April 2004

Compositional abstractions of hybrid control systems

Paulo Tabuada
University of Pennsylvania

George J. Pappas
University of Pennsylvania, pappasg@seas.upenn.edu

Pedro Lima
Instituto Superior Técnico

Postprint version. Published in *Formal Methods in System Design*, Volume 14, Issue 2, April 2004, pages 203-238.
The original publication is available at www.springerlink.com.
Publisher URL: http://dx.doi.org/10.1023/B:DISC.0000018571.14789.24

Compositional abstractions of hybrid control systems

Abstract

Abstraction is a natural way to hierarchically decompse the analysis and design of hybrid systems. Given a hybrid control system and some desired properties, one extracts and abstracted system while preserving the properties of interest. Abstractions of purely discrete systems is a mature area, whereas abstractions of continuous systems is a recent activity. In this paper we present a framework for abstraction that applies to discrete, continuous, and hybrid syustems. We introduce a composition operator that allows to build complex hybrid systems from simpler ones and show compatibility between abstractions and this compositional operator. Besides unifying the existing methodologies we also propose constructions to obtain abstractions of hybrid control systems.

Comments

Postprint version. Published in *Formal Methods in System Design*, Volume 14, Issue 2, April 2004, pages 203-238. The original publication is available at www.springerlink.com.
Publisher URL: http://dx.doi.org/10.1023/B:DISC.0000018571.14789.24

COMPOSITIONAL ABSTRACTIONS OF HYBRID CONTROL SYSTEMS

PAULO TABUADA, GEORGE J. PAPPAS, AND PEDRO LIMA

Abstract. Abstraction is a natural way to hierarchically decompose the analysis and design of hybrid systems. Given a hybrid control system and some desired properties, one extracts an abstracted system while preserving the properties of interest. Abstractions of purely discrete systems is a mature area, whereas abstractions of continuous systems is a recent activity. In this paper we present a framework for abstraction that applies to discrete, continuous, and hybrid systems. We introduce a composition operator that allows to build complex hybrid systems from simpler ones and show compatibility between abstractions and this compositional operator. Besides unifying the existing methodologies we also propose constructions to obtain abstractions of hybrid control systems.

1. Introduction

In the last decade, increasing attention has been paid to the modeling, analysis and control of large-scale, multi-agent, complex, hybrid systems. The impulse from the applications side has been tremendous and includes among others: automotive engines [7, 6], air-traffic management [45], chemical batch plants [29], manufacturing systems [14], TCP congestion control [16] and biomolecular networks [1] among many others.

One approach to deal with the inherent complexity of hybrid control systems is to organize them in a hierarchical framework where different layers of abstraction represent different aspects of the same system. Analysis tasks are performed on simpler, abstracted models that are equivalent with respect to the relevant properties. Synthesis tasks also benefit from this approach since the design starts as the top of the hierarchy on a simple model and is then successively refined by incorporating the modeling details of each layer of abstraction.

The notion of abstraction is quite mature in theoretical computer science, and, in particular, in the areas of concurrency theory [27] [46], and computer aided verification [26]. This has resulted in formal and very meaningful notions of abstraction which are used to tackle exponential explosion of purely discrete systems. Given a discrete system, an *abstraction* is simply a quotient system that preserves some properties of interest while ignoring detail. Language equivalence, simulation, and bisimulation are established notions of abstraction for discrete systems that preserve properties expressed in various temporal logics [25].

For purely continuous systems, the notions of simulation, and bisimulation had no counterparts. Recently, similar notions were introduced in the collection of papers [32, 31, 33, 30, 42, 43]. This research resulted in automatic constructions of abstractions for linear control systems [31], while characterizing abstracting maps that preserve properties of interest such as controllability. Such notions were furthermore generalized for nonlinear control affine systems [33] and fully nonlinear systems [43]. Notions of bisimulation for purely continuous control systems were introduced in [30] where linear control systems are embedded in the class of transition systems for which the notion of bisimulation was originally introduced in [34] and also [27]. It is shown in [30] that different embeddings give rise to semantically different notions of bisimulation being characterized by different conditions. For nonlinear systems, bisimulation was introduced in [42] and it was shown that under certain conditions the abstractions described in [33] are in fact bisimilar to the original system.

This research was partially supported by Fundação para a Ciência e Tecnologia under grant PRAXIS XXI/BD/18149/98, and the National Science Foundation Information Technology Research grant CCR01-21431.

The notion of bisimulation has also proved useful in the context of control of discrete event systems. In [9] the relation between bisimulation, supervisory control of discrete event systems and model matching problems is clarified. Furthermore, it is shown how recasting supervisory control problems for discrete event systems as a bisimulation problem leads to more efficient algorithms. In [24], bisimulation is fundamentally used at the level of the problem formulation by requiring the closed loop system to simulate or bisimulate the specification.

Based on these results, in [41], we took the first steps towards constructing abstractions of hybrid systems while preserving timed languages. Even though *only* the continuous part of the system was abstracted, the important property that needed to be preserved in this abstraction was the detectability of the discrete switching conditions. Related but orthogonal work considers purely discrete abstractions of hybrid systems [2, 10, 12, 36].

The similarities between notions of abstraction for discrete, continuous, and hybrid systems immediately raise the question of a more unified theory of abstraction. In this paper, we begin addressing this important issue. We start by first considering a more unified and abstract model for control systems. Our abstract control systems model is inspired by categorical definitions of systems that are as old as [4, 37] and as recent as [39]. Although categorically inspired, the paper is accessible to readers that are not familiar with category theory, except for some proofs that rely on simple category theory notions.

We show that purely discrete, continuous, and hybrid systems can be easily captured by our abstract model. Furthermore, at this level of abstraction, one can show many useful properties regarding abstraction or composition that are independent of the discrete, continuous, or hybrid structure of the system.

As abstraction clearly depends on the property to be preserved, in this paper, we focus on *simulations* and *bisimulations* where the properties that are preserved are the trajectories of the original system. In other words, given an original hybrid control system and an abstracting map, which performs state aggregation, we would like to extract another hybrid control system which *simulates* all trajectories of the original system. This is clearly useful for verification purposes since in order to determine if a system satisfies certain properties it is *sufficient* to check if its abstraction verifies the desired properties. However in many situations one needs lower complexity models that are both *sufficient* and *necessary*. This motivates the need for *bisimulations* which are symmetric simulation relations, that is, each system simulates the other.

We also introduce an abstract operator that allows to built systems by interconnection of subsystems. This *compositional* operator, based on the categorical ideas in [46], turns out to be compatible with simulations and even with bisimulations in certain cases. Compatibility means that instead on computing an abstraction of a complex large-scale system one can compute abstractions of the subsystems and is guaranteed that the interconnection of those abstractions is an abstraction of the original large-scale system. Our composition operator differs from the approaches described in [23, 13], in that we model synchronization by restricting the behavior of the systems without a priori defining inputs and outputs.

We specialize the developed results for hybrid systems, presenting a construction to obtain abstractions of hybrid control systems. Furthermore, we also provide sufficient conditions for the resulting abstraction to be a bisimulation of the original system. These results are then illustrated in a concrete application. We consider the hybrid model of a spark ignition engine described in [8] and show how our methods can be applied to obtain a smaller abstract model.

The structure of this paper is as follows:

In Section 2, an abstract notion of control systems which captures discrete, continuous, and hybrid control systems is introduced as well as a notion of abstraction and bisimulation. It is also shown how these notions can be used for verification of reachability based properties. Compositionality is discussed in Section 3 by introducing the notion of parallel composition with synchronization and showing how abstractions and bisimulations preserve this composition operator. In Section 4 we specialize these results to hybrid control systems, and present a construction to obtain abstractions of hybrid control systems that simulate the trajectories of

the original system. The proposed methodology is illustrated with a spark ignition engine example at Section 5 and at Section 6 we list many interesting issues for future research. To keep the presentation of ideas fluid, we have collected some mathematical facts regarding partial maps and monoids in Appendix A, while in Appendix B we introduce the categorical notions of product and equalizer used in some of the proofs.

2. Abstract Control Systems

In this section we seek to extract the essential features common to purely discrete and continuous systems that will allow to develop a fruitful theory of abstractions for hybrid systems. This approach has the clear advantage of presenting in a unified way notions, results and algorithms common to discrete, continuous and hybrid control systems.

We start by recalling some know interpretations of continuous and discrete control systems to gain some motivation for the general definitions.

2.1. Discrete Control Systems.

One of the usual models for discrete control systems are finite state automata [11, 20], defined by a triple (Q, Σ, δ) where:

- Q is a finite set of states,
- Σ is a finite set of input symbols,
- $\delta : Q \times \Sigma \longrightarrow Q$ is the next-state function.

We regard the partially defined map δ as defining the controlled dynamics, in the sense that for each $q \in Q$ there exists a set of choices (the elements $\sigma \in \Sigma$ such that $\delta(q, \sigma)$ is defined) that will influence the evolution of the state. This controlled evolution is also usually modeled as a transition relation $\longrightarrow \in Q \times \Sigma \times Q$, but we will restrict our attention to deterministic[1] systems for which the relation $\longrightarrow$ can be represented as a (partial) function δ.

We now look at finite state automata from a different but equivalent perspective. Let us denote by Σ^* the set of all finite strings obtained by concatenating elements in Σ. In particular, the empty string ε also belongs to Σ^*. Regarding concatenation of strings as a map from $\Sigma^* \times \Sigma^*$ to Σ^*, we can give Σ^* the structure of a monoid. Concatenation of strings is clearly an associative operation and the empty string ε can be taken as the monoid identity since it satisfies $s \cdot \varepsilon = \varepsilon \cdot s = s$ for any $s \in \Sigma^*$. We now recall from basic automata theory [17] that the transition function δ defines a *unique* partial map from $Q \times \Sigma^*$ to Q with the following properties:

$$\begin{aligned} \delta^*(q, \varepsilon) &= q \\ \delta^*(q, \sigma_1\sigma_2) &= \delta^*(\delta^*(q, \sigma_1), \sigma_2) \end{aligned}$$

These properties are in fact the definition of a monoid action, that is, δ^* is a (right) partially defined action of the monoid Σ^* on the set Q.

To clarify the similarities to the continuous case that we describe next, we elaborate a little on the structure of the monoid Σ^*. This monoid has been defined as the set of all finite sequences of elements in Σ. Alternatively we can regard Σ^* as the disjoint union of all maps Σ^n, represented by:

$$\Sigma^* = \coprod_{n \in \mathbb{N}_0} \Sigma^n$$

[1] Nondeterministic transition relations can also be captured by parameterizing the nondeterminism. This can be accomplished by modeling Σ as $\Sigma = \Sigma_c \times \Sigma_u$, where the labels in Σ_c are regarded as controllable inputs, while the labels in Σ_u are regarded as uncontrollable inputs or disturbances. This allows to model nondeterminism since $\delta(q, (\sigma_c, \sigma_u))$ is a set of states parameterized by the labels $\sigma_u \in \Sigma_u$, which are independent of the choice σ_c.

Each set Σ^n, in the previous disjoint union, can be identified with the set of all maps from a set with n elements to Σ. In fact, choosing $\{1,2,3,\ldots,n\}$ as the set with n elements, we see that $(\sigma_1,\sigma_2,\sigma_3,\ldots,\sigma_n)\in\Sigma^n$ is simply the map $u:\{1,2,3,\ldots,n\}\longrightarrow\Sigma$ defined by $u(i)=\sigma_i$ for $i=1,2,3,\ldots,n$. The empty string ε is identified with the (unique) map from the empty set to Σ. Concatenation of strings can now be seen as concatenation of maps defined as follows:

$$\begin{array}{rcl} \cdot:\Sigma^n\times\Sigma^m & \to & \Sigma^{n+m} \\ (u(i),v(i)) & \mapsto & (u\cdot v)(i)=\left\{\begin{array}{ll} u(i) & \text{if}\quad 1\leq i\leq n \\ v(i-n) & \text{if}\quad n+1\leq i\leq n+m \end{array}\right. \end{array} \tag{2.1}$$

The above operation is well defined only if the number n is finite, otherwise we could not append the second map after the end of the first. Furthermore, as the concatenation $u\cdot v\in\Sigma^{n+m}$ must belong to Σ^* we need also to ensure that $n+m$ is finite which implies that m must also be finite. This shows that we are forced to work with finite length strings which will not be the case for continuous systems as we will see shortly.

2.2. **Continuous Control Systems.** For simplicity of presentation, we consider only time-invariant control systems, although the construction to be presented is generalizable to time varying systems. We assume also that the control systems satisfy the usual conditions for existence and uniqueness of solutions [40]. Consider a continuous control system, described by the triple (M,U,f), where:

- M is a smooth manifold modeling the state space,
- U is a smooth manifold modeling the input space,
- $f:M\times U\longrightarrow TM$ is a smooth map assigning for each $u\in U$ the vector field $f(-,u):M\longrightarrow TM$.

Similarly to the discrete case, continuous control systems can also be understood by means of a monoid action. To reveal this fact, we define the set U^t as the set of all maps[2] from the interval $[0,t[$ to the space of inputs U:

$$U^t=U^{[0,t[}\quad [0,t[\in\mathbb{R}_0^+ \tag{2.2}$$

An element of U^t is denoted by u^t, and represents a map from $[0,t[$ to U. Consider now the set U^* which is the disjoint union of all U^t for $t\in\mathbb{R}_0^+$:

$$U^*=\coprod_{t\in\mathbb{R}_0^+}U^t \tag{2.3}$$

The set U^* can also be regarded as a monoid under the operation of concatenation, that is, if $u^{t_1}\in U^{t_1}\subset U^*$ and $v^{t_2}\in U^{t_2}\subset U^*$ then $u^{t_1}v^{t_2}=w^{t_1+t_2}\in U^{t_1+t_2}\subset U^*$ with concatenation given by:

$$u^{t_1}v^{t_2}(t)=\left\{\begin{array}{ll} u^{t_1}(t) & \text{if}\quad 0\leq t<t_1 \\ v^{t_2}(t-t_1) & \text{if}\quad t_1\leq t<t_1+t_2 \end{array}\right. \tag{2.4}$$

The identity element is given by the empty input, that is $\varepsilon=u^0$. This construction is analogous to the construction that obtains Σ^* from Σ, however the fact that t_1 and t_2 are (finite) real numbers does not imply that u^{t_1} is a finite concatenation of elements in U^*. We can have an infinite number of concatenations as long as the sum of the duration times converges. This should be contrasted with the finite case, where a finite number of concatenations is required.

Choosing an admissible input trajectory u^t (an element of U^*), $f(x,u^t)$ is a well defined vector field and as such, it induces a flow which we denote by $\gamma_x:[0,t[\to M$, satisfying $\gamma_x(0)=x$. We can then regard any smooth control system as a monoid action by defining:

$$\begin{array}{rcl} \Phi:M\times U^* & \to & M \\ (x,u^t) & \mapsto & \gamma_x(t) \end{array} \tag{2.5}$$

[2]Technically speaking, we allow only classes of maps $u(t)$ for which the solution of $f(x(t),u(t))$ is well defined. However, our results are independent of the chosen class.

It is not difficult to see that Φ is in fact a well defined monoid action since

$$\Phi(x, \varepsilon) = \gamma_x(0) = x$$

and

$$\Phi(x, u^{t_1}u^{t_2}) = \gamma_x(t_1 + t_2) = \gamma_{\gamma_x(t_1)}(t_2) = \Phi(\Phi(x, u^{t_1}), u^{t_2})$$

It is interesting to note that when U is a singleton (there are no choices to be made) the set U^t can be identified[3] with the number t so that U^* is given by $U^* = \coprod_{t\in\mathbb{R}_0^+} t = \mathbb{R}_0^+$ and our control system Φ degenerates into an action of $\mathbb{R}_0^+$ on M, that is, the solution of a differential equation (a degenerate control system).

2.3. **Abstract Control Systems.** Motivated by the previous examples, we are lead to consider monoid actions as good candidates for an abstract model of control systems. This is quite classic in the discrete case and has also been explored in [40] for the continuous case. Recently, similar ideas have been used to lift several results from regular expressions to timed expressions [5]. The fact that the same characterization also captures the hybrid case, is perhaps surprising, but motivates the need to formalize the discussion so far.

Definition 2.1 (Abstract Control System)**.** An abstract control system is a triple $(S, \mathcal{M}, \Phi)$, where S is a set, $\mathcal{M}$ is a monoid and Φ is a (possibly partially defined) action of the monoid $\mathcal{M}$ on the set S, that is, a map $\Phi : S \times \mathcal{M} \to S$ satisfying:

1. **Identity:** $\Phi(s, \varepsilon) = s$
2. **Semi-group:** $\Phi(s, m_1m_2) = \Phi(\Phi(s, m_1), m_2)$

We will usually denote an abstract control system simply by Φ or Φ_S if we wish to emphasize the set S. We also represent by $s \xrightarrow{m} s'$ the evolution from s to s' controlled by m and described by Φ, that is, $\Phi(s, m) = s'$.

We are now ready to see how the present formalism can also describe hybrid control systems.

2.4. **Hybrid Control Systems as Abstract Control Systems.** Hybrid control systems also fit in the previous abstract framework. The state space of an hybrid control system is usually described as $Q \times X$, where Q is a finite set of states and X a smooth manifold. However it will be convenient to relax this concept, and consider a set of smooth manifolds X_q parameterized by the discrete states, denoted by $X = \{X_q\}_{q\in Q}$ as the state space. This is natural, as different discrete states may be associated with different continuous control systems defined on different continuous state spaces. A point in X is represented by the pair (q, x), where $x \in X_q$.

As monoid we will use the set:

$$\mathcal{M} = \coprod_{n\in\mathbb{N}} (U^* \cup \Sigma^*)^n \tag{2.6}$$

assuming that $U^* \cap \Sigma^* = \{\varepsilon\}$ and regarding U^* and Σ^* simply as sets. Let us elaborate on the product operation on $\mathcal{M}$. This operation is defined as the usual concatenation and therefore it requires finite length strings. To accommodate this requirement and still be able to have an infinite number of concatenations of elements in U^* we proceed as follows. Suppose that we want to show that $\sigma_1 u^{t_1} u^{t_2} \dots u^{t_n} \dots \sigma_2$ belongs to $\mathcal{M}$, where t_n is a convergent sequence. Instead of regarding each element in the string as an element in $\mathcal{M}$, which would not allow us to define the last concatenation since it would happen after ∞, we regard σ_1 and σ_2 as elements of $\mathcal{M}$ and $u^{t_1} u^{t_2} \dots u^{t_n} \dots = u^{t'}$ as an element of U^* and consequently as an element of $\mathcal{M}$, where $t' = \sum_{n=1}^{\infty} t_n$. This string is then regarded as the map $u : \{1, 2, 3\} \to \mathcal{M}$ defined by $u(1) = \sigma_1$, $u(2) = u^{t'}$ and $u(3) = \sigma_3$. The product in $\mathcal{M}$ is then the usual concatenation on reduced strings, that is, strings where all consequent sequences of elements of U^* or Σ^* have been replaced by their product in U^* or Σ^*, respectively. The monoid $\mathcal{M}$ obtained by this construction is called the free product of U^* and Σ^* and is

[3] There exists only one function from $[0, t[$ to a singleton, the constant map.

in fact the coproduct[4] in the category of monoids [18]. Furthermore, we have the following characterization of $\mathcal{M}$ that will be useful in the next sections:

Proposition 2.2 ([18]). *The monoid $\mathcal{M}$ is freely generated by the symbols $U^* \cup \Sigma^*$.*

Hybrid control systems are now cast into the abstract control systems framework as:

Definition 2.3 (Hybrid Control Systems). A hybrid control system $H = (X, \mathcal{M}, \Phi)$ consists of:

- The state space $X = \{X_q\}_{q \in Q}$,
- A monoid $\mathcal{M} = \coprod_{n \in \mathbb{N}} (U^* \cup \Sigma^*)^n$,
- A partial action Φ of $\mathcal{M}$ on X such that:
  - **Existence of invariants:** For all $q \in Q$, there exists a set $Inv(q) \subseteq X_q$
  - **Invariance of invariants:** For all $x \in Inv(q)$ there exists at least one $u^t \in U^*$ such that $\Phi((q,x), u^t)$ is defined and satisfies $\Phi((q,x), u^{t'}) \in Inv(q)$ for every prefix $u^{t'}$ of u^t.

The semantics associated with the evolution from (q,x) governed by Φ and controlled by $m \in \mathcal{M}(q,x)$ is the standard transition semantics of hybrid systems [15]. Suppose that $m = u^{t_1}\sigma_1\sigma_2 u^{t_2}$, then $(q,x) \xrightarrow{m} (q',x')$ means that the system starting at (q,x) evolves during t_1 units of time under continuous input u^{t_1}, jumps under discrete input σ_1 and them jumps again under σ_2. After the two consecutive jumps, the system evolves under the continuous control input u^{t_2} reaching (q',x'), t_2 units of time after the last jump. From the hybrid system construction we can clearly extract the purely discrete case presented in Section 2.1 when X_q is a singleton and $U_q = \varnothing$ for each $q \in Q$. The purely continuous case presented in Section 2.2 is also recovered when Q is a singleton and $\Sigma = \varnothing$. This shows that the above model seems to provide the right generalization from the discrete and continuous models.

2.5. **Control System Abstractions.** Having defined the structure of control systems and hybrid control systems in particular, we now consider relationships between abstract control systems that preserve their structure and can therefore be seen as abstract control systems homomorphisms. We shall call such maps, simulation maps, for reasons to be discussed shortly.

Definition 2.4 (Simulations of Abstract Control Systems). Let Φ_X and Φ_Y be two abstract control systems over X and Y with monoids $\mathcal{M}_X$ and $\mathcal{M}_Y$, respectively. A pair of maps (ϕ, φ) is said to be a simulation from Φ_X to Φ_Y when:

- $\phi : X \to Y$ is a total map,
- $\varphi : X \times \mathcal{M}_X \to \mathcal{M}_Y$ is a partial map defined on $\Phi_X^{-1}(X)$ satisfying for every $x \in X$:

$$\varphi(x, \varepsilon) = \varepsilon \tag{2.7}$$

$$\varphi(x, m_1 m_2) = \varphi(x, m_1)\varphi(\Phi_X(x, m_1), m_2) \tag{2.8}$$

- the maps ϕ and φ satisfy $(\phi, \varphi)(X, \mathcal{M}_X) \subseteq \Phi_Y^{-1}(Y)$ and relate Φ_X to Φ_Y as expressed in the following commutative diagram:

$$\begin{array}{ccc} Y \times \mathcal{M}_Y & \xrightarrow{\Phi_Y} & Y \\ {\scriptstyle (\phi,\varphi)}\uparrow & & \uparrow{\scriptstyle \phi} \\ X \times \mathcal{M}_X & \xrightarrow[\Phi_X]{} & X \end{array} \tag{2.9}$$

  or equivalently $\phi \circ \Phi_X(x, m) = \Phi_Y(\phi(x), \varphi(x, m))$.

[4] See, for example, Appendix A for a definition of coproduct.

When (ϕ, φ) is a simulation from Φ_X to Φ_Y we also say that Φ_Y is a simulation of Φ_X since for every evolution of Φ_X, the map ϕ transforms that evolution into an evolution of Φ_Y. It is in this sense, that Φ_Y simulates Φ_X. The map ϕ (when it is not injective) is to be understood as a state aggregation map, specifying which state information is propagated from the original system Φ_X to Φ_Y. Similarly, the map φ transforms the inputs of the original system Φ_X to the inputs of system Φ_Y.

This definition of simulation slightly generalizes the usual notions of morphisms between transition systems as described in [46]. Instead of considering maps $\phi : X \longrightarrow Y$ and $\varphi : \mathcal{M}_X \longrightarrow \mathcal{M}_Y$, we allow φ to depend also on X. This is necessary in order to correctly describe the relation between the input spaces of continuous control systems and its abstractions as discussed in [43]. Nevertheless, abstract control systems and simulation maps still define a category, where composition of morphisms $(\phi_1(x_1), \varphi_1(x_1, m_1))$ and $(\phi_2(x_2), \varphi_2(x_2, m_2))$ is given by $(\phi_2 \circ \phi_1(x_1), \varphi_2(\phi_1(x_1), \varphi_1(x_1, m_1)))$ and identity morphisms are simply the identity maps.

The conditions expressed in Definition 2.4 may seem difficult to check in concrete examples. However, we shall take a constructive approach by introducing a construction that builds the map φ and the system Φ_Y from a given system Φ_X and map ϕ. Furthermore, φ and Φ_Y will satisfy all the conditions of Definition 2.4 by construction, thereby overcoming the necessity of determining if they are indeed satisfied.

It is within this category that we shall develop our study of abstractions, considering any simulation of a system as an abstraction of that system. We introduce also the celebrated notion of bisimulation [34, 27], a special simulation in the current setting. As the morphisms in this category are functions, we will only introduce bisimulations induced by maps. A different approach would consider defining morphisms as relations, in which case a bisimulation would simply be a symmetric simulation relation, that is, a relation R such that both R and R^{-1} are simulations. We direct the reader to [44] for an account of such an approach and proceed with the definition:

Definition 2.5 (Bisimulations of Abstract Control Systems)**.** Let Φ_X and Φ_Y be abstract control systems over X and Y with monoids $\mathcal{M}_X$ and $\mathcal{M}_Y$ respectively. A simulation (ϕ, φ) from Φ_X to Φ_Y is a bisimulation when $(\phi(x), n) \in \Phi_Y^{-1}(Y)$ implies:

$$\forall x' \in \phi^{-1}(\phi(x)) \quad \exists\, m \in \mathcal{M}_X(x') \quad \text{such that} \quad \varphi(x', m) = n \tag{2.10}$$

We note that in the special case where φ is in fact the identity map on $\mathcal{M}_X$, that is, $\varphi = id : \mathcal{M}_X \longrightarrow \mathcal{M}_X = \mathcal{M}_Y$, we recover the notion of bisimulation introduced in [27] by regarding the graph Γ of ϕ:

$$\Gamma = \{(x, y) \in X \times Y \ : \ y = \phi(x)\}$$

as the bisimulation relation. In fact, if $(x, y) \in \Gamma$ and $x \xrightarrow{m} x'$, then by commutativity of diagram (2.9) we have that $\phi(x) = y \xrightarrow{m} y'$ and $(x', y') \in \Gamma$ by noting that $\varphi(x, m) = id(m) = m$ and $\phi(x') = y'$.

Several other approaches to bisimulation are reported in the literature and we point the reader to the comparative study in [38] and the references therein.

The importance of simulations lies on the fact that simulations capture all trajectories of the simulated abstract control system after being transformed by the state aggregation map ϕ. This allows to transfer the study of properties over the trajectories of Φ_X to the study of the same properties over trajectories of Φ_Y. We now make this fact precise. Instead of trying to define trajectories of abstract control systems (which would be as difficult as defining trajectories of hybrid control systems, see the different approaches in [19, 28, 35]) we will restrict our attention to the orbits of abstract control systems.

Definition 2.6. Let Φ_X be an abstract control system over X with monoid $\mathcal{M}_X$. The set $\mathcal{O}_x$ is an orbit of Φ_X through the point $x \in X$ if there exists a $m \in \mathcal{M}$ such that:

$$\mathcal{O}_x = \{x' \in X \ : \ x' = \Phi_X(x, m') \text{ for some prefix } m' \text{ of } m\} \tag{2.11}$$

Intuitively, the orbit $\mathcal{O}_x$ through x is the set of all the points that are visited on a evolution starting at x and controlled by m. Note that this notion differs from the usual notions of monoid or group action orbit, which

can be recovered as the union of all orbits through x. We can now relate the orbits of abstract control systems to the orbits of the corresponding simulations:

Proposition 2.7 (Orbit Propagation)**.** *Let Φ_X and Φ_Y be abstract control systems over X and Y, respectively and (ϕ, φ) a simulation from Φ_X to Φ_Y, then for every $x \in X$:*

$$\phi(\mathcal{O}_x) \subseteq \mathcal{O}_{\phi(x)} \tag{2.12}$$

Proof. Assume that Φ_Y is a simulation of Φ_X. If $x' \in \mathcal{O}_x$, then, by definition of orbit, there exists a $m \in \mathcal{M}_X$ such that $\Phi_X(x, m) = x'$. Applying ϕ on both sides we get:

$$\begin{aligned} \phi \circ \Phi_X(x, m) &= \phi(x') \\ \Leftrightarrow \Phi_Y(\phi(x), \varphi(x, m)) &= \phi(x') \end{aligned}$$

where the second equality holds by definition of simulation. This shows that $\phi(x') \in \mathcal{O}_{\phi(x)}$, and as x' was any point in $\mathcal{O}_x$, the desired inclusion is proved. □

This result is important as it shows that any abstraction, in this context, can be used to verify reachability based properties of which safety [26] is an important example. This, we state formally in the next corollary:

Corollary 2.8 (Reachability Propagation)**.** *Let Φ_X and Φ_Y be abstract control systems over X and Y, respectively, (ϕ, φ) a simulation from Φ_X to Φ_Y, $x \in X$ and $B \subseteq X$. Then, if every orbit of Φ_Y from $\phi(x)$ does not intersect $\phi(B)$, every orbit of Φ_X from x does not intersect B.*

Proof. Assume for the sake of contradiction that no orbit of Φ_Y from $\phi(x)$ intersects $\phi(B)$ and that there exists an orbit of Φ_X from x intersecting B. Denoting by $\mathcal{O}_x$ such orbit we have $\mathcal{O}_x \cap B \neq \varnothing$. It then follows by Proposition 2.7 that for any $x' \in \mathcal{O}_x \cap B$:

$$\phi(x') \in \phi(\mathcal{O}_x) \cap \phi(B) \subseteq \mathcal{O}_{\phi(x)} \cap \phi(B)$$

which shows that $\mathcal{O}_{\phi(x)} \cap \phi(B) \neq \varnothing$ and leads to the desired contradiction. □

This result provides only a sufficient condition, since if one shows that orbits of Φ_Y do intersect B, one cannot conclude anything about the original system. It is, therefore, desirable to determine abstractions which are not only sufficient but also necessary with respect to reachability based properties. This is the case for bisimulations as we now state:

Corollary 2.9 (Reachability Equivalence)**.** *Let Φ_X and Φ_Y be abstract control systems over X and Y, respectively, (ϕ, φ) a bisimulation from Φ_X to Φ_Y, $x \in X$ and $B \subseteq X$. Then, every orbit of Φ_Y from $\phi(x)$ does not intersect $\phi(B)$ iff every orbit of Φ_X from x does not intersect $\phi^{-1}(\phi(B))$.*

Proof. We only need to show that if some orbit of Φ_Y from $\phi(x)$ intersects $\phi(B)$ then there exist an orbit of Φ_X from x that intersects $\phi^{-1}(\phi(B))$. Assume that $y \in \mathcal{O}_{\phi(x)} \cap \phi(B)$, then by definition of orbit there exist a $n \in \mathcal{M}_Y$ such that $\Phi_Y(\phi(x), n) = y$. Using the fact that (ϕ, φ) is a bisimulation, we know that there exists a $m \in \mathcal{M}_X$ such that $\phi \circ \Phi_X(x, m) = y$ which shows that $\Phi_X(x, m) \in \phi^{-1}(y) \subseteq \phi^{-1}(\phi(B))$. □

3. Compositional Abstractions

We have introduced the basic framework to discuss abstractions of hybrid control systems when regarded as a single component. However, in many situations, we also have knowledge about the internal interconnection structure of hybrid systems and we seek to take advantage of such knowledge. We thus need to model in the current framework, composition and synchronization of hybrid systems. We will follow the categorical description of transition systems in [46], and introduce a notion of parallel composition with synchronization for abstract control systems. Furthermore, we will also determine when such notions are compatible with simulation and bisimulation.

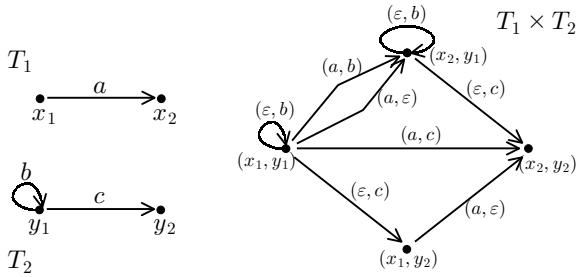


FIGURE 1. Transition systems T_1 and T_2 on the left and the corresponding product transition system $T_1 \times T_2$ on the right.

3.1. **Parallel Composition with Synchronization.** We model parallel composition with synchronization in two steps. In the first step, we perform what can be seen as an asynchronous product, where no interaction between systems is modeled and all possible evolutions of the two systems are allowed. Then we restrict this product by removing undesired evolutions thereby modeling synchronization. We start by introducing the product[5] of abstract control systems:

Definition 3.1 (Product of abstract control systems)**.** Let $\Phi_X : X \times \mathcal{M}_X \rightarrow X$ and $\Phi_Y : Y \times \mathcal{M}_Y \rightarrow Y$ be two abstract control systems. The product of Φ_X and Φ_Y is the abstract control system $\Phi_X \times \Phi_Y : (X \times Y) \times (\mathcal{M}_X \otimes \mathcal{M}_Y) \rightarrow X \times Y$ defined by:

$$\Phi_X \times \Phi_Y((x, y), (m, n)) = (\Phi_X(x, m), \Phi_Y(y, n))$$

We now present a simple example of a product of two systems.

Example 3.2. Consider the transition systems T_1 and T_2 inspired from [46] and displayed on the left of Figure 1, where only the transitions labeled by the monoid generators are represented. The product of these transitions systems $T_1 \times T_2$ will consist of all possible evolutions of both systems as displayed on the right of Figure 1.

Example 3.3. For a purely continuous example, consider two control systems (M, U, f) and (N, V, g). On the product state space $M \times N$ and the product input space $U \times V$ we can define the product control system defined by $f \times g : M \times N \rightarrow T(M \times N)$ defined by $(f \times g)((x, u), (y, v)) = (f(x, u), g(y, v))$. Clearly, this control system captures all the possible trajectories of the individual control systems f and g.

As this notion of product captures all possible evolutions of both systems, we cannot model how the behavior of one of the systems influences the behavior of the other system. This will be achieved through the operation of restriction, which allows to remove undesired evolutions from the product system.

Definition 3.4 (Restriction of abstract control systems)**.** Let $\Phi_X : X \times \mathcal{M}_X \rightarrow X$ be an abstract control system and L a subset of $X \times \mathcal{M}_X$.The restriction of Φ_X to L is the abstract control system $\Phi_X|_L : L_X \times \mathcal{M}_X \rightarrow L_X$ defined by:

$$\Phi_X|_L(x, m) = \Phi_X(x, m) \quad \text{iff} \quad (x, m) \in L \text{ and for any prefix } m' \text{ of } m,\ \Phi_X(x, m') \in L_X \wedge (x, m') \in L$$

where L_X is the projection of L on X.

This restriction operation captures synchronization notions for continuous and discrete control systems as we now describe in the following examples:

[5]This notion of product corresponds to the product in the category of abstract control systems. See, for example, Appendix B for a definition of product in a category.

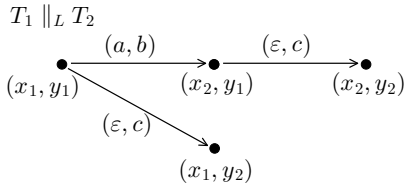


FIGURE 2. Parallel composition with synchronization of the transition systems T_1 and T_2 displayed on the left of Figure 1.

Example 3.5. As an example of continuous synchronization we consider two robots in the plane. The state space for each robot is $\mathbb{R}^4$ consisting of (x_1, x_2, x_3, x_4), where (x_1, x_2) represent the robot position and (x_3, x_4) the robot velocity. The coordinates for the second robot are (y_1, y_2, y_3, y_4) and have similar interpretation. We now assume that the robots are cooperatively transporting a rigid object. The synchronization of the two robots can be modeled as the subset $L_{\mathbb{R}^4\times\mathbb{R}^4} \subseteq \mathbb{R}^4 \times \mathbb{R}^4$ defined by:

$$L_{\mathbb{R}^4\times\mathbb{R}^4} = \left\{((x_1, x_2, x_3, x_4), (y_1, y_2, y_3, y_4)) \in \mathbb{R}^4 \times \mathbb{R}^4 \ : \ x_1 = y_1 + k_1 \wedge x_2 = y_2 + k_2\right\}$$

where k_1 and k_2 are positive constants related with the dimensions of the transported object. The final synchronizing set is obtained by appending to $L_{\mathbb{R}^4\times\mathbb{R}^4}$ the monoids to obtain $L = L_{\mathbb{R}^4\times\mathbb{R}^4} \times \mathcal{M}_X \times \mathcal{M}_Y$.

The previous example modeled restriction at the level of the states, however for purely discrete systems one of the most popular notions of synchronization consists of synchronizing state machines or transition systems on common events. This is captured in this framework by properly choosing the set L as shown in the next example:

Example 3.6. Consider the transition system displayed on the left of Figure 1. By specifying the set L as the product and prefix closure of:

$$\{x_1, x_2\} \times \{y_1, y_2\} \times \{(\varepsilon, c), (a, b)\}$$

it is possible to synchronize the event a with the event b on the parallel composition of these systems. The resulting transition system is displayed in Figure 2.

With the notions of product and restriction at hand, we can now define a general operation of parallel composition with synchronization.

Definition 3.7 (Parallel Composition with synchronization)**.** Let $\Phi_X : X \times \mathcal{M}_X \to X$ and $\Phi_Y : Y \times \mathcal{M}_Y \to Y$ be two abstract control systems and consider the set $L \subseteq (X \times Y) \times (\mathcal{M}_X \otimes \mathcal{M}_Y)$. The parallel composition of Φ_X and Φ_Y with synchronization over L is the abstract control system denoted by $\Phi_X \|_L \Phi_Y$ and defined as:

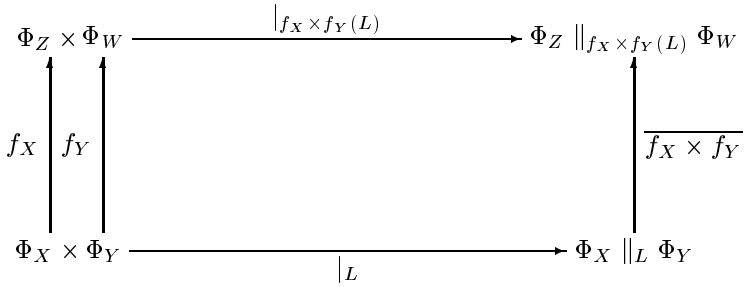
$$\Phi_X \|_L \Phi_Y = (\Phi_X \times \Phi_Y)|_L \tag{3.1}$$

3.2. **Compositionality of Simulations.** We now determine if composition of subsystems is compatible with simulation. This compatibility allows to break the computation of abstractions by computing abstractions of each subsystem individually. The resulting abstractions are then composed and synchronized to obtain an abstraction of the original system. The next result shows that this is always possible for simulations and describes how the process of computing abstractions can be rendered more efficient by exploiting the interconnection structure of hybrid systems.

Theorem 3.8 (Compositionality of Simulations)**.** *Let Φ_X, Φ_Y, Φ_Z, Φ_W be abstract control systems and let $f_X = (\phi_X, \varphi_X)$ and $f_Y = (\phi_Y, \varphi_Y)$ be simulations from Φ_X to Φ_Z and from Φ_Y to Φ_W, respectively. Consider a set $L \subseteq (X \times Y) \times (\mathcal{M}_X \otimes \mathcal{M}_Y)$ and its projection $L_{X\times Y}$ on $X \times Y$. The parallel composition of Φ_Z and Φ_W with synchronization over $(f_X \times f_Y)(L)$ is a simulation of the parallel composition of Φ_X and Φ_Y*

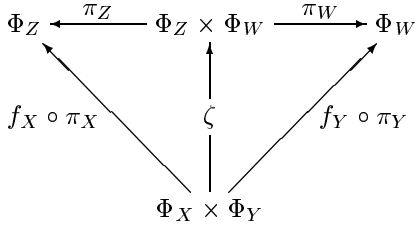
with synchronization over L, where the simulation from $\Phi_X \parallel_L \Phi_Y$ to $\Phi_Z \parallel_{(f_X \times f_Y)(L)} \Phi_W$ is $\overline{f_X \times f_Y} = f_X \times f_Y|_{(\Phi_X \parallel_L \Phi_Y)^{-1}(L_{X\times Y})}$.

The contents of the previous theorem can equivalently be expressed in the following commutative diagram:

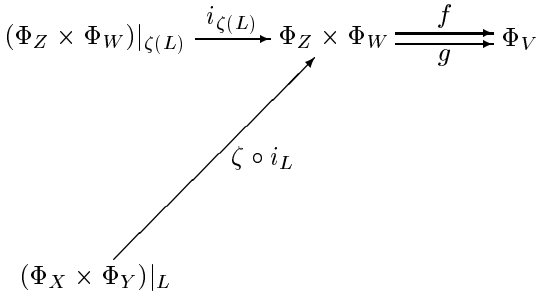


which emphasizes its importance. We can see that, in general, it is much easier to abstract each individual subsystem Φ_X and Φ_Y and by parallel composition with synchronization on $f_X \times f_Y(L)$ obtain $\Phi_Z \parallel_{f_X \times f_Y(L)} \Phi_W$, then is to abstract directly $\Phi_X \parallel_L \Phi_Y$. The above result was stated for parallel composition of two abstract control systems but it can be easily extended to any finite number of abstract control systems.

Proof. We shall make use of the categorical notions of product and equalizer reviewed in Appendix A. Consider the product system $(\Phi_Z \times \Phi_W, \pi_Z, \pi_W)$ and the triple $(\Phi_X \times \Phi_Y, f_X \circ \pi_X, f_Y \circ \pi_Y)$. By definition of product we know that there is one and only one morphism ζ such that:



commutes and this morphism is given by $\zeta =< f_X, f_Y >= f_X \times f_Y$, meaning that $f_X \times f_Y$ is a simulation from $\Phi_X \times \Phi_Y$ to $\Phi_Z \times \Phi_W$. We now make use of the fact that the operation of restriction can be categorically defined by an equalizer. With this in mind we consider the following diagram:



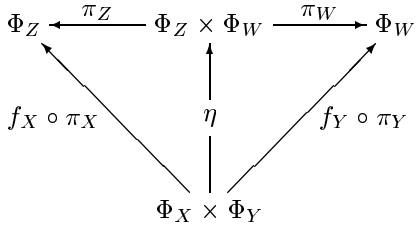
(3.2)

where $(\Phi_Z \times \Phi_W|_{\zeta(L)}, i_{\zeta(L)})$ is the equalizer of f and g, which agree on $\zeta(L)$. We now show that $f \circ \zeta \circ i_L = g \circ \zeta \circ i_L$, for i_L the inclusion morphism from $\Phi_X \times \Phi_Y|_L$ to $\Phi_X \times \Phi_Y$. This follows from $(\Phi_X \parallel_L \Phi_Y)^{-1}(L_{X\times Y}) \subseteq L$, which implies $\zeta \circ i_L((\Phi_X \parallel_L \Phi_Y)^{-1}(L_{X\times Y})) = \zeta((\Phi_X \parallel_L \Phi_Y)^{-1}(L_{X\times Y})) \subseteq \zeta(L)$ since $f = g$ on $\zeta(L)$, we have $f \circ \zeta \circ i_L = g \circ \zeta \circ i_L$. Therefore, by definition of equalizer, there exists one and only one simulation η from $\Phi_X \parallel_L \Phi_Y$ to $\Phi_Z \parallel_{\zeta(L)} \Phi_W$ which is given by $\eta = \zeta \circ i_L = f_X \times f_Y|_{(\Phi_X \parallel_L \Phi_Y)^{-1}(L)}$. □

3.3. **Compositionality of Bisimulations.** We now extend the previous compatibility results from simulations to bisimulations. We start with a very simple lemma stating that product respects bisimulations:

Lemma 3.9. *Given abstract control systems Φ_X, Φ_Y, Φ_Z, Φ_W and bisimulations f_X and f_Y from Φ_X to Φ_Z and from Φ_Y to Φ_W, respectively, the product system $\Phi_Z \times \Phi_W$ is a bisimulation of $\Phi_X \times \Phi_Y$, where the bisimulation from $\Phi_X \times \Phi_Y$ to $\Phi_Z \times \Phi_W$ is $f_X \times f_Y$.*

Proof. Consider the following commutative diagram:



By definition of product there exists one and only one morphism η such that the above diagram commutes. In fact, η is the morphism $\eta = (f_X \circ \pi_X, f_Y \circ \pi_Y) = f_X \times f_Y$ meaning that $\Phi_Z \times \Phi_W$ is a simulation of $\Phi_X \times \Phi_Y$ with simulation $f_X \times f_Y$ from $\Phi_X \times \Phi_Y$ to $\Phi_Z \times \Phi_W$. We now show that $f_X \times f_Y$ is also a bisimulation. Let $((z,w),(o,p)) \in (Z \times W) \times (\mathcal{M}_Z \otimes \mathcal{M}_Z)$. Since Φ_Z is a bisimulation of Φ_X, there is a $(x,m) \in X \times \mathcal{M}_X$ such that $f_X(x,m) = (z,o)$ and similarly there exists a $(y,n) \in Y \times \mathcal{M}_Y$ such that $f_Y(y,n) = (w,p)$. We now see that $f_X \times f_Y((x,y),(m,n)) = ((z,w),(o,p))$ which shows that $f_X \times f_Y$ is also a bisimulation. □

Although the product respects bisimulations the same does not happen with the operation of restriction. Consider the example displayed in Figure 3 where the abstract control system on top is bisimilar to the

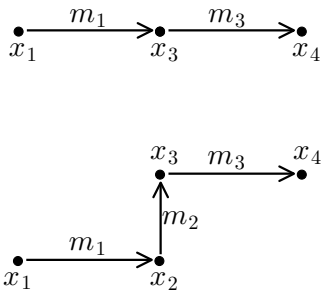


FIGURE 3. Bisimilar abstract control systems.

system below with respect to the maps defined by:

$$\phi(x_1) = x_1,\ \phi(x_2) = x_3,\ \phi(x_3) = x_3,\ \phi(x_4) = x_4$$
$$\varphi(x_1,\varepsilon) = \varepsilon,\ \varphi(x_1,m_1) = m_1,\ \varphi(x_2,\varepsilon) = \varepsilon,\ \varphi(x_2,m_2) = \varepsilon,\ \varphi(x_3,\varepsilon) = \varepsilon,\ \varphi(x_3,m_3) = m_3,\ \varphi(x_4,\varepsilon) = \varepsilon$$

If we now restrict the system below to the product and prefix closure of:

$$L = \big\{(x_1,\varepsilon),(x_1,m_1),(x_2,\varepsilon),(x_3,\varepsilon),(x_3,m_3),(x_4,\varepsilon)\big\} \tag{3.3}$$

and restrict the system on top to $(\phi,\varphi)(L)$ the systems will cease to be bisimilar since the system on top can move from x_3 to x_4 by m_3 but the restricted system can not simulate that evolution when on $x_2 \in \phi^{-1}(x_3)$. This difficulty can be overcomed by additional assumptions as stated in the next proposition:

Proposition 3.10. *Let Φ_X and Φ_Y be abstract control systems, f a bisimulation from Φ_X to Φ_Y, L a subset of $X \times \mathcal{M}_X$ satisfying $f^{-1}(f(L)) = L$. The restriction $\Phi_X|_L$ is a bisimulation of $\Phi_Y|_{f(L)}$ where $f|_{\Phi_X|_L^{-1}(L_X)}$ is the bisimulation from $\Phi_X|_L$ to $\Phi_Y|_{f(L)}$.*

Proof. Let $f = (\phi, \varphi)$ and $(y, n) \in \Phi_Y|^{-1}_{f(L)}(\phi(L_X))$. Since $(y, n) \in \Phi_Y^{-1}(Y)$ and Φ_Y is bisimilar to Φ_X we know that for every $x \in \phi^{-1}(y)$ there exists a m such that $\varphi(x, m) = n$. We now show that such (x, m) also belongs to $\Phi_X|^{-1}_L(L_X)$. Let $x' = \Phi_X(x, m)$ and note that $\phi(x') = \Phi_Y(y, n)$ since f is a simulation from Φ_X to Φ_Y. From the assumption $f^{-1}(f(L)) = L$, it follows that $x' \in L_X$ and $(x, m) \in L$ so that by definition of restriction $(x, m) \in \Phi_X|^{-1}_L(L_X)$ as desired. □

The above propositions lead to the following result concerning the compositionality of bisimulations:

Theorem 3.11 (Compositionality of Bisimulations)**.** *Let Φ_X, Φ_Y, Φ_Z, Φ_W be abstract control systems, $f_X = (\phi_X, \varphi_X)$ and $f_Y = (\phi_Y, \varphi_Y)$ bisimulations from Φ_X to Φ_Z and from Φ_Y to Φ_W, respectively, L a subset of $\subseteq (X \times Y) \times (\mathcal{M}_X \otimes \mathcal{M}_Y)$ satisfying $(f_X \times f_Y)^{-1}(f_X \times f_Y(L)) = L$ and denote by $L_{X\times Y}$ the projection of L on $X \times Y$. The parallel composition of the simulations Φ_Z and Φ_W with synchronization over $(f_X \times f_Y)(L)$ is a bisimulation of the parallel composition of Φ_X and Φ_Y with synchronization over L, where the simulation from $\Phi_X \parallel_L \Phi_Y$ to $\Phi_Z \parallel_{(f_X\times f_Y)(L)} \Phi_W$ is $\overline{f_X \times f_Y} = f_X \times f_Y|_{(\Phi_X \parallel_L \Phi_Y)^{-1}(L_{X\times Y})}$.*

From the previous result we conclude that if we have a means of computing bisimulations and choose the synchronization set L carefully, we can compute bisimulations by exploring interconnection of large-scale systems. In the next section we provide a construction to effectively compute abstractions and in certain situations bisimulations of hybrid control systems.

4. Hybrid Control Systems

Having developed the general theory at a fairly abstract level, allowing to show the desired results with relatively ease, we turn to the application of such results to hybrid control systems. The results presented in this section can be seen as a translation to the language of hybrid systems of the introduced results for abstract control systems. Inevitably, the statement of such results will become extremely complicated by making explicit all the relevant conditions at the level of invariants, guards, etc. The reader is urged to contrast the simplicity of the abstract control systems formulation with the tedious version for hybrid systems.

We start by reviewing the hybrid automaton model to set the notation for the remaining section.

Definition 4.1 (Hybrid Automata)**.** An hybrid automaton is a tuple
$H_X = (X, X_0, \Sigma_X, U_X, f_X, Inv_X, Guard_X, Reset_X)$ consisting of:

- the state space $X = \{X_q\}_{q\in Q}$, a collection of smooth manifolds X_q parameterized by a finite set of discrete states Q,
- a set of initial states $X_0 \subseteq X$,
- a finite set of labels Σ_X, parameterizing the discrete transitions between discrete states,
- the continuous input space $U_X = \{U_X^q\}_{q\in Q}$, a collection of inputs spaces parameterized by the discrete states,
- a collection of control systems $f_X = \{f_X^q\}_{q\in Q}$, $f_X^q : Inv_X^q \times U_X^q \to TInv_X^q$, parameterized by the discrete locations,
- the invariant $Inv_X = \{Inv_X^q\}_{q\in Q}$, $Inv_X^q \subseteq X_q$, a collection of subsets of X parameterized by the discrete locations,
- the guards $Guard_X = \{Guard_X^{(q,\sigma,q')}\}_{(q,\sigma,q')\in\Pi}$, $Guard_X^{(q,\sigma,q')} \subseteq Inv_X^q$, a collection of subsets of the invariant, parameterized by Π, the subset of all the triples $(q, \sigma, q') \in Q \times \Sigma_X \times Q$ such that there exists a discrete transition from q to q' labeled by σ,
- the reset maps, $Reset_X = \{Reset_X^{(q,\sigma,q')}\}_{(q,\sigma,q')\in\Pi}$, $Reset_X^{(q,\sigma,q')} : Guard_X^{(q,\sigma,q')} \to 2^{Inv_{q'}}$, a collection of set valued maps defining the possible locations of the continuous state after the discrete transition.

We now translate the theory developed for abstract control systems to the language of hybrid automata. The principal link between the two formulations will be given by the notion of simulation that we now characterize in terms of hybrid automata:

Proposition 4.2. *Let*

$$\begin{aligned} H_X &= (X, X_0, \Sigma_X, U_X, f_X, Inv_X, Guard_X, Reset_X) \\ H_Y &= (Y, Y_0, \Sigma_Y, U_Y, f_Y, Inv_Y, Guard_Y, Reset_Y) \end{aligned}$$

be hybrid control systems and $\mathcal{D}_X = \{\mathcal{D}_X^q\}_{q\in Q}$ *and* $\mathcal{D}_Y = \{\mathcal{D}_Y^p\}_{p\in P}$ *the collection of vector fields pointwise defined by:*

$$\begin{aligned} \mathcal{D}_X^q(x) &= \bigcup_{u\in U_X^q} f_X^q(x,u) \\ \mathcal{D}_Y^p(y) &= \bigcup_{v\in U_Y^p} f_Y^p(y,v) \end{aligned}$$

A pair of maps $\phi = (\phi_D, \phi_C) : X \longrightarrow Y$ *consisting of:*

- **Discrete state aggregation map:** $\phi_D : Q \longrightarrow P$,
- **Continuous state aggregation map:** $\phi_C = \{\phi_C^q\}_{q\in Q}$, $\phi_C^q : Inv_X^q \longrightarrow Inv_Y^{\phi_D(p)}$

satisfying:

- **Preservation of initial conditions:** $f(X_0) \subseteq Y_0$,
- **Continuous abstraction:** $T_x\phi_C^q(\mathcal{D}_X^q(x)) \subseteq \mathcal{D}_Y^{\phi_D(q)} \circ \phi_X^q(x)$,

and a map $\varphi_D : Q \times \Sigma_X \longrightarrow \Sigma_Y$ *satisfying:*

- **Preservation of transition enabling:** $\phi_C^q(Guard_X^{(q,\sigma,q')}) \subseteq Guard_Y^{(\phi_D(q),\varphi_D(q,\sigma),\phi_D(q'))}$,
- **Preservation of resets:** $\phi_C^q(Reset_X^{(q,\sigma,q')}(x)) \subseteq Reset_Y^{(\phi_D(q),\varphi_D(q,\sigma),\phi_D(q'))} \circ \phi_C^q(x)$.

defines a simulation from H_X *to* H_Y.

Proof. We will show that the given data defines a simulation (ϕ, φ) from H_X to H_Y. For the state aggregation map ϕ we simply take $\phi = (\phi_D, \phi_C)$. The definition of φ takes more work and will make use of the freeness properties of the monoid $\mathcal{M}_X$ associated with the hybrid system H_X. Recall that by Proposition 2.2 the monoid $\mathcal{M}_X = \coprod_{n\in\mathbb{N}_0}(\Sigma_X^* \cup U_X^*)^n$ is freely generated by the set $\Sigma_X^* \cup U_X^*$ and that Σ_X^* is freely generated by the set Σ_X, therefore we only need to define φ for elements in $\Sigma_X \cup U_X^*$. We treat the discrete case first:

As we considered only deterministic systems in the abstract framework we start by enlarging the set Σ_X to $\overline{\Sigma_X}$ so as to parameterize the nondeterminism. We replace every $\sigma \in \Sigma_X$ with (σ, x') for every $x' \in Reset_X^{(q,\sigma,q')}$ and similarly for Σ_Y. This ensures that the abstract control systems Φ_X and Φ_Y associated with H_X and H_Y are now deterministic, since different elements in $Reset_X^{(q,\sigma,q')}$ are parameterized by different symbols in $\overline{\Sigma_X}$. We also extend φ_D to $\overline{\varphi_D} : Q \times \overline{\Sigma_X} \longrightarrow \overline{\Sigma_Y}$ by $\overline{\varphi_D}(q,(\sigma,x')) = (\sigma', x'')$, where $\sigma' = \varphi_D(q,\sigma)$ and x'' satisfies:

$$\begin{aligned} \Phi_Y((\phi_D(q), \phi_C^q(x)), \overline{\varphi_D}(q,(\sigma,x'))) &= \phi(q', x'') \\ &= \phi \circ \Phi_X((q,x),(\sigma,x')) \end{aligned} \tag{4.1}$$

which is always possible given the preservation of transition enabling and resets assumptions on the map φ_D. This allows to define φ by $\overline{\varphi_D}$ for elements in $\overline{\Sigma_X}$. Freeness of Σ_X^* and (4.1) now ensures the existence of an extension of $\overline{\varphi_D}$ satisfying conditions (2.7), (2.8) and (2.9) of Definition 2.4.

For elements in U_X^* we consider an arbitrary but fixed $q \in Q$ and make use of the fact (shown in [43]) that a map $\phi_C^q : Inv_X^q \longrightarrow Inv_Y^{\phi_D(p)}$ satisfying $T_x\phi_C^q(\mathcal{D}_X^q(x)) \subseteq \mathcal{D}_Y^{\phi_D(q)} \circ \phi_C^q(x)$ defines a unique map $\varphi_C^q : Inv_X^q \times U_X^q \longrightarrow U_Y^{\phi_D(p)}$ such that for any pair $(x(t), u(t))$ of state and input trajectories of f_X^q, $(\phi_C^q, \varphi_C^q)(x(t), u(t))$ is a

state and input trajectory of $f_Y^{\phi_D(q)}$. We now see that for any $u^{t'} \in U_X^*$ with codomain U_X^q we have:

$$\begin{aligned} \phi \circ \Phi_X(x, u^{t'}) &= (\phi_D(q), \phi_C^q \circ \gamma_x(t')) \\ &= \Phi_Y((\phi_D(q), \phi_C^q(x)), \varphi_C^q(x, u^{t'})) \end{aligned} \tag{4.2}$$

where γ_x is the integral curve of $f_q^X(x, u^{t'})$. Furthermore, by definition of ϕ_C^q we have $\phi_C^q(Inv_X^q) \subseteq Inv_Y^{\phi_D(q)}$ so that $\Phi_X(x, u^t) \in Inv_X^q$ for every prefix u^t of $u^{t'}$ implies that $\Phi_Y(\phi_C^q(x), \varphi_C^q(x, u^t)) \in Inv_Y^{\phi_D(q)}$ for every prefix $\varphi_C^q(x, u^t)$ of $\varphi_C^q(x, u^{t'})$. It then follows from (4.2) that we can define φ by $\{\varphi_C^q\}_{q \in Q}$ for elements in U^* as it satisfies conditions (2.7), (2.8) and (2.9) of Definition 2.4.

Having defined φ for elements in Σ_X and U^*, it follows that φ extends uniquely to $\mathcal{M}_X$ thereby defining a simulation (ϕ, φ) from H_X to H_Y. □

The previous proposition can also be used in a more constructive way if used as a construction for hybrid abstractions. Before presenting the construction, we review some notions of invariance. Given a smooth map $f : M \longrightarrow N$ we denote by Tf the tangent map or derivative of f. We will also use the notation $\ker(Tf)$ to denote the distribution consisting of all vector fields X such that $Tf(X) = 0$. Given a set A, we say that A is invariant under a vector field X if every trajectory of X starting in A will remain in A for all time. This notion is extended to invariance under a distribution by considering that A is invariant under every vector field belonging to a distribution. These concepts are now used in the following construction, which given H_X and ϕ, constructs H_Y simulating H_X.

Construction 4.3 (Construction of Abstractions)**.** *Let $H_X = (X, X_0, \Sigma_X, U_X, f_X, Inv_X, Guard_X, Reset_X)$ be an hybrid system, $\phi = (\phi_D, \{\phi_C^q\}_{q \in Q}) : X \longrightarrow Y$ a pair of maps, with $\phi_D : Q \longrightarrow P$ and $\phi_C^q : Inv_X^q \longrightarrow Inv_Y^{\phi_D(q)}$. Hybrid system $H_Y = (Y, Y_0, \Sigma_Y, U_Y, f_Y, Inv_Y, Guard_Y, Reset_Y)$ is obtained from H_X and ϕ by the following steps:*

1. $Y := f(X)$,
2. $Y_0 := f(X_0)$,
3. $\Sigma_Y := \Sigma_X$,
4. $U_Y := \{U_Y^p\}_{p \in P}$, $U_Y^{\phi_D(q)} = \cup_{q' \in \phi_D^{-1} \circ \phi_D(q)} \varphi_C^{q'}(U_X^{q'})$,
5. $f_Y^{\phi_D(q)} :=$ *the continuous abstraction*[6] *of every* $f_X^{q'}$ *such that* $q' \in \phi_D^{-1} \circ \phi_D(q)$,
6. $Inv_Y^{\phi_D(q)} := \cup_{q' \in \phi_D^{-1} \circ \phi_D(q)} \phi_C^{q'}(Inv_X^{q'})$,
7. $Guard_Y^{(\phi_D(q), \sigma, \phi_D(q'))} := \cup_{q'' \in \phi_D^{-1} \circ \phi_D(q)} \phi_C^{q''}(Guard_X^{(q'', \sigma, q')})$,
8. $Reset_Y^{(\phi_D(q), \sigma, \phi_D(q'))} \circ \phi_C^q(x) := \cup_{q'' \in \phi_D^{-1} \circ \phi_D(q)} \phi_C^{q''}(Reset_X^{(q'', \sigma, q')}(x))$.

The steps of the previous construction were designed so as to provide the conditions described in Proposition 4.2. The first step defines the new state space as the image of X by f. Since the invariants, guards and resets can be seen as subsets of the state space, they are also defined as the image of H_X invariants, guards and resets by f as described in steps 6, 7 and 8. Similarly, the space of continuous inputs U_Y is obtained as the image of U_X by the map φ_C, uniquely determined by ϕ as described in [43]. Note, that in concrete applications the explicit computation of U_Y and φ is not necessary as the construction of continuous abstractions described in [33] and required by step 5 provides the equations for the abstracted control system. The second step ensures that "preservation of initial conditions" is met while the third step defines the discrete labels in H_Y as the discrete labels in H_X.

[6] Continuous abstractions are discussed in [31] for linear systems and in [33] for nonlinear systems. However, in these references abstractions of a *single* control system are considered whereas in the present situation it may be necessary to determine an abstraction of *several* control systems for which the results of [31, 33] can be easily extended. Consider, for example, two control affine systems defined by the affine distributions $X_1 + \Delta_1$ and $X_2 + \Delta_2$ and a aggregation map $\phi : M \longrightarrow N$. If $T_x\phi(X_1(x)) = T_x\phi(X_2(x))$ for every $x \in M$, then the abstraction of both control systems is simply given by $T\phi(X_1 + \Delta_1) + T\phi(X_2 + \Delta_2)$, otherwise we can take as an abstraction $T\phi(X_1 + \Delta_1) + span(T\phi(X_2)) + T\phi(\Delta_2)$ or $T\phi(X_2 + \Delta_2) + span(T\phi(X_1)) + T\phi(\Delta_1)$.

This construction determines an abstraction of a given hybrid control system H_X based on the state aggregation map ϕ as asserted in the next result where the relevant assumptions on the input data are provided:

Theorem 4.4 (Computation of Hybrid Abstractions)**.** *Given an hybrid control system H_X where the control systems f_X^q are control affine and a map $\phi = (\phi_D, \phi_C)$ where $\phi_D : Q \longrightarrow P$ and $\phi_C = \{\phi_C^q\}_{q\in Q}$, $\phi_C^q : Inv_X^q \longrightarrow Inv_Y^{\phi_D(q)}$ is a submersion, Construction 4.3 defines a hybrid control system H_Y which is a simulation of H_X.*

Proof. The result follows by Proposition 4.2 by considering:

- the map $\varphi_D : Q \times \Sigma_X \longrightarrow \Sigma_Y$ defined by $\varphi_D(q,\sigma) = \sigma$ for every (q,σ) such that there exists a $x \in Inv_X^q$ satisfying $((q,x),\sigma) \in \Phi_X^{-1}(X)$, where Φ_X is the abstract control system associated with hybrid control system H_X.
- the fact that $\mathcal{D}_Y^{\phi_D(q)}$ satisfies $T_x\phi_C^{q'}(\mathcal{D}_X^{q'}(x)) \subseteq \mathcal{D}_Y^{\phi_D(q)} \circ \phi_C^q(x)$ for every $x \in Inv_X^{q'}$ and every $q' \in \phi_D^{-1} \circ \phi_D(q)$.

□

As was already discussed for abstract control systems, bisimulations provide sufficient as well as necessary conditions regarding reachability equivalence. It is therefore important to determine when the abstraction determined by Construction 4.3 is in fact a bisimulation. Sufficient conditions are presented in the following result:

Theorem 4.5 (Bisimilar Hybrid Systems)**.** *Let $H_X = (X, X_0, \Sigma_X, U_X, f_X, Inv_X, Guard_X, Reset_X)$ be a hybrid control system, $\phi = (\phi_D, \phi_C)$ a state aggregation map satisfying the conditions of Theorem 4.4 and $H_Y = (Y, Y_0, \Sigma_Y, U_Y, f_Y, Inv_Y, Guard_Y, Reset_Y)$ the hybrid control system defined by Construction 4.3. If the following conditions are met:*

- **Continuous invariance** - for every $q \in Q$:
  - $\ker(T\phi_X^q)$ is controlled invariant for f_X^q,
  - Inv_X^q is invariant for $\ker(T\phi_X^q)$,
  - Every guard on discrete state q is invariant for $\ker(T\phi_X^q)$,
  - The image by the reset maps (of discrete state q) of every point in its domain is invariant for $\ker(T\phi_X^q)$.
- **Discrete invariance** - $\phi_D(q) = \phi_D(q')$ *implies:*
  - $T_x\phi_C^q(\mathcal{D}_X^q(x)) = T_x\phi_C^{q'}(\mathcal{D}_X^{q'}(x))$,
  - $\phi_C^q(Guard_X^{(q,\sigma,q'')}) = \phi_C^{q'}(Guard_X^{(q',\sigma,q''')})$, $\forall q''' \in \phi_D^{-1} \circ \phi(q'')$,
  - $\phi_C^q(x) = \phi_C^{q'}(x) \quad \Rightarrow \quad \phi_C^q(Reset_X^{(q,\sigma,q'')}(x)) = \phi_C^{q'}(Reset_X^{(q',\sigma,q''')}(x))$, $\forall q''' \in \phi_D^{-1} \circ \phi(q'')$.

then H_Y is a bisimulation of H_X.

Proof. As in the proof of Proposition 4.2 we consider that Σ_X, Σ_Y and φ_D have been extended so that (ϕ, φ) is a simulation from Φ_X to Φ_Y. We will show that (2.10) holds for elements in $\Sigma_X \cup U_X^*$ since the freeness properties of $\mathcal{M}_X$ will then imply that (2.10) holds for every element in $\mathcal{M}_X$.

We treat the continuous case first.

From the results reported in [42] we know that if $\ker(T\phi_C^q)$ is controlled invariant for a control system f_X^q, then its abstraction under the map ϕ_C^q is a bisimulation. If several discrete states are aggregated to the same continuous case, the resulting continuous abstraction is still a bisimulation since $\phi_D(q) = \phi_D(q')$ implies that the abstractions of f_X^q and $f_X^{q'}$ are the same. Furthermore, from invariance of Inv_X^q under $\ker(T\phi_X^q)$ we conclude that any continuous trajectory $x(t)$ leaves the invariant iff the abstracted trajectory $\phi_c^q(x(t))$ leaves the invariant, which means that continuous trajectories are well defined on the original system iff they are well

defined in the abstraction. This shows that for any $(\phi(x), n) \in \Phi_Y^{-1}(Y)$, there exists a $(x, m) \in \mathcal{M}_X$ such that $\varphi(x, m) = n$, where Φ_X and Φ_Y are the abstract control systems associated with H_X and H_Y, respectively.

We now consider the discrete case. Assume that we have $\Phi_Y((p, y), \sigma) = (p', y')$ for the abstract control system Φ_Y associated with H_Y. This means that $(p, y) \in Guard_Y^{(p,\sigma,p')}$ and as the guards on H_X are invariant for $\ker(T\phi_C^q)$ and $\phi_D(q) = \phi_D(q')$ implies $\phi_C^q(Guard_X^{(q,\sigma,q'')}) = \phi_C^{q'}(Guard_X^{(q',\sigma,q'')})$ we have that every point $x \in (\phi_C^q)^{-1}(y)$ for $q \in \phi_D^{-1}(p)$ belongs to the guard associated with the transitions (q, σ, q'') where $q'' \in \phi_D^{-1} \circ \phi_D(p')$. It now follows from $\phi_C^q(Reset_X^{(q,\sigma,q'')}(x)) = \phi_C^{q'}(Reset_X^{(q',\sigma,q''')}(x))$ that for every $(q, x) \in \phi^{-1}(p, y)$ there exists a σ such that such that $\varphi_D((q, x), \sigma) = \sigma$ as required by (2.10). The result now follows from the freeness properties of $\mathcal{M}_X$. □

The results regarding the computation of simulations and bisimulation are illustrated in the next section where a spark ignition example is discussed in detail. Furthermore, the proposed construction combined with Theorem 3.8 can be applied to complex hybrid systems to exploit interconnection. In that case the synchronizing set L depends on the specific composition operator used.

5. A spark ignition engine example

We now illustrate the use of Construction 4.3 with a spark ignition engine model taken from [8]. Consider the block diagram representation displayed in Figure 4.

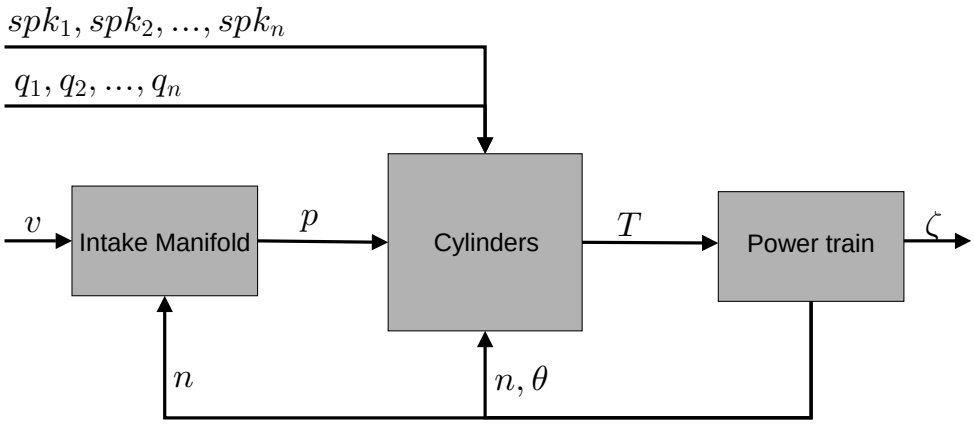


FIGURE 4. Block diagram representation of the spark ignition engine model.

The intake manifold block models the throttle dynamics which can be controlled by applying a voltage v to regulate the throttle angle. This angle, regulates the pressure p from which the air inflow rate into the combustion chamber can be determined. The discrete operation of the n cylinders is modeled by the cylinders block which admits as inputs the injected fuel described by $q_1, q_2, \ldots, q_n$ and discrete inputs $spk_1, \ldots, spk_n$ controlling the spark advance with respect to the top most position of the pistons. Finally, the torque T produced by the cylinders is used in the power train block to determine the crack shaft angle θ, angular velocity n as well as other variables contained in the vector ζ. We defer the reader to [8] for a more detailed explanation of the model.

5.1. Modeling the cylinders.

The hybrid nature of the system comes from the interaction of the continuous dynamics describing the intake manifold and the power train, with the discrete nature of the cylinders

evolution. Considering an engine with four cylinders, each cylinder is modeled by a state machine with 6 states, as displayed in Figure 5.

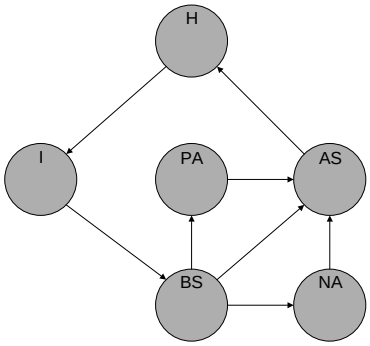


FIGURE 5. State machine modeling the discrete operation of each cylinders.

In the notation of [8], the state I models the *intake* phase where the combustion chamber is filled with the air-fuel mixture until reaching its top most position. The states BS and PA model the compression phase where the piston compresses the air-fuel mixture. The state BS models the *before spark* phase where no spark has yet occurred while the state PA models the *positive advance* phase, where a spark occurs before the piston reaches its top most position. The states AS and NA model the expansion phase. In the state AS, *after spark*, torque is being produced by the explosion of the mixture ignited by a spark. However, if no spark occurs before the piston reaches the top most position, the expansion phase initiates and no torque is produced, this is modeled by the state NA, *negative advance*. Finally the state H models the *exhaust* phase, where the gases produced in the combustion process are expelled from the combustion chamber. While the transitions $H \longrightarrow I$, $I \longrightarrow BS$, $BS \longrightarrow NA$, $AS \longrightarrow H$ depend only on the cylinder position, the remaining transitions depend on the discrete input SPK modeling the occurrence of a spark. Since several transitions depend on the position of the cylinder which is given by the continuous dynamics governing the power train, it is natural to combine the power train model with the state machine describing the cylinders in the hybrid automaton presented in Figure 6.

The continuous dynamics is also displayed in Figure 6, where the continuous states z_1, z_2 and z_3 are used to record the mass of air and fuel injected at the intake phase as well as the spark advance which will be used to determine the torque produced at the state AS as described in [8]. Recording these values is in fact the justification for the several reset maps appearing on the hybrid automaton. The continuous dynamics is equal in every discrete state, except for the state AS where the produced torque T is added to the external torque T_e produced by the other pistons. The production of the torque T_e generated by the other pistons is not captured in this modeled so that T_e is treated as an input. The invariants of each mode are defined by bounds on the piston position measured by the crank shaft angle while the guards involve conditions on the crank shaft angle as well as the external input SPK modeling the occurrence of a spark. The remaining undefined constants and functions are irrelevant for our analysis and can be obtained from [8].

5.2. **Abstracting the hybrid model.** As in an engine several pistons run in parallel to generate torque, the model of the several pistons can be obtained by performing the parallel composition with synchronization of the hybrid automaton describing each piston. We consider an engine with four pistons, where each piston is described by the hybrid automaton displayed in Figure 6 having state space $Q \times \mathbb{R}^6$ and input space $\mathbb{R}$. The synchronization between the pistons is then defined by the set $L \subseteq (Q \times \mathbb{R}^6 \times \mathbb{R})^4$:

(5.1)

$$L = \left\{ (q,x,u) \in (Q \times \mathbb{R}^6 \times \mathbb{R})^4 \ : \ \theta_1 = \theta_2 + 90^o = \theta_3 + 180^o = \theta_4 + 270^o \ \wedge \ T_{ei} = \sum_{j \neq i} T_j \ \ i,j \in \{1,2,3,4\} \right\}$$

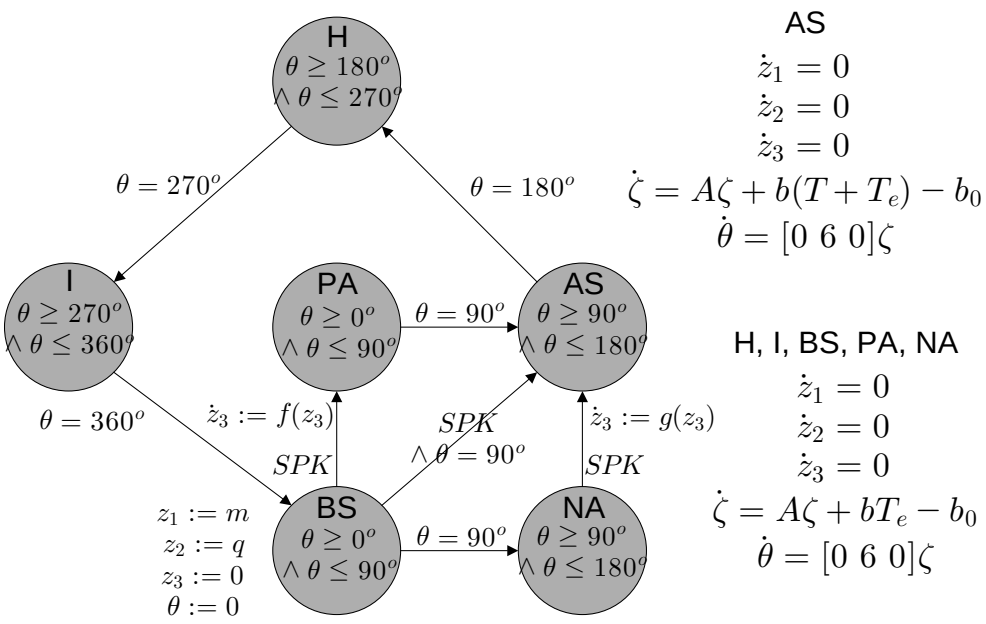


FIGURE 6. Hybrid automaton modeling each cylinder.

We have, therefore, two approaches to compute an abstraction of the engine model. Either we abstract the model of each piston individually and then we compose the abstractions, or we abstract the model of the engine as a whole. Theorem 3.8 ensures that both approaches produce the same results which lead us to abstract each piston individually so as the reduce the complexity of the involved computations.

By inspecting Figure 6 we see that the states PA, BS and NA represent different phases leading the torque production phase. This suggests that they can be aggregated into a single *before torque* state, denoted by BT. This is accomplished by defining the discrete aggregation map $\phi_D : Q \longrightarrow P$ to be:

$$\phi_D(PA) = \phi_D(BS) = \phi_D(NA) = BT \quad \phi_D(AS) = AS \quad \phi_D(H) = H \quad \phi_D(I) = I$$

For the continuous state aggregation map we take the identity, that is:

$$\phi_C^q(x) = x \quad \forall q \in \{I, PA, BS, AS, NA, H\}$$

Following the steps of Construction 4.3 we obtain:

1. $Y = \{BT, AS, H, I\} \times \mathbb{R}^7 = P \times \mathbb{R}^7 = f(X)$,
2. $Y_0 = Y = f(X)$ since the $X_0 = X$,
3. $\Sigma_X = \Sigma_Y$,
4. $U_Y^p = \mathbb{R}$ since we are not aggregating the continuous dynamics,
5. $f_Y^{\phi_D(q)} = f_X^q$ since we are not aggregating the continuous dynamics and the aggregated discrete states have the same continuous dynamics,
6. The invariants remain the same for the states AS, H and I, while the invariant of RT is given by $Inv_Y^{RT} = \phi_C^{PA}(Inv_X^{PA}) \cup \phi_C^{BS}(Inv_X^{BS}) \cup \phi_C^{NA}(Inv_X^{NA}) = Inv_X^{PA} \cup Inv_X^{BS} \cup Inv_X^{NA}$ and is given by the condition $0^o \leq \theta \leq 180^o$.
7. The guard $Guard_X^{(BS,BS \longrightarrow PA,PA)}$ associated with the transition $BS \longrightarrow PA$ remains unchanged as there is no continuous state aggregation. It is therefore expressed as $SPK \wedge \theta \leq 90^o$ since the guard is contained in the invariant. Similarly $Guard_X^{(PA,PA \longrightarrow AS,AS)}$ is transformed to $\theta = 90^o$, $Guard_X^{(BS,BS \longrightarrow AS,AS)}$

transformed to $SPK \wedge \theta = 90^o$, $Guard_X^{(BS,BS\rightarrow NA,NA)}$ transformed to $\theta = 90^o$ and finally $Guard_X^{(NA,NA\rightarrow AS,AS)}$ is transformed to $SPK \wedge 90^o \leq \theta$.

8. The reset $Reset_X^{(BS,BS\rightarrow PA,\overline{PA})}$ is now a reset from BT to BT, while the functional form f of the reset map does not change as there is no continuous state aggregation. Similarly $Reset_X^{(PA,PA\rightarrow AS,AS)}$, $Reset_X^{(BS,BS\rightarrow AS,AS)}$ and $Reset_X^{(BS,BS\rightarrow NA,NA)}$ remain the identity maps and $Reset_X^{(NA,NA\rightarrow AS,AS)}$ is now a reset from BT to AS with the same functional form given by the map g.

The resulting hybrid automaton is displayed in Figure 7.

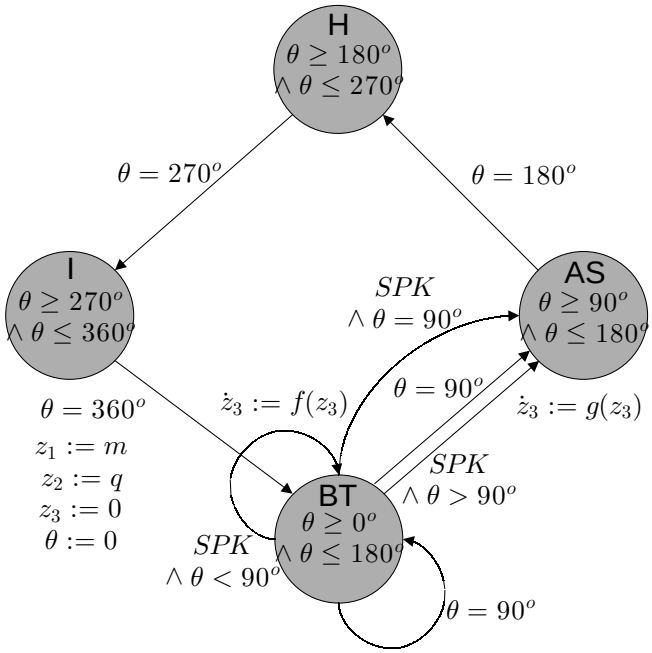


FIGURE 7. Abstraction of the hybrid automaton displayed in Figure 6.

We note that this abstraction fails to be a bisimulation since $\phi_C(Inv_X^{NA}) = Inv_X^{NA} \subseteq Inv_X^{BT} \neq Inv_X^{PA} = \phi_C(Inv_X^{PA})$ and this implies that for a continuous evolution starting at $\theta = 100^o$ on discrete state BT, there is no possible evolution of $PA \in \phi_D^{-1}(BT)$ that can be mapped to the evolution on sate BT as evolutions in PA have to conform to the invariant.

This model can be further simplified by identifying the transition $BT \rightarrow BT$ guarded by $\theta = 90^o$ with the ε transition which is defined for every point in the invariant. Similarly, the transitions from BT to AS with guards $\theta = 90^o$ and $\theta = 90^o \wedge SPK$ can also be identified as they have equal reset maps. This can be formally done by considering a simulation map which does not aggregate the states, but aggregates these transitions. Further aggregation is possible by identifying the states H, I and T which results in the hybrid automaton displayed in Figure 8.

This abstraction can now be composed with similar models of the remaining pistons to obtain the complete hybrid automaton describing the engine. For an engine with 4 pistons this is achieved by determining the product of 4 copies of the hybrid system displayed in Figure 8 followed by the operation of restriction. Following Theorem 3.8, the synchronization set is defined by $f_1 \times f_2 \times f_3 \times f_4(L)$ for the set L defined in (5.1) and maps $f_1 = f_2 = f_3 = f_4$ defining the state aggregation. Since the sequence of abstractions leading to the hybrid model in Figure 8 was based on identity maps for continuous aggregation, it follows that:

$$f_1 \times f_2 \times f_3 \times f_4(L) = L$$

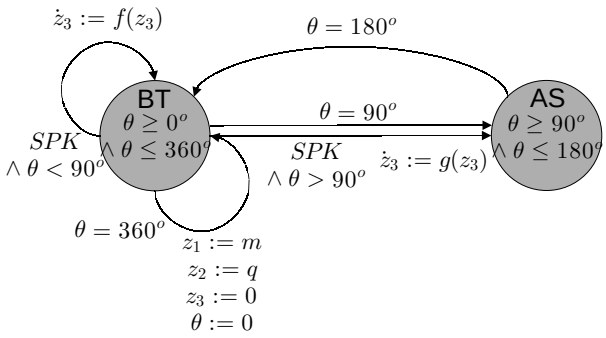


FIGURE 8. Abstraction of the hybrid automaton displayed in Figure 7.

Restricting the product hybrid system to the set L leads to the following possible discrete configurations:

$$\begin{aligned} q_1 &= (BT, BT, BT, AS) \\ q_2 &= (BT, BT, AS, BT) \\ q_3 &= (BT, AS, BT, BT) \\ q_4 &= (AS, BT, BT, BT) \\ q_5 &= (BT, BT, BT, BT) \end{aligned}$$

However, the first 4 discrete states can also be abstracted into a single discrete state TP describing torque production, which leads to an hybrid system with only two discrete states TP and NTP, where NTP corresponds to state q_5, where no piston is producing torque. We note that a similar abstraction has been used in [7] to determine the maximal safe set for idle speed control of automotive engines, although it has not been obtained in any formal framework. The model used in [7] has three discrete states S, S_+ and S_-. State S_- corresponds to state NTP while both states S and S_+ correspond to our state TP. Two states are used in [7] to model the torque production phase to be able to distinguish between the before spark and after spark phases in the compression mode. In our model no such distinction is visible at the level of discrete states, but the continuous reset maps allow to update the variables z_1, z_2 and z_3 required to determine the correct value of the produced torque. We have thus been able to abstract the initial model, where each piston was modeled by a 6 states hybrid automaton to an hybrid system with only two discrete states modeling the synchronized operation of the 4 pistons. Furthermore, by exploiting compositionality we only performed the product of 4 hybrid systems with 2 discrete states each, resulting in a hybrid system with $2^4 = 16$ discrete states. If one would have abstracted the engine model as a whole, the required product hybrid system would have $6^4 = 1296$ discrete states. These numbers clearly illustrate the computational advantages of exploiting compositionality provided by Theorem 3.8. The resulting abstraction can now be used for analysis as is done, for example in [7], or synthesis.

6. CONCLUSIONS

In this paper we have addressed the problem of computing abstractions for hybrid control systems. A notion of abstraction was proposed based on the notions of simulation and bisimulation. These notions were presented in a general setting comprising discrete, continuous and hybrid control systems. Several important properties were proved in this general setting which are directly applicable to hybrid systems. We also introduced a composition operator that allows to construct large-scale, complex hybrid systems by interconnecting smaller hybrid systems. We showed that this composition operator is compatible with abstractions and under certain

conditions also with bisimulation. These results were then instantiated to hybrid control systems where a construction was proposed to compute abstractions based on state aggregation maps.

Several interesting directions for future research remain. It is important to understand how the proposed notions of bisimulation and abstraction can be compared with several other discussed in the literature, specially in the case when inputs and outputs are explicitly defined. Also important is to render the proposed results more computational by looking at special classes of hybrid control systems for which the abstraction process can be completely automated.

Acknowledgments: The authors would like to thank Esfandiar Haghverdi for extremely stimulating discussions on category theory, and its use for hybrid systems.

References

[1] Rajeev Alur, Calin Belta, Franjo Ivancic, Vijay Kumar, Max Mintz, George J. Pappas, Harvey Rubin, and Jonathan Schug. Hybrid modeling and simulation of biomolecular networks. In Maria Domenica Di Benedetto and Alberto Sangiovanni-Vicentelli, editors, *Hybrid Systems: Computation and Control*, volume 2034 of *Lecture Notes in Computer Science*, pages 19–32. Springer Verlag, 2001.

[2] Rajeev Alur, Tom Henzinger, Gerardo Lafferriere, and George J. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(7):971–984, July 2000.

[3] M.A. Arbib and E.G. Manes. *Arrows, Structures and Functors - The Categorical Imperative*. Academic Press Inc, 1975.

[4] Michael Arbib and Ernest G. Manes. Machines in a category: An expository introduction. *SIAM Review*, 16(2):163–192, 1974.

[5] E. Asarin, P. Caspi, and O. Maler. Timed regular expressions. *Journal of the ACM*, (2):172–206, 2002.

[6] A. Balluchi, L. Benvenuti, M. D. Di Benedetto, C. Pinello, and A. L. Sangiovanni-Vicentelli. Automotive engine control and hybrid systems: Challenges and opportunities. *Proceedings of the IEEE*, 88(7):888–912, July 2000.

[7] Andrea Balluchi, Luca Benvenuti, Guido M. Miconi, Ugo Pozzi, Tiziano Villa, Maria D. Di Benedetto, Howard Wong-Toi, and Alberto L. Sangiovanni-Vincentelli. Maximal safe set computation for idle speed control of an automotive engine. In Nancy Lynch and Bruce H. Krogh, editors, *Hybrid Systems: Computation and Control*, volume 1790 of *Lecture Notes in Computer Science*, pages 32–44. Springer Verlag, 2000.

[8] Andrea Balluchi, Luca Benvenutti, Maria Domenica di Benedetto, Claudio Pinello, and Alberto Sangiovanni-Vincentelli. Automotive engine control and hybrid systems: Challenges and opportunities. *Proceedings of the IEEE*, 88:888–912, July 2000.

[9] G. Barret and S. Lafortune. Bisimulation, the supervisor control problem and strong model matching for finite state machines. *Journal of Discrete Event Systems*, 8(4):337–429, December 1998.

[10] P.E. Caines and Y.J. Wei. Hierarchical hybrid control systems: A lattice theoretic formulation. *IEEE Transactions on Automatic Control : Special Issue on Hybrid Systems*, 43(4):501–508, April 1998.

[11] C. Cassandras and S. Lafortune. *Introduction to discrete event systems*. Kluwer Academic Publishers, Boston, MA, 1999.

[12] J.E.R. Cury, B.H. Krogh, and T. Niinomi. Synthesis of supervisory controllers for hybrid systems based on approximating automata. *IEEE Transactions on Automatic Control : Special Issue on Hybrid Systems*, 43(4):564–568, April 1998.

[13] Luca de Alfaro and Thomas A. Henzinger. Interface theories for component-based design. In *Proceedings of the First International Workshop on Embedded Software*, volume 2211 of *Lecture Notes in Computer Science*, pages 148–165, 2001.

[14] Kagan Gokbayrak and Christos G. Cassandras. Hybrid controller for hierarchically decomposed systems. In Nancy Lynch and Bruce H. Krogh, editors, *Hybrid Systems: Computation and Control*, volume 1790 of *Lecture Notes in Computer Science*, pages 117–129. Springer Verlag, 2000.

[15] T.A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, pages 278–292. IEEE Computer Society Press, 1996.

[16] Joao Hespanha, Stephan Bohacek, Katia Obraczka, and Junsoo Lee. Hybrid modeling of tcp congestion control. In Maria Domenica Di Benedetto and Alberto Sangiovanni-Vicentelli, editors, *Hybrid Systems: Computation and Control*, volume 2034 of *Lecture Notes in Computer Science*, pages 291–304. Springer Verlag, 2001.

[17] John E. Hopcroft and Jeffery D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley Publishing Company, USA, 1979.

[18] John M. Howie. *Fundamentals of semigroup theory*. Oxford Science Publications, 1995.

[19] Karl Henrick Johansson, Magnus Egersted, John Lygeros, and S. Sastry. On the regularization of hybrid automata. *Systems and Control Letters*, 38:141–150, 1999.

[20] R. Kumar and V.K. Glarg. *Modeling and Control of Logical Discrete Event Systems*. Kluwer Academic Publishers, 1995.

[21] Saunders Mac Lane. *Categories for the Working Mathematician*. Springer-Verlag, 1971.

[22] F. William Lawvere and Stephen H. Schannel. *Conceptual Mathematics: A First Introduction to Categories*. Cambridge University Press, New York, 1997.

[23] Nancy Lynch, Roberto Segala, and Frits Vaandrager. Hybrid I/O automata revisited. In *Hybrid Systems : Computation and Control*, volume 2034 of *Lecture Notes in Computer Science*, pages 403–417. Springer Verlag, 2001.
[24] P. Madhusudan and P.S. Thiagarajan. Branching time controllers for discrete event systems. *Theoretical Computer Science*, 274:117–149, March 2002.
[25] Z. Manna and A. Pnueli. *The temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, Berlin, 1992.
[26] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, Berlin, January 1995.
[27] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
[28] T. Moor and J. M. Davoren. Robust controller synthesis for hybrid systems using modal logic. In *Hybrid Systems : Computation and Control*, volume 2034 of *Lecture Notes in Computer Science*, pages 433–446. Springer Verlag, 2001.
[29] Peter Niebert and Sergio Yovine. Computing optimal operation schemes for chemical plants in multi-batch mode. In Nancy Lynch and Bruce H. Krogh, editors, *Hybrid Systems: Computation and Control*, volume 1790 of *Lecture Notes in Computer Science*, pages 338–351. Springer Verlag, 2000.
[30] George J. Pappas. Bisimilar linear systems. Technical Report MS-CIS-01-19, University of Pennsylvania, June 2001.
[31] George J. Pappas, Gerardo Lafferriere, and Shankar Sastry. Hierarchically consistent control systems. *IEEE Transactions on Automatic Control*, 45(6):1144–1160, June 2000.
[32] George J. Pappas and Shankar Sastry. Towards continuous abstractions of dynamical and control systems. In P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors, *Hybrid Systems IV*, volume 1273 of *Lecture Notes in Computer Science*, pages 329–341. Springer Verlag, Berlin, Germany, 1997.
[33] George J. Pappas and Slobodan Simic. Consistent hierarchies of affine nonlinear systems. *IEEE Transactions on Automatic Control*, 47(5):745–756, 2002.
[34] D.M.R. Park. *Concurrency and automata on infinite sequences*, volume 104 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.
[35] N. Halbwachs T.A. Henzinger P.H. Ho X. Nicollin A.Olivero J. Sifakis S. Yovine R. Alur, C.Courcoubetis. Hybrid automata: An algorithmic approach to specification and verification of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.
[36] J. Raisch and S.D. O'Young. Discrete approximations and supervisory control of continuous systems. *IEEE Transactions on Automatic Control : Special Issue on Hybrid Systems*, 43(4):569–573, April 1998.
[37] P. J. Ramadge and W. M. Wonham. Supervisory control of discrete event processes. In A. V. Balakrishnan and M. Thoma, editors, *Feedback Control of Linear and Nonlinear Systems*, volume 39 of *Lecture Notes in Control and Information Sciences*, pages 202–214. Springer-Verlag, Berlin, 1982.
[38] Markus Roggenbach and Mila Majster-Cederbaum. Towards a unified view of bisimulation: a comparative study. *Theoretical Computer Science*, (238):81–130, 2000.
[39] J.J.M.M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000.
[40] Eduardo D. Sontag. *Mathematical Control Theory*, volume 6 of *Texts in Applied Mathematics*. Springer-Verlag, New-York, 2nd edition, 1998.
[41] Paulo Tabuada and George J. Pappas. Hybrid abstractions that preserve timed languages. In Marica Domenica di Benedetto and Alberto L. Sangiovanni-Vincentelli, editors, *Hybrid Systems : Computation and Control*, volume 2034 of *Lecture Notes in Computer Science*, pages 501–514. Springer Verlag, 2001.
[42] Paulo Tabuada and George J. Pappas. Bisimilar control affine systems. *Systems and Control Letters*, 2002. Submitted, available at `www.seas.upenn.edu/~tabuadap`.
[43] Paulo Tabuada and George J. Pappas. Quotients of fully nonlinear control systems. In D. Gilliam and J. Rosenthal, editors, *Proceedings of 15th International Symposium on Mathematical Theory of Networks and Systems*, South Bend, Indiana, August 2002. Extended version available at `www.seas.upenn.edu/~tabuadap`.
[44] Paulo Tabuada, George J. Pappas, and Pedro Lima. Composing abstractions of hybrid systems. In Claire Tomlin and Mark R. Greenstreet, editors, *Hybrid Systems: Computation and Control*, Lecture Notes in Computer Sience, pages 436–450. Springer-Verlag, 2002.
[45] C. Tomlin, G. J. Pappas, and S. Sastry. Conflict resolution for air traffic management : A study in muti-agent hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):509–521, April 1998.
[46] Glynn Winskel and Mogens Nielsen. Models for concurrency. In Abramsky, Gabbay, and Maibaum, editors, *Handbook of Logic and Foundations of Theoretical Computer Science*, volume 4. Oxford University Press, London, 1994.

Appendix A - Functions, partial functions and monoids

Functions and partial functions. We start by reviewing some facts regarding functions to set notation. Let $f : A \longrightarrow B$ be a map, if S is a subset of A we denote by $f(S)$ the subset of B defined by:

$$f(S) = \bigcup_{s \in S} f(s) \tag{6.1}$$

We also use the set notation $f^{-1}(b)$ to refer to all points $a \in A$ such that $f(a) = b$ and if S is a subset of B we denote by $f^{-1}(S)$ the set:

$$f^{-1}(S) = \bigcup_{s \in S} f^{-1}(s) \tag{6.2}$$

Given maps $f : A \longrightarrow B$ and $g : C \longrightarrow D$, we represent by $f \times g : A \times C \longrightarrow B \times D$ the map defined by $(a, c) \mapsto (f(a), g(c))$ for every $(a, c) \in A \times C$.

We now introduce some ideas regarding partially defined maps. Given a partially defined map $f : A \longrightarrow B$, we denote the subset of A for which f is defined by $f^{-1}(B)$. Furthermore, given another partially defined map $g : A \longrightarrow B$, we shall consider the two maps equal iff $g^{-1}(B) = f^{-1}(B)$ and $g\big|_{g^{-1}(B)} = f\big|_{f^{-1}(B)}$. Intuitively, two partially defined maps are equal when they are defined on the same subset of A and when restricted to that set, they are equal as ordinary maps. Composition is defined for partially defined maps $f : A \longrightarrow B$ and $g : B \longrightarrow C$ providing $g \circ f : A \longrightarrow C$ defined on $(g \circ f)^{-1}(C) = f^{-1}(g^{-1}(C))$. We note that composition of partially defined maps is still an associative operation. We also extend the notion of restriction of a function to this context. For a given (partially defined or not) function $f : A \longrightarrow B$ and a set $C \subseteq f^{-1}(B)$, we denote that by $f|_C : A \longrightarrow B$ the partial function defined by:

$$\begin{array}{ll} f|_C(a) = f(a) & \text{for every } a \in C \\ \text{undefined} & \text{otherwise} \end{array}$$

Monoids. A monoid is a triple $(\mathcal{M}, \cdot, \varepsilon)$ where $\mathcal{M}$ is a set closed under the associative operation $\cdot : \mathcal{M} \times \mathcal{M} \longrightarrow \mathcal{M}$ and ε is a special element of $\mathcal{M}$ called identity. This element satisfies $\varepsilon \cdot m = m \cdot \varepsilon = m$ for any $m \in \mathcal{M}$. We will usually denote $m_1 \cdot m_2$ simply by $m_1 m_2$ and refer to the monoid simply as $\mathcal{M}$. Given two elements m_1 and m_2 from $\mathcal{M}$ we say that m_1 is a prefix of m_2 iff there exists another $m \in \mathcal{M}$ such that $m_1 m = m_2$.

Given two monoids $(\mathcal{M}_X, \cdot, \varepsilon_X)$ and $(\mathcal{M}_Y, \cdot, \varepsilon_Y)$ we denote by $\mathcal{M}_X \otimes \mathcal{M}_Y$ the direct product of $\mathcal{M}_X$ and $\mathcal{M}_Y$. The direct product, which is still a monoid, is defined by the set $\mathcal{M}_X \times \mathcal{M}_Y$ equipped with pairwise multiplication defined by:

$$(m, n)(m', n') = (mm', nn') \in \mathcal{M}_X \times \mathcal{M}_Y$$

for $(m, n), (m', n') \in \mathcal{M}_X \times \mathcal{M}_Y$ and where mm' denotes multiplication in $\mathcal{M}_X$ and nn' denotes multiplication in $\mathcal{M}_Y$. Finally, the unit in $\mathcal{M}_X \otimes \mathcal{M}_Y$ is naturally given by $(\varepsilon_X, \varepsilon_Y)$.

Appendix B - Elementary notions of category theory

Categories. In this paper we only use elementary notions of category theory. We point the reader to [21] for further details as well to [22] and [3] for a "softer" introduction to the topic. Informally speaking, a category is a universe of mathematical discourse and is perhaps better described by examples. If one is interested in group theory one would certainly work in the universe of groups and group homomorphism, whereas if one is learning elementary topology the natural universe are topological spaces and continuous maps between then. In linear algebra one deals with vector spaces and linear maps, in differential geometry with smooth manifolds and smooth maps between then, etc. This idea of universe of mathematical discourse can be formally defined as follows:

Definition 6.1 (Category)**.** A category is a tuple $(\mathcal{O}, hom, id, \circ)$ consisting of:

- A class of objects $\mathcal{O}$.
- For each pair of objects (A, B) belonging to $\mathcal{O}$, a set $hom(A, B)$. The elements of $hom(A, B)$ are called morphisms from A to B. An element of this set $f \in hom(A, B)$ is usually denoted graphically as $A \xrightarrow{f} B$.
- For each object $A \in \mathcal{O}$ a special morphism $A \xrightarrow{id_A} A$, called the identity on A.

- A binary operation which maps a pair of morphisms $(A \xrightarrow{f} B, B \xrightarrow{g} C)$ to the composite[7] $A \xrightarrow{g \circ f} C$ while satisfying:
  - Associativity: $h \circ (g \circ f) = (h \circ g) \circ f$ whenever the composition is defined.
  - Identity: for a morphism $A \xrightarrow{f} B$ we have $id_B \circ f = f = f \circ id_A$.
  - The sets $hom(A, B)$ are pairwise disjoint.

In the above examples the objects are the groups, topological spaces, etc, while the arrows are the group homomorphisms, continuous maps, etc, between them. As morphisms are displayed graphically, more elaborate relations between morphisms are usually displayed in commutative diagrams. We shall say that a diagram commutes iff the composition of morphisms in any path from one object to another object is the same. Consider for example the following diagram

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle g} \\ C & \xrightarrow[j]{} & D \end{array} \tag{6.3}$$

where commutativity simply means that the two existing paths from A to D are equal, that is $g \circ f = j \circ h$.

Products. Let A and B be objects in a category. The product of A and B is the triple (C, π_A, π_B) such that for any other triple (C', π'_A, π'_B) there exists one and only one morphism η making the following diagram commutative:

$$\begin{array}{ccccc} A & \xleftarrow{\pi_A} & C & \xrightarrow{\pi_B} & B \\ & {\scriptstyle \pi'_A}\nwarrow & \uparrow{\scriptstyle \eta} & \nearrow{\scriptstyle \pi'_B} & \\ & & C' & & \end{array} \tag{6.4}$$

Note that the product captures the relevant notion of product with respect to the corresponding category. The product on the category of sets and maps between them is the usual Cartesian product, while in the category of groups is the direct product, in the category of topological spaces is the Cartesian product of the supports equipped with the product topology, etc. Reversing the arrows in diagram (6.4) leads to the dual notion of coproduct.

Equalizers. Let g and h be morphisms in a category. The equalizer of g and h is the morphism f satisfying $g \circ f = h \circ f$ and such that for any other morphism f' satisfying $g \circ f' = h \circ f'$ there is one and only one morphism $\overline{f}$ such that the following diagram commutes:

$$\begin{array}{ccccc} A & \xrightarrow{f} & B & \underset{h}{\overset{g}{\rightrightarrows}} & C \\ {\scriptstyle \overline{f}}\uparrow & \nearrow{\scriptstyle f'} & & & \\ A' & & & & \end{array}$$

[7] Note that composition of f and g is only defined if the target of f equals the source of g.

GRASP LAB, 3401 Walnut Street, Suite 300C, University of Pennsylvania, Philadelphia, PA 19104-6228

E-mail address: tabuadap@seas.upenn.edu

Department of Electrical Engineering, 200 South 33rd Street, University of Pennsylvania, Philadelphia, PA 19104

E-mail address: pappasg@ee.upenn.edu

Instituto de Sistemas e Robótica, Instituto Superior Técnico, Torre Norte - Piso 6, Av. Rovisco Pais, 1049-001 Lisboa - Portugal

E-mail address: pal@isr.ist.utl.pt