A Model Checker for Epistemic Hybrid Automata Using Constraints Logic Programming

Ammar Mohamed¹, Rana Mohamed², Hesham Hefny³

¹Department of Computer Science, Arab East Colleges
Riyadh, KSA

Institute of Statistical Studies and Research,
Cairo University
Cairo, Egypt

²,³Department of Computer Science Institute of Statistical Studies and Research
Cairo University
Cairo, Egypt

Abstract— In a previous work, we showed how to formally model systems in epistemic hybrid automata by extending the definition of hybrid automata with epistemic notations. Additionally, we presented a formal specification language to specify the requirements of those systems which are modeled using epistemic hybrid automata. In this paper, we present a model checker for epistemic hybrid automata. As far as we know no research presented a model checker for epistemic hybrid automata. The model checker allows us to reason about the dynamical behaviors of systems as well as the satisfaction of several epistemic properties. The proposed model checker is implemented using mathematical intervals in Constraints logic programming.

Keywords— Epistemic Hybrid Automata, Epistemic Tree Logic, Constraint Logic Programming.

I. INTRODUCTION

In a previous work [6], we extended the definition of reactive [9] systems with knowledge and introduced the so-called logic Epistemic Hybrid Tree Logic (EHTL) that combines both the region Computational Tree [5] and epistemic logic. We used the underline model of epistemic hybrid automata to interpret the formulas of EHTL. The underline model is used to describe the knowledge properties [16] in addition to quantitative properties [22].

The idea of epistemic hybrid automata (EHP) goes around how a group of agents interact by sharing their knowledge using synchronized events. Each agent takes his decision according to the gained knowledge from the interacting agents. EPH overcome the shortages of hybrid automata [19] by allowing agents to reason about individual knowledge, common knowledge or distributed knowledge [23]. Broadly speaking, examining hybrid automata with epistemic properties forms a challenge, since the implementation of hybrid automata generates an infinite number of states [14]. Additionally, with adding knowledge to reactive systems, you will need to add extra features to examine the process of knowledge reasoning.

An automatic formal verification technique of reactive systems [24] is model checking [10]. Model checker takes a model of a system and specification of the properties as input and seeks if the properties are satisfied. Most of the model checkers guarantee the correctness of the specification of reactive systems in terms of reachability analysis [13], where a model checker computes the reachability of the states from the a start state of a model of a system under investigation. In [1], [3], [5], the authors presented a constraint logic programming (CLP) [15] model checker for hybrid automata. CLP [18] is a suitable tool for modeling real life multi-agent systems (MAS) [11] because CLP is a declarative programming language as well as contains efficient constraints solvers of various domains. Several formal verification of reactive systems [8] and hybrid automata by means of CLP [20] presented. However, up to our knowledge, no work tried to present a model checking for EPH.

In summary, the primary contribution of this paper is to present a novel model of Epistemic Hybrid automata by means of constraint logic programming (CLP). Additionally, the CLP model allows us not only to model epistemic behaviors of multi-agent systems, but also to check various properties by means of the reachability analysis.

The rest of paper is organized as follows. In section II we discuss related work; in section III we present the proposed Epistemic Hybrid Automata. In section, V we define Epistemic Hybrid Automata case study. In section IV we define Epistemic Hybrid Tree Logic. In section VI we introduce the knowledge and reachability analysis. Finally, we conclude in section VII by presenting the conclusion.

II. RELATED WORK

No model checker presented for knowledge and hybrid automata, but some researchers presented a model checkers for knowledge with time [17] that
In the following EPH components:

each EPH is an agent (named epistemic hybrid automata [2], where automata. However, their work cannot be used to check $r = o \cup s$.

A. Epistemic Hybrid Automaton Components (EPH):

Epistemic hybrid automaton (EPH) is a tuple of:

$\langle X, Q, knowledge, Local, Cont, Aut, E, Jump, Reset, Event, \sim_{\text{EPH}}, N_{\text{EPH}} \rangle$

- $X$: a finite set of real values that represent the continuous dynamics, where $X = \{x_1, x_2, \ldots, x_n\}$.
- $Q$: is a finite set of locations representing the location of each agent, $|Q| = n$ is the total number of locations for epistemic hybrid automaton.
- $Local$: $\sigma \rightarrow q_i$ is a function that returns the current location of a state.
- $knowledge$: $Q \rightarrow Kdge$ a function which assigns to each location $q \in Q$ a knowledge.
- $Aut$: $Q \rightarrow \mu(X)$, is a function that assigns automata constraint $Aut(q)$ to each location $q \in Q$.
- $Cont$: $Q \rightarrow C(X \cup X)$ is a labeling function that assigns to each location $q \in Q$ continuous constraint $Cont(q)$.

- $E \subseteq Q \times Q$ is a finite set of discrete transition among control locations. Each transition $(q_1, q_2) \in E$ has a source location $q_1$, and target location $q_2$.
- Jump: $E \rightarrow \mu(X)$ a function assign to each transition $e \in E$ a constraint $jump(e)$, which must hold on fire.
- Reset: $X \rightarrow X$ is a mapping function which assigns a real value to each variable, which reset the variables before the control of EPH goes from location $q_1$ to location $q_2$.
- Event: is a function that assigns an event $Event(e)$ to each transition.

$\sim_{\text{EPH}}$ is an epistemic (accessibility) relation for EPH $\sim_{\text{EPH}} \subseteq \Omega_1 \times \Omega_2$ for each agent $EPh \in A$, if $\forall \sigma_i \in \Omega_1, \forall \sigma_j \in \Omega_2, local(\sigma) = local(\sigma)$, and $\sigma_i = (q_i, v_i, Kdge_i, t_i)$ $\equiv \sigma_j = (q_j, v_j, Kdge_j, t_j)$.

$N_{\text{EPH}}$ is the agent knowledge update function $Q \rightarrow Kdge \cup \omega$, where $EPh \in A$.

B. Syntax

To understand the components of EPH we need first to understand some concepts, so in the following, we introduce some definitions.

Definition 1 (Agent Local Epistemic State)

The state of an agent $m \in \mathbb{A}$ is given by $\sigma_m = (q_i, v_i, Kdge_m, t_i)$, where $q_i \in Q, v_i$ is a valuation of real variables, $knowledge(q)^m = kdge^m$ is the knowledge of agent $m$ at state $i$, $t$ is the time variable, $1 \leq i \leq n$ and between each two states an epistemic relation. Epistemic state for agent changes when an agent takes a decision by firing an event.

Definition 2 (Epistemic Relation)

Let $\Gamma \subseteq \mathbb{A}$ be a subset of agents, $m \in \mathbb{A}$ is an agent, $\sigma_m$, $\sigma_{m+1}$, $\sigma_m^m = (q_i, v_i, Kdge_m^m, t_i), \sigma_{m+1}^m = (q_{i+1}, v_{i+1}, Kdge_{m+1}^m, t_{i+1})$ are two states and let $\sim_m$ is an epistemic relation between the two states.

Where $\sigma_m^m \sim_m \sigma_{m+1}^m$ if:

- $Local(\sigma_i) = Local(\sigma_{i+1})$.
- $v_i \equiv v_{i+1}$.
- $kdge_i^m = kdge_{i+1}^m$.
- $t_i, t_{i+1} \in [t_1, t_2]$.

$v_i \equiv v_{i+1}$ and $\forall t_i, t_{i+1}$ satisfies $Aut(\sigma_i^m) = Aut(\sigma_{i+1}^m)$ and $v_i, v_{i+1}$ satisfies $Aut(\sigma_i^m)$ and $t_i, t_{i+1} \in [t_1, t_2]$ means that $t_i, t_{i+1}$ within the same interval.

Where the epistemic relation for common and distributed knowledge is as follows:

Common knowledge epistemic relation $\sim_c^m = (\cup_{m \in \mathbb{R}^+})^+$.

Distributed knowledge epistemic relation $\sim_d^m = (\cup_{m \in \mathbb{R}^+})^*$.
Where is a synchronous event between agent \( \text{A} \). Behavior, in the following we define path interval and behaviors continuous behavior that generates finite \( t \) interval, knowledge remains the same during the path interval, \( V \) is the tuple of path interval valuation of the variables during the time interval \( T \) and \( t_{i+1} \) \( \leq T \leq t_{i+1} \) at location \( q \).

**B. Run:** A run of EPH consists of a set of path intervals between each two path intervals a transition, the run is represented as follows:

\[
\begin{align*}
\rho \text{EPH} &= \Omega_m \rightarrow \Omega_{m+1} \rightarrow \ldots \rightarrow \Omega_n, \\
\text{where } a_i: t_1 \rightarrow t_2 \rightarrow \ldots \rightarrow t_k \in \text{event }
\end{align*}
\]

**IV. EPISTEMIC HYBRID TREE LOGIC (EHTL)**

Now we are going to present a new logic called epistemic hybrid tree logic that composes of region computation tree logic and epistemic logic, EHTL is interpreted over all possible generated path intervals from the epistemic hybrid automata, where agent move from location to another according to the timed constraints and shared knowledge. Now we will declare the meaning of ticks \([5]\).

**C. Ticks**

Tick \( T \) is set of non-negative real variables and \( \mu(T) \) a set of automata constraints over \( T \). The valuation of \( T \) is a function \( \rho : T \rightarrow R^{\geq 0} \), given that \( \pi \in \mu(T) \), we write \( \rho \models \pi \) if \( \rho \) satisfies \( \pi \).

1) **EHTL Syntax**

Let \( \mathbb{X} \) be a set of real variables, \( \mathbb{T} \) be a set of non-negative real variables disjoint from \( \mathbb{X} \), \( \mu(\mathbb{X}) \) and \( \mu(\mathbb{T}) \) be sets of automata constraints with free variables from \( \mathbb{X}, \mathbb{T} \). \( \Delta \) be a set of atomic proposition denoting the locations, Event be a set of atomic proposition disjoint from \( \Delta \) and \( \Lambda \) be a set of all agents.

**a) Formulas of EHTL:**

\[
\begin{align*}
\varphi : = & p [\alpha] [\mu(\varphi_1 \land \varphi_2) - \varphi] \times [\varphi_1] \times [\varphi_2] [\exists (\varphi_1 \\
& \land \varphi_2) [k_m \varphi(C_{\varphi}D_{\varphi} \varphi)]]. \\
\text{Where } & \alpha \in \Delta, \mu \in \mu(\mathbb{X}), \alpha, \in \text{Event}, y, x \in \mathbb{X}, t \in \mathbb{T}, a \in \mathbb{A}, \Gamma \subseteq \mathbb{A}, \varphi_1, \varphi_2 \text{ are EHTL formulas and the logical connectives } \Lambda \rightarrow \text{ have their usual meaning. We have some abbreviations:}
\end{align*}
\]

\[
\begin{align*}
& \exists \varphi \equiv \exists(\text{true} \land \varphi) \\
& \forall \varphi \equiv \forall(\text{true} \land \varphi) \\
& \exists \varphi \equiv \neg \forall \varphi \land \varphi \equiv \neg [\mu(\varphi(C_{\varphi}D_{\varphi} \varphi))] \\
& \forall \varphi \equiv \neg \exists \varphi \land \varphi \equiv \neg K_{\mu} \varphi \land \varphi.
\end{align*}
\]

**VI. EPISTEMIC HYBRID AUTOMATA CASE STUDY**

Epistemic logic has achieved such prominence in the mid’s of 1980 since they discovered that epistemic logics could be given a natural interpretation according to the states of agents, so our
case study aims to study the epistemic state for each agent. Where agent takes his decision based on the available knowledge. We improved the case study presented in [3] to examine how agent takes decision depending on the received knowledge of the interacting agents, the scenario talks about 4 agents, “truck” responsible for delivering the items to specific destination with a specific percentage of decayed items, “cargo” responsible for assigning the delivery mission to a specific truck with a specific price and sending knowledge about the changing of price with time passing so the truck takes the decision of finishing the transportation mission or request help from the provider to send another truck to deliver the items, “disturbance” responsible for observing the occurrence of errors, finally “decay” is plugged into the truck and is responsible for observing errors and the putrefied items, and alarm the truck in case of errors.

Where EpistematonName is the name of the epistemic automaton, Location is the location that epistemic automaton sited in, Variable0 is the initial values that represent the continuous evolution of variables over time, Time0 is the initial time for each initial value, Time is the evolving time, Variables is the list of real values, C1, C2 are the constraints over the real variables, initKnow is the knowledge of automaton at the beginning of the location, the knowledge of the automata remains unchangeable during the continuous evolution and Events is the event that supposed to be fired during the run. As an example of automaton definition:

cargo(init3,[Z0],Z,T0,T,Know,Events):=
\[
\begin{align*}
Z &= Z0, T &= T0, calcTp(T, Price), \\
Know &= [Price], Event &:: event.
\end{align*}
\]

The query here calls function calcTp to calculate price depending on the passing time and set the price as cargo knowledge. EPH has two types of behaviors continuous behavior that represents the evolution of variables and knowledge over time and discrete behavior that represents how automaton moves from a location to another, we define the relation behavior that capture how system evolves over continuous and discrete behavior.

During the continuous evolution, an infinite number of states is generated at the same location with different values of variables, different timing variables and the same knowledge along the evolution, while at the discrete step the knowledge is updated from state to another by concatenating the knowledge of the first
state with the shared knowledge coming from other agents.

When the discrete step occurs it means that an event has fired, where Event &::events means that the value of Event is within the domain of event, UpdatedKnow is the knowledge of the FirstLoc and the shared knowledge coming when a synchronized event is fired. Now we need to check if we have a synchronized event.

The previous query examines if EpistAutomaton1Event and EpistAutomaton2Event are two synchronized events, in this case, the shared knowledge for Automaton 1, 2 will be Knowledge1 and Knowledge2.

The continuous evolution shows how automaton variables are evolved at the same location with the same knowledge till a jump condition occurs, and it is represented as follows:

The most important implementation part in epistemic hybrid automata code is the part that allows automatons to interact with each other and share knowledge, we call it the drive:

When the discrete step occurs it means that an event has fired, where Event &::events means that the value of Event is within the domain of event, UpdatedKnow is the knowledge of the FirstLoc and the shared knowledge coming when a synchronized event is fired. Now we need to check if we have a synchronized event.

The previous query examines if EpistAutomaton1Event and EpistAutomaton2Event are two synchronized events, in this case, the shared knowledge for Automaton 1, 2 will be Knowledge1 and Knowledge2.

The continuous evolution shows how automaton variables are evolved at the same location with the same knowledge till a jump condition occurs, and it is represented as follows:

The most important implementation part in epistemic hybrid automata code is the part that allows automatons to interact with each other and share knowledge, we call it the drive:

The most important implementation part in epistemic hybrid automata code is the part that allows
write("disturbance know"), write(Knowh), write(":") , write("decay know"), write(Knowh1), write(":") , write("cargo know"), write(Knowh2), write(":") , write("truck know"), writeln(Knowh3), write("Event :") , writeln(Event), write("Time = "), writeln(Tx), writeln("-----------") ,
Steps > 0, Steps1 is Steps -1, drive((disturbance, NextS1, YY0, Knowa), (decay, NextS2, DD0, Knowb), (cargo, NextS3, XX0, Knowc), (truck, NextS4, ZZ0, Knowd), Newstarttime, Steps1, R).

At the drive part, it starts with the definition of each automaton with its initial variables, timing, and knowledge, then checks the event that may fire from each state to set the shared knowledge for each epistemic automaton, after that the epistemic automata start to interact by calling behavior. At the end of each state, the model checks if there is any redundant knowledge and removes it by calling function set. At last, we print the values of each variable, timing, and knowledge for each automaton. The output of the run shows the reached states for each epistemic automaton from the initial state with the knowledge of each agent and their shared knowledge.

VII. KNOWLEDGE AND REACHABILITY ANALYSIS
A. Reachability
Reachability [13] in EHTL is achieved when an agent knows that a certain state can be reached from the initial state.

?- reached((disturbance, S1), (decay, S2), (cargo, S3), (truck, S4), S):- drive((disturbance, init1, 0, K1), (decay, init2, 0, K2), (cargo, init3, 0, K3), (truck, init4, 0, K4), 0, S, Output), append(List, [((disturbance, _), (decay, _), (cargo, S3, _), (truck, S4, _), _, _, _) | _], Output).

B. Knowledge Properties in CLP
At [2] knowledge has some properties defined using EHTL as positive introspection and negative introspection, they can also be defined using CLP. Positive introspection states that when the agent knows something, then he knows that he knows that thing, which means knowing of your knowledge, you can check if an agent knows that he knows a specific knowledge at a specific state if this knowledge in the agent knowledge base.

∃◊ k_m φ → ∃◊ k_m k_m φ

We represent it in CLP as follows:

?- knowsTknows(disturbance, S1, X, ST):- drive((disturbance, init1, 0, K1), (decay, init2, 0, K2), (cargo, init3, 0, K3), (truck, init4, 0, K4), 0, ST, R), append(First, [((disturbance, _,_), (decay, _,_), (cargo, S3, _), (truck, S4, _) | _], R), (member(X, KN) -> write("disturbance knows that it knows "), write(X)).

The previous query checks the knowledge of the agent at a certain state, we use append and member prolog predicates. The second knowledge property is negative introspection which means the knowing of your ignorance. in EHTL:

∃◊ ¬k_m φ → ∃◊ k_m ¬k_m φ

it can be represented in CLP as:

?- knowsdknows(disturbance, S1, X, ST):- drive((disturbance, init1, 0, K1), (decay, init2, 0, K2), (cargo, init3, 0, K3), (truck, init4, 0, K4), 0, ST, R), append(First, [((disturbance,_,_), (decay, _,_), (cargo, _,_), (truck, _,_) | _], R), ( + member(X, KN) -> write("disturbance knows that it doesn’t knows "), write(X)).

This property can be checked by checking the knowledge of the agent at a certain state, if not exists then he knows that he doesn’t know. Both properties present a powerful tool to eliminate skepticism since you know your knowledge and ignorance.

Knowledge becomes a common knowledge when it is shared with each agent participating in the system, the following CLP code checks if knowledge X is a common knowledge for disturbance, decay, cargo and truck at time T.

?- commonknow(X, (disturbance, decay, cargo, truck), T):- drive((disturbance, S1, X0, Know), (decay, S2, D0, Know1), (cargo, S3, X0, Know2), (truck, S4, Z0, Know3), 0, 6, R), append(First, [((disturbance,_,_), (decay, _,_), (cargo, _), (truck, _), T, _) | _], R), ( member(X, KN) -> write("disturbance knows that it doesn’t knows "), write(X)).

Distributed knowledge means that a specific knowledge is distributed over agents, which means after collecting the knowledge of the distributed agent we will know the willing piece of knowledge. Let’s
imagine that the `decay` notice anonymous signal says “crowd” and `cargo` knows that crowd means that price will be decreased to 1000 but `cargo` doesn’t know the signal yet, so price = 1000 is a distributed knowledge over `cargo` and `decay`. The following CLP code shows how to check distributed knowledge.

distributed(1000, cargo, decay):-
  drive((disturbance, S1, X0, Know),
  ( decay, S2, D0, Know1), (cargo, S3, X0, Know2),
  (truck, S4, Z0, Know3),
  0,4,R), append(Know1,_,NK),
  member((crowdis1000,Know2),member(crowd,NK),member(_,_,( decay,decay,NK),_,_,_,R).

VIII. CONCLUSION

The verification of multi-agents that have epistemic states is a critical task, especially when Agents situated in safety critical applications. So they need a model checker to check their epistemic state at each step of the system, to examine the changing of agent knowledge and it’s continuous dynamic behavior. This paper presented a model checker using constraint logic programming (CLP) to check agent behavior based on quantitative and knowledge analysis. The paper applied epistemic hybrid tree logic (EHTL) that verified using epistemic hybrid automata (EHP) in a planning case study and checked the behavior of the interacted agents using the CLP model, to check if it follows the definition of EHP.

REFERENCES


