# A Tag Contract Framework for Heterogeneous Systems

Thi Thieu Hoa Le, Roberto Passerone, Uli Fahrenberg, Axel Legay

# A Tag Contract Framework for Heterogeneous Systems

Thi Thieu Hoa Le[1], Roberto Passerone[1], Uli Fahrenberg[2], and Axel Legay[2]

[1] DISI, University of Trento, Italy
[2] INRIA/IRISA, Rennes, France

**Abstract.** In the distributed development of modern IT systems, contracts play a vital role in ensuring interoperability of components and adherence to specifications. The design of embedded systems, however, is made more complex by the heterogeneous nature of components, which are often described using different models and interaction mechanisms. Composing such components is generally not well-defined, making design and verification difficult. Several frameworks, both operational and denotational, have been proposed to handle heterogeneity using a variety of approaches. However, the application of heterogeneous operational models to contract-based design has not yet been investigated. In this work, we adopt the operational mechanism of tag machines to represent heterogeneous systems and construct a full contract model. We introduce heterogeneous composition, refinement, dominance, and compatibility between contracts, altogether enabling a formalized and rigorous design process for heterogeneous systems.

## 1 Introduction

Modern computing systems are increasingly being built by composing components which are developed concurrently by different design teams. In such a paradigm, the distinction between what is constrained on environments, and what must be guaranteed by a system given the constraint satisfaction, reflects the different roles and responsibilities in the system design procedure. Such distinction can be captured by a component model called *contract* [1]. Formally, a contract is a pair of *assumptions* and *guarantees* which intuitively are properties that must be satisfied by all inputs and outputs of a design, respectively. The separation between assumptions and guarantees supports the distributed development of complex systems and allows subsystems to synchronize by relying on associated contracts.

In the particular context of embedded systems, *heterogeneity* is a typical characteristic since these systems are usually composed from parts developed using different methods, time models and interaction mechanisms. To deal with heterogeneity, several modeling frameworks have been proposed oriented towards the representation and simulation of heterogeneous systems, such as the Ptolemy framework [2], or towards the unification of their interaction paradigms, such as those based on *tagged events* [3]. The latter can capture different notions of time, e.g., physical time, logical time, and relate them by mapping tagged events over a common tag structure. However, due to the significant inherent complexity of heterogeneity, there have been only very few attempts at addressing heterogeneity in contract-based models. For instance, the HRC model

from the SPEEDS project[3] was designed to deal with different viewpoints (functional, time, safety, etc.) of a single component [4,5]. However, the notion of heterogeneity in general is much broader than that between multiple viewpoints, and must take into account diverse interaction paradigms. Meanwhile, heterogeneous modeling frameworks have not been related to contract-based design flows. This has motivated us to study a methodology which allows heterogeneous systems to be modeled and interconnected in a contract-based fashion. To this end, we advocate using Tag Machines [6] (TMs) to represent sets of tagged events or *tag systems*. The original notion of TMs was *homogeneous* [6] and was only recently extended to become *heterogeneous* [7]. Our main contribution in this paper is a tag contract framework built on top of heterogeneous TMs and the definition of a full set of contract operations such as implementation satisfaction, contract refinement, contract dominance and contract compatibility.

The rest of the paper is organized as follows. In Sect. 2, we recall basic notions of tag behaviors and tag machines. Section 3 presents our tag contract methodology for heterogeneous systems built on top of TM operations such as composition, quotient, conjunction and refinement. In the same section, we discuss an application of our methodology to a simplified water control problem and model it using incrementing TMs. Finally we concludes in Sect. 4.

**Related Work.** The notion of contract was first introduced by Bertrand Meyer in his design-by-contract method [8], based on ideas by Dijkstra [9], Lamport [10], and others, where systems are viewed as abstract boxes achieving their common goal by verifying specified contracts. De Alfaro and Henzinger subsequently introduced interface automata [11] for documenting components and establish a more general notion of contract, where pre-conditions and post-conditions, which originally appeared in the form of predicates, are generalized to behavioral interfaces. The differentiation between assumptions and guarantees, which is somewhat implicit in interface automata, is made explicit in the contract framework of the SPEEDS HRC model [4,12]. The relationship between specifications of component behaviors and contracts is further studied by Bauer et al. [13] where a contract framework can be built on top of any *specification theory* equipped with a composition operator and a refinement relation which satisfy certain properties. Trace-based contract theories [4,12] are also demonstrated to be instances of such framework. We take advantage of this formalization in this work to construct our contract theory.

Heterogeneity theory has been evolving in parallel with contract theory, to assist designers in dealing with heterogeneous 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 [2], meaning that each level of the hierarchy is homogeneous while different interacting mechanisms are allowed to be specified at different levels in the hierarchy. The metroII framework [14] relaxes this limitation, and allows designers to build direct model adapters. However, metroII treats components mostly as black boxes using a wrapping mechansim to guarantee flexibility in the system integration, making the development of an underlying theory complex. These and other similar frameworks are mainly focused on handling heterogeneity at the level of simulation. Another body of work is instead oriented to-

[3] www.speeds.eu.com

wards the formal representation, verification and analysis of these system. In particular, Benveniste et al. [3] propose a heterogeneous denotational semantics inspired by the Lee and Sangiovanni-Vincentelli formalism of tag signal models [15], which has been long advocated as a unified modeling framework capable of capturing heterogeneous MoCC. In both models, tags play an important role in capturing various notions of time, where each tag system has its own tag structure expressing an MoCC. Composing such system is thus done by applying mappings between different tag structures. Tag machines (TM) [6] are subsequently introduced as finite representations of homogeneous tag systems. We have chosen to use this formalism for our work, as it provides an operational representation based on rigorous and proven semantics, and extended their definition to encompass heterogeneous components [7]. TMs are quite expressive, e.g., they have been applied to model a job-shop specification [16] 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 [17]. The approach is similar to that of Ptolemy II in that both use actors to represent basic components.

## 2 Background

We consider a component to be a set of behaviors 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 tag ordering is used to resolve the ordering among events at the system interface. For instance, 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.

### 2.1 Tag Behaviors

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 $(\tau, d)$ 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. To construct behaviors incrementally, the events of a variable are indexed into a sequence, with the understanding that events later in the sequence have larger tags [3]. A behavior for a variable $v$ is therefore a function $\mathbb{N} \mapsto (T \times D)$. A behavior $\sigma$ for a component assigns a sequence of events to every variable in $V$, and is thus a function:

$$\sigma \in V \mapsto (\mathbb{N} \mapsto (T \times D)).$$

Each event of behavior $\sigma$ is identified by a tuple $(v, n, \tau, d)$, capturing the $n$-th occurrence of variable $v$ as a pair of a tag $\tau$ 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 $\mathcal{T}$.

Combining behaviors over the same tag structure, or *homogeneous* behaviors, amounts to computing their intersection provided that they are consistent, or *unifiable*, with each other on the shared variables. Two behaviors $\sigma_1$ and $\sigma_2$ are considered unifiable, written $\sigma_1 \bowtie \sigma_2$, if $\sigma_1|_{V_1 \cap V_2} = \sigma_2|_{V_1 \cap V_2}$, where $\sigma|_W$ denotes the restriction of behavior $\sigma$ to the variables in set $W$. When unifiable, we may construct a unified behavior $\sigma = \sigma_1 \sqcup \sigma_2$ on the set of variables $V_1 \cup V_2$ as the combination of the two behaviors:

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

When the behaviors are defined on different tag structures, before unifying them, the set of tags must be equalized by mapping them onto a third tag structure that functions as a common domain. The mappings are called *tag morphisms* and must preserve the order.

**Definition 1 ([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 $\sigma \circ \rho \in V \mapsto (\mathbb{N} \mapsto (T' \times D))$ by simply replacing all tags $\tau$ in $\sigma$ with the image $\rho(\tau)$. Unification of heterogeneous behaviors can be done on the common tag structure.

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 $\mathcal{T}$. Two behaviors $\sigma_1$ and $\sigma_2$ defined on $\mathcal{T}_1$ and $\mathcal{T}_2$ respectively are *unifiable in the heterogeneous sense*, written $\sigma_1 \ {}_{\rho_1}\!\bowtie_{\rho_2} \sigma_2$, if and only if $(\sigma_1 \circ \rho_1) \bowtie (\sigma_2 \circ \rho_2)$. The unified behavior $\sigma$ over $\mathcal{T}$ is then $\sigma = (\sigma_1 \circ \rho_1) \sqcup (\sigma_2 \circ \rho_2)$.

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 behavior 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 as follows:

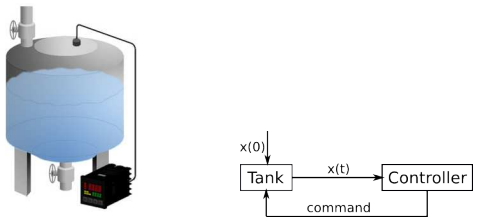
$$(\tau_1, \tau_2) \leq (\tau_1', \tau_2') \iff \tau_1 \leq \tau_1' \wedge \tau_2 \leq \tau_2',$$

where $T_1 \ {}_{\rho_1}\!\times_{\rho_2} T_2 = \{(\tau_1, \tau_2) \in T_1 \times T_2 : \rho_1(\tau_1) = \rho_2(\tau_2)\}$.

*Example 1.* We consider a simplified version of the water controlling system proposed by Benvenuti et al. [12]. It consists of two components: a water tank and a water level controller, connected in a closed-loop fashion, c.f. Fig. 1. We assume that the water level $x(t)$ is changed linearly as follows:

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

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 $\Delta_t$ denotes the time elapsed since $t_0$ at which the tank reaches the maximum/minimum water level $\mathbf{H}$, i.e., $\Delta_t = t - t_0$. The tank behaviors are naturally defined on tag structure $\mathcal{T}_1 = (\mathbb{R}_+ \cup \{\epsilon_1\}, \leq)$ and the controller




(a) A water-controller (b) System diagram

**Fig. 1.** Water controlling system

behaviors on $\mathcal{T}_2 = (\mathbb{N} \cup \{\epsilon_2\}, \leq)$ where $\epsilon_1 = \epsilon_2 = -\infty$. In addition, both components contains behaviors for two system variables, namely the command variable $m$ and the water level $x$, thus $V_1 = V_2 = \{m, x\}$. The command values can be Open $(\mathbf{p})$ or Close $(\mathbf{l})$ and the water level is of positive real type and between 0 and $\mathbf{h}$, i.e., $D_m = \{\mathbf{p}, \mathbf{l}\}$ and $D_x = [0, \mathbf{h}]$. Consider the tank behavior $\sigma_1$ and the controller behavior $\sigma_2$ as follows, where $\sigma(v, n)$ is described when the parameter setting is $\mathbf{f_i} = 2, \mathbf{f_o} = 1, \mathbf{h} = 1$:

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\sigma_1$ : | $m$ : | | 0.5;**p** | | 1.5;**l** | | 2.5;**p** | | 3.5;**l** | | 4.5;**p** | ... |
| | $x$ : | 0;0 | 0.5;0 | 1;0.5 | 1.5;1 | 2;0.5 | 2.5;0 | 3;0.5 | 3.5;1 | 4;0.5 | 4.5;0 | ... |
| $\sigma_2$ : | $m$ | | 1;**p** | | 3;**l** | | 5;**p** | | 7;**l** | | 9;**p** | ... |
| | $x$ : | 0;0 | 1;0 | 2;0.5 | 3;1 | 4;0.5 | 5;0 | 6;0.5 | 7;1 | 8;0.5 | 9;0 | 10;0.5 | ... |

These are different behaviors whose composition is only possible under the presence of morphisms such as $\rho_i : \mathcal{T}_i \mapsto \mathcal{T}_1$ given by $\rho_1(\tau_1) = \tau_1, \rho_2(\tau_2) = 0.5 * \tau_2$.

### 2.2 Operational Tag Machines

Tag machines were first introduced to represent sets of behaviors in a homogeneous context [6] and have been recently extended to encompass the heterogeneous context [7]. In order to construct behaviors, the TM transitions 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 2 ([6]).** *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:*

1. $(T, \cdot)$ *is a monoid with identity element* $\hat{\imath}_{\mathcal{T}}$
2. $\forall \tau_1, \tau_1', \tau_2, \tau_2' \in T : \tau_1 \leq \tau_1' \wedge \tau_2 \leq \tau_2' \Rightarrow \tau_1 \cdot \tau_2 \leq \tau_1' \cdot \tau_2'$
3. $\exists \epsilon_T \in T : \forall \tau \in T : \epsilon_T \leq \tau \wedge \epsilon_T \cdot \tau = \tau \cdot \epsilon_T = \epsilon_T$

Tags can be organized as *tag vectors* $\boldsymbol{\tau} = (\tau^{v_1}, \ldots, \tau^{v_n})$, where $n$ is the number of variables in $V$. During transition, tag vectors evolve according to a matrix $\mu : V \times V \mapsto T$ called a *tag piece* [6]. The new tag vector is $\boldsymbol{\tau}_\mu = \boldsymbol{\tau} \cdot \mu$ where $\tau_\mu^v \stackrel{\text{def}}{=} \max(\tau^u \cdot \mu(u, v))^{u \in V}$ and the maximum is taken with respect to the tag ordering. As the order is partial, the maximum may not exist, in which case the operation is not defined.

Intuitively, a tag piece $\mu$ represents increments in all variable tags over a transition and provides a way to operationally renew them. To represent also changes in variable values, $\mu$ can be labeled with a partial assignment $\nu : V \to D$, which assigns new values to the variables. Tag piece $\mu$ is said to specify an event for all variables for which $\nu$ is defined. In the following, we denote by $\mathfrak{dom}(\nu)$ the domain of $\nu$ and by $L(V, \mathcal{T})$ the universe of all labels over a variable set $V$ and tag structure $\mathcal{T}$.

*Example 2.* The algebraic tag structure $(\mathbb{N} \cup \{-\infty\}, \leq, +)$ can be used to capture logical time by structuring tag pieces $\mu$ so that they represent an integer increment of 1. For instance, $\left[\begin{smallmatrix} 1 & 3 \end{smallmatrix}\right] \cdot \left[\begin{smallmatrix} 0 & 1 \\ -\infty & 1 \end{smallmatrix}\right] = \left[\begin{smallmatrix} 1 & 4 \end{smallmatrix}\right]$. The tag of the second variable is increased by 1 while that of the first variable remains since the least element $-\infty = \epsilon$ is used to cancel the contribution of an entry in the tag vector.

A tag machine $M$ is a finite automaton where transitions are marked by labeled tag pieces, or simply *labels*.

**Definition 3.** *[7] A* tag machine *is a tuple* $(V, \mathcal{T}, S, s_0, F, E)$ *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,*
- $E \subseteq S \times L(V, \mathcal{T}) \times S$ *is the transition relation.*

A run $r$ of a TM is a sequence of states and transitions $r : s_0 \overset{\mu_0}{\to} s_1 \overset{\mu_1}{\to} s_2 \dots s_{m-1} \overset{\mu_{m-1}}{\to} s_m$ 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 TM is used to construct a behavior by following its labeled transitions over a run, and applying the tag pieces sequentially to the initial tag vector $\boldsymbol{\tau} = (\hat{\imath}_{\mathcal{T}}, \dots, \hat{\imath}_{\mathcal{T}})$. A new event is added to the behavior whenever a new value is assigned by the label function $\nu_i$. Run $r$ is *valid* if the concatenation is always defined along the run and $s_m \in F$. The language $\mathcal{L}(M)$ of tag machine $M$ is given by the behaviors of all its valid runs.

### 2.3 Tag Machine Composition

As TMs are used to represent sets of behaviors, combining TMs amounts to considering only behaviors which are consistent with every TMs. In particular, over every transition, the TMs involved in the composition must agree on the tag increment and the value of the shared variables, i.e., their labels are *unifiable*. While TMs defined on the same tag structure, or *homogeneous* TMs, can always be composed, TMs on different tag structures, or *heterogeneous* TMs, can be composed if there exists a pair of *algebraic* tag morphisms mapping the tag structures $\mathcal{T}_1, \mathcal{T}_2$ to a common tag structure $\mathcal{T}$ and preserving the concatenation operator. The homogeneous composition can thus be regarded as a special case of the heterogeneous one when tag morphisms are identity functions mapping a tag to itself.

**Definition 4.** *[7] A tag morphism* $\rho : \mathcal{T} \mapsto \mathcal{T}'$ *is* algebraic *if* $\rho(\hat{\imath}_{\mathcal{T}}) = \hat{\imath}_{\mathcal{T}'}$ *and* $\rho(\epsilon_{\mathcal{T}}) = \epsilon_{\mathcal{T}'}$ *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 labels $\mu_1$ and $\mu_2$ are *unifiable* under morphisms $\rho_1$ and $\rho_2$, written $\mu_1 \ {}_{\rho_1}\!\bowtie_{\rho_2} \mu_2$, whenever for all pairs $(w, v) \in W \times W$ where $W = V_1 \cap V_2$:

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

Their unification $\mu = \mu_1 \ {}_{\rho_1}\!\sqcup_{\rho_2} \mu_2$ is 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 unified labeling function agrees with individual functions on the shared variables:

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

The composition $M = M_1 \ {}_{\rho_1}\|_{\rho_2} \ M_2$ of heterogeneous TMs can then be defined over the unification of heterogeneous tag structures and labels.

**Definition 5.** *[7] The parallel composition of two tag machines $M_1$ and $M_2$ under two algebraic tag morphisms $\rho_1$ and $\rho_2$ is the tag machine $M = M_1 \ {}_{\rho_1}\|_{\rho_2} \ M_2 = (V, \mathcal{T}_1 \ {}_{\rho_1}\!\times_{\rho_2} \mathcal{T}_2, S, s_0, F, E)$ such that*

- $V = V_1 \cup V_2$,
- $S = S_1 \times S_2$, $s_0 = (s_{01}, s_{02})$, $F = F_1 \times F_2$,
- $E = \{((s_1, s_2), \mu_1 \ {}_{\rho_1}\!\sqcup_{\rho_2} \mu_2, (s_1', s_2')) : \mu_1 \ {}_{\rho_1}\!\bowtie_{\rho_2} \mu_2 \wedge (s_i, \mu_i, s_i') \in E_i, i = 1, 2\}$

Since the homogeneous composition is a special case of the heterogeneous one with identity morphisms, we shall omit the morphisms in the homogeneous notations in the sequel.

## 3 A Contract Framework for Heterogeneous Systems

Our goal is to use TMs as an operational means for modeling heterogeneous systems in contract-based design flows. To this end, we equip TMs with essential binary operators such as composition to combine two TMs [7] and refinement, quotient and conjunction to relate their sets of behaviors. Moreover, we limit TMs to their *deterministic* form where labeled tag pieces annotated on transitions going out of a state are all different. On top of these TM operators, we propose a heterogeneous contract theory for TM-based specifications with universal contract operators such as composition, refinement and compatibility.

### 3.1 Tag Machine Operators

Two TMs can be related in a refinement relation when the behavior set of one machine is included in that of the other under the morphisms. In the operational point of view, the refined TM can always take a transition unifiable with that taken by the refining TM. Let $M_i = (V_i, \mathcal{T}_i, S_i, s_{0i}, F_i, E_i)$ be TMs and $\rho_i : \mathcal{T}_i \mapsto \mathcal{T}$ be algebraic tag morphisms, where $i \in \{1, 2\}$. The TM refinement is defined as follows.

**Definition 6.** *$M_1$ refines $M_2$, written $M_1 \ {}_{\rho_1}\preceq_{\rho_2} \ M_2$, if there exists a binary relation $R \subseteq S_1 \times S_2$ such that $R$ contains the initial composite state $(s_{01}, s_{02})$ and for all transitions $(s_1, \mu_1, s_1') \in E_1$ going out of the first component of any state $(s_1, s_2) \in R$ :*

$$\exists (s_2, \mu_2, s_2') \in E_2 : \mu_1 \ {}_{\rho_1}\bowtie_{\rho_2} \ \mu_2 \wedge (s_1', s_2') \in R \wedge (s_1' \in F_1 \Rightarrow s_2' \in F_2)$$

The following theorem shows that our TM theory supports (homogenous) *independent implementability*: refinement is preserved when composing components.

**Theorem 1.** *Let $M_i'$ be TMs defined on $\mathcal{T}_i$ and $V_i$. If $M_1 \preceq M_1'$ and $M_2 \preceq M_2'$ then $M_1 \ {}_{\rho_1}\|_{\rho_2} \ M_2 \preceq M_1' \ {}_{\rho_1}\|_{\rho_2} \ M_2'$.*

We remark that Theorem 1 only holds for *homogenous* TM refinement, and note that heterogeneous refinement in general is *not* preserved even by homogeneous composition. The reason is that the morphisms involved in the former are generally many-to-one functions and can map two different tags into the same tag.

*Example 3.* We consider an example where:

- $\mathcal{T}_1 = \{\tau_1\}, \mathcal{T}_2 = \{\tau_2, \tau_2'\}$
- $V_1 = V_2 = \{z\}, D_z = \{\top\}$
- $\rho_1(\tau_1) = \rho_2(\tau_2) = \rho_2(\tau_2') = \tau$

Let $M_i, M_i'$ be defined on $\mathcal{T}_i$ and $V_i$ where $i \in \{1, 2\}$. For the sake of simplicity, assume all TMs have a single state which is both initial and accepting state. In addition, there is only one self-loop at this state annotated with $\mu_i$ for machine $M_i$ and $\mu_i'$ for machine $M_i'$ such that $\mu_1 = \mu_1' = [\tau_1], \mu_2 = [\tau_2], \mu_2' = [\tau_2'], \nu_1(z) = \nu_1'(z) = \nu_2(z) = \nu_2'(z) = \top$. It is easy to see that $M_1 \ {}_{\rho_1}\preceq_{\rho_2} \ M_2$ since $\mu_1 \ {}_{\rho_1}\bowtie_{\rho_2} \ \mu_2$ and $M_1' \ {}_{\rho_1}\preceq_{\rho_2} \ M_2'$ since $\mu_1' \ {}_{\rho_1}\bowtie_{\rho_2} \ \mu_2'$. However, $(M_1 \parallel M_1') \ {}_{\rho_1}\not\preceq_{\rho_2} \ (M_2 \parallel M_2')$ since the right composition is empty while the left is not.

While the refinement operator enables us to compare two TMs in terms of sets of behaviors, the composition and quotient operators allow us to synthesize specifications. The TM composition computes the most general specification that retains all unifiable behaviors of two TMs. The dual operator to TM composition is TM quotient which computes the maximal specification as follows.

**Definition 7.** *The quotient $M_1 {}_{\rho_1}/_{\rho_2} M_2$ is a machine $M = (V, \mathcal{T}_{12}, S, s_0, F, E)$, where*

- $V = V_1 \cup V_2, \mathcal{T}_{12} = \mathcal{T}_1 \ {}_{\rho_1}\times_{\rho_2} \ \mathcal{T}_2, s_0 = (s_{01}, s_{02})$,
- $S = (S_1 \times S_2) \cup \{\mathfrak{u}\}$, *where* $\mathfrak{u}$ *is a new* universal *state,*

– $F = ((S_1 \times S_2) \setminus ((S_1 \setminus F_1) \times F_2)) \cup \{\mathfrak{u}\} = (F_1 \times F_2) \cup (S_1 \times (S_2 \setminus F_2)) \cup \{\mathfrak{u}\}$, *and*

$$\begin{aligned} E = \{&((s_1, s_2), \mu_1 \,{}_{\rho_1}\sqcup_{\rho_2} \mu_2, (s_1', s_2')) \mid \\ &(\mu_1 \,{}_{\rho_1}\bowtie_{\rho_2} \mu_2) \wedge ((s_1, \mu_1, s_1') \in E_1) \wedge ((s_2, \mu_2, s_2') \in E_2)\} \\ \cup \{&((s_1, s_2), \mu_1 \,{}_{\rho_1}\sqcup_{\rho_2} \mu_2, \mathfrak{u}) \mid \\ &(\forall s_2' \in S_2 : (s_2, \mu_2, s_2') \notin E_2) \wedge (\exists \mu_1 \in L(V_1, \mathcal{T}_1) : \mu_1 \,{}_{\rho_1}\bowtie_{\rho_2} \mu_2)\} \\ \cup \{&(\mathfrak{u}, \mu, \mathfrak{u}) \mid \mu \in L(V, \mathcal{T}_{12})\}. \end{aligned}$$

We give an example of a quotient construction in Fig. 4. The dual relation between composition and quotient is presented in the next theorem.

**Theorem 2.** *The quotient $M$ satisfies refinement $(M_2 \,{}_{\mathsf{id}_2}\|_{\mathsf{proj}_2} M) \,{}_{\mathsf{proj}_1'}\preceq_{\mathsf{id}_1} M_1$ where:*

$$\begin{aligned} \forall i \in \{1,2\}, \forall \tau_i \in \mathcal{T}_i : \mathsf{id}_i(\tau_i) &= \tau_i \\ \forall i \in \{1,2\}, \forall (\tau_1, \tau_2) \in \mathcal{T}_{12} : \mathsf{proj}_i((\tau_1, \tau_2)) &= \tau_i \\ \forall (\tau_2, \tau_{12}) \in \mathcal{T}_2 \,{}_{\mathsf{id}_2}\times_{\mathsf{proj}_2} \mathcal{T}_{12} : \mathsf{proj}_1'((\tau_2, \tau_{12})) &= \mathsf{proj}_1(\tau_{12}) \\ \forall (\tau_1, \tau_{12}) \in \mathcal{T}_1 \,{}_{\mathsf{id}_1}\times_{\mathsf{proj}_1} \mathcal{T}_{12} : \mathsf{proj}_2'((\tau_1, \tau_{12})) &= \mathsf{proj}_2(\tau_{12}) \end{aligned}$$

*Moreover, for $M'$ defined on $\mathcal{T}_{12}$ and $V$: $(M_2 \,{}_{\mathsf{id}_2}\|_{\mathsf{proj}_2} M') \,{}_{\mathsf{proj}_1'}\preceq_{\mathsf{id}_1} M_1 \Rightarrow M' \preceq M$*

Thus, the quotient $M$ is the *greatest*, in the (homogeneous) refinement preorder, of all TMs $M'$ defined in Theorem 2. This universal property is generally expected of quotients [13], and it alone implies that the quotient is uniquely defined up to two-sided homogeneous refinement [18]. As an example, Fig. 3(c) shows a homogeneous quotient and Fig. 4(b) shows a heterogeneous quotient using the morphisms of Example 1.

Finally, the operator of *heterogeneous conjunction*, denoted $\,{}_{\rho_1}\curlywedge_{\rho_2}$, is defined as the greatest lower bound of the refinement order. Conjunction, therefore, amounts to computing the intersection of the behavior sets, in order to find the largest common refinement. Thus, for tag machines, conjunction can be computed similarly to composition. The two operators, however, serve very different purposes, and must not therefore be confused.

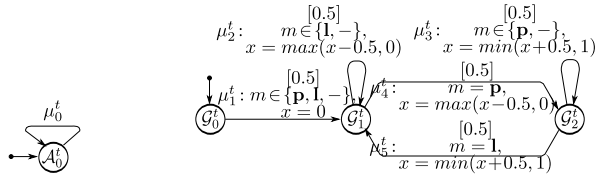
### 3.2 Tag Contracts

We use the term *tag contract* to mean that in our framework each contract is coupled with an algebraic tag structure, thereby allowing the contract assumption and guarantee to be represented as TMs.

**Definition 8.** *A tag contract $\mathcal{C}$ is a homogeneous pair of TMs $(M_\mathcal{A}, M_\mathcal{G})$ defined over an algebraic tag structure $\mathcal{T}$ and variable set $V$:*

$$\begin{aligned} M_\mathcal{A} &\stackrel{\text{def}}{=} (V, \mathcal{T}, S_\mathcal{A}, s_{0\mathcal{A}}, F_\mathcal{A}, E_\mathcal{A}) \\ M_\mathcal{G} &\stackrel{\text{def}}{=} (V, \mathcal{T}, S_\mathcal{G}, s_{0\mathcal{G}}, F_\mathcal{G}, E_\mathcal{G}) \end{aligned}$$

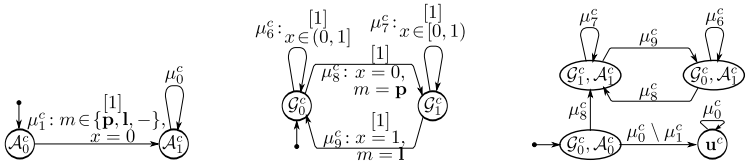
*Example 4.* We consider the simplified water controlling system in Example 1 and present a contract for each component. To simplify the behavioral construction, we rely on a special clock $\mathfrak{inc}$ added to the variable set of both components. Tag pieces $\mu$ are then structured to represent an increment of $\delta$ by always assigning $\delta$ to $\mu(\mathfrak{inc}, \mathfrak{inc})$ and assigning $\delta$ to all entries $\mu(\mathfrak{inc}, v)$ where $v \in \mathfrak{dom}(\mu)$, and the least element $-\infty$ to other entries. The tags of $x$ and $m$ are thus renewed to the tag of clock $\mathfrak{inc}$ over every transition. To keep the figures readable we represent tag pieces as $[\delta]$. In addition, the clock value is always equal to its tag and thus is omitted from the labeling function.




(a) $M_{\mathcal{A}_t}$ (b) $M_{\mathcal{G}_t}$

**Fig. 2.** The tank contract

Figure 2 depicts the tank contract $\mathcal{C}_t = (M_{\mathcal{A}_t}, M_{\mathcal{G}_t})$ which guarantees a linear evolution of the water level $x(t)$ (Fig. 2(b)) given the assumption satisfaction (Fig. 2(a)). That is, the water level will evolve linearly as specified in Example 1, provided that the controlling command is received at the right time (i.e., open when the tank is empty and close when it is full). For the sake of simplicity, the events described by the tank contract are timestamped periodically every 0.5 time unit.




(a) $M_{\mathcal{A}_c}$ (b) $M_{\mathcal{G}_c}$ (c) $M_{\mathcal{G}_c}/M_{\mathcal{A}_c}$

**Fig. 3.** The controller contract

The controller contract is shown Fig. 3, where it assumes the tank to be empty initially (Fig. 3(a)), i.e., $x = 0$ and places no requirement on its output which is the command signal. As long as such assumption is satisfied, the controller guarantees (Fig. 3(b)) to send a proper command upon knowing of the tank emptiness or fullness. Intuitively, the controller behaviors ensure timely control over the water evolution while the tank behaviors accept untimely control and allow water spillages or shortages. While the tank system uses physical time to stamp its behaviors, the controller system instead timestamps its events logically, which can be described by the integer tag set

$\mathbb{N}$. In both figures, the initial states are marked with short arrows arriving at them and all states are accepting states. For the 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 within an interval, we label with the tag piece expressions such as $x \in (0, 1)$, meaning that in such an event $x$ can take any value between 0 and 1. Similarly, $m \in \{\mathbf{p}, \mathbf{l}, -\}$ means the command value can either be open, close or undefined. In addition, we use $\mu_0^t$ to denote the universe set of labels $L(V_1, \mathcal{T}_1)$ and $\mu_0^c$ the set of labels $L(V_2, \mathcal{T}_2)$.

The tag contract semantics is subsequently defined through the notions of contract environments and implementations. Let $M_\mathcal{I}$ and $M_\mathcal{E}$ be TMs defined over tag structure $\mathcal{T}$ and variable set $V$ in Def. 8. We call $M_\mathcal{E}$ an environment of contract $\mathcal{C}$ when $M_\mathcal{E}$ refines $M_\mathcal{A}$. Let $[\![\mathcal{C}]\!]_\mathsf{e}$ be the set of all such environments, we call $M_\mathcal{I}$ an implementation of contract $\mathcal{C}$, if it holds that $\forall M_\mathcal{E} \in [\![\mathcal{C}]\!]_\mathsf{e} : M_\mathcal{I} \parallel M_\mathcal{E} \preceq M_\mathcal{G} \parallel M_\mathcal{E}$. The set of implementations is similarly denoted by $[\![\mathcal{C}]\!]_\mathsf{p}$. Hence, the implementation checking is done based on instantiating all possible environments of a contract. Such operation can be performed independently of the assumption instantiation only when the contract is in a special *normalized* form.

**Definition 9.** *A tag contract $\mathcal{C} = (M_\mathcal{A}, M_\mathcal{G})$ is in normalized form if and only if:*

$$\forall M_\mathcal{I} : M_\mathcal{I} \in [\![\mathcal{C}]\!]_\mathsf{p} \Leftrightarrow M_\mathcal{I} \preceq M_\mathcal{G}.$$

*Example 5.* We use the tag contracts in Example 4 and perform the quotient between the guarantees and assumptions in order to normalize them. Since the tank assumption is the universe of all possible behaviors, i.e., $\Sigma(V_1, \mathcal{T}_1)$, normalizing the tank guarantee adds no more behaviors to the guarantee, i.e., $M_{\mathcal{G}_t}/M_{\mathcal{A}_t} = M_{\mathcal{G}_t}$. Figure 3(c), on the other hand, shows the normalized controller guarantee having more behaviors than the un-normalized one. It is easy to see that the behavior $\sigma_1$ in Example 1 is included in $M_{\mathcal{G}_t}$ and $\sigma_2$ is in $M_{\mathcal{G}_c}/M_{\mathcal{A}_c}$.

The following theorem states the preservation of tag contract semantics under the normalization operation.

**Theorem 3.** *Normalizing a tag contract $\mathcal{C} = (M_\mathcal{A}, M_\mathcal{G})$ can be done by replacing $M_\mathcal{G}$ with $M_\mathcal{G}/M_\mathcal{A}$ without affecting its semantics.*

With Theorem 3, whenever a tag contract is in a normalized form, checking contract satisfaction is reduced to finding a refinement relation between two TMs. As we will see later, working with normalized tag contracts can simplify the formalization of contract operators as well as provide a unique representation for equivalent contracts, thus we will often assume contracts to be in normalized form.

Like other contract frameworks, our tag contract one can identify the refinement and dominance relations between two tag contracts whenever they exist.

**Tag Contract Refinement.** The refinement relation between two tag contracts is subject to the tag morphisms and is determined by that between their sets of implementations and environments as follows. Let $\mathcal{C}_i = (M_{\mathcal{A}_i}, M_{\mathcal{G}_i})$ be tag contracts defined on $\mathcal{T}_i$ and $V_i$ and $\rho_i : \mathcal{T}_i \mapsto \mathcal{T}$ be algebraic tag morphisms where $i \in \{1, 2\}$

**Definition 10.** *Contract $\mathcal{C}_1$ refines contract $\mathcal{C}_2$ under morphisms $\rho_1$ and $\rho_2$, written $\mathcal{C}_1 \ {}_{\rho_1}\preceq_{\rho_2} \mathcal{C}_2$, if the following two conditions hold:*

1. $\forall M_{\mathcal{E}_2} \in [\![\mathcal{C}_2]\!]_{\mathsf{e}} : \exists M_{\mathcal{E}_1} \in [\![\mathcal{C}_1]\!]_{\mathsf{e}} : M_{\mathcal{E}_2} \ {}_{\rho_2}\preceq_{\rho_1} M_{\mathcal{E}_1}$
2. $\forall M_{\mathcal{I}_1} \in [\![\mathcal{C}_1]\!]_{\mathsf{p}} : \exists M_{\mathcal{I}_2} \in [\![\mathcal{C}_2]\!]_{\mathsf{p}} : M_{\mathcal{I}_1} \ {}_{\rho_1}\preceq_{\rho_2} M_{\mathcal{I}_2}$

The following theorem shows that when two tag contracts are in normalized form, checking refinement can be done at the *syntactic* level, i.e., by finding a TM refinement relation between their assumptions and guarantees.

**Theorem 4.** *For two normalized tag contracts $\mathcal{C}_1$ and $\mathcal{C}_2$:*

$$\mathcal{C}_1 \ {}_{\rho_1}\preceq_{\rho_2} \mathcal{C}_2 \Leftrightarrow (M_{\mathcal{A}_2} \ {}_{\rho_2}\preceq_{\rho_1} M_{\mathcal{A}_1}) \wedge (M_{\mathcal{G}_1} \ {}_{\rho_1}\preceq_{\rho_2} M_{\mathcal{G}_2})$$

**Tag Contract Composition and Dominance.** In composing two heterogeneous tag contracts, it is essential to guarantee that composing implementations of each tag contract remains meaningful and results in a new implementation of the newly-composed contract. In addition, every environment of the composite contract should be able to work with any implementation of an individual contract in a way that their composition does not violate the other contract assumption. In fact, there exists a class of contracts, including the composite contract, able to provide such desirable consequences. We refer to them as *dominating* contracts [13].

**Definition 11.** *A contract $\mathcal{C} = (M_{\mathcal{A}}, M_{\mathcal{G}})$ is said to* dominate *the tag contract pair $(\mathcal{C}_1, \mathcal{C}_2)$ under morphisms $\rho_i$ if :*

1. *$\mathcal{C}$ is defined over tag structure $\mathcal{T}_{12} \stackrel{\text{def}}{=} \mathcal{T}_1 \ {}_{\rho_1}\times_{\rho_2} \mathcal{T}_2$ and variable set $V = V_1 \cup V_2$*
2. $\forall M_{\mathcal{I}_1} \in [\![\mathcal{C}_1]\!]_{\mathsf{p}}, \forall M_{\mathcal{I}_2} \in [\![\mathcal{C}_2]\!]_{\mathsf{p}} : M_{\mathcal{I}_1} \ {}_{\rho_1}\|_{\rho_2} \ M_{\mathcal{I}_2} \in [\![\mathcal{C}]\!]_{\mathsf{p}}$
3. $\forall M_{\mathcal{E}} \in [\![\mathcal{C}]\!]_{\mathsf{e}} : \begin{cases} \forall M_{\mathcal{I}_1} \in [\![\mathcal{C}_1]\!]_{\mathsf{p}} : (M_{\mathcal{I}_1} \ {}_{\mathsf{id}_1}\|_{\mathsf{proj}_1} M_{\mathcal{E}}) \ {}_{\mathsf{proj}_2'}\preceq_{\mathsf{id}_2} M_{\mathcal{A}_2} \wedge \\ \forall M_{\mathcal{I}_2} \in [\![\mathcal{C}_2]\!]_{\mathsf{p}} : (M_{\mathcal{I}_2} \ {}_{\mathsf{id}_2}\|_{\mathsf{proj}_2} M_{\mathcal{E}}) \ {}_{\mathsf{proj}_1'}\preceq_{\mathsf{id}_1} M_{\mathcal{A}_1} \end{cases}$

*where the morphisms are defined as in Theorem 2.*

The composition of heterogeneous tag contracts can then be defined as follows.

**Definition 12.** *The composition of tag contracts $\mathcal{C}_1$ and $\mathcal{C}_2$, written $\mathcal{C}_1 \ {}_{\rho_1}\|_{\rho_2} \ \mathcal{C}_2$, is another tag contract $((M_{\mathcal{A}_1} \, {}_{\rho_1}/_{\rho_2} M_{\mathcal{G}_2}) \curlywedge (M_{\mathcal{A}_2} \, {}_{\rho_2}/_{\rho_1} M_{\mathcal{G}_1})_{\mathsf{swap}}, M_{\mathcal{G}_1} \ {}_{\rho_1}\|_{\rho_2} \ M_{\mathcal{G}_2})$ where $\mathsf{swap} : \mathcal{T}_2 \ {}_{\rho_2}\times_{\rho_1} \mathcal{T}_1 \mapsto \mathcal{T}_1 \ {}_{\rho_1}\times_{\rho_2} \mathcal{T}_2$ is such that $\mathsf{swap}((\tau_2, \tau_1)) = ((\tau_1, \tau_2))$ and $M_{\mathsf{swap}}$ is $M$ where all pieces $\mu$ are replaced with $\mu \circ \mathsf{swap}$.*

The following theorem states important results for normalized tag contracts. Namely, their contract composition dominates the individual contracts and is the *least*, in the homogeneous refinement order, of all contracts dominating them under the same morphisms.

**Theorem 5.** *Let $\mathcal{C}_i$ be normalized tag contracts and $\mathcal{C} = \mathcal{C}_1 \ {}_{\rho_1}\|_{\rho_2} \mathcal{C}_2$. Then:*

1. *$\mathcal{C}$ dominates the contract pair $(\mathcal{C}_1, \mathcal{C}_2)$ under morphisms $\rho_1$ and $\rho_2$.*
2. *If $\mathcal{C}'$ dominates $(\mathcal{C}_1, \mathcal{C}_2)$ under morphisms $\rho_1$ and $\rho_2$ then $\mathcal{C} \preceq \mathcal{C}'$.*

The next theorem is another of *independent implementability*: homogeneous tag contract refinement is preserved under the heterogeneous contract composition.

**Theorem 6.** *Let $\mathcal{C}'_i$ be normalized tag contracts defined on $\mathcal{T}_i$ and $V_i$ such that $\mathcal{C}'_i \preceq \mathcal{C}_i$ where $i \in \{1, 2\}$.*

1. *If $\mathcal{C}$ dominates $(\mathcal{C}_1, \mathcal{C}_2)$ under morphisms $\rho_1$ and $\rho_2$ then it also dominates $(\mathcal{C}'_1, \mathcal{C}'_2)$ under the same morphisms.*
2. $(\mathcal{C}'_1 \ {}_{\rho_1}\|_{\rho_2} \ \mathcal{C}'_2) \preceq (\mathcal{C}_1 \ {}_{\rho_1}\|_{\rho_2} \ \mathcal{C}_2)$.

**Tag Contract Compatibility.** Of particular interest is the notion of *compatibility* between contracts. This notion depends critically on the particular partition of the variables into inputs and outputs, called *profile*.

**Definition 13.** *A profile is a tuple $\pi = (\mathfrak{i}, \mathfrak{o})$ where $\mathfrak{i}/\mathfrak{o}$ denotes a set of input/output ports, respectively. Moreover these sets have to be disjoint, i.e., $\mathfrak{i} \cap \mathfrak{o} = \emptyset$.*

A tag contract defined on some variable set $V$ has profile $\pi = (V^{\mathfrak{i}}, V^{\mathfrak{o}})$ if and only if $V = V^{\mathfrak{i}} \cup V^{\mathfrak{o}}$. When composing contracts $\mathcal{C}_i$ with profiles $\pi_i$, we enforce the property that each output port should be controlled by at most one contract, i.e., $V_1^{\mathfrak{o}} \cap V_2^{\mathfrak{o}} = \emptyset$. The composite contract is defined as in Def. 12 and with profile $\pi = ((V_1^{\mathfrak{i}} \cup V_2^{\mathfrak{i}}) \setminus (V_1^{\mathfrak{o}} \cup V_2^{\mathfrak{o}}), V_1^{\mathfrak{o}} \cup V_2^{\mathfrak{o}})$.

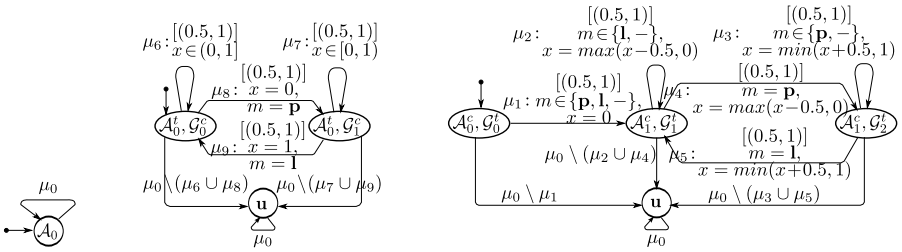
*Example 6.* The tank and controller contracts in Example 4 are naturally associated respectively with profiles $\pi_1 = (\{x\}, \{cmd\})$ and $\pi_2 = (\{cmd\}, \{x\})$. The profile of their composition is then $\pi_1 = (\emptyset, \{x, cmd\})$.

Intuitively, a contract can only constrain its inputs provided by its environment and provide certain guarantees on its outputs. This is visualized by enforcing the contract assumption to be *output-enabled* and the contract guarantee to be *input-enabled*. A tag machine is said to be input(output)-enabled when it accepts all possible combinations of the input(output) values.

When composing different contracts, it is often desirable to ensure that there exists some environment which can discharge all assumptions made by the composition. The contract compatibility is therefore essential in caring for such a need. Two tag contracts $\mathcal{C}_1$ and $\mathcal{C}_2$ are said to be *compatible* if there exists a contract $\mathcal{C}_e$ defined over the composite tag structure $\mathcal{T}_{12} = \mathcal{T}_1 \ {}_{\rho_1}\times_{\rho_2} \mathcal{T}_2$ and variable set $V = V_1 \cup V_2$ with profile $\pi_e = (V_1^{\mathfrak{o}} \cup V_2^{\mathfrak{o}}, (V_1^{\mathfrak{i}} \cup V_2^{\mathfrak{i}}) \setminus (V_1^{\mathfrak{o}} \cup V_2^{\mathfrak{o}}))$ such that:

- $M_{\mathcal{A}_e} \equiv M_{\mathfrak{u}}$, meaning that $\mathcal{C}_e$ makes no assumptions on its inputs and accepts all possible behaviors defined on $L(V, \mathcal{T}_{12})$. In addition, the composition of $\mathcal{C}_e$ and $\mathcal{C}_1 \ {}_{\rho_1}\|_{\rho_2} \ \mathcal{C}_2 = (M_{\mathcal{A}}, M_{\mathcal{G}}) = ((M_{\mathcal{A}_1} \, {}_{\rho_1}/_{\rho_2} M_{\mathcal{G}_2}) \curlywedge (M_{\mathcal{A}_2} \, {}_{\rho_2}/_{\rho_1} M_{\mathcal{G}_1})_{\mathsf{swap}}, M_{\mathcal{G}_1} \ {}_{\rho_1}\|_{\rho_2} M_{\mathcal{G}_2})$ should also weaken the assumption made on its environment to the greatest extent. That is $(M_{\mathcal{A}_e}/M_{\mathcal{G}}) \curlywedge (M_{\mathcal{A}}/M_{\mathcal{G}_e}) \equiv M_{\mathfrak{u}}$ as well.
- $M_{\mathcal{G}_e}$ is input-enabled so as to make contract $\mathcal{C}_e$ consistent.

In looking for such contract, it is important to notice that $M_{\mathcal{A}_e} \equiv M_{\mathfrak{u}}$, therefore the condition of $(M_{\mathcal{A}_e}/M_{\mathcal{G}}) \curlywedge (M_{\mathcal{A}}/M_{\mathcal{G}_e}) \equiv M_{\mathfrak{u}}$ holds when $M_{\mathcal{G}_e}$ is a refinement of $M_{\mathcal{A}}$. Therefore, the compatibility check is reduced to finding a refinement of $M_{\mathcal{A}}$ such that it is input-enabled.




(a) $M_{\mathrm{u}}$ (b) $M_{\mathcal{A}_t\ \rho_1/_{\rho_2} M_{\mathcal{G}_c}}$ (c) $(M_{\mathcal{A}_c\ \rho_2/_{\rho_1} M_{\mathcal{G}_t}})_{\mathsf{swap}}$

**Fig. 4.** Quotient components of the composite assumption of $\mathcal{C}_1\ {}_{\rho_1}\|_{\rho_2}\ \mathcal{C}_2$

*Example 7.* We consider again the water tank controlling problem in Example 4 and the two contracts on the tank and the controller. The composite assumption of these two contracts is the conjunction of the two quotients shown in Fig. 4(b) and 4(c). Since it is easy to verify that both quotients are equivalent to $M_{\mathrm{u}}$, therefore an input-enabled refinement of the composite assumption exists and we can take for example $(M_{\mathcal{A}_1\ \rho_1/_{\rho_2} M_{\mathcal{G}_2}})$ or $(M_{\mathcal{A}_2\ \rho_2/_{\rho_1} M_{\mathcal{G}_1}})_{\mathsf{swap}}$. Consequently the two contracts are compatible.

## 4 Conclusions

We have presented a modeling methodology based on contracts for designing heterogeneous distributed systems. Heterogeneous systems are usually characterized by their heterogeneity of components which can be of very different nature, e.g. real-time component or logical control component. Without a heterogeneous mechanism, modeling the interaction between components may not be feasible, thereby making it difficult to do verification and analysis based on the known properties of the components. This problem is further complicated for distributed systems where components are developed concurrently by different design teams and are synchronized by relying on their associated contracts. To deal with such problem, we adopt the TM formalism [6,7] for specifying components in terms of operational behaviors. We subsequently propose a contract methodology for synchronizing heterogeneous components based on a set of useful operations on TMs such as composition, quotient and refinement.

Our next step is to demonstrate our methodology through a prototype tool and validate it through case studies. The development of such a tool is therefore included in our future work.

## References

1. Sangiovanni-Vincentelli, A., Damm, W., Passerone, R.: Taming Dr. Frankenstein: Contract-based design for cyber-physical systems. European Journal of Control **18**(3) (2012) 217–238
2. Brooks, C., Lee, E., Liu, X., Neuendorffer, S., Zhao, Y., (eds.), H.Z.: Heterogeneous concurrent modeling and design in Java (Volume 1: Introduction to Ptolemy II). Technical Report UCB/ERL M05/21, University of California, Berkeley (July 2005)

3. Benveniste, A., Caillaud, B., Carloni, L.P., Caspi, P., Sangiovanni-Vincentelli, A.L.: Composing heterogeneous reactive systems. ACM Trans. Embed. Comput. Syst. **7**(4) (2008) 43:1–43:36
4. Benveniste, A., Caillaud, B., Ferrari, A., Mangeruca, L., Passerone, R., Sofronis, C.: Multiple viewpoint contract-based specification and design. In: Formal Methods for Components and Objects, $6^{th}$ International Symposium (FMCO 2007), Amsterdam, The Netherlands, October 24–26, 2007. Volume 5382 of LNCS. Springer (2008) 200–225
5. Damm, W., Hungar, H., Josko, B., Peikenkamp, T., Stierand, I.: Using contract-based component specifications for virtual integration testing and architecture design. In: Proceedings of the conference on Design, Automation and Test in Europe (DATE'11), Grenoble, France (March 14-18 2011)
6. Benveniste, A., Caillaud, B., Carloni, L.P., Sangiovanni-Vincentelli, A.: Tag machines. In: Proceedings of the 5th ACM International Conference On Embedded Software. EMSOFT05, Jersey City, NJ, USA, ACM (September 18-22 2005) 255–263
7. Le, H.T.T., Passerone, R., Fahrenberg, U., Legay, A.: Tag machines for modeling heterogeneous systems. In: Proceedings of the $13^{th}$ International Conference on Application of Concurrency to System Design. ACSD13, Barcelona, Spain (July 8–10, 2013)
8. Meyer, B.: Applying "Design by contract". Computer **25**(10) (1992) 40–51
9. Dijkstra, E.W.: Guarded commands, non-determinancy and a calculus for the derivation of programs. In: Language Hierarchies and Interfaces. Volume 46 of LNCS., Springer (1975) 111–124
10. Lamport, L.: win and sin: Predicate transformers for concurrency. ACM Trans. Program. Lang. Syst. **12**(3) (1990) 396–428
11. de Alfaro, L., Henzinger, T.A.: Interface automata. SIGSOFT Softw. Eng. Notes **26**(5) (September 2001) 109–120
12. Benvenuti, L., Ferrari, A., Mangeruca, L., Mazzi, E., Passerone, R., Sofronis, C.: 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) 142–147
13. Bauer, S.S., David, A., Hennicker, R., Larsen, K.G., Legay, A., Nyman, U., Wasowski, A.: Moving from specifications to contracts in component-based design. In: FASE. Volume 7212., Springer (2012) 43–58
14. Davare, A., Densmore, D., Guo, L., Passerone, R., Sangiovanni-Vincentelli, A.L., Simalatsar, A., Zhu, Q.: METROII: A design environment for cyber-physical systems. ACM Transactions on Embedded Computing Systems **12**(1s) (March 2013) 49:1–49:31
15. Lee, E., Sangiovanni-Vincentelli, A.: A framework for comparing models of computation. IEEE Trans. CAD of Integ. Circ. and Systems **17**(12) (1998) 1217–1229
16. Dey, S., Sarkar, D., Basu, A.: A tag machine based performance evaluation method for job-shop schedules. IEEE Trans. CAD of Integ. Circ. and Systems **29**(7) (2010) 1028–1041
17. Dey, S., Sarkar, D., Basu, A.: A Kleene algebra of tagged system actors. Embedded Systems Letters, IEEE **3**(1) (2011) 28 –31
18. Fahrenberg, U., Legay, A., Wasowski, A.: Make a difference! (semantically). In: Proceedings of the 14th International Conference on Model Driven Engineering Languages and Systems. Volume 6981 of LNCS., Wellington, New Zealand, Springer (October 16-21 2011) 490–500

# 5 Appendix

**Proof of Theorem 1.**

For every run $r : s_0 \xrightarrow{\mu_1} s_1 \ldots \xrightarrow{\mu_n} s_n$ in the composition $M = M_1 \ {}_{\rho_1}\|_{\rho_2} \ M_2$, there exists a run $r_i : s_{0i} \xrightarrow{\mu_{1i}} s_{1i} \ldots \xrightarrow{\mu_{ni}} \mu_{ni}$ in $M_i$ such that $\mu_k = \mu_{1k} \ {}_{\rho_1}\sqcup_{\rho_2} \ \mu_{2k}$ for $1 \leq k \leq n$. Because $M_i \preceq M_i'$ and $M_i, M_i'$ are defined on the same variable set $V_i$, there must exist another run $r_i' : s_{0i}' \xrightarrow{\mu_{1i}} s_{1i}' \ldots \xrightarrow{\mu_{ni}} \mu_{ni}'$ in $M_i'$ matching $r_i$ on all the labels and accepting states. Composing runs $r_1'$ and $r_2'$ results in a run $r' : s_0' \xrightarrow{\mu_1} s_1 \ldots \xrightarrow{\mu_n} s_n$ for which $r$ is a refinement. □

**Proof of Theorem 2.**

We first construct a refinement relation $R$ and then show that the quotient is the most general TM defined on the $\mathcal{T}_{12}$ and $V$ satisfying the refinement.

- Initially, $R$ contains the state $((s_{02}, (s_{01}, s_{02})), s_{01})$. If there is a transition from any state $(s_{k2}, (s_{k1}, s_{k2}))$ in the left TM of the refinement, i.e. the following holds: $\exists (s_{k2}, \mu_2, s_{k2}') \in E_2, \exists ((s_{k1}, s_{k2}), \mu, s) \in E : \mu_2 \ {}_{\mathsf{id}_2}\!\bowtie_{\mathsf{proj}_2} \mu$, then $s = (s_{k1}', s_{k2}')$ for some $s_{k1}' \in S_1$. Indeed, the unifiability $\mu_2 \ {}_{\mathsf{id}_2}\!\bowtie_{\mathsf{proj}_2} \mu$ implies $\mu = \mu_1 \ {}_{\rho_1}\sqcup_{\rho_2} \mu_2$ for some $\mu_1 \in L(V_1, \mathcal{T}_1)$ s.t. $\mu_1 \ {}_{\rho_1}\!\bowtie_{\rho_2} \mu_2$. By determinism, $\mu_1$ is uniquely defined. This fact together with the existence of two unifiable transitions in $M_2$ and the quotient $M$ implies there must exist a transition $s_{k1} \xrightarrow{\mu_1} s_{k1}'$ and so $s = (s_{k1}', s_{k2}')$. Since it is easy to see that $(\mu_2 \ {}_{\mathsf{id}_2}\sqcup_{\mathsf{proj}_2} \mu) \ {}_{\mathsf{proj}_1'}\!\bowtie_{\mathsf{id}_1'} \mu_1$, the second refinement condition follows immediately and so $((s_{k2}', (s_{k1}', s_{k2}')), s_{k1}') \in R$.
  In addition, if $(s_{k2}', (s_{k1}', s_{k2}'))$ is an accepting state, then $s_{k2}' \in F_2$ and $(s_{k1}', s_{k2}') \in F$ from which we can infer that $s_{k1}' \in F_1$ by construction of $F$.
- Assuming there exists some runs $r'$ in $M'$ where the last transition $s_n' \xrightarrow{\mu_n}$ cannot be matched by $M$. There are two cases that can happen, $r'$ can unify fully with some run $r_2$ in $M_2$ or partially with every such run.
  In the first case, the composition of $r'$ and $r_2$ then refines some run $r_1$ in $M_1$. The existence of $r_1$ and $r_2$ together implies the existence of a run $r$ which can fully match $r'$ and contradicts the assumption. Similarly, in the second case, assume that $r'$ is unifiable from the initial state $s_0'$ up to the $k^{th}$ state $s_k'$. Then there exists some run $r$ in $M$ that can match the first $k$ transitions of $r'$ and also the rest transitions of $r'$ since the $(k+1)^{th}$ state of $r$ is its universal state. This also contradicts the assumption. Hence the assumption is wrong and such $r'$ can always be matched by $M$.
  We next assume that $M'$ can reach an accepting state $s_n'$ in $r'$. As before, it can unify fully with some run $r_2$ in $M_2$ or partially with every such run.
  In the first case, the last state of $r$ is $(s_{n1}, s_{n2})$ where $s_{ni}$ is the last state of run $r_i$. If $s_{n2} \in F_2$ then $s_{n1} \in F_1$ (since the composition of $r'$ and $r_2$ refines $r_1$ by assumption) and so $(s_{n1}, s_{n2}) \in F$. Else, i.e. $s_{n2} \notin F_2$, by construction $(s_{n1}, s_{n2}) \in F$.
  In the second case, the last state of run $r$ is $\mathfrak{u}$ which is also an accepting state.
  Therefore $M' \preceq M$. □

**Proof of Theorem 3.**

Let $M_{\bar{\mathcal{G}}} = M_{\mathcal{G}}/M_{\mathcal{A}}$ and $M_q = M_{\bar{\mathcal{G}}}/M_{\mathcal{A}}$, then the refinement $M_q \preceq M_{\bar{\mathcal{G}}}$ must hold.

Indeed, assume that there exists some run $r_q$ in $M_q$ and $r_{\bar{\mathcal{G}}}$ in $M_{\bar{\mathcal{G}}}$ where the last transition of $r_q$ cannot be simulated by $r_{\bar{\mathcal{G}}}$:

$$r_q : s_{0q} \xrightarrow{\mu_0} \dots s_{nq} \xrightarrow{\mu_n}$$
$$r_{\bar{\mathcal{G}}} : s_{0\bar{\mathcal{G}}} \xrightarrow{\mu_0} \dots s_{n\bar{\mathcal{G}}} \not\xrightarrow{\mu_n}$$

By the quotient construction of $M_{\bar{\mathcal{G}}}$, $r_{\bar{\mathcal{G}}}$ implies the existence of runs $r_{\mathcal{A}}$ in $M_{\mathcal{A}}$ and $r_{\mathcal{G}}$ in $M_{\mathcal{G}}$ such that:

$$r_{\mathcal{A}} : s_{0\mathcal{A}} \xrightarrow{\mu_0} \dots s_{n\mathcal{A}} \xrightarrow{\mu_n}$$
$$r_{\mathcal{G}} : s_{0\mathcal{G}} \xrightarrow{\mu_0} \dots s_{n\mathcal{G}} \not\xrightarrow{\mu_n}$$

By the quotient construction of $M_q$, $r_q$ and $r_{\mathcal{A}}$ imply the existence of $r'_{\bar{\mathcal{G}}} : s'_{0\bar{\mathcal{G}}} \xrightarrow{\mu_0} \dots s'_{n\bar{\mathcal{G}}} \xrightarrow{\mu_n}$ in $M_{\bar{\mathcal{G}}}$. By determinism, $s_{k\bar{\mathcal{G}}} \equiv s'_{k\bar{\mathcal{G}}}$ for $0 \leq k \leq n$ and so $s_{n\bar{\mathcal{G}}} \xrightarrow{\mu_n}$, contradicting the assumption. So, every run $r_q \in M_q$ can always be matched by some run $r_{\bar{\mathcal{G}}} \in M_{\bar{\mathcal{G}}}$. In addition, if $s_{nq} \in F_q$, then one of the following cases can happen:

1. $s_{nq} \equiv \mathfrak{u}_q$ : this implies $s_{n\bar{\mathcal{G}}} \equiv \mathfrak{u}_{\bar{\mathcal{G}}}$ since all possible transitions of the former must be matched by the latter,
2. $s_{nq} \in F_{\bar{\mathcal{G}}} \times F_{\mathcal{A}}$ : then $s_{n\bar{\mathcal{G}}} \in F_{\bar{\mathcal{G}}}$ since $r_q$ is matched by $r_{\bar{\mathcal{G}}}$,
3. $s_{nq} \in S_{\bar{\mathcal{G}}} \times (S_{\mathcal{A}} \setminus F_{\mathcal{A}})$ : then $s_{n\bar{\mathcal{G}}} \in F_{\bar{\mathcal{G}}}$ since $s_{n\bar{\mathcal{G}}} = (s_{n\mathcal{G}}, s_{n\mathcal{A}}) \in S_{\mathcal{G}} \times (S_{\mathcal{A}} \setminus F_{\mathcal{A}})$.

We next show that $\bar{\mathcal{C}} = (M_{\mathcal{A}}, M_{\bar{\mathcal{G}}})$ is in a normalized form by showing that $M_{\mathcal{I}} \in [\![\bar{\mathcal{C}}]\!]_{\mathsf{p}} \Leftrightarrow M_{\mathcal{I}} \preceq M_{\bar{\mathcal{G}}}$.

- $\Rightarrow$: Because $M_{\mathcal{I}}$ is an implementation of $\bar{\mathcal{C}}$, it implies that $\forall M_{\mathcal{E}} \in [\![\bar{\mathcal{C}}]\!]_{\mathsf{e}} : (M_{\mathcal{I}} \parallel M_{\mathcal{E}}) \preceq (M_{\bar{\mathcal{G}}} \parallel M_{\mathcal{E}})$. Since they are defined on the same tag structure and variable set, we can infer the refinement $(M_{\bar{\mathcal{G}}} \parallel M_{\mathcal{E}}) \preceq M_{\bar{\mathcal{G}}}$. Thus, $\forall M_{\mathcal{E}} \in [\![\bar{\mathcal{C}}]\!]_{\mathsf{e}} : (M_{\mathcal{I}} \parallel M_{\mathcal{E}}) \preceq M_{\bar{\mathcal{G}}}$. By the quotient definition, we can then infer $\forall M_{\mathcal{E}} \in [\![\bar{\mathcal{C}}]\!]_{\mathsf{e}} : M_{\mathcal{I}} \preceq (M_{\bar{\mathcal{G}}}/M_{\mathcal{E}})$ from which it follows that:

$$M_{\mathcal{I}} \preceq \parallel_{\forall M_{\mathcal{E}} \in [\![\bar{\mathcal{C}}]\!]_{\mathsf{e}}} (M_{\bar{\mathcal{G}}}/M_{\mathcal{E}}) \tag{2}$$

  Since the right side of refinement 2 refines $M_{\bar{\mathcal{G}}}/M_{\mathcal{A}}$ which refines $M_{\bar{\mathcal{G}}}$, the refinement $M_{\mathcal{I}} \preceq M_{\bar{\mathcal{G}}}$ then follows.
- $\Leftarrow$: Because $M_{\mathcal{I}} \preceq M_{\bar{\mathcal{G}}}$, we can infer $\forall M_{\mathcal{E}} \in [\![\bar{\mathcal{C}}]\!]_{\mathsf{e}} : (M_{\mathcal{I}} \parallel M_{\mathcal{E}}) \preceq (M_{\bar{\mathcal{G}}} \parallel M_{\mathcal{E}})$. Thus, $M_{\mathcal{I}} \in [\![\bar{\mathcal{C}}]\!]_{\mathsf{p}}$.

We finally show that $\mathcal{C}$ and $\bar{\mathcal{C}}$ have the same set of environments as well as implementations. The former holds since they have the same assumption. The latter holds because of two facts. First, $(M_{\mathcal{G}} \parallel M_{\mathcal{E}}) \preceq (M_{\bar{\mathcal{G}}} \parallel M_{\mathcal{E}})$ as $M_{\mathcal{G}} \preceq (M_{\mathcal{G}}/M_{\mathcal{A}})$. Second, $(M_{\bar{\mathcal{G}}} \parallel M_{\mathcal{E}}) \preceq (M_{\mathcal{G}} \parallel M_{\mathcal{E}})$ since any sequence of labels $\omega = \mu_0 \dots \mu_n$ of $M_{\mathcal{E}}$ also exists in $M_{\mathcal{A}}$ and if it exists in $M_{\bar{\mathcal{G}}}$ as well, it does in $M_{\mathcal{G}}$, too, by the quotient construction of $M_{\bar{\mathcal{G}}}$. □

**Proof of Theorem 4.**

- $\Rightarrow$: Because of $\mathcal{C}_1 \preceq_{\rho_2} \mathcal{C}_2$ and $M_{\mathcal{G}_1} \in [\![\mathcal{C}_1]\!]_\mathsf{p}$, there must exist some implementation $M_{\mathcal{I}_2} \in [\![\mathcal{C}_2]\!]_\mathsf{p}$ such that $M_{\mathcal{G}_1} \preceq_{\rho_2} M_{\mathcal{I}_2}$ (by the first condition of Def. 10). Since $M_{\mathcal{I}_2} \preceq M_{\mathcal{G}_2}$, we can infer that $M_{\mathcal{G}_1} \preceq_{\rho_2} M_{\mathcal{G}_2}$. Using a similar line of reasoning, we can also infer that $M_{\mathcal{A}_2} \preceq_{\rho_1} M_{\mathcal{A}_1}$.
- $\Leftarrow$: Since $\mathcal{C}_1$ is a normalized contract, for all $M_{\mathcal{I}_1} \in [\![\mathcal{C}_1]\!]_\mathsf{p}$, it holds that $M_{\mathcal{I}_1} \preceq M_{\mathcal{G}_1}$. Together with the fact of $M_{\mathcal{G}_1} \preceq_{\rho_2} M_{\mathcal{G}_2}$, it implies $M_{\mathcal{I}_1} \preceq_{\rho_2} M_{\mathcal{G}_2}$. Using a similar line of reasoning, we can also deduce that $M_{\mathcal{E}_2} \preceq_{\rho_1} M_{\mathcal{A}_1}$ for any environment $M_{\mathcal{E}_2}$ of contract $\mathcal{C}_2$. □

**Proof of Theorem 5.**

Let $\mathcal{C} = (M_\mathcal{A}, M_\mathcal{G})$, where $M_\mathcal{A} = (M_{\mathcal{A}_1} {}_{\rho_1}/_{\rho_2} M_{\mathcal{G}_2}) \curlywedge (M_{\mathcal{A}_2} {}_{\rho_2}/_{\rho_1} M_{\mathcal{G}_1})_\mathsf{swap}$ and $M_\mathcal{G} = M_{\mathcal{G}_1} \|_{\rho_2} M_{\mathcal{G}_2})$.

1. Contract $\mathcal{C}$ dominates $(\mathcal{C}_1, \mathcal{C}_2)$ w.r.t. $\rho_1$ and $\rho_2$ for:
   (a) $\mathcal{C}$ is defined over $\mathcal{T}_{12} = \mathcal{T}_1 \times_{\rho_2} \mathcal{T}_2$ and $V = V_1 \cup V_2$, by Def. 12.
   (b) $M_{\mathcal{I}_i} \in [\![\mathcal{C}_i]\!]_\mathsf{p} \Rightarrow M_{\mathcal{I}_i} \preceq M_{\mathcal{G}_i}$ (by Theorem 3). Thus, $(M_{\mathcal{I}_1} \|_{\rho_2} M_{\mathcal{I}_2}) \preceq (M_{\mathcal{G}_1} \|_{\rho_2} M_{\mathcal{G}_2})$, or equivalently $(M_{\mathcal{I}_1} \|_{\rho_2} M_{\mathcal{I}_2}) \in [\![\mathcal{C}]\!]_\mathsf{p}$.
   (c) $M_\mathcal{E} \in [\![\mathcal{C}]\!]_\mathsf{e}$ means $M_\mathcal{E} \preceq M_\mathcal{A}$ which implies:

$$M_\mathcal{E} \preceq (M_{\mathcal{A}_1} {}_{\rho_1}/_{\rho_2} M_{\mathcal{G}_2}) \tag{3}$$
$$M_\mathcal{E} \preceq (M_{\mathcal{A}_2} {}_{\rho_2}/_{\rho_1} M_{\mathcal{G}_1})_\mathsf{swap} \tag{4}$$

By the quotient construction, we know that:

$$(M_{\mathcal{G}_2} \|_{\mathsf{proj}_2} (M_{\mathcal{A}_1} {}_{\rho_1}/_{\rho_2} M_{\mathcal{G}_2})) \preceq_{\mathsf{id}_1} M_{\mathcal{A}_1}$$
$$(M_{\mathcal{G}_1} \|_{\overline{\mathsf{proj}}_1} (M_{\mathcal{A}_2} {}_{\rho_2}/_{\rho_1} M_{\mathcal{G}_1})) \preceq_{\mathsf{id}_2} M_{\mathcal{A}_2}$$

where $\overline{\mathsf{proj}}_1(\tau_{21}) = \mathsf{proj}_1 \circ \mathsf{swap}(\tau_{21})$ for $\tau_{21} \in \mathcal{T}_{21}$ and $\overline{\mathsf{proj}}_2'((\tau_1, \tau_{21})) = \mathsf{proj}_2 \circ \mathsf{swap}(\tau_{21})$ for $(\tau_1, \tau_{21}) \in \mathcal{T}_1 \times_{\mathsf{id}_1 \overline{\mathsf{proj}}_1} (\mathcal{T}_2 \times_{\rho_1} \mathcal{T}_1)$. We can then rewrite the above refinements:

$$(M_{\mathcal{G}_2} \|_{\mathsf{proj}_2} (M_{\mathcal{A}_1} {}_{\rho_1}/_{\rho_2} M_{\mathcal{G}_2})) \preceq_{\mathsf{id}_1'} M_{\mathcal{A}_1} \tag{5}$$
$$(M_{\mathcal{G}_1} \|_{\mathsf{proj}_1} (M_{\mathcal{A}_2} {}_{\rho_2}/_{\rho_1} M_{\mathcal{G}_1})_\mathsf{swap}) \preceq_{\mathsf{id}_2'} M_{\mathcal{A}_2} \tag{6}$$

Because of refinement 3, 4 and $M_{\mathcal{I}_i} \preceq M_{\mathcal{G}_i}$:

$$M_{\mathcal{I}_2} \|_{\mathsf{proj}_2} M_\mathcal{E} \preceq M_{\mathcal{G}_2} \|_{\mathsf{proj}_2} (M_{\mathcal{A}_1} {}_{\rho_1}/_{\rho_2} M_{\mathcal{G}_2})$$
$$M_{\mathcal{I}_1} \|_{\mathsf{proj}_1} M_\mathcal{E} \preceq M_{\mathcal{G}_1} \|_{\mathsf{proj}_1} (M_{\mathcal{A}_2} {}_{\rho_2}/_{\rho_1} M_{\mathcal{G}_1})_\mathsf{swap}$$

which due to refinement 5, 6 becomes:

$$(M_{\mathcal{I}_2} \|_{\mathsf{proj}_2} M_\mathcal{E}) \preceq_{\mathsf{id}_1} M_{\mathcal{A}_1}$$
$$(M_{\mathcal{I}_1} \|_{\mathsf{proj}_1} M_\mathcal{E}) \preceq_{\mathsf{id}_2} M_{\mathcal{A}_2}$$

2. Since $\mathcal{C}$ and $\mathcal{C}'$ are defined on the same tag structure and variable set, to prove $\mathcal{C} \preceq \mathcal{C}'$, we first show that $M_{\mathcal{A}'} \preceq M_{\mathcal{A}}$. Since $\mathcal{C}'$ dominates $(\mathcal{C}_1, \mathcal{C}_2)$ w.r.t. the same morphisms, the last condition of Definition 11 is satisfied for any environment $M_{\mathcal{E}} \in [\![\mathcal{C}']\!]_{\mathsf{e}}$:

$$(\forall M_{\mathcal{I}_1} \in [\![\mathcal{C}_1]\!]_{\mathsf{p}} : (M_{\mathcal{I}_1}\ {}_{\mathsf{id}_1}\|_{\mathsf{proj}_1} M_{\mathcal{E}})_{\mathsf{proj}'_2} \preceq_{\mathsf{id}_2} M_{\mathcal{A}_2}) \wedge$$
$$(\forall M_{\mathcal{I}_2} \in [\![\mathcal{C}_2]\!]_{\mathsf{p}} : (M_{\mathcal{I}_2}\ {}_{\mathsf{id}_2}\|_{\mathsf{proj}_2} M_{\mathcal{E}})_{\mathsf{proj}'_1} \preceq_{\mathsf{id}_1} M_{\mathcal{A}_1})$$

Since $M_{\mathcal{A}'} \in [\![\mathcal{C}']\!]_{\mathsf{e}}$ and $M_{\mathcal{G}_i} \in [\![\mathcal{C}_i]\!]_{\mathsf{p}}$, we can infer:

$$((M_{\mathcal{G}_1}\ {}_{\mathsf{id}_1}\|_{\mathsf{proj}_1} M_{\mathcal{A}'})_{\mathsf{proj}'_2} \preceq_{\mathsf{id}_2} M_{\mathcal{A}_2}) \wedge ((M_{\mathcal{G}_2}\ {}_{\mathsf{id}_2}\|_{\mathsf{proj}_2} M_{\mathcal{A}'})_{\mathsf{proj}'_1} \preceq_{\mathsf{id}_1} M_{\mathcal{A}_1})$$

which by Theorem 2 implies:

$$(M_{\mathcal{A}'} \preceq (M_{\mathcal{A}_2}\ {}_{\rho_2}/_{\rho_1} M_{\mathcal{G}_1})_{\mathsf{swap}}) \wedge (M_{\mathcal{A}'} \preceq (M_{\mathcal{A}_1}\ {}_{\rho_1}/_{\rho_2} M_{\mathcal{G}_2}))$$

or equivalently $M_{\mathcal{A}'} \preceq M_{\mathcal{A}}$. To prove the second condition of Definition 10, it is sufficient to show that every implementation of $\mathcal{C}$ is also an implementation of $\mathcal{C}'$. By definition:

$$M_{\mathcal{I}} \in [\![\mathcal{C}]\!]_{\mathsf{p}} \Leftrightarrow \forall M_{\mathcal{E}} \in [\![\mathcal{C}]\!]_{\mathsf{e}} : M_{\mathcal{I}} \parallel M_{\mathcal{E}} \preceq M_{\mathcal{G}} \parallel M_{\mathcal{E}}.$$

By $M_{\mathcal{A}'} \preceq M_{\mathcal{A}}$, we can infer $[\![\mathcal{C}']\!]_{\mathsf{e}} \subseteq [\![\mathcal{C}]\!]_{\mathsf{e}}$. By the second condition of Definition 11, we can infer $(M_{\mathcal{G}_1}\ {}_{\rho_1}\|_{\rho_2}\ M_{\mathcal{G}_2}) \in [\![\mathcal{C}']\!]_{\mathsf{p}}$ or $M_{\mathcal{G}} \in [\![\mathcal{C}']\!]_{\mathsf{p}}$. Thus:

$$\forall M_{\mathcal{E}} \in [\![\mathcal{C}']\!]_{\mathsf{e}} : M_{\mathcal{I}} \parallel M_{\mathcal{E}} \preceq M_{\mathcal{G}} \parallel M_{\mathcal{E}} \preceq M_{\mathcal{G}'} \parallel M_{\mathcal{E}}.$$

Consequently, $M_{\mathcal{I}} \in [\![\mathcal{C}']\!]_{\mathsf{p}}$. □

**Proof of Theorem 6.**

The first property holds because the satisfaction of two conditions in Def. 11 can be deduced from the fact that $[\![\mathcal{C}'_1]\!]_{\mathsf{p}} \subseteq [\![\mathcal{C}_1]\!]_{\mathsf{p}}$, $[\![\mathcal{C}'_2]\!]_{\mathsf{p}} \subseteq [\![\mathcal{C}_2]\!]_{\mathsf{p}}$, $M_{\mathcal{A}_1} \preceq M_{\mathcal{A}'_1}$, $M_{\mathcal{A}_2} \preceq M_{\mathcal{A}'_2}$ and $\mathcal{C}$ dominates $(\mathcal{C}_1, \mathcal{C}_2)$ w.r.t. $\rho_1$ and $\rho_2$. The second property follows directly from the first property of this theorem and the second property of Theorem 5. □