Modular Counterexample Guided Abstraction Refinement for Temporal-Epistemic Logic

Andrew V. Jones and Alessio Lomuscio

Verification of Autonomous Systems Group
Department of Computing
Imperial College London, UK
{andrewj,alessio}@doc.ic.ac.uk

Abstract. We introduce a fully automatic technique for the modular abstraction refinement of multi-agent systems with respect to a formula in the universal fragment of the temporal-epistemic CTLK. Unlike previous approaches to temporal-epistemic abstraction refinement, our procedure is modular and works at the local component level. Consequently, the validation of counterexamples and the refinement of quotient structures can be done in a compositional manner without constructing the full system. We show that our abstraction refinement technique can be applied easily to a common multi-agent formalism pertaining to synchronous systems. We demonstrate the technique by means of a simple example.

1 Introduction

In [5], Clarke *et al.* present Counterexample Guided Abstraction Refinement (CEGAR), an automated technique to refine existential abstractions based on a counterexample demonstrating why a given formula is invalidated on an abstract model. If such a counterexample correlates to a behaviour in the concrete structure, then it can be decided that the given counterexample is a valid demonstration as to why the formula also does not hold on the concrete model. Alternatively, if it is shown to be invalid (i.e., there is no concrete behaviour correlating to the given counterexample), then the existential abstraction can be refined into one in which the spurious counterexample is no longer present. We summarise the presented CEGAR-based approach below.

Given a system of n agents $\mathcal{M}_1, \dots, \mathcal{M}_n$ and a formula φ in ACTLK (with agent-local atomic propositions), the technique is as standard [4,5]:

1. **Abstract.** Create n abstract agents $\widehat{\mathcal{M}_1}, \dots, \widehat{\mathcal{M}_n}$, such that the behaviours of the abstract composition $\widehat{\mathcal{M}_{\|}}$ are a superset of those in the concrete composition $\mathcal{M}_{\|}$.
2. **Verify.** Using model checking, verify whether the abstract compositional model $\widehat{\mathcal{M}_{\|}}$ satisfies φ. If the formula is satisfied, it can be deduced that the formula also holds on the concrete composition. Otherwise, generate an abstract counterexample that witnesses the failure of φ in $\widehat{\mathcal{M}_{\|}}$.
3. **Refine.** Component-wise, check if the abstract counterexample can be mapped onto a concrete counterexample in the components $\mathcal{M}_1, \dots, \mathcal{M}_n$. If it can, it then follows

that φ does not hold on the concrete model. Otherwise, the counterexample is spurious and can be used to refine the abstractions $\widehat{\mathcal{M}_1}, \ldots, \widehat{\mathcal{M}_n}$ to remove the invalid behaviour. The procedure is then iterated by verifying the refined structure.

The ability to perform counterexample refutation and abstraction refinement at component level is believed to be beneficial as this avoids constructing the full composed system for either model checking or counterexample validation. Should a counterexample be shown to be spurious with respect to a single component, it can then be deduced that the counterexample is also invalid in the full, concrete composition. This means that one possible refinement strategy is to refine only the single component that refutes the witness. Such a deduction removes the possibility of needlessly over-refining components.

Additionally, constructing the non-abstracted transition relation to perform counterexample refutation may not be possible as this step alone may exhibit the infamous state-space explosion problem. Component-level counterexample analysis is an approach to ameliorate this explosion.

In this paper, we combine the results of [7] with [1] to yield a modular abstraction refinement technique for ACTLK. It is important to note that for the agent formalism we look at, synchronisation between multiple components is handled in the abstract model; this allows us to deal with concrete components in a modular fashion. We do not need to reason about deadlock at the local level, as any provided abstract counterexample is already deadlock-free, or will be refuted by an individual component.

1.1 Related Work

Abstraction for Temporal-Epistemic Logic The work of Cohen *et al.* [7] for local abstraction is the most closely related to the method presented. Whilst the work of [7] does not automate the abstraction process nor does it perform refinement, it does present the theoretical foundations for a notion of simulation that preserves temporal-epistemic formulae across quotient structures. One main difference between the approach presented and the framework of Cohen *et al.* is that as well as clustering local states, they also cluster actions into abstract action classes. Comparatively, we only abstract the local states of each component; enabled actions and transitions are obtained via uniform substitution of concrete states with their abstract counterparts.

Zhou *et al.* [13] present the first approach to perform CEGAR (*à la* [5]) to temporal-epistemic logic. The approach of [13] is to first construct the composition of the concrete components, then perform abstraction on this composition, after which the abstract global state-space can be constructed. Comparatively, our approach performs abstraction at agent level and then constructs the global transition relation before finally calculating the abstract global state-space. Their refinement steps work at the global level, that is, the abstract trace witnessing the falsification of the specification is checked for validity over the concrete global state-space. This is clearly a limitation as, for some systems, even constructing the concrete global transition relation might be prohibitively expensive. In the same way as the work presented here, Zhou *et al.* do not cluster actions into abstract classes. Critically, it should also be noted that in [13], the authors neglect to refine the abstract model in light of a spurious epistemic link in the model. We overcome this limitation by making this step explicit in the refinement stage of the approach presented.

Modular Abstraction Refinement Chaki *et al.* [2,3,4] present an approach for the modular abstraction refinement of labelled transition systems (LTS) for specifications in both the branching [1] and linear-time [2] fragments of a state-event logic. In such a logic formulae can be defined over both state propositions and the labels on action edges. Our work is very close in spirit to these approaches, in that we have transferred their abstraction and refinement methods from LTS to an agent-specific framework.

2 Preliminaries

2.1 Interpreted Systems

The interpreted systems formalism is a standard semantics for multi-agent systems [8,11]. In this paper, we only consider the synchronous class of interpreted systems from [8]. Such systems comprise of a set $\mathscr{A}$ of n agents.

For each $i \in \mathscr{A}$, i's associated *agent program* $\mathcal{M}_i$ consists of:

- L_i – a non-empty set of local states;
- Σ_i – a non-empty set of local actions;
- $\iota_i \subseteq L_i$ – a non-empty set of initial states;
- $\rho_i : L_i \to 2^{\Sigma_i}$ – a protocol function that enables certain actions in a given state;
- $\delta_i : L_i \times \prod_{j \in \mathscr{A}} \Sigma_j \to L_i$ – an evolution function determining how an agent evolves over time w.r.t. a global action;
- AP_i – the atomic propositions of agent i; and
- $V_i : L_i \to 2^{AP_i}$ – a mapping from local states to propositions (where convenient, we also use $V_i : AP_i \to 2^{L_i}$).

Additionally, we write $\Sigma \subseteq \prod_{i \in \mathscr{A}} \Sigma_j$ for the global alphabet and $\overline{a} \in \Sigma$ for an arbitrary global action. Given $\overline{a}$, we write $\overline{a} \downharpoonright_i$ for the i^{th} component in $\overline{a}$.

A single global state $g = (l_i, \ldots, l_n)$ represents an instantaneous configuration of the whole system; we write $l_i(g)$ to represent agent i's local state in the global state g. For a subset G' of G, $l_i(G')$ returns the set of unique local states. Two global states are i-indistinguishable, written $g \sim_i g'$, iff $l_i(g) = l_i(g')$.

In synchronous systems every agent performs an action at each step, and can observe the global action that occurs and update its local state accordingly. Given the synchronous composition of the protocols and evolution functions for each agent, we can define a global transition relation $T \subseteq G \times \Sigma \times G$ such that $(g, \overline{a}, g') \in T$ iff for all $i \in \mathscr{A}$, $\overline{a} \downharpoonright_i \in \rho_i(l_i(g))$ and $\delta_i(l_i(g), \overline{a}) = l_i(g')$.

An infinite path $\pi = (g_0, \overline{a}_0, g_1, \overline{a}_1, \ldots)$ can be constructed by unwinding T starting at the global state $g_0 \in G$, such that for all $i \geq 0$, $T(g_i, \overline{a}_i) = g_{i+1}$. We write $\Pi(g)$ to represent the set of paths π beginning at g, and $\pi(n)$ to represent the n^{th} state in π.

A model $\mathcal{M}$ of an interpreted system is based upon the set of states G, the transition relation T and the interpretation of both $\sim_i$ and V_i over G, for all $i \in \mathscr{A}$. Finally, we write $\iota \subseteq \prod_{i \in \mathscr{A}} \iota_i$ to represent the initial state of the model.

2.2 Temporal-Epistemic Logic

The logic CTLK is the fusion of Computational Tree Logic (CTL) with the $S5_n$ logic for knowledge. We follow the presentation of [10], in that we write $K_i\varphi$ for agent *i knows* φ and $\overline{K_i}\varphi$ for agent *i considers* φ *possible*.

Syntax The BNF for CTLK is given below.

$$\varphi :== p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid AX\varphi \mid A\left[\varphi \overline{U} \varphi\right] \mid A[\varphi U \varphi] \mid K_i\varphi$$

We define the duals as follows: $EX\varphi \triangleq \neg AX\neg\varphi$; $E[\varphi U\varphi] \triangleq \neg A\left[\neg\varphi\overline{U}\neg\varphi\right]$; $E\left[\varphi\overline{U}\varphi\right] \triangleq \neg A[\neg\varphi U\neg\varphi]$; $\overline{K_i}\varphi \triangleq \neg K_i\neg\varphi$. We also use the following abbreviations: $EF\varphi \triangleq E[\mathsf{true}U\varphi]$; $EG\varphi \triangleq E\left[\mathsf{false}\overline{U}\varphi\right]$; $AF\varphi \triangleq A[\mathsf{true}U\varphi]$; $AG\varphi \triangleq A\left[\mathsf{false}\overline{U}\varphi\right]$. We define the universal fragment of CTLK, ACTLK, to contain only $AX, A\overline{U}, AU$ and K_i, as well as restricting negation to propositional atoms. The existential fragment, ECTLK, is its dual.

Satisfaction For a global state $g = (l_1, \ldots, l_n)$, the satisfaction of a formula (written $g \models \varphi$) is shown below.

$g \models p_i$	iff	$p_i \in V_i(l_i(g))$
$g \models \neg\varphi$	iff	it is not the case that $g \models \varphi$
$g \models \varphi \vee \psi$	iff	$g \models \varphi$ or $g \models \psi$
$g \models \varphi \wedge \psi$	iff	$g \models \varphi$ and $g \models \psi$
$g \models AX\varphi$	iff	$\forall\pi \in \Pi(g), \pi(1) \models \varphi$
$g \models A\left[\varphi\overline{U}\psi\right]$	iff	$\forall\pi \in \Pi(g), \forall i \geq 0, \pi(i) \models \neg\psi$ implies $\exists 0 \leq j < i, \pi(j) \models \varphi$
$g \models A[\varphi U\psi]$	iff	$\forall\pi \in \Pi(g), \exists m \geq 0, \pi(m) \models \psi$ and $\forall 0 \leq j < m, \pi(j) \models \varphi$
$g \models K_i\varphi$	iff	$\forall g' \in G, g \sim_i g'$ implies $g' \models \varphi$

2.3 Counterexamples

A counterexample is a model such that its associated formula does not hold upon it. In the context of this paper, we consider counterexamples that are submodels of the model of interest, such that the counterexample contains the states and transitions that cause the given universal formula to be invalidated. For the existential fragment, counterexamples are called witnesses, as they provide evidence as to why a formula holds in a model. Given that the negation of an existential formula is a universal formula, it follows that a counterexample to a universal formula is actually a witness to its negation.

For the existential fragment of CTLK, witnesses (*viz* counterexamples for their negation) are said to be "tree-like" [6]. Such witnesses can be constructed by layering indexed subsets of the global state-space onto each other. States are indexed to avoid introducing cycles into the graph that remove its tree-like structure. For a further treatment of the procedure to generate counterexamples for the temporal-only fragment of CTLK, we refer the reader to [6].

3 Abstraction

In this section we consider existential abstractions that can be obtained by collapsing sets of concrete states into a single abstract state. The benefit of abstracted components is that they usually have much more succinct representations, which allow us to ameliorate the state-space explosion problem. The process of abstraction is to group concrete states into abstract states, and then perform model checking on the abstract states.

For an agent $i \in \mathscr{A}$, we write $\widehat{L_i}$ to represent i's set of local abstract states – informally, we call each abstract state in $\widehat{L_i}$ a "cluster". An abstraction $\approx_i$: $L_i \to \widehat{L_i}$ is a surjection from concrete states to the abstract cluster they are collapsed into. We only consider admissible abstraction functions, where the states in each cluster agree on the evaluation of all atomic propositions, i.e., if $\approx_i (l_i) = \approx_i (l'_i)$ then $V_i(l_i) = V_i(l'_i)$.

For an abstract agent state cluster $\widehat{s} \in \widehat{L_i}$, $h_i(\widehat{s})$ represents the set of states that are clustered into $\widehat{s}$ (i.e., h_i is the inverse of $\approx_i$). For convenience, we denote $[s]$ with the abstract state that s is mapped to under $\approx_i$.

Given $\approx_i$ we say that $\widehat{\mathcal{M}_i} = \mathcal{M}_i / \approx_i$ is a quotient structure of $\mathcal{M}_i$. The local states of $\mathcal{M}_i / \approx_i$ are constructed by collapsing all states related under $\approx_i$ into the same abstract state. As $\approx_i$ is admissible, all related states satisfy the same propositions. The protocol and evolution of $\mathcal{M}_i / \approx_i$ can be constructed by substituting every concrete state with the abstract state it maps to. For each protocol rule, if the action $a_i \in \Sigma_i$ is enabled by a state l_i, then a_i is also enabled in the abstract state $[l_i]$. For each evolution rule, if $l_i \times \overline{a} \rightarrow l'_i$ exists in the concrete protocol then $[l_i] \times \overline{a} \rightarrow [l'_i]$ exists in the abstract protocol. A quotient interpreted systems model $\widehat{\mathcal{M}}$ can be constructed by composing n abstract agents in the standard way.

When performing iterated abstraction refinement, we write $\approx_i^n$ for the nth abstraction of agent i. Additionally, the initial abstraction (i.e., $\approx_i^0$) is the minimal admissible abstraction, i.e., it is based purely on the agent local propositions.

The indistinguishability relation for agent i ($\sim_i$) in the concrete model is finer than in the abstract model. This means that if a state g is epistemically related under agent i to a state g' in the concrete model, then $[g]$ is also epistemically related under the same agent to $[g']$ in the abstract structure. However, as the converse does not hold, we require that a CEGAR approach for epistemic logic also deals with spurious epistemic links. This will be covered in the next section.

Theorem 1. *Preservation of ACTLK*

For two models $\mathcal{M}$ and $\mathcal{M}'$ such that $\mathcal{M}'$ is a quotient structure of $\mathcal{M}$, then for an arbitrary ACTLK formula φ:

$$\mathcal{M}' \models \varphi \text{ implies } \mathcal{M} \models \varphi$$

The above theorem can be closely related to the simulation theorem of Cohen *et al.* [7]. We can therefore use their simulation relation results directly. In what follows, we only consider admissible abstractions that admit valid simulations of their concrete counterparts.

4 Compositional Abstraction Refinement

In this section, we first present a unique counterexample generation method for CTLK (§ 4.1) and then present an algorithmic method for validating a counterexample from an abstract model on a concrete system (§ 4.2).

4.1 Generating Counterexamples for ECTLK

Temporal For the temporal fragment, our abstraction-refinement methodology does not require counterexamples that differ from standard; we refer the reader to [6] for their construction.

Epistemic The generation of the coarsest possible refinement has been shown to be NP-hard [5]. As such our approach to counterexample generation attempts to extract as much information as possible from the current abstract model prior to refinement and re-verification.

For a formula of the form $\overline{K_i}\varphi$, its witness contains two states g and g' such that $g \sim_i g'$ and $g' \models \varphi$. As such, we wish to construct witnesses for epistemic formulae that are also formed of two-parts: firstly, a witness of the subformula φ at the state g', and secondly an "attack graph" (Section 4.1, [12]) from the abstract model that demonstrates all possible temporal paths from the initial state to g' in the abstract model. Attack graphs are used to identify the possible existence of a common, non-spurious path for each concrete component, as well as to check the reachability of a concrete state in $h_i(l_i(g'))$.

Should the epistemic formula be nested under further temporal-epistemic modalities, the witness for the current formula can be generated recursively from using either the temporal case (as above) or the presented epistemic case. This witness can then be appended to the witness of the parent formula. As per [6], we require that the set of states generated for each subformula have a fresh index. This allows for the correct counterexample to be selected during validation.

Attack Graphs. We review the notion of *attack graphs* from [12]. Given a model $\mathcal{M}$, its associated set of global states G and proposition *unsafe*, an attack graph is a model $\mathcal{A}$ with a set of states $G' \subseteq G$ such that for all $g \in G'$, for all $\pi \in \Pi(g)$, there exists an $n \geq 0$ such that $\pi(n) \models$ *unsafe*. Attack graphs, as presented in [12], are used for security purposes where the attack state modelling *unsafe* is a state where the intruder has gained access to the system. Each global path in the attack graph is therefore a witness to all possible ways in which the attacker can gain entry.

Given an existing model and a proposition representing the attack state, the procedure for generating an attack graph is shown in Algorithm 1. Given the formula $\overline{K_i}\varphi$, the set of states S' generated by GENERATE_ATTACK_GRAPH is the single abstract state where the witness for subformula φ begins.

We use attack graphs to generate all abstract traces from the initial state to the epistemically related state where the subformula witness begins. The proposition used to represent the attack state, which is used to generate the attack graph, is uniquely defined to be only the single target state. As a single, agent-local proposition could not uniquely define the global target state, we use a conjunction of agent-local propositions.

Extracting all acyclic traces from an attack graph can be done in a simple depth-first approach, where the previously visited states and actions are tracked for each visited state on the path so far. To avoid cycles in the attack graph, we ensure the DFS approach never visits the same state twice. Given that the attack graph contains all paths from the initial state to the indistinguishable state, visiting a state twice should never be required. From each non-initial state, there exists a path from that state to the end state, therefore the DFS can recursively start the new search from this state and then append the initial trace to the beginning of its non-initial trace.

Algorithm 1 GENERATE_ATTACK_GRAPH

```
Input: $G$                                   # set of states
Input: $T \subseteq G \times Act \rightarrow G$        # transition relation
Input: $\iota \subseteq G$                           # set of initial states
Input: $\varphi = AG(\neg unsafe)$                # a safety property
Output: $\mathcal{A} = (G', T', \iota', S')$          # attack graph for "unsafe"
1: $G' \leftarrow SAT_{\mathrm{CTLK}(\varphi)}$             # Using global model checking, find
                                               the set of states that violate
                                               $AG(\neg unsafe)$
2: $T' \leftarrow T \cap (G' \times Act \times G')$
3: $\iota' \leftarrow \iota \cap G'$                        # Restrict the states and transition
                                               relation to $G'$
4: $S' \leftarrow \{s \in G' \mid s \models unsafe\}$       # Generate the set of end states
5: return $(G', T', \iota', S')$
```

4.2 Validity of Counterexamples

A procedure for counterexample validation can be seen in Algorithm 2. The subprocedures CHECK_PATH and CHECK_LOOP are from [5] (where they are labelled as *SplitPath* and *SplitLoop* respectively). Informally, the procedure CHECK_PATH checks an abstract path for validity starting at the passed state, returning the set of concrete states if the path is valid, or NULL otherwise. Similarly, CHECK_LOOP checks an abstract loop (e.g., as generated by the witness of $EG\psi$) for validity, and returns the set of concrete states, or NULL otherwise. Algorithm 2 requires both formulae and their witnesses to be indexed, such that the correct subwitness can be checked at an arbitrary concrete state (see [6]). Intuitively, $witness(\varphi)$ returns either the path or loop counterexample for the ECTLK formula φ, treating any subformulae as if they were propositions; VALIDATE_CEX can be recursively called to check the validity of the subformulae of φ. We write $\mathcal{A}(\varphi)$ to represent the associated attack graph for a formula $\varphi = K_i\psi$.

After performing model checking on an abstracted interpreted system, the algorithm can be called with the negated universal formula and the initial, concrete state of $\mathcal{M}$ to check if the generated counterexample is valid.

In Algorithm 2, a subformula $K_i\varphi$ is spurious if none of the attacks to the target epistemic state reach the same local state where the formula is being evaluated. More precisely, in the abstract model we must have $l_i(\widehat{g}) = l_i(\widehat{g'})$, otherwise the counterexample would not have been generated. However, the concrete states reached under the attack graph of the state $\widehat{g'}$, may reach a set of local states $L_i' \subseteq L_i$, such that $l_i(s) \notin L_i'$. As such, there is not a corresponding epistemic link in the concrete model, so the epistemic link is spurious.

Should the epistemic link exist in the concrete model, but the witness for the subformula φ is shown spurious, then this is not a spurious epistemic link of $K_i\varphi$, but a spurious behaviour for φ. Therefore this behaviour should be refined separately.

Algorithm 2 VALIDATE_CEX

```
Input: $s \in G$                                  # A concrete global state
Input: $\varphi \in ECTLK$                         # current subformula
Output: $\{true, false\}$                          # spurious or not
1: $flag \leftarrow true$
2: if $\varphi \in AP$ then
3:     $flag \leftarrow true$
4: else if $\varphi = EX\psi$ then
5:     if $s_0, s_1 \leftarrow$ CHECK_PATH$(s, witness(\varphi))$ then
6:         $flag \leftarrow$ VALIDATE_CEX $(s_1, \psi)$
7:     else
8:         $flag \leftarrow false$
9:     end if
10: else if $\varphi = EF\psi$ then
11:     if $s_0, \ldots, s_n \leftarrow$ CHECK_PATH$(s, witness(\varphi))$ then
12:         $flag \leftarrow$ VALIDATE_CEX $(s_n, \psi)$
13:     else
14:         $flag \leftarrow false$
15:     end if
16: else if $\varphi = EG\psi$ then
17:     if $s_0, \ldots, s_n \leftarrow$ CHECK_LOOP$(s, witness(\varphi))$ then
18:         for $m \leftarrow 0 \ldots n$ do
19:             $flag \leftarrow flag \wedge$ VALIDATE_CEX $(s_m, \psi_m)$
20:         end for
21:     else
22:         $flag \leftarrow false$
23:     end if
24: else if $\varphi = K_i\psi$ then
25:     $attacks \leftarrow \{t \in paths(\mathcal{A}(\varphi)) \mid$ CHECK_PATH$(\iota, t) \neq$ NULL$\}$
26:     $i\_states \leftarrow \{l_i(s_n) \in L_i \mid t \in attacks \wedge s_0, \ldots, s_n =$ CHECK_PATH$(\iota, t)\}$
27:     if $l_i(s) \notin i\_states$ then
28:         $flag \leftarrow false$
29:     else
30:         $flag \leftarrow$ VALIDATE_CEX $(s, \psi)$
31:     end if
32: end if
33: return $flag$
```

4.3 Abstraction Refinement

In this section, we present two approaches for refinement of an abstraction function, to be used if a temporal-epistemic counterexample is shown to be spurious.

We present a refinement methodology for both the temporal case and the epistemic case.

Algorithm 3 SPLIT_PATH

```
Input: $\widehat{s}_0, \overline{a}_0, \dots, \overline{a}_{n-1}, \widehat{s}_n$    # A spurious path
Input: $\approx_i^n$    # i's $n^{th}$ partition
Input: $s \in G$    # Starting concrete global state, reached via the parent formula
Output: $\approx_i^{n+1}$    # i's $n+1^{th}$ partition, rejecting the spurious path
1: $j \leftarrow 0$
2: $Q_j \leftarrow l_i(s) \cap l_i(h_i(\widehat{s}_j))$
3: while $Q_j \neq \emptyset$ and $j \leq n$ do
4:     $j \leftarrow j+1$
5:     $Q_j \leftarrow \delta_i(Q_{j-1}, \overline{a}_{j-1}) \cap l_i(h_i(\widehat{s}_j))$
6: end while
7: if $Q_j = \emptyset$ then
8:     $j \leftarrow j-1$
9:     $\widehat{s'}_j \leftarrow Q_j$    # $Q_j$ is the set of actually reached states.
10:    $\widehat{s''}_j \leftarrow l_i(h_i(\widehat{s}_j)) \setminus Q_j$    # The remainder of the abstract set $\widehat{s}_j$.
11:    $\approx_i^{n+1} \leftarrow (\approx_i^n \setminus \widehat{s}_j) \cup \left\{\widehat{s'}, \widehat{s''}\right\}$    # Replace the set $\widehat{s}_j$ with $\widehat{s'}_j$ and $\widehat{s''}_j$.
12: end if
```

Temporal Refinement Given a temporal path and a concrete state, Algorithm 3 refines $\approx_i^n$ to remove the behaviour exhibited by the given path, should it be shown to be spurious. Our approach is similar to that of the validation and refinement methods of [5] and [3]. We refer the reader to these publications for proof of correctness.

Algorithm 3 creates two new abstract clusters: one containing the last concrete state reached and the other containing all other states.

For a spurious loop counterexample, the loop can be unrolled into a path counterexample, and therefore the same procedure for paths can be applied.

Epistemic Refinement Our refinement approach for spurious epistemic links is presented in Algorithm 4. The algorithm creates three new abstract clusters:

1. $\widehat{s_i^{\mathrm{E}}}$ – the set of correct, end local states (i.e., the set of states reachable using the parent formula);
2. $\widehat{s_i^{\mathrm{NE}}}$ – the set of all valid, but not correct, end states (i.e., the set of states that are reachable under a certain valid attack, but do not reach the correct state); and
3. $\widehat{s_i^{\mathrm{O}}}$ – all remaining states in the abstract cluster (i.e., $l_i\left(h_i\left([s]\right)\right) \setminus \left(h_i\left(\widehat{s_i^{\mathrm{E}}}\right) \cup h_i\left(\widehat{s_i^{\mathrm{NE}}}\right)\right)$).

```
Algorithm 4 SPLIT_EPISTEMIC
Input: $\approx_i^n$                                   # i's n^th partition
Input: $s \in G$                                       # Starting concrete global state,
                                                         reached via the parent formula
Input: $\mathcal{A}$                                   # The current associated attack graph
Output: $\approx_i^{n+1}$                              # i's n+1^th partition, rejecting the
                                                         spurious epistemic link
1: $valid\_attacks \leftarrow \{t \in \mathcal{A} \mid \text{CHECK\_PATH}(\iota,t) \neq \text{NULL}\}$
2: $incorrect\_i\_states \leftarrow \{l_i(s_n) \in L_i \mid t \in valid\_attacks \wedge s_0,\dots,s_n = \text{CHECK\_PATH}(\iota,t) \wedge l_i(s_n) \neq l_i(s)\}$
3: $\widehat{s_i^{\mathrm{E}}} \leftarrow l_i(s)$       # Correct local state reached by the
                                                         parent counterexample
4: $\widehat{s_i^{\mathrm{NE}}} \leftarrow incorrect\_i\_states$   # Incorrect local states reached via
                                                         the attack graph
5: $\widehat{s_i^{\mathrm{O}}} \leftarrow l_i(h_i([s])) \setminus \left(h_i\left(l_i^{\mathrm{E}}\right) \cup h_i\left(l_i^{\mathrm{NE}}\right)\right)$   # Remaining states in $l_i(h_i([s]))$
6: $\approx_i^{n+1} \leftarrow \left(\approx_i^n \setminus l_i([s])\right) \cup \left\{\widehat{s_i^{\mathrm{E}}}, \widehat{s_i^{\mathrm{NE}}}, \widehat{s_i^{\mathrm{O}}}\right\}$
```

5 Example

In the following, we consider a two-agent system ($\mathscr{A} = \{1,2\}$). We use labels on arcs to represent the joint action that the current transition is conditional upon. For example, 'w, v' denotes that the transition can only occur if Agent 1 performs action w and Agent 2 performs v. We write '$*$' if the arc is unconditional on the action of the agent at that position (e.g., '$w, *$' states the transition is only conditional upon Agent 1 performing w, and Agent 2's actions are unrestricted). Similarly, '$(w|v), *$' represents that the transition is conditional on Agent 1 performing either 'w' or 'v' (and unrestricted for Agent 2).

Figures 1 and 2 show the FSM representation of the synchronous agent programs $\mathcal{M}_1$ and $\mathcal{M}_1$. Some transitions in Agent 1 are conditional on the actions of Agent 2, but no transitions in Agent 2 are conditional on Agent 1.

We consider the existential specification $\varphi = EF\left(p \wedge \overline{K_1}EFq\right)$ (its universal equivalent is $\neg\varphi = AG(\neg p \vee K_i AG \neg q)$), where $AP_1 = \{p,q\}$, $AP_2 = \emptyset$, $V_1(p) = \left\{l_1^1, l_1^2\right\}$ and $V_1(q) = \left\{l_1^3\right\}$.

These propositions give rise to the abstraction shown in Figure 1, where the boxes R, G and B represent the initial admissible abstraction. The abstract structure shown in Figure 3 depicts the synchronous composition of $\widehat{\mathcal{M}_1}$ and $\widehat{\mathcal{M}_2}$ (which is equivalent to $\mathcal{M}_2$). Figure 4 is the abstract witness for φ.

Attacks on the abstract state $l_i^{\mathrm{G}} \times l_j^1$ in the composed model are: 'a,x', 'a,y', 'b,x' and 'b,y'. By simulating 'b,x', the initial part of the witness from Figure 4, along $\mathcal{M}_1$ it reaches the concrete state l_i^2. For the counterexample to be valid, we now need to find the set of attacks that also reach l_i^2 – this is only 'b,x'. Next, we see if 'a,y' (the second part of the counterexample) is accepted from the state l_i^2 in $\mathcal{M}_1$. Clearly, we can see that it is not. This means that we need to remove the state l_i^2 from the abstract set of states l_i^{G}.

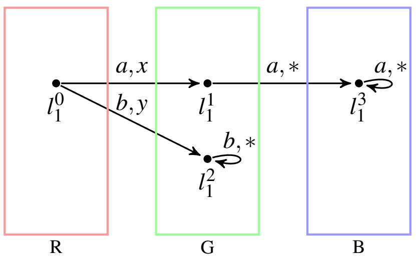

Fig. 1. The component $\mathcal{M}_1$ illustrating the initial clustering of states (R,G and B)

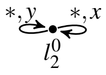

Fig. 2. The component $\mathcal{M}_2$

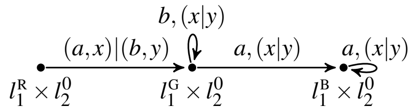

Fig. 3. The abstract model $\widehat{\mathcal{M}_{||}}$

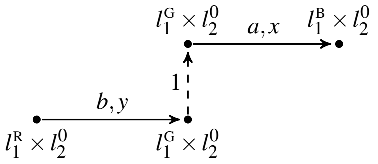

Fig. 4. An abstract counterexample from the formula $EF\left(p \wedge \overline{K_1} EF q\right)$ in the model $\widehat{\mathcal{M}_{||}}$

It should be noted that if the initial anchored part of the counterexample had been: 'a,x' rather than 'b,y', then the whole counterexample would have been valid. Unfortunately, as both counterexamples are valid in the abstract model, we have no way of constraining the model checker to generate the other counterexample.

Another point to note is that in this example, the 1-indistinguishable "source" and "target" global states are the same state. In practice, this is unlikely to be the case, hence the need for the presented epistemic-refinement methodology.

6 Conclusions

In this paper we have presented the theoretical foundations for an automatic and modular approach to counterexample abstraction refinement for the universal fragment of a branching time, temporal-epistemic logic. Our contribution is novel in two ways: 1) our approach handles abstraction, counterexample refutation and refinement at the local component level; and 2) we address the refinement of local components based on spuriously epistemically related states.

Unlike previous approaches that require the construction of the composition of the full concrete components for both abstraction and refutation, our approach avoids this possibly detrimental step by only reasoning about the composition of abstract components or the concrete components in isolation. It is possible that the coarsest abstraction that satisfies the universal formula is actually the full concrete model itself. Therefore, the construction of the full concrete global state-space cannot be avoided.

As demonstrated in the worked example, one issue with current CEGAR approaches is that they are dependent on a single counterexample for the temporal-only fragment. Although the provided temporal witness might be spurious, it is possible that another witness for the same formula may not be. One important avenue to investigate is an approach to obtain multiple counterexamples for a given formula (including all the corresponding epistemic attack graphs).

We are interested in implementing our approach into the open-source multi-agent model checker MCMAS [9].

References

1. Chaki, S., Clarke, E.M., Grumberg, O., Ouaknine, J., Sharygina, N., Touili, T., Veith, H.: State/Event Software Verification for Branching-Time Specifications. In: Proceedings of the 5th International Conference on Integrated Formal Methods (IFM 2005). Lecture Notes in Computer Science, vol. 3771, pp. 53–69. Springer (2005)
2. Chaki, S., Clarke, E.M., Ouaknine, J., Sharygina, N., Sinha, N.: State/Event-Based Software Model Checking. In: Proceedings of the 4th International Conference on Integrated Formal Methods (IFM 2004). Lecture Notes in Computer Science, vol. 2999, pp. 128–147. Springer (2004)
3. Chaki, S., Clarke, E.M., Ouaknine, J., Sharygina, N., Sinha, N.: Concurrent Software Verification with States, Events, and Deadlocks. Formal Asp. Comput. 17(4), 461–483 (2005)
4. Chaki, S., Ouaknine, J., Yorav, K., Clarke, E.M.: Automated Compositional Abstraction Refinement for Concurrent C Programs: A Two-Level Approach. Electr. Notes Theor. Comput. Sci. 89(3) (2003)
5. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-Guided Abstraction Refinement for Symbolic Model Checking. J. ACM 50(5), 752–794 (2003)
6. Clarke, E.M., Jha, S., Lu, Y., Veith, H.: Tree-Like Counterexamples in Model Checking. In: Proccedings of the 17th IEEE Symposium on Logic in Computer Science (LICS 2002). pp. 19–29. IEEE Computer Society (2002)
7. Cohen, M., Dam, M., Lomuscio, A., Russo, F.: Abstraction in Model Checking Multi-Agent Systems. In: Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009). pp. 945–952. IFAAMAS (2009)
8. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning About Knowledge. MIT Press (1995)
9. Lomuscio, A., Qu, H., Raimondi, F.: MCMAS: A Model Checker for the Verification of Multi-Agent Systems. In: Proceedings of 21st International Conference on Computer Aided Verification (CAV 2009). Lecture Notes in Computer Science, vol. 5643, pp. 682–688. Springer (2009)
10. Meyer, J.J.C., van der Hoek, W.: Epistemic Logic for AI and Computer Science. Cambridge University Press (1995)
11. Parikh, R., Ramanujam, R.: Distributed Processes and the Logic of Knowledge. In: Proceedings of the International Conference on Logics of Programs. Lecture Notes in Computer Science, vol. 193, pp. 256–268. Springer (1985)
12. Sheyner, O., Haines, J.W., Jha, S., Lippmann, R., Wing, J.M.: Automated Generation and Analysis of Attack Graphs. In: Proceedings of the 2002 IEEE Symposium on Security and Privacy. pp. 273–284 (2002)
13. Zhou, C., Sun, B., Liu, Z.: Abstraction for Model Checking Multi-Agent Systems. Frontiers of Computer Science in China 5(1), 14–25 (2011)