Fundamenta Informaticae XXI (2001) 1001–1019
IOS Press

Model Checking Temporal-Epistemic Logic Using Alternating Tree Automata

Francesco Belardinelli, Andrew V. Jones, Alessio Lomuscio
Department of Computing
Imperial College London, UK
{f.belardinelli,andrew.jones,a.lomuscio}@imperial.ac.uk

Abstract. We introduce a novel automata-theoretic approach for the verification of multi-agent systems. We present epistemic alternating tree automata, an extension of alternating tree automata, and use them to represent specifications in the temporal-epistemic logic CTLK. We show that model checking a memory-less interpreted system against a CTLK property can be reduced to checking the language non-emptiness of the composition of two epistemic tree automata. We report on an experimental implementation and discuss preliminary results. We evaluate the effectiveness of the technique using two real-life scenarios: a gossip protocol and the train gate controller.

1. Introduction

Multi-agent systems (MAS) are a useful paradigm for reasoning about distributed, concurrent systems where the individual components, or *agents*, act autonomously and collaboratively to meet private and common objectives. MAS are typically specified by using intensional languages, including multi-modal logics [2]. Considerable research over the past twenty years has focused on developing formalisms to specify MAS, including languages to reason about their knowledge (epistemic logic), their beliefs (doxastic logic), their intentions, and the obligations they should adhere to (deontic logic). In particular, various forms of epistemic logic [5, 16] have found application in a MAS setting when reasoning about robotics, web services, security, and, more generally, in areas where it is useful to consider the agents' information states.

More recently, attention has been given to the problem of automatic verification of MAS by model checking [4]. The objective is to develop efficient methodologies to check automatically whether a MAS of interest meets its specifications. Several techniques have been put forward over the past few years in this respect, ranging from bounded model checking [9], to symbolic model checking [14, 6], and partial order reduction [13].

While the approaches above have been demonstrated to be of value, they are on their own insufficient to tackle the complexity of scenarios arising from the industry. It is therefore of utmost importance to explore novel methodologies that may help in this respect, either on their own or in combination with existing techniques.

One method that has received little attention so far in the context of epistemic logic community is that of automata-based model checking. Yet, model checking via Büchi automata, originally explored in the

seminal paper by Vardi *et al.* [20], is one of the leading approaches in verification of reactive systems and constitutes the basis of the well-known model checker SPIN [8]. The work in [20] covered linear time only; this was extended to branching time logic in [12], where the formalism of alternating tree automata (ATA) was used.

The aim of this paper is to explore the extent to which automata-based model checking can be adopted in a MAS setting. Specifically, we work with CTLK (the customary temporal-epistemic logic with branching time) as a specification language, and plain interpreted systems [5] as the underlying semantics. These are at times referred to as memory-less interpreted systems, or systems with no memory. We first extend the notion of ATA to more general structures that can be used in this setting (Section 3). We then give a uniform translation of CTLK specifications into these automata (Section 3.1). This enables us to define a suitable notion of automata product and give an emptiness condition for satisfaction (Section 3.2). Having laid the theoretical foundations and demonstrated their soundness, we move on to report on a novel implementation we have developed (Section 4.1), and present experimental results (Section 4.2). Finally, we conclude in Section 5.

2. Prerequisites

In this section we introduce the temporal-epistemic logic CTLK, which combines the computation tree logic CTL with the epistemic logic $\mathrm{S5}_m^C$ with common knowledge for a set $A = \{1, \ldots, m\}$ of agents. Then we provide this language with a formal semantics in terms of interpreted systems [5, 16].

2.1. The Temporal Epistemic Logic CTLK

Given a set $P = \{p_1, p_2, \ldots\}$ of propositional variables, the temporal-epistemic language $\mathcal{L}_m$ contains the propositional variables in P, the connectives $\neg$ and $\rightarrow$, the linear time operators X and U, the branching time operators A and E, the epistemic operator K_i for each agent $i \in A$, and the common knowledge operator C_G for each non-empty set $G \subseteq A$ of agents.

Definition 2.1. The formulae in $\mathcal{L}_m$ are defined in BNF as:

$$\phi \quad ::= \quad p \mid \neg\phi \mid \phi \rightarrow \phi \mid AX\phi \mid A\phi U\phi \mid E\phi U\phi \mid K_i\phi \mid C_G\phi$$

The formulae $AX\phi$ and $A\phi U\phi'$ (resp. $E\phi U\phi'$) are read as "*for all paths, at the next step* ϕ" and "*for all paths (resp. for some path),* ϕ *until* ϕ'". The formula $K_i\phi$ means "*agent* i *knows* ϕ"; while $C_G\phi$ means "ϕ *is common knowledge in the set* G *of agents*". We define the connectives $\wedge$, $\vee$, $\leftrightarrow$ and the propositional constants **true** and **false** as standard.

The operator X is self-dual, that is, $EX\phi$ is defined as $\neg AX\neg\phi$. The linear time operator $\overline{U}$ is dual to U, that is, $A\phi\overline{U}\phi'$ is defined as $\neg E\neg\phi U\neg\phi'$, and $E\phi\overline{U}\phi'$ as $\neg A\neg\phi U\neg\phi'$. The epistemic possibility $\overline{K}_i$ is dual to K_i for each $i \in A$, that is, $\overline{K}_i\phi$ is defined as $\neg K_i\neg\phi$; while $\overline{C}_G$ is dual to C_G, i.e., $\overline{C}_G\phi$ is a shorthand for $\neg C_G\neg\phi$. Also, the operators AG, AF, EG and EF are defined as standard. Finally, $E_G\phi$ is defined as $\bigwedge_{i\in G} K_i\phi$, and for $n \in \mathbb{N}$, $E_G^0\phi = \phi$ and $E_G^{n+1}\phi = E_G E_G^n\phi$.

The U-formulae in $\mathcal{L}_m$ are the formulae of the form $A\phi U\phi'$ or $E\phi U\phi'$ for some ϕ, $\phi' \in \mathcal{L}_m$; the $\overline{U}$-, K_i- and C_G-formulae are defined similarly.

2.2. Interpreted Systems

We introduce interpreted systems by assuming a set L_i of local states $l_i, l_i', \ldots$ for each agent $i \in A$ in a multi-agent system, as well as for the environment e. The set $\mathcal{S} \subseteq L_e \times L_1 \times \ldots \times L_m$ contains the global states of the MAS. To represent the temporal evolution of the MAS we consider the flow of time $\mathbb{N}$ of the natural numbers. A *run* in an interpreted system is a function $\rho : \mathbb{N} \to \mathcal{S}$ that intuitively represents a possible evolution of the MAS. We define the interpreted systems for the language $\mathcal{L}_m$ as follows.

Definition 2.2. An *interpreted system* (IS) is a tuple $\mathcal{P} = \langle \mathcal{R}, s^0, I \rangle$ where (i) $\mathcal{R}$ is a non-empty set of runs; (ii) $s^0 \in \mathcal{S}$ is the initial state, i.e., $s^0 = \rho(0)$ for some $\rho \in \mathcal{R}$; (iii) $I : \mathcal{S} \to 2^P$ is an assignment for the propositional variables in P.

In what follows we assume without loss of generality that for every $s \in \mathcal{S}$, there exist $\rho \in \mathcal{R}$ and $n \in \mathbb{N}$ such that $s = \rho(n)$, i.e., $\mathcal{S}$ is the set of all reachable states. Following standard notation [5], a pair (ρ, n) is a *point* in $\mathcal{P}$; let Π be the set of all points in $\mathcal{P}$. If $\rho(n) = \langle l_e, l_1, \ldots, l_m \rangle$ is the global state at (ρ, n) then $\rho_e(n) = l_e$ and $\rho_i(n) = l_i$ are the environment's and agent i's local state at (ρ, n) respectively. Further, for $i \in A$ the equivalence relation $\sim_i$ is defined such that $(\rho, n) \sim_i (\rho', n')$ iff $\rho_i(n) = \rho_i'(n')$; while $\rho \mid n$ is the sequence of states $\rho(0), \ldots, \rho(n)$. Sometimes we do not distinguish between a point and the associated state when it is clear from the context.

Definition 2.3. The satisfaction relation $\models$ for $\phi \in \mathcal{L}_m$ and $(\rho, n) \in \mathcal{P}$ is defined as follows:

$(\mathcal{P}, \rho, n) \models p$	iff	$p \in I(\rho(n))$
$(\mathcal{P}, \rho, n) \models \neg\psi$	iff	$(\mathcal{P}, \rho, n) \not\models \psi$
$(\mathcal{P}, \rho, n) \models \psi \to \psi'$	iff	$(\mathcal{P}, \rho, n) \not\models \psi$ or $(\mathcal{P}, \rho, n) \models \psi'$
$(\mathcal{P}, \rho, n) \models AX\psi$	iff	for all runs ρ', $\rho' \mid n = \rho \mid n$ implies $(\mathcal{P}, \rho', n+1) \models \psi$
$(\mathcal{P}, \rho, n) \models A\psi U\psi'$	iff	for all runs ρ', if $\rho' \mid n = \rho \mid n$ then there is $k \geq n$, $(\mathcal{P}, \rho', k) \models \psi'$ and for all k', $n \leq k' < k$ implies $(\mathcal{P}, \rho', k') \models \psi$
$(\mathcal{P}, \rho, n) \models E\psi U\psi'$	iff	for some run ρ', $\rho' \mid n = \rho \mid n$ and there is $k \geq n$, $(\mathcal{P}, \rho', k) \models \psi'$ and for all k', $n \leq k' < k$ implies $(\mathcal{P}, \rho', k') \models \psi$
$(\mathcal{P}, \rho, n) \models K_i\psi$	iff	$(\rho, n) \sim_i (\rho', n')$ implies $(\mathcal{P}, \rho', n') \models \psi$
$(\mathcal{P}, \rho, n) \models C_G\psi$	iff	for all $k \in \mathbb{N}$, $(\mathcal{P}, \rho, n) \models E_G^k\psi$

The truth conditions for $\wedge$, $\vee$, $\leftrightarrow$, **true**, **false**, EX, $A\overline{U}$, $E\overline{U}$, $\overline{K}_i$ and $\overline{C}_G$ are defined from those above. A formula ϕ is *true on a IS* $\mathcal{P}$ iff it is satisfied at $(\rho, 0)$ such that $\rho(0) = s^0$.

3. Epistemic Alternating Tree Automata

In this section we present epistemic alternating tree automata, which are a generalisation of alternating tree automata for multi-modal logics. The latter were first introduced in [18] and have been used in [12] to define an automata-theoretic technique to model check branching time logics. Let A_t be the set $A \cup \{t\}$ containing all the agents in A plus the temporal index t. Hence, $|A_t| = m + 1$. Further, for some set U, U^* is the set of sequences of elements in U.

Definition 3.1. An A_t-tree is a set $T \subseteq (\mathbb{N} \times A_t)^*$ such that if $x \cdot (c,j) \in T$ for $x \in (\mathbb{N} \times A_t)^*$ and $(c,j) \in \mathbb{N} \times A_t$ then

(i) $x \in T$;

(ii) for all $0 \leq c' < c$, also $x \cdot (c',j) \in T$.

The sequences in T are called *nodes*; the empty sequence ϵ is the root of T. For $x \in T$, the nodes $x \cdot (c,j)$ are the *j-successors* of x. The number of j-successors of x is called the *j-degree* of x and is denoted by $d_j(x)$; the vector of all successor degrees of x is denoted by $\vec{d}(x)$. A *leaf* is a node with no successors.

Definition 3.2. A *path* in a tree T is a non-empty set $\pi \subseteq T$ such that for every $x \in \pi$, either x is a leaf or there exists a unique $(c,j) \in \mathbb{N} \times A_t$ such that $x \cdot (c,j) \in \pi$. A *temporal path* is a path where $j = t$; while an *epistemic path in G* is a path where $j \in G$.

For any path π, π^n represents the n-th element in the path, for $n \in \mathbb{N}$. Given an alphabet Σ, a Σ-labelled tree is a pair $\langle T, V \rangle$ where T is a tree and $V : T \to \Sigma$ maps each node of T to a letter in Σ. Note that an infinite word in Σ can be viewed as a Σ-labelled tree in which $|A_t| = 1$ and the degree of all nodes is 1. We focus on Σ-labelled trees in which $\Sigma = \Pi$ for some set Π of points, $\Sigma = \mathcal{S}$ for some set $\mathcal{S}$ of global states, or $\Sigma = 2^P$ so V can intuitively be seen as an assignment of propositional variables to nodes. Given an interpreted system $\mathcal{P} = \langle \mathcal{R}, s^0, I \rangle$ we can define a tree $\langle T_\mathcal{P}, V \rangle$ with $\Sigma = \Pi$ such that $V(\epsilon) = (\rho, 0)$ such that $\rho(0) = s^0$ and

1. if $V(x) = (\rho, n)$ and $s_0, \ldots, s_k$ are all s' such that $s' \sim_i \rho(n)$ for $i \in A$, then for $0 \leq c \leq k$, $V(x \cdot (c,i)) = (\rho_c, n_c)$ for some ρ_c and n_c such that $\rho_c(n_c) = s_c$;

2. if $V(x) = (\rho, n)$ and $\rho_0, \ldots, \rho_k$ are all $\rho' \in \mathcal{R}$ such that $\rho' \mid n = \rho \mid n$, then $V(x \cdot (c,t)) = (\rho_c, n+1)$ for $0 \leq c \leq k$.

We remark that the tree $\langle T_\mathcal{P}, V \rangle$ is not unique given the IS $\mathcal{P}$, as by (1) above for each s_c there might be more than one point (ρ_c, n_c) such that $\rho_c(n_c) = s_c$. However, we can recover uniqueness by defining a mapping $V_\mathcal{P} : T_\mathcal{P} \to \mathcal{S}$ such that $V_\mathcal{P}(x) = s$ iff $V(x) = (\rho, n)$ and $\rho(n) = s$. It is straightforward to check that given an IS $\mathcal{P}$, the labelled tree $\langle T_\mathcal{P}, V_\mathcal{P} \rangle$ is indeed unique.

Furthermore, given a tree $\langle T, V \rangle$ with $\Sigma = 2^P$ we can define a satisfaction relation $\models$ for $\phi \in \mathcal{L}_m$ and $x \in T$ as follows:

$(T,x) \models p$	iff	$p \in V(x)$
$(T,x) \models \neg\psi$	iff	$(T,x) \not\models \psi$
$(T,x) \models \psi \to \psi'$	iff	$(T,x) \not\models \psi$ or $(T,x) \models \psi'$
$(T,x) \models AX\psi$	iff	for all temporal paths π, for all $n \in \mathbb{N}$, if $\pi^n = x$ then $(T, \pi^{n+1}) \models \psi$
$(T,x) \models A\psi U\psi'$	iff	for all temporal paths π, for all $n \in \mathbb{N}$, if $\pi^n = x$ then there is $k \geq n$ such that $(T,\pi^k) \models \psi'$, and for all k', $n \leq k' < k$ implies $(T,\pi^{k'}) \models \psi$
$(T,x) \models E\psi U\psi'$	iff	for some temporal path π, for some $n \in \mathbb{N}$, $\pi^n = x$ and there is $k \geq n$ such that $(T,\pi^k) \models \psi'$, and for all k', $n \leq k' < k$ implies $(T,\pi^{k'}) \models \psi$
$(T,x) \models K_i\psi$	iff	for all $0 \leq c < d_i(x)$, $(T, x \cdot (c,i)) \models \psi$
$(T,x) \models C_G\psi$	iff	for all epistemic paths π in G, for all $n \in \mathbb{N}$, if $\pi^n = x$ then for all $k \geq n$, $(T,\pi^k) \models \psi$

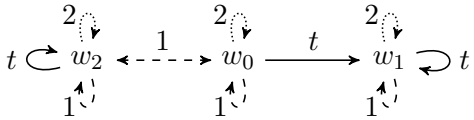

Figure 1. An example of an interpreted system.

Given an IS $\mathcal{P}$, there is a tree $\langle T_{\mathcal{P}}, V\rangle$ with $\Sigma = \Pi$. If we identify each $(\rho, n) \in \Pi$ with $\{p \in P \mid p \in I(\rho(n))\} \in 2^P$, then $\langle T_{\mathcal{P}}, V\rangle$ is also a tree with $\Sigma = 2^P$, and we can prove the following result:

Lemma 3.1. For every $\phi \in \mathcal{L}_m$, for $V(x) = (\rho, n)$,

$$(T_{\mathcal{P}}, x) \models \phi \text{ iff } (\mathcal{P}, \rho, n) \models \phi$$

Lemma 3.1 is proved by induction on the length of ϕ; we omit the proof. In particular, we have that $(T_{\mathcal{P}}, \epsilon) \models \phi$ iff ϕ is true in the IS $\mathcal{P}$.

Obviously, the correspondence between IS and trees is not one-to-one: there are many trees that cannot be represented as IS, this is indeed the case every time a node has no successor. However, this does not hinder the following discussion as we focus on the class $\mathcal{T}$ of trees $\langle T_{\mathcal{P}}, V_{\mathcal{P}}\rangle$ for some interpreted system $\mathcal{P}$.

Example – Unwinding an Interpreted System. A simple interpreted system with two agents 1 and 2 is shown in Figure 1. We use solid lines to represent temporal transitions and indexed, dashed lines to represent the epistemic indistinguishability relations for agents 1 and 2. Formally, this can be represented as the IS $\mathcal{P} = \langle \mathcal{R}, w_0, I\rangle$ such that:

- w_0 is the initial state;
- the local state $l_1(w_0)$ of agent 1 in w_0 is the same as her local state $l_1(w_2)$ in w_2;
- $\mathcal{R} = \{\rho_1, \rho_2\}$ where $\rho_1(0) = w_0$ and $\rho_1(n) = w_1$ for all $n \geq 1$, and $\rho_2(n) = w_2$ for all $n \geq 0$.

Further, we assign the proposition p to all states in $\mathcal{P}$, i.e., for all w, $I(w) = \{p\}$.

Figure 2 shows the $\mathcal{S}$-labelled tree $\langle T_{\mathcal{P}}, V_{\mathcal{P}}\rangle$ unwinding of $\mathcal{P}$. In the tree of Figure 2 each node of the form $x, V(x)$, represents the node of the tree T (i.e., x) along with the mapping in V of that node to a state in $\mathcal{S}$ (i.e., $V(x) \in \mathcal{S}$). The interpreted system of Figure 1 is cyclic (i.e., it contains reflexive loops), therefore its corresponding unwinding is an infinite tree. We only show a truncated part of the tree; the infinite part of the tree is represented with dotted lines. □

We now introduce epistemic alternating tree automata. Given a set $\mathcal{D} \subset \mathbb{N}$, a $\mathcal{D}$-tree is an A_t-tree in which all the nodes have degrees in $\mathcal{D}$. Further, let $\mathcal{B}^+(P)$ be the set of positive Boolean formulae over the set P of propositional variables. For instance, $\mathcal{B}^+(\{p, q\})$ includes $p \wedge q$, $p \vee q$, $p \wedge p$.

Definition 3.3. An *epistemic alternating tree automaton* (EATA) is a tuple $\mathcal{A} = \langle \Sigma, \mathcal{D}, Q, \delta, q_0, A_t, F\rangle$ such that:

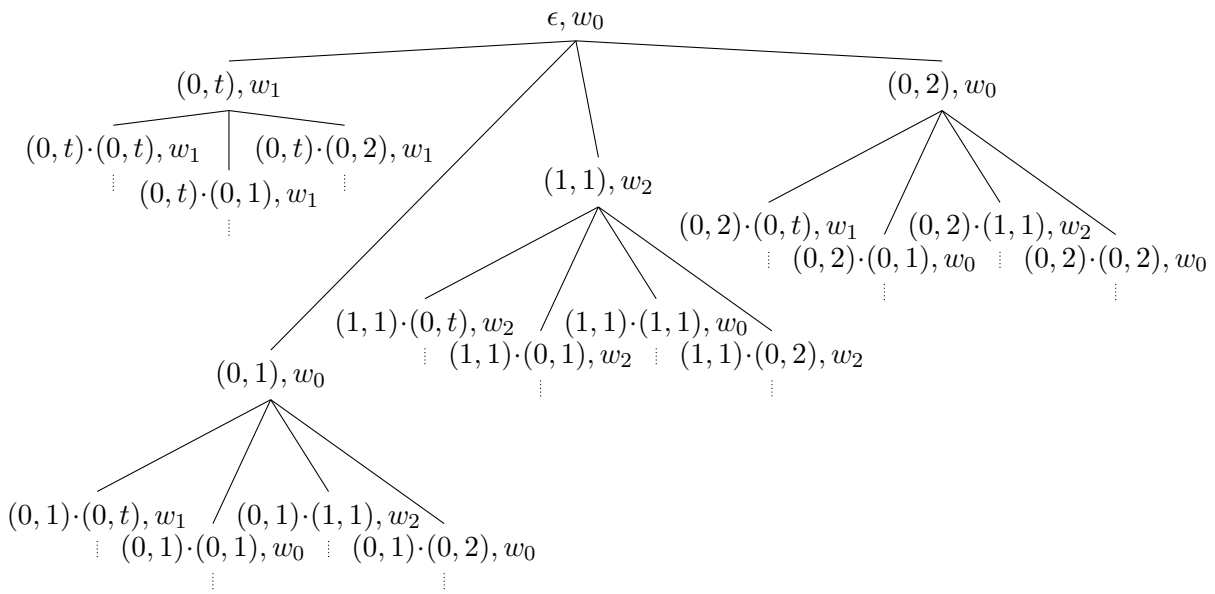

Figure 2. The epistemic alternating tree automata "unwinding" of the interpreted system in Figure 1.

(i) Σ, $\mathcal{D}$ and A_t are defined as above;

(ii) Q is a set of states;

(iii) $q_0 \in Q$ is the initial state;

(iv) $F \subseteq Q$ is the set of accepting states;

(v) $\delta : Q \times \Sigma \times \mathcal{D}^{|A_t|} \to \mathcal{B}^+(\mathbb{N} \times A_t \times Q)$ is the transition function.

Also, we require that if the atom (c, j, q') appears in $\delta(q, \sigma, k_t, k_1, \ldots, k_m)$, then $0 \leq c < k_j$.

When the automaton is in state q and reads a node that is labelled by σ and has k_j j-successors, it applies the transition $\delta(q, \sigma, k_t, k_1, \ldots, k_m)$. In what follows we denote the tuple $\langle k_t, k_1, \ldots, k_m \rangle$ as $\vec{k}$.

An EATA is a generalisation of an alternating tree automaton in that the successors of a node are indexed according to the elements in A_t. ATA [12] are a special case of EATA, in which $|A_t| = 1$, i.e., A_t contains only the temporal index t, which at this point can be ignored.

A run of an EATA $\mathcal{A}$ over a tree $\langle T, V \rangle$ is a tree $\langle T_r, r \rangle$ in which the root is labelled by (ϵ, q_0) and every other node is labelled by an element in $(\mathbb{N} \times A_t)^* \times Q$. Each node of T_r corresponds to a node of T. However, each node of T can correspond to many nodes of T_r.

Definition 3.4. A *run* $\langle T_r, r \rangle$ is a Σ_r-labelled tree where $\Sigma_r = (\mathbb{N} \times A_t)^* \times Q$ and $\langle T_r, r \rangle$ satisfies the following:

(i) $r(\epsilon) = (\epsilon, q_0)$

(ii) Let $y \in T_r$ with $r(y) = (x, q)$. If $\delta(q, V(x), \vec{d}(x)) = \theta \in \mathcal{B}^+(\mathbb{N} \times A_t \times Q)$ then there are (possibly empty) sets $S_j = \{(c_0, j, q_0), \ldots, (c_n, j, q_n)\} \subseteq \{0, \ldots, d_j(x) - 1\} \times \{j\} \times Q$ such that the following hold:

(a) the assignment which assigns **true** to all the atoms in $\bigcup_{j \in A_t} S_j$ satisfies θ;

(b) for $0 \leq i \leq n$ we have $y \cdot (i, j) \in T_r$ and $r(y \cdot (i, j)) = (x \cdot (c_i, j), q_i)$.

Note that if, for some y, the transition function δ has value **true**, then y need not have successors. Also, δ can never have the value **false** in a run. We use the same term *run* both for IS and for EATA to be consistent with [5, 12]; the context will disambiguate.

A run $\langle T_r, r \rangle$ is *accepting* if all its infinite paths satisfy the acceptance condition. Here we consider a Büchi acceptance condition. Given a run $\langle T_r, r \rangle$ and an infinite path $\pi \subseteq T_r$, let $inf(\pi) \subseteq Q$ be the set of $q \in Q$ such that there are infinitely many $y \in \pi$ for which $r(y) \in (\mathbb{N} \times A_t)^* \times \{q\}$, that is, $inf(\pi)$ contains exactly all the states that appear infinitely often in π. The acceptance condition is defined as follows:

- A path π satisfies a Büchi acceptance condition $F \subseteq Q$ if and only if $inf(\pi) \cap F \neq \emptyset$.

An automaton accepts a tree if and only if there exists a run that accepts it. We denote by $\mathcal{L}(\mathcal{A})$ the set of all Σ-labelled trees that $\mathcal{A}$ accepts. Note that an epistemic alternating tree automaton over infinite words is simply an EATA with $\mathcal{D} = \{1\}$ and $|A_t| = 1$. Formally, we define an EATA over infinite words as $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, F \rangle$ where $\delta : Q \times \Sigma \to \mathcal{B}^+(Q)$.

The model checking procedure for CTLK considers weak epistemic alternating automata (WEAAs), an extension of weak alternating automata as first introduced in [17].

Definition 3.5. A weak epistemic alternating automata (WEAA) is an EATA such that:

- there is a partition of Q into disjoint sets $Q_1, \ldots, Q_n$ such that for $1 \leq j \leq n$, either $Q_j \subseteq F$, and Q_j is an accepting set, or $Q_j \cap F = \emptyset$, and Q_j is a rejecting set.

- there is a partial order $\leq$ on the collection of the Q_js such that for every $q \in Q_i$ and $q' \in Q_j$ occurring in $\delta(q, \sigma, \vec{k})$ for some $\sigma \in \Sigma$, $\vec{k} \in \mathcal{D}^{|A_t|}$, we have $Q_j \leq Q_i$.

Thus, transitions from a state in Q_i lead to states in either the same Q_i or a lower one. It follows that every infinite path of a run of a WEAA ultimately gets trapped within some Q_i. The path then satisfies the acceptance condition if and only if Q_i is an accepting set. We call the partition of Q into sets the *weakness partition*, and we call the partial order over the sets of the weakness partition the *weakness order*.

3.1. Model Checking for CTLK

In this section we provide the construction of a weak alternating automaton $\mathcal{A}_{\mathcal{D},\psi}$ that accepts all the $\mathcal{D}$-trees in $\mathcal{T}$ satisfying a given CTLK formula $\psi \in \mathcal{L}_m$. In the next section the automaton $\mathcal{A}_{\mathcal{D},\psi}$ will be used to construct the product word automaton $\mathcal{A}_{\mathcal{P},\psi}$ for a given IS $\mathcal{P}$. We will then prove that the language $\mathcal{L}(\mathcal{A}_{\mathcal{P},\psi})$ is non-empty iff the tree $\langle T_{\mathcal{P}}, V_{\mathcal{P}} \rangle$ is accepted by $\mathcal{A}_{\mathcal{D},\psi}$, i.e., iff ψ is true in $\mathcal{P}$. By extending Theorems 3.1 and 4.7 in [12] we can show that all these steps can be performed in linear time in the size of ϕ and $\mathcal{P}$.

First, we remark that by using de Morgan's laws and the definitions of operators $\overline{U}$, $\overline{K}_i$ and $\overline{C}_G$, we can draw the negation inwards, such that it applies only to propositional variables.

Further, we define the closure $cl(\psi)$ of a formula $\psi \in \mathcal{L}_m$ as follows:

- $\psi \in cl(\psi)$;
- Let $\Box$ be any of operators $\neg$, AX, EX, K_i, $\overline{K}_i$, C_G or $\overline{C}_G$. If $\Box\phi \in cl(\psi)$ then $\phi \in cl(\psi)$.
- Let $\sharp$ be any of operators $\rightarrow$, AU, EU, $A\overline{U}$ or $E\overline{U}$. If $\phi\sharp\phi' \in cl(\psi)$ then ϕ, $\phi' \in cl(\psi)$.

Theorem 3.1. Given a CTLK formula $\psi \in \mathcal{L}_m$ and a set $\mathcal{D} \subset \mathbb{N}$, we can construct in linear time a WEAA $\mathcal{A}_{\mathcal{D},\psi} = \langle 2^P, \mathcal{D}, cl(\psi), \delta, \psi, A_t, F\rangle$ such that the $\mathcal{D}$-tree $\langle T_\mathcal{P}, V_\mathcal{P}\rangle$ is in $\mathcal{L}(\mathcal{A}_{\mathcal{D},\psi})$ iff ψ is true in $\mathcal{P}$.

Proof:
The WEAA $\mathcal{A}_{\mathcal{D},\psi} = \langle 2^P, \mathcal{D}, cl(\psi), \delta, \psi, A_t, F\rangle$ is such that $\Sigma = 2^P$, $Q = cl(\psi)$ and $q_0 = \psi$. Further, the set F of accepting states consists of all the $\overline{U}$-, K_i- and C_G-formulae in $cl(\psi)$. For $\sigma \in 2^P$ and $\vec{k} \in \mathcal{D}^{|A_t|}$ the transition function δ is defined as in Table 1. Each formula $\phi \in cl(\psi)$ constitutes a (singleton) set $\{\phi\}$ in the weakness partition. The weakness order is defined by $\{\phi_1\} \leq \{\phi_2\}$ iff $\phi_1 \in cl(\phi_2)$. Since each transition of the automaton from a state ϕ leads to states in $cl(\phi)$, the weakness conditions hold.

We now prove the correctness of this construction. By Lemma 3.1 the formula ψ is true in $\mathcal{P}$ iff $\langle T_\mathcal{P}, \epsilon\rangle \models \psi$. So it is left to prove that the $\mathcal{D}$-tree $\langle T_\mathcal{P}, V_\mathcal{P}\rangle$ is in $\mathcal{L}(\mathcal{A}_{\mathcal{D},\psi})$ iff $\langle T_\mathcal{P}, \epsilon\rangle \models \psi$. We first prove that $\mathcal{A}_{\mathcal{D},\psi}$ is sound, that is, given an accepting run $\langle T_r, r\rangle$ of $\mathcal{A}_{\mathcal{D},\psi}$ over the tree $\langle T_\mathcal{P}, V_\mathcal{P}\rangle$, we show that for every $y \in T_r$ such that $r(y) = (x, \phi)$ we have that $\langle T_\mathcal{P}, x\rangle \models \phi$. Thus, in particular $\langle T_\mathcal{P}, \epsilon\rangle \models \psi$. The proof is by induction on the structure of ϕ. If ϕ is an atomic proposition and $r(y) = (x, p)$ then $\delta(p, V_\mathcal{P}(x), \vec{d}(x)) = \textbf{true}$ iff $p \in V_\mathcal{P}(x)$, i.e., iff $\langle T_\mathcal{P}, x\rangle \models \phi$. The cases where ϕ is $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$, $AX\phi_1$, or $EX\phi_1$ follow easily, by the induction hypothesis, from the definition of δ. If ϕ is $K_i\phi_1$ and $r(y) = (x, K_i\phi_1)$ then $\delta(K_i\phi_1, V_\mathcal{P}(x), \vec{d}(x)) = \bigwedge_{c=0}^{d_i(x)-1}(c, i, \phi_1) \wedge \bigwedge_{c=0}^{d_i(x)-1}(c, i, K_i\phi_1)$. Thus, for all $0 \leq k < d_i(x)$ and $c = k$ we have $r(y\cdot(k, i)) = (x\cdot(c, i), \phi_1)$. By induction hypothesis, $\langle T_\mathcal{P}, x\cdot(c, i)\rangle \models \phi_1$ for $0 \leq c < d_i(x)$, and therefore $\langle T_\mathcal{P}, x\rangle \models K_i\phi_1$. The case of $\phi = \overline{K}_i\phi_1$ is similar. If ϕ is equal to $C_G\phi$ and $r(y) = (x, C_G\phi)$ then $\delta(C_G\phi, V_\mathcal{P}(x), \vec{d}(x)) = \delta(\phi, V_\mathcal{P}(x), \vec{d}(x)) \wedge \bigwedge_{i\in G}\bigwedge_{c=0}^{k_i-1}(c, i, C_G\phi)$. Thus, if π is an epistemic path for G such that $\pi^n = x$, then by induction hypothesis for all $k \geq n$, we have that $\langle T_\mathcal{P}, \pi^k\rangle \models \phi$. Therefore, $\langle T_\mathcal{P}, x\rangle \models C_G\phi$. The case of $\phi = \overline{C}_G\phi_1$ is similar. Consider now the case of ϕ equal to $A\phi_1 U\phi_2$ (resp. $E\phi_1 U\phi_2$). As $\langle T_r, r\rangle$ is an accepting run, it visits the state ϕ only finitely often. Since $\mathcal{A}_{\mathcal{D},\psi}$ keeps inheriting ϕ as long as ϕ_2 is not satisfied, then it is guaranteed, by the definition of δ and the induction hypothesis, that along all paths (resp. some path) ϕ_2 eventually holds and ϕ_1 holds until then. Finally, consider the case of ϕ equal to $A\phi_1\overline{U}\phi_2$ or $E\phi_1\overline{U}\phi_2$. By the definition of δ and the induction hypothesis, either ϕ_2 always holds or until both ϕ_2 and ϕ_1 hold.

We now prove that $\mathcal{A}_{\mathcal{D},\psi}$ is complete, that is, if $\langle T_\mathcal{P}, V_\mathcal{P}\rangle$ is a $\mathcal{D}$-tree such that $\langle T_\mathcal{P}, \epsilon\rangle \models \psi$, then $\mathcal{A}_{\mathcal{D},\psi}$ accepts $\langle T_\mathcal{P}, V_\mathcal{P}\rangle$. In fact, we show that there exists an accepting run $\langle T_r, r\rangle$ of $\mathcal{A}_{\mathcal{D},\psi}$ over $\langle T_\mathcal{P}, V_\mathcal{P}\rangle$ defined as follows: the run starts at the initial state, so $\epsilon \in T_r$ and $r(\epsilon) = (\epsilon, \psi)$, and it proceeds maintaining the invariant that for every $y \in T_r$, if $r(y) = (x, \phi)$ then $\langle T_\mathcal{P}, x\rangle \models \phi$. Since $\langle T_\mathcal{P}, \epsilon\rangle \models \psi$, the invariant holds for $y = \epsilon$. Also, by the semantics of CTLK and the definition of δ, the run can always proceed such that all the successors $y \cdot (k, j)$ of a node y that satisfies the invariant have $r(y \cdot (k, j)) = (x', \phi')$ with $\langle T_\mathcal{P}, x'\rangle \models \phi'$. Finally, the run always tries to satisfy eventualities of U-formulae. Thus, whenever ϕ is

$$
\begin{array}{rcl}
\delta\left(p,\sigma,\vec{k}\right) & = & \textbf{true} \text{ if } p \in \sigma \\
\delta\left(p,\sigma,\vec{k}\right) & = & \textbf{false} \text{ if } p \notin \sigma \\
\delta\left(\neg p,\sigma,\vec{k}\right) & = & \textbf{true} \text{ if } p \notin \sigma \\
\delta\left(\neg p,\sigma,\vec{k}\right) & = & \textbf{false} \text{ if } p \in \sigma \\
\delta\left(\phi_1 \star \phi_2,\sigma,\vec{k}\right) & = & \delta\left(\phi_1,\sigma,\vec{k}\right) \star \delta\left(\phi_2,\sigma,\vec{k}\right), \text{for} \star \in \{\wedge,\vee\} \\
\delta\left(AX\phi,\sigma,\vec{k}\right) & = & \bigwedge\limits_{c=0}^{k_t-1}(c,t,\phi) \\
\delta\left(EX\phi,\sigma,\vec{k}\right) & = & \bigvee\limits_{c=0}^{k_t-1}(c,t,\phi) \\
\delta\left(A\phi_1 U\phi_2,\sigma,\vec{k}\right) & = & \delta\left(\phi_2,\sigma,\vec{k}\right) \vee \left(\delta\left(\phi_1,\sigma,\vec{k}\right) \wedge \bigwedge\limits_{c=0}^{k_t-1}(c,t,A\phi_1 U\phi_2)\right) \\
\delta\left(E\phi_1 U\phi_2,\sigma,\vec{k}\right) & = & \delta\left(\phi_2,\sigma,\vec{k}\right) \vee \left(\delta\left(\phi_1,\sigma,\vec{k}\right) \wedge \bigvee\limits_{c=0}^{k_t-1}(c,t,E\phi_1 U\phi_2)\right) \\
\delta\left(A\phi_1 \overline{U}\phi_2,\sigma,\vec{k}\right) & = & \delta\left(\phi_2,\sigma,\vec{k}\right) \wedge \left(\delta\left(\phi_1,\sigma,\vec{k}\right) \vee \bigwedge\limits_{c=0}^{k_t-1}\left(c,t,A\phi_1 \overline{U}\phi_2\right)\right) \\
\delta\left(E\phi_1 \overline{U}\phi_2,\sigma,\vec{k}\right) & = & \delta\left(\phi_2,\sigma,\vec{k}\right) \wedge \left(\delta\left(\phi_1,\sigma,\vec{k}\right) \vee \bigvee\limits_{c=0}^{k_t-1}\left(c,t,E\phi_1 \overline{U}\phi_2\right)\right) \\
\delta\left(K_i\phi,\sigma,\vec{k}\right) & = & \bigwedge\limits_{c=0}^{k_i-1}(c,i,\phi) \wedge \bigwedge\limits_{c=0}^{k_i-1}(c,i,K_i\phi) \\
\delta\left(\overline{K}_i\phi,\sigma,\vec{k}\right) & = & \bigvee\limits_{c=0}^{k_i-1}(c,i,\phi) \\
\delta\left(C_G\phi,\sigma,\vec{k}\right) & = & \delta\left(\phi,\sigma,\vec{k}\right) \wedge \bigwedge\limits_{i\in G}\bigwedge\limits_{c=0}^{k_i-1}(c,i,C_G\phi) \\
\delta\left(\overline{C}_G\phi,\sigma,\vec{k}\right) & = & \delta\left(\phi,\sigma,\vec{k}\right) \vee \bigvee\limits_{i\in G}\bigvee\limits_{c=0}^{k_i-1}\left(c,i,\overline{C}_G\phi\right)
\end{array}
$$

Table 1. The transition function δ in the automaton $\mathcal{A}_{\mathcal{D},\psi}$.

of the form $A\phi_1 U \phi_2$ or $E\phi_1 U \phi_2$ and $\langle T_{\mathcal{P}}, x\rangle \models \phi_2$, it proceeds according to $\delta(\phi_2, V_{\mathcal{P}}(x), \vec{d}(x))$. It is easy to see that all the paths in such $\langle T_r, r\rangle$ are either finite or reach a state associated with a $\overline{U}$-, K_i- or C_G-formula and stay there thereafter. Thus, $\langle T_r, r\rangle$ is accepting.

Finally, it is easy to check that the construction of the automaton $\mathcal{A}_{\mathcal{D},\psi}$ can be performed in linear time in the size of ψ. □

Note that the particular way the δ function is defined for $K_i\phi$, and the loss of symmetry with $\overline{K}_i\phi$, depends on the fact that the epistemic indistinguishability relation is an equivalence relation. Also, the δ function for the epistemic E_G modality can be straightforwardly defined in terms of the δ function for K_i, for $i \in G$.

Example – $\mathcal{L}_m$ to WEAA. Here we demonstrate the technique by showing the translation of a $\mathcal{L}_m$ formula to a weak epistemic alternating automaton. We take the formula $\varphi = AGC_{\{1,2\}}K_2p$. To translate φ into a WEAA, we require the formula in negation-normal form with all abbreviations expanded; so we use $\varphi = A\left(\mathbf{false}\overline{U}C_{\{1,2\}}K_2p\right)$. The closure of φ is $cl(\varphi) = \left\{\varphi, C_{\{1,2\}}K_2p, K_2p, p\right\}$, which is the set Q of states in $\mathcal{A}_{\mathcal{D},\varphi}$. The accepting states F are $\left\{\varphi, C_{\{1,2\}}K_2p, K_2p\right\}$. The alphabet of $\mathcal{A}_{\mathcal{D},\varphi}$ has only the letter p; therefore the transitions are over $2^{\{p\}}$.

We formally define $\mathcal{A}_{\mathcal{D},\varphi} = (2^{\{p\}}, \mathcal{D}, \left\{\varphi, C_{\{1,2\}}K_2p, K_2p, p\right\}, \delta, \varphi, \{t, 1, 2\}, \left\{\varphi, C_{\{1,2\}}K_2p, K_2p\right\})$, where the transition relation δ is as follows:

q	$\delta\left(q, \{p\}, \vec{k}\right)$	$\delta\left(q, \emptyset, \vec{k}\right)$
φ	$\bigwedge_{c=0}^{k_t-1}(c,t,\varphi) \wedge \bigwedge_{c=0}^{k_2-1}(c,2,p) \wedge \bigwedge_{c=0}^{k_2-1}(c,2,K_2p) \wedge \bigwedge_{i\in\{1,2\}}\bigwedge_{c=0}^{k_i-1}\left(c,i,C_{\{1,2\}}K_2p\right)$	
$C_{\{1,2\}}K_2p$	$\bigwedge_{c=0}^{k_2-1}(c,2,p) \wedge \bigwedge_{c=0}^{k_2-1}(c,2,K_2p) \wedge \bigwedge_{i\in\{1,2\}}\bigwedge_{c=0}^{k_i-1}\left(c,i,C_{\{1,2\}}K_2p\right)$	
K_2p	$\bigwedge_{c=0}^{k_2-1}(c,2,p) \wedge \bigwedge_{c=0}^{k_2-1}(c,2,K_2p)$	
p	**true**	**false**

In the state φ the automata expects that φ recursively holds in all temporal successors and that both K_2p and $C_{\{1,2\}}K_2p$ hold in all epistemically related states. This is highlighted in Rows 2 and 3 of the transition relation above.

3.2. The Product Automaton

Let $\mathcal{A}_{\mathcal{D},\psi} = \langle 2^P, \mathcal{D}, Q_\psi, \delta_\psi, q_0, A_t, F_\psi\rangle$ be an epistemic alternating tree automaton that accepts all the $\mathcal{D}$-trees in $\mathcal{T}$ that satisfy ψ, as constructed in the previous section. Let $\mathcal{P} = \langle \mathcal{R}, s^0, I\rangle$ be an interpreted system such that the degrees of $\langle T_{\mathcal{P}}, V_{\mathcal{P}}\rangle$ are in $\mathcal{D}$. We introduce the weak epistemic alternating word automaton $\mathcal{A}_{\mathcal{P},\psi} = \langle\{a\}, \Pi \times Q_\psi, \delta, ((\rho,0), q_0), F\rangle$ such that $\rho(0) = s_0$ and δ, F are defined as follows:

- Let $q \in Q_\psi$, $(\rho, n) \in \Pi$, $\{s' \in \mathcal{S} \mid s' \sim_i \rho(n)\} = \langle s_{0,i}, \ldots, s_{d_i(\rho(n))-1,i}\rangle$ and $\{\rho' \in \mathcal{R} \mid \rho' \mid n = \rho \mid n\} = \langle \rho_{0,t}, \ldots, \rho_{d_t(\rho(n))-1,t}\rangle$. Further, let $\delta_\psi(q, I(\rho(n)), \vec{d}(\rho(n))) = \theta$. Then $\delta(((\rho,n),q),a) = \theta'$, where θ' is obtained from θ by replacing each atom (c_j, i, q_j) in θ by the atom $((\rho_{c_j,i}, n_{c_j,i}), q_j)$ for some point $(\rho_{c_j,i}, n_{c_j,i})$ such that $\rho_{c_j,i}(n_{c_j,i}) = s_{c_j,i}$, and by replacing each atom (c_j, t, q_j) in θ by the atom $((\rho_{c_j,t}, n+1), q_j)$.

- The acceptance condition F is defined according to the acceptance condition F_ψ of $\mathcal{A}_{\mathcal{D},\psi}$. If $F_\psi \subseteq Q_\psi$ is a Büchi condition, then $F = \Pi \times F_\psi$ is also a Büchi condition.

It is easy to see that if $\mathcal{A}_{\mathcal{D},\psi}$ is a WEAA with a weakness partition $\{Q_1, \ldots, Q_n\}$, then so is $\mathcal{A}_{\mathcal{P},\psi}$ with a partition $\{\Pi \times Q_1, \ldots, \Pi \times Q_n\}$.

We remark that the word automaton $\mathcal{A}_{\mathcal{P},\psi}$ defined above is not unique given $\mathcal{P}$. However, we can recover uniqueness by considering states in $\mathcal{S}$ rather than points in Π. Thus, the product automaton of $\mathcal{A}_{\mathcal{D},\psi}$ and $\mathcal{P}$ is defined as the weak epistemic alternating word automaton $\mathcal{A}_{\mathcal{P},\psi} = \langle \{a\}, \mathcal{S} \times Q_\psi, \delta, (s^0, q_0), F\rangle$ where $F = \mathcal{S} \times F_\psi$ and if $\delta(((\rho, n), q), a) = \theta$ the $\delta((\rho(n), q), a) = \theta'$, where θ' is obtained from θ by replacing each atom $((\rho, n), q)$ in θ by $(\rho(n), q)$.

Theorem 3.2. $\mathcal{L}(\mathcal{A}_{\mathcal{P},\psi})$ is nonempty iff ψ is true in $\mathcal{P}$.

Proof. We show that $\mathcal{L}(\mathcal{A}_{\mathcal{P},\psi})$ is nonempty if and only if $\mathcal{A}_{\mathcal{D},\psi}$ accepts the tree $\langle T_\mathcal{P}, V_\mathcal{P}\rangle$ built from the IS $\mathcal{P}$ as shown in Section 3.1. Since $\mathcal{A}_{\mathcal{D},\psi}$ accepts exactly all the $\mathcal{D}$-trees in $\mathcal{T}$ that satisfy ψ, and since all the degrees of $\mathcal{P}$ are in $\mathcal{D}$, the latter holds if and only if ψ is true in $\mathcal{P}$. Given an accepting run of $\mathcal{A}_{\mathcal{D},\psi}$ over $\langle T_\mathcal{P}, V_\mathcal{P}\rangle$, we construct an accepting run of $\mathcal{A}_{\mathcal{P},\psi}$. Also, given an accepting run of $\mathcal{A}_{\mathcal{P},\psi}$, we construct an accepting run of $\mathcal{A}_{\mathcal{D},\psi}$ over $\langle T_\mathcal{P}, V_\mathcal{P}\rangle$.

Assume first that $\mathcal{A}_{\mathcal{D},\psi}$ accepts $\langle T_\mathcal{P}, V_\mathcal{P}\rangle$. Thus, there exists an accepting run $\langle T_r, r\rangle$ of $\mathcal{A}_{\mathcal{D},\psi}$ over $\langle T_\mathcal{P}, V_\mathcal{P}\rangle$. Recall that T_r is labelled with $(\mathbb{N} \times A_t)^* \times Q_\psi$. A node $y \in T_r$ with $r(y) = (x, q)$ corresponds to a copy of $\mathcal{A}_{\mathcal{D},\psi}$ that is in the state q and reads the tree obtained by unwinding $\mathcal{P}$ from $V_\mathcal{P}(x)$. Consider the tree $\langle T_{r'}, r'\rangle$ where $T_{r'}$ is the tree obtained from T_r by the function f as follows. Suppose that $\delta_\psi(q, V_\mathcal{P}(x), \vec{d}(x)) = \theta$ and there are (possibly empty) sets $S_j = \{(c_0, j, q_0), \ldots, (c_{n_j}, j, q_{n_j})\} \subseteq \{0, \ldots, d_j(x) - 1\} \times \{j\} \times Q$ such that $\bigcup_{j \in A_t} S_j$ satisfies θ, and for $0 \leq i < n_j$, we have $y \cdot (i, j) \in T_r$ and $r(y \cdot (i, j)) = (x \cdot (c_i, j), q_i)$. Then,

- $f(\epsilon) = \epsilon$;
- $f(y \cdot (i, j)) = f(y) \cdot (\Sigma_{j' < j}^{j \in A_t} n_{j'} + i)$.

The tree $T_{r'}$ is labelled with $0^* \times \mathcal{S} \times Q$, and for every $y \in T_r$ with $r(y) = (x, q)$, we have $r'(f(y)) = (0^{|x|}, V_\mathcal{P}(x), q)$. We show that $\langle T_{r'}, r'\rangle$ is an accepting run of $\mathcal{A}_{\mathcal{P},\psi}$. In fact, since $F = \mathcal{S} \times F_\psi$, we only need to show that $\langle T_{r'}, r'\rangle$ is a run of $\mathcal{A}_{\mathcal{P},\psi}$; this follows from the definition of δ. Acceptance follows from the fact that $\langle T_r, r\rangle$ is accepting.

Assume now that $\mathcal{A}_{\mathcal{P},\psi}$ accepts a^ω. Thus, there exists an accepting run $\langle T_r, r\rangle$ of $\mathcal{A}_{\mathcal{P},\psi}$. Recall that T_r is labelled with $0^* \times \mathcal{S} \times Q_\psi$. Consider the tree $\langle T_{r'}, r'\rangle$ labelled with $(\mathbb{N} \times A_t)^* \times Q_\psi$, where $T_{r'}$ and r' are obtained from T_r and r by means of a function $g : T_r \to T_{r'}$ as follows:

- $g(\epsilon) = \epsilon$ and $r'(\epsilon) = (\epsilon, q_0)$;
- if $y \cdot c \in T_r$, $r'(g(y)) \in \{x\} \times Q_\psi$, $r(y \cdot c) = (0^{|x+1|}, s, q)$ and i, j are such that $V_\mathcal{P}(x \cdot (i, j)) = s$, then $g(y \cdot c) = g(y) \cdot (i, j)$ and $r'(g(y \cdot c)) = (x \cdot (i, j), q)$.

As in the previous direction, we can check that $\langle T_{r'}, r'\rangle$ is an accepting run of $\mathcal{A}_{\mathcal{D},\psi}$ over $\langle T_\mathcal{P}, V_\mathcal{P}\rangle$. □

By Theorem 4.7 in [12] we know that the 1-letter non-emptiness problem for weak alternating automata is decidable in linear time. This concludes the automata-theoretic model checking procedure for CTLK.

Example – The product automata $\mathcal{A}_{\mathcal{P},\varphi}$. Using the approach developed so far, it can be shown that the language of the product automaton obtained from the composition of $\mathcal{A}_{\mathcal{D},\varphi}$ and the tree unwinding of the IS from Figure 1 is non-empty.

Figure 3 shows a sub-tree of the full product automata. This sub-tree shows the accepting sub-tree for the formula $AX\left(A\left[false\overline{U}C_{\{1,2\}}\left(K_2\left(p\right)\right)\right]\right)$ starting at the world w_1 from Figure 1. While this sub-tree contains 13 nodes, the full product automata contains 31 nodes and has been omitted from brevity.

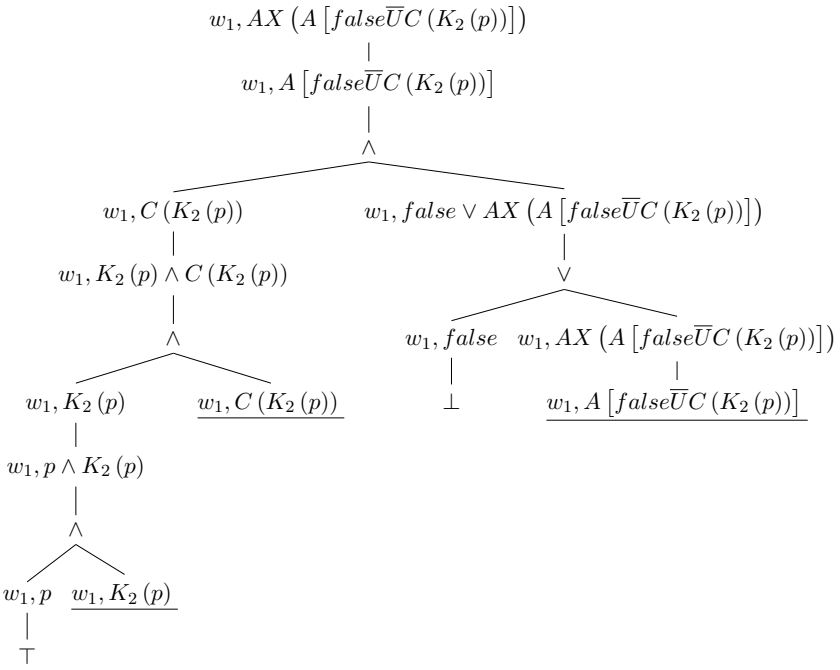

Figure 3. A sub-tree of the automata $\mathcal{A}_{\mathcal{P},\varphi}$.

Branches that reach a recurring node (e.g., the node "$w_1, K_2\left(p\right)$" that occurs twice in Figure 3) are underlined. The nodes that appear between the first and second occurrence of a recurring node make a path in the tree that is subsequently checked against the acceptance condition. For example, the set of states that occur infinitely often between the first and second occurrence of the node "$w_1, K_2\left(p\right)$" are "$w_1, K_2\left(p\right)$" and "$w_1, p \wedge K_2\left(p\right)$". This path is accepted as the intersection of this path and accepting states $F = \left\{\varphi, C_{\{1,2\}}K_2p, K_2p\right\}$ of the formula φ is non-empty.

4. Implementation and Evaluation

4.1. An Epistemic Tree Automata Verifier

We have implemented the technique above in C++ as part of a new, explicit-state model checker called ETAV (Epistemic Tree Automata Verifier). Currently, ETAV only supports models specified directly as Kripke structures, with all relations explicitly constructed, i.e., the full state space has to be enumerated prior to verification. An open source, GNU GPL-licenced release of ETAV is available from [1].

In the following, we use $X\downarrow_n$ to represent the n-th element of the tuple X. Additionally, we use $\bot$ and $\top$ to represent the evaluation of a node in an AND/OR graph to either **true** or **false** [12]. We use $\wp(A)$ to represent the power-set of possible groups.

4.1.1. Approach

The approach taken by ETAV is to perform depth first construction of $\mathcal{A}_{\mathcal{P},\psi}$, constructed as an AND/OR graph, interleaved with checking the non-emptiness of the tree. If it can be decided that the tree is accepting (or rejecting) without constructing the full product automata, ETAV will return this result early and save on over computation.

The crux of ETAV's construction of product automata relies upon the following structures:

- $visited : Formula \times World \rightarrow Bool$
- $eval : Formula \times World \rightarrow \{\top, \bot\}$
- $path : (Formula \times World)^+$
- $tf : Formula \rightarrow (Formula)^+ \times Node\_Type \times \{d \cup \circlearrowleft\}, d \in \{\wp(A) \setminus \emptyset\} \cup t$

The data structure $visited$, implemented using `std::multiset`, holds a set of nodes of $\mathcal{A}_{\mathcal{P},\psi}$ visited on a certain path. If a newly constructed node in the product automata is already in $visited$, then a cycle has been detected. Once a path in the tree reaches a node which evaluates to either $\bot$ or $\top$, or completes a cycle, that node is removed from $visited$ and added to $eval$ along with its evaluation.

In a similar way, $eval$, implemented with `std::map`, records the evaluation of previously seen nodes. This saves re-evaluating a formula at a given world, or searching for a cycle when one has already been detected. If a node has been previously explored, it will have a definitive value; ETAV can simply reuse that value from $eval$.

The list $path$ records all of the nodes on a path of the tree in the order that they appear. The acceptance of a path-suffix depends upon the non-emptiness of the intersection between the states occurring in the path-suffix and the acceptance condition. The path-suffix can be created by iterating backwards along $path$ until the cycle is found.

Finally, tf holds the encoding of the transition function δ. Taking inspiration from [19], we use a simplified transition relation in which rules are labelled with $\wedge$, $\vee$, $\top$ or $\bot$, representing a node-type in the product AND/OR graph. To support multiple directions, i.e., a direction in $G \subseteq A$ or t, tf either returns a member of $\wp(A) \setminus \emptyset$ (the set of all non-empty groups), t or $\circlearrowleft$. We use the first to locate (possibly many) R_i, the second to locate R_t; the latter represents evaluation at the current state. When $tf(\varphi)\downarrow_3 \in \{\wp(A) \setminus \emptyset\} \cup t$, we have $|tf(\varphi)\downarrow_1| = 1$, otherwise $|tf(\varphi)\downarrow_1| = 2$. If the number of successors in R_i or R_t for a given world is greater than two, the successors are iterated over and the AND/OR node is constructed in the intuitive manner. For example, if $\phi = E\left[\varphi \overline{U} \psi\right]$, then $tf(\phi)$ returns the tuple $((\psi, \varphi \vee EX\phi), \wedge, \circlearrowleft)$. This means that the current state must satisfy the conjunction of ψ and $\varphi \vee EX\phi$. It follows that $tf(K_i\varphi)\downarrow_3 = i$, $tf(C_G\varphi)\downarrow_3 = G$ ($G \subseteq A$) and $tf(AX\varphi)\downarrow_3 = t$. For $\phi \in P$, we have $tf(\phi)\downarrow_2 \in \{\top, \bot\}$.

The depth first construction is called recursively for all elements in $tf(\varphi)\downarrow_1$ until $tf(\varphi')\downarrow_2 \in \{\top, \bot\}$. This result is then stored in $eval$ and is also used to label the current node in the AND/OR graph of the product automata. Eventually, the procedure returns with the root of the AND/OR graph being labelled with $\top$ or $\bot$.

4.1.2. Efficiency

ETAV builds the product automata in such a way that it only constructs the parts of the product automata that are required for deciding the satisfiability of the formula. The $eval$ structure is used to remove the possibility of over computation. It is assumed that storing the acceptance or rejection of a node in $eval$ will use less memory than performing the computation more than once.

As another step, ETAV will only generate a sibling for a node if the current node is not sufficient to decide the acceptance of the path. For example, if the child of an $\wedge$-node evaluates to $\bot$, then ETAV does not check the acceptance of the other child.

A third optimisation step implemented in ETAV consists of constructing the transition rule for a formula only when it is required, i.e., the transition function tf is not fully instantiated prior to starting the construction of the product automata. This, in conjunction with the fact that ETAV only constructs world-formula pairs in the product graph when reached, leads to an "on-the-fly" construction of both $\mathcal{A}_{\mathcal{D},\psi}$ and $\mathcal{A}_{\mathcal{P},\psi}$. Despite this, the technique cannot be regarded as *truly* "on-the-fly" as the whole reachable state space for the model is known prior to verification.

4.2. Evaluation

In this section we evaluate the effectiveness of our technique by looking at two common scenarios: a gossip protocol (Section 4.2.1) and the faulty train gate controller (Section 4.2.2).

We do not draw a comparison between the current implementation, which accepts an explicitly defined state space, and existing symbolic model checkers such as MCMAS [14]. The verification of "concurrent structures", similar to those supported in symbolic model checking, is in a harder complexity class [15], making a direct comparison unjustified. Indeed, symbolic model checkers such as MCMAS [14] or NuSMV [3] use implicit declarations for each agent (or, in NuSMV's case, component) in the system. These component declarations are given programmatically, i.e., in a language closer to a conventional programming language than a finite state machine where each local state is explicitly defined. In such systems, the reachable state-space has to be fully enumerated prior to verification. This can be done by composing the implicit component declarations and then finding the states reachable under the synchronised transition relation.

Comparatively, ETAV requires the user to provide explicitly the reachable global state-space before verification even begins. It follows that if a model checker can avoid generating the composed system and finding the reachable states, then its verification task is easier. It should be noted that although the model checker MCK [6] purportedly supports an explicit-state mode, the input is still given as an implicit declaration. For these reasons, we do not provide an empirical comparison between ETAV and any other tool.

4.2.1. Gossip Protocol

Gossip, or epidemic, protocols are often used to represent the propagation of messages through large-scale distributed applications, much in the way that "gossip" disseminates through social groups, based only on periodic communication.

The central idea in gossip-based protocols is that the nodes ("participants") in the system periodically share information ("gossip") between a small, random subset of other nodes. The propagation of data throughout the system depends heavily upon the peers that a node chooses to communicate with. This is based upon a notion of peer sampling [10].

Table 2. Gossip Protocol Specifications

GP_1	$EF\left(\bigwedge_{i\in A} complete_i\right)$
GP_2	$K_{G1}EF\left(\bigwedge_{i\in A} complete_i\right)$
GP_3	$AG\left(complete_{G1} \rightarrow K_{G1}AF\left(\bigwedge_{i\in A} complete_i\right)\right)$
GP_4	$AG\left(complete_{G1} \rightarrow C_{all\_participants}AF\left(\bigwedge_{i\in A} complete_i\right)\right)$

Table 3. Model Checking The Gossip Protocol

$\|A\|$	Formula	Memory (KiB)	Time (s)	Nodes
3	GP_1	3392	0.002	35
	GP_2	3392	0.002	66
	GP_3	3392	0.001	131
	GP_4	3392	0.001	324
4	GP_1	3636	0.033	69
	GP_2	3636	0.031	531
	GP_3	3636	0.030	46
	GP_4	3636	0.033	75
5	GP_1	452516	84.027	95
	GP_2	452368	84.213	41596
	GP_3	452160	84.217	232
	GP_4	452336	83.892	207

We created a rudimentary gossip-based protocol in the input for ETAV, which is parametric in the number of agents, representing nodes, in the system. Initially, each agent possesses a unique piece of data. The aim of the protocol is for each agent to propagate its information, possibly indirectly, to every other agent. In a gossip protocol with n agents the state space is as follows: 3 agents, 14 states; 4 agents, 259 states; 5 agents, 13647 states.

The specifications used for verifying the gossip-protocol are reported in Table 2. We use $complete_i$ to represent that agent i holds all the information in the system. The first specification, GP_1, represents that there exists an execution of the protocol in which all the agents eventually learn the data. Property GP_2 states that the first agent knows GP_1 (i.e., that all the agents can learn the data). The next specification, GP_3, states that if one agent holds all the data, then that agent knows that all agents eventually learn the data. Finally, GP_4 states that if one agent holds all the data, then it is common knowledge between the participants that eventually they will all learn the data. Specifications GP_1 and GP_2 are satisfiable on models of all sizes, while GP_3 and G_4 are unsatisfiable for models with strictly greater than three agents,

Table 4. Faulty Train Gate Controller Specifications

TGC_1	$AG\,(train1\_in\_tunnel \rightarrow EF\neg train1\_in\_tunnel)$
TGC_2	$AG\,(\neg train1\_in\_tunnel \vee \neg train2\_in\_tunnel)$
TGC_3	$AG(train1\_in\_tunnel \rightarrow K_{Train1}\neg train2\_in\_tunnel)$
TGC_4	$AG(K_{Train1}(\neg train1\_in\_tunnel \vee \neg train2\_in\_tunnel))$
TGC_5	$AG(C_{all\_trains}(\neg train1\_in\_tunnel \vee \neg train2\_in\_tunnel))$

as the protocol does not ensure that all of the data will eventually reach all of the agents in larger scenarios.

Table 3 shows the results for verifying the gossip protocol with ETAV. The final column, Nodes, represents the number of AND/OR nodes in the graph of the product automata. For GP_2, we can see that increasing the number of reachable states increases the number of indistinguishable states for agent G_1. This leads to a greater number of states in the product automata. When comparing the results for GP_3 and GP_4, it can easily be seen that evaluating $C_{all\_participants}$ is more costly than K_{G1}. This is due to the fact that common knowledge leads to more indistinguishable states. It should be noted that the high execution time for a model with five agents arises from parsing the large, explicitly-declared state space.

4.2.2. Faulty Train Gate Controller

The faulty train gate controller model of [11] extends the epistemic version [7] of the train gate controller model by allowing the trains to display faults non-deterministically. In the standard model n trains attempt to access a shared resource of the tunnel. When the trains do not display faults, and with a correctly functioning controller, it is not possible for more than one train to enter the tunnel at one time. In the faulty model trains are extended with a counter variable which represents the number of actions that the train has performed since being last serviced (analogous to a car's mileage counter). Once the counter exceeds a given threshold, trains can non-deterministically "break", leading them to be stuck in the tunnel. The counter has an upper limit which, when reached, causes the trains to be serviced, resetting the counter.

The specifications in Table 4 for the faulty train gate controller have the following interpretations: TGC_1 states that when a train enters the tunnel, there exists a future time when it eventually leaves; TGC_2 represents a mutual exclusion over the model; TGC_3 means that when one train is in the tunnel, it knows that the other is not; TGC_4 states that $Train1$ always knows that there is a mutual exclusion of the tunnel between the trains. All of the specifications are unsatisfiable in a model with broken trains and satisfiable on a model with working trains; TGC_5 expresses that the mutual exclusion over the tunnel is common knowledge between all the trains.

In a model with two trains and a maximum counter of seven, varying the breaking depth affects the state space as follows: breaking depth of 1, states 3389; breaking depth of 6, states 2269; working model, states 1877.

The verification results for this model can be seen in Table 5. The column Depth represents the depth

Table 5. Model Checking The Train Gate Controller

Depth	Formula	Memory (KiB)	Time (s)	Nodes
1	TGC_1	12080	1.387	308
	TGC_2	12084	1.391	199
	TGC_3	12080	1.386	114
	TGC_4	30668	1.986	298284
	TGC_5	12080	1.381	53
6	TGC_1	7992	0.704	1751
	TGC_2	7992	0.715	1118
	TGC_3	7992	0.700	55
	TGC_4	12988	0.852	82098
	TGC_5	8124	0.698	901
w	TGC_1	9005	0.650	27822
	TGC_2	9007	0.658	27140
	TGC_3	9128	0.651	29401
	TGC_4	26507	1.113	307169
	TGC_5	42884	5.854	563027

at which a fault can appear; W represents a working model, but still with a service counter. These results demonstrate that verifying a satisfiable invariant formula, even over a smaller state space, requires greater memory than for an unsatisfiable invariant. For satisfiable invariants, the product automata is required to contain every reachable state. For the working model, it follows immediately that verifying a satisfiable formula induces a greater number of nodes in the product graph. There are fewer nodes in the product graph for TGC_5 than TGC_4 in a broken model, as the greater number of indistinguishable states means that a state invalidating the mutual exclusion can be reached using a shorter epistemic path.

5. Conclusions

It is often argued that a key concern for MAS to be deployed in applications of societal importance is their lack of verification and validation. Model checking is a prominent automatic verification technique that, together with other methodologies such as testing, can be used to validate systems.

While considerable work has been carried out in the area of model checking for MAS, the current state-of-the-art still falls short of enabling engineers to verify industrial-strength MAS. Therefore, it remains of paramount importance to develop novel techniques, as well as improve existing methodologies, such that forthcoming model checking toolkits can tackle complex MAS.

In this paper we have put forward an automata-theoretic methodology for verifying MAS specified by temporal-epistemic logic. Although automata form the underlying building blocks of considerable work in model checking for reactive systems, surprisingly they have not been employed for epistemic specifications yet. To achieve this, we extended the relevant notions of automata and provided a sound translation from the whole CTLK logic into automata, thereby providing a model checking algorithm. We implemented the technique from first principles by constructing a toolkit for epistemic tree automata.

The experimental results reported with ETAV are inferior to those that can be obtained with a symbolic checker, such as MCMAS [14]. Still, they validate the correctness of the approach and show promise as ETAV contains none of the traditional optimisations present in explicit-state model checkers (e.g., state-level compression and hashing). Most importantly, the automata-theoretic technique was not leveraged on top of other efficient techniques, such as partial order reduction [13]. We believe considerable gains can be achieved in this direction but leave this for further work. Automata are usually the basis of conventional "on-the-fly" methodologies, but we are not aware of similar approaches for MAS. The present work may form a stepping stone in this direction.

Currently, ETAV only accepts models specified as full Kripke structures, where all the temporal and epistemic successors are explicitly defined. As the toolkit matures along the lines above, we will extend this to work directly on implicit models, e.g., those provided by ISPL, the input to MCMAS.

Acknowledgements: The research described in this paper was partly supported by EU Marie Curie Fellowship FOMMAS (FP7/2007-2013, grant agreement n. 235329), and by the ACSI Project (FP7-ICT-2009-5-Objective 1.2, grant agreement n. 257593).

References

[1] ETAV – Epistemic Tree Automata Verifier, http://vas.doc.ic.ac.uk/tools/etav/.

[2] Blackburn, P., van Benthem, J., Wolter, F., Eds.: *Handbook of Modal Logic*, Elsevier, 2007.

[3] Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: NuSMV: A New Symbolic Model Verifier, *Proc. 11th International Computer Aided Verification Conference*, 1999, 495–499.

[4] Clarke, E. M., Grumberg, O., Peled, D. A.: *Model Checking*, The MIT Press, 1999.

[5] Fagin, R., Halpern, J. Y., Moses, Y., Vardi, M. Y.: *Reasoning About Knowledge*, MIT Press, 1995.

[6] Gammie, P., van der Meyden, R.: MCK: Model Checking the Logic of Knowledge, *Proc. of the 16th Int. Conf. on Computer Aided Verification (CAV '04)*, 3114, Springer, 2004, 479–483.

[7] van der Hoek, W., Wooldridge, M.: Tractable Multiagent Planning for Epistemic Goals, *Proc. of the 1st Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS '02)*, 2002, 1167–1174.

[8] Holzmann, G. J.: *SPIN Model Checker, The: Primer and Reference Manual*, Addison Wesley, 2003.

[9] Huang, X., Luo, C., van der Meyden, R.: Improved Bounded Model Checking for a Fair Branching-Time Temporal Epistemic Logic, *Proc. of the 6th Workshop on Model Checking and Artificial Intelligence (MoChArt '10)*, 2010.

[10] Jelasity, M., Voulgaris, S., Guerraoui, R., Kermarrec, A., van Steen, M.: Gossip-based Peer Sampling, *ACM Trans. Comput. Syst.*, **25**(3), 2007.

[11] Jones, A. V., Lomuscio, A.: Distributed BDD-based BMC for the Verification of Multi-Agent Systems, *Proc. of the 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS '10)*, 2010, 675–682.

[12] Kupferman, O., Vardi, M. Y., Wolper, P.: An Automata-Theoretic Approach to Branching-Time Model Checking, *J. ACM*, **47**(2), 2000, 312–360.

[13] Lomuscio, A., Penczek, W., Qu, H.: Partial order reductions for model checking temporal epistemic logics over interleaved multi-agent systems, *Proc. of the 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS '10)*, 2010, 659–666.

[14] Lomuscio, A., Qu, H., Raimondi, F.: MCMAS: A Model Checker for the Verification of Multi-Agent Systems, *Proc. of the 21st Int. Conf. on Computer Aided Verification (CAV '09)*, 5643, Springer, 2009, 682–688.

[15] Lomuscio, A., Raimondi, F.: The Complexity of Model Checking Concurrent Programs Against CTLK Specifications, *Proc. of 4th International Workshop on Declarative Agent Languages and Technologies (DALT 2006)*, 4327, Springer, 2006, 29–42.

[16] Meyer, J.-J. C., van der Hoek, W.: *Epistemic Logic for AI and Computer Science*, Cambridge University Press, 1995.

[17] Muller, D., Saoudi, A., Schupp, P.: Alternating Automata. The Weak Monadic Theory of the Tree, and its Complexity, *ICALP*, 226, Springer, 1986, 275–283.

[18] Muller, D., Schupp, P.: Alternating Automata on Infinite Trees, *Theoretical Computer Science*, **54**, 1987, 267–276.

[19] Qian, K., Nymeyer, A.: Language-Emptiness Checking of Alternating Tree Automata Using Symbolic Reachability Analysis, *ETCS*, **149**(2), 2006, 33–49.

[20] Vardi, M. Y., Wolper, P.: An Automata-Theoretic Approach to Automatic Program Verification, *Proc. 1st Symp. on Logic in Computer Science*, Cambridge, June 1986, 332–344.