2013 13th International Conference on Application of Concurrency to System Design

Tag Machines for Modeling Heterogeneous Systems

Thi Thieu Hoa Le, Roberto Passerone
DISI, University of Trento, Italy

Uli Fahrenberg, Axel Legay
INRIA/IRISA, Rennes, France

***Abstract*—Embedded systems are often composed from components of very different natures, e.g., mechanical and electronic. Composition of heterogeneous components is generally not well-defined, making design and verification difficult. Denotational mathematical frameworks for reasoning effectively on heterogeneous composition have recently been made available. In this work, we propose an operational version of this formalism, based on tag machines, that can represent heterogeneous composition, and we provide conditions under which the heterogeneous composition can be captured soundly and completely. We have implemented our operational framework in a prototype tool which we use for experimental evaluation.**

***Keywords*-heterogeneity, tag machines, simulation**

I. Introduction

Heterogeneity is a typical characteristic of embedded systems, which are composed from parts of different natures, such as mechanical, electronic and software components, interacting in feedback to optimize the system performance. To deal with such heterogeneity, several modeling frameworks have been proposed, which are geared towards the representation and simulation of heterogeneous systems, e.g., Ptolemy II [1] or Metropolis [2]. Recently, Benveniste et al. [3] have proposed a generic denotational framework based on *tagged events* where different notions of time and interaction paradigm, including physical time, logical time (synchronous and asynchronous), precedence relations, etc., can be captured and related by mapping tagged events over a common tag structure. The operational concept of Tag Machine [4] (TM), introduced earlier by the same authors, can be used to represent tag systems in a *homogeneous* context.

In this paper, we extend tag machines to the *heterogeneous* context. In particular, we study the relation between composition of denotational tag systems with that of tag machines. Our first contribution is to review and correct certain aspects of tag machines, and to provide conditions under which the operational model can fully and compositionally capture the denotational representation. Our second contribution is a simulation engine that supports heterogeneous tag machines, with which we experimentally evaluate our results on a significant case study.

The rest of the paper is organized as follows. In Section II, we recall notions of denotational tag systems and their composition. In Section III, we describe how tag machines are extended to represent heterogeneous tag systems. We discuss soundness and completeness of the TM composition in Section IV and V. Finally, we show the application of our prototype tool to an automotive use case in Section VI.

Related Work. The theory of heterogeneous systems has been evolving actively in the last decade to assist embedded systems designers in dealing with composition of components with various Models of Computation and Communication (MoCC). Handling heterogeneous MoCC can be done strictly hierarchically in the pioneering framework of Ptolemy II [1], meaning that each level of the hierarchy is homogeneous while different interaction mechanisms are allowed to be specified at different levels in the hierarchy. This and other similar frameworks [2], [5] are oriented towards the representation and simulation of heterogeneous systems but lack formal semantics.

Another framework based on tags [3] is instead oriented towards the formal verification and analysis of those systems. This framework was inspired by the Lee and Sangiovanni-Vincentelli formalism of *tagged signal models* [6], which has been long advocated as a unified modeling framework capable of capturing heterogeneous MoCC. In both models, tags play the important role of capturing various notions of time, where each tag system has its own tag structure that can be used to express an MoCC. By applying mappings between different tag structures, the authors define how to compose heterogeneous systems. Tag machines [4] are subsequently introduced as finite representations of tag systems, yet only their homogeneous composition has been defined. Tag machines are quite expressive. For instance, they have been applied to model a job-shop specification [7] where any trace of the composite tag machine from the start to the final state results in a valid job-shop schedule. Alternatively, tag systems can be represented by functional actors forming a Kleene algebra [8]. The approach is similar to that of Liu et al. [1] in that both use actors to represent basic components.

II. Denotational Tag Systems

We use denotational tag systems as our semantic domain [3], [6]. In intuitive terms, a tag system is a representation of the behaviors of a component in terms of sets of events that take place at its interface, intended as a collection of visible ports. Tags, which are associated to every event, characterize the temporal evolution of the behaviors.

By changing the structure of tags, one can choose among different notions of time. Formally, a *tag structure* $\mathcal{T}$ is a pair $(T, \leq)$ where T is a set of *tags* and $\leq$ is a partial order on the tags. The ordering among tags is used to resolve the ordering among events at the system interface. For instance, by using the set of real numbers as tags, with their usual ordering, one can place events anywhere in real time. Conversely, a set of partially ordered symbolic tags can be used to express precedence between events in a branching-time setting.

Events occur at the interface of a component. A component exposes a set V of *variables* (or *ports*) which can take values from a set D. An *event* is a snapshot of a variable state, capturing the variable value at some point in time. Formally, an *event* e on a variable $v \in V$ is a pair (τ, d) of a tag $\tau \in T$ and a value $d \in D$.

The simplest way of characterizing a behavior is as a collection of events for each variable. In this work, we are interested in constructing behaviors incrementally, using an executable model. For this reason, we index the events of a variable into a sequence, with the understanding that events later in the sequence have larger tags [3]. A behavior for a variable v is thus a function $\mathbb{N} \mapsto (T \times D)$. A behavior σ for a component assigns a sequence of events to every variable in V, and is then a function $\sigma \in V \mapsto (\mathbb{N} \mapsto (T \times D))$.

A component P with tag structure $\mathcal{T}$, or *tag system*, is then a tuple $P = (V, \mathcal{T}, \Sigma)$, where Σ is a set of behaviors over the set of variables V. Individual events of a behavior $\sigma \in \Sigma$ are identified by the tuple (v, n, τ, d), capturing the n-th occurrence of variable v as a pair of a tag τ and a value d. In the following, we denote with $\Sigma(V, \mathcal{T})$ the universe of all behaviors over a set of variables V and tag structure $\mathcal{T}$.

Homogeneous Composition. Combining tag systems over the same tag structure amounts to considering only those behaviors which are consistent with every component. When the sets of variables coincide, this operation corresponds to taking the intersection of the behaviors of all components. When the sets of variables are not the same, instead, two behaviors are considered consistent if they agree on the shared variables. In this case, we say that the behaviors are *unifiable*. Composition consists in retaining all and only the unifiable behaviors.

Formally, let $P_1 = (V_1, \mathcal{T}, \Sigma_1)$ and $P_2 = (V_2, \mathcal{T}, \Sigma_2)$ be two tag systems over the same tag structure $\mathcal{T}$. Two behaviors $\sigma_1 \in \Sigma_1$ and $\sigma_2 \in \Sigma_2$ are *unifiable*, written $\sigma_1 \bowtie \sigma_2$, whenever $\sigma_1|_{V_1 \cap V_2} = \sigma_2|_{V_1 \cap V_2}$, where $\sigma|_W$ denotes the restriction of behavior σ to the variables in set W. When unifiable, we may construct a new behavior $\sigma = \sigma_1 \sqcup \sigma_2$ on the set of variables $V_1 \cup V_2$ as the combination of the two behaviors:

$$\sigma(v) = (\sigma_1 \sqcup \sigma_2)(v) \overset{\text{def}}{=} \begin{cases} \sigma_1(v) & \text{for } v \in V_1, \\ \sigma_2(v) & \text{for } v \in V_2. \end{cases}$$

Composition for *homogeneous* tag systems, i.e., tag systems over the same tag structure, is therefore defined as follows.

Definition 1 (Composition [3])**.** The composition P of two tag systems $P_1 = (V_1, \mathcal{T}, \Sigma_1)$ and $P_2 = (V_2, \mathcal{T}, \Sigma_2)$, written $P = P_1 \parallel P_2$, is the tag system $P = (V_1 \cup V_2, \mathcal{T}, \Sigma_1 \wedge \Sigma_2)$, where

$$\Sigma_1 \wedge \Sigma_2 \overset{\text{def}}{=} \{\sigma_1 \sqcup \sigma_2 : \sigma_1 \in \Sigma_1 \wedge \sigma_2 \in \Sigma_2 \wedge \sigma_1 \bowtie \sigma_2\}.$$

An alternative definition uses the inverse of the restriction operator $\sigma|_W$, or inverse projection, to equalize the variables of the behaviors. If σ_1 is a behavior on variables V_1, its inverse projection to the set $V = V_1 \cup V_2$ is the set of behaviors $\sigma \in \Sigma(V)$ whose restriction is σ_1:

$$\text{proj}_V^{-1}(\sigma_1) = \{\sigma \in \Sigma(V, \mathcal{T}) : \sigma|_{V_1} = \sigma_1\}.$$

Inverse projection is naturally extended to sets of behaviors. Hence, $\Sigma_1 \wedge \Sigma_2$ can also be written as

$$\Sigma_1 \wedge \Sigma_2 \overset{\text{def}}{=} \text{proj}_{V_1 \cup V_2}^{-1}(\Sigma_1) \cap \text{proj}_{V_1 \cup V_2}^{-1}(\Sigma_2),$$

which makes the intersection operator involved with composition explicit.

Heterogeneous Composition. When the tag systems have different tag structures, we must equalize also the set of tags. This is done by mapping the tag structures onto a third tag structure that functions as a common domain. The mappings are called *tag morphisms* and must preserve the order.

Definition 2 (Tag morphism [3])**.** Let $\mathcal{T}$ and $\mathcal{T}'$ be two tag structures. A *tag morphism* from $\mathcal{T}$ to $\mathcal{T}'$ is a total map $\rho: \mathcal{T} \mapsto \mathcal{T}'$ such that $\forall \tau_1, \tau_2 \in \mathcal{T}, \tau_1 \leq \tau_2 \Rightarrow \rho(\tau_1) \leq \rho(\tau_2)$.

Here, the tag orders must be taken on the respective domains. Using tag morphisms, we can turn a T-behavior $\sigma \in V \mapsto (\mathbb{N} \mapsto (T \times D))$ into a T'-behavior $\sigma_\rho \in V \mapsto (\mathbb{N} \mapsto (T' \times D))$ by simply replacing all tags τ in σ with the image $\rho(\tau)$. Abusing the function composition operator $\circ$, we may alternatively refer to σ_ρ as $\sigma \circ \rho$.

Unification of heterogeneous behaviors can be done on the common tag structure. Let $P_1 = (V_1, \mathcal{T}_1, \Sigma_1)$ and $P_2 = (V_2, \mathcal{T}_2, \Sigma_2)$ be two tag systems, and let $\rho_1 : \mathcal{T}_1 \mapsto \mathcal{T}$ and $\rho_2 : \mathcal{T}_2 \mapsto \mathcal{T}$ be two tag morphisms into a tag structure $\mathcal{T}$. We say that two behaviors $\sigma_1 \in \Sigma_1$ and $\sigma_2 \in \Sigma_2$ are *unifiable in the heterogeneous sense*, written $\sigma_1 \;{}_{\rho_1}\!\!\bowtie_{\rho_2} \sigma_2$, if and only if $(\sigma_1 \circ \rho_1) \bowtie (\sigma_2 \circ \rho_2)$. When σ_1 and σ_2 are unifiable, we may construct the unified behavior σ over $\mathcal{T}$ as usual, by considering the corresponding behaviors in $\mathcal{T}$

$$\sigma = (\sigma_1 \circ \rho_1) \sqcup (\sigma_2 \circ \rho_2),$$

and hence build the composed tag system $P = (V, \mathcal{T}, \Sigma)$ over the common tag structure $\mathcal{T}$, where

$$\Sigma \overset{\text{def}}{=} \{(\sigma_1 \circ \rho_1) \sqcup (\sigma_2 \circ \rho_2) : \sigma_1 \in \Sigma_1 \wedge \sigma_2 \in \Sigma_2 \wedge \sigma_1 \;{}_{\rho_1}\!\!\bowtie_{\rho_2} \sigma_2\}.$$

It is convenient, however, to retain some information of the original tag structures in the composition, since they are often referred to in the heterogeneous composition, as we will see in the sequel. To do so, we construct the composition over the fibered product [3] $\mathcal{T}_1 \; {}_{\rho_1}\!\times_{\rho_2} \mathcal{T}_2 = (T_1 \; {}_{\rho_1}\!\times_{\rho_2} T_2, \leq)$ of the original tag structures, extending the order component-wise:

$$(\tau_1, \tau_2) \leq (\tau_1', \tau_2') \iff \tau_1 \leq \tau_1' \wedge \tau_2 \leq \tau_2'.$$

where $T_1 \, {}_{\rho_1}\!\times_{\rho_2} T_2 = \{(\tau_1, \tau_2) \in T_1 \times T_2 : \rho_1(\tau_1) = \rho_2(\tau_2)\}$. We denote by $\sigma|_{V_1, \mathcal{T}_1}$ the restriction of σ to the variables in V_1 and to the element $\mathcal{T}_1$ of the fibered product.[1] With this notion, we can define the *heterogeneous composition*.

Definition 3 (Heterogeneous composition [3])**.** Let $P_1 = (V_1, \mathcal{T}_1, \Sigma_1)$ and $P_2 = (V_2, \mathcal{T}_2, \Sigma_2)$ be tag systems and let $\rho_1 : \mathcal{T}_1 \mapsto \mathcal{T}$ and $\rho_2 : \mathcal{T}_2 \mapsto \mathcal{T}$ be tag morphisms. The *heterogeneous composition* $P = P_1 \; {}_{\rho_1}\!\|_{\rho_2} \; P_2$ is the tag system $P = (V_1 \cup V_2, \mathcal{T}_1 \; {}_{\rho_1}\!\times_{\rho_2} \mathcal{T}_2, \Sigma_1 \; {}_{\rho_1}\!\wedge_{\rho_2} \Sigma_2)$, where

$$\Sigma_1 \; {}_{\rho_1}\!\wedge_{\rho_2} \Sigma_2 \overset{\text{def}}{=} \{\sigma \in \Sigma(V, \mathcal{T}_1 \; {}_{\rho_1}\!\times_{\rho_2} \mathcal{T}_2) : \sigma|_{V_1, \mathcal{T}_1} \in \Sigma_1 \wedge \sigma|_{V_2, \mathcal{T}_2} \in \Sigma_2\}.$$

III. Tag Machines

Tag machines [4] (TM) have been introduced to represent tag systems in a homogeneous context. Since our aim is to provide an operational representation for heterogeneous systems, we extend the TM formalism to encompass the heterogeneous context.

In order to construct behaviors, the transitions of a tag machine must be able to *increment* time, i.e., to update the tags of the events. An operation of *tag concatenation* on a tag structure is used to accomplish this.

Definition 4 (Algebraic tag structure [4])**.** An *algebraic tag structure* is a tag structure $\mathcal{T} = (T, \leq, \cdot)$ where $\cdot$ is a binary operation on T called *concatenation*, such that:

i) $(T, \cdot)$ is a monoid with identity element $\hat{\imath}$
ii) $\forall \tau_1, \tau_1', \tau_2, \tau_2' \in T : \tau_1 \leq \tau_1', \tau_2 \leq \tau_2' \Rightarrow \tau_1 \cdot \tau_2 \leq \tau_1' \cdot \tau_2'$
iii) $\exists \epsilon \in T : \forall \tau \in T : \epsilon \leq \tau \wedge \epsilon \cdot \tau = \tau \cdot \epsilon = \epsilon$.

We will organize tags in *tag vectors* $\vec{\tau} = (\tau^{v_1}, \ldots, \tau^{v_n})$, where n is the number of variables in V. During transitions, tag vectors evolve according to a matrix $\mu : V \times V \mapsto T$ called a *tag piece* [4]. Given a tag vector $\vec{\tau}$ and a tag piece μ, the new tag vector is $\vec{\tau}_\mu = \vec{\tau} \cdot \mu$ given by

$$\tau_\mu^v \overset{\text{def}}{=} \max_{u \in V} (\tau^u \cdot \mu(u, v)),$$

where the maximum is taken with respect to the tag ordering.

In practice, one concatenates each element of the tag vector with the tags on a column of μ, and then takes the largest value; thus the new value of any tag may depend on the tag increments on the events of the other variables. As the order is partial, the maximum may not exist, in which case the operation is not defined.

[1]The restriction to $\mathcal{T}_1$ can be accomplished using a tag morphism $\pi : \mathcal{T}_1 \; {}_{\rho_1}\!\times_{\rho_2} \mathcal{T}_2 \mapsto \mathcal{T}_1$ with $\pi((\tau_1, \tau_2)) = \tau_1$.

Intuitively, a tag piece μ represents increments in all variable tags over a transition and provides a way to operationally renew them. To represent also changes in variable values, μ can be labeled with a partial assignment $\nu : V \to D$, which assigns new values to the variables. We say that a labeled tag piece μ has an event for all variables for which ν is defined, and denote by $\mathfrak{Dom}(\nu)$ the domain of ν. In the following, we assume that tag pieces are always labeled and implicitly associate a labeling function ν to a tag piece μ.

Example 1. The algebraic tag structure $(\mathbb{N} \cup \{-\infty\}, \leq, +)$ can be used to capture logical time by structuring tag pieces μ so that they represent an integer increment of 1 as follows:

$$\mu(u, v) = \begin{cases} 0 & \text{if } u = v \text{ and } \nu \text{ is not defined on } v \\ -\infty & \text{if } u \neq v \text{ and } \nu \text{ is not defined on } v \\ 1 & \text{if } \nu \text{ is defined on } v \end{cases}.$$

The least element $\epsilon = -\infty$ is used to cancel the contribution of an entry in the tag vector. With these definitions, every time a new value must be assigned to a variable (i.e., when $\nu(v)$ is defined), the tag is also incremented by 1. Otherwise, the tag is left unchanged and no new event is generated. For instance, $\left[\begin{smallmatrix} 1 & 3 \end{smallmatrix}\right] \cdot \left[\begin{smallmatrix} 0 & 1 \\ \epsilon & 1 \end{smallmatrix}\right] = \left[\begin{smallmatrix} 1 & 4 \end{smallmatrix}\right]$. The tag of the second variable is increased by 1 since the tag piece has an event for it.

Likewise, $(\mathbb{R}_+ \cup \{-\infty\}, \leq, +)$ can capture physical time.

A tag machine M is a finite automaton where transitions are marked by labeled tag pieces, or simply *labels*. Our definition below differs from that proposed by Benveniste et al. [4] for certain simplifications and for the addition of a set of accepting states.

Definition 5. A *tag machine* M is a tuple $(V, \mathcal{T}, S, s_0, F, L, E)$, where:

- V is a set of variables,
- $\mathcal{T}$ is an algebraic tag structure,
- S is a finite set of states and $s_0 \in S$ is the initial state,
- $F \subseteq S$ is a set of accepting states,
- L is a set of labeled tag pieces defined on V and $\mathcal{T}$,
- $E \subseteq S \times L \times S$ is the transition relation.

A run r of the tag machine is a sequence of states and transitions $r : s_0 \xrightarrow{\mu_0} s_1 \xrightarrow{\mu_1} s_2 \ldots s_{m-1} \xrightarrow{\mu_{m-1}} s_m$ such that $s_m \in F$ and for all i, $1 \leq i \leq m$, $(s_{i-1}, \mu_{i-1}, s_i) \in E$. Intuitively, a tag machine is used to construct a behavior by following its labeled transitions over a run, and applying the tag pieces sequentially to an initial vector of tags. A new event is added to the behavior whenever a new value is assigned by the label function ν_i. In order to formalize the language of a tag machine, we must keep track of both the tags and the number of events that have occurred for each variable. Thus, for every state s_i along run r, we define a tag vector $\vec{\tau}_i$ computed by accumulating the tag pieces:

$$\vec{\tau}_{i+1} = \vec{\tau}_i \cdot \mu_i,$$

and an index vector $\vec{k_i}$ computed by updating the event index at every new event:

$$\vec{k}_{i+1}(v) = \begin{cases} \vec{k}_i(v) & \text{if } v \notin \mathfrak{Dom}(\nu_i) \\ \vec{k}_i(v)+1 & \text{if } v \in \mathfrak{Dom}(\nu_i) \end{cases}.$$

For state s_0, the tag vector is initialized to the identity element $\hat{\imath}$, while the index vector is initialized to 0. The behavior $\sigma(r)$[2] of a run r is constructed incrementally by starting from an empty behavior σ_0 and computing:

$$\sigma_{i+1}(v,k) = \begin{cases} \sigma_i(v,k) & \text{if } v \notin \mathfrak{Dom}(\nu_i) \\ \sigma_i(v,k) & \text{if } v \in \mathfrak{Dom}(\nu_i) \wedge k < \vec{k}_i(v) \\ (\vec{\tau}_{i+1}(v), \nu_i(v)) & \text{if } v \in \mathfrak{Dom}(\nu_i) \wedge k = \vec{k}_i(v) \end{cases}$$

A run r of M is *valid* if the concatenation is always defined along the run, and if $s_m \in F$. The language $\mathcal{L}(M)$ of M is given by the behaviors of all its valid runs.

A. Composition of Tag Machines

Tag machines are composed in parallel by taking a form of product between their structures. Synchronization occurs by sharing variables. In particular, over every transition, the tag machines involved in the composition must agree on the tag increment and on the value of the shared variables.

We first examine the composition of homogeneous tag machines, by adapting the original definition [4]. Consider two tag machines $M_1 = (V_1, \mathcal{T}, S_1, s_{01}, F_1, L_1, E_1)$ and $M_2 = (V_2, \mathcal{T}, S_2, s_{02}, F_2, L_2, E_2)$ defined over the same tag structure $\mathcal{T}$. Two labeled tag pieces $\mu_1 \in L_1$ and $\mu_2 \in L_2$ are *unifiable*, written $\mu_1 \bowtie \mu_2$, if and only if they are the same on the shared variables. That is, if we denote the set of shared variables with $W = V_1 \cap V_2$, then for all pairs $(w,v) \in W \times W$,

$$\mu_1(w,v) = \mu_2(w,v),$$
$$\nu_1(v) = \nu_2(v).$$

When unifiable, their unification $\mu = \mu_1 \sqcup \mu_2$ is given by

$$\mu(w,v) = \begin{cases} \mu_1(w,v) & \text{if } (w,v) \in V_1 \times V_1 \\ \mu_2(w,v) & \text{if } (w,v) \in V_2 \times V_2 \\ \epsilon & \text{otherwise} \end{cases},$$
$$\nu(v) = \begin{cases} \nu_1(v) & \text{if } v \in V_1 \\ \nu_2(v) & \text{if } v \in V_2 \end{cases}.$$

Homogeneous composition can then be defined as follows:

Definition 6 (Homogeneous composition)**.** The parallel composition of homogeneous machines M_1 and M_2 is the machine $M = M_1 \parallel M_2 = (V, \mathcal{T}, S, s_0, F, L, E)$ such that

- $V = V_1 \cup V_2$,
- $S = S_1 \times S_2$, $s_0 = (s_{01}, s_{02})$, $F = F_1 \times F_2$,
- $L = \{\mu_1 \sqcup \mu_2 : \mu_1 \in L_1 \wedge \mu_2 \in L_2 \wedge \mu_1 \bowtie \mu_2\}$,
- $E = \{((s_1, s_2), \mu_1 \sqcup \mu_2, (s_1', s_2')) : (s_1, \mu_1, s_1') \in E_1 \wedge (s_2, \mu_2, s_2') \in E_2 \wedge \mu_1 \bowtie \mu_2\}$.

[2]We sometimes refer to $\sigma(r)$ as $\sigma(\omega)$ where $\omega = \mu_0\mu_1 \ldots \mu_{m-1}$

Heterogeneous tag machines can be composed if there exists a pair of morphisms which map the tag structures $\mathcal{T}_1$ and $\mathcal{T}_2$ to a common tag structure $\mathcal{T}$, preserving the concatenation operator. We refer to such morphisms as *algebraic* morphisms.

Definition 7. A tag morphism $\rho : \mathcal{T} \mapsto \mathcal{T}'$ is *algebraic* if $\rho(\hat{\imath}_{\mathcal{T}}) = \hat{\imath}_{\mathcal{T}'}$, $\rho(\epsilon_{\mathcal{T}}) = \epsilon_{\mathcal{T}'}$, and $\rho(\tau_1 \cdot \tau_2) = \rho(\tau_1) \cdot \rho(\tau_2)$ for all $\tau_1, \tau_2 \in \mathcal{T}$.

The newly-composed TM will be defined on a unified tag structure and a unified label set. Referring to the previous notation, two tag pieces μ_1 and μ_2 are *unifiable* under morphisms ρ_1 and ρ_2, written $\mu_1 \;{}_{\rho_1}\!\bowtie_{\rho_2}\; \mu_2$, whenever for all pairs $(w,v) \in W \times W$,

$$\rho_1(\mu_1(w,v)) = \rho_2(\mu_2(w,v)),$$
$$\nu_1(v) = \nu_2(v).$$

When unifiable, their unification $\mu = \mu_1 \;{}_{\rho_1}\!\sqcup_{\rho_2}\; \mu_2$ defined over the tag structure $\mathcal{T}_1 \,{}_{\rho_1}\!\times_{\rho_2} \mathcal{T}_2$ is a set of pieces given by

$$\mu(w,v) = \begin{cases} (\mu_1(w,v), \mu_2(w,v)) & \text{if } (w,v) \in W \times W \\ (\mu_1(w,v), \tau_2) & \text{if } w \in V_1, v \in V_1 \setminus V_2 \\ (\mu_1(w,v), \tau_2) & \text{if } w \in V_1 \setminus V_2, v \in V_1 \\ (\tau_1, \mu_2(w,v)) & \text{if } w \in V_2 \setminus V_1, v \in V_2 \\ (\tau_1, \mu_2(w,v)) & \text{if } w \in V_2, v \in V_2 \setminus V_1 \\ (\epsilon_{\mathcal{T}_1}, \epsilon_{\mathcal{T}_2}) & \text{otherwise} \end{cases}$$

where $\tau_2 \in T_2$ is such that $\rho_2(\tau_2) = \rho_1(\mu_1(w,v))$, and similarly $\tau_1 \in T_1$ is such that $\rho_1(\tau_1) = \rho_2(\mu_2(w,v))$. The labeling function is the same as in the homogeneous case:

$$\nu(v) = \begin{cases} \nu_1(v) & \text{if } v \in V_1 \\ \nu_2(v) & \text{if } v \in V_2 \end{cases}.$$

The composition $M = M_1 \;{}_{\rho_1}\!\|_{\rho_2}\; M_2$ of heterogeneous tag machines can then be defined exactly as in Definition 6, having replaced the operators for the unification of the tag pieces on the transition with the heterogeneous ones.

Definition 8 (Heterogeneous composition)**.** The parallel composition of heterogeneous machines M_1 and M_2 under morphisms ρ_1 and ρ_2 is the machine $M = M_1 \;{}_{\rho_1}\!\|_{\rho_2}\; M_2 = (V, \mathcal{T}_1 \,{}_{\rho_1}\!\times_{\rho_2}\, \mathcal{T}_2, S, s_0, F, L, E)$ such that

- $V = V_1 \cup V_2$,
- $S = S_1 \times S_2$, $s_0 = (s_{01}, s_{02})$, $F = F_1 \times F_2$,
- $L = \{\mu_1 \,{}_{\rho_1}\!\sqcup_{\rho_2} \mu_2 : \mu_1 \in L_1 \wedge \mu_2 \in L_2 \wedge \mu_1 \;{}_{\rho_1}\!\bowtie_{\rho_2}\; \mu_2\}$,
- $E = \{((s_1, s_2), \mu_1 \;{}_{\rho_1}\!\sqcup_{\rho_2}\; \mu_2, (s_1', s_2')) : (s_1, \mu_1, s_1) \in E_1 \wedge (s_2, \mu_2, s_2) \in E_2 \wedge \mu_1 \;{}_{\rho_1}\!\bowtie_{\rho_2}\; \mu_2\}$

IV. Interoperable Tag Machines and Composition Soundness

Ideally, we would like there to be a direct correspondence between tag systems and tag machines. So, if Σ_i is the language of machine M_i, and if Σ is the language of the composition $M = M_1 \;{}_{\rho_1}\!\|_{\rho_2}\; M_2$, we expect that every behavior of Σ be obtained by composing some pair of

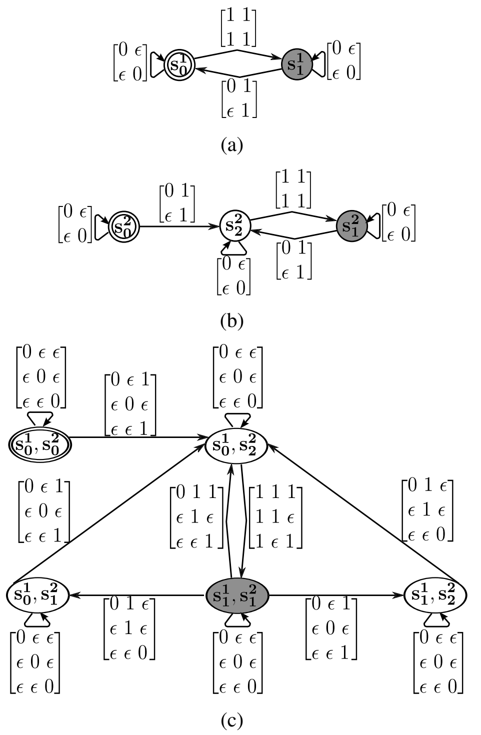


Figure 1. Tag machines accepting (a) Σ_1 (b) Σ_2 (c)$\Sigma_1 \wedge \Sigma_2$

behaviors from Σ_i. When this is the case, we say that composition is sound. Example 2 shows that this property generally does not hold even for homogeneous systems.

Example 2. We consider two sets of behaviors Σ_1 and Σ_2 defined on two sets, $V_1 = \{x, y\}$ and $V_2 = \{x, z\}$ respectively, of variables with values in $D = \{true\}$. Since D is a singleton set, we shall omit mentioning the variable value in the rest of the example. These behavioral sets are expressed formally as follows. Let $\sigma_i \in \Sigma_i$ and $\mathfrak{enum}_{\sigma_i}(v_i)$ be the total number of events on variable $v_i \in V_i$ in behavior σ_i where $i \in \{1, 2\}$ and let $k \geq 1$:

$$\Sigma_1 : \begin{cases} \sigma_1(x, k) = 2 * k - 1 \\ \sigma_1(y, k) = k \\ \mathfrak{enum}_{\sigma_1}(y) = 2 * \mathfrak{enum}_{\sigma_1}(x) - 1 \end{cases}$$

$$\Sigma_2 : \begin{cases} \sigma_2(x, k) = 2 * k \\ \sigma_2(z, k) = k \\ \mathfrak{enum}_{\sigma_1}(z) = 2 * \mathfrak{enum}_{\sigma_1}(x) \end{cases}$$

Let *reaction* be a maximal set of events with identical tags in a behavior, c.f. [4], these behavioral sets can then be organized in terms of successive reactions as follows:

Σ_1 :

x :	1		3		5		7		9		...
y :	1	2	3	4	5	6	7	8	9	10	...

Σ_2 :

x :		2		4		6		8		10	...
z :	1	2	3	4	5	6	7	8	9	10	...

We use the algebraic tag structure $(\mathbb{N} \cup \{\epsilon\}, \leqslant, +)$ and the tag piece structure described in Example 1 to model the behaviors in Σ_i as TMs, shown in Fig. 1, where $\epsilon \overset{\text{def}}{=} -\infty$, the row and column designation orders are (x, y) in Fig. 1(a), (x, z) in Fig. 1(b) and (x, y, z) in Fig. 1(c). The initial states are double-circled and the accepting states are shaded.

Tagging the shared variable x can depend on tagging non-shared variables y or z even though there is no real dependence between their tags. Going from s_0^1 to s_1^1, the TM 1(a) tags x and y simultaneously and equally. It then can go back to s_0^1, tagging only y and subsequently repeating the tagging cycle at this state. The TM 1(b) instead tags only z initially and goes to s_2^2. It then tags both x and z at the same time and goes to s_1^2 where it again tags only z before returning to s_2^2. It is easy to verify that TM 1(a) and 1(b) accept the behavioral sets Σ_1 and Σ_2 respectively. When composing them, the composed TM is expected not to accept any behavior since $\Sigma_1 \wedge \Sigma_2 = \emptyset$. Its set of accepted behavior is, however, not empty as shown in Fig. 1(c). This is because TM 1(a) can stay silent while TM 1(b) is tagging z. The two TMs then synchronize and tag all variables simultaneously, after which they can go on tagging their own internal variable.

We remark that the fact that TM composition is not sound has not been observed in Benveniste et.al. [4]; clearly it also holds for heterogeneous composition. The consequence is that the operational model overestimates the behaviors of composition, therefore building an abstraction. This may or may not be a problem, depending on what is done with the models. For instance, verification of safety properties would be correct, albeit less precise. On the other hand, the emergence of unexpected behaviors may adversely affect the design process, where a refinement rather than an abstraction would instead be more appropriate. It is therefore useful to look for conditions that guarantee soundness. In our example, the dependency effects of tagging non-shared variables on others, especially shared ones, are shown to be the critical factor. The cause lies in the fact that in the applications of tag pieces, the max tag evaluations for a shared variable can be different even though the pieces are unifiable. It is therefore desirable to eliminate such effects to make the composition sound. To this end, we propose an *interoperability* condition to prevent TMs from producing such effects.

The intuition behind TM interoperability is that tagging shared variables should not depend on tagging non-shared variables. A set $\overline{V} \subseteq V$ is *locally independent* in $M = (V, \mathcal{T}, S, s_0, F, L, E)$, written $\mathfrak{lind}(M, \overline{V})$, if tagging $v \in \overline{V}$ depends only on tagging other variables also in $\overline{V}$. If we define the predicate $\mathfrak{lind}(\mu, \overline{V}) \overset{\text{def}}{=} (\forall w \in V \setminus \overline{V}, \forall v \in \overline{V} : \mu(w, v) = \epsilon)$, then $\mathfrak{lind}(M, \overline{V}) \overset{\text{def}}{=} (\forall (s, \mu, s') \in E : \mathfrak{lind}(\mu, \overline{V}))$. The interoperability between M_1 and M_2 is formally defined as follows.

Definition 9. Two TMs $M_i = (V_i, \mathcal{T}_i, S_i, s_{0i}, F_i, L_i, E_i)$, for $i \in \{1,2\}$, are said to be *interoperable*, written $M_1 \bowtie M_2$, if their shared variables are locally independent in M_i:

$$M_1 \bowtie M_2 \overset{\text{def}}{=} \mathfrak{lind}(M_1, V_1 \cap V_2) \wedge \mathfrak{lind}(M_2, V_1 \cap V_2)$$

Interoperability behaves well under multiple composition.

Lemma 1. *Let $M_1, M_2, \ldots M_n$ be n pair-wise interoperable TMs, where $n \geq 2$. If M is the composition of $M_1, M_2, \ldots M_{n-1}$, then $M \bowtie M_n$.*

Proof: We prove the theorem by induction. The base case $n = 2$ is trivial, so assume now the statement holds for $k = n-1$, we prove that it also holds for $k = n$. Let V_i, μ_i be the variable set and label of M_i, μ the unification of $\mu_1, \mu_2 \ldots, \mu_{n-1}$, and $V = V_1 \cup V_2 \ldots \cup V_{n-1}$, we see that $\mathfrak{lind}(\mu, V \cap V_n)$ holds. Indeed, if $w \in V \setminus (V \cap V_n)$, then $w \in V_j \setminus (V \cap V_n)$ and hence $w \in V_j \setminus V_n$ for some j. Likewise, if $v \in V \cap V_n$, then $v \in V_r \cap V_n$ for some r. Thus, if $j = r$ then $\mu(w,v) = \epsilon_{\mathcal{T}}$ because $M_r \bowtie M_n$ implies $\mu_r(w,v) = \epsilon_{\mathcal{T}_r}$. If instead $j \neq r$, the result remains the same because the composition on labels puts $\epsilon_{\mathcal{T}}$ whenever w and v are not in the same variable set. To prove that $\mathfrak{lind}(\mu_n, V \cap V_n)$ holds, we consider $w \in V_n \setminus (V \cap V_n)$ and $v \in V_n \cap V_r$ for some r. The former implies $w \in V_n$ and $w \notin V_j$ for all j. This together with $M_n \bowtie M_r$ implies $\mu_n(w,v) = \epsilon_{\mathcal{T}}$. $\square$

Example 3. We use the algebraic tag structure in Example 2 but restructure the tag pieces so that they can represent any integer time increment $n \in \mathbb{N}$.

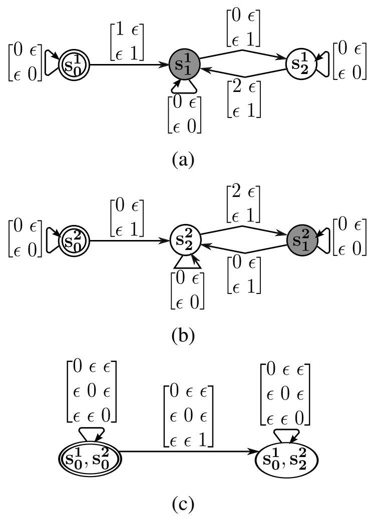
$$\mu^{uv} = \begin{cases} 0 & \text{if } u = v \text{ and } \mu \text{ has no event for } v \\ n & \text{if } u = v \text{ and } \mu \text{ has an event for } v \\ \epsilon & \text{otherwise} \end{cases}$$

Interoperable TMs representing Σ_i are depicted in Fig. 2(a) and 2(b). Tagging the shared interface variable x is now made locally independent in both machines, hence they can be composed interoperably. The composed TM is shown in Fig. 2(c) where no behavior can be accepted since the accepting state is not reachable from the initial state. This is because TM 2(a) has to stutter in s_0^1 while TM 2(b) tags its internal variable and moves to s_2^2. The two TMs then have to stutter there forever since only the stuttering labels $\left[\begin{smallmatrix} 0 & \epsilon \\ \epsilon & 0 \end{smallmatrix}\right]$ are unifiable.

Example 4. We consider a simplified version of the water controlling system proposed by Benvenuti et al. [9]. It consists of two components: a water tank and a water level controller, connected in a closed-loop fashion, c.f. Fig. 3. We assume that the water level $x(t)$ is changed linearly as follows:

$$x(t) \overset{\text{def}}{=} \begin{cases} \Delta_t * (\mathbf{f_i} - \mathbf{f_o}) & \text{when command is Open} \\ \mathbf{H} - \Delta_t * \mathbf{f_o} & \text{when command is Close} \end{cases} \quad (1)$$

where $\mathbf{f_i}$ and $\mathbf{f_o}$ denote the constant inlet and outlet flow, respectively, $\mathbf{H}$ denotes the height when the tank is full of water and Δ_t denotes the time elapsed since t_0 at which the tank reaches the maximum/minimum water level $\mathbf{H}$, i.e. $\Delta_t = t - t_0$. Let $P_1 = (V_1, \mathcal{T}_1, \Sigma_1)$ and $P_2 = (V_2, \mathcal{T}_2, \Sigma_2)$ be two tag systems representing the tank and the water controller, respectively, where $\mathcal{T}_1 = (\mathbb{R}_+ \cup \{\epsilon_1\}, +)$, $\mathcal{T}_2 = (\mathbb{N} \cup \{\epsilon_2\}, +)$, $\epsilon_1 = \epsilon_2 = -\infty$, and $V_1 = V_2 = \{\mathit{cmd}, x\}$. Variable *cmd* denotes the command values, which can be Open $(\mathbf{op})$ or Close $(\mathbf{cl})$, and variable x denotes the water level, which is of positive real type, i.e. $D_{\mathit{cmd}} = \{\mathbf{op}, \mathbf{cl}\}$ and $D_x = \mathbb{R}_+$. Assume that $\mathbf{f_i} = 2, \mathbf{f_o} = 1, \mathbf{H} = 1$, we model in this example a linear water level evolution of a water controlling system. The tank component shown in Fig. 4(a) depicts the water level linear evolution as specified in 1. Upon knowing of the tank emptiness/fullness, the controller component in Fig. 4(b) will issue an Open/Close command. Intuitively, the controller behaviors ensure that controlling commands are always issued timely (i.e., Open when the tank is empty and Close when it is full), no matter how the water level evolves, while the tank behaviors accept untimely controlling commands and allow water spillages or shortages, given that the water level evolves linearly.



$\left[\begin{smallmatrix} 0 & \epsilon \\ \epsilon & 0 \end{smallmatrix}\right]$
s_0^1
$\left[\begin{smallmatrix} 1 & \epsilon \\ \epsilon & 1 \end{smallmatrix}\right]$
s_1^1
$\left[\begin{smallmatrix} 0 & \epsilon \\ \epsilon & 1 \end{smallmatrix}\right]$
$\left[\begin{smallmatrix} 2 & \epsilon \\ \epsilon & 1 \end{smallmatrix}\right]$
s_2^1
(a)
s_0^2
$\left[\begin{smallmatrix} 0 & \epsilon \\ \epsilon & 1 \end{smallmatrix}\right]$
s_2^2
$\left[\begin{smallmatrix} 2 & \epsilon \\ \epsilon & 1 \end{smallmatrix}\right]$
$\left[\begin{smallmatrix} 0 & \epsilon \\ \epsilon & 1 \end{smallmatrix}\right]$
s_1^2
(b)
$\left[\begin{smallmatrix} 0 & \epsilon & \epsilon \\ \epsilon & 0 & \epsilon \\ \epsilon & \epsilon & 0 \end{smallmatrix}\right]$
s_0^1, s_0^2
$\left[\begin{smallmatrix} 0 & \epsilon & \epsilon \\ \epsilon & 0 & \epsilon \\ \epsilon & \epsilon & 1 \end{smallmatrix}\right]$
s_0^1, s_2^2
(c)

Figure 2. Interoperable TM accepting (a) Σ_1 (b) Σ_2 (c)$\Sigma_1 \wedge \Sigma_2$

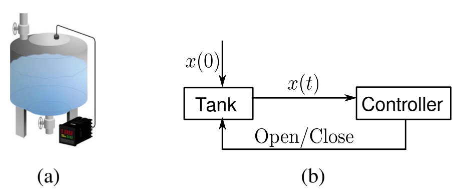
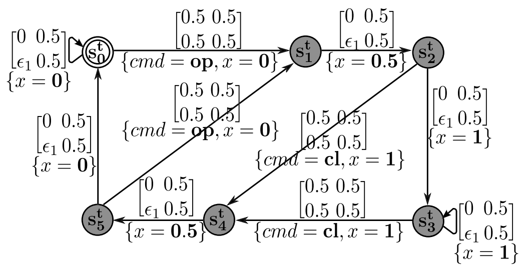
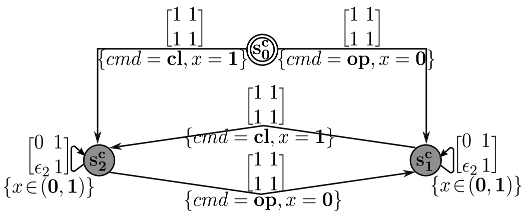


Figure 3. Water controlling system

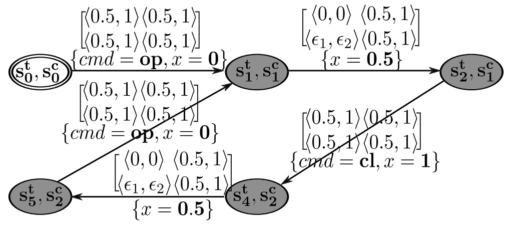
For the sake of simplicity, the events described by the tank component are timestamped periodically every 0.5 time unit. While the tank system uses physical time to stamp its behaviours, the controller system instead timestamps its events logically, described by the integer tag set $\mathbb{N}$. For the



(a)



(b)



(c)

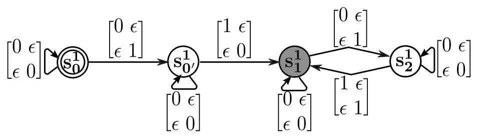
Figure 4. Interoperable TM representing (a) the tank's behaviors (b) controller's behaviors (c) the tank-controller composition's behaviors

sake of expressiveness, some of the labeled tag pieces can be represented symbolically. For example, to capture any event of variable x happening at a specific time point, we label with the tag piece capturing that time point expressions such as $x \in D_x$, meaning that in such an event x can take any value in its domain. Since the tank and the controller capture different behaviors, composing them is only possible under the presence of morphisms such as $\rho_1 : \mathcal{T}_1 \mapsto \mathcal{T}_1$ and $\rho_2 : \mathcal{T}_2 \mapsto \mathcal{T}_1$ given by $\rho_1(\tau_1) = \tau_1$, $\rho_2(\tau_2) = 0.5 * \tau_2$. Fig. 4(c) shows their composition, where accepted behaviors ensure timely controlling commands and linear water evolution.

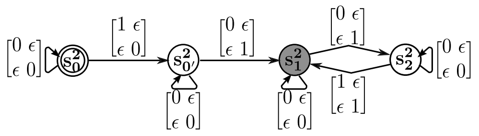
The following theorem shows that composition of interoperable TMs is sound.

Theorem 1. *For any behavior σ accepted by $M_1\ {}_{\rho_1}\|_{\rho_2}\ M_2$ where $M_1 \bowtie M_2$, there exist behaviors σ_1, σ_2 accepted by M_1, M_2 such that σ can be obtained by composing σ_1 and σ_2, i.e.* $\Sigma \subseteq \Sigma_1\ {}_{\rho_1}\wedge_{\rho_2}\ \Sigma_2$.

Proof: Let $r : s_0 \xrightarrow{\mu_1} s_1 \ldots \xrightarrow{\mu_n} s_n$ be a valid run in $M_1\ {}_{\rho_1}\|_{\rho_2}\ M_2$ where $s_k = (s_{k1}, s_{k2})$ for $0 \leq k \leq n$. There exists a valid run $r_i : s_{0i} \xrightarrow{\mu_{1i}} s_{1i} \ldots \xrightarrow{\mu_{ni}} s_{in}$ in M_i, for $i \in \{1, 2\}$, where $\mu_{k1}\ {}_{\rho_1}\!\bowtie_{\rho_2}\ \mu_{k2}$ and $\mu_k = \mu_{k1}\ {}_{\rho_1}\!\sqcup_{\rho_2}\ \mu_{k2}$. Inductively for tag vectors $\vec{\tau}_k/\vec{\tau}_{ki}$ obtained at states s_k/s_{ki} in



(a)



(b)

Figure 5. Interoperable TMs accepting Σ_1

run r/r_i we prove that first, $\rho_1(\tau^v_{k1}) = \rho_2(\tau^v_{k2})$ and second, $\tau^w_k \in \mathcal{T}_1\ {}_{\rho_1}\!\times_{\rho_2}\ \mathcal{T}_2$ and $\vec{\tau}_k|_{V_1} = \vec{\tau}_{k1}$ and $\vec{\tau}_k|_{V_2} = \vec{\tau}_{k2}$, where $v \in V_1 \cap V_2$ and $w \in V_1 \cup V_2$.

Let $v' \in V_1 \cap V_2, v_i \in V_i \setminus (V_1 \cap V_2)$ and μ^{vw} denote $\mu(v, w)$. The base case $k = 0$ is trivial since $\tau^v_{0i} = \hat{\imath}_{\mathcal{T}_i}$ by definition. Assuming the statements hold for any $0 \leq k \leq n - 1$, we prove that it also holds for $k = n$ using the following fact for the first:

$$\begin{aligned}\rho_i(\tau^v_{ni}) &= \rho_i(\max(\tau^{v'}_{(n-1)i} \cdot \mu^{v'v}_{ni}, \tau^{v_i}_{(n-1)i} \cdot \mu^{v_i v}_{ni})) \\ &= \max(\rho_i(\tau^{v'}_{(n-1)i}) \cdot \rho_i(\mu^{v'v}_{ni}), \rho_i(\tau^{v_i}_{(n-1)i}) \cdot \rho_i(\mu^{v_i v}_{ni}))\end{aligned}$$

and the interoperability condition and the inductive hypotheses for the second, e.g.

$$\begin{aligned}\tau^v_n &= \max(\tau^{v'}_{n-1} \cdot \mu^{v'v}_n, \tau^{v_1}_{n-1} \cdot \mu^{v_1 v}_n, \tau^{v_2}_{n-1} \cdot \mu^{v_2 v}_n) \\ &= \max((\tau^{v'}_{(n-1)1} \cdot \mu^{v'v}_{n1}, \tau^{v'}_{(n-1)2} \cdot \mu^{v'v}_{n2}))\end{aligned}$$

and $\rho_1(\tau^{v'}_{(n-1)1} \cdot \mu^{v'v}_{n1}) = \rho_2(\tau^{v'}_{(n-1)2} \cdot \mu^{v'v}_{n2})$. □

V. Self-synchronizing Tag Machines and Composition Completeness

To establish the correspondence between tag systems and TMs, TM compositions need to be not only sound but also complete. Composition completeness requires two unifiable behaviors to always be operationally composed, and Example 5 below shows that this property generally does not hold even for homogeneous systems. The reason of this incompleteness is due to the TM non-unique behavioral representations. Indeed, different TM runs can represent the same behavior, which in turn causes the operational composition on two behaviors to sometimes become impossible (while the denotational one may be possible) when their representations contain non-unifiable labels.

Example 5. Fig. 5 shows two different TMs representing the same set of behaviors as Σ_1, yet their composition does not accept Σ_1. This is because the TMs cannot synchronize on updating their variable tags.

As demonstrated in Example 5, the TM composition represents incompletely the tag system composition when

M_i fails to include all representations of some behavior. Therefore, it is a natural condition for completeness that M_i be *self-synchronizing*. The notion of self-synchronization for homogeneous composition [4] requires the language of $M = (V, \mathcal{T}, S, s_0, F, L, E)$ to be $\sim$-closed, where $\sim$ is a binary relation s.t. $\forall \omega, \omega' \in \Sigma(V, \mathcal{T}) : \omega \sim \omega' \Leftrightarrow \sigma(\omega) = \sigma(\omega')$.

In order to make machine M_i self-synchronize, all possible runs of any behavior of M_i must be added to the machine. In order for the TM semantics to remain unaffected, any sub-run's behavior should be excluded from the machine language. This, however, is not guaranteed in [4] because TMs there do not contain accepting states. Our TM definition (Def. 5) cares for such a need and thus can preserve the TM semantics under the self-synchronizing operation.

Self-synchronization is however not sufficient to guarantee completeness of composition, as two unifiable behaviors do not always have unifiable representations, which in turn is caused by the choice of tag structures and morphisms. We recall that events of a variable v are indexed into a sequence of $(v, 1, \tau_1, d_1), (v, 2, \tau_2, d_2) \ldots, (v, n, \tau_n, d_n)$ where $\tau_1 \leq \tau_2 \ldots \leq \tau_n$. If the tag increments between two successive events can always be mapped into the same increment under morphisms ρ_i, then completeness of composition is ensured by the following theorem.

Theorem 2. *For behaviors σ_1, σ_2 accepted by M_1, M_2 such that $\sigma_1 \;{}_{\rho_1}\!\bowtie_{\rho_2}\; \sigma_2$ where:*

i) M_1 and M_2 are self-synchronizing and $M_1 \bowtie M_2$,

ii) $\forall i \in \{1, 2\}, \forall (v, j, \tau_{ji}, d_{ji}) \in \sigma_i$:
$\exists \delta_{ji} \in \mathcal{T}_i : \tau_{ji} = \tau_{(j-1)i} \cdot \delta_{ji}$ where $\tau_{0i} \overset{\text{def}}{=} \hat{\imath}_{\mathcal{T}_i}$,

iii) $\forall i \in \{1, 2\}, \forall \delta_i \in \mathcal{T}_i, \forall (\tau_1, \tau_2), (\tau'_1, \tau'_2) \in \mathcal{T}_1 \,{}_{\rho_1}\!\times_{\rho_2} \mathcal{T}_2$:
$\tau'_i = \tau_i \cdot \delta_i \Rightarrow \exists (\delta_1, \delta_2) \in \mathcal{T}_1 \,{}_{\rho_1}\!\times_{\rho_2} \mathcal{T}_2 : \tau'_{3-i} = \tau_{3-i} \cdot \delta_{3-i}$

if a behavior $\sigma \in \Sigma(V_1 \cup V_2, \mathcal{T}_1 \,{}_{\rho_1}\!\times_{\rho_2} \mathcal{T}_2)$ is such that $\sigma|_{V_1, \mathcal{T}_1} = \sigma_1$ and $\sigma|_{V_2, \mathcal{T}_2} = \sigma_2$, then σ is also accepted by $M_1 \;{}_{\rho_1}\!\|_{\rho_2}\; M_2$, i.e. $\Sigma_1 \;{}_{\rho_1}\!\wedge_{\rho_2}\; \Sigma_2 \subseteq \Sigma$.

Proof: Let n_i be the greatest number of events of all variables in σ_i, then there exists a sequence of labels $\omega_i = \mu_{1i}\mu_{2i} \ldots \mu_{n_i i}$ where the diagonal entries of μ_{ji} are δ_{ji} specified in Theorem 2. ii and the entries outside the diagonal are all ϵ_{T_i}. At the end, we pad ω_i with $\max(n_1, n_2) - n_i$ stuttering labels. Since $\sigma(\omega_i) = \sigma_i$, self-synchronization ensures the existence of a valid run r_i in M_i over ω_i. By construction $\mu_{j1} \;{}_{\rho_1}\!\bowtie_{\rho_2}\; \mu_{j2}$, and this implies that there exists a run r in $M_1 \;{}_{\rho_1}\!\|_{\rho_2}\; M_2$ over the following sequence of labels $\omega = (\mu_{11} \;{}_{\rho_1}\!\sqcup_{\rho_2}\; \mu_{12}) \ldots (\mu_{\max(n_1,n_2)1} \;{}_{\rho_1}\!\sqcup_{\rho_2}\; \mu_{\max(n_1,n_2)2})$ where $\sigma(r)|_{V_1 \cap V_2} = \sigma|_{V_1 \cap V_2}$ since $\sigma(r)|_{V_1, \mathcal{T}_1} = \sigma_1$ and $\sigma(r)|_{V_2, \mathcal{T}_2} = \sigma_2$, $\sigma(r)|_{V_1 \setminus V_2} = \sigma|_{V_1 \setminus V_2}$ and $\sigma(r)|_{V_2 \setminus V_1} = \sigma|_{V_2 \setminus V_1}$ because of condition iii, thus $\sigma(r) = \sigma$. □

Being self-synchronizing is, however, not a trivial property for TMs in general and efficient methods of checking and providing this property is part of our on-going work.

```
TAGMACH Piston{
TAGSTRUCT MAXPLUSINTEGER;
MSTATE state;
MLABEL label;
MVAR rot:  BOOLEAN;
INIT (state = H) && (rot = FALSE);
EDGE (state == H) && (label == [[1]]) ->
     (NEXT(state) = I) && (NEXT(rot) = TRUE); ...}
```

(a)

```
MORPH Sync{
ORGVAR x1 :  MAXPLUSINTEGER;
DESVAR x2 :  MAXPLUSFLOAT;
MAP x2 = x1*0.001181;}
```

(b)

Figure 6. High-level description of (a) a piston TM (b) its morphism

VI. Implementation and Evaluation

In this section, we demonstrate the application of TM composition in dealing with the problem of heterogeneity in a practical use case of an automotive system.

A. Tag Machine Simulator

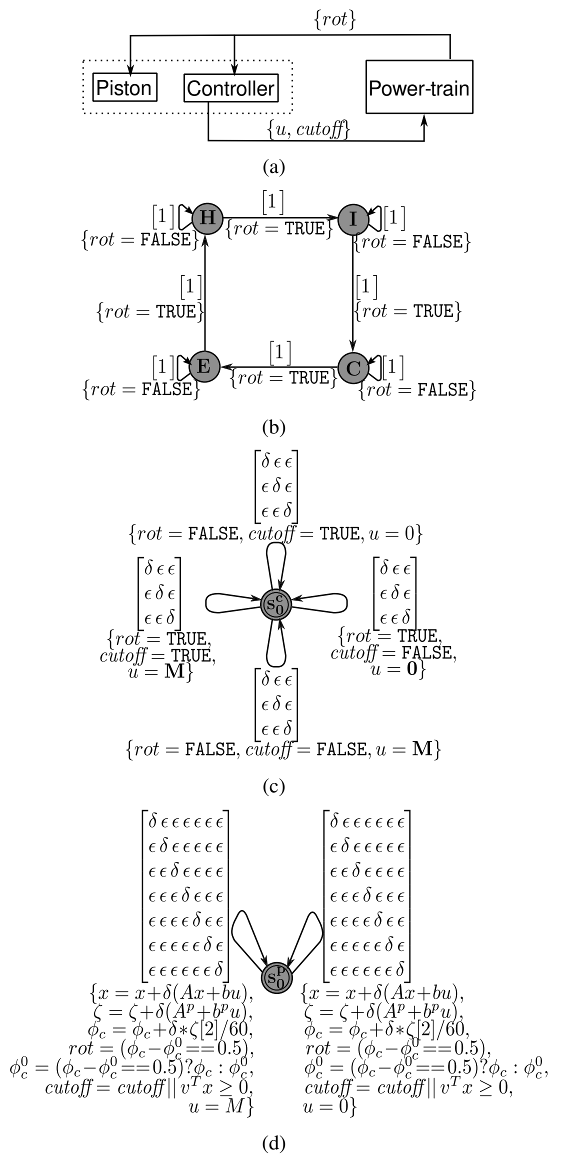
We have implemented a prototype tool to simulate the heterogeneous TM composition under morphisms. Our simulator, written in approximately 5000 lines of C++ code, supports a high level script language to specify tag machines. Each TM is described as a module consisting of a set of constraints on the declared variables. In particular, a module must contain declarations about the machine tag structure (`TAGSTRUCT`), the state (`MVAR`) and label (`MLABEL`) variable. Declarations about the machine variables (`MVAR`), initial state (`INIT`) and transitions (`EDGE`) are optional. A morphism associated with a machine can be declared likewise. Fig. 6 shows an example of our script language.

The inputs to our simulator are pairs of TMs and morphisms under which the TMs can be composed. At every step, our simulator explores all composition possibilities and provides the user with a set of simulation choices which can be carried out randomly or interactively.

B. An Automotive Example

We consider a simplified version of the automotive engine control model proposed by Balluchi et al. [10]. It consists of a sub-model capturing the piston sequential behavior and another capturing the power-train behavior under a cut-off control policy. The goal of the control policy is to reduce the unpleasant oscillations when the driver releases the gas pedal and requests no torque to the engine. The piston model is naturally expressed by a finite state machine while the power-train model can be represented by a continuous time equation. Thus, in order to understand the behavior of the car, it is important for designers to be able to compose these heterogeneous models.

A piston abstractly cycles through four phases: *1)* the *intake* (I) phase during which the piston loads the air-fuel



{rot}
Piston
Controller
Power-train
{u, cutoff}
(a)
H
I
C
E
[1]
{rot = TRUE}
{rot = FALSE}
(b)
{rot = FALSE, cutoff = TRUE, u = 0}
{rot = TRUE, cutoff = TRUE, u = M}
{rot = TRUE, cutoff = FALSE, u = 0}
{rot = FALSE, cutoff = FALSE, u = M}
(c)
{x = x+δ(Ax+bu), ζ = ζ+δ(A^p+b^p u), φ_c = φ_c+δ∗ζ[2]/60, rot = (φ_c−φ_c^0==0.5), φ_c^0 = (φ_c−φ_c^0==0.5)?φ_c : φ_c^0, cutoff = cutoff || v^T x ≥ 0, u = M}
{x = x+δ(Ax+bu), ζ = ζ+δ(A^p+b^p u), φ_c = φ_c+δ∗ζ[2]/60, rot = (φ_c−φ_c^0==0.5), φ_c^0 = (φ_c−φ_c^0==0.5)?φ_c : φ_c^0, cutoff = cutoff || v^T x ≥ 0, u = 0}
(d)

Figure 7. The automotive engine control model consisting of (a) a general view (b) pistons (c) a bang-bang controller, (d) the power-train

mix $q \in \mathbb{R}_+$; *2*) the *compression* (C) phase in which the loaded mix is compressed; *3*) the *expansion* (E) phase in which the compressed mix is combusted, producing the spark ignition; *4*) the *exhaust* (H) phase during which the piston expels combustion exhaust gases. We assume the torque evolution $u(t)$ to be a piece-wise constant function which is zero everywhere except in the E-phase when the spark ignition is set, i.e., $u(t) = G * q$ and $u(t) = 0$ otherwise, where G is the mix-to-torque gain.

The continuous time power-train behavior is modeled by the following linear system:

$$\dot{\zeta} = A^p \zeta + b^p u$$
$$\dot{\phi}_c = \omega_c$$

where $\zeta = [\alpha_e, \omega_c, \omega_p]$ represents the axle torsion angle, the crankshaft revolution speed and the wheel revolution speed, and ϕ_c represents the crankshaft angle.

In this paper we model the bang-bang control law [10] where the fuel injection is cut when $v^T x \geq 0$. The reduced state $x = [x_1, x_2]^T$ represents the system's oscillation component and is obtained by applying the state transformation $\dot{x}(t) = Ax(t) + bu(t)$ where $A = \left[\begin{smallmatrix} \lambda & -\mu \\ \mu & \lambda \end{smallmatrix}\right]$ and $\lambda \pm j\mu$ are the conjugate complex poles of A^p. The oscillation acceleration can then be computed based on x as $a(t) = cx(t)$. The starting point of the cut-off control horizon is the time at which all the loaded cylinders' potential torques are at the steady value $M = G * q^0$.

The piston model needs only information about the sequencing of events, while the power-train model requires the exact timing of events. Thus we can use $\mathcal{T}_1 = (\mathbb{N} \cup \{\epsilon_1\}, +)$ and $\mathcal{T}_2 = (\mathbb{R}_+ \cup \{\epsilon_2\}, +)$ where $\epsilon_1 = \epsilon_2 = -\infty$, to model them. Fig. 7(b) shows the piston behavior where the transitions of the automaton occur when a piston reaches the bottom or top dead point, i.e. when the flag *rot* is set. Fig. 7(c) and 7(d) describe the bang-bang control policy and the evolution of the power-train, respectively. We have approximated the power-train using the forward integration Euler method with a step of δ. The state variables are fixed as follows [10]:

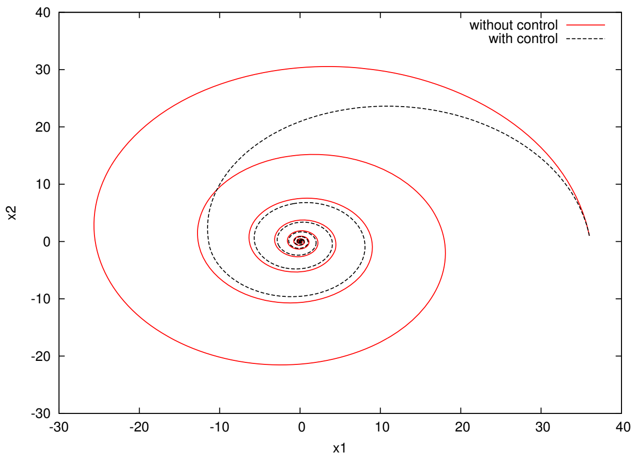
$$A^p = \begin{bmatrix} 0 & 1 & -7.556 \\ -448.1 & -5.186 & 30.87 \\ 3.042 & 0.02773 & -0.2105 \end{bmatrix}, b^p = \begin{bmatrix} 0 \\ 15.05 \\ 0 \end{bmatrix},$$
$$A = \begin{bmatrix} -2.671 & -21.54 \\ 21.54 & -2.671 \end{bmatrix}, b = \begin{bmatrix} 1.92339 \\ -14.32309 \end{bmatrix}, v = \begin{bmatrix} 0.01 \\ -1 \end{bmatrix},$$
$$c = \begin{bmatrix} 0.0379945 & -0.00257 \end{bmatrix}, M = 12.41, \delta = 0.001181.$$

C. Evaluation

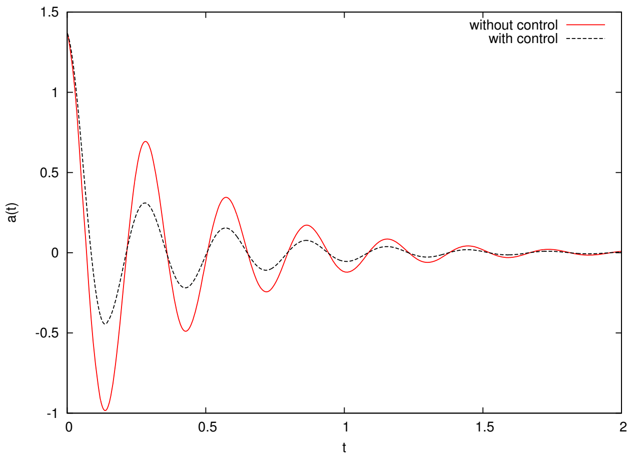
We have described the cut-off problem using our prototype TM-simulator and performed simulations of 2000 steps to evaluate the effect of the cut-off control on the oscillation acceleration. We present here the results of two such simulations on the above set of parameters and under the presence of morphisms such as $\rho_i : \mathcal{T}_i \mapsto \mathcal{T}_2$ given by $\rho_1(\tau_1) = \delta * \tau_1$ and $\rho_2(\tau_2) = \tau_2$.

Consistent with the previous result [10], Fig. 8 shows that better acceleration peaks and driving comfort are obtained when cut-off control is enforced. This consistent result can assist automotive engineers and designers in estimating the impact of the control implementation on the engine performance, thereby justifying and selecting suitable system parameter values to obtain a desirable performance.

Another important advantage offered by the heterogeneity methodology is in terms of component modeling. While the



(a)



(b)

Figure 8. The evolution of $x(t)$ and $a(t)$ without and with control

components in [10] had to be modeled in the same domain in order to solve the cut-off control problem, they can now be expressed in their natural domain. The component interaction can then be precisely quantified and modeled by means of morphisms. This advantage becomes much more crucial when designers have to deal with large and complex systems.

VII. Conclusion

We have developed an operational formalism for representing heterogeneous tag systems. To this end, we have extended tag machines, which were introduced to represent only homogeneous tag systems, to the heterogeneous context. Our heterogeneous TM formalism is able to operationally represent heterogeneous tag systems.

A natural concern is whether composition of TMs can soundly and completely represent the that of tag systems. The concern can be addressed under certain conditions such as interoperability and self-synchronization. Checking for self-synchronization in general, however, is a non-trivial property for which effective checking procedures are an important part of our future work.

Using an industrial case study and our prototype tool, we have demonstrated that TM modeling is useful in practice. This is especially important in systems consisting of components with differing natural domains, as in our case study. Contrary to previous work, our heterogeneous approach allows components to be modeled in their natural domain.

In the future, we aim at several extensions to our simulator. In particular, we plan to integrate it with a model checker and SMT solver, in order to enable automated compositional verification of heterogeneous systems and to support their contract-based design. Since our simulator is based on a formal representation, development and implementation of formal verification techniques is a natural extension which we will consider.

References

[1] X. Liu, Y. Xiong, and E. A. Lee, "The Ptolemy II framework for visual languages," in *HCC*. IEEE, 2001, pp. 50–51.

[2] A. Davare, D. Densmore, L. Guo, R. Passerone, A. L. Sangiovanni-Vincentelli, A. Simalatsar, and Q. Zhu, "METROII: A design environment for cyber-physical systems," *ACM Trans. on Embed. Comput. Syst.*, vol. 12, no. 1s, pp. 49:1–49:31, March 2013.

[3] A. Benveniste, B. Caillaud, L. P. Carloni, P. Caspi, and A. L. Sangiovanni-Vincentelli, "Composing heterogeneous reactive systems," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 4, pp. 43:1–43:36, 2008.

[4] A. Benveniste, B. Caillaud, L. P. Carloni, and A. Sangiovanni-Vincentelli, "Tag machines," in *EMSOFT*, 2005, pp. 255–263.

[5] http://www.mathworks.com/products/simulink/.

[6] E. Lee and A. Sangiovanni-Vincentelli, "A framework for comparing models of computation," *IEEE Trans. CAD of Integ. Circ. and Systems*, vol. 17, pp. 1217–1229, 1998.

[7] S. Dey, D. Sarkar, and A. Basu, "A tag machine based performance evaluation method for job-shop schedules," *IEEE Trans. CAD of Integ. Circ. and Systems*, vol. 29, no. 7, pp. 1028–1041, 2010.

[8] ——, "A Kleene algebra of tagged system actors," *Embedded Systems Letters, IEEE*, vol. 3, no. 1, pp. 28 –31, 2011.

[9] L. Benvenuti, A. Ferrari, L. Mangeruca, E. Mazzi, R. Passerone, and C. Sofronis, "A contract-based formalism for the specification of heterogeneous systems," in *Proceedings of the Forum on Specification, Verification and Design Languages (FDL08)*, Stuttgart, Germany, September 23–25, 2008, pp. 142–147.

[10] A. Balluchi, M. Benedetto, C. Pinello, C. Rossi, and A. Sangiovanni-Vincentelli, "Hybrid control for automotive engine management: The cut-off case," in *Hybrid Systems: Computation and Control*. Springer Berlin Heidelberg, 1998, vol. 1386, pp. 13–32.