S$, then $\tau(\alpha, s) = s'$ if $\Theta_\alpha(\{s\}) = \{s'\}$.*

The system that we derive from policy $\mathcal{C}$ is a special case of interpreted systems where the local states are the valuation of local propositions that are generated by the procedure INCKNOWLEDGE.

6 Abstraction technique

In an interpreted system, the state space exponentially increases when extra propositions are added into the system. Considering a fragment of CTLK properties known as ACTLK as the specification language, we are able to verify the properties over an over-approximated abstract model instead of the concrete one. ACTLK is defined as follows:

Definition 9. *Let Φ be the set of atomic propositions and Ω set of agents. If $p \in \Phi$ and $i \in \Omega$, then ACTLK formulae are defined by:*

$$\phi ::= p \mid \neg p \mid \phi \wedge \phi \mid \phi \vee \phi \mid K_i\phi \mid AX\phi \mid A(\phi U \phi) \mid A(\phi R \phi)$$

where the symbol A is universal path quantifier which means "for all the paths".

To provide a relation between the concrete model and the abstract one, we extend the *simulation relation* introduced in [17] to cover the epistemic relation between states. Using the abstraction technique that preserves simulation relation between the concrete model and the abstract one, we are able to verify ACTLK specification formulas over the model. In this paper and for abstraction and refinement, we focus on safety properties expressed in ACLK. The advantages of safety properties are first, they are capable of expressing *policy invariants*, and second, the generated counterexample contains finite sequence of actions (or transitions). We can extend the abstraction refinement method to the full ACTLK by unfolding the loops in the counterexamples into finite transitions as described in [6], which is outside the scope of this paper.

6.1 Existential abstraction

The general framework of existential abstraction was first introduced by Clark et. al in [17]. Existential abstraction partitions the states of a model into clusters, or equivalence classes. The clusters form the states of the abstract model. The transitions between the clusters in the abstract model give rise to an over-approximation of the original (or concrete) model that *simulates* the original one. So, when a specification in ACTL (or in the context of this paper, ACTLK) logic is true in the over-approximated model, it will be true in the concrete one. Otherwise, a counterexample will be generated which needs to be verified over the concrete model.

Notation 1 *For simplicity, we use the same notation ($\sim_i$) for the epistemic accessibility relation in both the concrete and abstract interpreted systems.*

Definition 10 (Simulation). *Let I and $\widetilde{I}$ be two interpreted systems, Ω be the set of agents in both systems, and Φ and $\widetilde{\Phi}$ the corresponding set of propositions where $\widetilde{\Phi} \subseteq \Phi$. The relation $H \subseteq S \times \widetilde{S}$ is* simulation relation *between I and $\widetilde{I}$ if and only if:*

1. *For all $s_0 \in S_0$, there exists $\widetilde{s}_0 \in \widetilde{S_0}$ st. $(s_0, \widetilde{s}_0) \in H$.*

   and for all $(s, \widetilde{s}) \in H$:

2. *For all $p \in \widetilde{\Phi}$: $\gamma(s,p) = \widetilde{\gamma}(\widetilde{s},p)$*
3. *For each state $s' \in S$ such that $\tau(s,\alpha) = s'$ for some $\alpha \in ACT$, there exists $\widetilde{s}' \in \widetilde{S}$ and $\widetilde{\alpha} \in \widetilde{ACT}$ such that $\widetilde{\tau}(\widetilde{s},\widetilde{\alpha}) = \widetilde{s}'$ and $(s', \widetilde{s}') \in H$.*
4. *For each state $s' \in S$ such that $s \sim_i s'$, there exists $\widetilde{s}' \in \widetilde{S}$ such that $\widetilde{s} \sim_i \widetilde{s}'$ and $(s', \widetilde{s}') \in H$.*

The above definition for simulation relation over the interpreted systems is similar to the one for Kripke model [6], except that the relation for the epistemic relation is introduced. If such simulation relation exists, we say that $\widetilde{I}$ *simulates* I (denoted by $I \preceq \widetilde{I}$).

If H is a function, that is, for each $s \in S$ there is a unique $\widetilde{s} \in \widetilde{S}$ such that $(s, \widetilde{s}) \in H$, we write $h(s) = \widetilde{s}$ instead of $(s, \widetilde{s}) \in H$.

Lemma 2. *Let $I \preceq \widetilde{I}$, $s_1 \in S$, $\widetilde{s}_1 \in \widetilde{S}$ and $(s_1, \widetilde{s}_1) \in H$ where H is the simulation relation between I and $\widetilde{I}$. Then for each path $s_1 \xrightarrow{\alpha_2} \dots$ in I, there exists a path $\widetilde{s}_1 \xrightarrow{\widetilde{\alpha}_2} \dots$ in $\widetilde{I}$ such that for all $i \geq 1$, $(s_i, \widetilde{s}_i) \in H$ holds.*

Proof. The proof is trivial by item 3 in definition 10 and induction over the state transitions.

Proposition 1. *For every ACTLK formula φ over propositions $\widetilde{\Phi}$, if $I \preceq \widetilde{I}$ and $\widetilde{I} \models \varphi$, then $I \models \varphi$.*

Proof. To prove the proposition, we first prove if $I \preceq \widetilde{I}$ and H is the simulation relation, then for all $\widetilde{s} \in \widetilde{S}$ and $s \in S$ where $(s, \widetilde{s}) \in H$, $(\widetilde{I}, \widetilde{s}) \models \varphi$ implies $(I, s) \models \varphi$. We assume φ is in NNF. The proof proceeds by induction over the structure of φ. Let $s \in S$, $\widetilde{s} \in \widetilde{S}$ and $(s, \widetilde{s}) \in H$.

- If $(\widetilde{I}, \widetilde{s}) \models p$ where p an atomic formula, then $\gamma(\widetilde{s}, p) = \top$. By item 2 in definition 10 we have $\gamma(s, p) = \top$ which implies $(I, s) \models p$. The case is similar for $\varphi = \neg p$.
- If $(\widetilde{I}, \widetilde{s}) \models \varphi_1 \wedge \varphi_2$, then $(\widetilde{I}, \widetilde{s}) \models \varphi_1$ and $(\widetilde{I}, \widetilde{s}) \models \varphi_2$. By induction hypothesis we have $(I, s) \models \varphi_1$ and $(I, s) \models \varphi_2$. Therefore, $(I, s) \models \varphi_1 \wedge \varphi_2$. The case is similar for $\varphi = \varphi_1 \vee \varphi_2$.
- Assume $(\widetilde{I}, \widetilde{s}) \models AX\varphi_1$. If $s \xrightarrow{\alpha} s'$ is a path in I, then by Lemma 2 there exists a path $\widetilde{s} \xrightarrow{\widetilde{\alpha}} \widetilde{s}'$ in $\widetilde{I}$ where $(s', \widetilde{s}') \in H$. By the assumption we have $(\widetilde{I}, \widetilde{s}') \models \varphi_1$. Then the induction hypothesis implies $(I, s') \models \varphi_1$. Thus we can conclude that $(I, s) \models AX\varphi_1$.
- Assume $(\widetilde{I}, \widetilde{s}) \models A(\varphi_1 U \varphi_2)$. Let $s_1 \xrightarrow{\alpha_2} \dots$ be a path in I where $s_1 = s$ and $\widetilde{s}_1 \xrightarrow{\widetilde{\alpha}_2} \dots$ the corresponding path in $\widetilde{I}$ where $\widetilde{s}_1 = \widetilde{s}$. By the assumption, there exists some $i \geq 1$ where $(\widetilde{I}, \widetilde{s}_i) \models \varphi_2$ and $(\widetilde{I}, \widetilde{s}_i) \models \varphi_1$ for all $j < i$. By induction hypothesis and Lemma 2, $(I, s) \models \varphi_1 U \varphi_2$. As this property holds for all the path starting at s, we can conclude $(I, s) \models A(\varphi_1 U \varphi_2)$.
- Assume $(\widetilde{I}, \widetilde{s}) \models A(\varphi_1 R \varphi_2)$. The proof is similar to the case for $(\widetilde{I}, \widetilde{s}) \models A(\varphi_1 U \varphi_2)$.
- Assume $(\widetilde{I}, \widetilde{s}) \models K_i\varphi$. We pick a state $s' \in S$ where $s' \sim_i s$. By item 4 in definition 10, there exists $\widetilde{s}' \in \widetilde{S}$ where $\widetilde{s}' \sim_i \widetilde{s}$ and $(s', \widetilde{s}') \in H$. By the assumption, $(\widetilde{I}, \widetilde{s}') \models \varphi$. Induction hypothesis implies that $(I, s') \models \varphi$. As this property holds for all the states with accessibility relation $\sim_i$ to s, we have $(I, s') \models K_i\varphi$.

Now, if $\widetilde{I} \models \varphi$ or in the other words, for all $\widetilde{s}_0 \in \widetilde{S}_0$: $(\widetilde{I}, \widetilde{s}) \models \varphi$, then by item 1 in definition 10 and the above proof we have for all $s_0 \in S_0$: $(I, s) \models \varphi$ or equivalently $I \models \varphi$.

6.2 Variable hiding abstraction

Variable hiding is a popular technique in the category of existential abstraction. In our methodology, we consider factorizing the concrete state space into equivalence classes that act as abstract states by abstracting away a set of system propositions. In our approach, the states in each equivalence class are only different in the valuation of the hidden propositions. Also the transitions between the states of the abstract model are defined in such a way that the abstract model simulates the concrete one. Our refinement procedure will be splitting the abstract states by putting back some of the atomic proportions that were hidden in the abstract model. We refine the model by analysing the counterexample generated when verifying safety properties described in ACTLK logic. The model checker will output a counterexample if the property does not hold.

Definition 11. *(Local state relation) Let $I_{\mathcal{C}}$ be an interpreted system derived from policy $\mathcal{C}$, L_i and Φ_i be the set of local states and local propositions for the agent i, and $\widetilde{\Phi}_i \subseteq \Phi_i$. The local relation $\Re_i$ is defined as:*

$$\textit{for all } l_1, l_2 \in L_i : \qquad l_1 \Re_i l_2 \quad \textit{iff} \quad \textit{for all } p \in \widetilde{\Phi}_i : \gamma_i(l_1, p) = \gamma_i(l_2, p)$$

where γ_i is the local interpretation for the agent i. The function $h_i : L_i \rightarrow L_i/\Re_i$ is the surjection which maps elements of L_i into equivalence classes of $\Re_i$.

Definition 12 (Action classification). *Let $\alpha : \varepsilon \leftarrow \ell \in ACT$ and $\widetilde{\Phi} \subseteq \Phi$. We define $\alpha' : \varepsilon' \leftarrow \ell' \in [\alpha]$ iff $\{\pm p \in \varepsilon' \mid p \in \widetilde{\Phi}\} = \{\pm p \in \varepsilon \mid p \in \widetilde{\Phi}\}$, $\exists(\Phi \backslash \widetilde{\Phi}).\ell' \equiv \exists(\Phi \backslash \widetilde{\Phi}).\ell$ and $\boldsymbol{Ag}(\alpha') = \boldsymbol{Ag}(\alpha)$.*

In the above definition, the infix notation $\equiv$ denotes the semantically equivalence relation. Formally $\exists x.f$ for a Boolean function f is defined as $f[0/x] \vee f[1/x]$ which means f could be made to true by putting x to 0 or to 1. If $X = \{x_1, \ldots, x_n\}$, then $\exists X.f = \exists x_1 \ldots \exists x_n.f$.

Definition 13 (Abstract interpreted system). *Given a policy $\mathcal{C}$, let Ω, Φ and $\mathcal{A}^u_{\mathcal{C}}$ be deduced as described in section 5 and $I_{\mathcal{C}}$ be the derived interpreted system. Let $\widetilde{\Phi} \subseteq \Phi$ and $\widetilde{\Omega} = \Omega$. We define Interpreted system $\widetilde{I}_{\mathcal{C}}$ as:*

$$\widetilde{I}_{\mathcal{C}} = \langle (\widetilde{L}_i)_{i \in \widetilde{\Omega}}, (\widetilde{P}_i)_{i \in \widetilde{\Omega}}, (\widetilde{ACT}_i)_{i \in \widetilde{\Omega}}, \widetilde{S}_0, \widetilde{\tau}, \widetilde{\gamma} \rangle$$

where

1. *$\widetilde{L}_i = L_i/\Re_i$ where $\Re_i$ is defined in definition 11 over L_i, and $\widetilde{S} = \widetilde{L}_e \times \widetilde{L}_1 \times \cdots \times \widetilde{L}_n$*
2. *$\widetilde{ACT}_i = \{[\alpha] \mid \alpha \in \mathcal{A}^u_{\mathcal{C}}$ and $\boldsymbol{Ag}(\alpha) = i\}$ and a joint action is a $|\widetilde{\Omega}|$-tuple such that at most one of the elements is non-Λ - i.e. the system is asynchronous. As before, each joint action is shown by its non-Λ element. If $\widetilde{\alpha} = [\alpha]$, then the evolution rule for $\widetilde{\alpha}$ is $\widetilde{\varepsilon} \leftarrow \widetilde{\ell}$ where $\widetilde{\varepsilon} = \{\pm p \in \varepsilon \mid p \in \widetilde{\Phi}\}$ and $\widetilde{\ell} = \exists(\Phi \backslash \widetilde{\Phi}).\ell$*
3. *$\widetilde{S}_0 = \{(h_i(l_i(s)))_{i \in \widetilde{\Omega}} \mid s \in S_0\}$ where h_i as in definition 11 maps the elements of L_i to $\widetilde{L}_i$*
4. *For all $\widetilde{l} \in \widetilde{L}_i$ and for all $p \in \widetilde{\Phi}_i$ we have $\widetilde{\gamma}_i(\widetilde{l}, p) = \gamma_i(l, p)$ where $\widetilde{l} = h_i(l)$*
5. *$\widetilde{P}_i$ is the protocol for agent i where for all $\widetilde{l} \in \widetilde{L}_i$: $\widetilde{P}_i(\widetilde{l}) = \widetilde{ACT}_i$*
6. *$\widetilde{\tau}$ is the transition function defined as follows: If $\widetilde{\alpha}$ is a joint action, $\widetilde{s} \in \widetilde{S}$ and $\widetilde{\Theta}_{\widetilde{\alpha}}$ is the symbolic transition function for interpreted system $\widetilde{I}_{\mathcal{C}}$ and action $\widetilde{\alpha}$, then $\widetilde{\tau}(\widetilde{\alpha}, \widetilde{s}) = \widetilde{s}'$ if $\widetilde{\Theta}_{\widetilde{\alpha}}(\{\widetilde{s}\}) = \{\widetilde{s}'\}$*

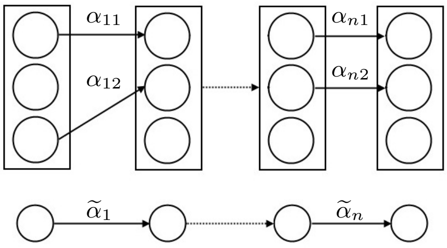

Fig. 1. The counterexample provided by the abstract model may not be valid on the concrete one. The labels represent the actions that result in the transitions.

Proposition 2. *If $I_{\mathcal{C}}$ is the interpreted system derived from policy $\mathcal{C}$ and $\widetilde{I}_{\mathcal{C}}$ is defined as in definition 13, then $I_{\mathcal{C}} \preceq \widetilde{I}_{\mathcal{C}}$.*

Proof. Let $h : S \to \widetilde{S}$ be a function where $h(s) = (h_i(l_i(s)))_{i \in \widetilde{\Omega}}$ and h_i is defined as in definition 11. We show that $\widetilde{I}_{\mathcal{C}}$ simulates $I_{\mathcal{C}}$ under h. Item 1 in definition 10 trivially holds by property (3). Item 2 holds by property (4) and the fact that if $p \in \widetilde{\Phi}$, then there is an agent i where $p \in \widetilde{\Phi}_i$ and we have $\widetilde{\gamma}(\widetilde{s}, p) = \widetilde{\gamma}_i(\widetilde{l}_i(\widetilde{s}), p)$.

Now assume that $h(s) = \widetilde{s}$ and $\tau(\alpha, s) = s'$, which is equivalent to $\Theta_\alpha(\{s\}) = \{s'\}$. If $\alpha : \varepsilon \leftarrow \ell$ can be performed in s, then we have $(I, s) \models \ell$. It is trivial to show that $(I, s) \models \exists(\Phi \backslash \widetilde{\Phi}).\ell$ using structural induction. Since the formula $\exists(\Phi \backslash \widetilde{\Phi}).\ell$ only contains the propositions in $\widetilde{\Phi}$, then by item 2 in definition 10 we have $(\widetilde{I}, \widetilde{s}) \models \exists(\Phi \backslash \widetilde{\Phi}).\ell$. Let $\widetilde{\alpha} = [\alpha]$. By definition 12, $\widetilde{\alpha}$ can be performed in $\widetilde{s}$. From $\widetilde{\varepsilon} \subseteq \varepsilon$ we infer that the performance of $\widetilde{\alpha}$ on $\widetilde{s}$ results in a state $\widetilde{s}'$ where all the propositions in $\widetilde{\Phi}$ have the same value in $\widetilde{s}'$ as in s'. Hence, $h(s') = \widetilde{s}'$ as required for item 3 in definition 10.

Let us assume that $h(s) = \widetilde{s}$ and $s \sim_i s'$. Therefore $l_i(s) = l_i(s')$ which means that for all $p \in \Phi_i : \ \gamma(s, p) = \gamma(s', p)$. Since $\widetilde{\Phi} \subseteq \Phi$, then $\widetilde{\Phi}_i \subseteq \Phi_i$. By item 2 in definition 10, for all $p \in \widetilde{\Phi}_i : \ \gamma(s, p) = \widetilde{\gamma}(\widetilde{s}, p)$. Let us assume that $h(s') = \widetilde{s}'$. Then for all $p \in \widetilde{\Phi}_i : \ \gamma(s', p) = \widetilde{\gamma}(\widetilde{s}', p)$. Hence we have for all $p \in \widetilde{\Phi}_i : \ \widetilde{\gamma}(\widetilde{s}, p) = \widetilde{\gamma}(\widetilde{s}', p)$. Therefore $\widetilde{s} \sim_i \widetilde{s}'$ as required for item 4.

Definition 14. *We define $h_A : ACT \to \widetilde{ACT}$ as the surjection that maps the actions in the concrete model to the actions in the abstract one.*

Given a policy, by using Proposition 2 we can build up an abstract access control system by hiding a set of propositions and abstracting the evolution rules. Now by proposition 1, it is possible to verify ACTLK properties over the abstract model, and refine the abstraction, if the property does not hold and the counterexample is found to be spurious.

7 Automated refinement

Our counterexample based abstraction refinement method consists of three steps:

- *Generating the initial abstraction*: It is done by examining transition blocks corresponding to the variables and constructing clusters of variables which interfere with each other via transition conditions. In our approach, we build the simplest possible initial abstract model by only retaining only the propositions appear in specification φ that we aim to verify.
- *Model-checking the abstract structure*: Model-checking will be performed on the abstract model for a specification φ. If the abstract model satisfies φ, then it can be concluded that the concrete model also satisfies φ. If the abstract model checking generates a counterexample, it should be checked if the counterexample is an actual counterexample for the concrete model. If it is a spurious counterexample in the concrete model as in figure 1, the abstract system should be refined by proceeding to the next step.
- *Refining the abstraction*: The counterexample guided framework refines the abstract model by partitioning the states in abstract model in such a way that the refined model does not admit the same counterexample. For the refinement, we turn some of the invisible variables into visible. After refinement of the abstract model, step 2 will be proceeded.

The process of abstraction and refinement will eventually terminate, as in the worst case, the refined model becomes the same as the concrete one, which is a finite state model. Therefore in the worst case, the verification will turn into the verification of the concretised model.

7.1 Generating the initial abstraction

For automatic abstraction refinement, we build the initial model as simple as possible. For an ACTLK formula φ, we keep all the atomic propositions that appear in φ visible in the abstract model and hide the rest. The abstract model is built up by definition 13.

7.2 Validation of counterexamples

The structure of a counterexample created by the verification of an ACTLK formula is different from the counterexample generated in the absence of knowledge modality. In an ACTLK counterexample, we have epistemic relations as well as temporal ones. Analysis of such counterexamples is more complicated than the counterexamples for temporal properties.

A counterexample for a safety property in ACTLK is a loop-free tree-like graph with states as vertices, and temporal and epistemic transitions as edges. Every counterexample has an initial state as the root. A temporal transition in the graph is labelled with its corresponding action and epistemic transition is labelled with the corresponding epistemic relation. We define a *temporal path* as a path that contains only temporal transitions. An *epistemic path* contains at least one epistemic transition. Every state in the counterexample is *reachable from an initial state* in the model, which may differ from the root. For any state s, we write also s for the empty path which starts and finishes in s.

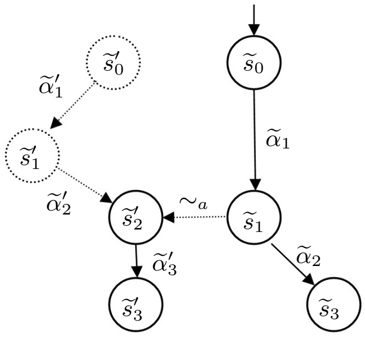

Fig. 2. A tree-like counterexample generated by the verification of an ACTLK safety property over the abstract model. In the diagram, $\widetilde{s}_0, \widetilde{s}'_0 \in S_0$ and $\widetilde{s}_1 \sim_a \widetilde{s}'_2$. As reachability is a requirement for $\widetilde{s}_1 \sim_a \widetilde{s}'_2$ and $\widetilde{s}_1$ is already reachable, the temporal path $\widetilde{s}'_0 \xrightarrow{\widetilde{\alpha}'_1} \widetilde{s}'_1 \xrightarrow{\widetilde{\alpha}'_2} \widetilde{s}'_2$ provides the witness for the reachability of $\widetilde{s}'_2$. Considering this witness is required in counterexample checking.

Counterexample formalism: A tree is a finite set of temporal and epistemic paths with an initial state as the root. Each path begins from the root and finishes at a leaf. For an epistemic transition over a path, we use the same notation as the epistemic relation while we consider the transition to be from left to the right. For instance, the tree in the figure 2 is formally presented by:

$$\{\widetilde{s}_0 \xrightarrow{\widetilde{\alpha}_1} \widetilde{s}_1 \xrightarrow{\widetilde{\alpha}_2} \widetilde{s}_3,\ \widetilde{s}_0 \xrightarrow{\widetilde{\alpha}_1} \widetilde{s}_1 \sim_a \widetilde{s}'_2 \xrightarrow{\widetilde{\alpha}'_3} \widetilde{s}'_3\}$$

To verify a tree-like counterexample, we traverse the tree in a *depth-first* manner. An abstract counterexample is valid in the concrete model if a real counterexample in the concrete model corresponds to it.

We use the notation $s \rightarrow s'$ when the type of the transition from s to s' is not known.

Definition 15 (Vertices, root). *Let $\widetilde{ce}$ be a counterexample. Then **Vert**$(\widetilde{ce})$ denotes the set of all the states that appear in $\widetilde{ce}$. **Root**$(\widetilde{ce})$ denotes the root of $\widetilde{ce}$. For a path $\widetilde{\pi}$, **Root**$(\widetilde{\pi})$ denotes the state that $\widetilde{\pi}$ starts with.*

Definition 16 (Corresponding paths). *Let $\widetilde{I}$ be an abstract model of the interpreted system I, h be the abstraction function, and h_A be the function that maps the actions in I to the ones in $\widetilde{I}$. The concrete path $\pi = s_1 \rightarrow \cdots \rightarrow s_n$ in the concrete model corresponds to the path $\widetilde{\pi} = \widetilde{s}_1 \rightarrow \cdots \rightarrow \widetilde{s}_n$ in the abstract model, if*

- *For all $1 \leq i \leq n : \widetilde{s}_i = h(s_i)$*
- *If $\widetilde{s}_i \xrightarrow{\widetilde{\alpha}_{i+1}} \widetilde{s}_{i+1}$ is a temporal transition, we have $s_i \xrightarrow{\alpha_{i+1}} s_{i+1}$ where $h_A(\alpha_{i+1}) = \widetilde{\alpha}_{i+1}$.*
- *If $\widetilde{s}_i \sim_a \widetilde{s}_{i+1}$ is an epistemic transition, then $s_i \sim_a s_{i+1}$ and s_{i+1} is reachable in the concrete model.*

$$\textsc{TemporalCheck}\ \frac{h_A^{-1}(\widetilde{\alpha}) = \{\alpha_1, \ldots, \alpha_n\}}{(\widetilde{s} \xrightarrow{\widetilde{\alpha}} \widetilde{s}' \,||\, \pi, st) \Rightarrow_t (\pi, \bigcup_{i=1}^{n} \Theta_{\alpha_i}(st) \cap h^{-1}(\widetilde{s}'))}$$

$$\textsc{EpistemicCheck}\ \frac{\begin{array}{c}\pi' = \widetilde{s}'_0 \xrightarrow{\widetilde{\alpha}'_1} \ldots \xrightarrow{\widetilde{\alpha}'_m} \widetilde{s}' \text{ is a temporal path to } \widetilde{s}' \text{ where } \widetilde{s}'_0 \in \widetilde{S}_0 \\ (\pi', S_0 \cap h^{-1}(\widetilde{s}'_0)) \Rightarrow_t^* (\widetilde{s}', st') \qquad \hat{st} = \{s \in st' \mid l_a(s) \in L_a(st)\}\end{array}}{(\widetilde{s} \sim_a \widetilde{s}' \,||\, \pi, st) \Rightarrow_e (\pi, \hat{st})}$$

Fig. 3. Temporal and epistemic transition rules. In EPISTEMICCHECK rule, π' is the witness for the reachability of $\widetilde{s}'$ in the abstract model, and st' is the concrete states that are reachable through the concrete paths corresponding to π'. In the case that the model-checker returns all the abstract paths to $\widetilde{s}'$, let us say $\widetilde{\Pi}'$, then st' will be calculated as $st' = \bigcup\{st \mid \pi' = \widetilde{s}'_0 \to \cdots \to \widetilde{s}' \in \widetilde{\Pi}', \widetilde{s}'_0 \in \widetilde{S}_0$ and $(\pi', S_0 \cap h^{-1}(\widetilde{s}'_0)) \Rightarrow_t^* (\widetilde{s}', st)\}$.

Definition 17 (Concrete counterexample). *Let $\widetilde{ce}$ be a tree-like counterexample in the abstract model where **Root**$(\widetilde{ce}) \in \widetilde{S}_0$. A concrete counterexample ce corresponds to $\widetilde{ce}$ if **Root**$(ce) \in S_0$ and there exists a one-to-one correspondence between the states and the paths of the counterexamples ce and $\widetilde{ce}$ according to the definition 16.*

To *verify a path* in the counterexample, we define two transition rules TEMPORALCHECK and EPISTEMICCHECK denoted by $\Rightarrow_t$ and $\Rightarrow_e$ as in figure 3. For a path with the transition $\widetilde{s} \xrightarrow{\widetilde{\alpha}} \widetilde{s}'$ as the head and for the concrete states st, the rule $\Rightarrow_t$ finds all the successors of the states in st which reside in $h^{-1}(\widetilde{s}')$. If the head of the path is the epistemic transition $\widetilde{s} \sim_a \widetilde{s}'$, then the rule $\Rightarrow_e$ extracts all the *reachable states* in $h^{-1}(\widetilde{s}')$ corresponding to π' as the witness of reachability of $\widetilde{s}'$, which has common local states with some states in $st \subseteq h^{-1}(\widetilde{s})$. Both the temporal and epistemic rules are deterministic.

Definition 18. *We write $\Rightarrow_t^*$ to denote a sequence of temporal transitions $\Rightarrow_t$. We use $\Rightarrow^*$ to denote a sequence of the transitions $\Rightarrow_t$ or $\Rightarrow_e$.*

Proposition 3 (Soundness of $\Rightarrow_t^*$). *Let $\widetilde{\pi}$ be a temporal path in the abstract model which starts at $\widetilde{s}_1$ and ends in $\widetilde{s}_n$. If $st_1 \subseteq h^{-1}(\widetilde{s}_1)$ and $(\widetilde{\pi}, st_1) \Rightarrow_t^* (\widetilde{s}_n, st_n)$ for some $\emptyset \subset st_n \subseteq S$, then there exists a concrete path that starts from a state in st_1 and ends in a state in st_n.*

Proof. We use induction over the length of the path.

Base case: $\widetilde{\pi} = \widetilde{s}_1$. Then there is no transition from $(\widetilde{s}_1, st_1)$ and therefore, the concrete path is a state in st_1.

Inductive case: Assume by inductive hypothesis that for all $\widetilde{\pi} = \widetilde{s}_i \xrightarrow{\widetilde{\alpha}_{i+1}} \ldots \xrightarrow{\widetilde{\alpha}_{i+k}} \widetilde{s}_{i+k}$ of length k, if $(\widetilde{\pi}, st_i) \Rightarrow_t^* (\widetilde{s}_{i+k}, st_{i+k})$ for some $st_i, st_{i+k} \subseteq S$, then there exists a concrete path which begins at a state in st_i and ends in a state in st_{i+k}. Consider that $\widetilde{\pi}' = \widetilde{s}_{i-1} \xrightarrow{\widetilde{\alpha}_i} \widetilde{s}_i \,||\, \widetilde{\pi}$ is a path of the length $k+1$ where $(\widetilde{s}_{i-1} \xrightarrow{\widetilde{\alpha}_i} \widetilde{s}_i \,||\, \widetilde{\pi}, st_{i-1}) \Rightarrow_t (\widetilde{\pi}, st_i) \Rightarrow_t^* (\widetilde{s}_{i+k}, st_{i+k})$. By induction hypothesis, there exists a concrete path that begins at some state $s_i \in st_i$ and ends in $s_{i+k} \in st_{i+k}$. By the definition of $\Rightarrow_t$, every state in st_i is the successor of some states in st_{i-1}. Therefore, there exists $s_{i-1} \in st_{i-1}$ and $\alpha_i \in h_A^{-1}(\widetilde{\alpha}_i)$ such that $\{s_i\} = \Theta_{\alpha_i}(\{s_{i-1}\})$. So we select the corresponding transition in the concrete model to be $s_{i-1} \xrightarrow{\alpha_i} s_i$ which allows s_{i-1} to reach s_{i+k} by the existence of a concrete path from s_i to s_{i+k}.

By proposition 3 and definition 17, if $\widetilde{\pi} = \widetilde{s}_0 \xrightarrow{\widetilde{\alpha}_1} \dots \xrightarrow{\widetilde{\alpha}_n} \widetilde{s}_n$ is a path in the counterexample where $(\widetilde{\pi}, S_0 \cap h^{-1}(\widetilde{s}_0)) \Rightarrow_t^* (\widetilde{s}_n, st_n)$, then there exists a corresponding concrete path beginning at an initial state $s_0 \in S_0 \cap h^{-1}(\widetilde{s}_0)$ which ends at some state $s_n \in st_n$.

Proposition 4 (Soundness of $\Rightarrow^*$). *Let $\widetilde{\pi} = \widetilde{s}_1 \to \dots \to \widetilde{s}_n$ be a path in the abstract model. If $st_1 \subseteq h^{-1}(\widetilde{s}_1)$ and $(\widetilde{\pi}, st_1) \Rightarrow^* (\widetilde{s}_n, st_n)$ for some $\emptyset \subset st_n \subseteq S$, then there exists a concrete path that starts from a state in st_1 and ends in a state in st_n.*

Proof. For the general form of a path that contains both temporal and epistemic transitions, we use the similar approach as in proposition 3.

Base case: $\widetilde{\pi} = \widetilde{s}_1$. Then there is no transition from $(\widetilde{s}_1, st_1)$ and therefore, the concrete path is a state in st_1.

Inductive case: Assume by inductive hypothesis that for all $\widetilde{\pi} = \widetilde{s}_i \to \dots \to \widetilde{s}_{i+k}$ of length k, if $(\widetilde{\pi}, st_i) \Rightarrow^* (\widetilde{s}_{i+k}, st_{i+k})$ for some $st_i, st_{i+k} \subseteq S$, then $\widetilde{\pi}$ has a corresponding concrete path which begins at a state in st_i and ends in a state in st_{i+k}.

- Consider that $\widetilde{\pi}' = \widetilde{s}_{i-1} \xrightarrow{\widetilde{\alpha}_i} \widetilde{s}_i \,||\, \widetilde{\pi}$ is a path of length $k+1$ where $(\widetilde{s}_{i-1} \xrightarrow{\widetilde{\alpha}_i} \widetilde{s}_i \,||\, \widetilde{\pi}, st_{i-1}) \Rightarrow_t (\widetilde{\pi}, st_i) \Rightarrow^* (\widetilde{s}_{i+k}, st_{i+k})$. By induction hypothesis, there exists a concrete path that begins at some state $s_i \in st_i$ and ends in $s_{i+k} \in st_{i+k}$. By the same analysis as in the proof of proposition 3, there exists $s_{i-1} \in st_{i-1}$ and $\alpha_i \in h_A^{-1}(\widetilde{\alpha}_i)$ such that $s_{i-1} \xrightarrow{\alpha_i} s_i$. Hence, there exists a concrete path from s_{i-1} to s_{i+k}.
- Consider that $\widetilde{\pi}' = \widetilde{s}_{i-1} \sim_a \widetilde{s}_i \,||\, \widetilde{\pi}$ is a path of length $k+1$ where $(\widetilde{s}_{i-1} \sim_a \widetilde{s}_i \,||\, \widetilde{\pi}, st_{i-1}) \Rightarrow_e (\widetilde{\pi}, st_i) \Rightarrow^* (\widetilde{s}_{i+k}, st_{i+k})$. By induction hypothesis, there exists a concrete path that begins at some state $s_i \in st_i$ and ends in $s_{i+k} \in st_{i+k}$. By the definition of $\Rightarrow_e$ and proposition 3, s_i is reachable from some initial states in the concrete model, which is a requirement by definition 16. From $l_a(s_i) \in L_a(st_{i-1})$ we conclude that there exists $s_{i-1} \in st_{i-1}$ such that $l_a(s_i) = l_a(s_{i-1})$. Hence we select $s_{i-1} \sim_a s_i$ as the corresponding epistemic transition in the concrete model. Therefore, there exists a concrete path from s_{i-1} to s_{i+k}.

In the case that $\widetilde{\pi} = \widetilde{s}_0 \to \dots \to \widetilde{s}_n$ is a path in the counterexample and $(\widetilde{\pi}, S_0 \cap h^{-1}(\widetilde{s}_0)) \Rightarrow^* (\widetilde{s}_n, st_n)$, then there exists a corresponding concrete path beginning at some initial state $s_0 \in S_0 \cap h^{-1}(\widetilde{s}_0)$ which ends at some state $s_n \in st_n$.

Proposition 5 (Completeness of $\Rightarrow^*$). *Let $\widetilde{\pi} = \widetilde{s}_1 \to \dots \to \widetilde{s}_n$ be a path in the abstract model. If there exists a concrete path $\pi = s_1 \to \dots \to s_n$ corresponding to $\widetilde{\pi}$ and $s_1 \in st_1 \subseteq h^{-1}(\widetilde{s}_1)$, then $(\widetilde{\pi}, st_1) \Rightarrow^* (\widetilde{s}_n, st_n)$ for some $\emptyset \subset st_n \subseteq S$.*

Proof. For the completeness proof, we use induction over the length of the counterexamples.

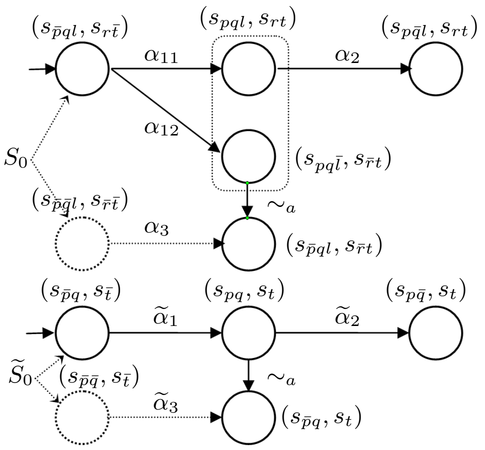

Fig. 4. The transition system on the top is the concrete model and on the bottom is the abstract one obtained by making the propositions l and r invisible.

Base case: $\widetilde{\pi} = \widetilde{s}_1$ and $\pi = s_1$. Then we will have no transition and the proposition automatically holds.

Inductive case: Assume by inductive hypothesis that for all $\widetilde{\pi} = \widetilde{s}_i \rightarrow \cdots \rightarrow \widetilde{s}_{i+k}$ of length k, if there exists a path $\pi = s_i \rightarrow \cdots \rightarrow s_{i+k}$ which corresponds to $\widetilde{\pi}$ and $s_i \in st_i \subseteq h^{-1}(\widetilde{s}_i)$, then $(\widetilde{\pi}, st_i) \Rightarrow^* (\widetilde{s}_{i+k}, st_{i+k})$ for some $\emptyset \subset st_{i+k} \subseteq S$.

- Consider that $\widetilde{s}_{i-1} \xrightarrow{\widetilde{\alpha}_i} \widetilde{s}_i \,||\, \widetilde{\pi}$ is a path of length $k+1$ which has the corresponding concrete path $s_{i-1} \xrightarrow{\alpha_i} s_i \,||\, \pi$. Let $st_{i-1} \in h^{-1}(\widetilde{s}_{i-1})$ be a set of states where $s_{i-1} \in st_{i-1}$. Then the transition $(\widetilde{s}_{i-1} \xrightarrow{\widetilde{\alpha}_i} \widetilde{s}_i \,||\, \widetilde{\pi}, st_{i-1}) \Rightarrow_t (\widetilde{\pi}, st_i)$ leads to the set st_i as the successors of the states in st_{i-1} with respect to the actions in $h_A^{-1}(\widetilde{\alpha}_i)$. As $\alpha_i \in h_A^{-1}(\widetilde{\alpha}_i)$, we have $s_i \in st_i$. Therefore by inductive hypothesis, we have $(\widetilde{s}_{i-1} \xrightarrow{\widetilde{\alpha}_i} \widetilde{s}_i \,||\, \widetilde{\pi}, st_{i-1}) \Rightarrow_t (\widetilde{\pi}, st_i) \Rightarrow^* (\widetilde{s}_{i+k}, st_{i+k})$ or equivalently $(\widetilde{s}_{i-1} \xrightarrow{\widetilde{\alpha}_i} \widetilde{s}_i \,||\, \widetilde{\pi}, st_{i-1}) \Rightarrow^* (\widetilde{s}_{i+k}, st_{i+k})$.
- Consider that $\widetilde{s}_{i-1} \sim_a \widetilde{s}_i \,||\, \widetilde{\pi}$ is a path of length $k+1$ which has the corresponding concrete path $s_{i-1} \sim_a s_i \,||\, \pi$. Let $st_{i-1} \in h^{-1}(\widetilde{s}_{i-1})$ be a set of states where $s_{i-1} \in st_{i-1}$. Then the transition $(\widetilde{s}_{i-1} \sim_a \widetilde{s}_i \,||\, \widetilde{\pi}, st_{i-1}) \Rightarrow_e (\widetilde{\pi}, st_i)$ leads to the set st_i which contains the reachable states with the same local states as the states in st_{i-1}. Therefore, $s_i \in st_i$ and by inductive hypothesis we have $(\widetilde{s}_{i-1} \sim_a \widetilde{s}_i \,||\, \widetilde{\pi}, st_{i-1}) \Rightarrow_e (\widetilde{\pi}, st_i) \Rightarrow^* (\widetilde{s}_{i+k}, st_{i+k})$ or equivalently $(\widetilde{s}_{i-1} \sim_a \widetilde{s}_i \,||\, \widetilde{\pi}, st_{i-1}) \Rightarrow^* (\widetilde{s}_{i+k}, st_{i+k})$.

Forward transition rules in figure 3 are sufficient to check *linear counterexamples* or equivalently, paths. To extend the counterexample checking to tree-like counterexample, extra procedures are required. We show the problem in the following example:

Example 2. Figure 4 demonstrates the transition system for a concrete interpreted system on top, and the abstract system on the bottom. The model contains two agents, e

$$\textsc{BackwardTCheck}\ \frac{\begin{array}{c}(\pi, S_0 \cap h^{-1}(\mathbf{Root}(\pi))) \Rightarrow^* (\widetilde{s}, st') \\ h_A^{-1}(\widetilde{\alpha}) = \{\alpha_1, \ldots, \alpha_n\} \qquad rs = \bigcup_{i=1}^{n} \Theta_{\alpha_i}^{-1}(st) \cap st'\end{array}}{(\pi \,||\, \widetilde{s} \xrightarrow{\widetilde{\alpha}} \widetilde{s}', st) \Leftarrow_t (\pi, rs) \qquad r_{\widetilde{s}} := rs}$$

$$\textsc{BackwardECheck}\ \frac{\begin{array}{c}(\pi, S_0 \cap h^{-1}(\mathbf{Root}(\pi))) \Rightarrow^* (\widetilde{s}, st'') \\ \pi' = \widetilde{s}'_0 \xrightarrow{\widetilde{\alpha}'_1} \ldots \xrightarrow{\widetilde{\alpha}'_m} \widetilde{s}' \text{ is the temporal path to } \widetilde{s}' \text{ where } \widetilde{s}'_0 \in \widetilde{S}_0 \\ (\pi', S_0 \cap h^{-1}(\widetilde{s}'_0)) \Rightarrow^* (\widetilde{s}', st') \\ \hat{st} = \{s \in st'' \mid l_a(s) \in L_a(st \cap st')\}\end{array}}{(\pi \,||\, \widetilde{s} \sim_a \widetilde{s}', st) \Leftarrow_e (\pi, \hat{st}) \qquad r_{\widetilde{s}} := \hat{st}}$$

Fig. 5. Backward temporal and epistemic transition traversal. $\Theta_{\alpha}^{-1}(st)$ computes the set of predecessors of the states in st with respect to the transitions made by action α.

as the environment and a as regular agent. States are shown as tuples where the first element is the local state of e and the second is the local state of a. The diagram distinguishes the states by using the value of local propositions as the subscript. The abstract model is generated by making the local proposition l of environment and r of agent a invisible.

We aim to verify $AG(p \to (K_a p \vee AGq))$ over the concrete model. This property holds for the original model, while it does not hold for the abstract one. The counterexample generated is:

$$\widetilde{ce} = \{(s_{\bar{p}q}, s_{\bar{t}}) \xrightarrow{\widetilde{\alpha}_1} (s_{pq}, s_t) \xrightarrow{\widetilde{\alpha}_2} (s_{p\bar{q}}, s_t), (s_{\bar{p}q}, s_{\bar{t}}) \xrightarrow{\widetilde{\alpha}_1} (s_{pq}, s_t) \sim_a (s_{\bar{p}q}, s_t)\}$$

To find out if there exists any concrete counterexample that corresponds to $\widetilde{ce}$, we check the paths in $\widetilde{ce}$ one by one. We show the paths in $\widetilde{ce}$ by $\widetilde{\pi}_1$ and $\widetilde{\pi}_2$. The paths $\widetilde{\pi}_1$ and $\widetilde{\pi}_2$ correspond to the concrete paths $\pi_1 = (s_{\bar{p}ql}, s_{r\bar{t}}) \xrightarrow{\alpha_{11}} (s_{pql}, s_{rt}) \xrightarrow{\alpha_2} (s_{p\bar{q}l}, s_{rt})$ and $\pi_2 = (s_{\bar{p}ql}, s_{r\bar{t}}) \xrightarrow{\alpha_{12}} (s_{pq\bar{l}}, s_{\bar{r}t}) \sim_a (s_{\bar{p}ql}, s_{\bar{r}t})$. Although all the paths in the counterexample have corresponding concrete paths, the tree does not correspond to a concrete tree. This is because if we select (s_{pql}, s_{rt}) as the corresponding state for (s_{pq}, s_t), then the leaf $(s_{\bar{p}ql}, s_{\bar{r}t})$ is not reachable from it. A similar situation happens when we select $(s_{pq\bar{l}}, s_{\bar{r}t})$. Therefore, the tree-like counterexample is spurious.

To verify a tree-like counterexample, we introduce two transition rules BACKWARDTCHECK and BACKWARDECHECK denoted by $\Leftarrow_t$ and $\Leftarrow_e$. The transition rules find all the predecessors of the states in st (figure 5) with respect to the temporal or epistemic transitions in a backward manner which reside in the set of reachable states through the path. We write $\Leftarrow^*$ to denote a sequence of backward transitions $\Leftarrow_t$ and $\Leftarrow_e$.

Assume that $\widetilde{\pi} = \widetilde{s}_0 \to \cdots \to \widetilde{s}_n$ is a path in the counterexample $\widetilde{ce}$ which $(\widetilde{\pi}, S_0 \cap h^{-1}(\widetilde{s}_0)) \Rightarrow^* (\widetilde{s}_n, st_n)$ for some $\emptyset \subset st_n \subseteq S$. st_n contains all the states in the leaves of the concrete paths corresponding to $\widetilde{\pi}$. The point is not all the concrete states that are traveresed in $\Rightarrow^*$ can reach the states in st_n. If $\widetilde{s} \in \mathbf{Vert}(\widetilde{\pi})$, then $(\widetilde{\pi}, st_n) \Leftarrow^* (\widetilde{s}_0, st_0)$ finds the set of states $r_{\widetilde{s}}$ which contains the reachable states in $h^{-1}(\widetilde{s})$ that lead to some states in st_n along the concrete paths corresponding to $\widetilde{\pi}$. st_0 contains the initial states that lead to the states in st_n. We use the notation $r^{\widetilde{\pi}}_{\widetilde{s}}$ to relate $r_{\widetilde{s}}$ with the path $\widetilde{\pi}$. Note that to find $r^{\widetilde{\pi}}_{\widetilde{s}}$, we first need to find st_n through $\Rightarrow^*$ transition.

Assume that $\widetilde{\Pi} \subseteq \widetilde{ce}$. If $\widetilde{s} \in \textbf{Vert}(\widetilde{ce})$ then we define $r^{\widetilde{\Pi}}_{\widetilde{s}} = \cap_{\widetilde{\pi}\in\widetilde{\Pi}} r^{\widetilde{\pi}}_{\widetilde{s}}$. If $\widetilde{s} \notin \textbf{Vert}(\widetilde{\pi})$, then we stipulate $r^{\widetilde{\pi}}_{\widetilde{s}} = h^{-1}(\widetilde{s})$. We also stipulate $r^{\emptyset}_{\widetilde{s}_0} = S_0 \cap h^{-1}(\widetilde{s}_0)$ where $\widetilde{s}_0 = \textbf{Root}(\widetilde{ce})$ and $r^{\emptyset}_{\widetilde{s}} = h^{-1}(\widetilde{s})$ for all $\widetilde{s} \in \textbf{Vert}(\widetilde{ce})$ where $\widetilde{s} \neq \widetilde{s}_0$.

Proposition 6 (Soundness of counterexample checking). *A counterexample $\widetilde{ce}$ in the abstract model has a corresponding concrete one if:*

1. *for each path $\widetilde{\pi} \in \widetilde{ce}$, there exists $\emptyset \subset st \subseteq S$ such that $(\widetilde{\pi}, S_0 \cap h^{-1}(\widetilde{s}_0)) \Rightarrow^* (\widetilde{s}', st)$ where $\widetilde{s}_0 = \textbf{Root}(\widetilde{ce})$ and $\widetilde{\pi}$ ends in $\widetilde{s}'$.*
2. *for all $\widetilde{s} \in \textbf{Vert}(\widetilde{ce}) : r^{\widetilde{ce}}_{\widetilde{s}} \neq \emptyset$.*

Proof. By the soundness of $\Rightarrow^*$, all the paths in $\widetilde{\pi}$ correspond to some concrete paths which satisfy the requirements in the definitions 16 and 17. Now for each $\widetilde{s} \in \textbf{Vert}(\widetilde{ce})$, we pick a state $s \in r^{\widetilde{ce}}_{\widetilde{s}}$ as the corresponding state. For each path in $\widetilde{ce}$ and between all the corresponding concrete paths, we pick the one which contains the selected states as its vertices. The union of the selected paths builds a concrete counterexample that satisfies the requirements in definition 17.

Proposition 7 (Completeness of counterexample checking). *Assume that $\widetilde{ce}$ corresponds to a concrete counterexample ce. Then both the items 1 and 2 in proposition 6 hold.*

Proof. By definition 17, there is a one-to-one correspondence between the paths of the two counterexamples. By completeness of $\Rightarrow^*$, item 1 holds for all the paths in $\widetilde{ce}$. Now Assume that $\widetilde{s} \in \textbf{Vert}(\widetilde{ce})$ and s is the corresponding state in ce. Then for all $\widetilde{\pi} \in \widetilde{ce}$, we have $s \in r^{\widetilde{\pi}}_{\widetilde{s}}$, and therefore $s \in r^{\widetilde{ce}}_{\widetilde{s}}$. Hence we have $r^{\widetilde{ce}}_{\widetilde{s}} \neq \emptyset$, as required for item 2.

Procedure 2 expresses the tree-like counterexample checking method in a more refined manner. CHECKCE iterates over the paths in $\widetilde{ce}$ and checks if they corresponds to some paths in the concrete model by using proposition 4 and the transition rule $\Rightarrow^*$. If $\widetilde{\pi}$ corresponds to some concrete paths, then for each state $\widetilde{s}$ in $\widetilde{\pi}$, the algorithm finds all the concrete states $r^{\widetilde{\pi}}_{\widetilde{s}}$ in $h^{-1}(\widetilde{s})$ that lead to the leaf states of the concrete paths by applying $\Leftarrow^*$ over $\widetilde{\pi}$. In each loop iteration, $\widetilde{\Pi}$ stores the paths in $\widetilde{ce}$ that are processed in previous iterations. The set $r^{\widetilde{\Pi}}_{\widetilde{s}}$ stores the concrete states that are common between the paths in $\widetilde{\Pi}$ and should remain non-empty during the process of counterexample checking. The procedure returns false if no corresponding tree-like counterexample for $\widetilde{ce}$ exists. Otherwise it returns true.

Example 3. We recall the transition system in example 2. As also discovered in the example, the paths $\widetilde{\pi}_1$ and $\widetilde{\pi}_2$ correspond to the concrete paths $\pi_1 = (s_{\bar{p}ql}, s_{r\bar{t}}) \xrightarrow{\alpha_{11}} (s_{pql}, s_{rt}) \xrightarrow{\alpha_2} (s_{p\bar{q}l}, s_{rt})$ and $\pi_2 = (s_{\bar{p}ql}, s_{r\bar{t}}) \xrightarrow{\alpha_{12}} (s_{pq\bar{l}}, s_{\bar{r}t}) \sim_a (s_{\bar{p}ql}, s_{\bar{r}t})$. By backward traversing through the first path and for the states in $h^{-1}((s_{pq}, s_t))$, we find that only the state (s_{pql}, s_{rt}) leads to the final state on π_1 and so, $r^{\widetilde{\pi}_1}_{(s_{pq},s_t)} = \{(s_{pql}, s_{rt})\}$.

Procedure 2 Counterexample checking algorithm

```
function CheckCE($\widetilde{ce}, I, h$)
    ▷ Input: $\widetilde{ce}$ is the counterexample, $I$ is the concrete model and $h$ is the abstraction function
    ▷ Output: returns true if a concrete counterexample exists. Returns false otherwise.
    $\{\widetilde{s}_0, \ldots, \widetilde{s}_n\} = \mathbf{Vert}(\widetilde{ce})$        ▷ $\widetilde{s}_0 = \mathbf{Root}(\widetilde{ce})$
    $\widetilde{\Pi} = \emptyset$
    $r^{\widetilde{\Pi}}_{\widetilde{s_0}} = S_0 \cap h^{-1}(\widetilde{s}_0), r^{\widetilde{\Pi}}_{\widetilde{s}_1} = h^{-1}(\widetilde{s}_1), \ldots, r^{\widetilde{\Pi}}_{\widetilde{s}_n} = h^{-1}(\widetilde{s}_n)$
    for all $\widetilde{\pi} \in \widetilde{ce}$ do
        if $(\widetilde{\pi}, r^{\widetilde{\Pi}}_{\widetilde{s_0}}) \Rightarrow^* (\widetilde{s}', st)$ and $st \neq \emptyset$ then        ▷ $\widetilde{\pi}$ ends at the state $\widetilde{s}'$
            ▷ there exists some concrete path corresponding to $\widetilde{\pi}$
            for all $\widetilde{s} \in \mathbf{Vert}(\widetilde{ce})$ do
                determine $\hat{r}^{\widetilde{\pi}}_{\widetilde{s}}$ from $(\widetilde{\pi}, st) \Leftarrow^* (\widetilde{s}_0, st')$
                ▷ determine the concrete states corresponding to $\widetilde{s}$
                $r^{\widetilde{\Pi} \cup \{\widetilde{\pi}\}}_{\widetilde{s}} := r^{\widetilde{\Pi}}_{\widetilde{s}} \cap r^{\widetilde{\pi}}_{\widetilde{s}}$
                if $r^{\widetilde{\Pi} \cup \{\widetilde{\pi}\}}_{\widetilde{s}} = \emptyset$ then
                    ▷ no common concrete state for $\widetilde{s}$ between concrete paths exists
                    return false
                end if
            end for
            $\widetilde{\Pi} := \widetilde{\Pi} \cup \{\widetilde{\pi}\}$
        else
            return false
        end if
    end for
    return true
end function
```

$$\textsc{TemporalTree}\ \frac{h_A^{-1}(\widetilde{\alpha}) = \{\alpha_1, \dots, \alpha_n\}}{(\widetilde{s} \xrightarrow{\widetilde{\alpha}} \widetilde{s}' \,||\, \pi, st) \Rightarrow_t^{\widetilde{\Pi}} (\pi, \bigcup_{i=1}^{n} \Theta_{\alpha_i}(st) \cap r_{\widetilde{s}'}^{\widetilde{\Pi}})}$$

$$\textsc{EpistemicTree}\ \frac{\begin{array}{c}\pi' = \widetilde{s}'_0 \xrightarrow{\widetilde{\alpha}'_1} \dots \xrightarrow{\widetilde{\alpha}'_m} \widetilde{s}' \text{ is a temporal path to } \widetilde{s}' \text{ where } \widetilde{s}'_0 \in \widetilde{S}_0 \\ (\pi', S_0 \cap h^{-1}(\widetilde{s}'_0)) \Rightarrow_t^{*} (\widetilde{s}', st') \qquad \hat{st} = \{s \in st' \cap r_{\widetilde{s}'}^{\widetilde{\Pi}} \mid l_a(s) \in L_a(st)\}\end{array}}{(\widetilde{s} \sim_a \widetilde{s}' \,||\, \pi, st) \Rightarrow_e^{\widetilde{\Pi}} (\pi, \hat{st})}$$

Fig. 6. Transition rules for finding failure state in a tree-like counterexample.

The same approach for π_2 results in $r^{\widetilde{\pi}_2}_{(s_{pq}, s_t)} = \{(s_{pq\bar{l}}, s_{\bar{r}t})\}$. As $r^{\widetilde{\pi}_1}_{(s_{pq}, s_t)} \cap r^{\widetilde{\pi}_2}_{(s_{pq}, s_t)} = \emptyset$, the state (s_{pq}, s_t) can not be assigned to a concrete single state. Therefore, $\widetilde{ce}$ is spurious.

7.3 Refinement of the abstraction

If the counterexample is found to be spurious, then the abstraction should be refined. The abstract model is generated by making some propositions in the concrete model invisible. For the refinement, we split some states in the abstract model by putting some of the invisible propositions back into the model. These propositions should be selected in such a way that when verifying the refined model, the same counterexample does not appear again. In this section, we provide the mechanism for refining the abstraction.

Let $\widetilde{ce}$ be a spurious counterexample. We define two transition rules TEMPORALTREE which is denoted by $\Rightarrow_t^{\widetilde{\Pi}}$ and EPISTEMICTREE denoted by $\Rightarrow_e^{\widetilde{\Pi}}$ where $\widetilde{\Pi} \subseteq \widetilde{ce}$ in figure 6. As before, $\Rightarrow_*^{\widetilde{\Pi}}$ denotes a sequence of temporal and epistemic transitions of the type $\Rightarrow_t^{\widetilde{\Pi}}$ and $\Rightarrow_e^{\widetilde{\Pi}}$. We use the following technique in order to find the state in the spurious counterexample which needs to be split:

The state $\widetilde{s}_i \in \textbf{Vert}(\widetilde{ce})$ is a *failure state* if there exists $\widetilde{\Pi} \subseteq \widetilde{ce}$ and $\widetilde{\pi} \in \widetilde{ce} \backslash \widetilde{\Pi}$ such that:

1. For all $\widetilde{s} \in \textbf{Vert}(\widetilde{\Pi}) : r_{\widetilde{s}}^{\widetilde{\Pi}} \neq \emptyset$
2. $\widetilde{\pi} = \widetilde{\pi}_1 \,||\, \widetilde{s}_i(\xrightarrow{\widetilde{\alpha}_{i+1}} \mid \sim_a)\widetilde{s}_{i+1} \,||\, \widetilde{\pi}_2$ such that $(\widetilde{\pi}, r_{\widetilde{s}_0}^{\widetilde{\Pi}}) \Rightarrow_*^{\widetilde{\Pi}} (\widetilde{\pi}_1, st_d) \Rightarrow_{(t|e)}^{\widetilde{\Pi}} (\widetilde{\pi}_2, \emptyset)$ for some $st_d \neq \emptyset$.

For a spurious counterexample, such $\widetilde{\Pi}$ and $\widetilde{\pi}$ exists. Otherwise, we will have $r_{\widetilde{s}}^{\widetilde{ce}} \neq \emptyset$ for all $\widetilde{s} \in \textbf{Vert}(\widetilde{ce})$, which contradicts proposition 6.

Based on Item 1), the sub-tree $\widetilde{\Pi}$ has a corresponding counterexample in the concrete model. In item 2), $\widetilde{\pi}$ traverses over the concrete states that belong to the set of concrete trees corresponding to $\widetilde{\Pi}$ and gets to the set of states $st_d \subseteq h^{-1}(\widetilde{s}_i)$ with no transition to a state in $r_{\widetilde{s}_{i+1}}^{\widetilde{\Pi}}$. In the standard terminology as in [6], $\widetilde{s}_i$ is called *failure state*. We use the term *dead end state* for the states in st_d which the concrete paths end up with and can not go further. *Bad states* are the states in $h^{-1}(\widetilde{s}_i)$ that have transition to some states in $r^{\widetilde{\Pi}}_{\widetilde{s}_{i+1}}$. Note that in a path counterexample, we have that $r^{\widetilde{\Pi}}_{\widetilde{s}_{i+1}} = h^{-1}(\widetilde{s}_{i+1})$.

The process of finding a failure state in the counterexample $\widetilde{ce}$ proceeds as follows:

1. Set $\widetilde{\Pi}$ to empty set at the beginning
2. Find $r^{\widetilde{\Pi}}_{\widetilde{s}}$ for all $\widetilde{s} \in \textbf{Vert}(\widetilde{ce})$ (as also mentioned in section 7.2, $r^{\emptyset}_{\widetilde{s}_0} = S_0 \cap h^{-1}(\widetilde{s}_0)$ where $\widetilde{s}_0 = \textbf{Root}(\widetilde{ce})$ and $r^{\emptyset}_{\widetilde{s}} = h^{-1}(\widetilde{s})$ for all $\widetilde{s} \in \textbf{Vert}(\widetilde{ce})$ where $\widetilde{s} \neq \widetilde{s}_0$)
3. Pick a path $\widetilde{\pi} \in \widetilde{ce}$ that does not exist in $\widetilde{\Pi}$
4. Apply $\Rightarrow^{\widetilde{\Pi}}_{*}$ over $(\widetilde{\pi}, r^{\widetilde{\Pi}}_{\widetilde{s}_0})$ to find failure state. If a failure state exists over $\widetilde{\pi}$, then exit and refine the model
5. Add $\widetilde{\pi}$ to $\widetilde{\Pi}$ and return to step 2. Note that we are considering that the counterexample is found to be spurious (by the procedure 2) and therefore, such failure state will be found before all the paths in $\widetilde{ce}$ are added to $\widetilde{\Pi}$.

For the implementation, the above process can be easily incorporated into the procedure 2.

To refine the model, we find the propositions that having them invisible results in generating spurious counterexample. First assume that the transition from $\widetilde{s}_i$ to $\widetilde{s}_{i+1}$ is temporal, say $\widetilde{s}_i \xrightarrow{\widetilde{\alpha}_{i+1}} \widetilde{s}_{i+1}$. Two situations can result in a transition of type $\Rightarrow^{\widetilde{\Pi}}_{t}$ from st_d to an empty set of states:

- There exists no $\alpha_{i+1} \in h^{-1}(\widetilde{\alpha}_{i+1})$ such that $\Theta_{\alpha_{i+1}}(st_d) \neq \emptyset$. Therefore, no action has the permission to be performed on the states of st_d. Assume that ϕ_d is the formula that represents the set of states st_d. As the state space is finite, the formula representing the states always exists. Therefore, for all $\alpha_{i+1} \in h^{-1}_{A}(\widetilde{\alpha}_{i+1})$ with ℓ_{i+1} as the permission, we have $\phi_d \wedge \ell_{i+1} \equiv \bot$. We call ℓ_{i+1} *conflict formula* and ϕ_d *base formula*.
- For some $\alpha_{i+1} \in h^{-1}(\widetilde{\alpha}_{i+1})$ we have $\Theta_{\alpha_{i+1}}(st_d) \neq \emptyset$. By the definition of $\Rightarrow_t$ we have $\Theta_{\alpha_{i+1}}(st_d) \cap r^{\widetilde{\Pi}}_{\widetilde{s}'_{i+1}} = \emptyset$ where $r^{\widetilde{\Pi}}_{\widetilde{s}'_{i+1}} \neq \emptyset$. If ϕ is the formula representing $\Theta_{\alpha_{i+1}}(st_d)$ and ψ the formula representing $r^{\widetilde{\Pi}}_{\widetilde{s}'_{i+1}}$, then we have $\psi \wedge \phi \equiv \bot$. We call ϕ conflict formula and ψ base formula.

The other situation is when the transition $\widetilde{s}_i$ and $\widetilde{s}_{i+1}$ is epistemic, say $\widetilde{s}_i \sim_a \widetilde{s}_{i+1}$. Three situations can result in the epistemic transition $\Rightarrow^{\widetilde{\Pi}}_{e}$ to an empty set of states:

- π' as the witness of the reachability of $\widetilde{s}_{i+1}$ in $\Rightarrow^{\widetilde{\Pi}}_{e}$ is spurious. Then the refinement should be guided by analysing π' instead of the main spurious path.
- Suppose that π' has corresponding concrete paths, i.e. $(\pi', S_0 \cap h^{-1}(\widetilde{s}'_0)) \Rightarrow^{*}_{t} (\widetilde{s}_{i+1}, st')$ where $st' \neq \emptyset$. By the definition of $\Rightarrow_e$, the epistemic transition results in an empty set of states if $st' \cap r^{\widetilde{\Pi}}_{\widetilde{s}'_{i+1}} = \emptyset$. If ϕ is the formula representing st' and ψ the formula representing $r^{\widetilde{\Pi}}_{\widetilde{s}'_{i+1}}$, then we call ϕ conflict formula and ψ base formula.

- The third reason for the epistemic transition to an empty set is when no shared *local state* exists between the states of st_d and $st' \cap r^{\widetilde{\Pi}}_{\widetilde{s}'_{i+1}}$ where st' is the set of reachable states according to the previous item and both the sets are non-empty. In the other words, $L_a(st_d) \cap L_a(st' \cap r^{\widetilde{\Pi}}_{\widetilde{s}'_{i+1}}) = \emptyset$. The formula representing the local states in st_d with respect to the agent a is called base formula, and the formula representing the local states of $st' \cap r^{\widetilde{\Pi}}_{\widetilde{s}'_{i+1}}$ is the conflict formula.

To refine the model, we return some hidden propositions to separate the set of dead end states from the rest of the states. This can simply be done by adding all the propositions occurring in *conflict clauses* to the abstract model.

Definition 19. *(conflict clause) Let ϕ be the base formula and ψ the conflict formula. Let **cnf**(ψ) denote the set containing all the conjuncts appear in conjunctive normal form of ψ. Then $c \in$ **cnf**(ψ) is a* conflict clause *if $c \wedge \phi \equiv \bot$.*

If the propositions that occur in one of the conflict clauses become visible, then the spurious strategy will not happen in the refined model again. In the case of temporal transition, we add the propositions in the conflict clauses for *all the conflicting actions*. To have the smallest possible refinement, we should look for the conflict classes with the *smallest number* of literals.

7.4 Going beyond ACTLK

While this section develops a fully automated abstraction refinement method for the verification of temporal-epistemic properties that reside the category of ACTLK over an access control system which is modelled by an interpreted system, some important epistemic safety properties does not reside in this category. For instance and in a conference paper review system, it is valuable for policy designers to verify that for all reachable states, an author of a paper cannot find out ($\neg K$) who is the reviewer of his own paper (see the first property in example 1). Although we are able to verify such properties in the concrete model, we cannot apply automated counterexample-guided abstraction and refinement for such properties.

Let us explore the problem. Assume that for the abstract system $\widetilde{I}$, abstract state $\widetilde{s}$ and agent a, $(\widetilde{I}, \widetilde{s}) \models \neg K_a \varphi$. That means there exists a state $\widetilde{s}'$ such that $\widetilde{s}' \sim_a \widetilde{s}$ and $(\widetilde{I}, \widetilde{s}') \models \neg\varphi$. If s is a state in the concrete model where $h(s) = \widetilde{s}$, then the satisfaction relation $(\widetilde{I}, \widetilde{s}) \models \neg K_a \varphi$ implies $(I, s) \models \neg K_a \varphi$ if it guarantees the existence of a *reachable* state $s' \in h^{-1}(\widetilde{s}')$ such that $s' \sim_a s$ and $(I, s') \models \neg\varphi$.

First of all, if such s' exists, the satisfaction relation $(\widetilde{I}, \widetilde{s}') \models \neg\varphi$ still does not imply $(I, s') \models \neg\varphi$ when φ is ACTLK except if φ is simply a propositional formula which is the case for many of the properties that we are interested in. Second, the relation $\widetilde{s}' \sim_a \widetilde{s}$ in the abstract model does not imply $s' \sim_a s$ in the concrete model for some reachable state $s' \in h^{-1}(\widetilde{s}')$. In the case that $(\widetilde{I}, \widetilde{s}') \not\models \neg\varphi$, the model-checker produces a counterexample that can be checked using the method that is developed in this section and then the abstract model can be refined. In the case that the satisfaction relation holds, the model-checker does not produce any witness.

To complete our work for the properties that deal with the negation of knowledge operator, we restrict the formula in scope of the knowledge operators to propositional formulas. Then we use an interactive refinement procedure in the following way: we abstract the interpreted system in the standard way that we described. If the property does not hold in the abstract model, the counterexample will be checked in the concrete model and the abstract model will be refined if it is required. If the property turned to be true in the abstract model as a result of the satisfaction of $\neg K_a$ (which we would not have any witness in the abstract model), then we refine the local state of the agent a in an interactive manner. In this way, the tool asks the user to selects a set of invisible *local propositions* to be added in the next round if required. This process will continue until a valid counterexample is found, or the local state becomes concretized. In the case that the safety property does not hold in the concrete model (where information leakage vulnerability exists), then there is a chance to find it out with the abstract model when the local states are still abstract.

8 Experimental results

We have implemented a tool in F# functional programming language. The font end is a parser that accepts a set of action and read permission rules, a set of objects and a query in the form of $\iota : \varphi$ where ι is the formula representing the initial states and φ is the property we aim to verify. Given the above information, the tool derives an interpreted system based on definition 8 where the initial states of the system are determined by parameter ι in the query. On the back end, we use MCMAS [8] as the model-checking engine. In the presence of abstraction and refinement, the tool feeds MCMAS with the abstracted version of the original interpreted system together with the property φ. If model-checker returns true for an ACTLK property, then the tool returns true to the user. Otherwise, the tool automatically checks the generated counterexample based on proposition 6, and reports if it is a real counterexample, which will be returned to the user, or verification needs a refinement round. The tool performs an automated refinement if it is required. For the properties that are discussed in section 7.4, the tool asks user to select a set of invisible local variables to be added to the abstract model for the refinement when model-checker returns true. This will continue until all the related invisible local variables turn to visible, or a valid counterexample is found.

For this section, we choose one temporal and three epistemic properties for the case study of conference paper review system (CRS) with the information leakage vulnerability described in the introduction. We first verify the query (Query 1) "$\mathsf{author}(\mathsf{p}_1, \mathsf{a}_1) \wedge \neg\mathsf{reviewer}(\mathsf{p}_1, \mathsf{a}_1) : AG(\neg\mathsf{reviewer}(\mathsf{p}_1, \mathsf{a}_1))$" which states that if in the initial states, agent a_1 is the author of paper p_1 and not the reviewer of his own paper, then it is not possible for a_1 to be assigned as the reviewer of his paper p_1. Query 2 "$\neg\mathsf{submittedreview}(\mathsf{p}_1, \mathsf{a}_1) \wedge \mathsf{reviewer}(\mathsf{p}_1, \mathsf{a}_2) : AG(K_{\mathsf{a}_1}\mathsf{review}(\mathsf{p}_1, \mathsf{a}_2) \rightarrow AG(\neg\mathsf{submittedreview}(\mathsf{p}_1, \mathsf{a}_1))$ checks if in the initial states, a_1 and a_2 are the reviewers of paper p_1, and a_1 has not submitted his own review of p_1, then a_1 cannot submit her review if he *reads* the review of a_2 (knowledge by readability). Query 3 $\mathsf{author}(\mathsf{p}_1, \mathsf{a}_1) : AG(\mathsf{AllPapersAssigned} \wedge \mathsf{reviewer}(\mathsf{p}_1, \mathsf{a}_2) \rightarrow \neg K_{\mathsf{a}_1}\mathsf{reviewer}(\mathsf{p}_1, \mathsf{a}_2))$ asks if a_1 is the author of p_1, then it is not possible for a_1 to find the reviewer of his paper when his paper is assigned to a_2, which

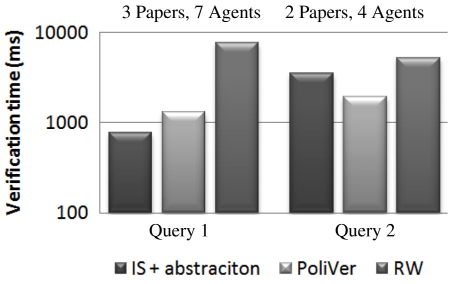

Fig. 7. Comparison of the verification time for the queries 1 and 2 between our tool which uses MCMAS as the model-checking engine, PoliVer and RW.

	Concrete model		Abstraction and refinement			
	time(s)	BDD vars	time(s)	Max BDD vars	last ref time	num of ref
Query 3	6576.5	180	148.3	80	3.28	7
Query 4	6546.4	180	174.1	98	21	12

Fig. 8. A comparison of query verification time (in second) and runtime memory usage (in MB) between the concrete model and automated abstraction refinement method.

is not ACTLK. Query 4 $\mathsf{author}(\mathsf{p}_1, \mathsf{a}_1) : AG(\mathsf{AllPapersAssigned} \wedge \mathsf{reviewer}(\mathsf{p}_1, \mathsf{a}_2) \rightarrow K_{\mathsf{a}_1}\mathsf{reviewer}(\mathsf{p}_1, \mathsf{a}_2))$ has ACTLK property, which checks if a_1 can always find who the reviewer of his paper is whenever all the papers are assigned.

Queries 1 and 2 can be verified in access control policy verification tools like RW and PoliVer, which model knowledge by readability. We compare our tool in the presence of abstraction and refinement with RW and PoliVer from the point of verification time in figure 7. It is important to note that when applying abstraction and refinement, a high percentage of evaluation time is spent on generating the whole concrete model at the beginning, invoking executable MCMAS which also invokes Cygwin library, generating abstract model and verifying the counterexample. In most of our experiments, verification of the final abstract model by MCMAS takes less than 10ms.

The novel outcome of our research is the verification of the queries 3 and 4 where PoliVer and RW are unable to detect information leakage in CRS policy. In PoliVer and RW models, the author never finds a chance to see who the reviewer of his paper is and therefore safety property holds in the system. Modeling in interpreted systems reveals that the author can reason who is the reviewer of his paper when all the papers are assigned. For Query 3, the tool also outputs the counterexample which demonstrates the sequence of actions that allows the author to reason about the reviewer of his paper. Figure 8 shows the practical importance of our abstraction method (interactive refinement for Query 3 and fully automated for Query 4).

9 Conclusion

In this research, we introduced a framework for verifying temporal and epistemic properties over access control policies. In order to verify knowledge by reasoning, we used interpreted systems as the basic framework and to make the verification practical for medium to large systems, we extended counterexample-guided refinement known as CEGAR to cover safety properties in ACTLK. Case studies and experimental results show a considerable reduction in time and space when abstraction and refinement are in use. We also applied an interactive refinement for some useful properties that does not reside in ACTLK like the ones that contain the negation of knowledge modality. As future work, we would like to use these technique to detect information-flow in real world systems such as electronic voting systems [18–20] and social networks.

Acknowledgement: We would like to acknowledge Microsoft Research and EPSRC project TS/I002529/1 “Trust Domains” for funding this research.

References

1. Becker, M.Y.: Specification and analysis of dynamic authorisation policies. In: Proc. IEEE Computer Security Foundations Symposium. (July 2009) 203–217
2. Zhang, N., Ryan, M., Guelev, D.P.: Synthesising verified access control systems through model checking. Journal of Computer Security **16**(1) (2008) 1–61
3. Dougherty, D.J., Fisler, K., Krishnamurthi, S.: Specifying and reasoning about dynamic access-control policies. In: Proc. International Joint Conference on Automated Reasoning. (August 2006) 632–646
4. Mardare, R., Priami, C.: Dynamic epistemic spatial logics. Technical report, The Microsoft Research-University of Trento Centre for Computational and Systems Biology (2006)
5. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning About Knowledge. MIT Press, Cambridge (1995)
6. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Proc. Computer Aided Verification. (July 2000) 154–169
7. Clarke, E.M., Lu, Y., Com, B., Veith, H., Jha, S.: Tree-like counterexamples in model checking. In: Proc. IEEE Symposium on Logic in Computer Science. (July 2002) 19–29
8. Lomuscio, A., Raimondi, F.: MCMAS: A model checker for multi-agent systems. In: proc. Tools and Algorithms for the Construction and Analysis of Systems. (April 2006) 450–454
9. Aucher, G., Boella, G., van der Torre, L.: Privacy policies with modal logic: The dynamic turn. In: Deontic Logic in Computer Science. (2010) 196–213
10. Koleini, M., Ryan, M.: A knowledge-based verification method for dynamic access control policies. In: Proc. International Conference on Formal Engineering Methods. (2011)
11. Cohen, M., Dam, M., Lomuscio, A., Russo, F.: Abstraction in model checking multi-agent systems. In: AAMAS 2009: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems. (2009) 945–952
12. Zhou, C., Sun, B., Liu, Z.: Abstraction for model checking multi-agent systems. Frontiers of Computer Science in China **5** (2011) 14–25
13. Fagin, R., Halpern, J.Y., Moses, Y., Vardis, M.Y.: Knowledge-based programs. Distributed Computing **10**(4) (1997) 199–225
14. Lomuscio, A., Raimondi, F.: Model checking knowledge, strategies, and games in multi-agent systems. In: Proc. International Conference on Autonomous Agents and Multiagent Systems. (May 2006) 161–168
15. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM **49**(5) (2002) 672–713
16. Lomuscio, A., Raimondi, F.: The complexity of model checking concurrent programs against CTLK specifications. In: Proc. International Conference on Autonomous Agents and Multi-agent Systems. (May 2006) 548–550
17. Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. ACM Transactions on Programming Languages and Systems **16**(5) (1994) 1512–1542

18. Clarkson, M.R., Chong, S., Myers, A.C.: Civitas: Toward a secure voting system. In: Proc. IEEE Symposium on Security and Privacy. (May 2008) 354–368
19. Bursuc, S., Grewal, G.S., Ryan, M.D.: Trivitas: Voters directly verifying votes. In: Proc. E-Voting and Identity. (September 2011) 190–207
20. Grewal, G.S., Ryan, M.D., Bursuc, S., Ryan, P.Y.A.: Caveat coercitor: Coercion-evidence in electronic voting. In: Proc. IEEE Symposium on Security and Privacy. (May 2013) 367–381