Modular, Hierarchical Models of Control Systems in SpaceEx

Alexandre Donzé[1], Goran Frehse[2]

***Abstract*— The Hybrid I/O-automaton (HIOA) is a rigorous formal model designed for the analysis of complex hybrid (discrete-continuous) dynamical systems. The use of the HIOA formalism renders compositional reasoning possible, in the sense that once a property has been established for an automaton, it still holds if the automaton is composed with other automata. In this paper, we show how control systems can be modeled and verified in SpaceEx as HIOA in a modular fashion. Formally, HIOA models distinguish between controlled and uncontrolled variables. With examples and usage guidelines we relate these to the concepts of input/output and state/algebraic variables most control designers are familiar with. Additionaly, we enlargen the applicability of our HIOA by allowing variables to be controlled in more than one automaton. While this invalidates compositionality, it gives users who do not intend to use compositional reasoning more freedom in their modeling choices. Finally, we show how we can algorithmically bring control systems given by semi-explicit differential algebraic equations to the form understood by the SpaceEx reachability algorithm.**

I. Introduction

With recent progress in verification techniques, tools such as SpaceEx can now be used to compute the reachable states of systems with piecewise affine dynamics involving hundreds of state variables [1]. Control systems are a promising application domain for these methods, but they are not typically given directly in the form suitable to SpaceEx, whose algorithms are based on a monolithic (flat) model involving ODEs. Consider the classic control system structure shown in Fig. 1. It consists of set of connected components – a plant, an observer, a controller, and a sensor. SpaceEx models are hybrid automata, which define the evolution of continuous variables over time with a set of differential equations. The automaton can instantaneously switch between different locations (modes), each with its own set of equations. A model hierarchy is realized using the SpaceEx model language [2], since components (hybrid automata) can be composed to form new components. Common components can be shared and reused between models in much the same way as Simulink blocks are in Matlab/Simulink [3].

The verification algorithms implemented in SpaceEx require the continuous dynamics of the system to be given in the form of a linear ordinary differential equation (ODE),

$$\dot{x}(t) = Ax(t) + u(t), \qquad u(t) \in \mathcal{U}, \tag{1}$$

*This work was supported in part by the EU project MULTIFORM under grant INFSO-ICT-224249.

[1]A. Donzé is with the Department of Electrical Engineering and Computer Science of UC Berkeley, donze@eecs.berkeley.edu

[2]G. Frehse is with the Department of Computer Science of Grenoble University at Verimag labs, frehse@imag.fr

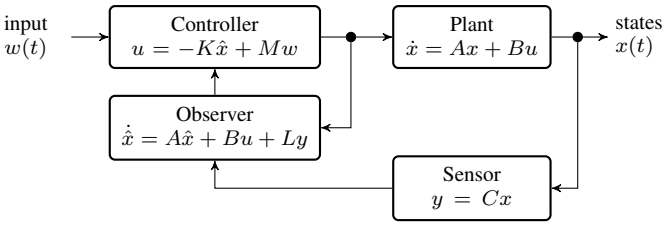


Fig. 1. Block diagram of a linear plant with proportional state feedback controller and Luenberger observer

where $x(t) \in \mathbb{R}^n$, A is a real-valued $n \times n$ matrix and $\mathcal{U} \subseteq \mathbb{R}^n$ is a closed and bounded convex set. The compositional modeling of control system components requires algebraic constraints. For the classic proportional state feedback controller shown in Fig. 1, the algebraic constraints are the feedback controller and the sensor equations,

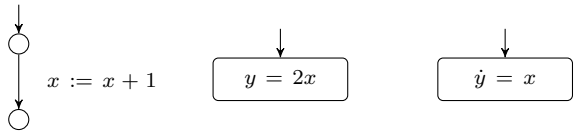
$$u = -K\hat{x} + Mw \quad \text{and} \quad y = Cx.$$

In this paper, we discuss how control systems can be modeled in a modular way using the modeling formalism of SpaceEx, hybrid I/O automata. In particular, we motivate the use of the HIOA formalism and illustrate its application to control problems through examples. We also show how we bring the semi-explicit DAEs that arise in control systems to the ODE form used in the SpaceEx reachability algorithm. This transformation is non-trivial since the SpaceEx models can include nondeterministic disturbances in both ODEs and discrete assignments to variables. For lack of space, we omit syntactic details and refer the reader to [2].

In Sect. II, we try to illustrate why it is useful to distinguish different types of variables, and how these types are related. In Sect.III, we present the HIOA formalism used in SpaceEx and show that it does not reduce the modeling capabilities compared to using more general hybrid automata. Finally, Sect. IV presents a transformation from DAEs to ODEs to bring control system models to a form suited to our reachability algorithm.

II. A Brief Zoology of Variables

Browsing the literature on hybrid systems and control theory (both theory and tools), one finds that variables are classed in different categories with distinct semantics: state vs algebraic variables, input vs output variables, and controlled vs uncontrolled variables. While the first category arises from the model equations, the latter two may be less obvious to choose in the modeling process. But they can help us to simplify the notation and to model systems in a modular



(a) component H_1 (b) component H_2 (c) component H_3

Fig. 2. In this example, a local change of x in H_1 is intended to change y in H_2, but leave y unchanged in H_3

yet formally sound and unambiguous manner. This in turn helps with model reuse and with compositional analysis.

The above categories are a priori unrelated, but – as will be detailed below – a good starting point is to consider

- input and algebraic variables as uncontrolled,
- state variables as controlled in the component where their derivative is defined.

Example II.1. Consider a hybrid system with two components shown in Fig. 2(a) and Fig. 2(b). The first component, H_1, has a discrete transition that changes the value of a variable x, say $x := x + 1$. The second component, H_2 has no transitions, only the invariant $y = 2x$. This could model, e.g., a sensor measurement of the variable x, or a feedback law, so it is entirely reasonable to have this modeled in a component separate from H_1. If the intended behavior is that y is determined by the value of x, then the discrete transition in component H_1 must change both x and y simultaneously, even though component H_2 makes no explicit mention of any discrete transition, never mind synchronization.

Now consider H_1 together with H_3 shown in Fig. 2(c), which has no invariant but the flow equation $\dot{y} = x$. Here, the intended meaning is that y integrates x, but does not change in any other way. Running H_1 and H_3 together, the discrete transition in H_1 is not supposed to change y. Comparing the two settings, H_1 together with H_2 and H_1 together with H_3, the transition that is local to H_1 has a completely different effect. Since in both cases the component H_1 is the same, one must somehow be able to specify in the other components whether y is supposed to remain unchanged or not if H_1 carries out a transition.

A. Controlled/Uncontrolled Variables

The notion of controlled/uncontrolled variables allows one to specify whether transitions of *other* components may change a variable [4], [5], [6]. A hybrid automaton has a set of labels, and each discrete transition in the automaton is associated with one of these labels. When two automata are put together or "composed", they synchronize on shared labels as follows. If both automata are in a location with a discrete transition of the same label, both transitions are executed as a single instantaneous event. If either automata in a location does not have a transition with a shared label or can not take it, the transition of the other automaton is blocked as well. Transitions with labels that are not shared can take place independently. We call such transitions *local*.

With that out of the way, we are ready to define controlled variables: A *controlled variable* remains constant whenever other components carry out local transitions, except if they also declare it as a controlled variable. An *uncontrolled variable* may change at any time to any value, as long as it satisfies the invariant.

The latter requirement is necessary to be consistent with the fact that local transitions in other components *are* allowed to change an uncontrolled variable at any time. In a compositional framework, these changes need to be present in any component that has the uncontrolled variable. This is the essence of what differentiates a compositional modeling framework from one that is not, as will be discussed in Sect. III. Formally, one defines implicit self-loop transitions in every location that may at any time reset the uncontrollable variables to any value in the invariant.

Example II.2. Consider again Ex. II.1. In component H_2, we need to declare variable y as uncontrolled, so that y can be subjected to change when component H_1 takes its transition. In component H_3, we need to declare variable y as controlled, so y remains constant when component H_1 takes its transition.

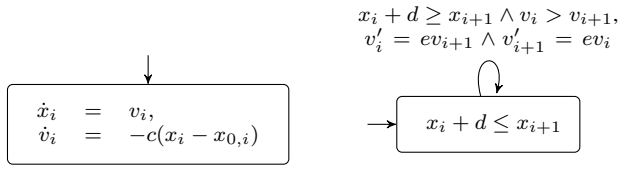
When modeling and visualizing hybrid automata, it is established practice to only mention in transitions the variables that change, with the implicit assumptions that all other variables remain constant [7]. As Ex. II.1 illustrates, this practice can easily lead to modeling mistakes. Declaring variables as controlled or uncontrolled can to some extent help with preventing mistakes. In SpaceEx, variables that are not explicitly assigned in a transition remain constant if they are controlled, and can change arbitrarily otherwise.

B. Input and Output Variables

In control systems such as the one in Fig. 1, input and output variables arise naturally from the definition of the system: the output variables are those that are measurable or relevant to the outside (or the specification), here x, while the input variables are those with which one intends to control the behavior of the system, here w. If the system is given by an ODE $\dot{x} = f(x, u)$, the inputs are the independent variables u, which are characterized by the fact that their derivatives are unconstrained.

One usually studies the evolution of the outputs as a function of the "open" (undefined) inputs. Our verification algorithms are not suited to open systems, and require the inputs of the control system to be given or at least restricted to a useful degree by an input generator. In modular modeling, it is often imposed that components can only use variables of other components if they declare them as inputs, and the other components declare them as outputs.

The distinction between inputs and outputs is less evident when acausal relationships between different elements of a system may exist, which is easily the case when regarding low level components. Acausal relationships abound in domains like mechanics, electrical circuits and thermodynamic systems. They are difficult to model in a modular way by models with sequential operational semantics, such as standard Matlab/Simulink models [3], in which the inputs



(a) Model P_i of the ith pendulum, freely swinging

(b) Model C_i of the collision between two adjacent pendulums

Fig. 3. A modular model of N colliding pendulums, consisting of one component for each of the N pendulums in its freely swinging form, and of one component for each of the $N-1$ collisions

have to be known at each time instant before the corresponding outputs can be computed. The need for more general component interactions is satisfied by modeling frameworks such as Modelica [8], Matlab extensions such as SimScape [3], and tools such as gPROMS [9].

Example II.3. A modular model of N colliding pendulums (linearized) is shown in Fig. 3. It consists of one automaton for each of the N pendulums in its freely swinging form, and of one automaton for each of the $N-1$ collisions. The pendulum positions are given x_i, their velocities by v_i and their position at rest by $x_{0,i}$. The pendulum bobs are considered balls of diameter d. The parameter c determines their oscillation frequency and e is a factor between 0 and 1 that models the energy loss at collision. The model is elegant but which variables are to be considered in- and outputs? The collision model uses position and velocity variables from the pendulum models, so they could be inputs. But it changes the velocity variables, which is somewhat contrary to the idea of an input. How about controlled and uncontrolled variables? Clearly, we don't want x_i and v_i to be changed in arbitrary ways at any time, so they have to be declared controlled in the pendulum models. If we declare v_i as uncontrolled in the collision models, then changing their variables would not be allowed and the collision transitions would be blocked. We therefore must declare v_i controlled in both the pendulum and the collision models that use it. This violates the compatibility assumptions of compositional reasoning, but if we don't care about compositionality this is a perfectly good model.

A compositional solution requires the discrete updates of shared variables to be carried out with synchronized transitions. This can be achieved with a separate model for each velocity, in which we include the differential equation for v_i as well as the transition update for v_i only. This transition is synchronized with the collision model by using the same shared label, say a_{ij}. The v_i are controlled in the velocity models and uncontrolled everywhere else. Despite v_i being controlled elsewhere, the transition in the collision model is not forced to leave v_i unchanged, since it is no longer a local transition. Because of the synchronization it is irrelevant whether the updates to v_i are associated with the transition in the velocity model or in the collision model, the latter preserving the neatness of the original solution.

C. State and Algebraic Variables

The state of a system is a set of values for a minimal set of variables that suffices to predict the future evolution of the system from that state onward. With a system of differential equations or inclusions, the *state variables* are those whose derivative is defined or constrained. The other variables, defined or constrained by purely algebraic equations or inequalities, are called *algebraic*. From the above definition it is clear that a state variable can not be uncontrolled, since a variable that can change arbitrarily at any time can not possibly be essential to knowing the future. The algebraic variables by definition may take any value that satisfies the algebraic equations. Assuming the algebraic equations are part of the invariant, this corresponds to the notion of an uncontrolled variable.

Example II.4. Going back to the colliding pendulum example, Ex. II.3, we see that in all three model variants that were discussed, the variables are controlled in the component where its derivative is defined. However, the velocities v_i are algebraic in the collision component in Fig 3 (b), yet they need to be controlled in order to keep the transition from being blocked by the pendulum component. This is because they are modified by the discrete transition.

As the example indicates, algebraic variables should not automatically be considered uncontrolled. While we forgo any claim to generality, we conclude that state variables correspond to controlled variables (but not vice versa), and algebraic variables correspond to uncontrolled variables unless they are modified by discrete transitions.

III. Formalities of Sharing Variables: To I/O or not to I/O

The notion of controlled variables described in the previous section restricts what one is allowed to do with variables that are shared between different components. This raises the question whether this "compositional" modeling framework is handicapped with respect to more general, non-compositional variants. As we will see in this section, the answer is no: Every non-compositional model can be cast to a compositional model with a little extra effort, namely by explicitly modeling what is considered to happen to shared variables. To support this claim, we need a formal description of our models and their interaction. To model and analyze complex systems, it useful to divide the system into components, and model each component with an automaton. An automaton modeling the entire system is then obtained by combining these components with a so-called parallel composition operator. The complexity of an analysis of such a composed system increases exponentially with the number of components. In *compositional reasoning*, one tries to circumvent this problem by analyzing the components themselves, and deriving properties of the composed system from the results. A simple form of compositional reasoning is possible when any safety property of a component also holds when the component is composed with the rest of the system. If this is the case, the automata are called *compositional*

with respect to parallel composition. For this to be true, restrictions must be imposed on the way variables can change [10], i.e., restrictions on the automaton model as well as on the parallel composition operator.

To discuss the difference between compositional and non-compositional models, the next section defines hybrid automata without restrictions on how variables are shared, and hybrid input/output-automata that include such restrictions. In the section afterwards, we will compare these models using a notion of equivalence called bisimulation, using the colliding pendulums as a running example.

Related Work: Most hybrid automata models in literature do not include restrictions that enforce compositionality [11], or do not model shared variables in the composition [12], [13]. Compositional reasoning is possible with the model in [4], see, e.g., [14]. The formalism is relatively simple and intuitive, so we use it in two variations: without controlled variables and stutter transitions it is comparable to the one in [13]; with input-, controlled and output variables it provides a simple compositional model that we call *hybrid input/output-automaton*. More sophisticated compositional hybrid input/output-automata are proposed and studied in detail in [15].

A. Hybrid Automata and Semantics

Preliminaries: Given a set $X = \{x_1, \ldots, x_n\}$ of variables, a *valuation* is a function $v : X \to \mathbb{R}$. Let $V(X)$ denote the set of valuations over X. Let $\dot{X} = \{\dot{x}_1, \ldots, \dot{x}_n\}$ and $X' = \{x'_1, \ldots, x'_n\}$. The *projection* of v to variables $Y \subseteq X$ is $v\downarrow_Y = \{x \to v(x) | x \in Y\}$. The *embedding* of a set $U \subseteq V(X)$ into variables $\bar{X} \supseteq X$ is the largest subset of $V(Y)$ whose projection is in U, written as $U|^{\bar{X}}$. Given that a valuation u over X and a valuation v over Y *agree*, i.e., $u\downarrow_{X\cap\bar{X}} = v\downarrow_{X\cap\bar{X}}$, we use $u \sqcup v$ to denote the valuation w defined by $w\downarrow_X = u$ and $w\downarrow_{\bar{X}} = v$. An *activity* over X is a function $f : \mathbb{R}^{\geq 0} \to V(X)$. Let $Acts(X)$ denote the set of activities over X. The *derivative* $\dot{f}$ of an activity f is an activity over $\dot{X}$, defined analogously to the derivative in $\mathbb{R}^n$. The extension of operators from valuations to activities is done pointwise. Let $const_X(Y) = \{(v, v') | v, v' \in V(X), v\downarrow_Y = v'\downarrow_Y\}$.

We first define hybrid automata without restrictions on how variables are shared. For easier comparison with HIOA, we require a stutter transitions in every location. The stutter transitions may leave the variables constant, which is usually the case in a non-compositional model, but will be used later to model changes of the input variables.

Definition III.1 (Hybrid Automaton)**.** A *hybrid automaton* (HA) $A = (Loc, X, Lab, Edg, Flow, Inv, Init)$ consists of:

- A finite set Loc called *locations*.
- A finite set called *variables* X. A pair $p = (l, v)$ of a location and a valuation over X is a *state* of the automaton and the *state space* is $S_H = Loc \times V(X)$. For a state $p = (l, v)$ we define $loc(p) := l$ and $val(p) := v$. For a set of variables Y, let $val_Y(p) := v\downarrow_Y$.
- A finite set Lab of synchronization labels including the *stutter label* τ.
- A finite set Edg of edges called *transitions*. Each transition $e = (l, a, \mu, l')$ consists of a *source*, respectively *target* locations $l, l' \in Loc$, a synchronization label $a \in Lab$, and a *jump relation* $\mu \subseteq V(X)^2$. We require that for every location $l \in Loc$ there is a *stutter transition* $(l, \tau, \mu, l) \in Edg$ with some μ such that $const_X(X) \subseteq \mu$.
- A set $Flow \subseteq Loc \times V(X \cup \dot{X})$ called *flows*.
- A set $Inv \subseteq Loc \times V(X)$ called *invariant*.
- A set $Init \subseteq Inv$ called *initial states*.

In our compositional model a variable is either an *input*, and can therefore change arbitrarily at any time, or *controlled*. In parallel composition, controlled variables can not be changed independently by other automata in the composition. These elements are essential to compositionality [10]. A subset of the controlled variables are *output* variables, which, together with the input variables, define the externally visible behavior of the automaton. Note that the inputs may be restricted in their derivatives and can change arbitrarily as long as they satisfy the invariant. This allows us to model causal and noncausal coupling between variables. The resulting model is a slight variation of the one in [4]. [1]

Definition III.2 (Hybrid I/O-Automaton)**.** A *hybrid Input/-Output-automaton* (HIOA) $H = (A, C, O)$ consists of a hybrid automaton A, a subset C of the variables X that are called *controlled* variables, and a subset $O \subseteq C$ called *output* variables. Let $I = X \setminus C$ be the *input* variables and $E = I \cup O$ the *external* variables. We require that for every location $l \in Loc$ there is a *stutter transition* $(l, \tau, const_X(C), l) \in Edg$.

Semantics: We define the semantics of automata in terms of runs and traces, and will use them in the next section to define different notions of equivalence. A run defines the internal behavior of a hybrid automaton, a trace defines the externally observable behavior. An activity $f(t) \in Acts(X)$ is called *admissible* over an interval $[0, \delta]$ in a location l if $\delta = 0$, or $\forall t, 0 \leq t \leq \delta : f(t) \in Inv(l), f(t) \sqcup \dot{f}(t) \in Flow(l)$.

Definition III.3 (Run)**.** An *atomic run* $\sigma = (l, v) \xrightarrow{\delta, f, a} (l', v')$ consists of source and target states $(l, v), (l', v')$, a *delay* $\delta \in \mathbb{R}^{\geq 0}$, an activity f and a label $a \in Lab$ such that

- $v \in Inv(l)$, $v' \in Inv(l')$,
- f is admissible over $[0, \delta]$ in l and $f(0) = v$,
- there is a μ such that $(l, a, \mu, l') \in Edg, (f(\delta), v') \in \mu$.

A *run* of a hybrid automaton H is a finite or infinite sequence

$$\sigma = (l_0, v_0) \xrightarrow{\delta_0, f_0, a_0} (l_1, v_1) \xrightarrow{\delta_1, f_1, a_1} (l_2, v_2) \ldots$$

such that $\sigma_i = (l_i, v_i) \xrightarrow{\delta_i, f_i, a_i} (l_{i+1}, v_{i+1})$ is an atomic run for all $i \geq 0$.

[1] It differs in that we define a subset of the controlled variables as output variables, specify the activities via their derivatives, include a set of initial states, and the controlled variables are the same for all locations.

Definition III.4 (Trace)**.** A *trace* ρ over a set Y of variables, Lab of labels is a finite or infinite sequence $\rho = (\delta_0, f_0, a_0)(\delta_1, f_1, a_1)\ldots$ of delays $\delta_i \in \mathbb{R}^{\geq 0}$, activities $f_i \in Acts(Y)$ and labels $a_i \in Lab$. The trace of a run σ is its projection onto the activities of the external variables, i.e., the sequence $trace(\sigma) = (\delta_0, f_0{\downarrow_E}, a_0)(\delta_1, f_1{\downarrow_E}, a_1)\ldots$. A trace ρ is a trace of a hybrid automaton H if there exists some run σ such that $\rho = trace(\sigma)$.

B. Equivalence and Expressiveness

We wish to express that a hybrid automaton G is a valid abstraction of a hybrid automaton H (or equivalently that H refines G). To do so, we define a *simulation relation* over the product of their states. It relates a state in H to those in G that have the same, or more, behavior. To be consistent with our input/output framework, two HIOA can only be compared if they have comparable inputs and outputs. If G has more inputs than H it could be blocked by some K that does not block H, which violates compositionality.

Definition III.5. H is *comparable* with G if $Lab_H = Lab_G$, $I_H \supseteq I_G$ and $O_H = O_G$.

Definition III.6 (Trace Simulation)**.** Let H and G be either both hybrid automata or both HIOA. If they are hybrid automata we require that $X_H = X_G$. Let $E = E_G$ if they are HIOA, and $E = X_G$ otherwise. A relation $R \subseteq S_H \times S_G$ is a *trace simulation relation* between H and G iff for all $(p, q) \in R$, δ, f, a, p', an atomic run $p \xrightarrow{\delta,f,a}_H p'$ in H implies a matching atomic run in G, i.e., that there are g, q' with $q \xrightarrow{\delta,g,a}_G q' \wedge (p', q') \in R \wedge \forall t : f(t){\downarrow_E} = g(t){\downarrow_E}$. We write $H \preceq_t G$ iff there exists a trace simulation relation R such that $Init_H \subseteq R^{-1}(Init_G)$.

If $G \preceq H$ and $H \preceq G$ in a symmetric way, i.e., witnessed by the same simulation relation, G and H are, for all practical purposes, equivalent. This is referred to as *bisimulation*:

Definition III.7 (Bisimulation)**.** A simulation relation R is a *bisimulation relation* between H and G iff R is simulation relation for $H \preceq_t G$ and R^{-1} is a simulation relation for $G \preceq_t H$. Bisimulation equivalence is denoted with $\cong$.

To be able to compare hybrid automata and HIOA we transform the hybrid automata into a HIOA. To fulfill the restriction on τ-transitions in HIOA, we add such transitions.

Definition III.8 (Simulation between HA and HIOA)**.** Given a hybrid automaton A and a set of controlled variables C, let $A^\tau(C) = (Loc, X, Lab, Edg^\tau(C), Flow, Inv, Init)$, where $Edg^\tau(C) = Edg \cup \{(l, \tau, const_X(C), l) \mid l \in Loc\}$. A hybrid automaton A is *simulated* by a HIOA $H = (B, C_H, O_H)$ if $X_A = X_B$ and the HIOA $(A^\tau(C_H), C_H, O_H)$ simulates H, and vice versa.

If all variables are controlled, adding τ-transitions does not affect the behavior of the hybrid automaton, as the τ-transitions leave all variables unchanged. In the following composition operator, the jump relations of synchronized transitions result from the conjunction of the participating transitions. Independent transitions, i.e., those that do not synchronize, are allowed to change variables arbitrarily and the variables over which their jump relation is not defined are set to remain constant. This is non-compositional, and in principle similar to the composition operator in [11].

Definition III.9 (Composition of HA)**.** The *parallel composition* of hybrid automata A_1 and A_2 is the hybrid automaton $A = (Loc_1 \times Loc_2, X, Lab_1 \cup Lab_2, Edg, Flow, Inv, Init)$, written as $A = A_1||A_2$, such that $X = X_1 \cup X_2$ and

- $((l_1, l_2), a, \mu, (l_2', l_2')) \in Edg$ with $\mu = \{(v, v') \mid (v{\downarrow_{X_i}}, v'{\downarrow_{X_i}}) \in \mu_i\}$ iff for $i = 1, 2$,
  - $a \in Lab_i$ and $(l_i, a_i, \mu_i, l_i') \in Edg_i$, or
  - $a \notin Lab_i$, $l_i' = l_i$, and $\mu_i = const_{X_i}(Z_i)$, where $Z_1 = X_1 \setminus X_2$ and $Z_2 = X_2 \setminus X_1$;
- $Flow(l_1, l_2) = Flow_1(l_1)|^{X \cup \dot{X}} \cap Flow_2(l_2)|^{X \cup \dot{X}}$;
- $Inv(l_1, l_2) = Inv_1(l_1)|^X \cap Inv_2(l_2)|^X$;
- $Init(l_1, l_2) = Init_1(l_1)|^X \cap Init_2(l_2)|^X$.

Example III.1. Consider the hybrid automata P_i and C_i shown in Fig 3. The system of N colliding pendulums is modeled by the HA $P_1||\ldots P_N||C_1||\ldots||C_{N-1}$ if we add the constraint $x_i' = x_i \wedge x_{i+1}' = x_{i+1}$ to the transition of C_i. This is necessary, since otherwise the variables x_i and x_{i+1} can take any value in the composition. Note that the transition label τ and the stutter transitions are not shown.

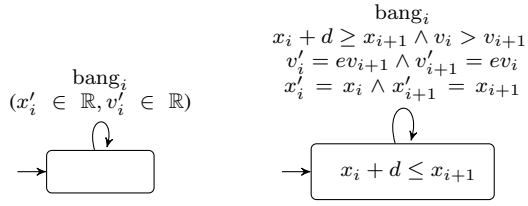
Definition III.10 (Composition of HIOA)**.** [4] The *parallel composition* of HIOA $H_1 = (A_1, C_1, O_1)$ and $H_2 = (A_2, C_2, O_2)$ is the HIOA $H = (A, C_1 \cup C_2, O_1 \cup O_2)$, written as $H = H_1||H_2$, where A is the parallel composition of A_1 and A_2 according to Def. III.9, but with $Z_1 = C_1 \setminus C_2$ and $Z_2 = (C_2 \setminus C_1)$. H_1 and H_2 are *compatible* if their sets of controlled variables C_1 and C_2 are disjoint.[2]

The point of HIOA is that they make simulation and bisimulation *compositional*, which means that any property established between two automata still holds if they are composed with another automaton

Proposition III.1. *Let H_1, H_2 be comparable HIOA with $H_1 \preceq_t H_2$. For any HIOA H_3 compatible with H_1,H_2 holds $H_1||H_3 \preceq_t H_2||H_3$. The same holds for bisimulation.*

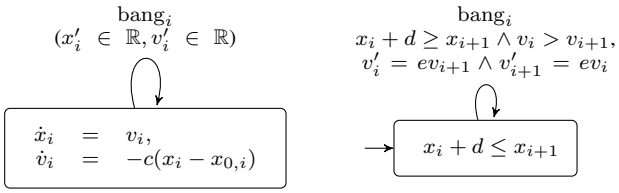
Example III.2. With the automata P_i and C_i shown in Fig 3 we define HIOA $H_{P_i} = (P_i, \{x_i, v_i\}, \{x_i, v_i\})$ and $H_{C_i} = (C_i, \{v_i, v_{i+1}\}, \{v_i, v_{i+1}\})$. Note that, in constrast to Ex.III.1, the transition of C_i can change x_i and x_{i+1} to arbitrary values. The system is modeled by the HIOA $H_{P_1}||\ldots H_{P_N}||H_{C_1}||\ldots||H_{C_{N-1}}$. Since x_i is controlled in H_{P_i}, the composition operator forces x_i to remain constant in the transition from C_i. The H_{P_i} and H_{C_i} are not compatible, so compositionality does not hold. The behavior of v_i in P_i alone is not a subset of that in $P_i||C_i$ since C_i adds jumps that are not in P_i. For instance, let the initial state of P_1 be

[2]We use controlled variables in a more relaxed fashion than in [4], [5], where $Z_1 = C_1$, $Z_2 = C_2$. If H_1 and H_2 are compatible, which is required for compositionality in any case, then C_1 and C_2 are disjoint and the two definitions are identical.



(a) Shared Variable Model S (b) Modified HA model $\bar{C}_i$

Fig. 4. HIOA transformation of the HA model of Ex. III.3 using a SVM



(a) P_i'' (b) C_i''

Fig. 5. A compositional HIOA model of Ex. II.3 where the Shared Variable Model is distributed over the P_i''

$x_1 = 0$ and $v_1 = 0$. Computing the reachable states of H_{P_1} will give the result that x_1 and v_1 are zero at all times. This is obviously not the case if any of the other pendulums collide with P_1. Since the model is not compositional, analyzing H_{P_1} in isolation may give results that do not hold for the whole system.

We now turn to the main claim of this section: that for every non-compositional model there exists an equivalent compositional model. To transform a composition of HA into HIOA we need an explicit model of what happens to shared variables. Recall that a variable can be controlled in at most one of the HIOA, since otherwise they would not be compatible. To circumvent this problem, we centralize control of the variables in a single place. We let the variables be uncontrolled in all of the HA, and add the following HIOA, in which they are controlled.

Definition III.11. Let the *shared variable model* (SVM) of a hybrid automaton A be the HIOA $S(A) = (A_S, X_S, X_S)$, with $A_S = (Loc_S, X_S, Lab_S, Edg_S, Flow_S, Inv_S, Init_S)$,

- $Loc_S = \{l\}$, $X_S = X_A$, $Lab_S = Lab_A$,
- $Edg_S = \{l, a, V(X_A \cup X_A'), l \mid a \in Lab_A \setminus \{\tau\}\} \cup \{(l, \tau, const_{X_S}(X_S), l)\}$, $Flow_S = V(X \cup \dot{X})$,
- $Inv_S = l \times V(X)$, $Init_S = l \times V(X)$.

To avoid that the SVM blocks stutter transitions of the HA, we require that the stutter transitions of the HA do not modify the variables. Should a HA have a stutter transition that does, one can simply replace every concerned transition the stutter label with a label that is not in the alphabet of any of the other HA. Intuitively, this enforces that any discrete change in the variables is synchronized with a transition in the SVM. In composition with the other automata, the SVM does not modify the behavior:

Proposition III.2. *Let A_1, A_2 be hybrid automata whose stutter transitions do not modify the variables, i.e., $(l_i, \tau, \mu_i, l_i) \in Edg_i \Rightarrow \mu_i = const_{X_i}(X_i)$. Let H_1, H_2 be the HIOA $H_1 = (A_1^\tau(\emptyset), \emptyset, \emptyset)$, $H_2 = (A_2^\tau(\emptyset), \emptyset, \emptyset)$. Then $A_1||A_2 \cong_t H_1||H_2||S(A_1||A_2)$.*

The proof is straightforward since any transition of $H_1||H_2$ synchronizes with a transition of the SVM such that the resulting jump relation is exactly that of $A_1||A_2$.

Example III.3. For an SVM transformation of the HA model of the colliding pendulums given in Ex. III.1, we replace the stutter label τ in the transition of C_i by a new label bang_i, obtaining the HA $\bar{C}_i$ and the SVM S shown in Fig.4. Note that S has one transition for every value of i. Let $H'_{P_i} = (P_i^\tau(\emptyset), \emptyset, \emptyset)$ and $H'_{C_i} = (\bar{C}_i^\tau(\emptyset), \emptyset, \emptyset)$. The system is modeled by the HIOA $H'_{P_1}||\dots H'_{P_N}||H'_{C_1}||\dots||H'_{C_{N-1}}||S$. To illustrate that the model is compositional, let the initial state of P_1 be $x_1 = 0$ and $v_1 = 0$. Contrary to Ex. III.1, computing the reachable states of H'_{P_1} does not give finite bounds on x_1 and v_1, since the stutter transition in H'_{P_1} can change them to arbitrary values.

We end this section with a pendulum model that is similar to the non-compositional HIOA model from Ex. III.1 but achieves compositionality by including the transitions and labels from the SVM of Ex. III.3.

Example III.4. Figure 5 shows a compositional variation of the HIOA model from Ex. III.1. The system is modeled by the HIOA $H''_{P_1}||\dots H''_{P_N}||H''_{C_1}||\dots||H''_{C_{N-1}}$, where $H''_{P_i} = (P_i'', \{x_i, v_i\}, \{x_i, v_i\})$ and $H''_{C_i} = (C_i'', \emptyset, \emptyset)$. Here, the Shared Variable Model is distributed over the H''_{P_i} by including the transitions with label bang_i in the H''_{P_i}.

IV. Semi-explicit DAEs in SpaceEx

As we have seen in the previous sections, models of control systems give rise to differential-algebraic equation systems (DAEs). We now present the way we handle these DAEs in SpaceEx. The support function-based reachability algorithms in SpaceEx can handle dynamics of the form

$$\dot{x} = Ax + u, \qquad x \in \mathcal{I}, u \in \mathcal{U}, \tag{2}$$

where $\mathcal{I}$ is the invariant of the corresponding location of the hybrid automaton. Both sets $\mathcal{I}$ and $\mathcal{U}$ are given by conjunctions of linear constraints (equalities and inequalities) on the respective variables. In the SpaceEx model format, the set $\mathcal{U}$ is joined with $\mathcal{I}$ and both together are given as the invariant. Syntactically, we have a model of the form

$$\dot{x} = \hat{A}y + \hat{b}, \qquad y \in \mathcal{Q}, \tag{3}$$

where y consists of variables of x and of u and $\mathcal{Q}$ is given by a conjunction of linear constraints. This model is brought to the form (2) in the following preprocessing:

1) Let Y be the set of all variables in the automaton.
2) Find variables X with derivatives constrained in (3).
3) Let $U = Y \setminus X$.

4) Split $\hat{A}$ into matrices A, B, i.e., $\begin{pmatrix} A & B \end{pmatrix}\begin{pmatrix} x \\ u \end{pmatrix} = \hat{A}y$.
5) Compute invariant and input set

$$\mathcal{I} = \left\{ x \;\middle|\; \exists u : \begin{pmatrix} x \\ u \end{pmatrix} \in \mathcal{Q} \right\}, \tag{4}$$

$$\mathcal{U} = \left\{ Bu + \hat{b} \;\middle|\; \exists x : \begin{pmatrix} x \\ u \end{pmatrix} \in \mathcal{Q} \right\}. \tag{5}$$

The latter step is problematic in that existential quantification over sets of linear inequalities is of exponential complexity. We can easily circumvent this problem thanks to the use of support functions:

5.a Instead of $\mathcal{I}$, use $\hat{\mathcal{I}} = \mathcal{Q}$. Our support function algorithm handles the extra variables in the computation with little overhead. It is based on computing the reachable values of x by computing maximizers in a set of template directions defined over X. Using $\hat{\mathcal{I}}$ instead of $\mathcal{I}$ simply means that these computations are taking place embedded in a higher dimensional space.

5.b Instead of $\mathcal{U}$, use

$$\hat{\mathcal{U}} = \left\{ \begin{pmatrix} 0 & B \end{pmatrix}\begin{pmatrix} x \\ u \end{pmatrix} + \hat{b} \;\middle|\; \begin{pmatrix} x \\ u \end{pmatrix} \in \mathcal{Q} \right\}. \tag{6}$$

In the algorithm, we use only the support of $\mathcal{U}$, i.e., we only compute maximizers of linear cost functions over the set. The embedding with variables x incurs only negligible overhead.

The above processing has a fundamental flaw: The set $\mathcal{U}$ is constant over time, so any dependencies of u on x are eliminated in step 5.

Example IV.1. The *proportional state feedback controller* is

$$u = -Kx. \tag{7}$$

Applying existential quantification in (4) results in the differential inclusion

$$\dot{x} = Ax + u, \qquad u \in -B\begin{pmatrix} K \\ 0 \end{pmatrix}\mathcal{Q},$$

which renders the model practically useless, since it corresponds to a model *without* feedback, with dynamics A instead of $A - BK$.

To take into account the algebraic equations in a reachability algorithm based on (2) we must, as much as possible, eliminate the dependecies of the algebraic variables on x. Such dependencies may be introduced inadvertently via the inequalities that define $\mathcal{Q}$. Our elimination process must therefore take the inequalities into account as well. This is illustrated by the following example.

Example IV.2. Consider the system

$$\begin{array}{rclll} \dot{x} &=& ax + u_1 - 10u_2, & -1 \le & u_1 \le 1, \\ 0 &=& x + u_1 + 20u_2, & -\alpha \le & x \le \alpha. \end{array} \tag{8}$$

If we ignore the inequalities, there is no way of telling whether it is better to eliminate u_1 or u_2. Eliminating u_1 in (8), we get $u_1 = -x - 20u_2$, and substituted into (8)

$$\dot{x} = (a-1)x - 30u_2, \quad -1 \le x + 20u_2 \le 1, |x| \le \alpha. \tag{9}$$

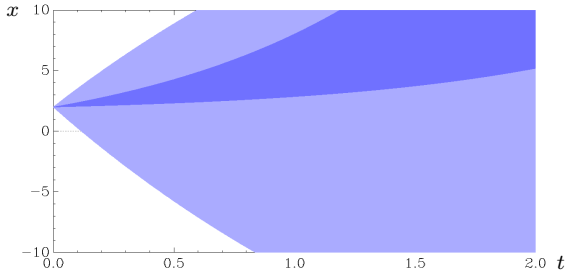


Fig. 6. Reachable states of Ex. IV.2 over time for two different choices of eliminated algebraic variables, (9) shown light and (11) shown dark

By eliminating u_1 we have introduced a dependency of u_2 on x in the inequalities. When using a reachability algorithm based on (2), via either (5) or (6), we obtain an input set $\mathcal{U}$ with values

$$-\frac{\alpha+1}{20} \le u_2 \le \frac{\alpha+1}{20}, \tag{10}$$

which is an overapproximation of the correct solution (9). Eliminating u_2 in (8), we get

$$u_2 = -\frac{1}{20}u_1 - \frac{1}{20}x,$$

which substituted into (8) yields

$$\dot{x} = (a + \tfrac{1}{2})x + \tfrac{3}{2}u_1, \qquad -1 \le u_1 \le 1, |x| \le \alpha. \tag{11}$$

Here, u_1 is independent of x, so a reachability algorithm based on (2) can compute the exact solution.

The reachable states computed by SpaceEx for both solutions are shown in Fig. 6 for parameters $a = \frac{1}{2}$, $\alpha = 10$, and with initial condition $x(0) = 2$. As expected, (11) is much more precise, even though it has unstable dynamics, while (9) is stable.

To obtain a canonical form, we rewrite inequalities in $\mathcal{Q}$ as equalities by introducing slack variables, plus we consider that there might be auxiliary variables in $\mathcal{Q}$ that do not occur in the ODE (even possibly arising from the specification). We rewrite the system equations (3) as follows, where u refers to non-state variables that influence $\dot{x}$, and v to all other non-state variables:

$$\dot{x} = Ax + Bu + b, \tag{12}$$

$$0 = Cx + Du + Ev + d, \tag{13}$$

$$0 \le Fv. \tag{14}$$

Our goal is to eliminate as much as possible the influence of x in the second equation of (12). We distinguish three cases:

a) Case 1: If D is invertible, we can eliminate u:

$$\dot{x} = (A - BD^{-1}C)x + (b - BD^{-1}d) - BD^{-1}Ev. \tag{15}$$

This may leave us with a dependency of x on v, but it is not state-dependent and directly fits the form (2) with

$$\dot{x} = (A - BD^{-1}C)x + u^*, \tag{16}$$

$$u^* \in \left\{ (b - BD^{-1}d) - BD^{-1}Ev \;\middle|\; Fv \ge 0 \right\}. \tag{17}$$

The proportional feedback controller (7) falls under case 1.

b) Case 2: If $\begin{pmatrix} D & E \end{pmatrix}$ is invertible, we eliminate u,v:

$$\dot{x} = \left(A - \begin{pmatrix} B & 0 \end{pmatrix}\begin{pmatrix} D & E \end{pmatrix}^{-1} C\right)x + b - \begin{pmatrix} B & 0 \end{pmatrix}\begin{pmatrix} D & E \end{pmatrix}^{-1} d, \quad (18)$$

$$0 \leq -\begin{pmatrix} 0 & F \end{pmatrix}\begin{pmatrix} D & E \end{pmatrix}^{-1}(Cx + d). \quad (19)$$

While here we may end up with inequalities over x, they do not involve u and v, so that they can be included in (2) as part of the invariant $\mathcal{I}$. Note that case 2 is distinct from case 1 if there are variables v, since D and $\begin{pmatrix} D & E \end{pmatrix}$ can not both be invertible at the same time.

c) Case 3: In the general case, we attempt to get rid of as many elements in C as possible. Putting u and v together in a single vector w, we start instead of (12)–(14) with

$$\begin{aligned} \dot{x} &= Ax + Bw + b, && (20)\\ 0 &= Dw + Cx + d, && (21)\\ 0 &\leq Fw. && (22) \end{aligned}$$

We apply the Gauss-Joran algorithm to bring (21) into reduced row-echelon form, turning as many columns of D as possible into unit vectors (with only one nonzero element). Then we split w into variables w' with a unit column vector in the transformed (21) and the remaining, denoted as w''. Splitting also the matrices accordingly, we get

$$\begin{aligned} \dot{x} &= Ax + B'w' + B''w'' + b,\\ 0 &= Iw' + D'w'' + C'x + d',\\ 0 &= C''x + d''.\\ 0 &\leq F'w' + F''w''. \end{aligned} \quad (23)$$

Eliminating w' we get

$$\begin{aligned} \dot{x} &= (A - B'C')x + (B'' - B'D')w'' + b - B'd',\\ 0 &= C''x + d''.\\ 0 &\leq -F'C'x + (F'' - F'D')w'' - F'd'. \end{aligned} \quad (24)$$

Example IV.3. The system from Ex. IV.2 falls neither under case 1 nor case 2. We bring the equations to canonical form, introducing slack variables $e_1, \ldots, e_4$ for the inequalities:

$$\begin{array}{lllllllll} \dot{x} = & ax & +u_1 & -10u_2, & & & & & \\ 0 = & 1u_1 & +20u_2 & & & & +1x, & & \\ 0 = & 1u_1 & & +1e_1 & & & & -1, & \\ 0 = & -1u_1 & & & +1e_2 & & & -1, & \\ 0 = & & & & & +1e_3 & +1x & -\alpha, & \\ 0 = & & & & & & +1e_4 & -1x & -\alpha, \\ 0 \leq e_1, & & 0 \leq e_2, & & 0 \leq e_3, & & 0, \leq e_4. & & \end{array} \quad (25)$$

We choose $w' = \begin{pmatrix} u_1 & u_2 & e_1 & e_3 & e_4 \end{pmatrix}^T$, which leaves $w'' = \begin{pmatrix} e_2 \end{pmatrix}$. Splitting the matrices accordingly we get

$$\begin{array}{lll} \dot{x} = & (a + \frac{1}{2})x + \frac{3}{2}e_2 - \frac{3}{2}, & \\ 0 \leq & -1e_2 & +2 \\ 0 \leq & -1x & +\alpha \\ 0 \leq & 1x & +\alpha \\ 0 \leq & 1e_2 & 0 \end{array}$$

This solution is equivalent to (9), which can be easily seen after observing that the third line of (25) reads $e_2 = u_1 + 1$. We obtain the same solution starting with any order of u_1 and u_2 in (25).

REFERENCES

[1] G. Frehse, C. L. Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "SpaceEx: Scalable verification of hybrid systems," in *CAV*, ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 379–395.

[2] O. L. Scott Cotton, Goran Frehse, "The SpaceEx modeling language," http://spaceex.imag.fr/, 2010.

[3] MATLAB version 7.5, Natick, Massachusetts: The MathWorks Inc., 2007. [Online]. Available: http://www.mathworks.com/

[4] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of hybrid systems," *Theor. Comp. Science*, vol. 138, no. 1, pp. 3–34, 1995.

[5] G. Frehse, "Compositional verification of hybrid systems using simulation relations," Ph.D. dissertation, Radboud Universiteit Nijmegen, Oct. 2005.

[6] D. E. N. Agut, D. A. van Beek, and J. E. Rooda, "Syntax and semantics of the compositional interchange format for hybrid systems," *J. Log. Algebr. Program.*, vol. 82, no. 1, pp. 1–52, 2013.

[7] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, "Hytech: A model checker for hybrid systems," in *CAV*, ser. LNCS, O. Grumberg, Ed., vol. 1254. Springer, 1997, pp. 460–463.

[8] P. Fritzson, *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Computer Society Press, 2003.

[9] P. I. Barton and C. C. Pantelides, "gPROMS - a combined discrete/-continuous modelling environment for chemical processing systems," *Simulation Series*, vol. 25, no. 3, pp. 25–34, 1993.

[10] N. A. Lynch and M. J. Fischer, "On describing the behavior and implementation of distributed systems," *Theoretical Computer Science*, vol. 13, no. 1, pp. 17–43, 1981.

[11] R. Alur, T. A. Henzinger, and P.-H. Ho, "Automatic symbolic verification of embedded systems," *IEEE Trans. Soft. Engineering*, vol. 22, pp. 181–201, 1996.

[12] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho, "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems," in *Hybrid Systems*, ser. LNCS, R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, Eds., vol. 736. Springer, 1993, pp. 209–229.

[13] T. A. Henzinger, "The theory of hybrid automata," in *LICS'96*, 1996.

[14] E. Ábrahám-Mumm, U. Hannemann, and M. Steffen, "Verification of hybrid systems: Formalization and proof rules in pvs," in *ICECCS'01*, June 2001.

[15] N. A. Lynch, R. Segala, and F. W. Vaandrager, "Hybrid I/O automata," *Information and Computation*, vol. 185, no. 1, 2003.