Modular SMT-Based Analysis of Nonlinear Hybrid Systems

Kyungmin Bae
POSTECH, Republic of Korea
kmbae@postech.ac.kr

Sicun Gao
University of California, San Diego, USA
sicung@ucsd.edu

***Abstract*—We present SMT-based techniques for analyzing networks of nonlinear hybrid systems, which interact with each other in both discrete and continuous ways. We propose a modular encoding method to reduce reachability problems of hybrid components, involving *continuous I/O* as well as usual discrete I/O, into the satisfiability of first-order logic formulas over the real numbers. We identify a generic class of logical formulas to modularly encode networks of hybrid systems, and present an SMT algorithm for checking the satisfiability of such logical formulas. The experimental results show that our techniques significantly increase the performance of SMT-based analysis for networks of nonlinear hybrid components.**

I. Introduction

Formal analysis of hybrid systems can be reduced to the satisfiability of SMT formulas over the real numbers. This approach can combine state-of-the-art SMT techniques with numerical methods to analyze continuous dynamics, governed by ordinary differential equations (ODEs). The satisfiability of these formulas are in general undecidable for nonlinear hybrid systems, but important advances have been made by various approaches, e.g., [1]–[6].

In principle, these methods can deal with networks of nonlinear hybrid systems by combining the SMT encodings of all components. Following SMT-based approaches for digital systems, *discrete communication* between components, such as synchronization or message passing, can be encoded using first-order variables that commonly occur in the encodings of several components. In order for this technique to work, all interactions between components must be discrete.

However, many networks of hybrid systems also include *continuous interaction* as well as discrete communication. For example, consider the problem of controlling the temperature of several adjacent rooms. The temperature of one room can continuously affect the temperature of all adjacent rooms. If we model this system as a network of several thermostat systems, it is quite clear that such continuous interactions cannot be captured only using discrete communication. This kind of *continuous I/O* is common in control systems that are composed of mechanically connected components; e.g, cars [7], airplanes [8], plants [9], etc. Indeed, formal models of hybrid systems, such as hybrid automata [10] and hybrid I/O automata [11], can precisely specify continuous interactions.

This work was partially supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2016R1D1A1B03935275), and NSF CPS-1446675.

The analysis of networks of nonlinear hybrid systems has not been studied much in existing SMT-based approaches. To apply existing SMT algorithms, one may define a sound discrete approximation of continuous I/O. But this is very difficult for nonlinear systems because continuous I/O can involve nonlinear functions, not just single values. Or one can build a single hybrid component that is equivalent to the entire network, but at the cost of the state explosion problem.

The goal of this paper is to provide an SMT technique to analyze networks of nonlinear hybrid systems involving continuous I/O as well as discrete I/O. The contribution of this paper is twofold: (1) to directly provide a new modular SMT encoding for networks of nonlinear hybrid systems with continuous I/O, and (2) to develop an SMT solving algorithm to check the satisfiability of such logical formulas.

The basic idea is to encode continuous interaction by means of *uninterpreted real functions*, not first-order variables. We then use *universally quantified equalities over time* to encode a continuous synchronization of uninterpreted real functions. To logically decompose continuous I/O expressed as systems of ODEs, we make use of *parameterized integration operators*. We identify a syntactic subclass of formulas that is expressive enough to modularly encode continuous I/O.

We present a novel SMT algorithm to check the satisfiability of the proposed class of formulas. Existing algorithms cannot deal with uninterpreted real functions, universally quantified equalities, and parameterized integration operators at the same time. Our algorithm is based on an equisatisfiable process that removes uninterpreted real functions and parameterized integration operators. This process can easily be combined with existing DPLL($\mathcal{T}$)-based algorithms with minimal cost.

We have implemented our algorithm in the dReal solver [12], and performed experiments on a range of nontrivial networks of nonlinear hybrid systems. These case studies include both discrete and continuous interactions between hybrid components. The experimental results show that our techniques can greatly increase the performance of SMT-based analysis for networks of general nonlinear hybrid systems.

The paper is organized as follows. Section II explains networks of hybrid systems. Section III proposes a modular encoding. Section IV presents an SMT solving algorithm. Section V provides an overview of the case studies. Section VI presents the experimental results. Section VII discusses related work, and Section VIII presents concluding remarks.

II. Network of Hybrid System Components

We consider a network of hybrid systems that interact with each other in discrete or continuous ways. As shown in Fig. 1, each component has two types of input and output; continuous I/O (denoted by bold lines) and discrete I/O (denoted by thin lines). A hybrid system can be specified by *hybrid automata* [10]. This paper uses an extended version of hybrid automata with explicit I/O, similar to one presented in [11].

A. Hybrid I/O Automata

In *hybrid I/O automata*, discrete states are given by a finite set of *modes* Q, and continuous states are specified by using a finite set of real-valued *variables* $X = \{x_1, \ldots, x_l\}$. A combined state is then a pair $\langle q, \boldsymbol{v}\rangle$ of a mode $q \in Q$ and a vector $\boldsymbol{v} = (v_1, \ldots, v_l) \in \mathbb{R}^l$ of real numbers. Given a (possibly infinite) set of actions Σ, a discrete transition between two states $\langle q, \boldsymbol{v}\rangle \xrightarrow{a} \langle q', \boldsymbol{v'}\rangle$, identified with an *action* $a \in \Sigma$, is specified by a *jump condition* $jump_{q,q'}(\boldsymbol{v}, a, \boldsymbol{v'})$.

Each mode $q \in Q$ of a hybrid I/O automaton defines extra conditions to specify the continuous behavior of the variables X in the mode q. An *invariant condition* inv_q defines all possible values of the variables X in mode q. A *flow condition* $flow_q$ defines *trajectories* of the variables X—describing continuous changes of X's values over time—in mode q, typically using ordinary differential equations (ODEs). An *initial condition* $init_q$ defines a set of initial states.

Discrete input and output of a component are identified by using disjoint sets of *input actions* $\Sigma_I \subseteq \Sigma$ and *output actions* $\Sigma_O \subseteq \Sigma$. Other "local" actions in $\Sigma \setminus (\Sigma_I \cup \Sigma_O)$ are called *internal actions*. Likewise, continuous input and output are identified by using disjoint sets of *input variables* $X_I \subseteq X$ and *output variables* $X_O \subseteq X$, and other "local" variables in $X \setminus (X_I \cup X_O)$ are called *internal variables*.

Definition 1. *A* hybrid I/O automaton *(HIOA) is defined as a tuple* $H = (Q, X, \Sigma, \{inv_q\}_{q\in Q}, \{flow_q\}_{q\in Q}, \{init_q\}_{q\in Q}, \{jump_{q,q'}\}_{q,q'\in Q}, (X_I, X_Y), (\Sigma_I, \Sigma_O))$.

The *init*, *inv*, and *jump* conditions are often written as predicates $init_q(\boldsymbol{x})$, $inv_q(\boldsymbol{x})$, and $jump_{q,q'}(\boldsymbol{x}, a, \boldsymbol{x'})$ over the variables X. The *flow* condition is written as a system of ODEs of the form $\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\iota})(t)$, where the input variables $\boldsymbol{\iota}$ in X_I appear as *free variables* in the ODEs.

A network of hybrid system components is specified by a parallel composition $\mathcal{H} = H_1 \parallel H_2 \parallel \cdots \parallel H_n$ of hybrid I/O automata. A communication between components is specified using I/O actions and variables. We assume that: (i) output actions and variables of one component are not output of any other components, and (ii) internal actions and variables of one component are not parts of any other components.

There are two types of communications in a network $\mathcal{H}$. Consider a source component H_o and target components $H_{i_1}, \ldots, H_{i_m}$. A *discrete communication* is modeled by *joint synchronous actions* in $\Sigma_O^o \cap \Sigma_I^{i_1} \cap \cdots \cap \Sigma_I^{i_m}$ that are output of the source and input of the targets. A *continuous interaction* is modeled by *shared variables* in $X_O^o \cap X_I^{i_1} \cap \cdots \cap X_I^{i_m}$ that are output of the source and input of the targets.

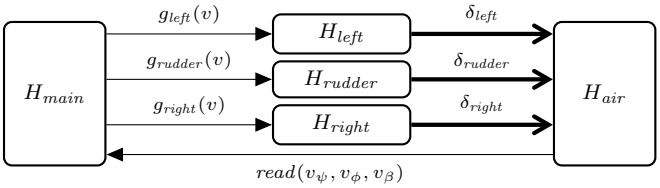

Fig. 1. Controllers for turning an airplane

A discrete state of a network $\mathcal{H}$ is defined as a vector $(q_1, \ldots, q_n)$ of modes of its components. Components in $\mathcal{H}$ synchronize their discrete transitions with a joint action. When one component performs a transition labeled with an action a, every component that includes the action a must perform a transition with the same action. A component can individually perform a discrete transition with an internal action.

A continuous state of a network $\mathcal{H}$ is defined as trajectories of the variables $X_1 \cup \cdots \cup X_n$ of its components. The values of $\mathcal{H}$'s variables evolve simultaneously over time according to their flow and invariant conditions. Flow conditions of an input variable are given by those of its connected output variable. That is, an output variable and its corresponding input variables must have the same value at all times.

Consequently, a network of HIOA is semantically equivalent to a single HIOA (see [11] for a formal definition).

B. Example: Turning an Airplane

We consider a networked controller for turning an airplane, adapted from [8], [13]. An aircraft makes a turn by controlling two *ailerons* (surfaces attached to the end of the wings) and a *rudder* (a surface attached to the vertical tail). As depicted in Fig. 1, the main controller orchestrates the subcontrollers for the ailerons and the rudder to achieve a coordinated turn. The entire system is $H_{main} \parallel H_{left} \parallel H_{rudder} \parallel H_{right} \parallel H_{air}$.

A subcontroller H_M for $M \in \{left, right, rudder\}$ has three modes acc, dec, and con. Its angle δ_M and the moving rate r_M changes according to the ODEs with constant $c_M > 0$:

$$\dot{\delta}_M = r_M,$$
$$\dot{r}_M = c_M \text{ (acc)}, \quad \dot{r}_M = -c_M \text{ (dec)}, \quad \dot{r}_M = 0 \text{ (con)}.$$

H_M performs a jump transition with an input action $g_M(v)$ to determine a next mode based on the goal angle v. The sets of I/O variables and I/O actions are defined by: $X_O^M = \{\delta_M\}$, $\Sigma_I^M = \{g_M(v) \mid v \in \mathbb{R}\}$, and $X_I^M = \Sigma_O^M = \emptyset$.

The lateral dynamics of the aircraft is modeled by H_{air} that has a single mode with the nonlinear ODEs:

$$\dot{\beta} = Y(\beta, \boldsymbol{\delta})/mV - r + V\cos\beta\sin\phi/g, \quad \dot{\psi} = g/V \cdot \tan\phi,$$
$$\dot{p} = (c_1 r + c_2 p)r\tan\phi + c_3 L(\beta, \boldsymbol{\delta}) + c_4 N(\beta, \boldsymbol{\delta}), \qquad \dot{\phi} = p,$$
$$\dot{r} = (c_8 p - c_2 r)r\tan\phi + c_4 L(\beta, \boldsymbol{\delta}) + c_9 N(\beta, \boldsymbol{\delta})$$

with β the yaw angle, ψ the direction, p the rolling moment, ϕ the roll angle, and r the yawing moment. Y, L, and N are linear functions of β and angles $\boldsymbol{\delta} = (\delta_{left}, \delta_{right}, \delta_{rudder})$. The sets of I/O variables and I/O actions are defined by: $X_I^{air} = \{\boldsymbol{\delta}\}$, $\Sigma_O^{air} = \Sigma_I^{main}$, and $X_O^{air} = \Sigma_I^{air} = \emptyset$.

The main controller H_{main} provides a goal angle to a subcontroller M using a discrete transition with an output action g_M, and monitors the current position from H_{air} using a discrete transition with an input action $read$. The component H_{main} has no I/O variables, but has the sets of I/O actions:

$$\Sigma_I^{main} = \{read(v_\psi, v_\phi, v_\beta) \mid v_\psi, v_\phi, v_\beta \in \mathbb{R}\}$$
$$\Sigma_O^{main} = \Sigma_I^{left} \cup \Sigma_I^{right} \cup \Sigma_I^{rudder}$$

We are interested in a reachability problem to find outputs of H_{main} to reach a goal direction in a reasonable time, while keeping the yaw angle β close to 0 during the turn.

The continuous interaction between $(H_{left}, H_{rudder}, H_{right})$ and H_{air} cannot be specified using discrete synchronization without I/O variables. We can eliminate I/O variables by building a single HIOA that is equivalent to the composition of the network, but at the cost of the state explosion problem. For example, a single hybrid automaton for $H_1 \parallel \ldots \parallel H_n$ has total $\prod_{i=1}^{n} m_i$ modes, if each H_i has m_i modes.

III. SMT Encoding of Hybrid Components

In this section we propose a modular method to encode a network of hybrid system components that involve *both discrete and continuous I/O* at the same time. Our technique generalizes previous SMT-based approaches that only focus on discrete communication using synchronous actions.

A. Encoding of Discrete Transitions

We use first-order logic formulas over the real numbers, called $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formulas, along with a collection $\mathcal{F}$ of Type 2 computable real functions [3]. A Type 2 computable real function can be numerically evaluated up to an arbitrary precision, such as polynomials, exponentiation, trigonometric functions, and solutions of Lipschitz-continuous ODEs. The syntax is defined in the standard way, with c real number constants, y first-order variables, and f real functions in $\mathcal{F}$:

$$t \ ::= \ c \mid y \mid f(t_1, \ldots, t_n)$$
$$\varphi \ ::= \ t > 0 \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \exists y.\varphi \mid \forall y.\varphi.$$

Consider a HIOA H. As usual, a discrete transition of H with an action can simply be encoded by using a first-order variable w that denotes the action. Discrete synchronization by the action is encoded by using the same variable w in the formula for each corresponding component.

Definition 2. *An $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-encoding for a (synchronized) discrete transition of H from state $\langle q, \boldsymbol{y}\rangle$ to $\langle q', \boldsymbol{y'}\rangle$ by an action w is:*

$$dt(q, \boldsymbol{y}, w, q', \boldsymbol{y'}) \ \equiv \ (w \in \Sigma \ \rightarrow \ jump^H_{q,q'}(\boldsymbol{y}, w, \boldsymbol{y'})) \wedge (w \notin \Sigma \ \rightarrow \ q = q' \ \wedge \ \boldsymbol{y} = \boldsymbol{y'}).$$

B. Encoding of Continuous Flows

The continuous behavior of one component is parameterized by the trajectories of its input variables, which are composed of the trajectories of the corresponding output variables in other components. Unlike actions, these (potentially nonlinear) trajectories cannot be encoded as first-order variables in $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$, because they are actually real-valued functions.

We use an *uninterpreted real function symbol* $\mathbb{R} \rightarrow \mathbb{R}$ to represent a trajectory for each input variable. Consider a flow condition for mode q, expressed as a system of ODEs:

$$\left[\tfrac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = \boldsymbol{f_q}(\boldsymbol{x}, \boldsymbol{o}, \boldsymbol{\iota})(t), \quad \tfrac{\mathrm{d}\boldsymbol{o}}{\mathrm{d}t} = \boldsymbol{g_q}(\boldsymbol{x}, \boldsymbol{o}, \boldsymbol{\iota})(t)\right]$$

with $\boldsymbol{x}$ the internal variables, $\boldsymbol{o}$ the output variables, and $\boldsymbol{\iota}$ the input variables. Mathematically, these ODE variables in the terms $\boldsymbol{f_q}(\boldsymbol{x}, \boldsymbol{o}, \boldsymbol{\iota})(t)$ and $\boldsymbol{g_q}(\boldsymbol{x}, \boldsymbol{o}, \boldsymbol{\iota})(t)$ denote unary real functions $\mathbb{R} \rightarrow \mathbb{R}$ over time t.

Given function symbols $\dot{o}_1, \ldots, \dot{o}_l$ to denote the derivatives of $\boldsymbol{o} = (o_1, \ldots, o_l)$, and function symbols $\dot{x}_1, \ldots, \dot{x}_n$ to denote the derivatives of $\boldsymbol{x} = (x_1, \ldots, x_n)$, we can express the trajectories of $\boldsymbol{x}$ and $\boldsymbol{o}$ as the $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formula

$$\forall u \in [0, t]. \ [\dot{\boldsymbol{x}}, \dot{\boldsymbol{o}}](u) = [\boldsymbol{f_q}(\boldsymbol{x}, \boldsymbol{o}, \boldsymbol{\iota})(u), \ \boldsymbol{g_q}(\boldsymbol{x}, \boldsymbol{o}, \boldsymbol{\iota})(u)].$$

The trajectories of the input variables $\boldsymbol{\iota}$ (that appear as free variables in $\boldsymbol{f_q}$ and $\boldsymbol{g_q}$) are then defined by using *output derivatives of other components*.

Using these I/O derivative symbols, we can express the continuous change of H's states from initial values $\boldsymbol{y^0}$ to new values $\boldsymbol{y^t}$ for duration t as the $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formula:

$$\boldsymbol{y^t} = \boldsymbol{y^0} + \textstyle\int_0^t [\dot{\boldsymbol{x}}, \dot{\boldsymbol{o}}, \boldsymbol{\dot{\iota}}] \, \mathrm{d}s$$

This $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formula is parameterized by the internal derivatives $\dot{\boldsymbol{x}} = (\dot{x}_1, \ldots, \dot{x}_n)$, the output derivatives $\dot{\boldsymbol{o}} = (\dot{o}_1, \ldots, \dot{o}_l)$, and the input derivatives $\boldsymbol{\dot{\iota}} = (\dot{\iota}_1, \ldots, \dot{\iota}_m)$.

The ODE solution term $\boldsymbol{y^0} + \int_0^t [\dot{\boldsymbol{x}}, \dot{\boldsymbol{o}}, \boldsymbol{\dot{\iota}}] \, \mathrm{d}s$ can be considered as a computable real function $F(\boldsymbol{y^0}, t)$ in the collection $\mathcal{F}$, provided that $\boldsymbol{f_q}$, $\boldsymbol{g_q}$, and the ODEs for the input variables are all Lipschitz-continuous [14], [15]. The extra derivative function symbols are also in the collection $\mathcal{F}$. In sum, an encoding of continuous flows is defined as follows:

Definition 3. *An $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-encoding for a continuous flow of H from $\boldsymbol{y^0}$ to $\boldsymbol{y^t}$ in mode q for duration t, that involves I/O variables $(\boldsymbol{\iota}, \boldsymbol{o})$ and satisfies the invariant condition inv_q^H, where $F(\boldsymbol{y^0}, t)$ denotes $\boldsymbol{y^0} + \int_0^t [\dot{\boldsymbol{x}}, \dot{\boldsymbol{o}}, \boldsymbol{\dot{\iota}}] \, \mathrm{d}s$, is:*

$$ct(q, \boldsymbol{y^0}, \boldsymbol{y^t}, t \mid \boldsymbol{\iota}, \boldsymbol{o}, \boldsymbol{x}) \ \equiv \ \boldsymbol{y^t} = F(\boldsymbol{y^0}, t) \wedge \forall u \in [0, t]. \ [\dot{\boldsymbol{x}}, \dot{\boldsymbol{o}}](u) = [\boldsymbol{f_q}, \boldsymbol{g_q}](u) \wedge \forall u \in [0, t]. \ inv_q^H(F(\boldsymbol{y^0}, u)).$$

We use different uninterpreted functions for variables in different components, even if they are "shared" variables in a network of HIOA. The use of the same uninterpreted functions for different components may cause unintended semantic effects, as explained in Sec. IV-A below. This is not a strict restriction, because we can always use "syntactic copies" to enforce that input variables $\boldsymbol{\iota}$ and output variables $\boldsymbol{o}$ follow the isomorphic system of ODEs $\boldsymbol{f}$ up to renaming, by means of *connection formulas* between $\boldsymbol{\iota}$ and $\boldsymbol{o}$ as follows.

Definition 4. *An $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-encoding of connections between input variables $\boldsymbol{\iota}$ and output variables $\boldsymbol{o}$ is as follows (it includes internal variables $\boldsymbol{x}$ and $\boldsymbol{x'}$, since $\boldsymbol{o}$ may depend on $\boldsymbol{x}$):*

$$conn(\boldsymbol{\iota}, \boldsymbol{o}, \boldsymbol{x}, \boldsymbol{x'}) \ \equiv \ (\forall u \in [0, t]. \ [\dot{\boldsymbol{o}}, \dot{\boldsymbol{x}}](u) = \boldsymbol{f}(\dot{\boldsymbol{o}}, \boldsymbol{x})(u)) \leftrightarrow \ (\forall u \in [0, t]. \ [\boldsymbol{\dot{\iota}}, \dot{\boldsymbol{x'}}](u) = \boldsymbol{f}(\boldsymbol{\dot{\iota}}, \boldsymbol{x'})(u)).$$

C. Encoding of Bounded Reachability

For a network of HIOA $H_1 \parallel \cdots \parallel H_N$, the reachability up to the k-th discrete step—that involves continuous I/O as well as discrete synchronization—can be encoded in $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ as follows. This formula is defined as just a conjunction of N subformulas, each of which encodes the reachability of each individual component H_j up to the k-th discrete step.

Definition 5. *An $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-encoding for the k-step reachability of a network of HIOA $H_1 \parallel \cdots \parallel H_N$ is the $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formula of size $O(\sum_{j=1}^N k \cdot |Q_j|^2)$ ($\exists$-quantified at the top):*

$$\begin{aligned}
&\bigwedge_{j=1}^{N} \left[\begin{array}{l} init^j_{m_0^j}(\boldsymbol{y_{j_0}^0}) \ \wedge \ ct(m_0^j, \boldsymbol{y_{j_0}^0}, \boldsymbol{y_{j_0}^t}, t_0 \mid \boldsymbol{\iota_0^j}, \boldsymbol{o_0^j}, \boldsymbol{x_0^j}) \\ \wedge \ \bigwedge_{i=1}^{k} \left[\begin{array}{l} dt(m_{i-1}^j, \boldsymbol{y_{j_{i-1}}^t}, w_i, m_i^j, \boldsymbol{y_{j_i}^0}) \wedge \\ ct(m_i^j, \boldsymbol{y_{j_i}^0}, \boldsymbol{y_{j_i}^t}, t_i \mid \boldsymbol{\iota_i^j}, \boldsymbol{o_i^j}, \boldsymbol{x_i^j}) \end{array} \right] \end{array} \right] \\
&\wedge \ \ goal(m_k^1, \ldots, m_k^N, \boldsymbol{y_{1_k}^t}, \ldots, \boldsymbol{y_{N_k}^t}) \\
&\wedge \ \ \textstyle\bigwedge_{i=0}^{k} conn(\boldsymbol{\iota_i^1}, \ldots, \boldsymbol{\iota_i^N}, \boldsymbol{o_i^1}, \ldots, \boldsymbol{o_i^N}, \boldsymbol{x_i^1}, \ldots, \boldsymbol{x_i^N})
\end{aligned}$$

For each i-th discrete step of duration t_i, component H_j is in mode m_i^j, and the values of H_j's variables begin with $\boldsymbol{y_{j_i}^0}$ and end with $\boldsymbol{y_{j_i}^t}$. At the first step ($i = 0$), the initial values $\boldsymbol{y_{j_0}^0}$ of H_j's variables satisfy the initial condition. To begin the $(i+1)$-th step, every component synchronizes its transition with the same action w_i. For the k-th step, the goal formula holds for H_j's final state. The connection formulas $conn$ link input and output variables.

D. Example

Consider the airplane controller example in Sec. II-B. For a subcontroller M, an $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-encoding of a continuous flow from initial values (δ_M^0, r_M^0) to new values (δ_M^t, r_M^t) with the invariant condition $-45 < \delta_M < 45$ is the formula below.

$$\begin{aligned}
&(\delta_M^t, r_M^t) = (\delta_M^0, r_M^0) + \textstyle\int_0^t [\dot\delta_M, \dot r_M] \,\mathrm{d}s \\
\wedge \ &\left[\begin{array}{l} (q_M = acc \ \rightarrow \ \forall u \in [0,t].\ [\dot\delta_M, \dot r_M](u) = [r_M, c_M]) \wedge \\ (q_M = dec \ \rightarrow \ \forall u \in [0,t].\ [\dot\delta_M, \dot r_M](u) = [r_M, -c_M]) \wedge \\ (q_M = con \ \rightarrow \ \forall u \in [0,t].\ [\dot\delta_M, \dot r_M](u) = [r_M, 0]) \end{array} \right] \\
&\wedge \forall u \in [0,t].\ -45 < \pi_1\big((\delta_M^0, r_M^0) + \textstyle\int_0^u [\dot\delta_M, \dot r_M] \,\mathrm{d}s\big) < 45.
\end{aligned}$$

The last line expresses the invariant condition $-45 < \delta_M < 45$ using the solution term $(\delta_M^0, r_M^0) + \int_0^u [\dot\delta_M, \dot r_M] \,\mathrm{d}s$ and the projection function $\pi_1(a,b) = a$.

For the airplane component H_{air}, an $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-encoding of its continuous flow is simply the single ODE solution term:

$$(\beta^t, \psi^t, \phi^t, p^t, r^t, \boldsymbol{\delta}^t) = (\beta^0, \psi^0, \phi^0, p^0, r^0, \boldsymbol{\delta}^0) + \textstyle\int_0^t \boldsymbol{F}(s) \,\mathrm{d}s,$$

that is parameterized by the component H_{air}'s input variables $\boldsymbol{\delta^{air}} = (\delta^{air}_{left}, \delta^{air}_{right}, \delta^{air}_{rudder})$, where

$$\boldsymbol{F}(s) = \left[\begin{array}{l} Y(\beta, \boldsymbol{\delta^{air}})/mV - r + V/g \cdot \cos\beta \sin\phi \\ g/V \cdot \tan\phi \\ p \\ (c_1 r + c_2 p) r \tan\phi + c_3 L(\beta, \boldsymbol{\delta^{air}}) + c_4 N(\beta, \boldsymbol{\delta^{air}}) \\ (c_8 p - c_2 r) r \tan\phi + c_4 L(\beta, \boldsymbol{\delta^{air}}) + c_9 N(\beta, \boldsymbol{\delta^{air}}) \\ \dot{\boldsymbol{\delta}}^{\boldsymbol{air}} \end{array} \right]$$

The behavior of the input variables $\boldsymbol{\delta^{air}}$ is given by connection formulas such as: $(\forall u \in [0,t].\ [\dot\delta_M, \dot r_M](u) = [r_M, c_M]) \ \leftrightarrow \ (\forall u \in [0,t].\ [\dot\delta^{air}_M, \dot r^{air}_M](u) = [r^{air}_M, c_M])$, where variable r^{air}_M is a "copy" of the internal variable r_M of M.

E. The Correctness of the Encoding

Our modular encoding is correct in the sense that it is equisatisfiable to the encoding of the composition, which is a single hybrid automaton (see [16] for the proof).

Theorem 1. *Given a network of HIOA $\mathcal{H} = H_1 \parallel \cdots \parallel H_n$, its modular encoding (of size $O(k \sum_{j=1}^N |Q_j|^2)$) and the encoding of its composition (of size $O(k \prod_{j=1}^N |Q_j|)$) are equisatisfiable.*

IV. SMT Algorithm for Hybrid Components

We syntactically identify a generic class of $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formulas that involve universal quantification for uninterpreted real functions. This class includes $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formulas for networks of hybrid I/O automata in Def. 5. We present an SMT procedure for checking the satisfiability of these $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-formulas. The proofs of lemmas and theorems in this section are in [16].

A. Syntactic Classification

We explicitly decompose a collection $\mathcal{F}$ as $\mathcal{F}' \cup \mathcal{G}$. A collection $\mathcal{G}$ is composed of uninterpreted unary differentiable functions $\mathbb{R} \to \mathbb{R}$ to denote variables $x_1, \ldots, x_l$ in ODEs and their derivatives $\dot x_1, \ldots, \dot x_l$ (including I/O variables and I/O derivatives), whereas $\mathcal{F}'$ includes only interpreted real functions. For each pair $(x, \dot x)$ in $\mathcal{G}$, the function $\dot x \in \mathcal{G}$ is a derivative of the function $x \in \mathcal{G}$.

We explicitly take into account *parameterized integration operators* over ODE variables $\boldsymbol{x} \in \mathcal{G}$ and derivatives $\boldsymbol{\dot\imath} \in \mathcal{G}$:

$$\boldsymbol{y^0} + \int_0^t [\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\iota})(s), \boldsymbol{\dot\imath}(s)] \,\mathrm{d}s,$$

specifying solution functions of parameterized ODE systems of the form $\frac{\mathrm{d}}{\mathrm{d}s}[\boldsymbol{x}, \boldsymbol{\iota}] = [\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\iota})(s), \boldsymbol{\dot\imath}(s)]$ with initial values $\boldsymbol{y^0}$ over time t. As mentioned, we can define the behavior of each input derivative ι by universally quantified $\mathcal{L}_{\mathbb{R}_{\mathcal{F}' \cup \mathcal{G}}}$-formulas of the form $\forall u \in [0,t].\ i(u) = g(\boldsymbol{z})(u)$, where $\boldsymbol{z} \in \mathcal{G}$ denote unary function symbols for ODE variables.

We identify a subclass of $\mathcal{L}_{\mathbb{R}_{\mathcal{F}' \cup \mathcal{G}}}$-formulas that allows a reduction to $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$ without extra uninterpreted real functions in $\mathcal{G}$. We can apply any (existing) algorithms for $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-formulas after the reduction. As a matter of fact, the entire class of $\mathcal{L}_{\mathbb{R}_{\mathcal{F}' \cup \mathcal{G}}}$-formulas is too expressive to have any kinds of efficient decision procedures. For example, control problems of nonlinear systems are undecidable in general [17], but can be written as $\mathcal{L}_{\mathbb{R}_{\mathcal{F}' \cup \mathcal{G}}}$-formulas.[1]

We require that each uninterpreted function symbol in $\mathcal{G}$ only occurs in a single integration term. Otherwise, a "shared" function symbol in different terms leads to an unintended semantic restriction that all the integration terms have the same solution function. The decomposition using parameterized integration in Sec. III-B is therefore no longer equisatisfiable to the original formula. This is why we encode variables in different components as different functions.

[1] Consider a nonlinear system $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$ with initial values $\boldsymbol{x_0}$ and controls $\boldsymbol{u}$. The controllability to $\boldsymbol{x}_t$ can be expressed as the satisfiability of $\boldsymbol{x}_t = \boldsymbol{x_0} + \int_0^t \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) \,\mathrm{d}s$ with uninterpreted real functions $\boldsymbol{u}$.

For example, consider the $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-formula that involves the ODE $\frac{\mathrm{d}x}{\mathrm{d}s} = x$ with different initial values 1 and 2:

$$x_1 = 1 + \int_0^t x(s)\,\mathrm{d}s \;\wedge\; x_2 = 2 + \int_0^t x(s)\,\mathrm{d}s \;\wedge\; t = 1,$$

which is satisfiable with $x_1 = e$ and $x_2 = 2e$. Now let us replace the term $x(s)$ with an uninterpreted function $\dot{x}(s)$:

$$x_1 = 1 + \int_0^t \dot{x}(s)\,\mathrm{d}s \;\wedge\; x_2 = 2 + \int_0^t \dot{x}(s)\,\mathrm{d}s \;\wedge\; t = 1 \;\wedge\; \forall u \in [0,t].\, \dot{x}(u) = x(u).$$

This formula is *not* satisfiable, because there are no single interpretations of $\dot{x}$ and x that denote solution functions of $\frac{\mathrm{d}x}{\mathrm{d}s} = x$ for both initial values 1 and 2 at the same time.

We restrict our attention to $\mathcal{G}$ with the following constraints. Every uninterpreted function $x \in \mathcal{G}$ that occurs in the same integration term has the same domain $[0, t_x]$. Each derivative $\dot{x} \in \mathcal{G}$ is bounded by a finite set of Lipschitz-continuous functions $G_x = \{g_1(\boldsymbol{z}), g_2(\boldsymbol{z}), \ldots, g_m(\boldsymbol{z})\}$. This *boundedness constraint* for $\dot{x} \in \mathcal{G}$ can be expressed as the $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-formula: $bound_{G_x}(\dot{x}) \equiv \bigvee_{g(\boldsymbol{z})\in G_x} \forall u \in [0,t_x].\, \dot{x}(u) = g(\boldsymbol{z})(u)$.

Definition 6. *An $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-formula for networks of hybrid I/O automata has the form: $\exists \boldsymbol{y}.\, \varphi \wedge \bigwedge_{\dot{x}\in\mathcal{G}} bound_{G_x}(\dot{x})$, where:*

- *each subformula of φ is quantifier-free or containing only universally quantified subformulas over time;*
- *each uninterpreted function in $\mathcal{G}$ appears in integration terms, universally quantified formulas over time of the form $\forall u \in [0,t_x].\, \dot{x}(u) = g(\boldsymbol{z})(u)$, or formulas of the negated form $\exists u \in [0,t_x].\, \dot{x}(u) \neq g(\boldsymbol{z})(u)$;*
- *each uninterpreted function in $\mathcal{G}$ appears in at most one integration term (but the same integration term can appear in a formula many times).*

The formula in Def. 5 is in the syntactic class of Def. 6, if every input variable corresponds to some output variable.

B. $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-Reduction of $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-Formulas

A key part of our algorithm is *the $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-reduction* that removes uninterpreted functions in $\mathcal{G}$ from $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-formulas, summarized in Alg. 1. For a conjunction μ of $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-literals that *satisfy the boundedness constraint and the syntactic restriction in Def. 6*, the $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-reduction procedure builds an $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-formula that is equisatisfiable to the conjunction μ.

Definition 7. *An $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-*literal *is an atomic $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-formula, a universally quantified formula $\forall u \in [0,t_x].\, \dot{x}(u) = g(\boldsymbol{z})(u)$, or a negation of these $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-atoms.*

Without loss of generality, we assume that for each derivative $\dot{x} \in \mathcal{G}$, the conjunction μ includes a universally quantified $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-literal of the form $\forall u \in [0,t_x].\, \dot{x}(u) = g(\boldsymbol{z})(u)$ from the boundedness constraint $bound_{G_x}(\dot{x})$.

The $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-reduction begins with choosing a universally quantified $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-literal $\forall u \in [0,t_x].\, \dot{x}(u) = g(\boldsymbol{z})(u)$ for each derivative $\dot{x} \in \mathcal{G}$ (line 2). We replace every occurrence of derivative $\dot{x}$ in parameterized integration terms by the term $g(\boldsymbol{z})(u)$ (line 3). This procedure preserves the satisfiability of the formula as stated in Lemma 1, since each uninterpreted function in $\mathcal{G}$ appears in at most one integration term.

Algorithm 1: $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-reduction for $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-formulas.

Input: A conjunction $\mu = l_1 \wedge \cdots \wedge l_m$ of $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-literals
Output: An equisatisfiable $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-formula

1 **for** *each $\dot{x} \in \mathcal{G}$ inside μ's integration terms* **do**
2 pick a literal $l = \forall u \in [0,t_x].\, \dot{x}(u) = g(\boldsymbol{z})(u)$ in μ;
3 replace each $\dot{x}$ by $g(\boldsymbol{z})$ in μ's integration terms;
4 **for** *each literal l' in μ including $\dot{x}$ other than l* **do**
5 **if** $l' = \forall u \in [0,t_x].\, \dot{x}(u) = g'(\boldsymbol{z})(u)$ **then**
6 combine $\forall u \in [0,t_x].\, g(\boldsymbol{z})(u) = g'(\boldsymbol{z})(u)$ with μ;
7 **else**
8 combine $\exists u \in [0,t_x].\, g(\boldsymbol{z})(u) \neq g'(\boldsymbol{z})(u)$ with μ;
9 **return** μ;

Lemma 1. *Given a conjunction μ of $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-literals including $\forall u \in [0,t_x].\, \dot{x}(u) = g(\boldsymbol{z})(u)$, the formula μ' obtained from μ by replacing each occurrence of $\dot{x}$ in integration terms by $g(\boldsymbol{z})$ is equisatisfiable to the conjunction μ.*

As a result, every parameterized integration term in the conjunction μ becomes *concrete solution $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-term* without free ODE variables. That is, each integration term is now considered as a computable real function in $\mathcal{F}'$.

We add extra constraints to ensure the consistency between the chosen $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-literals and the other $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-literals in μ (line 4). For each pair of $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-literals in μ, we must ensure that they are satisfiable at the same time. For example, we need the constraint $\forall u \in [0,t_x].\, g(\boldsymbol{z})(u) = g'(\boldsymbol{z})(u)$ for a universally quantified literal $\forall u \in [0,t_x].\, \dot{x}(u) = g'(\boldsymbol{z})(u)$ in μ. These new constraints are often *not* $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-formulas, since they can still include ODE variables (e.g., $\boldsymbol{z}$).

We express these constraints in $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$ by adding extra ODEs to corresponding integration terms. For a constraint $\forall u \in [0,t_x].\, g(\boldsymbol{z})(u) = g'(\boldsymbol{z})(u)$, we define a new ODE $\frac{\mathrm{d}w}{\mathrm{d}s} = g'(\boldsymbol{z})(s) - g(\boldsymbol{z})(s)$ with a fresh variable w. If the constraint is true, the value of the variable w is always 0; that is, the invariant condition $\forall u \in [0,t_x].\, w(u) = 0$ must hold. We formally define this process as follows.

Definition 8. *For constraints $\forall u \in [0,t_x].\, g(\boldsymbol{z})(u) = g'(\boldsymbol{z})(u)$ or $\exists u \in [0,t_x].\, g(\boldsymbol{z})(s) \neq g'(\boldsymbol{z})(s)$, the $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-reduction of integration $F(\boldsymbol{y^0}, t) \equiv \boldsymbol{y^0} + \int_0^t \boldsymbol{f}(\boldsymbol{z})(s)\,\mathrm{d}s$ is the term:*

$$\hat{F}(\boldsymbol{y^0}, t) \equiv [0, \boldsymbol{y^0}] + \int_0^t [g'(\boldsymbol{z})(s) - g(\boldsymbol{z})(s),\ \boldsymbol{f}(\boldsymbol{z})(s)]\,\mathrm{d}s,$$

and the $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-reductions of the constraints are respectively:

$$\forall u \in [0,t_x].\, \pi_1(\hat{F}(\boldsymbol{y^0}, u)) = 0,\ \exists u \in [0,t_x].\, \pi_1(\hat{F}(\boldsymbol{y^0}, u)) \neq 0.$$

This process also preserves the satisfiability of the formula as stated in Lemma 2, because every corresponding integration term of a constraint is identical by assumption.

Lemma 2. *A conjunction μ with an extra constraint is equisatisfiable to the formula μ' obtained from μ by replacing corresponding integration terms by their $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-reductions and replacing the constraint by its $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-reduction.*

The time complexity of the entire $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-reduction process is $O(n^2)$, where n denotes the number of $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-literals in the conjunction μ.[2] The correctness of our $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-reduction procedure follows from the fact that each $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-reduction step in Alg. 1 preserves the satisfiability.

Theorem 2. *Given a conjunction μ that meet the boundedness constraint and the syntactic restriction in Def. 6, Algorithm 1 generates an $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-formula that is equisatisfiable to μ.*

C. Checking Satisfiability Using $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-Reduction

We consider the satisfiability of $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-formulas of Def. 6. We apply the $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-reduction process to have equisatisfiable $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-formulas without extra function symbols in $\mathcal{G}$, and then employ an existing algorithm to check $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-satisfiability as a subroutine. Our algorithm terminates if the $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-satisfiability subroutine is terminated. This provides a theory solver for $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$, by means of $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-reduction and an $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-solver.

Algorithm 2: SMT procedure for $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-formulas.

Input: An $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-formula $\exists \boldsymbol{y}.\, \varphi \wedge \bigwedge_{\dot{x}\in\mathcal{G}} bound_{G_x}(\dot{x})$
Output: Unsat, or Sat with a satisfiable assignment

1 **while** $\exists$ *a propositionally satisfiable set L of literals* **do**
2 $\quad \phi \leftarrow \mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-reduction($\bigwedge_{l\in L} l$);
3 $\quad$ **if** ϕ *is* Sat *by $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-satisfiability solving* **then**
4 $\quad\quad$ **return** Sat *and a satisfiable assignment of* $\boldsymbol{y}$;
5 $\quad$ learn the conflicts between $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-literals;
6 **return** Unsat;

Algorithm 2 summarizes our algorithm. We employ the DPLL($\mathcal{T}$) framework to obtain a propositionally satisfiable set of literals by using Boolean satisfiability solving (line 1). The formula is unsatisfiable if no propositionally satisfiable set exists (line 6). Otherwise, each set imposes a conjunction of $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-literals. Since we only consider formulas in the class of Def. 6, the conjunction satisfies the boundedness constraint.

We apply the $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-reduction procedure in Alg. 1 to obtain an *equisatisfiable $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-formula* ϕ (line 2). We use an SMT algorithm for $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-formula as a subroutine (line 3). If the resulting formula ϕ is satisfiable, then the original formula is also satisfiable (line 4).[3] If ϕ is not satisfiable, conflict clauses can be used by SAT solving for the next iteration based on conflict-driven clause learning (line 5).

D. δ-Complete SMT for $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-Formulas

Our algorithm is best suited for nonlinear hybrid systems where continuous I/O cannot be encoded as discrete actions. For example, δ-complete SMT can be applied to check the satisfiability of $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-formulas up to a given precision $\delta > 0$, called *δ-satisfiability* [3]. The satisfiability of $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-formulas is in general undecidable for nonlinear real functions, but the δ-satisfiability of $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-formulas is decidable.

[2]There can be many identical integration terms for each step, but the replacements can be performed in constant time using subformula sharing.

[3]Since the $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-reduction does not alter first-order variables, satisfiable assignments for μ and ϕ have the same values for the first-order variables.

Finding conflict $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-literals is very important for the performance of DPLL($\mathcal{T}$) and conflict-driven clause learning, but this process is nontrivial for δ-complete SMT. The reason is that δ-consistency actually depends on the value of δ. For example, $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-literals $\forall u \in [0, 0.5].\, \dot{x}(u) = 1 + \frac{1}{2}u$ and $\forall u \in [0, 0.5].\, \dot{x}(u) = \sqrt{u+1}$ are inconsistent up to precision $\delta = 0.01$, but consistent up to different precision $\delta = 0.1$.

To facilitate the process of finding conflict $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-literals, we use *uniqueness lemmas* for incompatible $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-literals. Consider two $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-literals $\forall u \in [0, t_x].\, \dot{x}(u) = g(\boldsymbol{z})(u)$ and $\forall u \in [0, t_x].\, \dot{x}(u) = g'(\boldsymbol{z})(u)$. If we know in advance that two functions g and g' are not equal, we add the formula $\neg[\forall u \in [0, t_x].\, \dot{x}(u) = g(\boldsymbol{z})(u) \wedge \forall u \in [0, t_x].\, \dot{x}(u) = g'(\boldsymbol{z})(u)]$ and the consistency checking can be performed at line 1.

For the reachability of networks of hybrid I/O automata, we apply a heuristic specialized for the encoding in Def. 5. Since each mode q can correspond to only one flow condition, if a formula $\forall u \in [0, t].\ [\dot{\boldsymbol{x}}, \dot{\boldsymbol{o}}](u) = [\boldsymbol{f_q}, \boldsymbol{g_q}](u)$ for one continuous output is true, the truths of other continuous output formulas are not relevant. In Alg. 1, we choose one universally quantified $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-literal for each mode, and disregard extra consistency checking in line 4 in this case.

V. Case Studies

This section shows a number of examples of networks of hybrid system components, besides the airplane example. They include discrete and continuous interactions between components, and involve nontrivial nonlinear ODEs.

A. Driving Simple Cars.

A number of cars are running in sequence, while each car follows the behavior of the car in front (the first car moves according to its own scenario). The position (x_i, y_i) and the direction θ_i of each car i of length L_i depends on its speed v_i and steering angle ϕ_i, given by the nonlinear ODEs [18]:

$$\dot{x_i} = v_i \cos\theta_i, \qquad \dot{\theta_i} = v_i / L_i \cdot \tan\phi_i,$$
$$\dot{y_i} = v_i \sin\theta_i, \qquad \dot{\phi_i} = -k_i(\phi_i - \phi_{i-1}),$$

To keep a safe distance, each car has three modes acc, dec, and $keep$ for acceleration, deceleration, and following the speed of the front car, respectively: $\dot{v_i} = -K_i(v_i - v_{i-1})$ if $q_i = keep$, $\dot{v_i} = C$ if $q_i = acc$, and $\dot{v_i} = -C$ if $q_i = dec$. The goal is to find a mode change schedule for driving cars safely.

B. Network of Thermostat Controllers.

A number of rooms are interconnected by open doors (Fig. 2). The temperature x_i of each room i is separately controlled by each thermostat, depending on both the heater's mode $q_i \in \{m_{\texttt{on}}, m_{\texttt{off}}\}$ and the temperatures of the adjacent rooms. The value of x_i changes according to the ODEs:

$$\frac{\mathrm{d}x_i}{\mathrm{d}t} = \begin{cases} K_i\big(h_i - (c_i x_i - d_i \sum_{j\in A_i} x_j)\big) & \text{if } q_i = m_{\texttt{on}} \\ -K_i\big(c_i x_i - d_i \sum_{j\in A_i} x_j\big) & \text{if } q_i = m_{\texttt{off}}, \end{cases}$$

where A_i is the set of the adjacent rooms, and K_i, h_i, c_i, d_i depend on the size of the room, the heater's power, and the size of the open doors. The goal is to keep each temperature in a desired range, while the outside temperatures change.

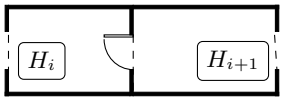

Fig. 2. Connected rooms

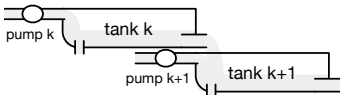

Fig. 3. Connected water tanks

C. Network of Water Tanks.

A number of water tanks are connected by pipes (Fig. 3), adapted from [9]. The water level x_i of each tank i is separately controlled by its pump, depending on the pump's mode $m_i \in \{m_{\mathtt{on}}, m_{\mathtt{off}}\}$ and the levels of the adjacent tanks. The value of x_i changes according to the nonlinear ODEs:

$$A_i\dot{x}_i = (q_i + a\sqrt{2g}\sqrt{x_{i-1}}) - a\sqrt{2g}\sqrt{x_i} \quad \text{if } m_i = m_{\mathtt{on}}$$
$$A_i\dot{x}_i = a\sqrt{2g}\sqrt{x_{i-1}} - a\sqrt{2g}\sqrt{x_i} \quad \text{if } m_i = m_{\mathtt{off}}$$

($x_0 = 0$ for the leftmost tank 1), where A_i, q_i, a depend on the size of the tank, the power of the pump, and the width of the pipe, and g is the standard gravity constant. The goal is to keep each water level in a desired range.

D. Multiple battery Usage.

Given a number of fully charged batteries, a controller switches load between them to achieve longer lifetime out of the batteries, adapted from [19]. Each battery i has three modes $switchedOn$, $switchedOff$, and $dead$. The battery charge dynamics is expressed by the ODEs:

$$\text{(on)}\ \begin{array}{l}\dot{d}_i = L/c - kd_i\\ \dot{g}_i = -L,\end{array} \quad \text{(off)}\ \begin{array}{l}\dot{d}_i = -kd_i\\ \dot{g}_i = 0,\end{array} \quad \text{(dead)}\ \begin{array}{l}\dot{d}_i = 0\\ \dot{g}_i = 0,\end{array}$$

with d_i its kinetic energy difference, g_i its total charge, L its load, and $c \in [0, 1]$ its threshold. If $g_i > (1-c)d_i$, battery i is dead. Otherwise, it can be either on or off. When k batteries are on, load to each battery is divided by k. A goal is to find a switching schedule to achieve a desired lifetime.

VI. Experimental Results

We have implemented our algorithm in version 2 of the dReal solver [12]. It can decide the δ-satisfiability of a wide range of $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-formulas (containing universal quantification over time).[4] We base our implementation on dReal, since the tool supports nontrivial *nonlinear ODEs* in our case studies.[5] However, in principle our techniques can be combined with any other SMT-based approaches for hybrid systems.

We have compared the performance of our algorithm for the $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-encoding with one by a non-modular $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-encoding. Because our examples involve continuous I/O that cannot be encoded by discrete actions, no modular $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-encoding is possible. Therefore, we use an $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-encoding of a single hybrid automaton that is equivalent to the composition, which has been studied in many approaches [1]–[5], [14], [20].

[4] dReal uses interval constraint propagation (ICP) to numerically evaluate ODE integration functions up to a precision $\delta > 0$ [14].

[5] For example, many analysis tools for nonlinear hybrid systems only support polynomials or linear ODEs, not nonlinear ODEs.

The experimental results are summarized in Fig. 4. The case studies and the experimental results are available in [16]. We have performed reachability analysis for the five case studies up to bound $k = 5$. We consider two variants for each example, with double or triple components (for the airplane example, nonlinear ODEs or linear ODEs). We consider both sat (reachable) and $unsat$ (unreachable) cases using different goals. All experiments were conducted on Intel Xeon 2.6 GHz with 512 GB memory. We set a timeout of 12 hours.

The results show that our approach with the new modular $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-encoding *significantly outperforms* the old approach with the non-modular encoding. For example, the analysis with the non-modular $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-encoding for three interconnected thermostats did not terminate within 12 hours for bound $k = 2$, whereas the same analysis by our algorithm with the modular $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'\cup\mathcal{G}}}$-encoding gave the result less than 15 seconds.

According to the results, the performance improvement tends to be more apparent for bigger models and for the $unsat$ cases. For the sat cases, it is possible that a satisfiable assignment can be found in an early stage of DPLL($\mathcal{T}$), e.g., the sat case of two batteries where the difference is small. In a bigger model the size of the generated formula becomes larger, and thus the benefit of using the modular encoding is clearly more explicit (e.g., the cases of three batteries).

This performance improvement is due to the fact that the modular encoding allows a much compact size of the formulas, and a more efficient ODE solving by decomposition of complex systems of ODEs. Also, conflict-driven clause learning can be fully exploited for continuous connections in this way, because conflicts caused by continuous I/O can be detected and then learned in our algorithm.

VII. Related Work

One of the early studies on SMT-based analysis of nonlinear hybrid systems is [21], which proposes constraint solving algorithms for nonlinear reachability problems. There are several approaches that explicitly formulate analysis problems of nonlinear hybrid systems as SMT formulas over the real numbers, such as MathSAT/HyCOMP [5], [6], hydlogic [4], iSAT/HySAT [1], [2], and dReal/dReach [12], [20].

But a modular SMT encoding of networks of nonlinear hybrid systems has not been much investigated, which is what we aim to improve in this paper. All of these techniques assume that interactions between components can be specified through discrete synchronization (e.g., using joint actions). Hence, continuous interactions between components, which occur frequently in many networks of hybrid systems as shown in Sec. V, cannot be properly dealt with in a modular way.

In iSAT-ODE [1], the syntax allows ODE fragments to occur positively in formulas, and thus the encoding of Def. 6 can be expressed in principle. But a modular encoding has not been studied for iSAT-ODE, and the tool does not support it either. Our work formally identifies a generic class of formulas that *strictly* includes one supported by iSAT-ODE. Our algorithm is based on equisatisfiable $\mathcal{L}_{\mathbb{R}_{\mathcal{F}'}}$-reduction, whereas iSAT-ODE uses a specialized algorithm that extends DPLL.

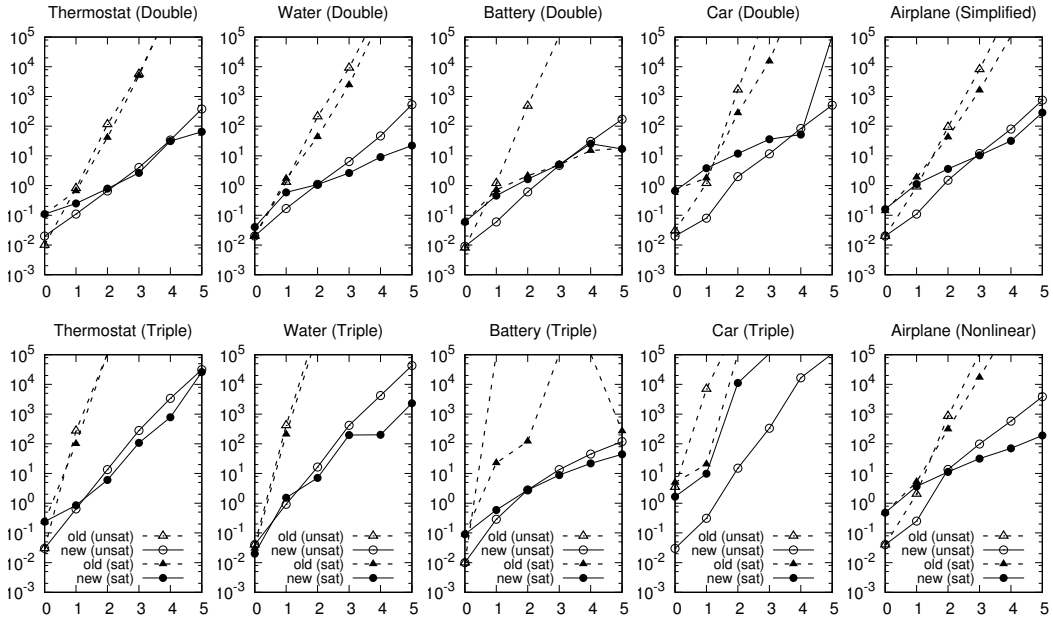

Fig. 4. Running time (in seconds) of k-step reachability analysis (solid lines for the new approach and dashed lines for the non-modular approach)

In addition to SMT-based approaches, there are many other approaches to analyze nonlinear hybrid systems by calculating a set of reachable states, e.g., [22], [23]. Our technique and such reachable-set computation techniques focus on different aspects of hybrid system analysis. Reachable-set computation can be used as an ODE solver in Alg. 2, while our technique addresses higher-level composition and modular analysis.

VIII. Conclusions

We have presented new SMT-based techniques for analyzing networks of nonlinear hybrid systems. We have shown that continuous interactions between hybrid components, which cannot be captured by discrete communication in general, can be decomposed and modularly encoded as SMT formulas. Since existing SMT algorithms cannot deal with the modular encoding, we have presented a new SMT solving algorithm, which can greatly increase the performance of SMT-based analysis for networks of nonlinear hybrid systems.

Future work will include investigating approximation and decomposition methods to achieve further scalability when analyzing networks of nonlinear hybrid systems.

References

[1] A. Eggers, N. Ramdani, N. S. Nedialkov, and M. Fränzle, "Improving the SAT modulo ODE approach to hybrid systems analysis by combining different enclosure methods," *Softw. Syst. Model.*, vol. 14, no. 1, 2015.

[2] M. Fränzle and C. Herde, "HySAT: An efficient proof engine for bounded model checking of hybrid systems," *Form. Methods Syst. Des.*, vol. 30, no. 3, pp. 179–198, 2007.

[3] S. Gao, J. Avigad, and E. M. Clarke, "δ-complete decision procedures for satisfiability over the reals," in *IJCAR*. Springer, 2012, pp. 286–300.

[4] D. Ishii, K. Ueda, and H. Hosobe, "An interval-based SAT modulo ODE solver for model checking nonlinear hybrid systems," *Int. J. Softw. Tools Technol. Transf.*, vol. 13, no. 5, pp. 449–461, Oct. 2011.

[5] A. Cimatti, A. Griggio, S. Mover, and S. Tonetta, "HyComp: An SMT-based model checker for hybrid systems," in *TACAS*, ser. LNCS, vol. 9035. Springer, 2015, pp. 52–67.

[6] A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani, "The MathSat5 SMT solver," in *TACAS*. Springer, 2013, pp. 93–107.

[7] X. Jin, J. V. Deshmukh, J. Kapinski, K. Ueda, and K. Butts, "Powertrain control verification benchmark," in *HSCC*. ACM, 2014, pp. 253–262.

[8] K. Bae, P. C. Ölveczky, S. Kong, S. Gao, and E. M. Clarke, "SMT-based analysis of virtually synchronous distributed hybrid systems," in *HSCC*. ACM, 2016, pp. 145–154.

[9] J. Raisch, E. Klein, C. Meder, A. Itigin, and S. O'Young, "Approximating automata and discrete control for continuous systems — two examples from process control," in *Hybrid systems V*. Springer, 1999.

[10] T. A. Henzinger, "The theory of hybrid automata," in *LICS*. IEEE Computer Society, 1996, pp. 278–292.

[11] N. Lynch, R. Segala, and F. Vaandrager, "Hybrid I/O automata," *Inf. Comput.*, vol. 185, no. 1, pp. 105–157, 2003.

[12] S. Gao, S. Kong, and E. M. Clarke, "dReal: An SMT solver for nonlinear theories over the reals," in *CADE*. Springer, 2013, pp. 208–214.

[13] K. Bae, J. Krisiloff, J. Meseguer, and P. C. Ölveczky, "Designing and verifying distributed cyber-physical systems using Multirate PALS: An airplane turning control system case study," *Sci. Comput. Program.*, vol. 103, no. C, pp. 13–50, 2015.

[14] S. Gao, S. Kong, and E. M. Clarke, "Satisfiability modulo ODEs," in *FMCAD*. IEEE, 2013, pp. 105–112.

[15] K.-I. Ko, *Complexity theory of real functions*. Birkhäuser, 1991.

[16] K. Bae and S. Gao, "Modular smt-based analysis of nonlinear hybrid systems," manuscript, August 2017. [Online]. Available: https://kquine.github.io/hybrid/fmcad17

[17] V. D. Blondel and J. N. Tsitsiklis, "A survey of computational complexity results in systems and control," *Automatica*, vol. 36, no. 9, 2000.

[18] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[19] M. Fox, D. Long, and D. Magazzeni, "Plan-based policies for efficient multiple battery load management," *J. Artif. Int. Res.*, vol. 44, 2012.

[20] S. Kong, S. Gao, W. Chen, and E. M. Clarke, "dReach: δ-reachability analysis for hybrid systems," in *TACAS*. Springer, 2015, pp. 200–205.

[21] S. Ratschan and Z. She, "Safety verification of hybrid systems by constraint propagation-based abstraction refinement," *ACM Trans. Embed. Comput. Syst.*, vol. 6, no. 1, 2007.

[22] G. Frehse, R. Kateja, and C. Le Guernic, "Flowpipe approximation and clustering in space-time," in *HSCC*. ACM, 2013, pp. 203–212.

[23] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Flow*: An analyzer for non-linear hybrid systems," in *CAV*. Springer, 2013, pp. 258–263.