ISSN:0975-9646

A Model Checker for Epistemic Hybrid Automata Using Constraints Logic Programming

Ammar Mohamed[1], Rana Mohamed[2], Hesham Hefny[3]

[1]*Department of Computer Science, Arab East Colleges*
Riyadh, KSA
Institute of Statistical Studies and Research,
Cairo University
Cairo, Egypt

[2,3] *Department 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 (EPH) 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

interpreted using interpreted systems as in [21] authors proposed MCK, 'Model Checking Knowledge', which is a model checker for real temporal epistemic logic. The system composed of a set of agents communicating in an environment, the interaction runs according to a specific protocol, and actions are taken according to it. Agents gain knowledge according to the current actions and clock value. Actions cause a change in the epistemic state of the system. Also in [11], [12] MCMAS, for Model Checking Multi-Agent Systems, is a model checker similar to MCK, For Knowledge and temporal logic, it allows to specify LTL and CTL. Paper [7] presented Bounded model checking which tests the presented formula against counterexamples. Other papers presented CLP as a model checker language to verify hybrid automata using arithmetic intervals as in [20] they provided a rigorous approach to modelling, simulating, and analyzing Hybrid Systems using CLP, also in [1], [3] ,[4], [5] authors provided a model checker using CLP, arithmetic intervals and they verify the system using reachability analysis. Also, [2] Authors presented a CLP for hierarchical hybrid automata. However, their work cannot be used to check epistemic logic with hybrid automata.

III. PROPOSED EPISTEMIC HYBRID AUTOMATA

In this section, we present new formal model named epistemic hybrid automata (EPH) [2], where each EPH is an agent ($EPH \in \mathbb{A}$), where $\mathbb{A}$ is the set of all agents, and each EPH have his own knowledge. In the following EPH components:

A.. Epistemic Hybrid Automaton Components (EPH):

Epistemic hybrid automaton (EPH) is a higher class of hybrid automata, which is consisting of the HA and knowledge requirements, each EPH represents an agent and each EPH is a tuple of:

$EP\,H =$
$(\mathbb{X}, Q, knowledge, Local, Cont, Aut, E, Jump,$
$Reset, Event, \sim^{EPH}, N_{EPH})$

- $\mathbb{X}$: a finite set of real values that represent the continuous dynamics, where $\mathbb{X} = \{x_1, x_2, \dots, x_n\}$.
- 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.
- $Aut: Q \rightarrow \mu(\mathbb{X})$, is a function that assigns automata constraint $Aut(q)$ to each location $q \in Q$.
- Cont: $Q \rightarrow C(\mathbb{X} \cup \dot{\mathbb{X}})$ is a labeling function that assigns to each location $q \in Q$ continuous constraint $Cont(q)$.
- $E \subseteq QxQ$ 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: $E \rightarrow \mu(\mathbb{X})$ a function assign to each transition $e \in E$ a constraint $Jump(e)$, which must hold on fire.
- Reset: $\mathbb{X} \rightarrow \mathbb{X}$ 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.
- $\sim^{EPH}$is an epistemic (accessibility) relation for EPH $\sim^{EPH} \subseteq \Omega_1 x \Omega_2$ for each agent $EPH \in A$, if $\forall\sigma_i \in \Omega_1, \forall\sigma_j \in \Omega_2, local(\sigma_i) = local(\sigma_j)$, and
  $\sigma_i = (q_i, v_i, Kdge_i, t_i) \equiv$
  $\sigma_j = (q_j, v_j, Kdge_j, t_j)$.
- N_{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 agent $m \in \mathbb{A}$ is given by $\sigma_i^m = (q_i, v_i, kdge_i^m, t_i)$, where $q_i \in Q$, v_i is a valuation of real variables, $knowledge(q_i^m) = kdge_i^m$ 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, σ_i^m, σ_{i+1}^m $\sigma_i^m = (q_i, v_i, kdge_i^m, t_i), \sigma_{i+1}^m = (q_{i+1}, v_{i+1}, kdge_{i+1}^m, t_{i+1})$are two states and let $\sim^m$ is an epistemic relation between the two states.

Where $\sigma_i^m \sim^m \sigma_{i+1}^m$ if :

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

$v_i \equiv v_{i+1}$ When $Aut(\sigma_i^m) = Aut(\sigma_{i+1}^m)$ 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
$\sim_\Gamma^C = (\cup_{m \in \Gamma} \sim^m)^+$.
Distributed knowledge epistemic relation

$\sim_{\Gamma}^{D} = \cap_{m\in\Gamma}\sim^{m}$.
The relation here is an equivalence relation.

Definition-3 (Shared Knowledge)

Let $m, o \in \mathbb{A}$, are two agents, and σ_i^m,σ_{i+1}^m,σ_j^o,σ_{j+1}^o are four states and $a \in Event$ which is a synchronous event between agent m and o.

$$\sigma_i^m \xrightarrow[t]{a,\varphi_1} \sigma_{i+1}^m, \text{ and } \sigma_j^o \xrightarrow[t]{a,\varphi_2} \sigma_{j+1}^o$$

Where $knowledge(\sigma_i^m) = \varphi_1$ and $knowledge(\sigma_j^o) = \varphi_2$ are the shared knowledge from agent m and o, so $knowledge(\sigma_{i+1}^m) = knowledge(\sigma_i^m) \cup \varphi_2$, and $knowledge(\sigma_{j+1}^o) = knowledge(\sigma_i^o) \cup \varphi_1$.

Definition-4 (Transitional Semantics)

A transition between two epistemic states σ_1^m=$(q_1, v_1, kdge_1^m, t_1)$ and σ_2^m =$(q_2, v_2, kdge_2^m, t_2)$ have two types discrete transition $\sigma_1^m \xrightarrow[t]{Event(e),\varphi} \sigma_2^m$ and continuous transition.

1) *Discrete Transition:* it occurs when we have $(q_1, q_2) \in E$, $t_1 = t_2$, $v_1 \vDash Jump(e)$, $N_m(q_1) = Knowledge(q_1) \cup \varphi_1$, $N_m(q_2) = Knowledge(q_2) \cup \varphi_2$ where $Knowledge(q_2) = Knowledge(q_1) \cup \varphi_1$ and $v_2 \vDash Aut(q_2)$, such that φ_1 , φ_2 is shared knowledge, v_2 is the valuation coming from Reset(e, *X*), in this case Event(e) occurs.

2) *Continuous Transition:* it occurs when we have $q_1 = q_2$, $(t_2 - t_1) > 0$ is the time elapsed at location q_1 and there is a differentiable function f with $f \vDash_* Cont(q_1)$, $N_m(t_1) = Knowledge(t_1) \cup \varphi$, $N_m(t_2) = Knowledge(t_2)$ where $Knowledge(t_2) = Knowledge(t_1) \cup \varphi$, $f(t_1) \vDash v_1$, $f(t_2) \vDash v_1$ such that φ is shared knowledge and for all $t \in [t_1, t_2]$, $f(t) \vDash Auto(q_1)$.

Definition -5: Execution Path

EPH implementation generates two types of behaviors continuous behavior that generates finite number of states named a path interval and discrete behavior, in the following we define path interval and run of EPH:

A. Path Interval: Path interval is a sub sequence of local epistemic states $\Omega = (\sigma_{i+1}^m, ..., \sigma_{i+j}^m) \subseteq \rho$, where ρ is EPH path. Path interval is used to represent the evolution of variables at the same location for a period of time. Path interval is a tuple of $\Omega = (q, V, Kdge, T)$, where $\{Kdge\}$ is the knowledge at the end of the 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 $t_{i+1} \leq T \leq t_{i+j}$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: $\rho EPH = \Omega_0, \xrightarrow[t_1]{a_1,\omega_1}, \Omega_1, \xrightarrow[t_2]{a_2,\omega_2}, \ldots, \xrightarrow[t]{a,\omega}$, where $a_i \in event$ is an event, that generated before the agent goes to Ω_{i+1}, and ω_i is the shared knowledge coming from other agents.

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 $\mathbb{T}$ is set of non-negative real variables and $\mu(\mathbb{T})$ a set of automata constraints over $\mathbb{T}$. The valuation of $\mathbb{T}$ is a function $\wp : \mathbb{T} \to R^{\geq 0}$, given that $\pi \in \mu(\mathbb{T})$, we write $\wp \vDash \pi$ if $\wp$ satisfies π.

1) EHTL Syntax

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

a)Formulas of EHTL:

$$\varphi ::= p|a|\mu|\varphi_1 \wedge \varphi_2|\neg\varphi|x.\varphi|\pi|\forall(\varphi_1 \sqcup \varphi_2)|\exists(\varphi_1 \sqcup \varphi_2)|k_m\varphi|C_\Gamma\varphi|D_\Gamma\varphi.$$

Where$p \in \Delta, \mu \in \mu(\mathbb{X}), a \in Event, y, x \in \mathbb{T}, m \in \mathbb{A}, \Gamma \subseteq \mathbb{A}$, φ_1 , φ_2 are EHTL formulas and the logical connectives $\wedge$, $\rightarrow$ have their usual meaning. We have some abbreviations:

$$\exists \Diamond \varphi \equiv \exists(true \sqcup \varphi)$$
$$\forall \Diamond \varphi \equiv \forall(true \sqcup \varphi)$$
$$\exists\Box\varphi \equiv \neg\forall \Diamond \neg\varphi \forall\Box\varphi \equiv \neg\exists \Diamond \neg\varphi$$
$$K_m\varphi \equiv \neg K_m\neg\varphi.$$

V.

VI. EPISTEMIC HYBRID AUTOMATA CASE STUDY

Epistemic logic has achieved such prominence attention 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" ıs plugged into the truck and is responsible for observing errors and the putrefied items, and alarm the truck in case of errors.

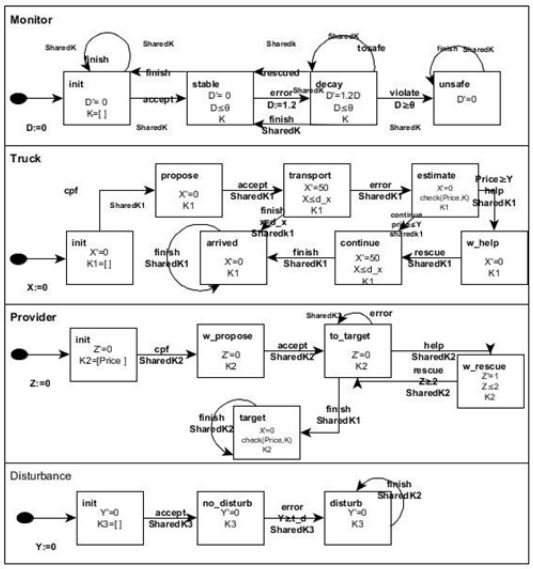

Figure 1- Case Study

A. *CLP Representation*

In this section, we declare how to encode the implementation of epistemic hybrid automata for the previous scenario in CLP, the reason of choosing CLP that it is enriched with a lot of domain solver as interval domain (IC) that is used to represent the continuous evolution of automata and symbolic domain to represent fired events. The code is implemented using eclipse prolog [18]. Our model follows the definition of EPH, where each automaton is defined by the automaton location, the real variables, time and the agent knowledge.

```
%%EpistautomatonName(+Location,?Variable0,+Variables,?Time0,+Time,?initKnow,?Event)

EpistautomatonName(Location,Variable0,Variables,Time0,Time,initKnow,Event):-C1(Variables)is (Variable0) * (Time-Time0), C2(auto),Time$>Time0, initKnow=[automaton knowledge∪shared knowledge] ,Event & :: event.
```

Where *EpistautomatonName* 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* is 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):-
Z$=Z0,T$=T0,calcTp(T,Price),
Know=[Price],Event & :: event.
```

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.

```
%%%behavior(+EpistAutomatonName,+State,-Nextstate,+T0,+Time,+Shared,+Know1,-Know2,?Event)
```

**Continuous:*

```
behavior(EpistAutomatonName(State,Start-state,Know),(State,Newstartstate,Know),
Shared,T0,T,Event):-
continuous(EpistAutomatonName(State,
Startstate,Know),(State,Newstartstate,
Know),Shared,T0,T,Event).
```

*Discrete

```
behavior(EpistAutomatonName,(State,
Value1,Know),(Nextstate,Value2,Know1),
Shared,T0,T,Event):-
discrete(EpistAutomatonName,State,
Nextstate,Value1,Value2,T0,T,Shared,Know
,Know1,Event).
```

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.

```
%discrete(+EpisAutomaton,+State1,-
State2,?Shared,+Kn,-Kn2,+IntTime,-
Time,-Event)

discrete(EpistAutomatonName,FirstLoc,Se
cLoc,FirstVar,ResetVar,T0,T,SharedKnow,
FirstKnow,UpdatedKnow,Event):-
Event&::events,Event&=Eventname,
EpistAutomatonName(FirstLoc,[FirstVar],
[Var],T0,TT,FirstKnow,Event),jump(Var),
Reset(ResetVar),conc(FirstKnow,
SharedKnow,UpdatedKnow).
```

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.

```
%check((+Epist1Event,?Shared1,+Kn1),
(+Epist2Event,?Shared2,+Kn2))

check((EpistAutomaton1Event,Shared1,Kno
wledge1),(EpistAutomaton2Event,Shared2,
Knowledge2),(EpistAutomaton3Event,Share
d3,Knowledge3),(EpistAutomaton4Event,Sh
ared4,Knowledge4)):-
Event1&=Event2,conc(Knowledge1,Knowledg
e1,Shared1),Shared2=Shared1,Event3 &=
Event4,conc(Knowledge3,Knowledge4,Share
d3), Shared4 =Shared3.
```

The previous query examines if *EpistAutomaton-1Event* 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:

```
% continuous (+EpisAutomaton,(+State1,
+Var0,+Know),(+State1,-Var,+Know),
?Shared,+IntTime,-Time,-Event)

continuous(EpistAutomatonName,(State,
Var0,Know),(State,Var,Know),_,Time0,Time
,Event):- Event &::events, Event &\ =
accept, Event & \ = otherEvents,
EpistAutomatonName(State,[Var0],[Var],T0
,TT,Know,_),getbounds(TT,TTl,TT2),
getbounds(Time,Tl,T2),Time2 is T2-Tl,
TT2-TTl>Time2,
EpistAutomatonName(State,[Var0],[Var],
Time0,T2,Know,_).
```

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:

```
drive(_,_,_,_,_,0,[]):- !.
drive((disturbance,S1,Y0,Know),(deca
y,S2,D0,Know1),(cargo,S3,X0,Know2),(
truck,S4,Z0,Know3),
Starttime,Steps,[((disturbance,S1,Kn
ow),(decay,S2,Know1),(cargo,S3,Know2
),(truck,S4,Know3),Time,Event,D)|R])
:-
disturbance(S1,[Y0],[Y],Starttime,Ty
,Know,Event1),
decay(S2,[D0],[D],Starttime,Td,Know1
,Event2),
cargo(S3,[X0],[X],Starttime,Tx,Know2
,Event3),
truck(S4,[Z0],[Z],Starttime,Tz,Know3
,Event4),
Ty $=Td, Ty $=Tx, Ty $=Tz,Time $=Ty,

check((Event1,Shared1,Know),(Event2,
Shared2,Know1),(Event3,Shared3,Know2
),(Event4,Shared4,Know3)),

Event1 &= Event2, Event1 &= Event3,
Event1 &= Event4, Event&= Event1,

behavior(disturbance,(S1,Y0,Know),(N
extS1,YY0,Knowa),Shared1,Starttime,T
y,Event),
behavior(decay,(S2,D0,Know1),(NextS2
,DD0,Knowb),Shared2,Starttime,Td,Eve
nt),
behavior(cargo,(S3,X0,Know2),(NextS3
,XX0,Knowc),Shared3,Starttime,TX,Eve
nt),
behavior(truck,(S4,Z0,Know3),(NextS4
,ZZ0,Knowd),Shared4,Starttime,Tz,Eve
nt),

get_bounds(Time,_,Newstarttime),
write("timeinterval"),
writeln(Time),
write(S1),write(":"),write(S2),write
(":"),write(S3),write(":"),
writeln(S4),write(Y),write(":"),
write(D),write(":"),write(X),
write(":"), writeln(Z),
set(Know,Knowh),set(Know1,Knowh1),se
t(Know2,Knowh2),set(Know3,Knowh3),
%remove redundant knowledge
```

```
write("disturbanceknow"),
write(Knowh),write(":"),write("deca
y
know"),write(Knowh1),write(":"),wri
te("cargo know"),write(Knowh2),
write(":"),write("truck know"),
writeln(Knowh3),write("Event  :"),wr
iteln(Event),write("Timex="),
writeln(Tx),writeln("----------"),
Steps > 0,Steps1 is Steps -1,
drive((disturbance,NextS1,YY0,Knowa
),(decay,NextS2,DD0,Knowb),(cargo,N
extS3,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 automatons 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.

$$init \longrightarrow \exists \Diamond k_m \varphi$$

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

$$\exists \Diamond k_m \varphi \longrightarrow \exists \Diamond k_m k_m \varphi$$

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,S1,KN),_,_,_
,_,_)|_],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:

$$\exists \Diamond \neg k_m \varphi \longrightarrow \exists \Diamond k_m \neg k_m \varphi$$

it can be represented in CLP as:

```
?-     knowsdknows(disturbance,S1,X,ST):-
drive((disturbance,init1,0,K1),(decay,in
it2,0,K2),(cargo,init3,0,K3),(truck,init
4,0,K4),0,ST,R),
append(First,[((disturbance,S1,KN),_,_,_
,_,_)|_],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,Y0,Know),(decay,S2
,D0,Know1),(cargo,S3,X0,Know2),(truck,S4
,Z0,Know3), 0,6,R),
append(First,[((disturbance,_,KX),(decay
,_,KN),(cargo,_,KY),(truck,_,KZ),T,_)|_]
,M),append(M,_,R),(member(X,KX),member(X
,KN),member(X,KY),member(X,KZ) ->
write(X)),write("  is Common Knowledge
Between Agents").
```

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,Y0,Know),
(decay,S2,D0,Know1),(cargo,S3,X0,Kno
w2),(truck,S4,Z0,Know3),
0,4,R),append(Know1,_,NK),
member(crowdis1000,Know2),member(cro
wd,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 (EPH) 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 EPH

REFERENCES

[1] A. Mohammed, and U. Furbach, *Using CLP to model hybrid systems*, Proceedings of Annual ERCIM Workshop on Constraint Solving Programming (CSCLP2008). *2008b*,
URL: *http://pst.istc.cnr.it/CSCLP08/program*

[2] A. Mohammed and F. Stolzenburg, *Using constraint logic programming for modeling and verifying hierarchical hybrid automata*, Technical Report 6/2009, *Department of Computer Science, Universit¨at Koblenz–Landau*, 2009..

[3] A. Mohammed and U. Furbach, *From Reactive to Deliberative Multiagent Planning*. MSVVEIS. *pages(67–75)*. Springer. 2009.

[4] A. Mohammed, and U. Furbach, *Multi-agent systems: Modeling and verification using hybrid automata*. Springer. 2010.

[5] A. Mohammed, and U.Furbach, *MAS: qualitative and quantitative reasoning. Programming Multi-Agent Systems*. pages(114 -132). Springer. 2012.

[6] A. Mohammed, R. Mohamed and H. Hefny "Epistemic Hybrid Tree Logic", ICCTA 2015, pages(21-27), conference proceeding, 2015.

[7] A. Lomuscio and W. Penczek, , *Bounded model checking for knowledge and real time*, Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, *pages(165—172)*, ACM, 2005.

[8] A. Pnueli, *Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends*, Springer ,1986.

[9] D. Harel, and A. Pnueli, . *On the development of reactive systems. Logics and models of concurrent systems*
pages (477–498), Springer,1985.

[10] E.Clarke, O. Grumberg, and D.Peled,. *Model checking*, Springer,1999.

[11] F. Raimondi and A. Lomuscio. *Automatic verification of deontic properties of multi-agent systems*. In Proceedings of DEON04, *pages(228–242).* Springer Verlag, 2004.

[12] F. Raimondi and A. Lomuscio. *Verification of multiagent systems via ordered binary decision diagrams: An algorithm and its implementation*. In Proceedings of the Third International Joint Conference on Autonomous Agents and Multi- Agent Systems (AAMAS 04), *pages(630–637).*, IEEE Computer Society 2004.

[13] G. Lafferriere, G. J Pappas and S. Yovine, *Reachability computation for linear hybrid systems*, Proceedings of the 14th IFAC World Congress, *Pages(7—12)*, 1999.

[14] H. Lin, and P., J. Antsaklis, *Hybrid dynamical systems: An introduction to control and verification. Found. Trends Syst. Control*. volume(1). *number(1).* pages(1–172). 2014.

[15] J. Jaffar and M. J Maher *Constraint logic programming: A survey*, The journal of logic programming, *Volume (19)*, Pages (503—581), Elsevier, *1994*.

[16] J-J Ch Meyer, and W. Van Der H., *Epistemic logic for AI and computer science*. volume(41). *Cambridge University Press*. 2004.

[17] J. van Benthem, and E. Pacuit, *Temporal Logics of Agency*. Journal of Logic, Language and Information. *volume(19).* number(4). *pages(389–393).* Springer, 2010.

[18] K. R. Apt. and M. Wallace. *Constraint Logic Programming Using Eclipse*. Cambridge University Press, Cambridge, UK, 2007.

[19] T., A. Henzinger, *The theory of hybrid automata*. Springer, 2000.

[20] T. Hickey and D. Wittenberg, *Rigorous modeling of hybrid systems using interval arithmetic constraints*, International Workshop on Hybrid Systems Computation and Control, *pages(402–416)*, Springer,2004,.

[21] P. Gammie and R. van der Meyden. *MCK: Model checking the logic of knowledge*. In R. Alur and D. Peled, editors, *Proceedings of the 16thInternational Conference on Computer Aided Verification (CAV 2004)*, P(479–483). 2004.

[22] R. Alur, and T. A. Henzinger,. *Logics and models of real time: A survey. Real-Time: Theory in Practice*. pages(74–106). 1992.

[23] W.,Verbrugge, Epistemic logic: *A survey. Game theory and applications*. volume(8). *pages(53).* Nova Science Publishers. Springer. 2002.

[24] Z. Manna and A. Pnueli *Temporal verification of reactive systems: safety*, Springer Science and Business Media. 2012,