Superposition Principle in Composable Hybrid Automata

Jafar Akhundov, Peter Tröger, and Matthias Werner

Operating Systems Group, TU Chemnitz, Germany
jafar.akhundov | peter.troeger | matthias.werner@cs.tu-chemnitz.de

Abstract. In the existing abundance of different hybrid automata formalisms concurrent composition is seldom considered or requires additional semantics which is not always defined. This work considers three common reasons of problems with hybrid automata composition: contradicting resets in the discrete transitions, global time reference with contradicting initial conditions and redundant non-determinism for firing time. An overview is provided of the existing formalisms and the attempts to solve these particular problems. A reduced hybrid automata formalism, called linear time-invariant hybrid automata, is introduced. It avoids all those problems and yet provides a powerful modeling tool with practical applications. Also, a short discussion is provided for the problem of Zeno behavior and what conditions are demanded for a model to fulfill so that Zeno behavior would not arise during composition.

1 Introduction

Hybrid systems modeling has various applications in model-driven design and verification of embedded and reactive systems. It has been a topic of intensive research in the past 20 years [MMP91]. Their hallmark is the combination of discrete and continuous behavior. Most hybrid systems include computational components which operate in discrete steps and physical components with continuous behavior over time. Typical examples are aerospace systems, robotic systems, or process control systems. Since most of these systems are too complex to design and build as a whole, they are decomposed into subsystems and components with reduced complexity and simpler behavior. This process can, of course, be recursively repeated until complexity is manageable. In order for this process to be supported on the modeling level, it is necessary that the applied formalism allows for (de)compositionThe process of decomposition has been historically used in the control systems engineering applications [Nis11]. An important property which is often used to simplify design and analysis is the superposition[1] principle which is mathematically defined as:

$$h(ax_1 + bx_2) = ah(x_1) + bh(x_2)$$

[1] also called linearity

Since their introduction, hybrid automata formalisms have been emerging with restricted properties to simplify analysis and sometimes composition [AD94] [Hen96] [LSV03] [Ábr12]. Examples of subsets of hybrid automata are timed automata [AD94], linear hybrid automata [Hen96], rectangular hybrid automata [HKPV98], hybrid I/O automata [LSV03], etc. Several definitions of the general hybrid automata exist as well, each with slight deviations in the underlying semantics.

A handful of frameworks leave some of the semantics unspecified which makes it difficult for the designer to apply them - separately or compositionally [Hen96] [Ras05] [Ábr12] [LLL09]. An example is a general structural definition of HA where each location has an invariant and several outgoing transitions with respective guards [Hen96] [Ras05] [LLL09] [Ábr12]. The problem arises when the invariant is violated thus forcing the automaton to switch its location to another one but no guarding condition of the outgoing transitions is enabled. It remains unspecified what happens to the model in such a situation. Another example is the passage of time in several parallel composed automata with synchronising labeled transitions [Ábr12]. Since the event (action) semantics is not always specified fully and consistently, i.e. are events buffered or ignored, or what is the global time reference for two composed automata, it is unclear whether one synchronising edge should wait for another one with the same label in the second automaton. There are formalisms which allow for such "waiting" which enables to model physical systems where objects are floating in space waiting for some other event to occur.

Thorough comparison of the existing HA formalisms has lead to the conclusion that three common reasons of problems for hybrid automata composition exist:

1. contradicting resets in the discrete transitions,
2. global time reference with contradicting initial conditions and
3. redundant non-determinism for firing time.

For the practical application of composable control systems a formalism is needed which has none of the aformentioned problems and fulfills the property of superposition of continuous functions.

The contribution of this work lies in the introduction of a new formalism for modeling hybrid systems with a fully specified timing, firing, event and composition semantics and fulfilling the property of superposition motivated by the applicability from the control systems engineering. Our approach is driven by the motivating example of a dedicated domain specific language for the verification of a space mission at the early design phases where superposition is a critical issue [ASGW16], [ATW15], [STF$^+$13].

The article starts with an overview of the existing hybrid automata formalisms which experience and/or partially solve the composition problems. The general definition of the utilized hybrid automata variation is given in Section 3. The text continues with a detailed discussion of composition semantics and the arising problems. Section 4 shows how the LTI-HA solve these problems. The paper is concluded by a discussion of further work and possible applications of the formalism.

2 State of the Art

In the existing abundance of different hybrid automata formalisms concurrent composition is seldom considered in full depth or requires additional semantics which is not always defined [Ras05] [LLL09] [Ábr12]. In the general setting, HA experience all of the three mentioned problems [Hen96] [Ras05] [Ábr12] [LLL09] [Alu15]. In [Ábr12], an overview is provided of the existing hybrid automata formalisms with rising complexity, starting from labeled transition systems, timed automata and ending with the general hybrid and rectangular automata. That work provides a conceptualized structural view on the hybrid systems. All three types of problems occur in the generalized HA and at least partially in the other formalisms. Furthemore, many formalisms suffer from incompleteness of semantics definition [Hen96] [Ras05] [LLL09] [Ábr12].

Hybrid I/O automata (HIOA) were introduced by Lynch et al. first in 1996 [LSVW96] but have been modified several times since [LSV03]. The definition of hybrid I/O automata is unique in the sence that it eliminates a handful of problems by defining the hybrid automata by the notion of hybrid traces. Hybrid I/O automata have been demonstrated to be both composable and receptive[2]. However, HIOA are too restrictive for some of the control applications where explicit notion of superposition is important. For example, HIOA are required to have disjunct output trajectories [LSV03, p.131,p.141] which excludes the possibility of superposition.

Superposition of the flow functions of hybrid automata has been exploited in the linear hybrid automata, however, the introduced formalisms still have at least one of the semantic problems listed in the problem statement [Hen96] [Pap98].

3 Linear Time-Invariant Hybrid Automata (LTI-HA)

3.1 Definition

Before the linear time-invariant hybrid automata are defined, several supporting definitions are provided.

Definition 1 (Valuation of a variable). *A valuation $V(x)$ of a variable x is the assignment to x of a value from its domain $\mathcal{D}$: $V(x) : x \mapsto \mathcal{D}(x)$.*

This definition can be extended to a set of variables:

Definition 2 (Valuation of a set of variables). *A valuation $V(X)$ of a variable set X is the union of all valuations for all $x \in X$ of a value from the corresponding domains $\mathcal{D}(x)$: $V(X) : X \mapsto \mathcal{V}(\mathcal{X})$, where $\mathcal{V}(X) = D(x_1) \times D(x_2) \times ... \times D(x_n)$ is the set of all possible valuations.*

[2] Not experiencing Zeno behavior, even under the composition.

The

Definition 3 (State of a hybrid system). *A state of hybrid system is a pair $(L, V)(t)$ consisting of two time-dependent components: the discrete state (L) and the continuous state (V).*

Definition 4 (LTI-HA). *A linear time-invariant hybrid automaton $\mathcal{H}$ is a tuple $(\mathcal{L}, \mathcal{T}, \mathcal{X}, \mathcal{S}_I, \mathcal{S}_O, \mathcal{E}, \mathcal{A}, G, \mathcal{F}, \mathcal{I})$, where:*

- *$\mathcal{L} = (L_1, \dots, L_n)$ is a set of discrete* ***locations*** *also called modes;*
- *$\mathcal{T} \subseteq \mathcal{L} \times \mathcal{L}$ is a (not necessarily complete) multiset* ***transition relation****;*
- *$\mathcal{X}$ is a set of continuous* ***state variables****. To each $x \in \mathcal{X}$, a value from $\mathcal{D}(x) \subseteq \mathbb{R}^m \cup \{dc\}$ can be assigned where $m \geq 1$ but is finite and dc is a special term for unspecified ("don't care") value;*
- *$\mathcal{S}_I$ and $\mathcal{S}_O$ are two disjunct sets of input and output events, respectively, which define the automaton's event signature;*
- *$\mathcal{E} : \mathcal{T} \mapsto \mathfrak{P}(\mathfrak{P}(\mathcal{S}_I))$ and $\mathcal{A} : \mathcal{T} \mapsto \mathfrak{P}(\mathcal{S}_O)$ are assignments of the interface events $\mathcal{S}_I, \mathcal{S}_O$ to the transitions of the automaton;*
- *$G : \mathcal{T} \times \mathcal{V}(\mathcal{X}) \times \mathfrak{P}(\mathcal{S}_O) \times \mathfrak{P}(\mathfrak{P}(\mathcal{S}_I)) \mapsto \{\text{true, false}\}$ is a* ***guard function****. For all transitions τ, $G(\tau, \mathcal{V}(\mathcal{X}), \mathcal{A}(\tau), \mathcal{E}(\tau)) = g_\tau(\mathcal{V}(\mathcal{X}), \mathcal{A}(\tau), \mathcal{E}(\tau))$ is called the* ***guard (function)*** *of τ;*
- *For any location L and for all variables from $\mathcal{X}$ there exists an ordinary linear differential equation $\mathcal{F}(f_L(x,t)) = g(x,t), x \in \mathcal{X}, t \in \mathbb{R}^{\geq 0}$ with $g(x,t) : \mathcal{X} \times \mathbb{R}^{\geq 0} \mapsto \mathbb{R}^m$ describing the change of the corresponding variable, where $f_L(x,t) : \mathcal{X} \times \mathbb{R}^{\geq 0} \mapsto V(\mathcal{X})$ is called a* ***flow function****, $\mathcal{F}(f_L)$ is a linear operator $P_0(t)f_L^{(k)} + P_1(t)f_L^{(k-1)} + ... + P_k(t)f_L$ with $f_L^{(l)} = \dfrac{d^l f_L}{dt^l}$ and $P_i(t) : \mathbb{R} \mapsto \mathbb{R}$ being any functions. The set of all flow functions in a given location describe how the valuations of continuous state variables change over time in that location;*
- *$\mathcal{I}$ is the* ***initial state*** *of the system $(L_\mathcal{I}, V_\mathcal{I})$, where $L_\mathcal{I} \in \mathcal{L}$ is the initial active mode and $V_\mathcal{I} = V_\mathcal{I}(\mathcal{X})$ is the initial valuation of all the variables in $\mathcal{X}$.*

In contrast, the general definition of hybrid automata usually also includes additional constructs such as location invariants, variable resets along the transitions, flow functions are uncostrained and events that are labels with a simple synchronisation semantics.

Definition 5 (Time Semantics). *Evaluation of flow functions is based only on the duration of time interval spent in the corresponding location. In each active location, time elapses at the same rate. Transitions are timeless.*

Global time reference can be implemented by taking any fixed reference time value which is progressing along with the automata execution. At any time, exactly one location is *active*, beginning with the $L_\mathcal{I}$. Automaton's state changes either with time with respect to the flow functions of the corresponding locations or the discrete transitions, starting in the initial state $\mathcal{I}$.

Definition 6 (Transition Semantics). *As long as an automaton has an active location L, the valuation of continuous variables $V(\mathcal{X})$ changes according the location's flow function f_L. If no explicit flow function is given for some variables from $\mathcal{X}$ their rate of change is assumed to be 0 at the given location. If at some time point a guard g_τ of an outgoing transition (L_c, L_d) evaluates to true, L_d becomes the new active location without delay and all events $e \in \mathcal{A}(\tau)$ occur. If more than one guard of an outgoing transition evaluates to true, one of the transitions is chosen non-deterministically.*

Definition 7 (Event Propagation Semantics). *The output events have a one-to-all semantics, that is, every output event is broadcasted. The input events have a one-to-one semantics and are therefore only generated by a single other automaton. Each input event has to be defined and specified.*

Definition 8 (Event Structure Semantics). *The input events for a transition τ form a set $\mathcal{E}(\tau) \in \mathfrak{P}^2(\mathcal{S}_I)$ where $\mathfrak{P}^2(\mathcal{S}_I)$ is a power set of a power set over the set of input events, that is complex events can be formed by coupling the (elementary) input events in the following way: for the transition τ to become enabled, at least one of the (complex) events $S \in \mathfrak{P}^2(\mathcal{S}_I)$ in the set $\mathcal{E}(\tau)$ has to occur. Occurrence of such an event implies that **all** participating events $s \in S$ have occurred (simultaneously).*

Definition 9 (Event Timing Semantics). *Events don't have duration and occurrences are not buffered.*

There are two possibilities to describe *interval events*: by two events, one for the start e_s and one for the completion e_f, respectively, or by setting global variables values. Problem with modeling by just events arises when they are not caught thus leading to either offsets in the interval perceptions or overly complex conditions for well-definedness and composability. Overlapping intervals are easily modeled by global variables with constant values.

3.2 Semantics of the LTI-HA

Definition 10 (Timed Transition System (TTS)). *A timed transition system (TTS) is a tuple $(\Sigma, \Sigma_0, \mathcal{S}, \rightarrow)$ where Σ is a (possibly infinite) state space with $\Sigma_0 \subseteq \Sigma$ being the initial state and $\mathcal{S}$ is a (finite) set of labels. Transition relation is defined as $\rightarrow \subseteq \Sigma \times \mathcal{S} \cup \mathbb{R}^{\geq 0} \times \Sigma$.*

Definition 11 (Trace Semantics of a Hybrid Automaton). *Trace semantics of a hybrid automaton $\mathcal{H} = (\mathcal{L}, \mathcal{T}, \mathcal{X}, \mathcal{S}_I, \mathcal{S}_O, \mathcal{E}, \mathcal{A}, G, \mathcal{F}, \mathcal{I})$ is defined as a transition system where:*

– *the (possibly infinite) state space is the set of pairs $(l, V_l(\mathcal{X}))$, where $V_l(\mathcal{X})$ is in the range of possible valuations in l, defined by f_l*
– *initial state is $\mathcal{I}$*
– *and the transitions "$\rightarrow$" are either:*
    • ***discrete:*** $\forall T \in \mathcal{T} \quad \exists (l_i, V_i(\mathcal{X})) \rightarrow_\sigma (l_j, V_i(\mathcal{X})), \sigma \in \mathfrak{P}(\mathcal{S}), l_i, l_j \in \mathcal{L}$

- *or* ***continuous:*** *$\exists\delta \in \mathbb{R}^{\geq 0}, \delta$ being the time point when the location is left, $\exists(l_i, V_i(\mathcal{X})) \rightarrow_\delta (l_i, V_j(\mathcal{X})) \wedge f_{l_i}$ is differentiable on $[0, \delta]$ and the following conditions hold:*
  1. *$f_{l_i}(0) = V_i(\mathcal{X})$,*
  2. *$f_{l_i}(\delta) = V_j(\mathcal{X})$, and*
  3. *$f_{l_i}[0, \delta]$ is closed under subintervals.*

Thus, a trace of a hybrid automaton is a finite sequence alternating between continuous evolutions with finite durations and discrete transitions:

$$\pi = s_0\epsilon_0 s_1\epsilon_1, ..., \epsilon_{n-1}s_n,$$

where s_i are the states in TTS, ϵ_i are the transitions (discrete or continuous) between them and $s_0 = \mathcal{I}$. Duration of a trace $d(\pi)$ is defined as the sum of all durations along that trace. Since a HA can be non-deterministic, many different traces are possible. Generating a control sequence of external events $\mathcal{C} \subset (S, t)$, where $S \subset \mathcal{S}_I$ and t is a time point with respect to the global time reference, is not part of the model but a task for an external solver.

3.3 Composition of LTI Hybrid Automata

Definition 12 (Composability). *Two LTI hybrid automata $\mathcal{H}^1, \mathcal{H}^2$ are called composable, $\mathcal{H}^1 \mho \mathcal{H}^2$, iff $\forall x \in \mathcal{X}^1 \cap \mathcal{X}^2 : V^1_{\mathcal{I}}(x) = V^2_{\mathcal{I}}(x)$, where $dc = \kappa$ is always true for all values of κ.*

Definition 13 (Composition). *Given two composable hybrid automata $\mathcal{H}^1$ and $\mathcal{H}^2$, the composition $\mathcal{H}^1 \circ \mathcal{H}^2$ provides a new hybrid automaton $\mathcal{H}^c = (\mathcal{L}^c, \mathcal{T}^c, \mathcal{X}^c, \mathcal{S}^c_I, \mathcal{S}^c_O, \mathcal{E}^c, \mathcal{A}^c, G^c, F^c, \mathcal{I}^c)$ where*

1. *$\mathcal{L}^c = \mathcal{L}^1 \times \mathcal{L}^2 = \{(L^1_1, L^2_1), \ldots, (L^1_1, L^2_{n_2}), (L^1_2, L^2_1), \ldots, (L^1_{n_1}, L^2_{n_2})\}$ $= \{L^c_{11}, L^c_{12}, \ldots, L^c_{1n_1}, \ldots, L^c_{n_1n_2}\}$;*
2. *$\mathcal{T}^c = \{t = (L^c_{ij}, L^c_{kl}) | t \in \mathcal{L}^c \times \mathcal{L}^c, (L^1_i, L^1_k) \in \mathcal{T}^1 \vee (L^2_j, L^2_l) \in \mathcal{T}^2\}$*
3. *$\mathcal{X}^c = \mathcal{X}^1 \cup \mathcal{X}^2$,*
4. *$\mathcal{S}^c_I = (\mathcal{S}^1_I \cup \mathcal{S}^2_I) \backslash \mathcal{S}^1_O \backslash \mathcal{S}^2_O$*
5. *$\mathcal{S}^c_O = \mathcal{S}^1_O \cup \mathcal{S}^2_O$*
6. *$\forall \tau = ((L_i, L_j), (L_k, L_l)) \in \mathcal{T}$, where $(L_i, L_j) \in \mathcal{T}^1$ and $(L_k, L_l) \in \mathcal{T}^2$:* *$\forall e \in S^1 \cap S^2 \neq \emptyset$:*
   (a) $e \in S \mid S \in \mathcal{E}^1((L_i, L_j)) \wedge e \in \mathcal{A}^2((L_k, L_l)) \Rightarrow$ $\forall E \in \mathcal{E}^1((L_i, L_j)) : \mathcal{E}^c(\tau) := \mathcal{E}^c(\tau) \cup \{E \backslash e\}$
   (b) $e \in S \mid S \in \mathcal{E}^1((L_i, L_j)) \wedge e \notin \mathcal{A}^2((L_k, L_l)) \Rightarrow$ $\mathcal{T}^c := \mathcal{T}^c \backslash \tau$
   (c) else $\mathcal{E}^c(\tau) = \mathcal{E}^1(\tau) \cup \mathcal{E}^2(\tau)$
   The same process is repeated with the inverted indexes 1 and 2.
7. *$\forall \tau \in \mathcal{T}$: $\mathcal{A}^c(\tau) = \{e \mid \tau = (L^c_{ij}, L^c_{kl}) \wedge$ $((e \in \mathcal{A}^1(L^1_i, L^1_k) \wedge j = l) \vee (e \in \mathcal{A}^2(L^2_j, L^2_l) \wedge i = k) \vee$ $(e \in (\mathcal{A}^1(L^1_i, L^1_k) \cup \mathcal{A}^2(L^2_j, L^2_l))))\}$*

8. $G^c = \{g^c_{(L^c_{ij}, L^c_{kl})} = g^1_{(L^1_i, L^1_k)} | (L^c_{ij}, L^c_{kl}) \in \mathcal{T} \wedge j = l\} \cap$
 $\{g^c_{(L^c_{ij}, L^c_{kl})} = g^2_{(L^2_j, L^2_l)} | (L^c_{ij}, L^c_{kl}) \in \mathcal{T} \wedge i = k\}$
9. $\forall L_{ij} \in \mathcal{L}^c$: $f_{L_{ij}} = f_{L_i} + f_{L_j}$
10. $\mathcal{I}^c = (L^c_{i_1 i_2}, V^1_{\mathcal{I}} \cup V^2_{\mathcal{I}})$

Properties from semantics definitions 5-9 remain preserved.

Properties 1 and 2 define the new location set which is now a cartesian product of two initial location sets, and the transition in a new location set exists if there was at least one transition in the corresponding locations of initial automata $\mathcal{H}^1$ and $\mathcal{H}^2$. The variable set is defined as a union set. All variables with the same names are to be considered global and can be adjusted in a composed manner (property 9). Properties 4-7 describe how the input and output events of the automata are composed. Since the input events have the one-to-one semantics, if one of the two composed automata $\mathcal{H}^1$ and $\mathcal{H}^2$ has an output event which is the input event for the second automata this event can be cancelled out in the input set of the resulting composed automaton. This, however, does not apply for the output events because of their one-to-all semantics. Properties 6-8 define how the input and output events and the guards are assigned to their corresponding transitions in the initial automata. For assignment of input interfaces to the new transitions four possibilities are distinguished: an edge waiting for an event in one automaton is combined with the generating edge of the other automaton, a waiting edge with non-generating edge, non-waiting with the generating edge and non-waiting with non-generating edges. In the first case, the generated event is removed from all the complex events of the input assignment. In the second case, the transition will never be taken since a waiting edge is waiting for an event which is not generated and not buffered. For the remaining two combinations it is safe to just unite the input assignments. Mildly speaking, G builds a cut set of enabling valuations in two composed transitions and a union of the input and output events, respectively. Property 9 follows trivially from the linearity property of the superpositioned differential equations. Since $\mathcal{H}^1$ and $\mathcal{H}^2$ are composable it is safe to apply property 10.

4 Semantics of Composition of Hybrid Automata

4.1 General Hybrid Automata

In the general setting, hybrid automata are defined as follows [Hen96, p.2] [Ras05, p.4] [LLL09] [Ábr12]:

Definition 14. *A hybrid automaton consists of:*

- *A set of continuous variables X;*
- *A finite directed multigraph (V, E) representing the discrete modes and the transitions between them;*
- *Initial conditions describe how the continuous variables are reset after a timeless discrete transition has been taken;*

- *Invariants are the predicates assigned to the discrete locations and must hold in the respective location when it is active;*
- *Flow conditions describe how the variables in X change continuously with time;*
- *Guard conditions which are the predicates over the values of variables of X and are the enabling conditions for a transition to be taken;*
- *Events which are assigned to the transitions of the automaton. Transitions with with the same labels in different automatas must synchronise.*

There are some extensions and small differences between the definitions which both solve some problems and introduce others. For instance, in [Ábr12] it is proposed that discrete transitions in concurrent automata are interleaved, and synchronisation is enhanced with special τ-transitions for the automaton which is waiting on the synchronising edge. It is immediately clear that this extension solves the problem of race conditions of the resets with non-determinism for the case when, e.g. two transitions with different labels fire simultaneously with the following resets: $x = y + 1$ and $y = x + 1$. However, introducing τ-transitions allows for modeling of a satellite in the orbit which, while waiting for some maneuver comand from an operations control center, suddenly freezes dead in its orbit since τ-transitions are also timeless. Furthermore, it does not solve a problem of possible Zeno behavior when time is prevented from passing. A trivial example is given in Fig. 1. If the automata start in states A and C with $x = 0, y = 0$, then after 10 time units, the system will converge and generate an infinite amount of discrete events. The execution trace would be $(A \to B) \Rightarrow (C \to D) \Rightarrow (B \to A) \Rightarrow (D \to C) \Rightarrow ...$

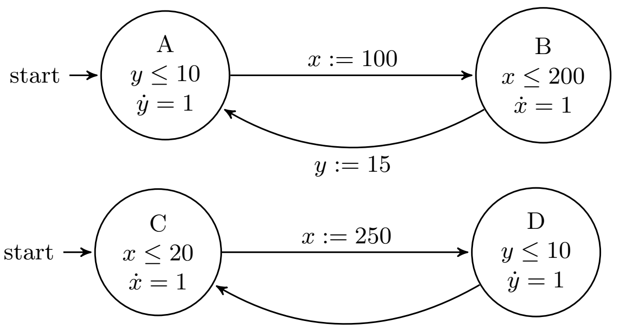

Fig. 1: Example of time convergence because of the discrete resets

In the general setting, usually no assumption is made[3] about the type of differential equations governing the continuous change of the state variables. If those are not time invariant, it leaves the question open as to how the flow functions are overlapped during parallel composition. Another important drawback

[3] An exception would be [LLL09]

is the lack of specification of the semantics in the case when a state invariant is violated prior to enabling of any outgoing transition [HKPV98] [Ras05] [LLL09] [Ábr12].

4.2 Other Formalisms

Several other formalisms based on the general hybrid automata have been presented [Hen96] [Ras05] [LLL09] [Ábr12].

An overview of the three possible problems mentioned earlier occurring in the HA formalisms is provided in table (1). Timed automata, being the simplest form of the HA, avoid most of the problems of composition, as well as lack expressivity to describe complex hybrid phenomena [LLL09] [Cas05]. Linear hybrid automata have support for superposition but still have the invariants and resets, and, hence, the implied problems that could arise. Rectangular automata, just as the general automata, experience all of the three problems and also increase the complexity by introducing randomness and uncertainties [Hen96] [HKPV98].

Hybrid I/O automata solve all of the mentioned problems of composition but have explicitly eliminated the possibility for superposition of trajectories for the continuous variables [LSV03, p.131,p.141].

HA Formalism	Problems with composition			Superposition Support
	R	IC	FTND	
Timed Automata [AD94]	-	-	✓	-
Linear Hybrid Automata	✓	-	✓	✓
I/O Hybrid Automata	-	-	-	-
General Hybrid Automata	✓	✓	✓	-
Rectangular Hybrid Automata	✓	✓	✓	-

Table 1. Compositional problems with the HA formalisms - R: resets; IC: initial conditions; FTND: fire time non-determinism.

4.3 Composition in LTI Hybrid Automata

Since composition is the cornerstone of the new formalism that is introduced in this paper, the absence of each of the three undesirable prooerties $\mathcal{P}_i$ can be only guaranteed if and only if LTI hybrid automata as a formalism fulfill the following two conditions:

1. the property $\mathcal{P}_i$ cannot exist in a single automaton;
2. the $\mathcal{P}_i$-freeness is preserved and $\mathcal{P}_i$ not induced by composition,

where $\mathcal{P}_1 \equiv$ R, $\mathcal{P}_2 \equiv$ IC and $\mathcal{P}_3 \equiv$ FTND.

Theorem 1. *No contradicting resets are possible in the LTI hybrid automata.*

Proof

1. Discrete transition resets are not a part of the LTI definition 4. Furthermore, since there is no composition taking place, no contradictions are possible. The proof follows trivially.
2. The proof of preservation also follows trivially from the definition 13 of composition, since no rule introduces discrete resets. The only discrete jumps of the variable values are possible due to Dirac impulses which do not violate the superposition and time-invariance property.□

Theorem 2. *No global time reference exists with the contradicting initial conditions in the LTI hybrid automata.*

Proof

1. As per definition 5, time flow is identical for all discrete states, hence time invariance. Definition 4 implies that the flow functions are also linear. Since every variable can only be assigned with value once in the initial state, no contradictions are possible.
2. Proving that this statement is preserved and not induced by composition is equivalent to proving that if two automata are composed with each other the induced automaton is also composable with some other third automaton, since non-contradicting initial conditions are the necessary and sufficient condition for composability.
   We assume that automata $\mathcal{H}^1, \mathcal{H}^2$ and $\mathcal{H}^3$ are pairwise composable. Without loss of generality, rules 3 and 10 of the composition definition 13 are applied to automatas $\mathcal{H}^1, \mathcal{H}^2$:

$$\text{Theorem 2}\frac{\mathcal{H}^1 \mho \mathcal{H}^2, \mathcal{H}^2 \mho \mathcal{H}^3, \mathcal{H}^1 \mho \mathcal{H}^3}{(\mathcal{H}^1 \circ \mathcal{H}^2) \mho \mathcal{H}^3}$$

   After applying rule 13.11, $V_{\mathcal{I}}^{12} = V_{\mathcal{I}}^1 \cup V_{\mathcal{I}}^2$. Thus, set $V_{\mathcal{I}}^{12}$ can be divided to three subsets, elements only from $V_{\mathcal{I}}^1$, elements only from $V_{\mathcal{I}}^2$ and elements from $V_{\mathcal{I}}^1 \cap V_{\mathcal{I}}^2$. Let us assume that $V_{\mathcal{I}}^1 \cap V_{\mathcal{I}}^3 \neq \emptyset$ and $V_{\mathcal{I}}^2 \cap V_{\mathcal{I}}^3 \neq \emptyset$. Then, from the initial assumption and the definition 12,

$$\forall x \in \mathcal{X}^1 \cap \mathcal{X}^3 : V^1(x) = V^3(x) \wedge$$
$$\forall x \in \mathcal{X}^2 \cap \mathcal{X}^3 : V^2(x) = V^3(x)$$

   Hence,

$$\forall x \in \mathcal{X}^1 \cap \mathcal{X}^2 \cap \mathcal{X}^3 : V^1(x) = V^2(x) = V^3(x)$$

   Linearity and time-invariance are preserved with respect to superposition [Nis11]. Since all of the flow functions in the LTI-HA are linear and time-invariant, the same applies for the composed automata after applying rules 1 and 9 of definition 13. □

Theorem 3. *Execution of LTI-HA never stalls due to the contradicting invariants and guard conditions.*

Proof

1. Invariants are absent in the definition of the LTI-HA 4. Hence, it is not possible for the behavior of the model to be unspecified due to a violated invariant with no guards enabled.
2. The rule for guards in the composition definition, 13.8, combines several guards of the initial automata. If the guards are contradicting each other, transition is omitted altogether. No invariants are created and the firing enforcement (definition 6) is preserved. □

5 Discussion

Although some of the common problems mentioned in the introduction of this work have been eliminated in the LTI-HA, others may remain which cannot be completely excluded for hybrid systems or are implied by the semantics of the modeled system itself. It is therefore useful to determine a set of properties, which, combined, will introduce a notion of well-definedness of the model. A well-defined model would guarantee correct behavior with respect to the property of interest, that is, model would behave without experiencing unexpected or unwanted behavior. One of such properties is divergence of time. If a model is Zeno-free, time never converges, i.e. it is impossible to find a subsequence of the model execution trace which includes an infinite amount of events in a finite time. This notion has been extended by Lynch et al. in [LSV03] with the case where a trajectory of a continuous variable is never asymptotic.

Introduction of such a notion into the LTI-HA formalism would allow for automatic checking if a model can or cannot end in a Zeno executionOne possible condition for guaranteeing Zeno-freeness would be the absence of closed transition loops of length ≥ 1 consisting of only transient modes, i.e. modes for which at least one outgoing transition is enabled when the mode is entered. If there exists such a cycle, then the model will be executed, following the transition semantics 6, endlessly without the progress of time.

6 Conclusion

We presented a formalism that allows the semantic description of linear control systems based on hybrid automata. Several problems of composition of other formalisms have been demonstrated with a comparative analysis to our method.

As next steps, we intend to derive the necessary and sufficient conditions for non-Zenoness of the LTI-HA models and demonstrate that this property is preserved by composition. Furthermore, the problem of analysis of liveness and reachability and the implementation of a corresponding tool support will follow.

References

[Ábr12] Erika Ábrahám. Modeling and analysis of hybrid systems: Lecture notes, April 2012.

[AD94] Rajeev Alur and David L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126:183–235, 1994.

[Alu15] R. Alur. *Principles of Cyber-physical Systems*. 2015.

[ASGW16] Jafar Akhundov, Volker Schaus, Andreas Gerndt, and Matthias Werner. Using timed automata to check space mission feasibility in the early design phases. In *IEEE Aerospace 2016 Proceedings*, Big Sky, Montana, USA, March 2016.

[ATW15] Jafar Akhundov, Peter Tröger, and Matthias Werner. Considering concurrency in early spacecraft design studies. In *CS&P 2015 Proceedings*, pages 22–30, Rzeszow, Poland, 9 2015.

[Cas05] B. Brard F. Cassez. Comparison of the expressiveness of timed automata and time Petri nets. In *In Proc. FORMATS05, vol. 3829 of LNCS*, pages 211–225. Springer, 2005.

[Hen96] T. A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, LICS '96, pages 278–, Washington, DC, USA, 1996. IEEE Computer Society.

[HKPV98] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94 – 124, 1998.

[LLL09] Jan Lunze and Franoise Lamnabhi-Lagarrigue, editors. *Handbook of hybrid systems control : theory, tools, applications*. Cambridge University Press, Cambridge, UK, New York, 2009.

[LSV03] Nancy Lynch, Roberto Segala, and Frits Vaandrager. Hybrid I/O automata. *Inf. Comput.*, 185(1):105–157, August 2003.

[LSVW96] Nancy Lynch, Roberto Segala, Frits Vaandrager, and H. B. Weinberg. *Hybrid I/O automata*, pages 496–510. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[MMP91] Oded Maler, Zohar Manna, and Amir Pnueli. From timed to hybrid systems. In *Real-Time: Theory in Practice, REX Workshop, Mook, The Netherlands, June 3-7, 1991, Proceedings*, pages 447–484, 1991.

[Nis11] N.S. Nise. *Control Systems Engineering*. Wiley, 2011.

[Pap98] G. Pappas. Hybrid Systems: Computation and Abstraction. 1998.

[Ras05] Jean-François Raskin. *An Introduction to Hybrid Automata*, pages 491–517. Birkhäuser Boston, Boston, MA, 2005.

[STF$^+$13] Volker Schaus, Michael Tiede, Philipp M. Fischer, Daniel Lüdtke, and Andreas Gerndt. A Continuous Verification Process in Concurrent Engineering. In *AIAA Space Conference*, September 2013.

Appendix: A Modelling and Composition Example

To further motivate the use of composable hybrid automata as introduced in this work, a small practical example for satellite functionality is discussed in this Appendix.

In the early conceptual study phase of development, a satellite downlink module which sends gathered information back to Earth can be modelled as having only two distinct states: *Sending*, when a ground station is visible and there is data to send, or *Not Sending*, when either no ground station is available or no data is there to be sent (or both). In the *Sending* state the rate of change of available data and sent data is the same with opposite signs, whereas in the *Not Sending* state both parameters remain constant (Fig. 2).

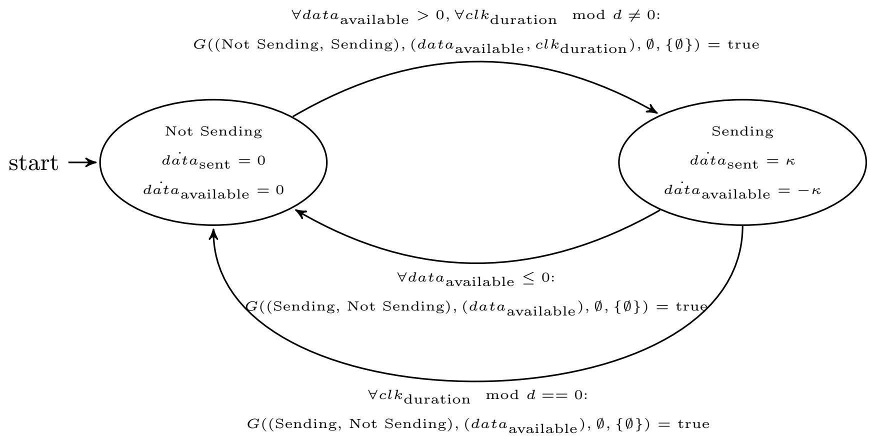
$\forall data_{\text{available}} > 0, \forall clk_{\text{duration}} \mod d \neq 0$:
$G((\text{Not Sending, Sending}), (data_{\text{available}}, clk_{\text{duration}}), \emptyset, \{\emptyset\}) = \text{true}$
start
Not Sending
$\dot{data}_{\text{sent}} = 0$
$\dot{data}_{\text{available}} = 0$
Sending
$\dot{data}_{\text{sent}} = \kappa$
$\dot{data}_{\text{available}} = -\kappa$
$\forall data_{\text{available}} \leq 0$:
$G((\text{Sending, Not Sending}), (data_{\text{available}}), \emptyset, \{\emptyset\}) = \text{true}$
$\forall clk_{\text{duration}} \mod d == 0$:
$G((\text{Sending, Not Sending}), (data_{\text{available}}), \emptyset, \{\emptyset\}) = \text{true}$

Fig. 2: Downlink module model definition using a composable LTI hybrid automaton.

The automaton representing ground station availability is presented in Fig. 3. Here, as well, the system has only two states, since a ground station is either available or not. For now, we ignore irregularities of this otherwise periodic process, such as orbit perturbations and communication faults - they can be integrated into the model by taking the worst, shortest possible availability period. Ground station visibility is modelled by two running clocks, one for the period and one for the duration. Once the period time is up, ground station becomes visible for the possible duration time which is measured by the second clock, clk_{duration}. When the clock reaches its maximum value, the automaton switches its discrete state back to the "Not Visible" mode. Since there are no resets in the LTI-HA formalism, it is not possible to reset the clocks. Hence, modular arithmetic is applied. When either of the transitions is taken in the ground station automaton, an output event is generated, one for the start of the visibility period, and one for the end, so that other concurrent automatas could synchronise their transitions with this periodic interval. However, it is also possible to model communication by using the global values of the two clocks of the ground station automaton. This approach is used in the Fig. 2 - it is easy to see that whenever the value of the clock clk_{duration} is not zero modulo the interval duration, the ground station automaton is in its visible state, so the transition from "Not Sending" to "Sending" modes remains enabled. However, when the value is zero modulo interval duration and the downlink module is active, or there is no data to be sent, it should switch back to the "Not Sending mode".

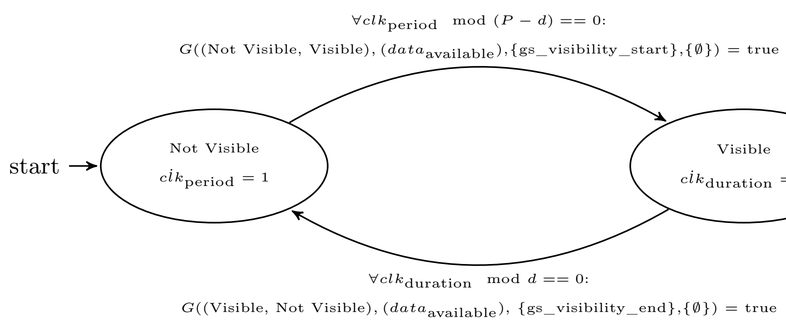
$\forall clk_{\text{period}} \mod (P - d) == 0$:
$G((\text{Not Visible, Visible}), (data_{\text{available}}), \{\text{gs\_visibility\_start}\}, \{\emptyset\}) = \text{true}$
start
Not Visible
$\dot{clk}_{\text{period}} = 1$
Visible
$\dot{clk}_{\text{duration}} = 1$
$\forall clk_{\text{duration}} \mod d == 0$:
$G((\text{Visible, Not Visible}), (data_{\text{available}}), \{\text{gs\_visibility\_end}\}, \{\emptyset\}) = \text{true}$

Fig. 3: Periodic ground station visibility model definition using a composable LTI hybrid automaton.

It is assumed that initial states for the initial automata are ("Not Sending", $\{data_{\text{sent}} = 0,\ data_{\text{available}} = C\}$) and ("Not Visible", $\{clk_{\text{period}} = 0, clk_{\text{duration}} = 0\}$). Obviously, both are composable, since the cut set of their initial valuations does not have contradictions. The composed automaton is built by applying composition rules 1-10 and its control graph is depicted in Fig. 4:

1. $\mathcal{L}^c$ = (('Not Sending', 'Not Visible'), ('Not Sending', 'Visible'), ('Sending', 'Not Visible'), ('Sending', 'Visible')) = (nsnv, nsv, snv, sv)
2. $\mathcal{T}^c$ = { ((nsnv, nsv), (nsnv, snv), (nsnv, sv), (snv, nsv), (snv, nsv), (snv, sv), (snv, nsnv), (snv, nsnv), (nsv, sv), (nsv, snv), (nsv, nsnv), (sv, nsv),(sv, nsv), (sv, nsnv), (sv, nsnv), (sv, snv))}
3. $\mathcal{X}^c = \{data_{\text{available}}, data_{\text{sent}}, clk_{\text{period}}, clk_{\text{duration}}\} \cup \{clk_{\text{period}}, clk_{\text{duration}}\} = \{data_{\text{available}}, data_{\text{sent}}, clk_{\text{period}}, clk_{\text{duration}}\}$,
4. $\mathcal{S}_I^c = \emptyset \cup \emptyset = \emptyset$
5. $\mathcal{S}_O^c = \emptyset \cup$ {gs_visibility_start, gs_visibility_end} = {gs_visibility_start, gs_visibility_end}
6. It is clear that for all the cases when only one of the states in a pair changes, transitions remain unchanged from the corresponding initial automaton. The output events of the corresponding edges are:
   – (nsnv, nsv): { gs_visibility_start };
   – (nsnv, snv): $\emptyset$;
   – (nsnv, sv): { gs_visibility_start };
   – (snv, nsv): { gs_visibility_start };

- (snv, nsv): { gs_visibility_start };
- (snv, sv): { gs_visibility_start };
- (snv, nsnv): $\emptyset$;
- (snv, nsnv): $\emptyset$;
- (nsv, sv): $\emptyset$;
- (nsv, snv): { gs_visibility_end };
- (nsv, nsnv): { gs_visibility_end };
- (sv, nsv): $\emptyset$;
- (sv, nsv): $\emptyset$;
- (sv, nsnv): { gs_visibility_end };
- (sv, nsnv): { gs_visibility_end };
- (sv, snv): { gs_visibility_end }.

7. Since the set of input events is empty, $\{\emptyset\}$ is assigned as input events to the all of the transitions.
8. The guards of the resulting transitions are:
   - (nsnv, nsv): $g((\forall clk_{\text{period}} \mod (P{-}d) == 0), \{\text{ gs\_visibility\_start }\}, \{\emptyset\}) = \mathit{true}$;
   - (nsnv, snv): $g((\forall data_{\text{available}} > 0, \forall clk_{\text{duration}} \mod d \neq 0), \emptyset, \{\emptyset\}) = \mathit{true}$; this transition will never be taken, since this condition cannot be fulfilled before the ground station becomes visible;
   - (nsnv, sv): $g((\forall data_{\text{available}} > 0, \forall clk_{\text{duration}} \mod d \neq 0, \forall clk_{\text{period}} \mod (P{-}d) == 0), \{\text{ gs\_visibility\_start }\}, \{\emptyset\}) = \mathit{true}$ this transition is never taken since decision about taking a transition falls *before* time starts ticking in the ground station visibility mode which is required to enable the second condition;
   - (snv, nsv): $g((\forall data_{\text{available}} \leq 0), \{\text{ gs\_visibility\_start }\}, \{\emptyset\}) = \mathit{true}$; this transition will never be taken, since the ground station should have been visible for the sending to be possible before this transition;
   - (snv, nsv): $g((\forall clk_{\text{duration}} \mod d == 0), \{\text{ gs\_visibility\_start }\}, \{\emptyset\}) = \mathit{true}$; same as the last point;
   - (snv, sv): $g((\forall clk_{\text{period}} \mod (P{-}d) == 0), \{\text{ gs\_visibility\_start }\}, \{\emptyset\}) = \mathit{true}$; same as last point;
   - (snv, nsnv): $g((\forall data_{\text{available}} \leq 0), \emptyset, \{\emptyset\}) = \mathit{true}$; this transition will never be taken since the automaton cannot send when the ground station is unavailable;
   - (snv, nsnv): $g((\forall clk_{\text{duration}} \mod d == 0), \emptyset, \{\emptyset\}) = \mathit{true}$; same as the last point. The only exception is if the snv state is transitive and was jumped in from the state sv;
   - (nsv, sv): $g((\forall data_{\text{available}} > 0, \forall clk_{\text{duration}} \mod d \neq 0), \emptyset, \{\emptyset\}) = \mathit{true}$
   - (nsv, snv): $g((\forall data_{\text{available}} > 0, \forall clk_{\text{duration}} \mod d \neq 0, \forall clk_{\text{duration}} \mod d == 0), \{\text{ gs\_visibility\_end }\}, \{\emptyset\}) = \mathit{true}$; contradicting conditions, transition will never be taken;
   - (nsv, nsnv): $g((\forall clk_{\text{duration}} \mod d == 0), \{\text{ gs\_visibility\_end }\}, \{\emptyset\}) = \mathit{true}$
   - (sv, nsv): $g((\forall data_{\text{available}} \leq 0), \{\emptyset\}, \{\emptyset\}) = \mathit{true}$

  - (sv, nsv): $g((\forall clk_{\text{duration}} \mod d == 0), \emptyset, \{\emptyset\}) = true$ this transition will never be taken since the automaton cannot stop sending available data while the ground station is available;
  - (sv, nsnv): $g((\forall data_{\text{available}} \leq 0, \forall clk_{\text{duration}} \mod d == 0), \{$ gs_visibility_-end $\}, \{\emptyset\}) = true$
  - (sv, nsnv): $g((\forall clk_{\text{duration}} \mod d == 0), \{$ gs_visibility_end $\}, \{\emptyset\}) = true$
  - (sv, snv): $g((\forall clk_{\text{duration}} \mod d == 0)$, $\{$ gs_visibility_end $\}$, $\{\emptyset\}) = true$; this transition will be immediately followed by the (snv, nsnv), since the guard is also fulfilled

9. The flow functions of the resulting automaton are:
   - nsnv: $\dot{data}_{\text{sent}} = 0,\ \dot{data}_{\text{available}} = 0,\ \dot{clk}_{\text{period}} = 1, \dot{clk}_{\text{duration}} = 0$
   - nsv: $\dot{data}_{\text{sent}} = 0,\ \dot{data}_{\text{available}} = 0,\ \dot{clk}_{\text{period}} = 0, \dot{clk}_{\text{duration}} = 1$
   - snv: $\dot{data}_{\text{sent}} = \kappa,\ \dot{data}_{\text{available}} = -\kappa,\ \dot{clk}_{\text{period}} = 1, \dot{clk}_{\text{duration}} = 0$
   - sv: $\dot{data}_{\text{sent}} = \kappa,\ \dot{data}_{\text{available}} = -\kappa,\ \dot{clk}_{\text{period}} = 0, \dot{clk}_{\text{duration}} = 1$
10. Initial state of the composed automaton is given as:
    $\mathcal{I}^c = (nsnv, \{data_{\text{sent}} = 0,\ data_{\text{available}} = C\} \cup \{clk_{\text{period}} = 0, clk_{\text{duration}} = 0\}) = (nsnv, \{data_{\text{sent}} = 0,\ data_{\text{available}} = C, clk_{\text{period}} = 0, clk_{\text{duration}} = 0\})$

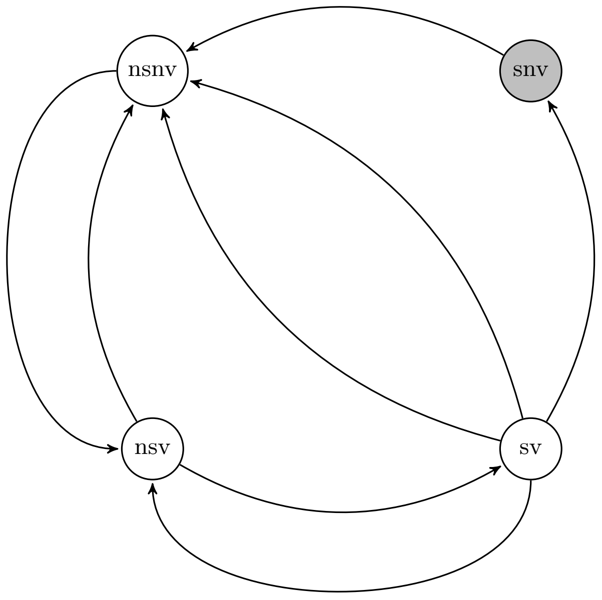

Fig. 4: The resulting control graph for the composed automaton. Labels are omitted for simplicity. Since snv-state is transient, it is coloured.