A Tag Contract Framework for Modeling Heterogeneous Systems

Thi Thieu Hoa Le[a], Roberto Passerone[a], Uli Fahrenberg[b], Axel Legay[b]

[a]*DISI, University of Trento, Italy*
[b]*INRIA/IRISA, Rennes, France*

Abstract

Development of distributed systems can be supported effectively by a contract-based methodology as contracts can ensure interoperability of components and adherence to specifications. Such development can become very complex since distributed systems can consist of components which are heterogeneous in terms of computational and interactive model. Several frameworks, both operational and denotational, have been proposed to handle heterogeneity using a variety of approaches. However, the application of those frameworks 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. Besides, we also develop a method to synthesize or refine the component models so that their composition satisfies a given contract.

Keywords:
Contract, heterogeneity, tag, synthesis, component model

1. Introduction

A distributed system often consists of a collection of components which are developed and executed across different computing nodes in a distributed manner. Development of such systems can be supported effectively by design methodologies such as those based on *contracts* [1]. A contract-based methodology rests on a model of the system that allows the designer to specify constraints on the environment of each component, called the component *assumptions*, as well as what can be (or must be) guaranteed given the satisfaction of such constraints, called the component *guarantees*. This explicit separation between assumptions and guarantees is the key in supporting the development of distributed systems.

On top of this, when components are distributed across different companies in the supply chain, and they are part of a complex system, they are also likely to

be developed using different models and interaction mechanisms, thereby making the model of the whole distributed system *heterogeneous*. Heterogeneity may imply different models of time, from simple precedence relations to discrete and continuous real time, as well as different semantics of synchronization, such as synchronous and asynchronous, and rendez-vous vs. broadcast [2]. This aspect aggravates the integration activities, given that it is difficult to define a complete virtual system that can be executed and analyzed prior to the physical implementation. To deal with heterogeneity, several modeling frameworks have been proposed oriented towards the representation and simulation of heterogeneous systems, such as the Ptolemy framework [3], or towards the unification of their interaction paradigms, such as those based on *tagged events* [4]. 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 [5].

Due to the significant inherent complexity of heterogeneity, there have been only very few attempts at addressing heterogeneity in the context of contract-based models. For instance, the HRC model from the SPEEDS project[1] was designed to deal with different viewpoints (functional, time, safety, etc.) of a single component [6, 7]. However, the notion of heterogeneity in general is much broader than that between multiple viewpoints, and must take into account diverse interaction paradigms. In our previous work, we have laid the foundation for a modeling methodology which is contract-based and heterogeneous [8]. Our methodology enables different distributed components to be specified as different contracts and the whole system model to be built by composing its component contract models. In addition, the underlying modeling mechanism of *heterogeneous tag machines* [9], which supports our methodology, allows the component models to be specified in different timed models and interaction styles. In this paper, we further develop our framework to cover also aspects related to design and development, and explore the issue of how to correctly decompose the specification hierarchically into a refined implementation.

Typically, development of distributed systems can be done in a bottom-up manner, i.e., by *composing* simpler predefined components together. Alternatively, they can be developed in a top-down fashion, by successively *decomposing* or *refining* the system into simpler components which are then independently developed and implemented. This approach is well supported by a contract-based methodology: the decomposition is correct as long as the composition of the contracts associated with the sub-components satisfies or refines the contract of the intended system. When this condition is *not* satisfied, i.e., the composition of the contracts of the sub-components does not refine the system contract, designers must adjust the component contract specifications until the inconsistency with the system model is cleared. In this work, we discuss which steps must be taken to check consistency of a decomposition with the specification, and to adjust the component specification when this is not satisfied.

More formally, we deal with the problem of checking if a contract $\mathcal{C}$ is cor-

[1] www.speeds.eu.com

rectly decomposed (or refined) into a pair $\{\mathcal{C}_1, \mathcal{C}_2\}$ of heterogeneous contracts. When this is not the case, we discuss methods to update the contract set in order to make the composition refine $\mathcal{C}$ if necessary. To this end, we study *decomposing conditions* under which the contract decomposition can be verified, and thereby propose a *synthesis strategy* for fixing wrong decompositions. In particular, this paper extends our previous work with a synthesis capability for the tag contract framework to provide for extra flexibility in component design and reusing.

The rest of the paper is organized as follows. We first review in Sect. 2 work on contract and synthesis which is related to our approach. In Sect. 3, we recall notions of tags and heterogeneous tag machines (TMs) and their composition. In Sect. 4, we present our contract framework for modeling heterogeneous systems with a full set of operations and relations such as contract satisfaction, contract refinement, contract dominance and contract compatibility. We also recall that a shorter version of this section without formal proofs and the ability of checking contract compatibility appeared at the FOCLASA workshop [8]. Based on the decomposition into a pair of *two* contracts provided in Sect. 4.4, we further extend our framework to be capable of synthezing the component contracts and propose decomposing conditions for a pair of contracts. In particular, we suggest a strategy for synthesizing those contracts in order to make their composition satisfy a predefined contract in Sect. 5. Finally we conclude in Sect. 6.

2. Related Work

Contracts were first introduced in Meyer's design-by-contract method [1], based on ideas by Dijkstra [10], Lamport [11], and others, where systems are viewed as abstract boxes achieving their common goal by verifying specified contracts. Such technique guarantees that methods of a class provide some post-conditions at their termination, as long as the pre-conditions under which they operate are satisfied. De Alfaro and Henzinger subsequently introduced interface automata [12] for documenting components thereby enabling them to be reused effectively. This formalism establishes 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 central issues when introducing the formalism of interface automata are compatibility, composition and refinement. Separating assumptions from guarantees, which was somewhat implicit in interface automata, has then been made explicit in the contract framework of the SPEEDS HRC model [6, 13]. A separation between specifying assumptions on expected behaviors and guarantees to achieve them at run time has recently been applied to the handling of synchronization requirements to improve the component-based development of real-time high-integrity systems [14].

The relationship between specifications of component behaviors and contracts is further studied by Bauer et al. [15] 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 [6] and

modal contract [16] theories are also demonstrated to be instances of such a framework. This formalization enables verifying if a contract can be decomposed into two other contracts by checking if that contract can *dominate* the others. In this work, we take advantage of this formalization and dominating notion and adapt them to construct our tag contract framework and its synthesis functionality. In particular, we extend these notions to a heterogeneous context.

Assume-Guarantee Reasoning (AGR) has also been applied extensively in declarative compositional reasoning [17] to help prove properties by decomposing them into simpler and more manageable steps. The classical AGR uses assumptions as hypotheses to establish whether a generic property holds. Naturally, this technique can be used in contract models as well, with possibly non-trivial transformation and formalization. In case of unsuccessful termination, AGR can also provide a counterexample showing how the property can be violated. Such a counterexample can then be used to synthesize the model so as to satisfy a given property [18]. However, this synthesis strategy is only applicable for homogeneous systems with trace-based semantics. Viewing the same assume-guarantee synthesis problem as a game, Chatterjee et al. solve it by finding a winning strategy on the global system state graph, but the method does not guarantee the inclusion of all traces satisfying the specification [19]. The synthesized model was shown to be a subset of that synthesized by counterexample-based synthesis [18]. Unlike these concrete notions of homogeneous synthesis, ours is more generic since it can be applied to a heterogeneous context.

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). The idea behind these theories and frameworks is to be able to combine well-established specification formalisms to enable analysis and simulation across heterogeneous boundaries. This is usually accomplished by providing some sort of common mechanism in the form of an underlying rich semantic model or coordination protocol. In this paper, we are mostly concerned with these lower level aspects. One such approach is the pioneering framework of Ptolemy II [20], where models, called *domains*, are combined hierarchically: each level of the hierarchy is homogeneous, while different interaction mechanisms are specified at different levels in the hierarchy. In the underlying model, intended for simulation, each domain is composed of a scheduler (the *director*) which exposes the same abstract interface to a global scheduler which coordinates the execution. This approach, which has clear advantages for simulation, has two limitations in our context. First, it does not provide access to the components themselves but only to their schedulers, limiting our ability to establish relations to only the models of computation, and not to the heterogeneous contracts of the components. Secondly, the heterogeneous interaction occurs implicitly as a consequence of the coordination mechanism, and can not be controlled by the user. The metroII framework [21] relaxes this limitation, and allows designers to build direct model adapters. However, metroII treats components mostly as black boxes using a

wrapping mechanism 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. More recently, the metroII and the Ptolemy II frameworks have been combined into Metronomy, to specifically address function-architecture timing verification through co-simulation [22]. Contracts are used to specify the timing properties, while the function and architecture are synchronized by a mapping function, which resembles our morphisms. The mapping function, however, is only used to establish simulation constraints, and heterogeneous composition is not further explored.

Another body of work is instead oriented towards the formal representation, verification and analysis of these system. The BIP framework uses the notion of connector, on top of a state based model, to implement both synchronous and asynchronous interaction patterns [2]. The relationship between synchronous and asynchronous interaction is however intrinsic in the connectors, and time cannot be easily expressed natively. This significantly limits the flexibility that system designers have to *design* the interaction between heterogeneous models according to their different needs. Benveniste et al. [5] propose a heterogeneous denotational semantics inspired by the Lee and Sangiovanni-Vincentelli formalism of tag signal models [4], 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 (TMs) [23] have subsequently been 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 [9]. TMs have been shown to be sufficiently expressive to represent several useful models and interaction paradigms, such as synchronous and asynchronous systems, timed and time-triggered systems, as well as causalities and scheduling specifications (such as event structures) and earliest execution times [23]. TMs have also been applied to model a job-shop specification [24] 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 [25]. The approach is similar to that of Ptolemy II in that both use actors to represent basic components. Tags have also been used to represent generic connectors in timed data streams (TDS) [26]. While TDS use two separate sequences for values and time, the way behaviors are defined in our settings is similar in that we employ sequences of increasing tag values.

To summarize, we propose a contract framework which can handle heterogeneous models. To do so, we extend the Tag machines model to operate across different tag systems by employing tag morphisms. This provides us both a formal definition of composition, which overcomes the limitations of simulation-based heterogeneous frameworks, as well as an operational semantics. We in-

herit the contract theory proposed by Bauer et al. [15], which is extended to the heterogeneous context. The notion of dominance is used to formulate decomposition conditions and a synthesis strategy, following the assume-guarantee paradigm, which are generic and independent of the particular model in use.

We remark that heterogeneity adds a new dimension of complexity to contract-based reasoning. In a homogeneous model, such as the ones of [12, 15], for example, different components can be directly related, composed and synthesized. In a heterogeneous paradigm, on the other hand, all these operations have to go through an intermediary. Different components may use conflicting models of time, for example, one using an untimed causality-based model and the other a time-triggered model. To relate the two components, a common time structure together with morphisms to the two conflicting time models has to be used. The same holds for heterogeneous composition, and as we show in previous work [9], this means that some desirable properties, which hold for homogeneous models, are not available in the heterogeneous framework.

3. Tag Machine Formalism

We consider a component to be a set of behaviors expressed in terms of the 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 and synchronizations. 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. To distinguish the tag order of $\mathcal{T}$, we refer to it as $\leq_{\mathcal{T}}$ when necessary. 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.

3.1. 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 (τ, d) of a tag $\tau \in T$ and a value $d \in D$. The simplest way of characterizing a behavior is as a collection of events for each variable. 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 [5]. A behavior for a variable v is thus a function $\mathbb{N} \to (T \times D)$. A behavior σ for a component assigns a sequence of events to every variable in V, i.e.,

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

Each event of behavior σ is identified by a tuple (v, n, τ, d), capturing the n-th occurrence of variable v as a pair of a tag τ and a value d. In the following, we

denote with $\Sigma(V, \mathcal{T})$ the universe of all behaviors over a set of variables V and tag structure $\mathcal{T}$.

Combining behaviors σ_1 and σ_2 on the same tag structure, or *homogeneous* behaviors, amounts to computing their intersection provided that they are consistent, or *unifiable*, written $\sigma_1 \bowtie \sigma_2$, with each other on the shared variables, i.e.,

$$\sigma_1|_{V_1 \cap V_2} = \sigma_2|_{V_1 \cap V_2},$$

where $\sigma|_W$ denotes the restriction of behavior σ to the variables in set W. We may then 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 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 ([5]). Let $\mathcal{T}$ and $\mathcal{T}'$ be two tag structures. A *tag morphism* from $\mathcal{T}$ to $\mathcal{T}'$ is a total map $\rho{:}\mathcal{T} \to \mathcal{T}'$ 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 $\sigma \in V \to (\mathbb{N} \to (T \times D))$ into a T'-behavior $\sigma \circ \rho \in V \to (\mathbb{N} \to (T' \times D))$ by simply replacing all tags τ in σ with the image $\rho(\tau)$.

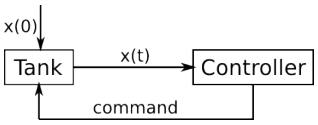
Unification of heterogeneous behaviors can be done on the common tag structure. Let $\rho_1{:}\mathcal{T}_1 \to \mathcal{T}$ and $\rho_2{:}\mathcal{T}_2 \to \mathcal{T}$ be two tag morphisms into a tag structure $\mathcal{T}$. Two behaviors σ_1 and σ_2 defined on $\mathcal{T}_1$ and $\mathcal{T}_2$ respectively 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 σ 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 (see Definition 7 below). To do so, we construct the heterogeneous composition over the fibered product [5] $\mathcal{T}_1 {}_{\rho_1}\!\times_{\rho_2} \mathcal{T}_2 = (T_1 {}_{\rho_1}\!\times_{\rho_2} T_2, \leq)$ of the original tag structures, extending the order component-wise:

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

where $T_1 \,{}_{\rho_1}\!\times_{\rho_2} T_2 = \{(\tau_1, \tau_2) \in T_1 \times T_2 : \rho_1(\tau_1) = \rho_2(\tau_2)\}$. Observe that $T_1 \,{}_{\rho_1}\!\times_{\rho_2} T_2$ does not contain all possible pairs of tags, but only those pairs which map onto the same tag on T. For this reason, and consistently with the original work by Benveniste et al. [5], we will consider only those tag systems which can *synchronize*, that is for which every tag of T_1 has a corresponding tag in T_2 which is mapped onto the same element in T, and vice-versa. Note



(a) System diagram

σ_1 :	m :		0.5;**p**		1.5;**l**	...
	x :	0;0	0.5;0	1;0.5	1.5;1	...
σ_2 :	m :		1;**p**		3;**l**	...
	x :	0;0	1;0	2;0.5	3;1	...

(b) A tank (σ_1) and controller (σ_2) behavior

Figure 1: Water controlling system

also that the fibered product $T \,{}_{\mathsf{id}}\!\times_{\mathsf{id}}\, T$ of a tag structure $\mathcal{T}$ with itself using the identity morphism is isomorphic to $\mathcal{T}$. In general, we will not distinguish between isomorphic tag structures. We will use this fact implicitly in the rest of the paper when applying results derived for heterogeneous tag structures to operations that involve homogeneous structures (such as the assumptions and the guarantees of a contract).

Example 2. We consider a simplified version of the water controlling system proposed by Benvenuti et al. [13]. It consists of two components: a water tank and a water level controller, connected in a closed-loop fashion, see Figure 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 Δ_t denotes the time elapsed since t_0 at which the tank reaches the maximum/minimum water level $\mathbf{H}$, i.e., $\Delta_t = t - t_0$. It will be convenient later on to have a *least* value for the tags (see Definition 3), which we denote as $\epsilon_1 = \epsilon_2 = -\infty$. Then, the tank behaviors can be naturally defined on tag structure $\mathcal{T}_1 = (\mathbb{R}_+ \cup \{\epsilon_1\}, \leq)$ and the controller behaviors on $\mathcal{T}_2 = (\mathbb{N} \cup \{\epsilon_2\}, \leq)$ representing continuous and discrete time respectively. In addition, both components contain 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 σ_1 and the controller behavior σ_2 shown in Figure 1(b), where $\sigma(v, n)$ is described when the parameter setting is $\mathbf{f_i} = 2$, $\mathbf{f_o} = 1$, and $\mathbf{h} = 1$. These are different behaviors whose composition is only possible under the presence of morphisms such as $\rho_i : \mathcal{T}_i \to \mathcal{T}_1$ given by

$$\rho_1(\tau_1) = \tau_1, \rho_2(\tau_2) = 0.5 * \tau_2.$$

Our interest in this system is to construct the contracts of these components and to prove the compatibility between the contracts of these components, all of which will be provided later in this paper. Intuitively, the tank contract guarantees a linear evolution of the water level $x(t)$ upon the reception of in-time commands. Meanwhile, the controller contract only assumes the initial emptiness of the tank and guarantees to send proper commands upon detecting its emptiness or fullness.

3.2. Operational Tag Machines

TMs were first introduced to represent sets of homogeneous behaviors [23] and have been recently extended to encompass the heterogeneous context [9]. 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 3 ([23]). An *algebraic tag structure* is a tag structure $\mathcal{T} = (T, \leq, \cdot)$ where $\cdot$ is a binary operator on T called *concatenation*, such that:

1. $(T, \cdot)$ is a monoid with identity element $\hat{\imath}$,
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. $\exists \epsilon \in T : \forall \tau \in T : \epsilon \leq \tau \wedge \epsilon \cdot \tau = \tau \cdot \epsilon = \epsilon$.

Tags can be organized as *tag vectors* $\vec{\tau} = (\tau^{v_1}, \ldots, \tau^{v_n})$, where n is the number of variables in V. During transitions, tag vectors evolve according to a matrix $\mu : V \times V \to T$ called a *tag piece* [23]. With some abuse of notation, we can lift the concatenation operator to tag vectors and tag pieces, and define the new tag vector as $\vec{\tau}_\mu \stackrel{\text{def}}{=} \vec{\tau} \cdot \mu$ where

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

and the maximum is taken with respect to the tag ordering. In practice, one concatenates each element of the tag vector with each tag on a column of μ, and then takes the largest value; thus the new value of any tag may depend on the tag increments on the events of the other variables. The special value ϵ can be used as a neutral element in the matrix. As the order is partial, the maximum may not exist, in which case the operation is not defined.

Intuitively, a tag piece μ represents increments in all variable tags over a transition and provides a way to operationally renew them. To represent also changes in variable values, μ can be labeled with a partial assignment $\nu : V \to D$, which assigns new values to the variables. We say that a *labeled* tag piece μ has an event for all variables for which ν is defined. In the following, we denote by $\mathfrak{dom}(\nu)$ the domain of ν and by $L(V, \mathcal{T})$ the universe of all labeled tag pieces, or simply labels, over a variable set V and tag structure $\mathcal{T}$. By abuse of notation, we assume that every tag piece μ is always labeled and has an associated assignment ν.

Example 4. The algebraic tag structure $(\mathbb{N} \cup \{-\infty\}, \leq, +)$, where $+$ is the concatenation operator, can be used to capture logical time by structuring tag pieces μ to represent an integer increment of 1:

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

With these definitions, every time a new value must be assigned to a variable, i.e., when $\nu(v)$ is defined, the tag is computed as 1 plus the largest tag of any event on any variable. Otherwise, if there is no event on a variable, the tag is unchanged and no new event is generated. 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 the same since the least element $-\infty = \epsilon$ is used to cancel the contribution of an entry in the tag vector. Similarly, with these definitions, $\left[\begin{smallmatrix}4 & 3\end{smallmatrix}\right] \cdot \left[\begin{smallmatrix}0 & 1 \\ -\infty & 1\end{smallmatrix}\right] = \left[\begin{smallmatrix}4 & 5\end{smallmatrix}\right]$, so that the event in the second variable will be past the latest event on all variables (in this case, in the first). Other definitions for the tag pieces would have different effects.

A tag machine M is a finite automaton where transitions are marked by labels.

Definition 5 ([9]). 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 TM run r is a sequence of states and transitions

$$r : s_0 \xrightarrow{\mu_1} s_1 \xrightarrow{\mu_2} s_2 \ldots s_{m-1} \xrightarrow{\mu_m} s_m$$

such that $(s_{i-1}, \mu_i, s_i) \in E$, for all $1 \leq i \leq m$. A run r of M is *valid* if the concatenation is always defined along the run, and if $s_m \in F$.

Intuitively, a TM is used to construct a behavior (as defined in Section 3.1) by following its labeled transitions over a run, and concatenating the tag pieces sequentially to the initial tag vector $\vec{\tau} = (\hat{\imath}, \ldots, \hat{\imath})$. A new event is added to the behavior whenever a new value is assigned by the label function ν_i. In order to formalize the language of a tag machine, we must keep track of both the tags and the number of events that have occurred for each variable. Thus, for every state s_i along run r, we define a tag vector $\vec{\tau}_i$ computed by accumulating the tag pieces:

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

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

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

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

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

The language $L(M)$ of M is given by the label sequences of all valid runs and the behavioral semantics $\Sigma(M)$ of M is the set of behaviors obtained from its language [9].

3.3. Tag Machine Composition

As TMs are used to represent sets of behaviors, combining TMs amounts to considering only behaviors which are consistent with every TM. 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 ρ_1 and ρ_2 mapping the tag structures $\mathcal{T}_1$ and $\mathcal{T}_2$ 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 6 ([9]). A tag morphism $\rho : \mathcal{T} \to \mathcal{T}'$ is *algebraic* if $\rho(\hat{\imath}_{\mathcal{T}}) = \hat{\imath}_{\mathcal{T}'}$ and $\rho(\epsilon_{\mathcal{T}}) = \epsilon_{\mathcal{T}'}$ and $\rho(\tau_1 \cdot_{\mathcal{T}} \tau_2) = \rho(\tau_1) \cdot_{\mathcal{T}'} \rho(\tau_2)$ for all $\tau_1, \tau_2 \in \mathcal{T}$.

The newly-composed TM will be defined on a unified label set and a unified tag structure

$$\mathcal{T}_1 \,{}_{\rho_1}\!\times_{\rho_2} \mathcal{T}_2 = (T_1 \,{}_{\rho_1}\!\times_{\rho_2} T_2, \leq, \cdot)$$

where $\leq \stackrel{\text{def}}{=} (\leq_{\mathcal{T}_1}, \leq_{\mathcal{T}_2})$ and $\cdot \stackrel{\text{def}}{=} (\cdot_{\mathcal{T}_1}, \cdot_{\mathcal{T}_2})$. Referring to the previous notation, two labels μ_1 and μ_2 are *unifiable* under morphisms ρ_1 and ρ_2, written $\mu_1 \,{}_{\rho_1}\!\bowtie_{\rho_2} \mu_2$, whenever

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

[2] We sometimes refer to $\sigma(r)$ as $\sigma(\omega)$ where $\omega = \mu_1\mu_2\ldots\mu_m$

for all pairs $(w, v) \in W \times W$ where $W = V_1 \cap V_2$. To compute the unification, we can take the inverse image, under the morphisms, of the unified labeled tag piece in $\mathcal{T}$. This results in a *set* of labeled tag pieces. More explicitly, when unifiable, the unification $\mu = \mu_1 \,{}_{\rho_1}\sqcup_{\rho_2}\, \mu_2$ is defined over $\mathcal{T}_1 \,{}_{\rho_1}\times_{\rho_2}\, \mathcal{T}_2$ and is any of the members of the unification 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 tags τ_1 and τ_2 exist because we assume the tag structures can be synchronized. 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 7 (Heterogeneous Composition [9]). The composition of M_1 and M_2 under algebraic tag morphisms ρ_1 and ρ_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:

- $V = V_1 \cup V_2$, $S = S_1 \times S_2$, $s_0 = (s_{01}, s_{02})$, $F = F_1 \times F_2$,
- $\mathcal{T}_1 \,{}_{\rho_1}\times_{\rho_2}\, \mathcal{T}_2 = (T_1 \,{}_{\rho_1}\times_{\rho_2}\, T_2, \leq, \cdot)$ where $\leq = (\leq_{\mathcal{T}_1}, \leq_{\mathcal{T}_2})$ and $\cdot = (\cdot_{\mathcal{T}_1}, \cdot_{\mathcal{T}_2})$,
- $E = \{((s_1, s_2), \mu_1 \,{}_{\rho_1}\sqcup_{\rho_2}\, \mu_2, (s_1', s_2')) : \mu_1 \,{}_{\rho_1}\bowtie_{\rho_2}\, \mu_2 \wedge (s_i, \mu_i, s_i') \in E_i, i = 1, 2\}$ where $\mu_1 \,{}_{\rho_1}\sqcup_{\rho_2}\, \mu_2$ extends to all the members of the unification set.

Heterogeneous composition is not necessarily sound and complete with respect to the language of the machines, because of the richness of the representation. Certain restrictions can however be imposed on the model to regain these desirable properties. The interested reader can find a discussion of these issues in our earlier work [9]. As 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.

4. A Tag Contract Framework

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 [9] and refinement,

quotient and conjunction to relate their sets of behaviors (Sect. 4.1). 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 (Sect. 4.2).

4.1. Tag Machine Operators

In this section, we will discuss the main relations and operators that involve tag machines. We have already discussed composition. Below, we will introduce the notion of *refinement*, and the operators of *quotient* and *conjunction*.

4.1.1. Tag Machine Refinement

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. From 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 $\rho_i : \mathcal{T}_i \to \mathcal{T}$ be algebraic tag morphisms, where $i \in \{1, 2\}$. TM refinement is defined in terms of a simulation relation as follows.

Definition 8. 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 $(s_{01}, s_{02}) \in R$ and for all $(s_1, s_2) \in R$ and $(s_1, \mu_1, s_1') \in E_1$ there exists $(s_2, \mu_2, s_2') \in E_2$ such that

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

Because we are restricting our attention to deterministic machines, refinement is equivalent to behavior containment, so we will use one or the other formulation indifferently to simplify our reasoning. When languages are expressed using different tag structures, the morphisms are used to equalize the tags of the labels.

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

Theorem 1. *Let M_i, M_i' be TMs defined on $\mathcal{T}_i$ and V_i:*

$$(M_1 \preceq M_1') \wedge (M_2 \preceq M_2') \Rightarrow (M_1 \ {}_{\rho_1}\|_{\rho_2} M_2) \preceq (M_1' \ {}_{\rho_1}\|_{\rho_2} M_2').$$

Proof. For every run $r : s_0 \xrightarrow{\mu_1} s_1 \ldots \xrightarrow{\mu_n} s_n$ in the composition $M_1 \ {}_{\rho_1}\|_{\rho_2} M_2$, 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. □

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

Example 9. We consider an example where

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

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 μ_i for machine M_i and μ_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 $M_1 \preceq_{\rho_1 \rho_2} M_2$ since $\mu_1 \bowtie_{\rho_1 \rho_2} \mu_2$ and $M_1' \preceq_{\rho_1 \rho_2} M_2'$ since $\mu_1' \bowtie_{\rho_1 \rho_2} \mu_2'$. However, $(M_1 \parallel M_1') \not\preceq_{\rho_1 \rho_2} (M_2 \parallel M_2')$ since the right composition is empty while the left is not.

On the other hand, if we add a second self-loop to machine M_2' with label $\mu_{22}' = [\tau_2]$ and $\nu_{22}(z) = \top$, then clearly $M_2 \preceq M_2'$ (since M_2 has a subset of the behaviors of M_2'), and by construction $M_1 \preceq M_1'$ (since they are identical). Carrying out the simple compositions, we see that indeed $(M_1 \parallel_{\rho_1 \rho_2} M_2) \preceq (M_1' \parallel_{\rho_1 \rho_2} M_2')$, as required by the theorem.

4.1.2. Tag Machine Quotient

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 $M = M_1 \parallel_{\rho_1 \rho_2} M_2$ computes the most general specification that retains all unifiable behaviors of two TMs (see Definition 7). The dual operator to TM composition is TM *quotient*, denoted $M = M_1 /_{\rho_1 \rho_2} M_2$, which computes the residual of a composition, i.e., the most general specification M that, when composed with M_2 satisfies M_1. This operator is specified as follows.

Definition 10. The quotient $M_1 /_{\rho_1 \rho_2} M_2$ is $M = (V, \mathcal{T}_{12}, S, s_0, F, E)$, where:

- $V = V_1 \cup V_2$, $\mathcal{T}_{12} \stackrel{\text{def}}{=} \mathcal{T}_1 \times_{\rho_1 \rho_2} \mathcal{T}_2$, and $s_0 = (s_{01}, s_{02})$,
- $S = (S_1 \times S_2) \cup \{\mathfrak{u}\}$, where $\mathfrak{u}$ is a new *universal* state,
- $F = ((S_1 \times S_2) \setminus ((S_1 \setminus F_1) \times F_2)) \cup \{\mathfrak{u}\} = (F_1 \times F_2) \cup (S_1 \times (S_2 \setminus F_2)) \cup \{\mathfrak{u}\}$,
- $E = \begin{array}{l} \{((s_1, s_2), \mu_1 \sqcup_{\rho_1 \rho_2} \mu_2, (s_1', s_2')) \mid \\ \quad (\mu_1 \bowtie_{\rho_1 \rho_2} \mu_2) \wedge ((s_1, \mu_1, s_1') \in E_1) \wedge ((s_2, \mu_2, s_2') \in E_2)\} \cup \\ \{((s_1, s_2), \mu_1 \sqcup_{\rho_1 \rho_2} \mu_2, \mathfrak{u}) \mid \\ \quad (\forall s_2' \in S_2 : (s_2, \mu_2, s_2') \notin E_2) \wedge (\exists \mu_1 \in L(V_1, \mathcal{T}_1) : \mu_1 \bowtie_{\rho_1 \rho_2} \mu_2)\} \cup \\ \{(\mathfrak{u}, \mu, \mathfrak{u}) | \mu \in L(V, \mathcal{T}_{12})\}. \end{array}$

We give an example of a quotient construction later in Figure 3(c). The idea behind the definition is exclude from the quotient only those behaviors which are allowed by M_2, but not by M_1. All other behaviors are permitted, since they are either allowed by M_1, or will be excluded by the composition with M_2, and therefore do not influence the refinement. The dual relation between composition and quotient is presented in the next theorem.

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

$$\forall i \in \{1,2\}, \forall \tau_i \in \mathcal{T}_i : \mathsf{id}_i(\tau_i) = \tau_i \tag{2}$$

$$\forall i \in \{1,2\}, \forall (\tau_1,\tau_2) \in \mathcal{T}_{12} : \mathsf{proj}_i((\tau_1,\tau_2)) = \tau_i \tag{3}$$

$$\forall (\tau_{12},\tau_2) \in \mathcal{T}_{12} \ {}_{\mathsf{proj}_2}\times_{\mathsf{id}_2} \mathcal{T}_2 : \mathsf{proj}'_1((\tau_{12},\tau_2)) = \mathsf{proj}_1(\tau_{12}) \tag{4}$$

$$\forall (\tau_{12},\tau_1) \in \mathcal{T}_{12} \ {}_{\mathsf{proj}_1}\times_{\mathsf{id}_1} \mathcal{T}_1 : \mathsf{proj}'_2((\tau_{12},\tau_1)) = \mathsf{proj}_2(\tau_{12}) \tag{5}$$

Moreover, for M' defined on $\mathcal{T}_{12}$ and V:

$$(M' \ {}_{\mathsf{proj}_2}\|_{\mathsf{id}_2} M_2) \ {}_{\mathsf{proj}'_1}\preceq_{\mathsf{id}_1} \ M_1 \Rightarrow M' \preceq M. \tag{6}$$

Proof. 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.

We add $(((s_{01},s_{02}),s_{02}),s_{01})$ to R. If there is a transition from the state $((s_{k1},s_{k2}),s_{k2})$ in the left TM of the refinement, i.e.,

$$\exists((s_{k1},s_{k2}),\mu,s) \in E, \exists(s_{k2},\mu_2,s'_{k2}) \in E_2 : \mu \ {}_{\mathsf{proj}_2}\bowtie_{\mathsf{id}_2} \mu_2,$$

then $s = (s'_{k1}, s'_{k2})$ for some $s'_{k1} \in S_1$. Indeed, the unifiability $\mu \ {}_{\mathsf{proj}_2}\bowtie_{\mathsf{id}_2} \mu_2$ implies $\mu = \mu_1 \ {}_{\rho_1}\sqcup_{\rho_2} \mu_2$ for some uniquely defined piece (by determinism) $\mu_1 \in L(V_1,\mathcal{T}_1)$ such that $\mu_1 \ {}_{\rho_1}\bowtie_{\rho_2} \mu_2$, implying $(s_{k1},\mu_1,s'_{k1}) \in E_1$, by definition of quotient. Hence $s = (s'_{k1}, s'_{k2})$. It is easy to see that $(\mu \ {}_{\mathsf{proj}_2}\sqcup_{\mathsf{id}_2} \mu_2) \ {}_{\mathsf{proj}'_1}\bowtie_{\mathsf{id}'_1} \mu_1$ and so $(((s'_{k1},s'_{k2}),s'_{k2}),s'_{k1}) \in R$. In addition, if $((s'_{k1},s'_{k2}),s'_{k2})$ is an accepting state, then $(s'_{k1},s'_{k2}) \in F$ and $s'_{k2} \in F_2$ from which we can infer that $s'_{k1} \in F_1$ by construction of F.

Assume 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 in M which can fully match r' and contradicts the assumption. Similarly, in the second case, assume that r' is unifiable with r_2 only for the first $k-1$ transitions. Then the $k-$th label μ'_k of r' can be decomposed uniquely into $\mu_1 \in L(V_1,\mathcal{T}_1)$ and $\mu_2 \in L(V_2,\mathcal{T}_2)$ such that $\mu'_k = \mu_1 \ {}_{\rho_1}\sqcup_{\rho_2} \mu_2$ and $\forall \bar{s}_2 : (s_2,\mu_2,\bar{s}_2) \notin E_2$. So there exists some run r in M that can match the first k transitions of r' and also the remaining transitions of r' since it can go to a universal state at the $k-$th transition. 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 $M' \preceq M$. □

Thus, the quotient M is the *greatest*, in the (homogeneous) refinement preorder, of all TMs M' defined on $\mathcal{T}_{12}$ and V and satisfying Property 6. This universal property is generally expected of quotients [15], and it alone implies that the quotient is uniquely defined up to two-sided homogeneous refinement [27]. As an example, Figure 3(c) shows a homogeneous quotient. Notice that when the quotient is computed between homogeneous tag structures, the fibered product $\mathcal{T}_{12}$ is isomorphic to the original tag structure using the identity morphisms. We can therefore simply reuse the original tag structure for the quotient.

4.1.3. Tag Machine Conjunction

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

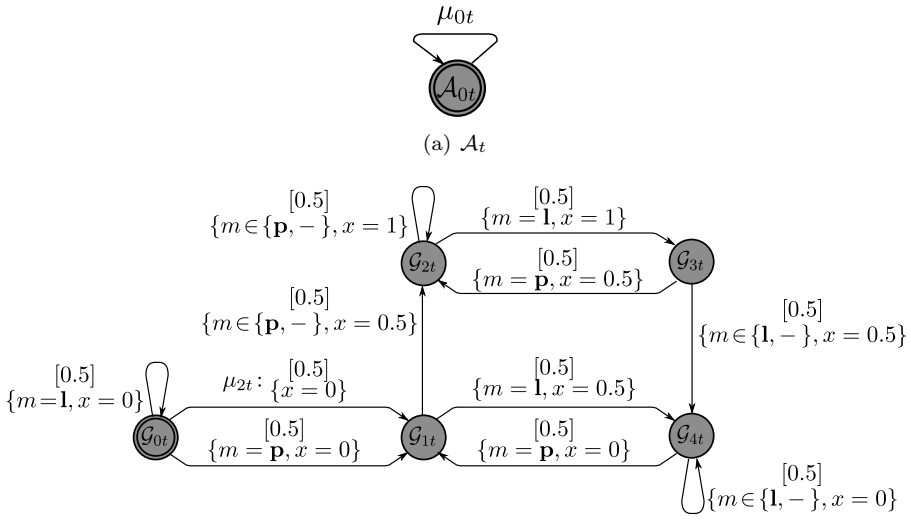
4.2. Tag Contracts

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 11. A tag contract is a homogeneous pair of TMs $(\mathcal{A}, \mathcal{G})$ where $\mathcal{A}$ - the assumption and $\mathcal{G}$ - the guarantee are TMs defined over the same tag structure $\mathcal{T}$ and variable set V.

Example 12. We consider the simplified water controlling system in Example 2 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 μ are then structured to represent an increment of δ by always assigning δ to $\mu(\mathfrak{inc}, \mathfrak{inc})$ and assigning δ 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.

Figure 2 depicts the tank contract $\mathcal{C}_t = (\mathcal{A}_t, \mathcal{G}_t)$ which guarantees a linear evolution of the water level $x(t)$ upon receiving controlling commands. The



(a) $\mathcal{A}_t$

(b) $\mathcal{G}_t$

Figure 2: The tank contract

controller contract is shown Figure 3, where it assumes the tank to be empty initially (Figure 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 (Figure 3(b)) to send a proper command upon knowing of the tank emptiness or fullness.

Similar to Example 2, the controller contract ensures timely control over the water evolution while the tank contract accepts untimely control and allow water spillages or shortages. In addition, we use the same tag structures, which are $\mathcal{T}_1 = (\mathbb{R}_+ \cup \{\epsilon_1\}, +)$ and $\mathcal{T}_2 = (\mathbb{N} \cup \{\epsilon_2\}, +)$ as in Example 2, to describe the tank and controller contracts respectively. We also use the same morphisms $\rho_1 : \mathcal{T}_1 \to \mathcal{T}_1$ and $\rho_2 : \mathcal{T}_2 \to \mathcal{T}_1$ given by $\rho_1(\tau_1) = \tau_1$, $\rho_2(\tau_2) = 0.5 * \tau_2$ when composing the two contracts. For the sake of expressiveness, some of the labeled tag pieces can also 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 Unknown. In addition, we use μ_{0t} to denote the universe set of labels $L(V_1, \mathcal{T}_1)$ and μ_{0c} the set of labels $L(V_2, \mathcal{T}_2)$.

The symbolic value representation could, as in our case, denote an infinite number of transitions. This has implications on the complexity of the analysis, including the algorithms that we will describe later in this paper. One solution to this problem is to consider a finite abstraction of the value domain, which could be accomplished using an abstract interpretation or a conservative approximation [28]. In our example, for instance, one could partition the $[0, 1]$

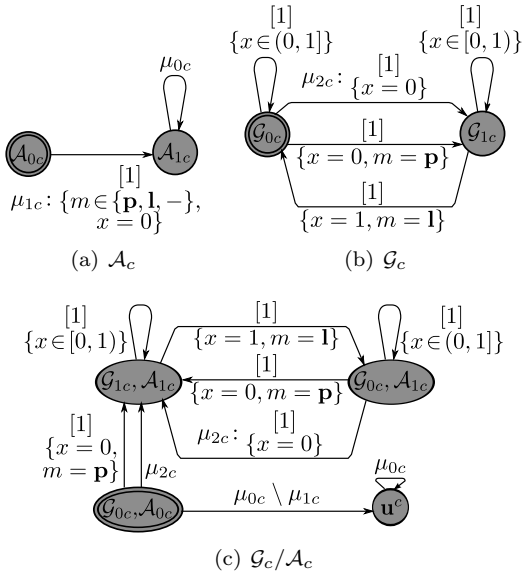


Figure 3: The controller contract

interval of the x variable finitely. Depending on the nature of the problem, in many cases, the abstraction will give exact results.

The tag contract semantics is subsequently defined through the notions of contract environments and implementations. Let $\mathcal{I}$ and $\mathcal{E}$ be TMs defined over tag structure $\mathcal{T}$ and variable set V in Definition 11. We call $\mathcal{E}$ an environment of contract $\mathcal{C}$ when $\mathcal{E}$ refines $\mathcal{A}$. Let $[\![\mathcal{C}]\!]_{\mathsf{e}}$ be the set of all such environments, we call $\mathcal{I}$ an implementation of contract $\mathcal{C}$, if it holds that $\forall \mathcal{E} \in [\![\mathcal{C}]\!]_{\mathsf{e}} : \mathcal{I} \,\|\, \mathcal{E} \preceq \mathcal{G} \,\|\, \mathcal{E}$. The set of implementations is similarly denoted by $[\![\mathcal{C}]\!]_{\mathsf{p}}$. We define the *semantics* of a contract $\mathcal{C}$ to be equal to its sets of environments $[\![\mathcal{C}]\!]_{\mathsf{e}}$ and implementations $[\![\mathcal{C}]\!]_{\mathsf{p}}$.

Checking that a component is an implementation of a contract involves checking a refinement relation for all the possible environments of the contract. When the contract is *normalized*, such a check can be done independently of the assumption instantiation and is reduced to finding a refinement relation between two tag machines.

Definition 13. Tag contract $\mathcal{C}$ is said to be *normalized* if and only if

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

To compute the normalized form of a contract, we need to find the most general specification of a component that, when composed with an environment which refines $\mathcal{A}$, satisfies (refines) the guarantees $\mathcal{G}$. This, intuitively, corresponds to the quotient $\mathcal{G}/\mathcal{A}$. The following results make these concepts precise.

Lemma 3. $\mathcal{G} \preceq \mathcal{G}/\mathcal{A}$.

PROOF. By contraposition, assume that $\mathcal{G} \not\preceq \mathcal{G}/\mathcal{A}$. Then, there must be runs in $\mathcal{G}$ which are not simulated by $\mathcal{G}/\mathcal{A}$. Consider one such minimal run r_g in $\mathcal{G}$, i.e., such that all its prefixes are also in $\mathcal{G}/\mathcal{A}$. There are two possible cases. In one case, there exists a run r in $\mathcal{G}/\mathcal{A}$ such that:

$$r_g : s_{0g} \xrightarrow{\mu_1} s_{1g} \dots s_{(n-1)g} \xrightarrow{\mu_n} s_{ng}$$
$$r : s_0 \xrightarrow{\mu_1} s_1 \dots s_{n-1} \not\xrightarrow{\mu_n}$$

where $n \geq 1$. It is easy to see that state s_i is not universal for all $0 \leq i \leq n-1$ since the last state s_{n-1} does not allow at least a transition labeled with μ_n. Thus, by definition of quotient and by determinism, r must derive from r_g by synchronizing with $\mathcal{A}$, i.e., $s_i = (s_{ig}, s_{ia})$. But then, $s_{n-1} \not\xrightarrow{\mu_n}$ implies that $s_{(n-1)g} \not\xrightarrow{\mu_n}$ and $s_{(n-1)a} \xrightarrow{\mu_n}$. Indeed, by the definition of quotient, if $s_{(n-1)g} \xrightarrow{\mu_n}$ then at the state $s_{n-1} = (s_{(n-1)g}, s_{(n-1)a})$ there must be a transition with the same label, i.e., $s_{n-1} \xrightarrow{\mu_n}$. The former implication obviously contradicts the existence of the last transition of r_g.

In the second case, s_{n-1} allows a transition labeled with μ_n but state s_n is not accepting while state s_{ng} is accepting. Therefore, state s_n is not universal and has a form of (s_{ng}, s_{na}) where s_{ng} is not accepting and s_{na} is accepting by the definition of quotient. This again contradicts the hypothesis which assumes that s_{ng} is accepting. As a result, the refinement $\mathcal{G} \preceq \mathcal{G}/\mathcal{A}$ holds. □

Normalization can be done by performing quotient between the contract guarantee and assumption, i.e. replacing $\mathcal{G}$ with $\mathcal{G}^{\mathfrak{n}} = \mathcal{G}/\mathcal{A}$. Indeed, this normalization is a weakening operation on the guarantee w.r.t. the assumption as shown in Lemma 3. This operation preserves the tag contract semantics, i.e., a contract and its normalized form have exactly the same set of environments and implementations, as shown in the following theorem.

Theorem 4. *Tag contract $(\mathcal{A}, \mathcal{G}/\mathcal{A})$ is in normalized form and has the same semantics as $\mathcal{C} = (\mathcal{A}, \mathcal{G})$ does.*

PROOF. Let $\bar{\mathcal{G}} = \mathcal{G}/\mathcal{A}$ and $M_q = \bar{\mathcal{G}}/\mathcal{A}$, then the refinement $M_q \preceq \bar{\mathcal{G}}$ must hold. Indeed, assume that there exists some run r_q in M_q and $r_{\bar{\mathcal{G}}}$ in $\bar{\mathcal{G}}$ where the last transition of r_q cannot be simulated by $r_{\bar{\mathcal{G}}}$ (considering, as in Lemma 3, the shortest such run).

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

This means the $s_{n\bar{\mathcal{G}}}$ is not universal. The construction of $r_{\bar{\mathcal{G}}}$ then implies the existence of runs $r_{\mathcal{A}}$ in $\mathcal{A}$ and $r_{\mathcal{G}}$ in $\mathcal{G}$:

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

The construction of 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 $\bar{\mathcal{G}}$. By determinism, $s_{k\bar{\mathcal{G}}} = s'_{k\bar{\mathcal{G}}}$ for $1 \leq k \leq n$ 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 $r_{\bar{\mathcal{G}}} \in \bar{\mathcal{G}}$. In addition, if $s_{nq} \in F_q$, then one of the following cases can happen:

i) $s_{nq} = \mathfrak{u}_q$: this implies $s_{n\bar{\mathcal{G}}} = \mathfrak{u}_{\bar{\mathcal{G}}}$ since all possible transitions allowed by s_{nq} must be simulated by $s_{n\bar{\mathcal{G}}}$,

ii) $s_{nq} \in F_{\bar{\mathcal{G}}} \times F_{\mathcal{A}}$: then $s_{n\bar{\mathcal{G}}} \in F_{\bar{\mathcal{G}}}$ by construction of r_q,

iii) $s_{nq} \in S_{\bar{\mathcal{G}}} \times (S_{\mathcal{A}} \setminus F_{\mathcal{A}})$: then $s_{n\mathcal{A}} \in S_{\mathcal{A}} \setminus F_{\mathcal{A}}$ and so $s_{n\bar{\mathcal{G}}} = (s_{n\mathcal{G}}, s_{n\mathcal{A}}) \in S_{\mathcal{G}} \times (S_{\mathcal{A}} \setminus F_{\mathcal{A}})$, implying $s_{n\bar{\mathcal{G}}} \in F_{\bar{\mathcal{G}}}$.

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

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

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

- $\Leftarrow$: $\mathcal{I} \preceq \bar{\mathcal{G}} \Rightarrow \forall \mathcal{E} \in [\![\bar{\mathcal{C}}]\!]_{\mathsf{e}} : (\mathcal{I} \parallel \mathcal{E}) \preceq (\bar{\mathcal{G}} \parallel \mathcal{E})$. Thus, $\mathcal{I} \in [\![\bar{\mathcal{C}}]\!]_{\mathsf{p}}$.

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

Thus implementation checking can be reduced to finding a refinement relation between an implementation and the normalized guarantee.

Example 14. We use the tag contracts in Example 12 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., $\mathcal{G}_t/\mathcal{A}_t = \mathcal{G}_t$. Figure 3(c), on the other hand, shows the normalized controller guarantee having more behaviors than the un-normalized one. Relying on the special clock $\mathbf{inc}$ to restructure the tank and controller machines in Example 2, it is easy to see that the tank machine refines $\mathcal{G}_t$ and the controller machine refines $\mathcal{G}_c/\mathcal{A}_c$. Therefore, both machines are implementations of the corresponding contract.

As we will see later, working with normalized tag contracts can simplify the formalization of contract operators (e.g. contract refinement and dominance) as well as provide a unique representation for equivalent contracts, thus we will often assume contracts to be in normalized form hereafter.

4.3. 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 $\mathcal{C}_i = (\mathcal{A}_i, \mathcal{G}_i)$ be tag contracts defined on $\mathcal{T}_i$ and V_i and $\rho_i : \mathcal{T}_i \to \mathcal{T}$ be algebraic tag morphisms where $i \in \{1, 2\}$.

Definition 15. Contract $\mathcal{C}_1$ refines contract $\mathcal{C}_2$ under morphisms ρ_1 and ρ_2, written $\mathcal{C}_1 \ {}_{\rho_1}\preceq_{\rho_2} \mathcal{C}_2$, if the following two conditions hold:

i) $\forall \mathcal{E}_2 \in [\![\mathcal{C}_2]\!]_{\mathsf{e}} : \exists \mathcal{E}_1 \in [\![\mathcal{C}_1]\!]_{\mathsf{e}} : \mathcal{E}_2 \ {}_{\rho_2}\preceq_{\rho_1} \mathcal{E}_1$

ii) $\forall \mathcal{I}_1 \in [\![\mathcal{C}_1]\!]_{\mathsf{p}} : \exists \mathcal{I}_2 \in [\![\mathcal{C}_2]\!]_{\mathsf{p}} : \mathcal{I}_1 \ {}_{\rho_1}\preceq_{\rho_2} \mathcal{I}_2$

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

Theorem 5. $\mathcal{C}_1 \ {}_{\rho_1}\preceq_{\rho_2} \mathcal{C}_2 \Leftrightarrow (\mathcal{A}_2 \ {}_{\rho_2}\preceq_{\rho_1} \mathcal{A}_1) \wedge (\mathcal{G}_1^{\mathsf{n}} \ {}_{\rho_1}\preceq_{\rho_2} \mathcal{G}_2^{\mathsf{n}})$

Proof.

- $\Rightarrow$: We first show that $(\mathcal{A}_2 \ {}_{\rho_2}\preceq_{\rho_1} \mathcal{A}_1)$ and then $(\mathcal{G}_1^{\mathsf{n}} \ {}_{\rho_1}\preceq_{\rho_2} \mathcal{G}_2^{\mathsf{n}})$.
  - $\mathcal{A}_2 \ {}_{\rho_2}\preceq_{\rho_1} \mathcal{A}_1$: The fact that $\mathcal{C}_1 \ {}_{\rho_1}\preceq_{\rho_2} \mathcal{C}_2$ (by assumption) and that $\mathcal{A}_2 \in [\![\mathcal{C}_2]\!]_{\mathsf{e}}$ together imply that
    $$\exists \mathcal{E}_1 \in [\![\mathcal{C}_1]\!]_{\mathsf{e}} : \mathcal{A}_2 \ {}_{\rho_2}\preceq_{\rho_1} \mathcal{E}_1,$$
    by the first condition of Definition 15. Because $\mathcal{E}_1$ is an environment of $\mathcal{C}_1$, it follows that $\mathcal{E}_1 \preceq \mathcal{A}_1$ by definition of environment, from which we can infer that $\mathcal{A}_2 \ {}_{\rho_2}\preceq_{\rho_1} \mathcal{A}_1$.
  - $\mathcal{G}_1^{\mathsf{n}} \ {}_{\rho_1}\preceq_{\rho_2} \mathcal{G}_2^{\mathsf{n}}$: The fact that $\mathcal{C}_1 \ {}_{\rho_1}\preceq_{\rho_2} \mathcal{C}_2$ (by assumption) and that $\mathcal{G}_1^{\mathsf{n}} \in [\![\mathcal{C}_1]\!]_{\mathsf{p}}$ together imply that
    $$\exists \mathcal{I}_2 \in [\![\mathcal{C}_2]\!]_{\mathsf{p}} : \mathcal{G}_1^{\mathsf{n}} \ {}_{\rho_1}\preceq_{\rho_2} \mathcal{I}_2,$$
    by the second condition of Definition 15. Because $\mathcal{I}_2$ is an implementation of $\mathcal{C}_2$, and $\mathcal{G}_2^{\mathsf{n}}$ is the normalized guarantee of $\mathcal{C}_2$, it follows that $\mathcal{I}_2 \preceq \mathcal{G}_2^{\mathsf{n}}$ by Definition 13, from which we can infer that $\mathcal{G}_1^{\mathsf{n}} \ {}_{\rho_1}\preceq_{\rho_2} \mathcal{G}_2^{\mathsf{n}}$.
- $\Leftarrow$: We show the satisfaction of the two conditions of Definition 15.
  - The first condition holds because by definition of environment, it follows that
    $$\forall \mathcal{E}_2 \in [\![\mathcal{C}_2]\!]_{\mathsf{e}} : \mathcal{E}_2 \preceq \mathcal{A}_2.$$
    Together with the fact that $\mathcal{A}_2 \ {}_{\rho_2}\preceq_{\rho_1} \mathcal{A}_1$, this implies $\mathcal{E}_2 \ {}_{\rho_2}\preceq_{\rho_1} \mathcal{A}_1$.
  - The second condition holds because by the normalization property of $\mathcal{C}_1$, it follows that
    $$\forall \mathcal{I}_1 \in [\![\mathcal{C}_1]\!]_{\mathsf{p}} : \mathcal{I}_1 \preceq \mathcal{G}_1^{\mathsf{n}}.$$
    Together with the fact that $\mathcal{G}_1^{\mathsf{n}} \ {}_{\rho_1}\preceq_{\rho_2} \mathcal{G}_2^{\mathsf{n}}$, this implies $\mathcal{I}_1 \ {}_{\rho_1}\preceq_{\rho_2} \mathcal{G}_2^{\mathsf{n}}$.

□

4.4. Tag Contract Dominance

In composing two heterogeneous tag contracts, it is essential to guarantee that composing implementations of each contract results in a new implementation of the composite contract. In addition, every environment of the composite contract should be able to work with any implementation of one 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 [15].

Definition 16. A contract $\mathcal{C} = (\mathcal{A}, \mathcal{G})$ is said to *dominate* the tag contract pair $(\mathcal{C}_1, \mathcal{C}_2)$ under morphisms ρ_1 and ρ_2 if $\mathcal{C}$ 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 $V = V_1 \cup V_2$ and the following conditions hold:

i) $\forall \mathcal{I}_1 \in [\![\mathcal{C}_1]\!]_{\mathsf{p}}, \forall \mathcal{I}_2 \in [\![\mathcal{C}_2]\!]_{\mathsf{p}} : \mathcal{I}_1 \,{}_{\rho_1}\|_{\rho_2}\, \mathcal{I}_2 \in [\![\mathcal{C}]\!]_{\mathsf{p}}$

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

where the morphisms are defined in (2), (3), (4), (5) of Theorem 2.

4.5. Tag Contract Composition

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

Definition 17. The composition of tag contracts $\mathcal{C}_1$ and $\mathcal{C}_2$, written as $\mathcal{C}_1 \,{}_{\rho_1}\|_{\rho_2}\, \mathcal{C}_2$, is the tag contract $((\mathcal{A}_1 \,{}_{\rho_1}/_{\rho_2}\, \mathcal{G}_2^{\mathfrak{n}}) \curlywedge (\mathcal{A}_2 \,{}_{\rho_2}/_{\rho_1}\, \mathcal{G}_1^{\mathfrak{n}})_{\mathsf{swap}}, \mathcal{G}_1^{\mathfrak{n}} \,{}_{\rho_1}\|_{\rho_2}\, \mathcal{G}_2^{\mathfrak{n}})$ where $\mathsf{swap} : \mathcal{T}_2 \,{}_{\rho_2}\!\times_{\rho_1} \mathcal{T}_1 \to \mathcal{T}_1 \,{}_{\rho_1}\!\times_{\rho_2} \mathcal{T}_2$ is such that $\mathsf{swap}((\tau_2, \tau_1)) = ((\tau_1, \tau_2))$ and M_{swap} is M where all pieces μ are replaced with $\mu \circ \mathsf{swap}$.

Such composition dominates the individual contracts and is the *least*, in the homogeneous refinement order, of all contracts dominating them under the same morphisms.

Theorem 6. *Let $\mathcal{C} = \mathcal{C}_1 \,{}_{\rho_1}\|_{\rho_2}\, \mathcal{C}_2$, then:*

i) $\mathcal{C}$ dominates the contract pair $(\mathcal{C}_1, \mathcal{C}_2)$ under morphisms ρ_1 and ρ_2.

ii) If $\mathcal{C}'$ dominates $(\mathcal{C}_1, \mathcal{C}_2)$ under morphisms ρ_1 and ρ_2 then $\mathcal{C} \preceq \mathcal{C}'$.

Proof. Let $\mathcal{C} = (\mathcal{A}, \mathcal{G}) = ((\mathcal{A}_1 \,{}_{\rho_1}/_{\rho_2}\, \mathcal{G}_2^{\mathfrak{n}}) \curlywedge (\mathcal{A}_2 \,{}_{\rho_2}/_{\rho_1}\, \mathcal{G}_1^{\mathfrak{n}})_{\mathsf{swap}}, \mathcal{G}_1^{\mathfrak{n}} \,{}_{\rho_1}\|_{\rho_2}\, \mathcal{G}_2^{\mathfrak{n}})$. Contract $\mathcal{C}$ dominates $(\mathcal{C}_1, \mathcal{C}_2)$ under ρ_1 and ρ_2 as:

a) $\mathcal{C}$ is defined over $\mathcal{T}_{12} = \mathcal{T}_1 \,{}_{\rho_1}\!\times_{\rho_2} \mathcal{T}_2$ and $V = V_1 \cup V_2$, by Definition 17.

b) $\mathcal{I}_i \in [\![\mathcal{C}_i]\!]_{\mathsf{p}} \Rightarrow \mathcal{I}_i \preceq \mathcal{G}_i^{\mathfrak{n}}$ (by Theorem 4). Thus, $(\mathcal{I}_1 \,{}_{\rho_1}\|_{\rho_2}\, \mathcal{I}_2) \preceq (\mathcal{G}_1^{\mathfrak{n}} \,{}_{\rho_1}\|_{\rho_2}\, \mathcal{G}_2^{\mathfrak{n}})$, or equivalently $(\mathcal{I}_1 \,{}_{\rho_1}\|_{\rho_2}\, \mathcal{I}_2) \in [\![\mathcal{C}]\!]_{\mathsf{p}}$.

c) We observe that $\overline{\mathsf{proj}}_1(\tau_{21}) = \mathsf{proj}_1 \circ \mathsf{swap}(\tau_{21})$ for $\tau_{21} \in \mathcal{T}_2 \ {}_{\rho_2}\!\times_{\rho_1} \mathcal{T}_1$ and $\overline{\mathsf{proj}}'_2((\tau_{21}, \tau_1)) = \mathsf{proj}_2 \circ \mathsf{swap}(\tau_{21})$ for $(\tau_{21}, \tau_1) \in (\mathcal{T}_2 \ {}_{\rho_2}\!\times_{\rho_1} \mathcal{T}_1) \, {}_{\overline{\mathsf{proj}}_1}\!\times_{\mathsf{id}_1} \mathcal{T}_1$. Then by the quotient construction:

$$((\mathcal{A}_1 \, {}_{\rho_1}\!/_{\rho_2} \mathcal{G}_2^{\mathfrak{n}}) \ {}_{\mathsf{proj}_2}\!\|_{\mathsf{id}_2} \mathcal{G}_2^{\mathfrak{n}}) \ {}_{\mathsf{proj}'_1}\!\preceq_{\mathsf{id}_1} \mathcal{A}_1 \Rightarrow ((\mathcal{A}_1 \, {}_{\rho_1}\!/_{\rho_2} \mathcal{G}_2^{\mathfrak{n}}) \ {}_{\mathsf{proj}_2}\!\|_{\mathsf{id}_2} \mathcal{G}_2^{\mathfrak{n}}) \ {}_{\mathsf{proj}'_1}\!\preceq_{\mathsf{id}_1} \mathcal{A}_1$$

$$((\mathcal{A}_2 \, {}_{\rho_2}\!/_{\rho_1} \mathcal{G}_1^{\mathfrak{n}}) \, {}_{\overline{\mathsf{proj}}_1}\!\|_{\mathsf{id}_1} \mathcal{G}_1^{\mathfrak{n}}) \ {}_{\overline{\mathsf{proj}}'_2}\!\preceq_{\mathsf{id}_2} \mathcal{A}_2 \Rightarrow ((\mathcal{A}_2 \, {}_{\rho_2}\!/_{\rho_1} \mathcal{G}_1^{\mathfrak{n}})_{\mathsf{swap}} \ {}_{\mathsf{proj}_1}\!\|_{\mathsf{id}_1} \mathcal{G}_1^{\mathfrak{n}}) \ {}_{\mathsf{proj}'_2}\!\preceq_{\mathsf{id}_2} \mathcal{A}_2$$

In addition, $\mathcal{E} \in [\![\mathcal{C}]\!]_{\mathsf{e}}$ means $\mathcal{E} \preceq \mathcal{A}$. Therefore:

$$\mathcal{E} \preceq (\mathcal{A}_1 \, {}_{\rho_1}\!/_{\rho_2} \mathcal{G}_2^{\mathfrak{n}}) \Rightarrow (\mathcal{E} \ {}_{\mathsf{proj}_2}\!\|_{\mathsf{id}_2} \mathcal{I}_2) \preceq ((\mathcal{A}_1 \, {}_{\rho_1}\!/_{\rho_2} \mathcal{G}_2^{\mathfrak{n}}) \ {}_{\mathsf{proj}_2}\!\|_{\mathsf{id}_2} \mathcal{G}_2^{\mathfrak{n}}) \ {}_{\mathsf{proj}'_1}\!\preceq_{\mathsf{id}_1} \mathcal{A}_1$$

$$\mathcal{E} \preceq (\mathcal{A}_2 \, {}_{\rho_2}\!/_{\rho_1} \mathcal{G}_1^{\mathfrak{n}})_{\mathsf{swap}} \Rightarrow (\mathcal{E} \ {}_{\mathsf{proj}_1}\!\|_{\mathsf{id}_1} \mathcal{I}_1) \preceq ((\mathcal{A}_2 \, {}_{\rho_2}\!/_{\rho_1} \mathcal{G}_1^{\mathfrak{n}})_{\mathsf{swap}} \ {}_{\mathsf{proj}_1}\!\|_{\mathsf{id}_1} \mathcal{G}_1^{\mathfrak{n}}) \ {}_{\mathsf{proj}'_2}\!\preceq_{\mathsf{id}_2} \mathcal{A}_2$$

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 $\mathcal{A}' \preceq \mathcal{A}$. Since $\mathcal{A}' \in [\![\mathcal{C}']\!]_{\mathsf{e}}$ and $\mathcal{G}_i \in [\![\mathcal{C}_i]\!]_{\mathsf{p}}$ and $\mathcal{C}'$ dominates $(\mathcal{C}_1, \mathcal{C}_2)$ under the same morphisms ρ_1 and ρ_2, the following holds:

$$((\mathcal{A}' \ {}_{\mathsf{proj}_1}\!\|_{\mathsf{id}_1} \mathcal{G}_1^{\mathfrak{n}}) \ {}_{\mathsf{proj}'_2}\!\preceq_{\mathsf{id}_2} \mathcal{A}_2) \ \wedge ((\mathcal{A}' \ {}_{\mathsf{proj}_2}\!\|_{\mathsf{id}_2} \mathcal{G}_2^{\mathfrak{n}}) \ {}_{\mathsf{proj}'_1}\!\preceq_{\mathsf{id}_1} \mathcal{A}_1)$$

implying $(\mathcal{A}' \preceq (\mathcal{A}_2 \, {}_{\rho_2}\!/_{\rho_1} \mathcal{G}_1^{\mathfrak{n}})_{\mathsf{swap}}) \wedge (\mathcal{A}' \preceq (\mathcal{A}_1 \, {}_{\rho_1}\!/_{\rho_2} \mathcal{G}_2^{\mathfrak{n}}))$ or $\mathcal{A}' \preceq \mathcal{A}$, by Theorem 2. We next show that an implementation of $\mathcal{C}$ is also an implementation of $\mathcal{C}'$.

$$\begin{aligned}
\mathcal{I} \in [\![\mathcal{C}]\!]_{\mathsf{p}} &\Rightarrow \forall \mathcal{E} \in [\![\mathcal{C}]\!]_{\mathsf{e}} : \mathcal{I} \parallel \mathcal{E} \preceq \mathcal{G} \parallel \mathcal{E} \\
&\Rightarrow \forall \mathcal{E} \in [\![\mathcal{C}']\!]_{\mathsf{e}} : \mathcal{I} \parallel \mathcal{E} \preceq \mathcal{G} \parallel \mathcal{E}, \text{ as } \mathcal{A}' \preceq \mathcal{A} \Rightarrow [\![\mathcal{C}']\!]_{\mathsf{e}} \subseteq [\![\mathcal{C}]\!]_{\mathsf{e}} \\
&\Rightarrow \forall \mathcal{E} \in [\![\mathcal{C}']\!]_{\mathsf{e}} : \mathcal{I} \parallel \mathcal{E} \preceq \mathcal{G}' \parallel \mathcal{E}, \text{ as } \mathcal{C}' \text{ dominates } (\mathcal{C}_1, \mathcal{C}_2) \Rightarrow \mathcal{G} \in [\![\mathcal{C}']\!]_{\mathsf{p}}
\end{aligned}$$

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

Let $\mathcal{C}'_i$ be tag contracts defined on $\mathcal{T}_i$ and V_i such that $\mathcal{C}'_i \preceq \mathcal{C}_i$. The next theorem is another of *independent implementability*: homogeneous tag contract refinement is preserved under the heterogeneous contract composition.

Theorem 7. *If $\mathcal{C}$ dominates $(\mathcal{C}_1, \mathcal{C}_2)$ under morphisms ρ_1 and ρ_2 then it also dominates $(\mathcal{C}'_1, \mathcal{C}'_2)$ under the same morphisms. In addition, $(\mathcal{C}'_1 \ {}_{\rho_1}\!\|_{\rho_2} \mathcal{C}'_2) \preceq (\mathcal{C}_1 \ {}_{\rho_1}\!\|_{\rho_2} \mathcal{C}_2)$.*

Proof. The first statement holds because the first two conditions in Definition 16 can be deduced from the fact that $[\![\mathcal{C}'_1]\!]_{\mathsf{p}} \subseteq [\![\mathcal{C}_1]\!]_{\mathsf{p}}$, $[\![\mathcal{C}'_2]\!]_{\mathsf{p}} \subseteq [\![\mathcal{C}_2]\!]_{\mathsf{p}}$, $\mathcal{A}_1 \preceq \mathcal{A}'_1$, $\mathcal{A}_2 \preceq \mathcal{A}'_2$ and $\mathcal{C}$ dominates $(\mathcal{C}_1, \mathcal{C}_2)$ under ρ_1 and ρ_2. Considering $\mathcal{C} = \mathcal{C}_1 \ {}_{\rho_1}\!\|_{\rho_2} \mathcal{C}_2$, the second statement follows directly from the first statement of this theorem and the second property of Theorem 6. □

4.6. Tag Contract Compatibility

Of particular interest is the notion of *compatibility* between contracts. This notion depends critically on the contract profiles. Tag contract $\mathcal{C}$ can also be associated with a *profile* $\pi = (V^{\mathfrak{i}}, V^{\mathfrak{o}})$ which is a partition of its variables into inputs and outputs, i.e. $V = V^{\mathfrak{i}} \cup V^{\mathfrak{o}}$ and $V^{\mathfrak{i}} \cap V^{\mathfrak{o}} = \emptyset$. When composing contracts $\mathcal{C}_i$ with profiles π_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 profile is then $\pi = ((V_1^{\mathfrak{i}} \cup V_2^{\mathfrak{i}}) \setminus (V_1^{\mathfrak{o}} \cup V_2^{\mathfrak{o}}), V_1^{\mathfrak{o}} \cup V_2^{\mathfrak{o}})$.

Figure 4: $M_{\mathfrak{u}}$

Example 18. The tank and controller contracts in Example 12 are naturally associated with profiles $\pi_t = (\{m\}, \{x\})$ and $\pi_c = (\{x\}, \{m\})$ respectively. The profile of their composition is then $\pi = (\emptyset, \{x, m\})$.

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*. Certain models are not input-enabled, e.g., interface automata, because they use input refusal to represent assumptions implicitly. We instead can afford this desirable property as assumptions are represented separately in our framework. 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 $\pi_e = (V_1^{\mathsf{o}} \cup V_2^{\mathsf{o}}, (V_1^{\mathsf{i}} \cup V_2^{\mathsf{i}}) \setminus (V_1^{\mathsf{o}} \cup V_2^{\mathsf{o}}))$ such that:

- $\mathcal{A}_e = M_{\mathfrak{u}}$, c.f. Figure 4, 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 $\mathcal{C}_1 \ {}_{\rho_1}\|_{\rho_2} \ \mathcal{C}_2 = (\mathcal{A}, \mathcal{G}) = ((\mathcal{A}_1 {}_{\rho_1}\!/_{\rho_2} \mathcal{G}_2^{\mathfrak{n}}) \between (\mathcal{A}_2 {}_{\rho_2}\!/_{\rho_1} \mathcal{G}_1^{\mathfrak{n}})_{\mathsf{swap}}, \mathcal{G}_1^{\mathfrak{n}} \ {}_{\rho_1}\|_{\rho_2} \ \mathcal{G}_2^{\mathfrak{n}})$ and $\mathcal{C}_e$ should also weaken the assumption made on its environment to the greatest extent. That is $(\mathcal{A}_e / \mathcal{G}^{\mathfrak{n}}) \between (\mathcal{A} / \mathcal{G}_e^{\mathfrak{n}}) = M_{\mathfrak{u}}$ as well.
- $\mathcal{G}_e$ is input-enabled so as to make contract $\mathcal{C}_e$ consistent.

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

Example 19. We consider again the water tank controlling problem in Example 12 and the two contracts on the tank and the controller. Since $\mathcal{A}_t \ {}_{\rho_1}\!\preceq_{\rho_2} \mathcal{G}_c^{\mathfrak{n}}$ and $\mathcal{A}_c \ {}_{\rho_2}\!\preceq_{\rho_2} \mathcal{G}_t^{\mathfrak{n}}$, the composite assumption of these two contracts which is the conjunction $(\mathcal{A}_t {}_{\rho_1}\!/_{\rho_2} \mathcal{G}_c^{\mathfrak{n}}) \wedge (\mathcal{A}_c {}_{\rho_2}\!/_{\rho_1} \mathcal{G}_t^{\mathfrak{n}})_{\mathsf{swap}}$ accepts all behaviors defined on variable set $V = \{x, m\}$ and tag structure $\mathcal{T}_1 {}_{\rho_1}\!\times_{\rho_2} \mathcal{T}_2$. Therefore $\mathcal{G}_e^{\mathfrak{n}}$ can always refine $(\mathcal{A}_t {}_{\rho_1}\!/_{\rho_2} \mathcal{G}_c^{\mathfrak{n}}) \wedge (\mathcal{A}_c {}_{\rho_2}\!/_{\rho_1} \mathcal{G}_t^{\mathfrak{n}})_{\mathsf{swap}}$. Hence the two contracts are compatible.

5. Tag Contract Synthesis

When tag contracts are used to represent properties of heterogeneous sub-components in a system, verifying whether composing the sub-components' properties retains the system's property amounts to verifying whether composing the sub-components' associated contracts refines the system's overall contract. To enable such verification, we rely on the fact that the composition of two tag contracts $\mathcal{C}_1$ and $\mathcal{C}_2$ refines a contract $\mathcal{C}$ if and only if $\mathcal{C}$ dominates $\mathcal{C}_1$ and $\mathcal{C}_2$ w.r.t. the same morphisms that are used in composing them. When the verification is negative, i.e., one of the conditions described in Definition 16 is not satisfied, we must adjust or *synthesize* the individual contracts in order to gain the dominance satisfaction. Such dominance conditions are first shown to be equivalent to simpler formulas in the following theorem.

Theorem 8. *Condition* (i) *is equivalent to condition* ***DC-1)*** *and condition* (ii) *equivalent to the conjunction of* ***DC-2a)*** *and* ***DC-2b)****:*

$$\mathcal{G}_1^{\mathfrak{n}} \;{}_{\rho_1}\|_{\rho_2}\; \mathcal{G}_2^{\mathfrak{n}} \preceq \mathcal{G}^{\mathfrak{n}} \qquad \textbf{\textit{(DC-1)}}$$

$$(\mathcal{A} \;{}_{\mathsf{proj}_1}\|_{\mathsf{id}_1}\; \mathcal{G}_1^{\mathfrak{n}}) \;{}_{\mathsf{proj}_2'}\preceq_{\mathsf{id}_2}\; \mathcal{A}_2 \qquad \textbf{\textit{(DC-2a)}}$$

$$(\mathcal{A} \;{}_{\mathsf{proj}_2}\|_{\mathsf{id}_2}\; \mathcal{G}_2^{\mathfrak{n}}) \;{}_{\mathsf{proj}_1'}\preceq_{\mathsf{id}_1}\; \mathcal{A}_1 \qquad \textbf{\textit{(DC-2b)}}$$

Proof. Condition (i) is equivalent to condition **DC-1)** because:

- $\Rightarrow$: By definition of normalized contract (Def. 13), $\mathcal{G}_i^{\mathfrak{n}} \in [\![\mathcal{C}_i]\!]_{\mathsf{p}}$. Therefore, by condition (i) and by definition of normalized contract:

$$(\mathcal{G}_1^{\mathfrak{n}} \in [\![\mathcal{C}_1]\!]_{\mathsf{p}} \wedge \mathcal{G}_2^{\mathfrak{n}} \in [\![\mathcal{C}_2]\!]_{\mathsf{p}}) \Rightarrow (\mathcal{G}_1^{\mathfrak{n}} \;{}_{\rho_1}\|_{\rho_2}\; \mathcal{G}_2^{\mathfrak{n}}) \in [\![\mathcal{C}]\!]_{\mathsf{p}} \Rightarrow (\mathcal{G}_1^{\mathfrak{n}} \;{}_{\rho_1}\|_{\rho_2}\; \mathcal{G}_2^{\mathfrak{n}}) \preceq \mathcal{G}^{\mathfrak{n}}$$

- $\Leftarrow$: $\mathcal{I}_i \preceq \mathcal{G}_i^{\mathfrak{n}} \Rightarrow (\mathcal{I}_1 \;{}_{\rho_1}\|_{\rho_2}\; \mathcal{I}_2) \preceq (\mathcal{G}_1^{\mathfrak{n}} \;{}_{\rho_1}\|_{\rho_2}\; \mathcal{G}_2^{\mathfrak{n}}) \preceq \mathcal{G}^{\mathfrak{n}} \Rightarrow (\mathcal{I}_1 \;{}_{\rho_1}\|_{\rho_2}\; \mathcal{I}_2) \in [\![\mathcal{C}]\!]_{\mathsf{p}}$

Condition (ii) is equivalent to the conjunction of **DC-2a)** and **DC-2b)** as:

- $\Rightarrow$: Let $\mathcal{E} = \mathcal{A}, \mathcal{I}_i = \mathcal{G}_i^{\mathfrak{n}}$.
- $\Leftarrow$: By the definition of environment and implementation, we have:

$$(\mathcal{E} \;{}_{\mathsf{proj}_1}\|_{\mathsf{id}_1}\; \mathcal{I}_1) \preceq (\mathcal{A} \;{}_{\mathsf{proj}_1}\|_{\mathsf{id}_1}\; \mathcal{G}_1^{\mathfrak{n}}) \;{}_{\mathsf{proj}_2'}\preceq_{\mathsf{id}_2}\; \mathcal{A}_2$$

$$(\mathcal{E} \;{}_{\mathsf{proj}_2}\|_{\mathsf{id}_2}\; \mathcal{I}_2) \preceq (\mathcal{A} \;{}_{\mathsf{proj}_2}\|_{\mathsf{id}_2}\; \mathcal{G}_2^{\mathfrak{n}}) \;{}_{\mathsf{proj}_1'}\preceq_{\mathsf{id}_1}\; \mathcal{A}_1$$

□

To obtain the satisfaction of **DC-1**, one could try to synthesize for example $\mathcal{G}_1^{\mathfrak{n}}$ by doing the following steps. First, a heterogeneous quotient operation between $\mathcal{G}^{\mathfrak{n}}$ and $\mathcal{G}_2^{\mathfrak{n}}$ (which could be $\mathcal{G}^{\mathfrak{n}} \,{}_{\mathsf{proj}_2}/_{\mathsf{id}_2} \mathcal{G}_2^{\mathfrak{n}}$) is computed. Since the tag structure of the quotient is a fibered product defined over $\mathcal{T}_1$ and $\mathcal{T}_2$, a second step is to extract from it behaviors over $\mathcal{T}_1$ only, obtaining $\bar{\mathcal{G}}_1^{\mathfrak{n}}$ in the end. However, doing so can still retain in the composition behaviors which cannot be simulated by $\mathcal{G}^{\mathfrak{n}}$ as shown in Example 20, i.e. $\bar{\mathcal{G}}_1^{\mathfrak{n}} \;{}_{\rho_1}\|_{\rho_2}\; \mathcal{G}_2^{\mathfrak{n}} \not\preceq \mathcal{G}^{\mathfrak{n}}$. This is because the tag morphisms can be many-to-one mappings in general.

Example 20. We consider an example where:

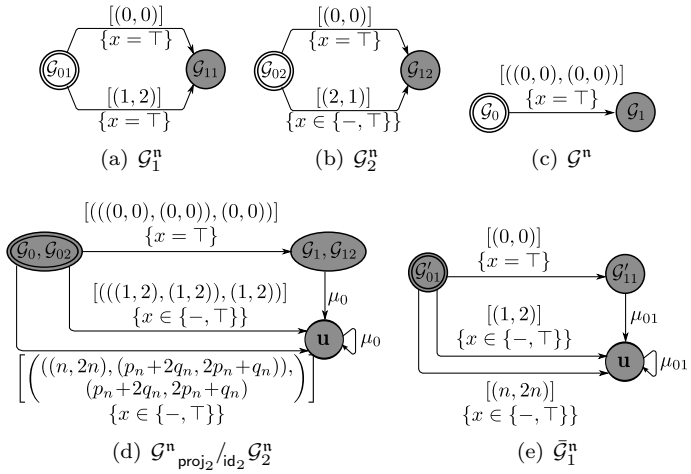
- $T_1 = \{(-\infty, -\infty), (0,0), (k, 2k)\}$ with $k \in \mathbb{N} \wedge k \geq 1$.
- $T_2 = \{(-\infty, -\infty), (0,0), (i+2j, 2i+j)\}$ with $i, j \in \mathbb{N} \wedge i, j \geq 0$.
- $\leq_1 = \leq_2$ and is defined such that $(\tau_1, \tau_2) \leq_1 (\tau_1', \tau_2') \Leftrightarrow (\tau_1 \leq \tau_1') \wedge (\tau_2 \leq \tau_2')$.
- $+_1 = +_2$ and is defined such that $(\tau_1, \tau_2) +_1 (\tau_1', \tau_2') = (\tau_1 + \tau_1', \tau_2 + \tau_2')$.

It is easy to see that $\mathcal{T}_1 \stackrel{\text{def}}{=} (T_1, \leq_1, +_1)$ and $\mathcal{T}_2 \stackrel{\text{def}}{=} (T_2, \leq_2, +_2)$ and $\mathcal{T} \stackrel{\text{def}}{=} (\mathbb{N} \cup \{-\infty\}, \leq, +)$ are algebraic tag structures. Assume that we have the algebraic tag morphisms $\rho_1 : \mathcal{T}_1 \to \mathcal{T}$ and $\rho_2 : \mathcal{T}_2 \to \mathcal{T}$ such that $\rho_1((\tau_1, \tau_2)) = \rho_2((\tau_1, \tau_2)) = \tau_1 + \tau_2$. We consider three sets of behaviors represented by TMs $\mathcal{G}_1^{\mathfrak{n}}$, $\mathcal{G}_2^{\mathfrak{n}}$ and $\mathcal{G}^{\mathfrak{n}}$ as shown in Figure 5(a), 5(b) and 5(c) respectively. These three sets are defined on tag structures $\mathcal{T}_1$, $\mathcal{T}_2$, $\mathcal{T}_1 {}_{\rho_1}\!\times_{\rho_2} \mathcal{T}_2$ and on the same variable set $V_1 = V_2 = V = \{x\}$ with the domain of value $D_x = \{\top\}$.

It is obvious that the composition $\mathcal{G}_1^{\mathfrak{n}} \,{}_{\rho_1}\|_{\rho_2}\, \mathcal{G}_2^{\mathfrak{n}}$ does not refine $\mathcal{G}^{\mathfrak{n}}$. Because machine $\mathcal{G}_1^{\mathfrak{n}}$ can take a transition labeled by tag piece $\mu_{11} \stackrel{\text{def}}{=} [(1,2)]$ at state $\mathcal{G}_{01}$ and machine $\mathcal{G}_2^{\mathfrak{n}}$ can take that labeled by $\mu_{12} \stackrel{\text{def}}{=} [(2,1)]$ at state $\mathcal{G}_{02}$, both agree on assigning variable x to $\top$. However, there is no transition with label $\mu_{11} \,{}_{\rho_1}\!\sqcup_{\rho_2}\, \mu_{12}$ allowed at state $\mathcal{G}_0$ of machine $\mathcal{G}^{\mathfrak{n}}$, hence the refinement failure.

Figure 5(d) shows the result of performing a heterogeneous quotient between $\mathcal{G}^{\mathfrak{n}}$ and $\mathcal{G}_2^{\mathfrak{n}}$ where $n, p_n, q_n \in \mathbb{N} \wedge n \geq 2 \wedge p_n + q_n = n$ and μ_0 is any label of the universe set $L(V, \mathcal{T}_1 \,{}_{\rho_1}\!\times_{\rho_2}\, \mathcal{T}_2)$. The result of projecting the quotient on the tag domain $\mathcal{T}_1$ is shown in Figure 5(e) where μ_{01} is any label of the universe set $L(V, \mathcal{T}_1)$. Its composition with machine $\mathcal{G}_2^{\mathfrak{n}}$ still does not refine machine $\mathcal{G}^{\mathfrak{n}}$. This is because the morphisms are many-to-one mappings and the projection operation erases the tag fibered relations formed by these morphisms.

The above example shows that undesirable behaviors cannot be eliminated in the heterogeneous quotient because morphisms can be many-to-one in general. In fact, whenever the unification of two behaviors cannot be simulated, one of them should be pruned away completely. Algorithm 1 demonstrates how this



(a) $\mathcal{G}_1^{\mathfrak{n}}$ (b) $\mathcal{G}_2^{\mathfrak{n}}$ (c) $\mathcal{G}^{\mathfrak{n}}$

(d) $\mathcal{G}^{\mathfrak{n}}{}_{\mathsf{proj}_2}\!/_{\mathsf{id}_2}\mathcal{G}_2^{\mathfrak{n}}$ (e) $\bar{\mathcal{G}}_1^{\mathfrak{n}}$

Figure 5: Synthesis based on heterogeneneous quotient and projection

can be done.

Input: $\mathcal{G}_1^{\mathfrak{n}} = (V_1, \mathcal{T}_1, S_{g_1}, s_{0g_1}, F_{g_1}, E_{g_1})$;
$\mathcal{G}_2^{\mathfrak{n}} = (V_2, \mathcal{T}_2, S_{g_2}, s_{0g_2}, F_{g_2}, E_{g_2})$;
$\mathcal{G}^{\mathfrak{n}} = (V, \mathcal{T}_1 {}_{\rho_1}\!\times_{\rho_2} \mathcal{T}_2, S_g, s_{0g}, F_g, E_g)$;

Output: $\bar{\mathcal{G}}_1^{\mathfrak{n}} = (V_1, \mathcal{T}_1, S_{g_1}, s_{0g_1}, \bar{F}_{g_1}, \bar{E}_{g_1})$ such that $\bar{\mathcal{G}}_1^{\mathfrak{n}} {}_{\rho_1}\|_{\rho_2} \mathcal{G}_2^{\mathfrak{n}} \preceq \mathcal{G}^{\mathfrak{n}}$

$\bar{F}_{g_1} = F_{g_1}, \bar{E}_{g_1} = E_{g_1}, R' = \emptyset, R = \{((s_{0g_1}, s_{0g_2}), s_{0g})\}$;
while $(R \neq R')$ **do**
  $R' = R$;
  for *every* $((s_{kg_1}, s_{kg_2}), s_{kg}) \in R'$ **do**
    for *every* $(s_{kg_1}, \mu_1, s_{(k+1)g_1}) \in \bar{E}_{g_1}$ **do**
      for *every* $(s_{kg_2}, \mu_2, s_{(k+1)g_2}) \in E_{g_2}$ **do**
        if $(\mu_1 {}_{\rho_1}\!\bowtie_{\rho_2} \mu_2)$ **then**
          if $(\nexists (s_{kg}, \mu, s_{(k+1)g}) \in E_g : \mu = \mu_1 {}_{\rho_1}\!\sqcup_{\rho_2} \mu_2)$ **then**
            Remove $(s_{kg_1}, \mu_1, s_{(k+1)g_1})$ from $\bar{E}_{g_1}$;
          else
            if
            $(s_{(k+1)g_1} \in \bar{F}_{g_1}) \wedge (s_{(k+1)g_2} \in F_{g_2}) \wedge (s_{(k+1)g} \notin F_g)$
            then
              Remove $s_{(k+1)g_1}$ from $\bar{F}_{g_1}$;
            end
            Add $((s_{(k+1)g_1}, s_{(k+1)g_2}), s_{(k+1)g})$ to R;
          end
        end
      end
    end
  end
end

Algorithm 1: Modifying $\mathcal{G}_1^{\mathfrak{n}}$ so as to satisfy **DC-1**

Lemma 9. $\bar{\mathcal{G}}_1^{\mathfrak{n}} \preceq \mathcal{G}_1^{\mathfrak{n}}$.

PROOF. Straightforward since Algorithm 1 only removes transitions and final states from $\mathcal{G}_1^{\mathfrak{n}}$ and does not add transitions or states to it. □

Lemma 10. $\bar{\mathcal{G}}_1^{\mathfrak{n}} \,{}_{\rho_1}\|_{\rho_2}\, \mathcal{G}_2^{\mathfrak{n}} \preceq \mathcal{G}^{\mathfrak{n}}$.

PROOF. By contraposition, assume that $\bar{\mathcal{G}}_1^{\mathfrak{n}} \,{}_{\rho_1}\|_{\rho_2}\, \mathcal{G}_2^{\mathfrak{n}} \not\preceq \mathcal{G}^{\mathfrak{n}}$ and consider the runs which cause the refinement violation (considering, as in Lemma 3, the shortest such run):

$$\bar{r}_1 : s_{0g_1} \xrightarrow{\mu_{11}} s_{1g_1} \dots \xrightarrow{\mu_{n1}} s_{ng_1}$$
$$r_2 : s_{0g_2} \xrightarrow{\mu_{12}} s_{1g_2} \dots \xrightarrow{\mu_{n2}} s_{ng_2}$$

where $\mu_{k1} \,{}_{\rho_1}\bowtie_{\rho_2} \mu_{k2}$ for $1 \leq k \leq n$. There are two possible cases. In the first case, there exists run $r : s_{0g} \xrightarrow{\mu_1} s_{1g} \dots \not\xrightarrow{\mu_n} s_{ng}$ where $\mu_k = \mu_{k1} \,{}_{\rho_1}\sqcup_{\rho_2} \mu_{k2}$ and the last transition $(s_{(n-1)g}, \mu_n, s_n g)$ is not included in E. This causes a contradiction since performing Algorithm 1 will remove the transition $(s_{(n-1)g_1}, \mu_{n1}, s_{ng_1})$ from $\bar{E}_1$. In the second case, there exists a run $r : s_0 \xrightarrow{\mu_1} s_1 \dots \xrightarrow{\mu_n} s_n$ where the last state s_{ng} is not an accepting state while s_{ng_1} and s_{ng_2} are. This similarly causes a contradiction since performing Algorithm 1 will remove s_{ng_1} from $\bar{F}_{g_1}$. □

Lemma 11. *Algorithm 1 finally terminates in finite time (assuming a finite number of transitions).*

PROOF. Obvious since the number of states $((s_{k1}, s_{k2}), s_k)$ is finite. □

Input: $\bar{\mathcal{G}}_1^{\mathfrak{n}} = (V_1, \mathcal{T}_1, S_{g_1}, s_{0g_1}, \bar{F}_{g_1}, \bar{E}_{g_1})$;
$\mathcal{A}_1 = (V_1, \mathcal{T}_1, S_{a_1}, s_{0a_1}, F_{a_1}, E_{a_1})$;
Output: $\bar{\mathcal{A}}_1 = (V_1, \mathcal{T}_1, \bar{S}_{a_1}, s_{0a_1}, \bar{F}_{a_1}, \bar{E}_{a_1})$ such that $\bar{\mathcal{G}}_1^{\mathfrak{n}}/\bar{\mathcal{A}}_1 \preceq \bar{\mathcal{G}}_1^{\mathfrak{n}}$
$\bar{S}_{a_1} = S_{a_1}, \bar{F}_{a_1} = F_{a_1}, \bar{E}_{a_1} = E_{a_1}$;
$R' = \emptyset, R = \{((s_{0g_1}, s_{0a_1}), s_{0g_1})\}$;
while $(R \neq R')$ **do**
  $R' = R$;
  for *every* $((s_{kg_1}, s_{ka_1}), s_{kg_1}) \in R'$ **do**
    for *every* $(s_{kg_1}, \mu_1, s_{(k+1)g_1}) \in \bar{E}_{g_1}$ **do**
      if $\exists (s_{ka_1}, \mu_1, s_{(k+1)a_1}) \in \bar{E}_{a_1}$ **then**
        Add $((s_{(k+1)g_1}, s_{(k+1)a_1}), s_{(k+1)g_1})$ to R;
        if $s_{(k+1)a_1} \notin \bar{F}_{a_1}$ **then**
          Add $s_{(k+1)a_1}$ to $\bar{F}_{a_1}$;
        end
      else
        Add a new state $s_{(k+1)a_1}$ to $\bar{S}_{a_1}$ and $\bar{F}_{a_1}$;
        Add $(s_{ka_1}, \mu_1, s_{(k+1)a_1})$ to $\bar{E}_{a_1}$;
        Add $((s_{(k+1)g_1}, s_{(k+1)a_1}), s_{(k+1)g_1})$ to R;
      end
    end
    $Q_{g_1} = L(V_1, \mathcal{T}_1) \setminus \{\mu_1 | s_{kg_1} \xrightarrow{\mu_1} \text{ holds}\}$;
    $Q_{a_1} = L(V_1, \mathcal{T}_1) \setminus \{\mu_1 | s_{ka_1} \xrightarrow{\mu_1} \text{ holds}\}$;
    if $(Q_{g_1} \cap Q_{a_1} \neq \emptyset)$ **then**
      Add a new state $s_{(k+1)a_1}$ to $\bar{S}_{a_1}$;
      for *every* $\mu_1 \in (Q_{g_1} \cap Q_{a_1})$ **do**
        Add $(s_{ka_1}, \mu_1, s_{(k+1)a_1})$ to $\bar{E}_{a_1}$;
      end
    end
  end
end

Algorithm 2: Weakening $\mathcal{A}_1$ to $\bar{\mathcal{A}}_1$ so that $\bar{\mathcal{G}}_1^{\mathfrak{n}}/\bar{\mathcal{A}}_1 \preceq \bar{\mathcal{G}}_1^{\mathfrak{n}}$ holds

Since the normalization $\bar{\mathcal{G}}_1^{\mathfrak{n}}/\mathcal{A}_1$ does not always coincide $\bar{\mathcal{G}}_1^{\mathfrak{n}}$, we need to further modify $\mathcal{A}_1$ into $\bar{\mathcal{A}}_1$ so as to make $\bar{\mathcal{G}}_1^{\mathfrak{n}}/\bar{\mathcal{A}}_1 = \bar{\mathcal{G}}_1^{\mathfrak{n}}$. This can be done by following Algorithm 2.

Lemma 12. $\mathcal{A}_1 \preceq \bar{\mathcal{A}}_1$.

Proof. Straightforward since Algorithm 2 only adds more transitions and states to $\mathcal{A}_1$. □

Lemma 13. $\bar{\mathcal{G}}_1^{\mathfrak{n}}/\bar{\mathcal{A}}_1 = \bar{\mathcal{G}}_1^{\mathfrak{n}}$.

Proof. Since $\bar{\mathcal{G}}_1^{\mathfrak{n}} \preceq \bar{\mathcal{G}}_1^{\mathfrak{n}}/\bar{\mathcal{A}}_1$ by Lemma 3 and $\bar{\mathcal{G}}_1^{\mathfrak{n}}/\bar{\mathcal{A}}_1 \preceq \bar{\mathcal{G}}_1^{\mathfrak{n}}$ by construction of Algorithm 2, it follows that $\bar{\mathcal{G}}_1^{\mathfrak{n}}/\bar{\mathcal{A}}_1 = \bar{\mathcal{G}}_1^{\mathfrak{n}}$. To conclude the proof, we show that Algorithm 2 indeed guarantees $\bar{\mathcal{G}}_1^{\mathfrak{n}}/\bar{\mathcal{A}}_1 \preceq \bar{\mathcal{G}}_1^{\mathfrak{n}}$.

By construction of the algorithm, the set R contains the refinement relation between $\bar{\mathcal{G}}_1^{\mathfrak{n}}/\bar{\mathcal{A}}_1$ and $\bar{\mathcal{G}}_1^{\mathfrak{n}}$. The new assumption $\bar{\mathcal{A}}_1$ is constructed so that the universal state of $\bar{\mathcal{G}}_1^{\mathfrak{n}}/\bar{\mathcal{A}}_1$ is not included in the refinement relation. To this end, all behaviors allowed by $\bar{\mathcal{G}}_1^{\mathfrak{n}}$ should be allowed also by $\bar{\mathcal{A}}_1$. In addition, any prefix of these behaviors (which may not be a behavior with an accepting state) should be allowed by $\bar{\mathcal{A}}_1$ as well, i.e., it should be a behavior accepted by $\bar{\mathcal{A}}_1$ (the first part of the first **for** loop containing another **for** loop). The new assumption should also contain behaviors which are neither accepted by $\bar{\mathcal{G}}_1^{\mathfrak{n}}$ nor prefixes of the accepted behaviors of $\bar{\mathcal{G}}_1^{\mathfrak{n}}$ (the second part of the first **for** loop).

The quotient $\bar{\mathcal{G}}_1^{\mathfrak{n}}/\bar{\mathcal{A}}_1$ will then contain only those behaviors accepted by $\bar{\mathcal{G}}_1^{\mathfrak{n}}$, and it refines $\bar{\mathcal{G}}_1^{\mathfrak{n}}$ as a result. □

Finally, composing $\bar{\mathcal{G}}_1^{\mathfrak{n}}$ together with $\bar{\mathcal{A}}_1$ obtains the guarantee $\bar{\mathcal{G}}_1$ which yields exactly $\bar{\mathcal{G}}_1^{\mathfrak{n}}/\bar{\mathcal{A}}_1$ through normalization. In other words, contract $(\bar{\mathcal{A}}_1, \bar{\mathcal{G}}_1)$ is semantically equivalent to contract $(\bar{\mathcal{A}}_1, \bar{\mathcal{G}}_1^{\mathfrak{n}})$. In addition, the former is more compact and convenient than the latter in terms of representation. The following lemma shows that the normalized guarantees of those contracts are behaviorally equivalent, that is they have the same set of behaviors.

Lemma 14. *Let* $\bar{\mathcal{G}}_1 = \bar{\mathcal{G}}_1^{\mathfrak{n}} \parallel \bar{\mathcal{A}}_1$. *Then* $\bar{\mathcal{G}}_1^{\mathfrak{n}}/\bar{\mathcal{A}}_1 = \bar{\mathcal{G}}_1/\bar{\mathcal{A}}_1$.

Proof. We show the mutual refinement between $\bar{\mathcal{G}}_1^{\mathfrak{n}}/\bar{\mathcal{A}}_1$ and $\bar{\mathcal{G}}_1/\bar{\mathcal{A}}_1$.

- $\bar{\mathcal{G}}_1^{\mathfrak{n}}/\bar{\mathcal{A}}_1 \preceq \bar{\mathcal{G}}_1/\bar{\mathcal{A}}_1$:

  First, $\bar{\mathcal{G}}_1^{\mathfrak{n}}/\bar{\mathcal{A}}_1 \preceq \bar{\mathcal{G}}_1^{\mathfrak{n}}$ holds by construction of Algorithm 2.

  Second, $\bar{\mathcal{G}}_1 = \bar{\mathcal{G}}_1^{\mathfrak{n}} \parallel \bar{\mathcal{A}}_1$ by assumption , therefore it holds that $\bar{\mathcal{G}}_1 \preceq \bar{\mathcal{G}}_1$ or equivalently $\bar{\mathcal{G}}_1^{\mathfrak{n}} \parallel \bar{\mathcal{A}}_1 \preceq \bar{\mathcal{G}}_1$. By the Quotient Property 6, $\bar{\mathcal{G}}_1^{\mathfrak{n}} \preceq \bar{\mathcal{G}}_1/\bar{\mathcal{A}}_1$ is true.

  Based on those two facts and the transitivity of refinement, it then follows that $\bar{\mathcal{G}}_1^{\mathfrak{n}}/\bar{\mathcal{A}}_1 \preceq \bar{\mathcal{G}}_1/\bar{\mathcal{A}}_1$.

- $\bar{\mathcal{G}}_1/\bar{\mathcal{A}}_1 \preceq \bar{\mathcal{G}}_1^{\mathfrak{n}}/\bar{\mathcal{A}}_1$:

  By assumption $\bar{\mathcal{G}}_1 = \bar{\mathcal{G}}_1^{\mathfrak{n}} \parallel \bar{\mathcal{A}}_1$ and by definition of composition, we can deduce $\bar{\mathcal{G}}_1 \preceq \bar{\mathcal{G}}_1^{\mathfrak{n}}$. By the Quotient Property 6, $(\bar{\mathcal{G}}_1/\bar{\mathcal{A}}_1) \parallel \bar{\mathcal{A}}_1 \preceq \bar{\mathcal{G}}_1$ is true. Combining with the implication before, it holds that $(\bar{\mathcal{G}}_1/\bar{\mathcal{A}}_1) \parallel \bar{\mathcal{A}}_1 \preceq \bar{\mathcal{G}}_1^{\mathfrak{n}}$. From this, it follows that $\bar{\mathcal{G}}_1/\bar{\mathcal{A}}_1 \preceq \bar{\mathcal{G}}_1^{\mathfrak{n}}/\bar{\mathcal{A}}_1$ by the Quotient Property 6 again. □

It is important to notice that strengthening $\mathcal{G}_1^{\mathfrak{n}}$ as above either maintains the refining property established in **DC-2** or may change it from false to true, but not vice-versa because:

$$\mathcal{A}_1 \preceq \bar{\mathcal{A}}_1,$$
$$\mathcal{A} \,{}_{\mathsf{proj}_1}\|_{\mathsf{id}_1}\, \bar{\mathcal{G}}_1^{\mathfrak{n}} \preceq \mathcal{A} \,{}_{\mathsf{proj}_1}\|_{\mathsf{id}_1}\, \mathcal{G}_1^{\mathfrak{n}}.$$

In order to satisfy **DC-2a** and **DC-2b**, we can strengthen the normalized guarantees by following respectively Algorithm 3 and 4 which are similar to

$\mathcal{G}_1^{\mathfrak{n}} = (V_1, \mathcal{T}_1, S_{g_1}, s_{0g_1}, F_{g_1}, E_{g_1})$;
Input: $\mathcal{A}_2 = (V_2, \mathcal{T}_2, S_{a_2}, s_{0a_2}, F_{a_2}, E_{a_2})$;
$\mathcal{A} = (V, \mathcal{T}_1 \,{}_{\rho_1}\!\times_{\rho_2} \mathcal{T}_2, S_a, s_{0a}, F_a, E_a)$;
Output: $\bar{\mathcal{G}}_1^{\mathfrak{n}} = (V_1, \mathcal{T}_1, S_{g_1}, s_{0g_1}, \bar{F}_{g_1}, \bar{E}_{g_1})$ such that
$(\mathcal{A} \,{}_{\mathsf{proj}_1}\!\|_{\mathsf{id}_1}\, \bar{\mathcal{G}}_1^{\mathfrak{n}}) \,{}_{\mathsf{proj}_2'}\!\preceq_{\mathsf{id}_2} \mathcal{A}_2$
$\bar{F}_{g_1} = F_{g_1}, \bar{E}_{g_1} = E_{g_1}, R' = \emptyset, R = \{((s_{0a}, s_{0g_1}), s_{0a_2})\}$;
while $(R \neq R')$ **do**
  $R' = R$;
  for *every* $((s_{ka}, s_{kg_1}), s_{ka_2}) \in R'$ **do**
    for *every* $(s_{ka}, \mu, s_{(k+1)a}) \in E_a$ **do**
      for *every* $(s_{kg_1}, \mu_1, s_{(k+1)g_1}) \in \bar{E}_{g_1}$ **do**
        if $(\mu \,{}_{\mathsf{proj}_1}\!\bowtie_{\mathsf{id}_1} \mu_1)$ **then**
          if $(\nexists (s_{ka_2}, \mu_2, s_{(k+1)a_2}) \in E_{a_2} : \mu = \mu_1 \,{}_{\rho_1}\!\sqcup_{\rho_2} \mu_2)$ **then**
            Remove $(s_{kg_1}, \mu_1, s_{(k+1)g_1})$ from $\bar{E}_{g_1}$;
          else
            Add $((s_{(k+1)a}, s_{(k+1)g_1}), s_{(k+1)a_2})$ to R;
          end
        end
      end
    end
    if $(s_{ka} \in F_a) \wedge (s_{(kg_1} \in \bar{F}_{g_1}) \wedge (s_{ka_2} \notin F_{a_2})$ **then**
      Remove s_{kg_1} from $\bar{F}_{g_1}$;
    end
  end
end

Algorithm 3: Refining $\mathcal{G}_1^{\mathfrak{n}}$ so as to satisfy **DC-2a**

$\mathcal{G}_2^{\mathfrak{n}} = (V_2, \mathcal{T}_2, S_{g_2}, s_{0g_2}, F_{g_2}, E_{g_2})$;
Input: $\mathcal{A}_1 = (V_1, \mathcal{T}_1, S_{a_1}, s_{0a_1}, F_{a_1}, E_{a_1})$;
$\mathcal{A} = (V, \mathcal{T}_1 \,{}_{\rho_1}\!\times_{\rho_2} \mathcal{T}_2, S_a, s_{0a}, F_a, E_a)$;
Output: $\bar{\mathcal{G}}_2^{\mathfrak{n}} = (V_2, \mathcal{T}_2, S_{g_2}, s_{0g_2}, \bar{F}_{g_2}, \bar{E}_{g_2})$ such that
$(\mathcal{A} \,{}_{\mathsf{proj}_2}\!\|_{\mathsf{id}_2}\, \bar{\mathcal{G}}_2^{\mathfrak{n}}) \,{}_{\mathsf{proj}_1'}\!\preceq_{\mathsf{id}_1} \mathcal{A}_1$
$\bar{F}_{g_2} = F_{g_2}, \bar{E}_{g_2} = E_{g_2}, R' = \emptyset, R = \{((s_{0a}, s_{0g_2}), s_{0a_1})\}$;
while $(R \neq R')$ **do**
  $R' = R$;
  for *every* $((s_{ka}, s_{kg_2}), s_{ka_1}) \in R'$ **do**
    for *every* $(s_{ka}, \mu, s_{(k+1)a}) \in E_a$ **do**
      for *every* $(s_{kg_2}, \mu_2, s_{(k+1)g_2}) \in \bar{E}_{g_2}$ **do**
        if $(\mu \,{}_{\mathsf{proj}_2}\!\bowtie_{\mathsf{id}_2} \mu_2)$ **then**
          if $(\nexists (s_{ka_1}, \mu_1, s_{(k+1)a_1}) \in E_{a_1} : \mu = \mu_1 \,{}_{\rho_1}\!\sqcup_{\rho_2} \mu_2)$ **then**
            Remove $(s_{kg_2}, \mu_2, s_{(k+1)g_2})$ from $\bar{E}_{g_2}$;
          else
            Add $((s_{(k+1)a}, s_{(k+1)g_2}), s_{(k+1)a_1})$ to R;
          end
        end
      end
    end
    if $(s_{ka} \in F_a) \wedge (s_{kg_2} \in \bar{F}_{g_2}) \wedge (s_{ka_1} \notin F_{a_1})$ **then**
      Remove s_{kg_2} from $\bar{F}_{g_2}$;
    end
  end
end

Algorithm 4: Refining $\mathcal{G}_2^{\mathfrak{n}}$ so as to satisfy **DC-2b**

Algorithm 1. We then invoke Algorithm 2 to weaken also the associated assumptions.

The following lemma shows the satisfaction of **DC-2a** after using Algorithm 3 and of **DC-2b** after using Algorithm 4.

Lemma 15. $(\mathcal{A} \,{}_{\mathsf{proj}_1}\|_{\mathsf{id}_1}\, \bar{\mathcal{G}}_1^{\mathfrak{n}}) \,{}_{\mathsf{proj}_2'}\preceq_{\mathsf{id}_2} \mathcal{A}_2$ *and* $(\mathcal{A} \,{}_{\mathsf{proj}_2}\|_{\mathsf{id}_2}\, \bar{\mathcal{G}}_2^{\mathfrak{n}}) \,{}_{\mathsf{proj}_1'}\preceq_{\mathsf{id}_1} \mathcal{A}_1$

Proof. Since the two algorithms 3 and 4 are similar, we prove the satisfaction of **DC-2a** and deduce the satisfaction for the other **DC-2b**.

By construction of the algorithm, the set R contains the refinement relation between $(\mathcal{A} \,{}_{\mathsf{proj}_1}\|_{\mathsf{id}_1}\, \bar{\mathcal{G}}_1^{\mathfrak{n}})$ and $\mathcal{A}_2$ w.r.t. morphisms proj_2' and id_2.

Assume, by contradiction, that $(\mathcal{A} \,{}_{\mathsf{proj}_1}\|_{\mathsf{id}_1}\, \bar{\mathcal{G}}_1^{\mathfrak{n}}) \,{}_{\mathsf{proj}_2'}\not\preceq_{\mathsf{id}_2} \mathcal{A}_2$. Then, their refinement relation contains some state $((s_{ka}, s_{kg_1}), s_{ka_2})$ where two cases can happen. In one case, there exists a transition at (s_{ka}, s_{kg_1}) with a label $\mu' = \mu_{\mathsf{proj}_1}\sqcup_{\mathsf{id}_1}$ for which there is no label μ_2 at s_{ka_2} to synchronize. In the other case, (s_{ka}, s_{kg_1}) is a final state while s_{ka_2} is not. However, in the first case, such a transition would be pruned by the second **if**. In the second case, the final state would be pruned by the third **if**.

As a result, $(\mathcal{A} \,{}_{\mathsf{proj}_1}\|_{\mathsf{id}_1}\, \bar{\mathcal{G}}_1^{\mathfrak{n}}) \,{}_{\mathsf{proj}_2'}\preceq_{\mathsf{id}_2} \mathcal{A}_2$ is true. □

Synthesis Strategy

Based on the above analysis, we propose a strategy for synthesizing the composition $\mathcal{C}_1 \,{}_{\rho_1}\|_{\rho_2}\, \mathcal{C}_2$ so that it can refine $\mathcal{C}$ as follows:

1. Apply Algorithm 1 so that **DC-1** is satisfied.
2. Apply Algorithm 3 so that **DC-2a** is satisfied and apply Algorithm 2 to weaken $\mathcal{A}_1$.
3. Apply Algorithm 4 so that **DC-2b** is satisfied and apply Algorithm 2 to weaken $\mathcal{A}_2$.

Our strategy to synthesize contracts proposed above is *heuristic* and not *optimal* in general, since it removes all transitions and states that it suspects to be the cause of the condition violation. It is however sufficiently simple for a prototype implementation of the synthesis.

Example 21. We consider again the simplified water controlling system in Example 12 as shown in Fig. 2 and Fig. 3.

We consider the specification $\mathcal{C} = (\mathcal{A}, \mathcal{G})$ where it makes no assumptions, i.e. μ_0 denotes the universe set of labels $L(V, \mathcal{T}_1 \,{}_{\rho_1}\times_{\rho_2}\, \mathcal{T}_2)$ and ensures timely control over the water evolution as shown in Figure 6. It is easy to verify that the guarantees of the two contracts $\mathcal{C}$ and $\mathcal{C}_t$ remain intact through the normalization operation. This is because the contracts accept all assumptions made to their variables. Meanwhile, the controller normalized guarantee specifies more behaviors than its un-normalized version as the controller does have some assumptions on its input. Composing $\mathcal{C}_t$ and $\mathcal{C}_c$ under morphisms ρ_1 and ρ_2,

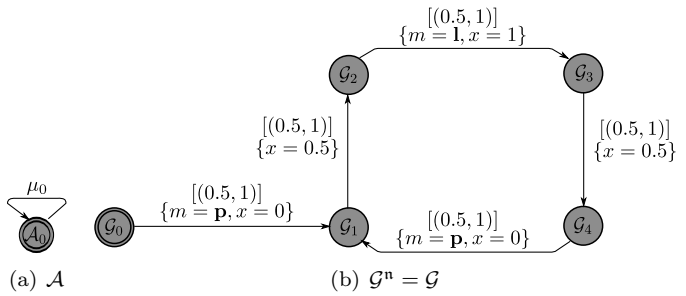


Figure 6: The desirable water control behavior

however, does not satisfy contract $\mathcal{C}$. This is because $\mathcal{G}_t^{\mathfrak{n}} \parallel_{\rho_1}\parallel_{\rho_2} \mathcal{G}_c^{\mathfrak{n}} \not\preceq \mathcal{G}^{\mathfrak{n}}$ which in turn is caused by the fact that both the tank and controller guarantees allow the water to be filled into the tank without issuing any Open command. Applying Algorithm 1 to synthesize the controller guarantee, the transitions labeled with μ_{2c} are removed as shown in Fig. 7(a). The controller assumption need not be weakened since $\bar{\mathcal{G}}_c^{\mathfrak{n}} / \mathcal{A}_c = \bar{\mathcal{G}}_c^{\mathfrak{n}}$ and a simpler un-normalized version of $\bar{\mathcal{G}}_c^{\mathfrak{n}}$ can then be computed as in Fig. 7(b). The new composition of the tank and controller contracts can now satisfy the desirable specification $\mathcal{C}$.

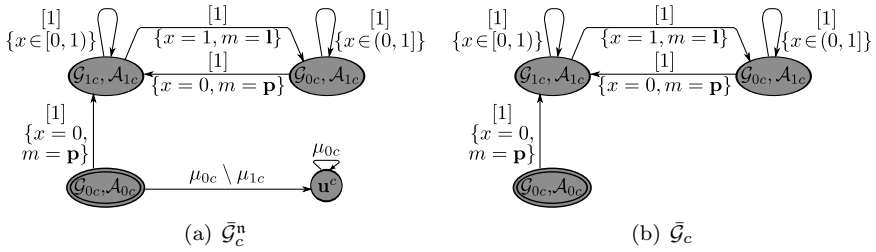


Figure 7: Controller synthesis

6. 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 [9, 23] 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, refinement and compatibility.

We have also shown how to extend a contract-based methodology for modeling heterogeneous components so that it can synthesize the component models to satisfy a predefined composition requirement. The decomposition conditions that we have identified for checking such satisfaction are based on finding a heterogeneous refinement between a specification and a composition of two other heterogeneous specifications. Although it is a natural approach for one to perform heterogeneous quotient and projection in order to provide for the satisfaction of these conditions, it does not work in many cases where the tag morphisms are many-to-one morphisms. Our synthesis approach instead is based on pruning away undesirable behaviors through traversing the transitions of the specification TM. Therefore our approach does not suffer from the effects of many-to-one morphisms and can provide a synthesis strategy for fixing wrong contract decompositions. Our strategy is heuristic, however, as it might generate decompositions which are more constrained than necessary. In addition, the strategy is limited to manipulating the existing contracts, and does not provide solutions with an alternative structure. Nonetheless, given several alternative structures as starting points, our strategy is able to provide correct implementations.

Our future work includes the implementation of the contract framework and its synthesis strategy. Validating the implementation as well as verification performance through cases studies will also be a next essential step.

References

[1] B. Meyer, Applying "Design by contract", Computer 25 (10) (1992) 40–51.

[2] S. Bliudze, J. Sifakis, The algebra of connectors: Structuring interaction in BIP, IEEE Transactions on Computers 57 (10) (2008) 1315–1330.

[3] X. Liu, Y. Xiong, E. A. Lee, The Ptolemy II framework for visual languages, in: Proceedings of the IEEE 2001 Symposia on Human Centric Computing Languages and Environments (HCC'01), HCC '01, IEEE Computer Society, Washington, DC, USA, 2001, pp. 50–.

[4] E. Lee, A. Sangiovanni-Vincentelli, A framework for comparing models of computation, IEEE Trans. CAD of Integ. Circ. and Systems 17 (12) (1998) 1217–1229.

[5] A. Benveniste, B. Caillaud, L. P. Carloni, P. Caspi, A. L. Sangiovanni-Vincentelli, Composing heterogeneous reactive systems, ACM Trans. Embed. Comput. Syst. 7 (4) (2008) 43:1–43:36.

[6] A. Benveniste, B. Caillaud, A. Ferrari, L. Mangeruca, R. Passerone, C. Sofronis, Multiple viewpoint contract-based specification and design, in: Formal Methods for Components and Objects, 6^{th} International Symposium, Amsterdam, LNCS, Springer, 2008, pp. 200–225.

[7] W. Damm, H. Hungar, B. Josko, T. Peikenkamp, I. Stierand, 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, 2011, pp. 1–6.

[8] T. T. H. Le, R. Passerone, U. Fahrenberg, A. Legay, A tag contract framework for heterogeneous systems, in: C. Canal, M. Villari (Eds.), Advances in Service-Oriented and Cloud Computing, Vol. 393 of Communications in Computer and Information Science, Springer Berlin Heidelberg, 2013, pp. 204–217.

[9] T. T. H. Le, R. Passerone, U. Fahrenberg, A. Legay, Tag machines for modeling heterogeneous systems, in: Proceedings of the 13^{th} International Conference on Application of Concurrency to System Design, ACSD13, Barcelona, Spain, 2013, pp. 186–195.

[10] E. W. Dijkstra, Guarded commands, non-determinancy and a calculus for the derivation of programs, in: Language Hierarchies and Interfaces, Springer, 1975, pp. 111–124.

[11] L. Lamport, win and sin: Predicate transformers for concurrency, ACM Trans. Program. Lang. Syst. 12 (3) (1990) 396–428.

[12] L. de Alfaro, T. A. Henzinger, Interface automata, SIGSOFT Softw. Eng. Notes 26 (2001) 109–120.

[13] L. Benvenuti, A. Ferrari, L. Mangeruca, E. Mazzi, R. Passerone, C. Sofronis, A contract-based formalism for the specification of heterogeneous systems, in: Proceedings of the Forum on Specification, Verification and Design Languages, Stuttgart, 2008, pp. 142–147.

[14] P. López Martínez, T. Vardanega, Handling synchronization requirements under separation of concerns in model-driven component-based development, in: Reliable Software Technologies Ada-Europe 2012, Vol. 7308 of LNCS, Springer Berlin Heidelberg, 2012, pp. 89–104.

[15] S. S. Bauer, A. David, R. Hennicker, K. G. Larsen, A. Legay, U. Nyman, A. Wasowski, Moving from specifications to contracts in component-based design, in: FASE, Vol. 7212, Springer, 2012, pp. 43–58.

[16] J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, A. Legay, R. Passerone, A modal interface theory for component-based design, Fundamenta Informaticae 108 (1–2) (2011) 119–149. doi:10.3233/FI-2011-416.

[17] W.-P. de Roever, The quest for compositionality—a survey of assertion-based proof systems for concurrent programs, part i: Concurrency based on shared variables, in: Proc. of the IFIP Working Conference "The role of abstract models in computer science", 1985.

[18] S.-W. Lin, P.-A. Hsiung, Counterexample-guided assume-guarantee synthesis through learning, IEEE Transactions on Computers 60 (5) (2011) 734–750.

[19] K. Chatterjee, T. Henzinger, Assume-guarantee synthesis, in: Tools and Algorithms for the Construction and Analysis of Systems, Vol. 4424 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2007, pp. 261–275.

[20] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, Y. Xiong, Taming heterogeneity - the ptolemy approach, in: Proceedings of the IEEE, 2003, pp. 127–144.

[21] A. Davare, D. Densmore, L. Guo, R. Passerone, A. L. Sangiovanni-Vincentelli, A. Simalatsar, Q. Zhu, METROII: A design environment for cyber-physical systems, ACM Transactions on Embedded Computing Systems 12 (1s) (2013) 49:1–49:31.

[22] L. Guo, Q. Zhu, P. Nuzzo, R. Passerone, A. L. Sangiovanni-Vincentelli, E. A. Lee, Metronomy: a function-architecture co-simulation framework for timing verification of cyber-physical systems, in: Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis, CODES14, New Delhi, India, 2014, pp. 24:1–24:10.

[23] A. Benveniste, B. Caillaud, L. P. Carloni, A. Sangiovanni-Vincentelli, Tag machines, in: Proceedings of the International Conference On Embedded Software, ACM, 2005, pp. 255–263.

[24] S. Dey, D. Sarkar, A. Basu, A tag machine based performance evaluation method for job-shop schedules, IEEE Trans. CAD of Integ. Circ. and Systems 29 (7) (2010) 1028–1041.

[25] S. Dey, D. Sarkar, A. Basu, A Kleene algebra of tagged system actors, Embedded Systems Letters, IEEE 3 (1) (2011) 28 –31.

[26] F. Arbab, J. Rutten, A coinductive calculus of component connectors, in: M. Wirsing, D. Pattinson, R. Hennicker (Eds.), Recent Trends in Algebraic Development Techniques, Vol. 2755 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2003, pp. 34–55.

[27] U. Fahrenberg, A. Legay, A. Wasowski, Make a difference! (semantically), in: Proceedings of the 14th International Conference on Model Driven Engineering Languages and Systems, Springer, Wellington, New Zealand, 2011, pp. 490–500.

[28] R. Passerone, J. R. Burch, A. L. Sangiovanni-Vincentelli, Refinement preserving approximations for the design and verification of heterogeneous systems, Formal Methods in System Design 31 (1) (2007) 1–33.