Towards an Argumentative Approach for Repair of Hybrid Logics Models

Anca Goron[†], Adrian Groza[†], Sergio Alejandro Gómez[‡] and Ioan Alfred Letia[†]

[†]Intelligent Systems Group
Department of Computer Science
Technical University of Cluj-Napoca
Baritiu 28, 400391, Cluj-Napoca, Romania
EMAIL: {Anca.Goron,Adrian.Groza,letia}@cs.utcluj.ro
[‡]Artificial Intelligence Research and Development Laboratory (LIDIA)
Department of Computer Science and Engineering
Universidad Nacional del Sur
Av. Alem 1253, (8000) Bahía Blanca, ARGENTINA
EMAIL: sag@cs.uns.edu.ar

Abstract. We propose an argumentation approach for hybrid logics model update. Argumentation theory is used to assist the process of updating the model. We view a Hybrid Kripke model as a description of the world that we are interested in. The update on this Kripke model occurs when the system has to accommodate some newly desired properties or norm constraints. When the model fails to verify a property, a defeasible logic program is used to analyze the current state. Depending on the status of the arguments, the system can warrant four primitive operations on the model: updating state variables, adding a new transition, removing a transition, or adding a new state. A running scenario is presented showing the verification of an unmanned aerial vehicle, by interleaving reasoning in Defeasible Logic Programming and the Hybrid Logic Model Checker.

1 Introduction

Model checking tools lack mechanisms to assist the user in the revision of a Kripke model. We propose to integrate model checking and structured argumentation towards supporting automated change of the model. The main application of our proposal is in the field of autonomous systems, by empowering agents with self-verification capabilities after they have updated the world model.

Safety assurance and compliance to safety standards-based methods of certification such as DO-178B [15] or HACCP [14] may prove to be a real challenge when we deal with adaptive systems, in which we consider with continuous changes and without a strict behavioral model. Traditional methods, which are mainly based on previous experiences and lessons learned from other systems are not effective in this case. Argument-based safety cases offer a plausible alternative basis for certification in these fast-moving fields [15].

Assuring safety in complex technical systems is a crucial issue [10] in several critical applications like air traffic control or medical devices. Autonomous agents, as embodied by entities such as unmanned aerial vehicles (UAVs), unmanned ground vehicles (UGVs), or software-agents, are expected to be compliant with a set of requirements and specifications. An air traffic control system is a system where accidents are mainly produced by human errors. Such accidents can be avoided by verifying the safety for the air control system in a logical manner in order to produce support for human air controllers to make rationally justified decisions.

Our hypothesis is that argumentation can be used to assure safety in complex critical systems by providing guidance to update the model of the world in case of contradictory information.

We will further use Hybrid Logics (HLs) [1, 6] for formalizing safety cases such that they can be further subjected to a complete verification. Although Linear Temporal Logic (LTL) or Computation Tree Logic (CTL) are generally used in such cases, we consider HL as a better alternative due to the higher degree of expressivity and with the advantages it brings along through the inclusion of nominals and specific operators such as @ and the $\downarrow$ binder. All the automatic verifications in our approach are to be performed by the model checker HLMC[1], that implements the model checking algorithms for the hybrid logics MCLite and MCFull [8]. The argumentative reasoning is performed in DeLP [9].

The rest of the paper is structured as follows. In Section 2 we present the fundamentals of model checking with Hybrid Logics and argumentation with Defeasible Logic Programming. In Section 3 we combine argumentation and model checking for implementing model update in Hybrid Logics. In Section 4 we present a case study where our approach is applied to model repair in an unmanned aerial vehicle. In Section 5 we discuss related work. Finally, in Section 6 we conclude the paper and outline possible future research lines.

2 Technical Instrumentation

2.1 Model Checking with Hybrid Logics

Model checking [13] refers to the problem of verifying, given a certain model, whether different properties hold for that model. Properties are represented using formulas usually specified in different types of logic languages such as Linear Temporal Logics, Description Logics or Hybrid Logics, while the model is given as a labeled graph known as a Kripke structure.

Temporal logics (TL) (see [8] for details) extend propositional logics with temporal operators future F, past P, until U, since S, so that with the set of propositional symbols $\mathcal{P} = \{p_1, p_2, \ldots\}$, the syntax of temporal logic is the one below

$$\varphi := \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathrm{F}\varphi \mid \mathrm{P}\varphi \mid \varphi \mathrm{U} \varphi \mid \varphi \mathrm{S} \varphi$$

The dual of P is $\mathrm{H}\alpha = \neg\mathrm{P}\neg\alpha$ and the dual of F is $\mathrm{G}\alpha = \neg\mathrm{F}\neg\alpha$.

[1] Available at http://luigidragone.com/software/hybrid-logics-model-checker/

The semantics of TL is presented in Figure 1, where $\mathcal{M} = \langle M, R, V \rangle$ is a Kripke structure, $m \in M$, and g is an assignment.

$$
\begin{array}{lll}
\mathcal{M}, m \models \top & & \\
\mathcal{M}, m \models p & \text{iff} & m \in V(p) \quad \text{for } p \in \mathcal{N} \\
\mathcal{M}, m \models \neg\varphi & \text{iff} & \mathcal{M}, m \not\models \varphi \\
\mathcal{M}, m \models \varphi \wedge \psi & \text{iff} & \mathcal{M}, m \vDash \varphi \text{ and } \mathcal{M}, m \vDash \psi \\
\mathcal{M}, m \models \mathrm{F}\varphi & \text{iff} & \exists m' \ ((m, m') \in R \wedge \mathcal{M}, m' \models \varphi) \\
\mathcal{M}, m \models \mathrm{P}\varphi & \text{iff} & \exists m' \ ((m, m') \in R \wedge \mathcal{M}, m' \models \varphi) \\
\mathcal{M}, m \models \psi \mathrm{U} \varphi & \text{iff} & \exists m' \ ((m, m') \in R \wedge \mathcal{M}, m' \models \varphi \wedge \\
 & & \forall m" \ ((m, m'') \in R \wedge (m'', m') \in R \rightarrow \mathcal{M}, m" \models \psi)) \\
\mathcal{M}, m \models \psi \mathrm{S} \varphi & \text{iff} & \exists m' \ ((m, m') \in R \wedge \mathcal{M}, m' \models \varphi \wedge \\
 & & \forall m" \ ((m, m') \in R \wedge (m', m'') \in R \rightarrow \mathcal{M}, m" \models \psi))
\end{array}
$$

Fig. 1. Semantics for temporal logic.

Hybrid logics (HL) extend temporal logics with special symbols that name individual states and access states by name [1]. With nominal symbols $\mathcal{N} = \{i_1, i_2, \ldots\}$ called *nominals* and $\mathcal{S}_{\text{var}} = \{x_1, x_2, \ldots\}$ called *state variables* the syntax of hybrid logics is shown below.

$$\varphi := TL \mid i \mid x \mid @x_t\varphi \mid \downarrow x.\varphi \mid \exists x.\varphi$$

With $i \in \mathcal{N}$, $x \in \mathcal{W}_{\text{var}}$, $t \in \mathcal{N} \cup \mathcal{W}_{\text{sym}}$, the set of *state symbols* $\mathcal{W}_{\text{sym}} = \mathcal{N} \cup \mathcal{W}_{\text{var}}$, the set of *atomic letters* $\mathcal{A}_{\text{let}} = \mathcal{P} \cup \mathcal{N}$, and the set of *atoms* $\mathcal{A} = \mathcal{P} \cup \mathcal{N} \cup \mathcal{W}_{\text{var}}$, the operators $@, \downarrow, \exists$ are called *hybrid operators*.

Definition 1. *A hybrid Kripke structure* $\mathcal{M}$ *consists of an infinite sequence of states* $m_1, m_2, ...,$ R *a family of binary accessibility relations on* $\mathcal{M}$ *and a valuation function* V *that maps ordinary propositions and nominals to the set of states in which they hold, i.e.* $\mathcal{M} = \langle \langle m_1, m_2 ... \rangle, R, V \rangle$.

In the graph-based representation of $\mathcal{M}$, the nodes correspond to the sequence of states brought about by different modalities represented as links between states. Each state is labeled by a different nominal, while links are labeled by the relation connecting two states.

Figure 2 presents the semantics of hybrid logic used in our approach, where the hybrid logic operator $@_t$ is used to shift the evaluation point to the state named by the nominal or state variable t, the downarrow binder $\downarrow x$ to assign the state variable x to the current state of evaluation and the existential binder $\exists x$ to refer to some state in the model $\mathcal{M}$ using the state variable x.

Example 1 (Safe landing). A landing is considered safe if all possible major risks are identified and managed. For this case here we will consider wind speed, remaining fuel and critical failure as possible risks. To ease the validation process,

$\mathcal{M}, g, m \models a$ iff $m \in [V, g](a)$, $a \in \mathcal{A}$
$\mathcal{M}, g, m \models @_t\varphi$ iff $\mathcal{M}, g, m' \models \varphi$, where $[V, g](t) = \{m'\}, t \in W_{sym}$
$\mathcal{M}, g, w \models \downarrow x.\varphi$ iff $\mathcal{M}, g^x_m, w \models \varphi$
$\mathcal{M}, g, m \models \exists x.\varphi$ iff there is $m' \in \mathcal{M}$ such that $\mathcal{M}, g^x_{m'}, w \models \varphi$

Fig. 2. Semantics for hybrid logic.

we will assume that the sensors used to measure the required data are failure-free and the two streams of sensor data received return accurate information. We know that a flight has always clearance to land no matter the wind speed and remaining fuel criteria if it has had a critical failure:

$$\downarrow x.\mathit{critical\_failure}([\mathrm{F}]x \rightarrow [\mathit{clearence}]) \tag{1}$$

The formula states that if there is a state in which critical failure is encountered, then for all upcoming states, clearance should be selected.

Denying the clearance to land for a flight implies that it is forbidden to land as long as there is no critical failure and viceversa.

$$\langle\neg(\mathit{clearence})\rangle \rightarrow \mathit{forbidden}\ \mathrm{U}\ \mathit{critical\_failure} \tag{2}$$

We also know that a flight is allowed to land at time T on runway A if the wind is calm on runway A at time T and it is forbidden if it is windy at that time. A runway is windy at time T if the wind speed is greater than 15 knots. Exceptionally a flight is not forbidden to land on a windy runway if it has fuel trouble. A flight is considered to have fuel trouble if its remaining fuel allows it to fly less than 15 minutes, otherwise the flight has no fuel trouble:

$$\downarrow x(\mathit{windSpeed} > \mathit{speed}_{threshold})(@_x\neg(\mathit{forbidden}) \rightarrow \mathit{remainingFuel} < \mathit{fuel}_{threshold}) \tag{3}$$

We also know that the wind speed, the remaining fuel and critical failure are measured values and the measures are considered accurate ($\mathit{measured} \rightarrow \top$).

$$[\mathit{windSpeed} > \mathit{speed}_{threshold}] \rightarrow \mathit{measured} \tag{4}$$

$$[\mathit{remainingFuel} < \mathit{fuel}_{threshold}] \rightarrow \mathit{measured} \tag{5}$$

$$[\mathit{critical\_failure}] \rightarrow \mathit{measured} \tag{6}$$

One can observe that knowing that formulas (4), (5), respectively (1) will always denote valid values as the measurements are considered accurate, then also formula (3), respectively (2) and (6) can be validated.

2.2 Defeasible Logic Programming

Defeasible Logic Programming (DeLP) [9] provides a language for knowledge representation and reasoning that uses *defeasible argumentation* [5, 2] to decide between contradictory conclusions through a *dialectical analysis*. Codifying knowledge by means of a DeLP program provides a good trade-off between expressiveness and implementability for dealing with incomplete and potentially contradictory information. In a defeasible logic program $\mathcal{P} = (\Pi, \Delta)$, a set Π of strict rules $P \leftarrow Q_1, \ldots, Q_n$, and a set Δ of defeasible rules $P \prec Q_1, \ldots, Q_n$ can be distinguished.

Definition 2. *An* argument $\langle \mathcal{A}, H \rangle$ *is a minimal non-contradictory set of ground defeasible clauses* $\mathcal{A}$ *of* Δ *that allows to derive a ground literal* H *possibly using ground rules of* Π*.*

Since arguments may be in conflict (concept captured in terms of a logical contradiction), an attack relationship between arguments can be defined. A criterion is usually defined to decide between two conflicting arguments. If the attacking argument is strictly preferred over the attacked one, then it is called a *proper defeater*. If no comparison is possible, or both arguments are equi-preferred, the attacking argument is called a *blocking defeater*. In order to determine whether a given argument $\mathcal{A}$ is ultimately undefeated (or *warranted*), a dialectical process is recursively carried out, where defeaters for $\mathcal{A}$, defeaters for these defeaters, and so on, are taken into account. Given a DeLP program $\mathcal{P}$ and a query H, the final answer to H w.r.t. $\mathcal{P}$ takes such dialectical analysis into account. The answer to a query can be: *yes*, *no*, *undecided*, or *unknown*.

Example 2 (Safety landing).

The main system safety verification is clearance for landing denoted by the literal $clearance(ID, A, T)$ meaning that flight ID has clearance to land on runway A at time T. For flight *f701* we know that the wind speed at runway *r01* is 3 knots at time 10. For the prospective decision $clearance(f701, r01, 10)$ there exists a warranted argument $\langle \mathcal{A}_1, clearance(f701, r01, 10) \rangle$ where:

$$\mathcal{A}_1 = \left\{ \begin{array}{l} clearance(f701, r01, 10) \prec \\ \qquad \sim forbidden(f701, r01, 10), \\ \qquad runway(r01), flight(f701) \\ \sim forbidden(f701, r01, 10) \prec calm(r01, 10) \\ calm(r01, 10) \prec wind\_speed(r01, 3, 10), 3 < 15 \end{array} \right\},$$

meaning that flight *f701* has permission to land on runway *r01* because it is a calm runway as the wind speed is only 3 knots at time 10.

3 Interleaving Argumentation and Model Checking

Given a Kripke structure $\mathcal{M}$ and a formula ϕ, with $\mathcal{M}\neg \vDash \phi$, the task of *model repair* is to obtain a new model $\mathcal{M}'$ such that $\mathcal{M}' \vDash \phi$. We consider the following primitive update operations [18].

Definition 3 (Primitive update operations). *Given $\mathcal{M} = (S, R, L)$, the updated model $\mathcal{M} = (S', R', L')$ is obtained from $\mathcal{M}$ by:*

1. *(PU_1) Adding one relation element: $S' = S$, $L' = L$, and $R' = R \cup \{(s_i, s_j)\}$ where $(s_i, s_j) \notin R$ for two states $s_i, s_j \in S$.*
2. *(PU_2) Removing one relation element: $S' = S$, $L' = L$, and $R' = R \setminus \{(s_i, s_j)\}$ where $(s_i, s_j) \notin R$ for two states $s_i, s_j \in S$.*
3. *(PU_3) Changing labeling function in one state: $S' = S$, $R' = R$, $s^\star \in SL'(s^*) \neq L(s^*)$, and $L'(s) = L(s)$ for all states $s \in S \setminus \{s^*\}$.*
4. *(PU_4) Adding one state: $S' = S \cup \{s^*\}, s \notin S$, $R' = R$, $\forall s \in S, L'(s) = L(s)$.*

Our task is to build an argumentative based decision procedure that takes as input a model $\mathcal{M}$ and a formula ϕ, it outputs a model $\mathcal{M}'$ where ϕ is satisfied. The task addressed here focuses on a situation on which the specification of the model is not consistent. Consider the following two "rules of the air" [16]:

R_3: *Collision Avoidance – "When two UAVs are approaching each other and there is a danger of collision, each shall change its course by turning to the right."*

R_4: *Navigation in Aerodrome Airspace – "An unmanned aerial vehicle passing through an aerodrome airspace must make all turns to the left unless [told otherwise]."*

Let

$$\mathcal{A}_2 = \left\{ \begin{array}{l} alter\_course(uav_1, right) \prec aircraft(uav_1), aircraft(uav_2) \\ \qquad collision\_hazard(uav_1, uav_2) \\ collision\_hazard(uav_1, uav_2) \prec approaching\_head\_on(uav_1, uav_2), \\ \qquad distance(uav_1, uav_2, X), X < 1000 \end{array} \right\}$$

in the argument $\langle \mathcal{A}_2, alter\_course(uav_1, right) \rangle$, a collision hazard occurs when two aerial vehicles uav_1 and uav_2 approach head on, and the distance between them is smaller than a threshold. The collision hazard further triggers the necessity to alter the course to the right, according to the R_3 specification. Let

$$\mathcal{A}_3 = \left\{ \begin{array}{l} alter\_course(uav_1, left) \prec aircraft(uav_1), nearby(uav_1, aerodrom), \\ \qquad change\_direction\_required(uav_1) \\ change\_direction\_required(uav_1) \prec collision\_hazard(uav_1, uav_2) \end{array} \right\}$$

in the argument $\langle \mathcal{A}_3, alter\_course(uav_1, left) \rangle$, if a change of direction is required in the aerodrome airspace, the direction should be altered to the left. A possible conflict occurs between arguments $\langle \mathcal{A}_2, alter\_course(uav_1, right) \rangle$ and $\langle \mathcal{A}_4, \sim alter\_course(uav_1, right) \rangle$ where:

$$\mathcal{A}_4 = \left\{ \sim alter\_course(uav_1, right) \prec alter\_course(uav_1, left) \right\}.$$

The command $\langle \mathcal{A}_5, \sim alter\_course(uav_1, left) \rangle$ conveyed from the ground control system to change direction to the right acts as a defeater for the argument

$\mathcal{A}_3$, where (notice that strict rules should not form part of argument structures as they are not points of attack, we abuse the notation here just for emphasis):

$$\mathcal{A}_5 = \left\{ \sim alter\_course(uav_1, left) \leftarrow conveyed\_command\_course(uav_1, right) \right\}$$

Assume that the current model $\mathcal{M}$ satisfies the specification R_3. The problem is how to repair $\mathcal{M}$ with the model $\mathcal{M}'$ which also satisfies R_4. Our solution starts by treating rules R_3 and R_4 as structured arguments. The conflict between them are solved by a defeasible theory encapsulated as DeLP program, which outputs a dialectical tree of the argumentation process. The information from this tree is further exploited to decide which primitive update operations PU_i are required to repair the model.

Firstly, consider the uav_1 is in the obstacle detect $od \in S$ state, where S is the set of states in $\mathcal{M}$ with the labeling function $L(od) = \{uav_2, \neg a\}$. It means that uav_1 has detected another aerial vehicle uav_2. Assume that in this state the DeLP program will warrant the opposite conclusion a. This triggers the application of the primitive operation PU_3 which updates the labeling function $L(od) = \{uav_2, \neg a\}$ with $L'(od) = \{uav_2, a\}$.

Secondly, assume that the DeLP program based on the state variables uav_2, and $\neg a$ and the nominal od infers a relation r_i between od and another nominal $i \in \mathcal{N}$ of the model. The repair consists of applying the operation PU_1 on $\mathcal{M}$, where the relation set R' is extended with a relation between the two states ob and i: $R' = R \cup \{(od, i)\}$. The reasoning mechanism is possible because hybrid logic provides the possibility to directly refer to the states in the model, by means of nominals.

Thirdly, the program can block the derivation of a relation r between the current state and a next state. For instance, if $L(od) = \{uav_2, a\}$ and the argument A_3 succeeds, the transition between state od and state $turn\_right$ can be removed. Formally, $R' = R \setminus \{(od, turn\_right)\}$.

Fourthly, if the DeLP program warrants, based on the current state variable and available arguments, a nominal i which does not appear in S, the set of states is extended with this state: $S' = S \cup \{i\}$.

These four heuristics are illustrated in the following section, by verifying the specifications in hybrid logics on the updated models.

4 Model Repair for an Unmanned Aircraft Vehicle

4.1 Illustrative Example

We consider the scenario presented in [17], referring to the safe insertion of an Unmanned Aircraft Vehicle (UAV) into the civil air traffic. The scope is to demonstrate that safety requirements are being met by such an UAV so that they do not interfere or put in danger human controlled aircrafts. A mission is considered safe if all the major risks for the UAV are identified and managed (e.g. collision with other objects or human-piloted aircrafts and loss of critical

functions). An UAV comes equipped with an autonomous control system, responsible for decision making during the mission and keeps a communication link open with a ground-base system (GBS), which provides all the required coordinates for the UAV. The autonomous decision making performed by the UAV control system must consider the general set of safety regulations imposed to a UAS during a mission at all times.

We propose a solution for modeling such Unmanned Aircraft Systems (UASs) in compliance to the set of safety regulations. We will zoom over the following subset of the "Rules of the Air" dealing with collision avoidance:

- R_1: *Obstacle Detection – "All obstacles must be detected within an acceptable distance to allow performing safely the obstacle avoidance maneuver"*
- R_2: *Obstacle Avoidance – "All obstacles must be avoided by performing safely a slight deviation from the preestablished path and an immediate return to the initial trajectory once all collision risks are eliminated."*
- R_3: *Collision Avoidance – "When two UAVs are approaching each other and there is a danger of collision, each shall change its course by turning to the right."*

The first rule states that all obstacles (e.g. human-controlled aircrafts, other UAVs, etc.) that are interfering with the initial trajectory of the UAV must be signaled within a certain limit of time such that to allow avoidance maneuvers to be performed by the UAV in safe conditions. The avoidance maneuver as shown by rules R_2 and R_3 consists of a slight change of the initial path to the right such that to allow the safe avoidance of the approaching UAV followed by a repositioning on the initial trajectory.

4.2 Kripke Model for the Unmanned Aerial Vehicle

We will further represent the behavior of the UAV noted by uav_1 captured in an obstacle avoidance scenario. The following states will be considered in constructing the Kripke model: path-following (pf), obstacle detection(od), turn left(tl) and turn right(tr). To each state we will attach the boolean state variable uav_2, which will indicate the presence or absence of another approaching UAV. In the path-following state pf, the UAV uav_1 performs a waypoint following maneuver, which includes periodical turns to the left or to the right. The appearance of an obstacle ($uav \rightarrow \top$) leads to the transition of the UAV into obstacle detection state od and from there in turn right tr state as part of the obstacle avoidance maneuver, followed by a return to the initial path-following state.

The initial model $\mathcal{M}_0$ is presented below:

$$\mathcal{M}_0 = \langle \{od, tr, tl, pf\},$$

$$\{r_0, r_1, r_2, r_3, r_4, r_5, r_6\},$$

$$\{(pf, \{\neg uav_2\}), (od, \{uav_2\}), (tr, \{\neg uav_2\}), (tl, \{\neg uav_2\})\}\rangle$$

The corresponding hybrid Kripke structure is illustrated in Figure 3.

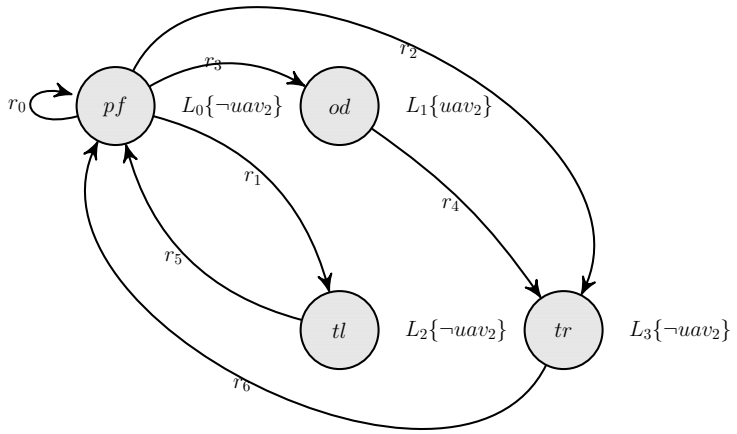

Fig. 3. Kripke Model for the UAV.

4.3 Verifying Compliance to Safety Regulations

Once the modeling of the UAS is done, we have to verify whether the mentioned safety regulations hold for this model. To be able to perform model checking, we will further express the two safety regulations using hybrid logics:

$$R_1 : [Next](od) \rightarrow tr \tag{7}$$

The above formula corresponds to the first safety regulation R_1 and states that once the *od* (*ObstacleDetect*) state is reached then the immediate transition step should be done towards an avoidance maneuver state, for our case here, state *tr*, meaning that the obstacle was detected in time and it allowed the avoidance maneuver to be safely performed.

$$R_2 : [Next](tr \vee tl) \rightarrow pf \tag{8}$$

The formula corresponding to safety regulation R_2 states that all the next transitions from the $TurnRight$ or $TurnLeft$ state should always lead to the $PathFollow$ state.

The formula below corresponding to safety regulation R_3 states that if another UAV is detected in the *od* (*ObstacleDetect*) state then all next transitions should be done towards the state *tr* ($TurnRight$):

$$R_3 : @_{od} uav_2 \rightarrow ([Next]od \rightarrow tr) \tag{9}$$

Model checking is performed to verify whether the formulas hold or not for that model. To perform the model checking automatically, the Kripke structure corresponding to the UAS model is translated into an XML file and given as input for the Hybrid Logic Model Checker (HLMC) [8]. Each formula in HL is also given as input to the HLMC. Once the tests are performed for each formula against the Kripke model, we can complete the verification of the model. The result confirms that the modeled Kripke structure of the UAS complies with the defined safety regulations.

4.4 Adapting the Model to New Specifications

We consider again the UAV scenario and we will present a solution for modeling the existing UAS to include the introduction of new rules. For this, we will consider the initial set of rules extended by a newly adopted norm for UAVs navigating in an Aerodrome Airspace:

> R_4: *Navigation in Aerodrome Airspace – "An unmanned aerial vehicle passing through an aerodrome airspace must make all turns to the left [unless told otherwise]."*

As a first step we will check whether the existing UAS model complies to the new regulation R_4. For this we will express the new rule as a HL formula and we will add to each possible state the boolean variable a, which will become true when the UAV enters an aerodrome airspace:

$$R_4 : @_i a \rightarrow ([Next]i \rightarrow (\neg tr)) \tag{10}$$

The formula states that all transitions from the states in which the state variable aerodrome a holds should not lead to the tr ($TurnRight$) state, the only state which is forbidden when navigating inside the aerodrome space. Since the only states from which turns are possible are pf and od, we will consider only this subset for model checking. One can observe that the formula does not hold for the existing model. Considering that the aerodrome a state variable is true in the od ($ObstacleDetect$) state, one can observe that the only allowed transition in the current model is to the tr ($TurnRight$) state. Therefore, the existing model does not comply to the new regulation. Moreover, from the pf state transitions are possible to the tl ($TurnLeft$) state, but also to the tr ($TurnRight$) states. We argue that the existing model could be extended to include also the new rules without having to construct a new model from the beginning. Although different solutions were proposed for Kripke Model repairing [4], we propose a solution based on argumentation for extending the model such that it complies to the updated set of regulations.

As a first step in our approach, we represent several possible extensions to the Kripke Model as defeasible arguments and include them in DeLP for choosing the best possible solution between different conflicting arguments. The proposed

solution does not only eliminate the complexity of proposed repair/updating algorithms [4], but it allows the system to adapt to new information in a faster and more efficient manner.

Going back to our example, one can observe that there is no possibility for the UAV to go into the *tl* state once it has reached the *od* state, but only to the *tr* state. Since inside the aerodrome space, only turns to the left are permitted, then the link connecting *od* and *tr* (r_4) should be taken out from the model.

We will consider a new argument $\langle \mathcal{A}_6, alter\_course(uav_1, left)\rangle$, which suggests updating rule R_3 by allowing the obstacles to be avoided to the left, instead of to the right when inside the aerodrome space, where:

$$\mathcal{A}_6 = \left\{ \begin{array}{l} alter\_course(uav_1, left) \prec aircraft(uav_1), aircraft(uav_2) \\ \qquad collision\_hazard(uav_1, uav_2) nearby(uav_1, aerodrom) \\ collision\_hazard(uav_1, uav_2) \prec approaching\_head\_on(uav_1, uav_2), \\ \qquad distance(uav_1, uav_2, X), X < 1000 \end{array} \right\}.$$

We argue that for compliance to the new regulations, we only need to change all the links in the model to point from the *od* and pf states only to the *tl* state instead of *tr* state to avoid the collision.

Therefore, we need to perform the following PU operations for updating the model:

1. (PU_2) Remove the relation elements (od, tr) and (pf, tr) such that we have: $S' = S$, $L' = L$, and $R' = R \setminus \{(od, tr), (pf, tr)\}$
2. (PU_1) Add the relation element (of, tl) such that we have: $S'' = S'$, $L'' = L'$, and $R'' = R' \cup \{(od, tl)\}$

However, the remove operation should be necessary only when that specific relation element causes a conflict between two arguments. In our case, if we consider arguments $\mathcal{A}_2$, sustaining the application of the initial rule R_2 and $\mathcal{A}_6$, sustaining a slight modification of the rule R_2 for navigation in aerodrome space, one can see that they do not attack each other as they offer solutions for different contexts: the $\mathcal{A}_2$ argument refers to collision avoidance outside the aerodrome space, while the $\mathcal{A}_6$ argument considers the case of collision avoidance when the UAV is nearby an aerodrome. A similar reasoning applies for the transition (pf, tr), which will be possible only when the state variable a does not hold at pf. Therefore, the PU_2 step can be left out and the updating of the model can be done only through a PU_1 operation. The decision to turn left or turn right will be taken in accordance to the value of the state variable a, which indicates the presence or absence of an aerodrome in the vicinity of the UAV.

We illustrate the update operation by adding a link r_7 from the *od* state to the *tl* state. Additionally, we attach to each state the boolean state variable a, such that it allows the UAV to perform only those transitions that comply to the set of regulations in different contexts, respectively inside or outside the aerodrome space. One can observe that if the UAV reaches the *od* state, then it will decide to perform the transition to the next state that has the same value for the state variable a as the *od* state. Therefore, if the UAV uav_1 detects another

approaching UAV uav_2 and it is outside the aerodrome space ($\neg a$), it will look for the next possible state that has the same value for the a state variable. As one can see from Figure 4, the state that complies to this condition is tr. Also, if uav_1 is in the pf state and the state variable a holds at that state, then the possible transitions will be tl or od.

If uav_1 reaches the od state, while in the vicinity of an aerodrome, it will perform a transition to the tl state, where the state variable a also holds. If uav_1 reaches pf then it will perform a transition to either tl or od states. The other transitions from the model are not dependent on the state variable a, therefore they will remain the same as in the initial model. By adding the condition $\neg a$ for reaching state tr, we can avoid transitions to that state when a holds for the model.

The updated model $\mathcal{M}_1$ is presented below:

$$\mathcal{M}_1 = \langle \{od, tr, tl, pf\},$$

$$\{r_0, r_1, r_2, r_3, r_4, r_5, r_6, r_7\},$$

$$\{(pf, \{\neg uav_2\}), (od, \{uav_2\}), (tr, \{\neg uav_2, \neg a\}), (tl, \{\neg uav_2\})\}\rangle$$

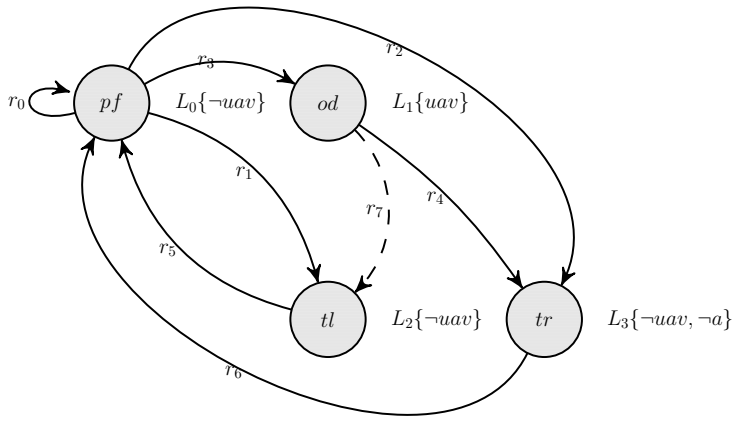

Fig. 4. Extended Kripke model for the UAV compliant with the new regulation.

By checking next the R_1, R_2, R_3 and R_4 formulas against $\mathcal{M}$, the results returned by HLMC showed that they hold for the updated model.

The illustrated example captures a simple scenario for UAV missions, but we argue that more complex conflicting situations can be handled by the presented argumentation framework.

5 Discussion and Related Work

A tighter connection between argumentation frameworks and model checking is given by means of modal logic. Basic notions of argumentation theory like acceptability, admissibility, complete or stable extensions have been formalized in the modal logic K extended with universal modality [11]. Consequently, the relationship between dialogue games and model checking games has been further investigated, where model checking games have been used to prove that an argument belongs to a certain extension of the argumentation framework. Our research is in the line opened by Grossi [12, 11] on interleaving argumentation theory with modal logics. The approach of Grossi translates an argumentation framework into a Kripke structure, while in our case we use structured arguments aiming at automatic model repair.

An orthogonal connection between model checking and argumentation theory is given by assurance cases in the Goal Structuring Notation (GSN). In this software engineering related approach, the arguments use the outputs of model checking as evidence to support general claims (or goals) like safety in different operative contexts. Thus, the safety argumentative cases in GSN is based on arguments on top of model checking. In our case, after each argumentation step, the new model is verified against the available specifications.

Program verification is used in [7] to prove that a rational agent always acts in line with the specifications and never chooses actions it believes that lead to unsafe situations. Hence, the verification regards the decisions of the agent and not the effects of these decisions in the environment. Thus an agent behavior is considered safe if all the available relevant information has been used to take the best decision for the moment. The computation tree logic is applied on the BDI model to verify sentences like: "the agents choose action they *believe* to be good", or "the agents never select an *intention* they *believe* will lead to something bad". In our case, the checker would verify that a proper argumentation tree has been generated for each decision, where "proper" means that the dialectical tree should satisfy specific requirements.

Checking of temporal properties of a model has been the focus of many research work. Linear Temporal Logics has been widely used in this direction. However, the advantages it brings on what it concerns the use of temporal operators is sometimes shadowed by the limitations encountered on what it concerns the knowledge and state based representation of the model. It lacks mechanisms for naming states, for accessing states by names, and for dynamically creating new names for states [8]. Hybrid Logics [13] comes as a solution in this direction as it allows to refer to states in a truly modal framework, mixing features

from first-order logics and modal logics. But hybridization is not simply about quantifying over states. Rather, hybridization is about handling different type of information in a uniform way [3]. This is useful in model checking where we need to combine different types of information to be verified against a model.

6 Conclusion

In this paper we presented preliminary work on applying argumentation theory to the task of model repair. We proposed using Defeasible Logic Programming for hybrid logics model update. In particular we view a Hybrid Kripke model as a description of the world that we are interested in and the update on this Kripke model occurs when the system has to accommodate some newly desired properties or norm constraints. The argumentation theory then acts as a control mechanism during the adaptive process. The main application of our approach is in autonomous systems, which should safely adapt their behavior to the new specifications. We presented a case study where our proposal is applied to assisting the flight decisions in an unmanned aerial vehicle. As part of our current research work we are formalizing these preliminary ideas, considering also communication between unmanned aerial vehicles.

Acknowledgments

Part of this work was supported by the Romania-Argentina Bilateral Agreement entitled "ARGSAFE: Using argumentation for justifying safeness in complex technical systems" (MINCYT-MECTS Project RO/12/05) and Universidad Nacional del Sur, Argentina. Adrian Groza is supported by the intern research project at Technical University of Cluj-Napoca, Romania: "GREEN-VANETS: Improving transportation using Car-2-X communication and multi agent systems".

References

1. Areces, C., ten Cate, B.: Hybrid logics. In: Blackburn, P., Van Benthem, J., Wolter, F. (eds.) Handbook of Modal Logic, pp. 821–868. Elsevier Amsterdam (2007)
2. Bench-Capon, T.J.M., Dunne, P.E.: Argumentation in Artificial Intelligence. Artif. Intell. 171(10-15), 619–641 (2007)
3. Blackburn, P., Tzakova, M.: Hybrid languages and temporal logic. Logic Journal of IGPL 7(1), 27–54 (1999)
4. Chatzieleftheriou, G., Bonakdarpour, B., Smolka, S., Katsaros, P.: Abstract model repair. In: Goodloe, A., Person, S. (eds.) NASA Formal Methods, Lecture Notes in Computer Science, vol. 7226, pp. 341–355. Springer Berlin Heidelberg (2012)
5. Chesñevar, C.I., Maguitman, A., Loui, R.: Logical Models of Argument. ACM Computing Surveys 32(4), 337–383 (Dec 2000)
6. Cranefield, S., Winikoff, M.: Verifying social expectations by model checking truncated paths. Journal of Logic and Computation 21(6), 1217–1256 (2011)

7. Fisher, M., Dennis, L., Webster, M.: Verifying autonomous systems. Communications of the ACM 56(9), 84–93 (2013)
8. Franceschet, M., de Rijke, M.: Model checking hybrid logics (with an application to semistructured data). Journal of Applied Logic 4, 279–304 (2006)
9. García, A., Simari, G.: Defeasible Logic Programming an Argumentative Approach. Theory and Prac. of Logic Program. 4(1), 95–138 (2004)
10. Graydon, P., Habli, I., Hawkins, R., Kelly, T., Knight, J.: Arguing conformance. Software, IEEE 29(3), 50–57 (2012)
11. Grossi, D.: On the logic of argumentation theory. In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1. pp. 409–416. AAMAS '10, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2010)
12. Grossi, D.: Argumentation in the view of modal logic. In: Argumentation in Multi-Agent Systems, pp. 190–208. Springer (2011)
13. Lange, M.: Model checking for hybrid logic. Journal of Logic, Language and Information 18(4), 465–491 (2009)
14. Letia, I.A., Groza, A.: Compliance checking of integrated business processes. Data Knowl. Eng. 87, 1–18 (2013)
15. Rushby, J.: A safety-case approach for certifying adaptive systems. In: In AIAA Infotech@Aerospace Conference, American Institute of Aeronautics and Astronautics, John Rushby (2009)
16. Webster, M., Fisher, M., Cameron, N., Jump, M.: Formal methods for the certification of autonomous unmanned aircraft systems. In: Computer Safety, Reliability, and Security, pp. 228–242. Springer (2011)
17. Webster, M., Fisher, M., Cameron, N., Jump, M.: Model checking and the certification of autonomous unmanned aircraft systems. Tech. Rep. ULCS-11-001, Department of Computer Science, University of Liverpool, Liverpool, United Kingdom (2011)
18. Zhang, Y., Ding, Y.: CTL model update for system modifications. J. Artif. Intell. Res.(JAIR) 31, 113–155 (2008)