Tag Machines *

Albert Benveniste† Benoît Caillaud Luca P. Carloni
Alberto L. Sangiovanni-Vincentelli

ABSTRACT

Heterogeneity is a challenge to overcome in the design of embedded systems. We presented in the recent past a theory for the composition of heterogeneous components based on tagged systems, a behavioral (denotational) framework. in this paper, we present an operational view of tagged systems, where we focus on tag machines as mathematical artifacts that act as finitary generators of tagged systems. Properties of tag machines are investigated. A fundamental theorem on *homogeneous* compositionality is given as a first step towards an operational theory of heterogeneous systems.

Categories and Subject Descriptors: C.3.3 [Special-purpose and application-based systems]: Real-time and embedded systems.

General Terms: Theory.

Keywords: Heterogeneous reactive systems, tagged systems, distributed deployment, GALS.

1. INTRODUCTION AND MOTIVATION

Dealing with heterogeneity is a fundamental issue in the agenda for embedded system research. In former papers [2] and [3], we addressed the lack of an all-encompassing mathematical framework for reasoning about heterogeneous composition. We proposed a mathematical framework for modeling heterogeneous reactive systems that provides a solid foundation to handle formally communication and coordination among heterogeneous components. This was done by developing a compositional *behavioral* theory of heterogeneous *tagged* systems. Logical time, physical time of various kinds, causalities, scheduling constraints, the simple local ordering of events of each individual signal, as well as their combination, can be captured by this formalism that was inspired by [6]. This made it possible to study formally how to deploy a specification into an implementation with different MoCC (Model of Computation and Communication), often a distributed architecture, so that their behaviors are equivalent. The theory was denotational. It dealt only with behaviors captured by "traces"; its value was mostly in providing a mathematical machinery to prove theorems about the correctness of particular methods and to develop solid foundations to design.

Automatic verification or synthesis techniques require finitary models or *machines* that act as generators of infinite traces. Examples of finitary models are finite state machines for synchronous systems. In this paper, a first step is made towards the development of a *machine-based* theory of heterogeneous systems based on *tag machines,* finite representations of tagged systems. Tag machines are, to the best of our knowledge, new mathematical objects and the analysis of their properties is far from trivial. We begin by studying in detail the map that associates each tag machine with the tagged system collecting its behaviors or traces. Then, we present a few examples on how to build tag machines for several concurrency models: asynchronous, synchronous reactive, Time-Triggered Architectures (TTA) and causality. Finally, we deal with the fundamental issue of composition of tag machines.

While our ultimate goal is to develop a theory of heterogeneous composition, we limit ourselves here to present homogeneous compositions. Even for this simpler case, there are fundamental (and non trivial) issues to address: the most important one is to relate parallel composition of tagged systems to an appropriate notion of product of tag machines. It should indeed not be a surprise that this can be a problem. It is well known for example that asynchronous systems can be generated by means of "partial order automata", *i.e.,* automata whose transitions are labeled by partial orders; closely related are High level Message Sequence Charts (HMSC) [4, 7], that are automata whose transitions are labeled with finite scenarios. For both cases, the product of the corresponding automata does not generate the parallel composition of associated asynchronous systems. This difficulty is related to undecidability problems for those models.

*This research was supported in part by the European Commission under the projects ARTIST2, IST-004527, by the NSF ITR CHESS, and by the GSRC.

†A. Benveniste and B. Caillaud are with Irisa/Inria, Campus de Beaulieu, 35042 Rennes cedex, France; Albert.Benveniste/Benoit.Caillaud@irisa.fr; www.irisa.fr/distribcom/benveniste/, www.irisa.fr/prive/Benoit.Caillaud/. L. Carloni is with the Department of Computer Science of Columbia University in the City of New York, NY 10027 luca@cs.columbia.edu; www.cs.columbia.edu/∼luca. A. Sangiovanni-Vincentelli is with the EECS Department of U.C. Berkeley, Berkeley, CA 94720; alberto@eecs.berkeley.edu; www-cad.eecs.berkeley.edu/∼alberto;

EMSOFT'05, September 19–22, 2005, Jersey City, New Jersey, USA.

Regarding this problem of relating parallel composition of tagged systems and products of tag machines, we show that while the product machine is always contained in the parallel composition of the component machines, only under particular assumptions (self-synchrony) on the machines to be composed, the converse is true. This result is indeed of interest even for already known models that are particular cases of tag machines (*e.g.,* the above examples of partial order automata or HMSC, but also for performance evaluation type of models). The ideas supporting the fundamental theorems presented here are borrowed from the work [8, 9] on correct-by-construction deployment of synchronous designs on Globally Asynchronous Locally Synchronous (GALS) architectures; see also interface automata proposed in [1]. Decidability problems underpinning the problem of tag machines product not matching tagged systems parallel composition have been investigated, *e.g.,* , in [7] and [4].

The paper is organized as follows: In Section 2, we recall and summarize our behavioral theory of tagged systems. In Section 3 we introduce the notion of tag machines and we present the relation between tag machines and tagged systems. In Section 4 we deal with the homogeneous composition of tag machines. We introduce the definition of product of tag machines and explore the conditions under which the product of tag machines is equivalent to their parallel composition.

2. BACKGROUND ON TAGGED SYSTEMS

Throughout this paper, $\mathbf{N} = \{1, 2, \dots\}$ denotes the set of positive integers; $\mathbf{N}$ is equipped with its usual total order $\leq$. $X \mapsto Y$ denotes the set of all partial functions from X to Y. If $(X, \leq_X)$ and $(Y, \leq_Y)$ are partial orders, $f \in X \mapsto Y$ is called *increasing* if $f(\leq_X) \subseteq \leq_Y$, i.e., $\forall x, x' \in X : x \leq_X x' \Rightarrow f(x) \leq_Y f(x')$.

2.1 Tag structures

Definition 1 (tag structure). *A* tag structure *is a triple $(\mathcal{T}, \leq, \sqsubseteq)$, where $\mathcal{T}$ is a set of* tags, *and $\leq$ and $\sqsubseteq$ are two partial orders satisfying the following property:*

$$\forall \tau, \tau' \in \mathcal{T} : \quad \tau \sqsubseteq \tau' \;\Rightarrow\; \tau \leq \tau'. \tag{1}$$

Partial order $\leq$ relates tags seen as time stamps, whereas partial order $\sqsubseteq$, called the *unification order*, defines how to unify tags and is essential to express coordination among events. Write

$$\tau_1 \bowtie \tau_2$$

if τ_1 and τ_2 possess a common upper bound in the unification order, and denote by $\tau_1 \sqcup \tau_2$ the least upper bound. Call $\bowtie$ and $\sqcup$ the *unification relation* and *unification map,* respectively.

Condition (1) is equivalently rephrased as follows: if $\tau_1 \bowtie \tau_2$ is a unifiable pair, then $\tau_i \leq (\tau_1 \sqcup \tau_2)$, for $i = 1, 2$. Therefore, Condition (1) expresses that unification is causal with respect to partial order of time stamps: the result of the unification cannot occur prior in time than its constituents. Condition (1) has the following consequence: if $\tau_1 \leq \tau_1'$, $\tau_2 \leq \tau_2'$, $\tau_1 \bowtie \tau_2$, and $\tau_1' \bowtie \tau_2'$ together hold, then $(\tau_1 \sqcup \tau_2) \leq (\tau_1' \sqcup \tau_2')$ must also hold. This will ensure that the system obtained via parallel composition preserves the agreed order of its components, see below.

Example 1 (tag structures).

1. Take $\mathcal{T} = \{.\}$, the singleton set, with $\leq = \sqsubseteq$ being the trivial order. As we shall see, this trivial tag structure will be used in modeling asynchrony, we denote it by $\mathcal{T}_{\text{triv}}$.
2. Take $\mathcal{T} = \mathbf{N}$, with $\leq$ being the usual (total) order, and $\sqsubseteq$ being the flat order. Therefore, $\tau_1 \bowtie \tau_2 \Leftrightarrow \tau_1 = \tau_2$, thus tag unification is by superimposition, like in the original tagged systems model by Lee and Sangiovanni-Vincentelli [6]. This tag structure will be used in modeling synchrony (tags are reaction indices), we denote it by $\mathcal{T}_{\text{synch}}$.
3. Take $\mathcal{T} = \mathbf{R}_+$, with $\leq$ being the usual (total) order, and $\sqsubseteq$ being the flat order. Therefore, $\tau_1 \bowtie \tau_2 \Leftrightarrow \tau_1 = \tau_2$. This tag structure will be used in modeling time-triggered systems (tags are dates from real-time), we denote it by $\mathcal{T}_{\text{tta}}$.
4. Take $\mathcal{T} = \mathbf{R}_+$, with $\leq = \sqsubseteq$ being the usual (total) order. Therefore, $\tau_1 \bowtie \tau_2$ always holds and $\tau_1 \sqcup \tau_2 = \max(\tau_1, \tau_2)$. This tag structure will be used in capturing execution times (tags will indicate earliest possible dates for execution of events), we denote it by $\mathcal{T}_{\text{wcet}}$.
5. Let $\mathcal{V}$ be some underlying set of variables, and set $\mathbf{N}_o =_{\text{def}} \mathbf{N} \cup \{-\infty\}$. Define a *dependency* to be a map: $\delta : \mathcal{V} \mapsto \mathbf{N}_o$. Take for $\mathcal{T}$ the set of all dependencies, with $\leq = \sqsubseteq$ are such that $\delta \leq \delta'$ iff $\forall v : \delta(v) \leq \delta'(v)$. Note that $\delta \sqcup \delta' = \max(\delta, \delta')$. This tag structure will be used in modeling dependencies or scheduling: $\delta(v) = n$ indicates that "tag δ cannot occur prior the n-th occurrence of variable v" (we take the convention that $n = -\infty$ indicates the absence of dependency), see Example 2 below for a formal definition. We denote the above tag structure of dependencies by $\mathcal{T}_{\text{dep}}$.

In Examples 1.1 – 1.3, tags can synchronize iff they are equal. This is the synchronization mechanism used in most formalisms dealing with functional aspects. In contrast, tags can always synchronize in Examples 1.4 and 1.5, but their unification function is non trivial. The reader may get the impression that this second class is rather dedicated to performance evaluation. This is, however, wrong: $\mathcal{T}_{\text{dep}}$ belongs to this second class and is used in composing systems while taking causality into account—this is not related to performance evaluation but is part of functional specification. ◇

2.2 Tagged systems

Let $\mathcal{V}$ be an underlying set of variables with domain D. For $V \subset \mathcal{V}$ finite, a *V-behavior,* or simply behavior, is an element:

$$\sigma \quad \in \quad V \mapsto \mathbf{N} \mapsto (\mathcal{T} \times D), \tag{2}$$

meaning that, for each $v \in V$, the n-th occurrence of v in behavior σ has tag $\tau \in \mathcal{T}$ and value $x \in D$. For v a variable, the map $\sigma(v) \in \mathbf{N} \mapsto (\mathcal{T} \times D)$ is called a *signal.* For σ a behavior, an *event* of σ is a tuple $(v, n, \tau, x) \in V \times \mathbf{N} \times \mathcal{T} \times D$ such that $\sigma(v)(n) = (\tau, x)$. Thus we can regard behaviors as sets of events. Call a *clock* any increasing function $(\mathbf{N}, \leq) \mapsto (\mathcal{T}, \leq)$. We require that, for each $v \in V$, the first projection of the map $\sigma(v)$ (it is a map $\mathbf{N} \mapsto \mathcal{T}$) is a clock and we call it the *clock of v in σ.* Thus, the clock of v yields the tags of the successive events of signal $\sigma(v)$.

Definition 2 (tagged systems). *A* tagged system *is a triple $P = (V, \mathcal{T}, \Sigma)$, where V is a finite set of variables, $\mathcal{T}$ is a tag structure, and Σ a set of V-behaviors.*

Consider two tagged systems $P_i = (V_i, \mathcal{T}, \Sigma_i), i \in \{1, 2\}$ with identical tag structure $\mathcal{T}$. Let $\sqcup$ be the unification function of $\mathcal{T}$. For two events $e_i = (v_i, n_i, \tau_i, x_i), i \in \{1, 2\}$:

write $e_1 \bowtie e_2$ iff $v_1 = v_2, n_1 = n_2, \tau_1 \bowtie \tau_2, x_1 = x_2 = x$
for $e_1 \bowtie e_2$, define $e_1 \sqcup e_2 = (v, n, \tau_1 \sqcup \tau_2, x)$

The unification map $\sqcup$ and relation $\bowtie$ extend point-wise to behaviors. Then, for $\sigma_i, i \in \{1,2\}$ a V_i-behavior, define, by abuse of notation:

$$\sigma_1 \bowtie \sigma_2 \text{ iff } \sigma_{1|V_1\cap V_2} \bowtie \sigma_{2|V_1\cap V_2},$$

and then

$$\sigma_1 \sqcup \sigma_2 =_{\text{def}} \left(\sigma_{1|V_1\cap V_2} \sqcup \sigma_{2|V_1\cap V_2}\right) \cup \sigma_{1|V\setminus V_2} \cup \sigma_{2|V_2\setminus V_1},$$

where $\sigma_{|W}$ denotes the restriction of behavior σ to the variables of W. Finally, for Σ_1 and Σ_2 two sets of behaviors, define their *conjunction*

$$\Sigma_1 \wedge \Sigma_2 \quad =_{\text{def}} \quad \{\sigma_1 \sqcup \sigma_2 \mid \sigma_i \in \Sigma_i \text{ and } \sigma_1 \bowtie \sigma_2\} \qquad (3)$$

The *homogeneous parallel composition* of P_1 and P_2 is then defined as follows:

$$P_1 \parallel P_2 \quad =_{\text{def}} \quad (V_1 \cup V_2, \mathcal{T}, \Sigma_1 \wedge \Sigma_2) \qquad (4)$$

The homogeneous parallel composition is associative and commutative. For technical reasons, we will also need an intermediate form of parallel composition that we call *gossip parallel composition*:

$$P_1 \underline{\parallel} P_2 \quad =_{\text{def}} \quad (V_1 \cup V_2, \mathcal{T}, \Sigma_1 \underline{\wedge} \Sigma_2), \text{ where} \qquad (5)$$
$$\Sigma_1 \underline{\wedge} \Sigma_2 \quad =_{\text{def}} \quad \{(\sigma_1, \sigma_2) \mid \sigma_i \in \Sigma_i \text{ and } \sigma_1 \bowtie \sigma_2\}$$

The reason for considering (5) is that $P_1 \underline{\parallel} P_2$ comes up with natural projections π_1 and π_2 defined by $\pi_i(\sigma_1, \sigma_2) =_{\text{def}} \sigma_i$. In contrast, no projection can be associated with $\parallel$ unless tag unification is by superimposition (see Example 1.4). Clearly, $P_1 \parallel P_2$ can be recovered from its gossip variant: just replace each pair (σ_1, σ_2) by $\sigma_1 \sqcup \sigma_2$.

EXAMPLE 2 (TAGGED SYSTEMS). The reader is referred to [2] for a more detailed discussion.

1. We first consider tagged systems with $\mathcal{T}_{\text{triv}}$ as tag structure, see Example 1.1. Their behaviors have the form $\sigma : V \mapsto (\mathbf{N} \mapsto (\{.\} \times D)$. Thus, all events have identical tag. Therefore, in this model, there is no synchronization constraint relating events associated to different variables. This is a suitable model for communication via unbounded FIFOs, *i.e.,* for asynchrony.

   Consider two identical asynchronous tagged systems $P = Q$ having two variables, b of boolean type with values in $\{\text{F}, \text{T}\}$ and x of integer type, and a single behavior given by:

b	:	T	F	T	F	T	F	...
x	:	1	1	1	.	.	.	

   Using our notations, this single behavior is formally given by:

   $$\sigma(b)(2n-1) = \text{T}, \sigma(b)(2n) = \text{F}, \text{ and } \sigma(x)(n) = 1.$$

   Of course, since, for tag structure $\mathcal{T}_{\text{triv}}$, homogeneous parallel composition is by intersection, we have $P \parallel Q = P = Q$.

2. Consider the tag structure $\mathcal{T}_{\text{synch}}$ introduced in Example 1.2. $\mathcal{T}_{\text{synch}}$ defines synchronous systems. Indeed, provided that all clocks are strictly increasing, the tag index set $\mathcal{T}_{\text{synch}}$ organizes behaviors into successive reactions as explained next. Call *reaction* a maximal set of events of σ with identical tag. Since clocks are strictly increasing, no two events of the same reaction can have the same variable. Regard a behavior as a sequence of reactions: $\sigma = \sigma_1, \sigma_2, \ldots$, with tags $\tau_1, \tau_2, \cdots \in \mathcal{T}_{\text{synch}}$. Thus $\mathcal{T}_{\text{synch}}$ provides a global, logical time basis: this feature characterizes synchrony.

   Consider two synchronous tagged systems P_s and Q_s having two variables, b of boolean type with values in $\{\text{F}, \text{T}\}$, and x of integer type. Assume that each system possesses only a single behavior, equal to:

P_s :	b	:	T	F	T	F	T	F	...
	x	:	1		1		1		...
Q_s :	b	:	T	F	T	F	T	F	...
	x	:		1		1		1	...

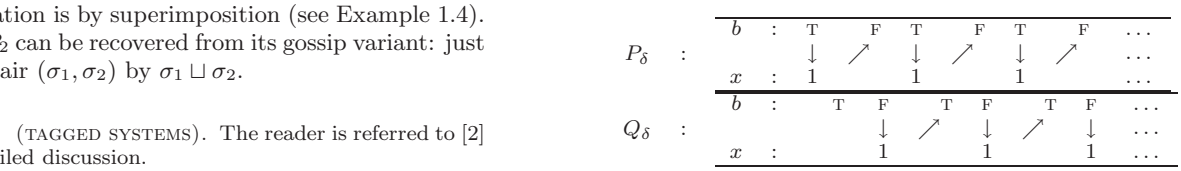
   In the above description, each behavior consists of a sequence of successive reactions, separated by vertical bars. Each reaction consists of an assignment of values to a subset of the variables; a blank indicates the absence of the considered variable in the considered reaction. These behaviors can be expressed formally in our framework as follows:

$$P_s : \begin{cases} \sigma(b)(2n-1) = (2n-1, \text{T}) \\ \sigma(b)(2n) = (2n, \text{F}) \\ \\ \sigma(x)(n) = (2n-1, 1) \end{cases}$$

$$Q_s : \begin{cases} \sigma(b)(2n-1) = (2n-1, \text{T}) \\ \sigma(b)(2n) = (2n, \text{F}) \\ \\ \sigma(x)(n) = (2n, 1) \end{cases}$$

   Now, the synchronous parallel composition of P_s and Q_s, defined by intersection: $P_s \parallel Q_s =_{\text{def}} P_s \cap Q_s$, is empty. The reason is that P_s and Q_s disagree on where to put absences for the variable x. Formally, they disagree on their respective tags.

3. Consider the tag structure $\mathcal{T}_{\text{dep}}$ associated with dependencies in Example 1.5. Consider two tagged systems P_δ and Q_δ having two variables, b and x, as above. Assume that each system possesses only a single behavior, equal to:

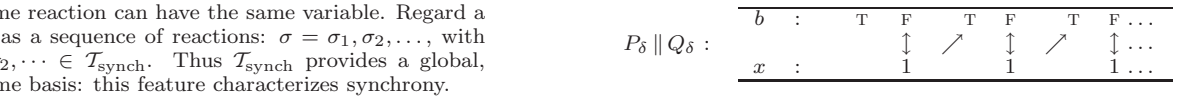


   The meaning of the directed graph is, for example and with reference to P_δ, to state that "the 2nd occurrence of x depends on the 3rd occurrence of b". For this model, an event has the form $e = (v, n, \delta, x)$, with the following interpretation: event e has v as associated variable, it is ranked n among the events with variable v, and it depends on the event of variable w that is ranked $\delta(w)$. The special case $\delta(w) = -\infty$ is interpreted as the absence of dependency. We take the convention that, for $e = (v, n, \delta, x)$ an event, $\delta(v) = n - 1$. Thus, on $\sigma(v)$, the set of dependencies reproduces the ranking (and corresponding branches are implicit in the above diagram). Corresponding formal descriptions therefore are:

$$P_\delta : \begin{cases} \sigma(b)(2n-1) = ((x, -\infty), \text{T}) \\ \sigma(b)(2n) = ((x, n), \text{F}) \\ \\ \sigma(x)(n) = ((b, 2n-1), 1) \end{cases}$$

$$Q_\delta : \begin{cases} \sigma(b)(2n-1) = ((x, n-1), \text{T}) \\ \sigma(b)(2n) = ((x, -\infty), \text{F}) \\ \\ \sigma(x)(n) = ((b, 2n), 1) \end{cases}$$

   The parallel composition $P_\delta \parallel Q_\delta$ is expected to yield:



   The reason for the double causality between x and F-occurrences of b is that the n-th x causes the $(2n)$-th b (i.e. the n-th F-occurrence of b) in P_δ whereas the $(2n)$-th b causes the n-th x in Q_δ. Formally, by the max rule for composing

dependency tags (see Example 1.5), the unique behavior of $P_\delta \,\|\, Q_\delta$ is written:

$$P_\delta \,\|\, Q_\delta : \left\{ \begin{array}{lcl} \sigma(b)(2n-1) & = & ((x,n-1),\mathrm{T}) \\ \sigma(b)(2n) & = & ((x,n),\mathrm{F}) \\ \\ \sigma(x)(n) & = & ((b,2n),1) \end{array} \right.$$

In conclusion, the single behavior of $P_\delta \,\|\, Q_\delta$ possesses causality loops and may be considered pathological and thus "rejected". Note that, in our model, dependencies induce a preorder, not a partial order. This should not be a surprise since preorders compose, whereas partial orders don't as shown by the above example.

◇

3. TAG MACHINES

In this section, we introduce tag machines as the fundamental tool to develop an operational theory of heterogeneous systems. In a first part, we study the modeling of the "progress of tags" by means of heaps of tag pieces. Heaps of pieces were originally introduced in [10] as a model for Mazurkiewicz trace languages. They were further used in [5] for performance evaluation studies. Here we adapt this notion to the more general context of tags. Then, in a second part, we introduce tag machines by taking values into account.

3.1 Heaps of tag pieces

DEFINITION 3 (ALGEBRAIC TAG STRUCTURE). *A tag structure $(\mathcal{T}, \leq, \sqsubseteq)$ is called* algebraic *if $\mathcal{T}$ is equipped with an internal binary operation written $\bullet$ and called* concatenation, *such that:*

1. *$(\mathcal{T}, \bullet)$ is a monoid with unit $\mathbf{1}$;*
2. *Concatenation $\bullet$ is compatible with both orders $\leq$ and $\sqsubseteq$:*

$$\begin{array}{rcl} \tau_1 \leq \tau_1' \text{ and } \tau_2 \leq \tau_2' & \Rightarrow & (\tau_1 \bullet \tau_2) \leq (\tau_1' \bullet \tau_2'). \\ \tau_1 \sqsubseteq \tau_1' \text{ and } \tau_2 \sqsubseteq \tau_2' & \Rightarrow & (\tau_1 \bullet \tau_2) \sqsubseteq (\tau_1' \bullet \tau_2'). \end{array}$$

Algebraic tag structures are generically denoted by $\widehat{\mathcal{T}}$, where $\mathcal{T}$ refers to the underlying set of tags.

DEFINITION 4 (TAG PIECES). *Let V be a finite subset of $\mathcal{V}$. A V-*tag piece *(or simply tag piece) is a tuple $(\widehat{\mathcal{T}}, V, \mu)$, where $\widehat{\mathcal{T}}$ is an algebraic tag structure, V is a finite set of variables, and μ is a matrix $\mu : V \times V \mapsto \mathcal{T}$.*

By abuse of language, the considered piece is denoted by μ. Pieces operate on vectors of tags as follows. Let

$$\vec{\tau} = (\tau_v)_{v \in V}$$

be a vector of tags belonging to $\mathcal{T}$ and indexed by the set V of variables. The result of applying μ to $\vec{\tau}$ is another vector of tags, written $\vec{\tau} \bullet \mu$, whose v-th coordinate $(\vec{\tau} \bullet \mu)_v$ is given by

$$(\vec{\tau} \bullet \mu)_v \quad =_{\mathrm{def}} \quad {}^{\sqsubseteq}\!\max_{w \in V} (\tau_w \bullet \mu_{wv}) \qquad (6)$$

where ${}^{\sqsubseteq}\max$ denotes the maximum with respect to unification order $\sqsubseteq$. Note that the map $(\vec{\tau}, \mu) \mapsto \vec{\tau} \bullet \mu$ is partial, since the maximum ${}^{\sqsubseteq}\max$ is not always defined. For M a finite alphabet of tag pieces, concatenation operation (6) allows to define the Kleene closure M^*, we call it the *heap of tag pieces* generated by M.

EXAMPLE 3 (HEAPS OF TAG PIECES). We detail the generic pieces from Example 1 and show how they operate. This is illustrated in Fig. 1.

1. Take $\mathcal{T}_{\mathrm{triv}} = \{.\}$ equipped with the trivial binary operation. Pieces do not increment tags—we regard them as having "zero-thickness", as shown in Fig. 1, first diagram. Pieces are infinitely flexible and fall down to adjust as shown.
2. Take $(\mathcal{T}_{\mathrm{synch}}, +)$. Since $\sqsubseteq$ is the flat order, for (6) to be defined, $\tau_w \bullet \mu_{wv}$ must not depend on w. For this property to be invariant when extending heaps, we must have that $\vec{\tau}$ itself must have all its component equal and ${}^{\sqsubseteq}\max_{w \in V} \mu_{wv}$ must not depend on v. Therefore, there is actually a unique form for synchronous pieces, namely:

$$\text{if } v = w \text{ then } \mu^{\mathrm{synch}}_{vw} = 1 \text{ else } \mu^{\mathrm{synch}}_{vw} = 0$$

   The integer tag is incremented by 1 for each new piece, *i.e.,* pieces model reactions. The presence/absence of a variable in a reaction is shown in Fig. 1 by a grey/white box, for the considered variable. It will be formally encoded by the concept of "support" we introduce below in Definition 5.
3. Take $(\mathcal{T}_{\mathrm{tta}}, +)$. The same comments hold, and the natural class for TTA pieces is

$$\text{if } v = w \text{ then } \mu^{\mathrm{tta}}_{wv} = \delta \text{ else } \mu^{\mathrm{tta}}_{wv} = 0,$$

   where δ is some positive increment of physical time.
4. Take $(\mathcal{T}_{\mathrm{wcet}}, +)$. Since $\sqsubseteq = \leq$ in $\mathcal{T}_{\mathrm{wcet}}$, operation (6) is now a total function and no condition is required on pieces. Operation (6) instantiates as

$$(\vec{\tau} \bullet \mu)_v \quad =_{\mathrm{def}} \quad \max_{w \in V} (\tau_w + \mu_{wv})$$

   i.e., we get the max-plus linear rule for performance evaluation.
5. Take $(\mathcal{T}_{\mathrm{dep}}, +)$, where the $+$ is meant component-wise, for each variable $v \in \mathcal{V}$; then, augment $\mathbf{N}_o$ with 0 so that the dependency defined by $\forall v, \mathbf{1}(v) = 0$ is the unit of $\mathcal{T}_{\mathrm{dep}}$. Coordinates of $\vec{\tau}$ as well as entries of the matrix μ are dependencies, *i.e.,* maps $V \ni v \mapsto \mathbf{N}_o$. Since pieces represent tag increments, we first specialize matrices μ as follows:

$$\forall w, v, \forall u \;\Rightarrow\; \mu_{wv}(u) \in \{\epsilon, 0, 1\}, \qquad (7)$$

   where $\epsilon =_{\mathrm{def}} -\infty$ encodes the absence of dependency. The generic concatenation rule (6) specializes as

$$\forall u \in V \quad : \quad (\vec{\tau} \bullet \mu)_v(u) = \max_{w \in V} (\tau_w(u) + \mu_{wv}(u)) \qquad (8)$$

   In (8), $\mu_{wv}(u)$ is to be interpreted as "the increment brought by μ, from w to v, regarding the dependency on u". In order for μ to model increments of dependencies as graphically specified such as in Fig. 1 and Fig. 3, μ should be taken as follows (symbol ϵ denotes $-\infty$ and notation "x/y if a/b" is a short hand to mean "x holds if a holds and y holds if b holds"):

   - $\mu_{vv}(v) = 0/1$ if v has no event/an event in μ.
   - For w, v, u not all identical, $\mu_{wv}(u) = \epsilon/0$ if there is no dependency/if there is a dependency from w to v in μ. Note that, subject to the constraint that w, v, u are not all identical, $\mu_{wv}(u)$ does not depend on u.

   See Fig. 1 for an illustration of this. The piece shown in this figure above the downward arrows is now detailed. We use the following conventions: u denotes an arbitrary variable

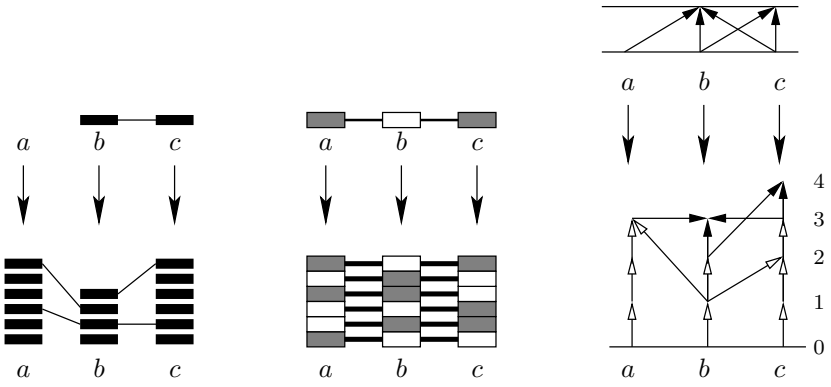


Figure 1: Heaps of tag pieces corresponding to $\mathcal{T}_{\text{triv}}$, $\mathcal{T}_{\text{synch}}$, and $\mathcal{T}_{\text{dep}}$, respectively. The downward arrows indicate how pieces "fall" to grow heaps. The piece on the top of each heap, *i.e.,* the last piece that has fallen, is shown also above the arrows. The numbers $1, 2, 3$ on the third diagram refer to index set $\mathbf{N}_o$, see Example 3.5. The reader is referred to Fig. 3 at the end for a detailed construction of the third heap.

and $\bar{a}$ denotes b or c, and similarly for $\bar{b}$ and $\bar{c}$. With these conventions, applying the above rules for chosing μ yields:

$$\begin{array}{lll} \mu_{aa}(a/\bar{a}) = 0/\epsilon & , \quad \mu_{ab}(u) = 0 & , \quad \mu_{ac}(u) = \epsilon \\ \mu_{ba}(u) = \epsilon & , \quad \mu_{bb}(b/\bar{b}) = 1/0 & , \quad \mu_{bc}(u) = 0 \\ \mu_{ca}(u) = \epsilon & , \quad \mu_{cb}(u) = 0 & , \quad \mu_{cc}(c/\bar{c}) = 1/0 \end{array} \tag{9}$$

Regarding the height of the heap, we have the following formulas, where τ_a, etc., are the coordinates of $\vec{\tau}$:

$$\begin{array}{lll} \tau_a(a) = 3 & , \quad \tau_a(b) = 1 & , \quad \tau_a(c) = \epsilon \\ \tau_b(a) = \epsilon & , \quad \tau_b(b) = 2 & , \quad \tau_b(c) = \epsilon \\ \tau_c(a) = \epsilon & , \quad \tau_c(b) = 1 & , \quad \tau_c(c) = 3 \end{array}$$

and therefore rule (8) with μ as in (9) yields $\vec{\tau}' = \vec{\tau} \bullet \mu$ such that

$$\begin{array}{lll} \tau'_a(a) = 3 & , \quad \tau'_a(b) = 1 & , \quad \tau'_a(c) = \epsilon \\ \tau'_b(a) = 3 & , \quad \tau'_b(b) = 3 & , \quad \tau'_b(c) = 3 \\ \tau'_c(a) = \epsilon & , \quad \tau'_c(b) = 2 & , \quad \tau'_c(c) = 4 \end{array}$$

As an example of this calculus, we detail the calculation of the term $\tau'_c(b)$. To apply (8), we compare the results of the right hand side of this formula, for the three choices $w = a, b, c$:

$$\begin{array}{llllll} w = a & : & \tau_a(b) = 1 & , \quad \mu_{ac}(b) = \epsilon & \text{yields} & \epsilon \\ w = b & : & \tau_b(b) = 2 & , \quad \mu_{bc}(b) = 0 & \text{yields} & 2 \\ w = c & : & \tau_c(b) = 1 & , \quad \mu_{cc}(b) = 0 & \text{yields} & 1 \end{array}$$

which yields $\tau'_c(b) = 2$, as expected. Rule (8) indeed implements the concatenation of basic Message Sequence Charts (MSC).

◇

So far we have considered heaps of tag pieces without attached events and values. We now introduce pieces that carry events with values attached to them, we call them tag machines.

3.2 Defining tag machines

Example 4 (tag machines). Consider Example 3.5 and the third diagram of Fig. 1 where the case of heaps of pieces modeling dependencies is discussed. The falling piece shown in this figure adds two events to the pre-existing behavior, for the two variables b and c, respectively. To this end, the piece of formula (9) must be enriched as follows (we have removed the first column of μ as a is not an event of this piece):

$$\begin{array}{lll} \mu_{ab} = 0 & , \quad \mu_{ac} = \epsilon \\ \mu_{bb} = 1 & , \quad \mu_{bc} = 0 \\ \mu_{cb} = 0 & , \quad \mu_{cc} = 1 \\ x_b & , \quad x_c \end{array} \tag{10}$$

where $x_b \in D$ and $x_c \in D$ are the values carried by the events associated with variables b and c, respectively. It remains to formalize how these new events are concatenated to the pre-existing behavior in a way consistent with the third diagram of Fig. 1. ◇

Definition 5 (tag machine). *A $(V, \mathcal{T})$-labeled tag piece is a pair $\boldsymbol{\mu} = (\mu, \nu)$, where μ is a V-tag piece with algebraic tag structure $\widehat{\mathcal{T}}$ and $\nu \in V \mapsto D$. The domain of partial function ν is denoted by $V_{\boldsymbol{\mu}}$ and is called the* support *of $\boldsymbol{\mu}$. If $v \in V_{\boldsymbol{\mu}}$ we say that $\boldsymbol{\mu}$* has an event for *v. The set of all $(V, \mathcal{T})$-labeled tag pieces is denoted by $\mathcal{M}(V, \mathcal{T})$.*

A tag machine *is a finite automaton whose transitions are labeled by labeled tag pieces. Formally, a tag machine is a tuple $A = (S, s_0, V, \widehat{\mathcal{T}}, \mathbf{M}, \Delta)$, where:*

- *S is a finite set of states and $s_0 \in S$ is the initial state;*
- *V is a finite set of variables with finite domain D;*
- *$\widehat{\mathcal{T}}$ is an algebraic tag structure;*
- *$\mathbf{M}$ is a finite set of $(V, \mathcal{T})$-labeled tag pieces;*
- *$\Delta \subseteq S \times \mathbf{M} \times S$ is a transition relation.*

Write $s \xrightarrow{\boldsymbol{\mu}} s'$ to indicate that $(s, \boldsymbol{\mu}, s') \in \Delta$, and $s \xrightarrow{\boldsymbol{\mu}}$ if $s \xrightarrow{\boldsymbol{\mu}} s'$ holds for some s'.

Whenever convenient, S_A, V_A, etc, will refer to the different components of the tuple building tag machine A.

Regarding $\mathbf{M}$ as a finite alphabet yields the language $\mathcal{L}(A) \subseteq \mathbf{M}^*$, where $\mathbf{M}^*$ denotes the free monoid over $\mathbf{M}$. For $w = \boldsymbol{\mu}_1 \bullet \ldots \bullet \boldsymbol{\mu}_K \in \mathcal{L}(A)$, write $s \xrightarrow{w} s'$ if

$$s \xrightarrow{\boldsymbol{\mu}_1} s_1 \xrightarrow{\boldsymbol{\mu}_2} s_2 \ldots s_{K-1} \xrightarrow{\boldsymbol{\mu}_K} s'$$

Concatenating the pieces of a word $w \in \mathcal{L}(A)$ yields a behavior in the form of a "heap of labeled tag pieces" $\sigma(w)$ that we need to define now.

Let $\sigma \in V \mapsto \mathbf{N} \mapsto (\mathcal{T} \times D)$ be a finite behavior, meaning that, for each $v \in V$, the map $\sigma(v) \in \mathbf{N} \mapsto (\mathcal{T} \times D)$ has $\{1, \ldots, \ell_v\}$ as a domain, for some finite integer ℓ_v. For σ a finite behavior, let the following map:

$$tail(\sigma) : V \ni v \mapsto \sigma(v, \ell_v) \in (\mathcal{T} \times D)$$

be the *tail* of σ. Then, for $\vec{\boldsymbol{\tau}} \in V \mapsto \mathcal{T} \times D$ and $\boldsymbol{\mu} = (\mu, \nu)$ a labeled tag piece, decompose $\vec{\boldsymbol{\tau}} = (\vec{\tau}, \nu')$ and set

$$\vec{\boldsymbol{\tau}} \bullet \boldsymbol{\mu} \quad =_{\text{def}} \quad (\vec{\tau} \bullet \mu, \nu) \tag{11}$$

and, finally:

$$\sigma \bullet \boldsymbol{\mu}(v, n) =_{\text{def}} \begin{cases} \sigma(v, n) & \text{if} \quad n \leq \ell_v \\ [tail(\sigma) \bullet \boldsymbol{\mu}](v) & \text{if} \quad n = \ell_v + 1 \\ \text{undefined} & \text{if} \quad n > \ell_v + 1 \end{cases} \tag{12}$$

Observe that $\boldsymbol{\mu}$ adds an event for v in $\sigma \bullet \boldsymbol{\mu}$ iff $\boldsymbol{\mu}$ has an event for v. The set of all so obtained behaviors is denoted by Σ_A. The map

$$\mathcal{L}(A) \ni w \quad \mapsto \quad \boldsymbol{\sigma}(w) \in \Sigma_A \tag{13}$$

induces a tag system

$$P_A \quad = \quad (V, \mathcal{T}, \Sigma_A). \tag{14}$$

4. HOMOGENEOUS COMPOSITION OF TAG MACHINES

In this paper, we restrict our analysis of composition of tag machines to the parallel homogeneous case, *i.e.,* we consider compositions of tag machines that have the same tag structure. We are interested in studying under which conditions the parallel composition defined in Equation 4 can be computed as the product of tag machines.

For $\boldsymbol{\mu} = (\mu, \nu)$ and $\boldsymbol{\mu}' = (\mu', \nu')$ two labeled tag pieces, set $U =_{\text{def}} V \cup V'$ and $W =_{\text{def}} V \cap V'$ and say

$$\boldsymbol{\mu} \bowtie \boldsymbol{\mu}' \quad \text{iff} \quad \forall (w, v) \in W \times W \Rightarrow \begin{cases} \mu_{wv} \bowtie \mu'_{wv} \\ \nu(v) = \nu'(v) \end{cases}$$

and then set

$$\boldsymbol{\mu} \sqcup \boldsymbol{\mu}' \quad =_{\text{def}} \quad \left(\mu \sqcup \mu', \nu \sqcup \nu'\right)$$

where

$$(\mu \sqcup \mu')_{wv} =_{\text{def}} \begin{cases} \mu_{wv} \sqcup \mu'_{wv} \text{ if } (w, v) \in W \times W \\ \mu_{wv} \text{ if } (w, v) \in (V \times V) \setminus (W \times W) \\ \mu'_{wv} \text{ if } (w, v) \in (V' \times V') \setminus (W \times W) \end{cases}$$

$$(\nu \sqcup \nu')(v) =_{\text{def}} \begin{cases} \nu(v) \text{ if } v \in V \\ \nu'(v) \text{ if } v \in V' \setminus W \end{cases}$$

Definition 6 (product of tag machines). *For A_1 and A_2 two tag machines defined over the same algebraic tag structure $\widehat{\mathcal{T}}$, their* product $A_1 \times A_2$ *is a tag machine $A = (S, s_0, V, \widehat{\mathcal{T}}, \mathbf{M}, \Delta)$, such that:*

- $S = S_1 \times S_2$ *and* $s_0 = (s_{0,1}, s_{0,2})$ *is the initial state;*
- $V = V_1 \cup V_2$*;*
- $\mathbf{M} = \{\boldsymbol{\mu}_1 \sqcup \boldsymbol{\mu}_2 \mid (\boldsymbol{\mu}_1, \boldsymbol{\mu}_2) \in \mathbf{M}_1 \times \mathbf{M}_2 \text{ and } \boldsymbol{\mu}_1 \bowtie \boldsymbol{\mu}_2\}$
- $\Delta \subseteq S \times \mathbf{M} \times S$ *is the set of tuples $t = (s, \boldsymbol{\mu}, s')$ such that $\boldsymbol{\mu} = \boldsymbol{\mu}_1 \sqcup \boldsymbol{\mu}_2$ and $\boldsymbol{\mu}_i \in \Delta_i$, for $i \in \{1, 2\}$.*

Again, in order to have projections at hand, we shall need a *gossip product*

$$A_1 \underline{\times} A_2 \tag{15}$$

where $\mathbf{M}$ is replaced by its gossip version

$$\underline{\mathbf{M}} =_{\text{def}} \{(\boldsymbol{\mu}_1, \boldsymbol{\mu}_2) \mid (\boldsymbol{\mu}_1, \boldsymbol{\mu}_2) \in \mathbf{M}_1 \times \mathbf{M}_2 \text{ and } \boldsymbol{\mu}_1 \bowtie \boldsymbol{\mu}_2\}$$

4.1 Self-synchronization and a fundamental theorem

Regarding the product of tag machines as defined in Definition 6, we always have the inclusion

$$P_{A_1 \times A_2} \quad \subseteq \quad P_{A_1} \parallel P_{A_2} \tag{16}$$

but the converse is generally not true. What are the practical consequences of this? The right model of two communicating tag machines P_{A_1} and P_{A_2} is $P_{A_1} \parallel P_{A_2}$, not $P_{A_1 \times A_2}$. The reason is that $P_{A_1 \times A_2}$ implies an extra protocol to synchronize the transitions of the two automata, something generally not provided by a communication medium implementing the MoCC corresponding to tag structure $\mathcal{T}$—see the discussion in the introduction. There are, however, favorable situations we discuss in this section.

Referring to the map $\mathcal{L}(A) \ni w \mapsto \boldsymbol{\sigma}(w) \in \Sigma_A$ defined in (13), it may be the case that there exists a word $w' \in \mathbf{M}^*$ such that $\boldsymbol{\sigma}(w') = \boldsymbol{\sigma}(w)$ but $w' \notin \mathcal{L}(A)$.

Definition 7 (self-synchronizing). *Let w and w' be two words of the free monoid $\mathcal{M}(V, \mathcal{T})^*$. If $\boldsymbol{\sigma}(w') = \boldsymbol{\sigma}(w)$ holds, write $w' \sim w$. Call* self-synchronizing *a tag machine A such that $\mathcal{L}(A)$ is $\sim$-closed.*

This is an important property as we shall see later. For $\mathcal{T} = \mathcal{T}_{\text{synch}}$ or $\mathcal{T} = \mathcal{T}_{\text{tta}}$, all tag machines are self-synchronizing—this property indeed characterizes synchronous MoCCs. This no longer holds for other MoCCs of Example 1. The above property is justified by the following result:

Theorem 1. *If A_1 and A_2 are self-synchronizing, then*

1. $P_{A_1 \times A_2} = P_{A_1} \parallel P_{A_2}$*;*
2. $P_{A_1 \times A_2}$ *is also self-synchronizing.*

Proof. Let us prove the first property. We shall prove the following stronger property involving gossip variants:

$$P_{A_1 \underline{\times} A_2} \quad = \quad P_{A_1} \underline{\parallel} P_{A_2}. \tag{17}$$

Clearly, (17) implies statement 1 of the theorem. The following diagram holds:

$$\begin{array}{ccccc} \mathcal{L}(A_1) & \xleftarrow{\pi_1} & \mathcal{L}(A_1 \times A_2) & \xrightarrow{\pi_2} & \mathcal{L}(A_2) \\ \downarrow \boldsymbol{\sigma} & & \downarrow \boldsymbol{\sigma} & & \downarrow \boldsymbol{\sigma} \\ P_{A_1} & & P_{A_1 \underline{\times} A_2} & & P_{A_2} \\ & {\scriptstyle \pi_1} \swarrow & \cap & \searrow {\scriptstyle \pi_2} & \\ & & P_{A_1} \underline{\parallel} P_{A_2} & & \end{array} \tag{18}$$

where π_i, for $i \in \{1,2\}$, denotes the ith projection associated to a gossip product or parallel composition, and $\boldsymbol{\sigma}$ denotes the map defined in (13). We first prove the existence of the converse inclusion $\bigcup$ in the bottom part of the diagram. To this end, pick $(\sigma_1, \sigma_2) \in P_{A_1} \underline{\|} P_{A_2}$. Let w_1 be any word belonging to $\mathcal{L}(A_1) \cap \boldsymbol{\sigma}^{-1}(\sigma_1)$. Since $\sigma_1 \bowtie \sigma_2$ and since A_2 is self-synchronizing, there exists $w_2 \in \mathcal{L}(A_2) \cap \boldsymbol{\sigma}^{-1}(\sigma_2)$ such that $w_1 \bowtie w_2$, seen as words. Therefore, $(w_1, w_2) \in \mathcal{L}(A_1 \underline{\times} A_2)$ such that $\pi_i(w) = w_i$ for $i \in \{1,2\}$. Clearly, $\boldsymbol{\sigma}(w) \in P_{A_1 \underline{\times} A_2}$ and $\boldsymbol{\sigma}(w) = \sigma$. This proves the converse inclusion $\cup$ in the bottom part of the diagram. From this, (17) follows immediately.

Up to this point we used only the fact that A_2 is self-synchronizing. If, furthermore, A_1 is also self-synchronizing, then the same argument can be used to prove the second statement. ◇

Remark. Note that, in proving statement 1, we did not use the fact that both components are self-synchronizing: only one would suffice. More generally, if n components are considered, then it would suffice that $n-1$ components be self-synchronizing in order for the composition being mirrored by heaps of pieces. Also, as a consequence of Theorem 1, for $\mathcal{T} = \mathcal{T}_{\text{synch}}$ or $\mathcal{T} = \mathcal{T}_{\text{tta}}$, equality holds in (16). ◇

4.2 An effective criterion for self-synchronization

The definition of self-synchronizing involves traces and is therefore not effective—it cannot be checked directly on the structure of the considered tag machine. Theorem 2 below fills this gap.

Say that two $(V,\mathcal{T})$-labeled pieces $\boldsymbol{\mu} = (\mu, \nu)$ and $\boldsymbol{\mu}' = (\mu', \nu')$ are *consistent,* written $\boldsymbol{\mu} \,[\!]\, \boldsymbol{\mu}'$, if, for every variable $v \in V_{\boldsymbol{\mu}} \cap V_{\boldsymbol{\mu}'}$, we have $\nu(v) = \nu'(v)$ and $\forall w \in V \Rightarrow \boldsymbol{\mu}_{wv} = \boldsymbol{\mu}'_{wv}$. In words, the two pieces agree on the common part of their supports, when seen as functions. (Consistency relation $[\!]$ should not be confused with unifiability relation $\bowtie$.) If $\boldsymbol{\mu} \,[\!]\, \boldsymbol{\mu}'$ holds, we can define their intersection $\boldsymbol{\mu} \cap \boldsymbol{\mu}'$ as the common restriction of $\boldsymbol{\mu}$ or $\boldsymbol{\mu}'$ to $V_{\boldsymbol{\mu}} \cap V_{\boldsymbol{\mu}'}$, and their union $\boldsymbol{\mu} \cup \boldsymbol{\mu}'$ as being the labeled piece that agrees with $\boldsymbol{\mu}$ on $V_{\boldsymbol{\mu}}$ and with $\boldsymbol{\mu}'$ on $V_{\boldsymbol{\mu}'}$.

THEOREM 2. *Consider the following conditions:*

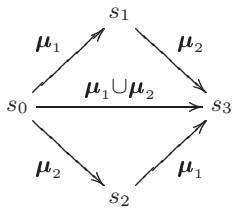
SS$_1$ *A is deterministic:* $s \xrightarrow{\boldsymbol{\mu}} s_i, i = 1,2 \Rightarrow s_1 = s_2$.

SS$_2$ *A is closed under sequential decomposition:*

$$\left.\begin{array}{r} s_0 \xrightarrow{\boldsymbol{\mu}} s_2 \\ \exists \boldsymbol{\mu}_1, \boldsymbol{\mu}_2 \in \mathbf{M}_A \\ \text{such that } \boldsymbol{\mu} = \boldsymbol{\mu}_1 \bullet \boldsymbol{\mu}_2 \end{array}\right\} \Rightarrow \left\{\begin{array}{l} \exists s_1 \in S \text{ such that} \\ s_0 \xrightarrow{\boldsymbol{\mu}_1} s_1 \xrightarrow{\boldsymbol{\mu}_2} s_2 \end{array}\right.$$

SS$_3$ *Let* $\boldsymbol{\mu}_1$ *and* $\boldsymbol{\mu}_2$ *belong to* $\mathbf{M}_A$ *and be such that* $\boldsymbol{\mu}_1 \,[\!]\, \boldsymbol{\mu}_2$. *Then,* $\boldsymbol{\mu}_1$ *decomposes as* $\boldsymbol{\mu}_1 = (\boldsymbol{\mu}_1 \cap \boldsymbol{\mu}_2) \bullet \boldsymbol{\mu}'_1$, *for some* $\boldsymbol{\mu}'_1 \in \mathbf{M}_A$.

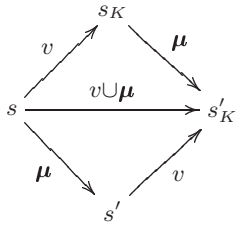
SS$_4$ *Let* $\boldsymbol{\mu}_1$ *and* $\boldsymbol{\mu}_2$ *belong to* $\mathbf{M}_A$ *and be such that* $s_0 \xrightarrow{\boldsymbol{\mu}_1} s_1$, $s_0 \xrightarrow{\boldsymbol{\mu}_2} s_2$, *and* $V_{\boldsymbol{\mu}_1} \cap V_{\boldsymbol{\mu}_2} = \emptyset$. *Then, there exists* $s_3 \in S$ *such that:*



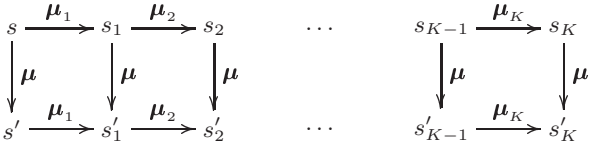
If A satisfies conditions SS$_1$*–SS*$_4$*, it is self-synchronizing.*

The proof of this theorem relies on Lemma 1 below. For $v = \boldsymbol{\mu}_1 \bullet \ldots \bullet \boldsymbol{\mu}_K$ a word of $\mathcal{L}(A)$, set $V_v =_{\text{def}} \bigcup_{k=1}^{K} V_{\boldsymbol{\mu}_k}$. If $v \in \mathcal{L}(A)$ and $\boldsymbol{\mu} \in \mathcal{L}(A)$ are such that $V_{\boldsymbol{\mu}} \cap V_v = \emptyset$, define $v \cup \boldsymbol{\mu} =_{\text{def}} (\boldsymbol{\mu}_1 \cup \boldsymbol{\mu}) \bullet \ldots \bullet \boldsymbol{\mu}_K$.

LEMMA 1. *Assume SS*$_4$. *Let* $\boldsymbol{\mu} \in \mathcal{L}(A)$ *be such that* $s \xrightarrow{v} s_K$, $s \xrightarrow{\boldsymbol{\mu}} s'$, *and* $V_{\boldsymbol{\mu}} \cap V_v = \emptyset$. *Then, there exists a state* s'_K *such that*



PROOF. Write $\boldsymbol{\mu}_1 \sim \boldsymbol{\mu}_2$ for two labeled tag pieces satisfying *SS*$_4$ and extend $\sim$ by taking its transitive closure: this yields a *commutation* equivalence relation. Repeatedly applying *SS*$_4$ yields the following commutative diagram:



This proves the Lemma. ◇

Proof of Theorem 2. Pick $w \in \mathcal{L}(A)$ and let $w' \in \mathcal{M}(V,\mathcal{T})^*$ be such that $\boldsymbol{\sigma}(w') = \boldsymbol{\sigma}(w)$. We need to prove that $w' \in \mathcal{L}(A)$ as well. Write $s_0 \xrightarrow{w} s$ to mean that $w = \boldsymbol{\mu}_1 \ldots \boldsymbol{\mu}_n$ and $s_0 \xrightarrow{\boldsymbol{\mu}_1} s_1 \ldots s_{n-1} \xrightarrow{\boldsymbol{\mu}_n} s$.

Call *atomic* a labeled piece $\boldsymbol{\mu} \in \mathbf{M}_A$ such that no non trivial decomposition $\boldsymbol{\mu} = \boldsymbol{\mu}_1 \bullet \boldsymbol{\mu}_2$ exists, where $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2 \in \mathbf{M}_A$. Set $\sigma =_{\text{def}} \boldsymbol{\sigma}(w)$. By *SS*$_2$, σ can be obtained as the image of a word of $\mathcal{L}(A)$ composed of atomic pieces only, call such a decomposition an *atomic decomposition.*

Suppose that two different atomic decompositions of σ exist, say, $\sigma = \boldsymbol{\sigma}(w_a) = \boldsymbol{\sigma}(w'_a)$, for $w_a \neq w'_a$ two atomic decompositions. Let $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}'_1$ be the heads of w_a and w'_a, respectively, and let $w_a \setminus \boldsymbol{\mu}_1$ and $w'_a \setminus \boldsymbol{\mu}'_1$ be the associated residuals. We have $\boldsymbol{\mu}_1 \,[\!]\, \boldsymbol{\mu}'_1$. Three cases can occur:

1. $V_{\boldsymbol{\mu}_1} = V_{\boldsymbol{\mu}'_1}$, and thus $\boldsymbol{\mu}_1 = \boldsymbol{\mu}'_1$. In this case, by *SS*$_1$ these two pieces must lead to the same state, and we can repeat the proof with a shorter word.

2. $\emptyset \neq V_{\boldsymbol{\mu}_1} \cap V_{\boldsymbol{\mu}'_1} \neq V_{\boldsymbol{\mu}_1} \cup V_{\boldsymbol{\mu}'_1}$. By SS_3, $\boldsymbol{\mu}_1$ is decomposable within $\mathbf{M}_A$, thus contradicting the atomicity of $\boldsymbol{\mu}_1$.

3. $\emptyset = V_{\boldsymbol{\mu}_1} \cap V_{\boldsymbol{\mu}'_1}$. Then, $\boldsymbol{\sigma}(w_a \setminus \boldsymbol{\mu}_1)$ and $\boldsymbol{\sigma}(w'_a \setminus \boldsymbol{\mu}'_1)$ are two suffixes of σ, having s_1 and s'_1 as respective initial states. Therefore, $\boldsymbol{\sigma}(w_a \setminus \boldsymbol{\mu}_1) \cap \boldsymbol{\sigma}(w'_a \setminus \boldsymbol{\mu}'_1)$ is also a suffix of σ. Consider $w_{a,1} =_{\text{def}} w_a \setminus \boldsymbol{\mu}_1$, and set

$$k_1 =_{\text{def}} \min(k \mid k > 1, V_{\boldsymbol{\mu}_k} \cap V_{\boldsymbol{\mu}'_1} \neq \emptyset)$$

Such an index exists. Set $v_{a,1} =_{\text{def}} \boldsymbol{\mu}_1 \bullet \cdots \bullet \boldsymbol{\mu}_{k_1-1}$. The support of word $v_{a,1}$ has empty intersection with that of $\boldsymbol{\mu}'_1$. Let s_{k_1} be the state reached by $v_{a,1}$. By Lemma 1, we still have $s_{k_1} \xrightarrow{\boldsymbol{\mu}'_1}$, and therefore $\boldsymbol{\mu}'_1 [] \boldsymbol{\mu}_{k_1}$, but, now, these two pieces do not have disjoint supports, so we conclude as for case 2.

Therefore the theorem is proved by induction. ◇

Comment. Condition SS_1 prohibits hiding state variables that are stored in registers, this is a mild and simple request.

Conditions $SS_{2,3,4}$ are satisfied by all tag machines for $\mathcal{T} = \mathcal{T}_{\text{synch}}$ or $\mathcal{T}_{\text{tta}}$. Indeed, all pieces are atomic for these MoCCs since logical or physical time is increased by one unit by each piece.

In contrast, the above conditions are generally not satisfied for $\mathcal{T} = \mathcal{T}_{\text{triv}}$ or $\mathcal{T}_{\text{wcet}}$ or $\mathcal{T}_{\text{dep}}$. Being self-synchronized is a non trivial property for tag machines having those MoCCs. This property is undecidable in general. Still, synthesizing this property by adding proper signaling or protocols in order to enforce the conditions $SS_{2,3,4}$ of Theorem 2 is sketched in Section 4.3. Its systematic study is the subject of future work.

4.3 Illustrative example

Figure 2 shows an example of a tag machine A using $\mathcal{T}_{\text{dep}}$. The 1st diagram shows the automaton. It has 4 states and its labeled tag pieces are $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\mu}_3, \boldsymbol{\mu}_4$, shown on the remaining part of the figure. We have $\mathcal{L}(A) = (\boldsymbol{\mu}_1\boldsymbol{\mu}_2)^*\boldsymbol{\mu}_3\boldsymbol{\mu}_4$. We have $\boldsymbol{\sigma}(\boldsymbol{\mu}_1\boldsymbol{\mu}_2\boldsymbol{\mu}_1\boldsymbol{\mu}_2) = \boldsymbol{\sigma}(\boldsymbol{\mu}_1\boldsymbol{\mu}_2\boldsymbol{\mu}_3\boldsymbol{\mu}_4)$, thus A is not self-synchronizing. The reader can verify that A violates condition SS_3 of Theorem 2, for the pair $(\boldsymbol{\mu}_1, \boldsymbol{\mu}_3)$, since $\boldsymbol{\mu}_1 [] \boldsymbol{\mu}_3$, $\boldsymbol{\mu}_1 \cap \boldsymbol{\mu}_3 = \boldsymbol{\mu}_3$ but no piece $\boldsymbol{\mu} \in \mathbf{M}_A$ such that $\boldsymbol{\mu}_1 = \boldsymbol{\mu}_3 \bullet \boldsymbol{\mu}$ does exist.

To see how Theorem 2 can be used to make tag machine A self-synchronizing, let us first focus on condition SS_3. Add to $\mathbf{M}_A$ the labeled piece $\boldsymbol{\mu}_5$ shown on the bottom part of the figure. We have $\boldsymbol{\mu}_1 = \boldsymbol{\mu}_3 \bullet \boldsymbol{\mu}_5$, thus A satisfies SS_3. However, the resulting tag machines does not satisfy SS_2. To correct this, an additional transition is added that allows our enriched tag machine A^+ to actually perform first $\boldsymbol{\mu}_3$ and then $\boldsymbol{\mu}_5$: this is shown in the bottom left diagram of figure 2. The resulting tag machine A^+ satisfies all conditions of Theorem 2; hence it is self-synchronizing. Finally, A^+ possesses as set of traces the same set of traces as A.

We could easily redraw the same example in the context of the timed tag systems model based on $\mathcal{T}_{\text{wcet}}$, whose nature is very similar to that of $\mathcal{T}_{\text{dep}}$. We leave this as an exercise to the reader.

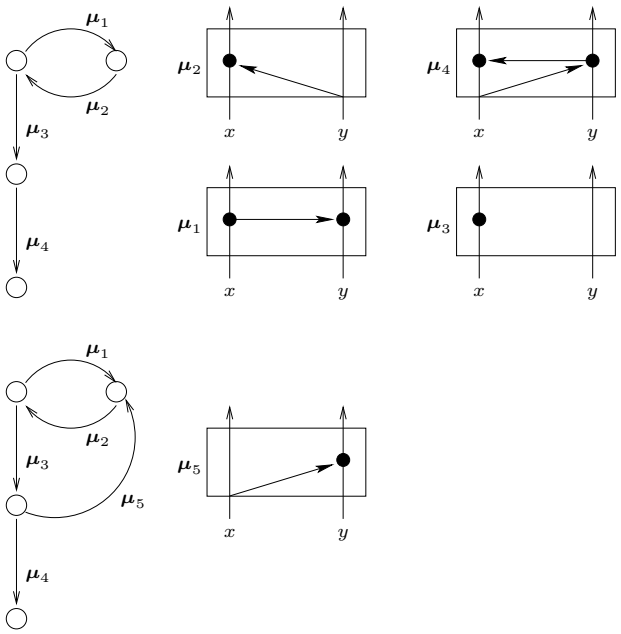


Figure 2: Example: tag machine A (top left) and its self-synchronizing modification A^+ (bottom left), which has same set of traces as the original one.

5. CONCLUSION

We presented a new concept in embedded system design: tag machines. Tag machines are finitary generators of tagged systems traces. As such, they form an important object upon which we will build a theory of composition of heterogeneous systems.

In this paper, we reviewed the notion of tagged systems and explored the relationships between tag machines and tagged systems. We introduced the notion of product of tag machines and gave conditions for tag machines product to be equivalent to parallel composition of tagged systems. The condition is non trivial and requires the machines to be self-synchronizing.

Future work includes the notion of heterogeneous composition and how to compute this with operations on tag machines so that, coupled with the heterogeneous behavioral theory developed for tagged systems, we will have a complete theory of heterogeneous systems giving raise to a formally sound and effective methodology for correct-by-construction deployment.

APPENDIX

Details of the tetris game of Fig. 1

We show in Figure 3 a movie of the tetris game that builds up the heap shown on the third diagram of Fig. 1.

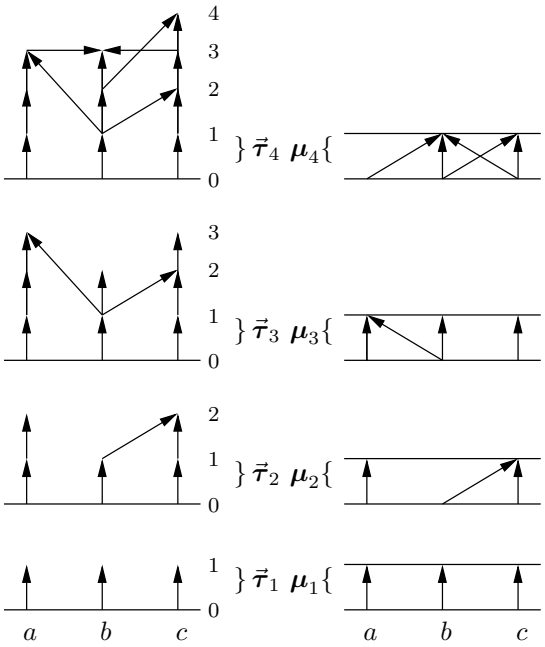


Figure 3: A movie of the tetris game that builds up the heap shown on the third diagram of Fig. 1.

Detailed formulas follow, with the same conventions as for Example 3.5:

$$\tau_0 = \begin{bmatrix} 0 & \epsilon & \epsilon \\ \epsilon & 0 & \epsilon \\ \epsilon & \epsilon & 0 \end{bmatrix} \qquad \mu_1 = \begin{bmatrix} 1/0 & \epsilon & \epsilon \\ \epsilon & 1/0 & \epsilon \\ \epsilon & \epsilon & 1/0 \end{bmatrix}$$

$$\tau_1 = \tau_0 \bullet \mu_1 = \begin{bmatrix} 1 & \epsilon & \epsilon \\ \epsilon & 1 & \epsilon \\ \epsilon & \epsilon & 1 \end{bmatrix} \qquad \mu_2 = \begin{bmatrix} 1/0 & \epsilon & \epsilon \\ \epsilon & 0/\epsilon & 0 \\ \epsilon & \epsilon & 1/0 \end{bmatrix}$$

$$\tau_2 = \tau_1 \bullet \mu_2 = \begin{bmatrix} 2 & \epsilon & \epsilon \\ \epsilon & 1 & 1 \\ \epsilon & \epsilon & 2 \end{bmatrix} \qquad \mu_3 = \begin{bmatrix} 1/0 & \epsilon & \epsilon \\ 0 & 1/0 & \epsilon \\ \epsilon & \epsilon & 1/0 \end{bmatrix}$$

$$\tau_3 = \tau_2 \bullet \mu_3 = \begin{bmatrix} 3 & \epsilon & \epsilon \\ 1 & 2 & 1 \\ \epsilon & \epsilon & 3 \end{bmatrix} \qquad \mu_4 = \begin{bmatrix} 0/\epsilon & 0 & \epsilon \\ \epsilon & 1/0 & 0 \\ \epsilon & 0 & 1/0 \end{bmatrix}$$

$$\tau_4 = \tau_3 \bullet \mu_4 = \begin{bmatrix} 3 & 3 & \epsilon \\ 1 & 3 & 2 \\ \epsilon & 3 & 4 \end{bmatrix}$$

A. REFERENCES

[1] L. de Alfaro, T.A. Henzinger. Interface Automata. In *Proc. of ESEC/FSE 01,* Proc. of the Joint 8th Eur. Software Eng. Conf. and 9th ACM SIGSOFT Intl. Symp. on the Foundations of Software Eng., 2001.

[2] A. Benveniste, B. Caillaud, L. P. Carloni, P. Caspi, and A. L. Sangiovanni-Vincentelli. Heterogeneous Reactive Systems Modeling: Capturing Causality and the Correctness of Loosely Time-Triggered Architectures (LTTA). In G. Buttazzo and S. Edwards, Eds., *Proc. of the 4th. Intl. Conf. on Embedded Software, EMSOFT'04*, ACM 2004, 220–229.

[3] A. Benveniste, B. Caillaud, L. P. Carloni, P. Caspi, and A. L. Sangiovanni-Vincentelli. Composing Heterogeneous Reactive Systems. Submitted for publication.

[4] B. Caillaud, P. Darondeau, L. Hélouët and G. Lesventes. HMSCs as specifications... with PN as completions. In *Modeling and Verification of Parallel Processes,* LNCS, vol. 2067, Springer, 125–152.

[5] S. Gaubert and J. Mairesse. Modeling and Analysis of Timed Petri Nets using Heaps of Pieces. *IEEE Trans. Aut. Control,* 44(4):683–697, 1999

[6] E.A. Lee and A. Sangiovanni-Vincentelli. A Framework for Comparing Models of Computation. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems,* 17(12), 1217–1229, Dec. 1998.

[7] A. Muscholl, D. Peled and Z. Su. Deciding Properties for Message Sequence Charts. In *Proc. of Fossacs'1998,* LNCS 1378, 226–242, 1998.

[8] D. Potop-Butucaru, B. Caillaud and A. Benveniste. Concurrency in Synchronous Systems. In *Proc. of the 4th Int. Conf. on Applications of Concurrency in System Design (ACSD).* Hamilton, Canada, 2004.

[9] D. Potop-Butucaru and B. Caillaud. Correct-by-construction asynchronous implementation of modular synchronous specifications. In *Proc. of the 5th Int. Conf. on Applications of Concurrency in System Design (ACSD).* Saint-Malo, France, 2005.

[10] G. Viennot. Heaps of pieces. 1: basic definitions and combinatorial lemmas. In Labelle and Leroux Eds., *Combinatoire Enumérative,* Lect. Notes in Math., vol 1234, 321–350, 1986.