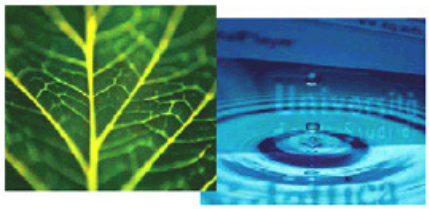
PhD Dissertation



International Doctorate School in Information and Communication Technologies

DIT - University of Trento

A Tag Contract Framework for Modeling Heterogeneous Systems

Thi Thieu Hoa Le

Advisor:

Professor Roberto Passerone

Università degli Studi di Trento

February 2014

Abstract

In the distributed development of modern IT systems, contracts play a vital role in ensuring interoperability of components and adherence to specifications. The design of embedded systems, however, is made more complex by the heterogeneous nature of components, which are often described using different models and interaction mechanisms. Composing such components is generally not well-defined, making design and verification difficult. Several denotational frameworks have been proposed to handle heterogeneity using a variety of approaches. However, the application of heterogeneous modeling frameworks to contract-based design has not yet been investigated. In this work, we develop an operational model with precise heterogeneous denotational semantics, based on tag machines, that can represent heterogeneous composition, and provide conditions under which composition can be captured soundly and completely. The operational framework is implemented in a prototype tool which we use for experimental evaluation. We then construct a full contract model and 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 generic algebraic method to synthesize or refine a set of contracts so that their composition satisfies a given contract.

Keywords

[Tag Machines, Contract Theory, Heterogeneity, Synthesis]

To my husband and family

ACKNOWLEDGEMENTS

This dissertation would not have been completed without the support of many individuals and institutions.

I would like to express my utmost gratitude to my advisor, Professor Roberto Passerone, for his guidance and support through years of my study. His expertise, advice and time have proven to be indispensable in my academic and life journey, especially in moments when I feel lost of my research track. His encouragement has also been a great motivation for me to go on with this research work.

I have had several great supporters at the research institution INRIA/IRISA in Rennes, France ever since I did my internship there. Professor Axel Legay and Doctor Uli Fahrenberg were my supervisors while I was an intern at INRIA. This research work was partially inspired by them and I owe them a special thanks for their elaborated discussion and fruitful collaboration which is spread throughout this thesis work.

The faculty and staff at University of Trento have all provided me with much help and support for which I am thankful. I would also like to thank all the members of the EECS lab with whom I have shared research ideas as well as difficulties in research life.

Finally, I owe the greatest thanks to my caring and patient husband whose unconditional love and support has always kept me on task and been my true motivation to complete this work.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 The Context

Modern computing systems are increasingly being built by composing components which can be developed concurrently by different design teams. In such a development paradigm, the distinction between what is constrained on environments, and what must be guaranteed by a system given the constraint satisfaction, reflects the different roles and responsibilities in the system design procedure. Such distinction can be captured by a component model called *contract* [42]. Formally, a contract ($\mathcal{C}$) is a pair of *assumptions* ($\mathcal{A}$) and *guarantees* ($\mathcal{G}$) (i.e. $\mathcal{C} = (\mathcal{A}, \mathcal{G})$) which intuitively are properties that must be satisfied by all inputs and outputs of a design, respectively. The separation between assumptions and guarantees supports the distributed development of complex systems and allows subsystems to synchronize by relying on associated contracts.

In the particular context of embedded systems, *heterogeneity* is a typical characteristic since these systems are usually composed from parts developed using different methods, time models and interaction mechanisms. Such heterogeneity usually appears across different layers of abstraction in the design flow, making the evaluation of whether certain properties passed from the higher level of abstraction are maintained at the lower

level become extremely difficult. It has been often the case that heterogeneous compositional mechanism is not sufficiently well-defined to enable the verification of some system property from the known properties of its components. To deal with heterogeneity, several modeling frameworks have been proposed oriented towards the representation and simulation of heterogeneous systems, such as the Ptolemy framework [39], or towards the unification of their interaction paradigms, such as those based on *tagged events* [37]. The former is geared towards the representation and simulation of heterogeneous systems while the latter can capture different notions of time and interaction paradigms, including physical time, logical time (synchronous and asynchronous), precedence relations, etc., 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 [7, 19]. However, the notion of heterogeneity in general is much broader than that between multiple viewpoints, and must take into account diverse interaction paradigms. Meanwhile, heterogeneous modeling frameworks have not been related to contract-based design flows. This has motivated us to study a methodology which allows heterogeneous systems to be modeled and interconnected in a contract-based fashion.

The central issues when studying such a methodology includes *refinement, composition* and *compatibility* between contracts in order to enable a formalized and rigorous design process for heterogeneous systems. Besides, it is often desirable to study how to fix individual contracts so as to make their composition satisfies or refines an abstract specification repre-

[1] www.speeds.eu.com

sented as another contract. This is an instance of the well-known classical *synthesis* problems:

"Can we construct a model that satisfies some given specification?".

This problem is very popular when designing systems in a top-down decomposing fashion because the overall contract's decomposition into sub-contracts is not always satisfactory.

1.2 Thesis Contributions

Our long term objective is to develop a modeling and analysis framework for the specification and verification of both heterogeneous components and contracts.

1.2.1 A Sound Underlying Representation

As a start, we have modeled a simplified version of a distributed Heterogeneous Communication System (HCS) such as one that one that could be found on board of air-crafts [27], using timed automata [1] augmented with parameters. Different components of a HCS system including server, communication network and devices are modelled as timed automata which allows us to compose them together and reason on their composite behavior. Since no heterogeneous machinery for composing different components has been available as will be discussed in Section 2.1, assembling the components of HCS is done homogeneously. The case study has provided us with a valuable understanding regarding how time can be captured in different models of time such as Uppaal [31], NuSMV [16], HyDI [17] and regarding the complexity of modeling various components through a homogeneous machinery such as timed automata.

With this understanding, the framework that we aim at developing should be able to support formal correctness proofs as obtained in the HCS case study. To this end it must employ an underlying (or intermediate) semantically sound model that can be used to represent different computation and interaction paradigms uniformly. Because simulation is an essential design activity, the model must also be executable. At the same time, the semantic model must be able to retain the individual features of each paradigm to avoid losing their specific properties. In particular, the framework must interact with the user through a front end that exposes familiar models that feel native and natural. In this work, we focus on the intermediate semantic model and defer the discussion on how specific front ends may be constructed to our future work. For this purpose, we advocate the use of Tag Machines (TMs) as a suitable semantic model for system specification. We have chosen to use this formalism for our work, as it provides an operational representation based on rigorous and proven semantics. Tag Machines can be used to represent *homogeneous* systems [6] and to achieve our goal, we extend TMs to encompass the *heterogeneous* context. In particular, we study the relation between composition of TMs with that of their denotational semantics. We first review and correct certain aspects of TMs, and provide conditions under which the operational model can fully and compositionally capture the denotational representation. We have also developed a simulation engine that supports heterogeneous TMs, with which we experimentally evaluate our results on a significant case study.

1.2.2 Heterogeneous Contract-based Design Methodology

Our second objective is to develop a methodology for modeling heterogeneous systems in a contract-based fashion. In this goal, we build a *contract* model on top of heterogeneous TMs and define a full set of operations and

relations between contracts such as satisfaction, refinement, composition and compatibility.

To achieve such goal, we rely on a generic meta-framework [4] that we extend with tags and mappings between tags to define model interactions. In particular, we study the contract synthesis capability in the *homogeneous* and *heterogeneous* contexts. For homogeneous contracts, we propose *decomposing conditions* for a set of contracts $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ under which the contract decomposition can be verified, and thereby proposing a generic *synthesis strategy* for fixing wrong decompositions. For heterogeneous contracts, we limit the size of contract set to two in order to make the synthesis procedure manageable and simple.

1.3 Structure of the Thesis

The rest of the thesis is organized as follows.

- In Chapter 2, we review the state of the art with respect to the evolution of theories of heterogeneous composition as well as that of theories of interface and contract.

- In Chapter 3, we present a summary of our preliminary investigation on modeling a distributed heterogeneous systems.

- In Chapter 4, we recall notions of tags, behaviors, denotational tag systems and their composition.

- In Chapter 5, we first describe how TMs are extended to represent heterogeneous systems and then discuss soundness and completeness of the TM composition. We also demonstrate the application of our prototype tool to an automotive use case in this chapter.

- In Chapter 6, we present our tag contract framework for modeling heterogeneous systems built on top of TM operations such as composition, quotient, conjunction and refinement. Also in this chapter, we discuss an application of our methodology to a simplified water control problem and model it using incrementing TMs. The material of Chapter 5 and Chapter 6 is mostly taken from [36, 35].
- In Chapter 7, we show how to synthesize a contract set in order to make their composition refine an overall contract when necessary in both homogeneous and heterogeneous contexts.
- In Chapter 8, we summarize our contribution and outline possible directions for future work.

Chapter 2

State of the Art

2.1 Theory of Heterogeneous Composition

Heterogeneity theory has been evolving actively 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 work we are mostly concerned with these lower level aspects.

One such approach is the pioneering framework of Ptolemy II [39], 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 [20] relaxes this limitation, and allows designers to build model adapters directly. 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.

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 [9]. Their relationship, however, can not be easily altered, and the framework lacks a native notion of time. Benveniste et al. [5] propose a heterogeneous denotational semantics inspired by the Lee and Sangiovanni-Vincentelli (LSV) formalism of tag signal models [37], which has been long advocated as a unified modeling framework capable of capturing heterogeneous MoCC. Starting from the LSV model, the authors have derived their preferred variation of tag system model where a system is modeled as a set of behaviors. Each behavior is modeled as a set of signals which are sequences of events and each event is characterized by a data value and a tag. 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 and homogeneous systems share the same tag structure while heterogeneous systems have different tag structures. Composing such systems is thus done by applying mappings between different tag structures.

Tag Machines [6] are subsequently introduced as finite representations

of homogeneous tag systems. They are quite expressive, and ways to map traditional interaction paradigms have been reported in the literature [6]. They have also been applied to model a job-shop specification [23] such that the composite tag machine represents the overall job-shop specification and any trace of the machine from the start to the final state results in a valid job-shop schedule. For the purpose of studying the asymptotic throughput of an infinite job-shop schedule, the authors have proposed a new tag structure to capture the aspect of performance evaluation and an algorithm for evaluating the throughput of job-shop schedules based on tag machine. The algorithm has also been applied to an SDFG model of periodic self-timed executions and a heterogeneous system composed of a dataflow component and a discrete-event component.

Alternatively, tag systems can be represented by functional actors forming a Kleene algebra [24]. The approach is similar to that of Ptolemy II in that both use actors to represent basic components.

2.2 Theory of Interface and Contract

The notion of contract was first introduced by Bertrand Meyer in his design-by-contract method [42], based on ideas by Dijkstra [25], Lamport [30], and others, where systems are viewed as abstract boxes achieving their common goal by verifying specified contracts. Such a technique essentially 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. The class itself can have invariants that must be true at all states of the class and in order to offer safe substitutability, a subclass is only allowed to weaken the pre-conditions and strengthen the post-conditions. Design-by-contract has then been adopted in component-based applications such as [13, 26]. In those approaches, work-flows and

activities are specified for a designer to follow in order to obtain complete component specifications which include the component interface, the inter-component collaboration and a set of contracts in forms of pre-conditions, post-conditions and invariants that apply to the component. The implementation patterns for pre-conditions, post-conditions and invariants were subsequently formalized to automatically generate component skeletons that already implemented such constraints [18].

To allow effective reuse of components in component-based design flows, De Alfaro and Henzinger introduced a light-weight formalism based on automata to document the component specification, called *interface automata* [21]. 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 so as to capture the temporal Input/Output (I/O) behaviours of a component. The I/O actions are expressed by transitions labelled with a "?"/"!" correspondingly. Although being syntactically similar to I/O automata proposed by Lynch [40], interface automata are not necessarily input-enabled and at each of its states, some inputs may not be accepted. By this, interface automata express the assumption that the environment may never generates those inputs and thus input actions are under the environment control. Meanwhile, all outputs are controlled by the component itself, hence are under the component responsibility. Although the assumptions and guarantees are not handled explicitly, interface automata do capture the different roles and responsibilities of a component and its environment. The central issues when introducing the formalism of interface automata are *compatibility*, *composition* and *refinement*. The authors highlight the issue of checking compatible interfaces from two views: pessimistic versus optimistic, and advocate the latter view in which two component interfaces are compatible if they can work together in some environment. Under the

optimistic view, the composition of two interface automata is obtained by restricting the product automaton to the set of compatible states from which there is some environment that can prevent going to error states. Then based on alternating simulation [2], the authors formalize the relation between an interface specification and its implementation by means of refinement, stating that an interface refines another if it has weaker input assumptions and stronger output guarantees. This definition allows a component $\mathcal{P}$ to always be replaced with a more refined version $\mathcal{Q}$ provided that they are connected to the environment by the same inputs. An important connection between refinement and compatibility, which captures also the essence of component-based design, is also exposed through this definition. That is the designer of the environment needs to ensure only compatibility with the component specification $\mathcal{P}$ which subsequently guarantees compatibility with the component implementation $\mathcal{Q}$.

The alternating refinement, in fact, has a drawback when it fails to enforce that the implementation does any useful activities at all. Larsen et.al.'s subsequent introduction of modality into the interface theory [32] helps to rule out such a trivial implementation since as long as some transition in the specification automata is associated with a *must* modality, it must appear in any implementation. In the modal context, modal refinement requires that the specification can mimic all allowed steps (marked with a *may* modality $\Diamond$) made by an implementation and an implementation needs to match all required steps (marked with a *must* modality $\Box$) made by the specification. The authors then show that the alternating refinement actually coincides with the modal refinement if all output transitions are assigned with $\Diamond$, inputs with $\Box$ and the *may* transition relation is made input-enabled. They further define the composition operator for modal interface similarly to that of interface automata. However, Raclet et.al. [45] has recently proved that the operator is indeed incorrect because

it is not monotonic with respect to modal refinement as claimed, thereby failing to ensure that two compatible interfaces may be implemented separately (call independent implementability in [32]). A correction has also been proposed by the authors, resulting in the notion of *relaxed* composition. Such notion relaxes all constraints on the future of the runs that drive the composition to a state where one interface may produce an output that may not be accepted as input by the other. The relaxed composition refers to such state as a "universal" state, meaning every action is assigned with a *may* modality.

Another core contribution made by Raclet et.al. [45] is the unification of two theories: interface automata [21] and modal specification [33] into a new theory addressing also the problem of dissimilar alphabets which was missing in previous work. It is worth noting that Larsen el.al.'s modal interfaces [32] can be viewed as a modal specification except for the modal composition operator and the occurrence of Input/Output distinction.

The contract theory has been evolving in parallel with the interface theory. Researchers from the SPEEDS project have attempted to use a set of constraints (i.e. pairs of Assumption/Guarantee), to describe the expected behaviour of a component (i.e. a set of traces or runs) [7]. The differentiation between assumptions and guarantees, which is implicit in interface automata or modal specification, is made explicit in the trace-based contract framework of the SPEEDS HRC model [7, 8]. Relevant notions such as *composability*, *compatibility* and *dominance* are formalized for contracts. Composability is a purely syntactic criterion on component profiles which consists of uncontrolled and controlled ports, and compatibility is defined as the receptiveness of the composite assumption with respect to the composite ports under the component's control. That is for any sequence of values on the controlled ports, there exists some environment accepting it. The notion of refinement between contracts is referred to as

dominance to distinguish it from the refinement between implementations of the contracts, following the usual scheme of weakening the assumptions and strengthening the guarantees.

The relationship between specifications of component behaviors and contracts is further studied by Bauer et al. [4] 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. The mentioned trace-based contract theories [7, 8] are also demonstrated to be instances of such framework. We take advantage of this formalization in this work to construct our tag contract theory. In addition, this formalization enables verifying if a contract can be decomposed into two other contracts by checking if that contract can *dominate* the others. Therefore we make a further advantage of such dominating notion and generalize it to a set of n homogeneous contracts and construct generic decomposing conditions for the homogeneous contract set.

The verification problem of decomposing a contract into a set of contracts was also studied by Cimatti et al. [15] and was addressed by property-based proof systems with SMT-based model checking techniques. The contract specifications allowed in such systems, however, are trace-based only. Our decomposing conditions can instead deal with generic contract specifications including both trace-based and modal ones.

Assume-guarantee reasoning has also been applied extensively in declarative compositional reasoning [22] to help prove properties by decomposing the process into simpler and more manageable steps. Our objective is conceptually different: assumptions specify a set of legal environments and are used to prove (or disprove) contract compatibility and satisfaction. In contrast, classical assume-guarantee reasoning 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 trans-

formation 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 [38]. However, this synthesis strategy is only applicable for 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 [12]. The synthesized model was shown to be a subset of that synthesized by counterexample-based synthesis [38]. Unlike these concrete notions of synthesis, ours is more generic since it is not tied to the system semantics. Moreover, while the application of our synthesis strategy to generic contract-based systems is direct and straightforward, the generalization of the previous approaches has not been studied and would require a conversion process from normalized contracts to un-normalized ones.

Chapter 3

Modeling and Verification of a Distributed HCS

In this section, we present a summary of our preliminary work on modelling a distributed real-time system and refer to our technical paper [34] for the comprehensive reading.

3.1 HCS Description

We have taken as a case study a simplified version of an HCS system which consists of a common server and many devices communicating through Network Access Controllers (NACs) as shown in Figure 3.1.

In the case study, we focus on audio devices which are required to distribute music and audio announcements to the main cabin. In addition, the devices must reproduce the audio at synchronized instants, hence the importance of the server-device clock synchronisation and their implementation of the Precision Time Protocol (PTP) [44]. The audio stream is transmitted through the network. Each audio packet is sent by the server every *audPeriod* (timeunits or tus) and characterized by the time it has to be played at the device t_{play}. The transmission priority of PTP messages are assumed to be higher than that of audio packets, however, an ongoing

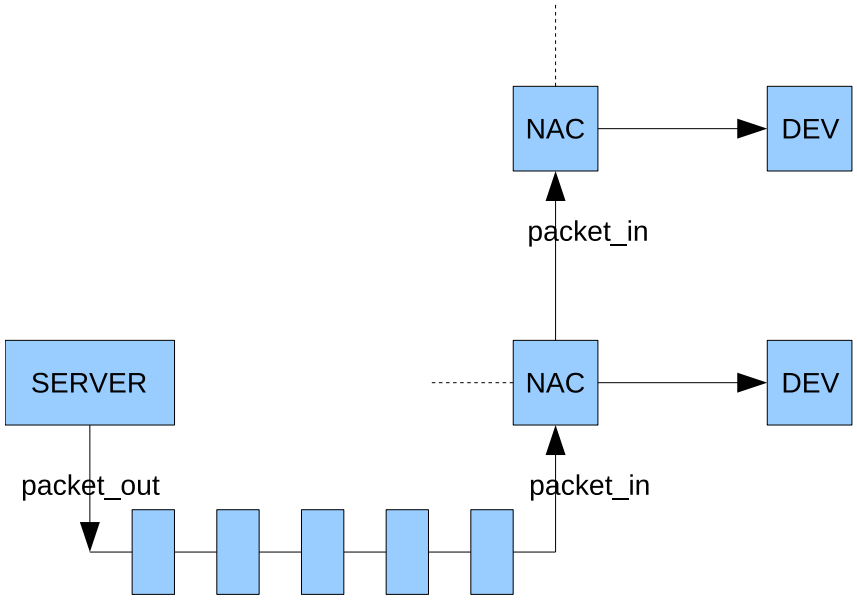


Figure 3.1: Heterogeneous Communication System

transmission of an audio packet will not be preempted by a PTP message. The packet streams can be shown logically as in Figure 3.2.

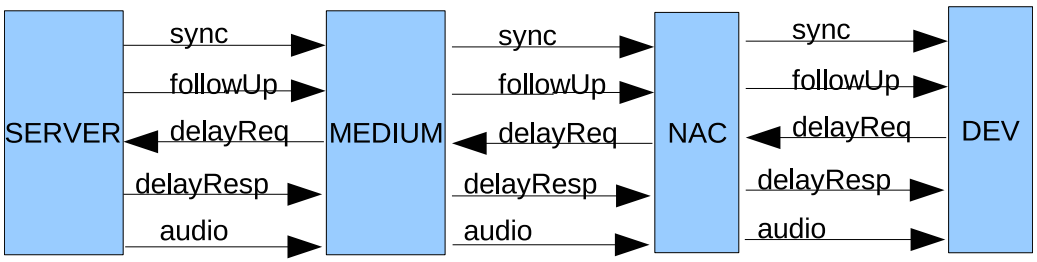


Figure 3.2: Packet streams in HCS

An PTP (Audio) packet transmitted through the network medium incurs a transmission delay $C_1(C_2)$. These two quantities can be considered as design parameters and are related to the packet size and to the channel bitrate.

The model of HCS resembles a contract-based model. As long as the assumption is respected (i.e. the parameter setting of C_1 and C_2 lies within the feasible regions), the correct functioning of packet (audio, PTP)

streaming and clock synchronization (i.e. that the system will not encounter any Error state) can be guaranteed. In this case study, we are interested in computing and representing the assumption on HCS environment, or in other words, verifying whether there are parameter settings that allow the composite automaton to stay away from an Error state. To do so, we employ parametric timed automata which are an extension of the classical timed automata [1] and adapt the methodology for parametric analysis of real-time systems proposed in [14] to derive regions of free design parameters that can provide such guarantee.

3.2 Parametric Modelling

Even the simplified HCS is too complex to be parametrically modelled completely. Therefore, we have worked out an abstraction of the system to limit the state space and to concentrate in isolation on each outstanding issue (the non-preemptive scheduler, the different criticality of the timing constraints, etc.).

The abstract model consists of two parts. The first models the release of the packets on the network according to a periodic pattern. The second models the network and device, including the scheduling policy and the real-time constraints which can be hard or firm real-time constraints.

3.2.1 Packet Release Modeling

We model the release of packets as activation automata, shown in Figure 3.3. Each stream of packets is characterized by the offset for the first release (transition from initial state to the second state), and is then periodic afterwards (self transition on the second state). A release signal is emitted every time a transition is taken, and is used to synchronize the automaton with the rest of the system.

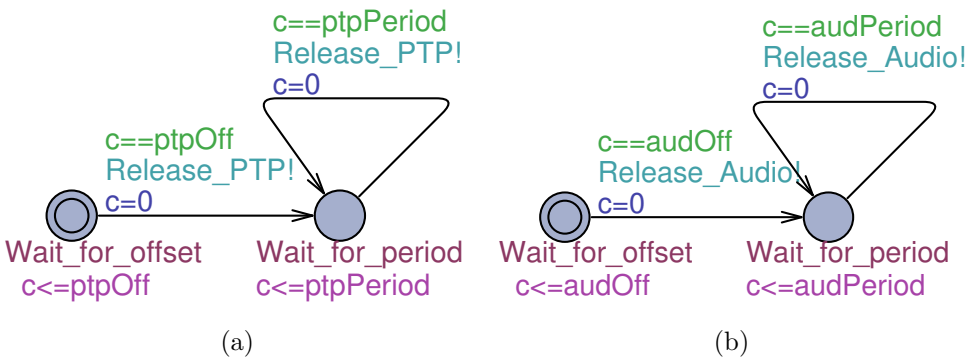


Figure 3.3: PTP and audio task activation automata

3.2.2 Schedulability Checker Modeling

The remaining part of the system is modelled as a set of schedulability checkers [14] that are *non-preemptive*, i.e. a transmission will not be interrupted if it has already started. The schedulers are also *prioritized*, so that when there is no ongoing transmission and many packets are ready, the PTP packets go first and the audio packets back off.

The scheduler checker for PTP packets is shown in Figure 3.4, where:

- D_1 is the deadline of PTP packets,
- C_1/C_2 are the transmission time of PTP/audio packets
- *task* denotes the currently-executed task,
- n_1 and n_2 record the number of PTP and audio packets released during the current execution
- c is a clock accumulating the time since the task queues were last idle
- r is the sum of the time needed to complete all tasks released since the checker was last idle.

The model of the Audio checker is similar to but simpler than the PTP checker because task audio has a lower priority. D_2 is the relative deadline

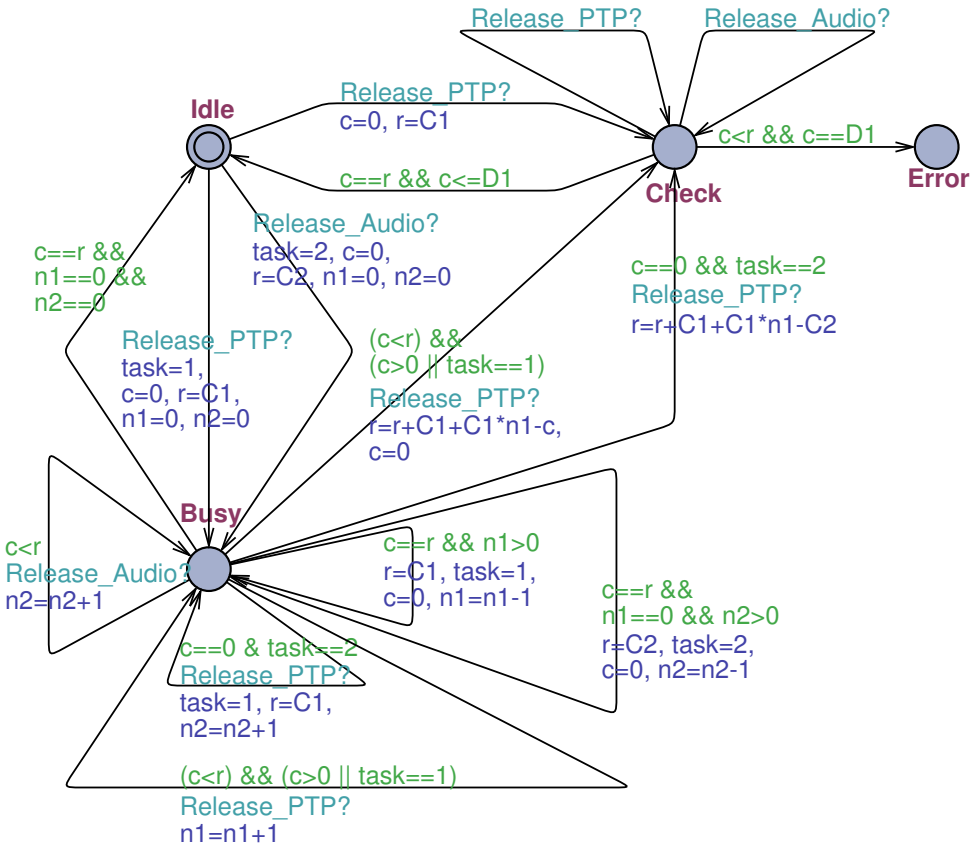
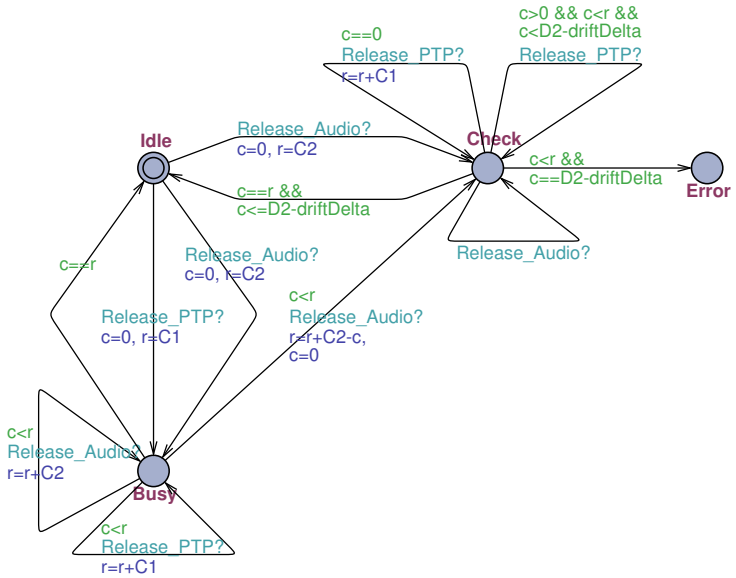


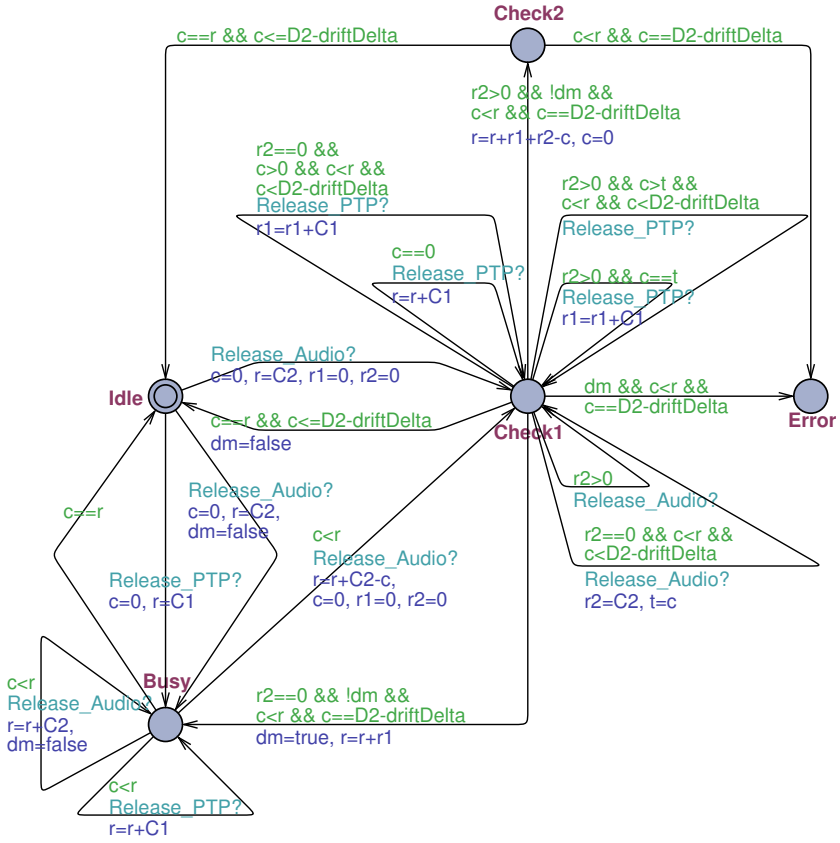
Figure 3.4: Schedulability checker for PTP packets

of audio packets and `driftDelta` is introduced to account for the offset time of the local clock compared to the server clock. The worst case happens when the local clock is substantially slower than the server clock and thus when an audio packet is received, the actual deadline to be verified would be $D_2 - \texttt{driftDelta}$ instead of D_2.

In fact, the requirement of no deadline miss (hard deadline) is difficult to obtain in real-time environments. Therefore, in order to make the analysis more practical, the requirement is relaxed by allowing an audio packet to sometimes miss its deadline (*firm real-time constraint*). A firm real-time constraint is given by a deadline and by a couple (m, n) meaning that m deadlines can be missed every n jobs [41]. In our case study, a packet may miss its deadline as long as the previous packet has not already missed the



(a) Hard deadline



(b) Firm deadline

Figure 3.5: Schedulability checker for audio packets

deadline ($m = 1$, $n = 2$). The checkers for both constraints are shown in Figure 3.5.

We introduce four new variables: dm is a boolean variable used to capture the fact that one deadline miss has already happened ($dm = true$), r_1 is a real variable used to record the total execution time of all PTP instances released after the currently-checked instance and before a deadline miss or the next audio arrival, r_2 marks the next audio arrival whose time is marked in t.

Intuitively, the transitions can be interpreted as follows:

- The transitions to Idle are taken when the task instance being checked in Check or a sequence of tasks arrived in Busy, has finished execution.
- The transitions to Busy are taken when an instance of task PTP or Audio is released. Self-loops are taken to queue the newly-released instances and to retrieve them when the current execution has finished.
- The transitions to Check are taken when a PTP instance is (non-deterministically) chosen for checking. Before verifying the deadline, the execution (or transmission) time of all other PTP instances in the queue must be taken into account as they would be scheduled before the current instance.
- The transition to Error is taken when the currently-executed instance misses its deadline.

3.3 Parametric Analysis

We have performed several experiments with a diverse set of parameters and the results of two such experiments which differ in the amount of offset by which packets are issued to the network are briefly summarized in this

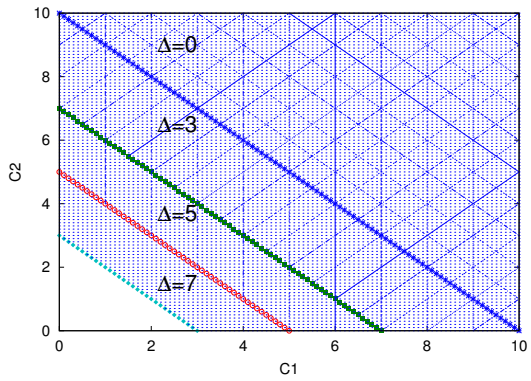
	Experiment 1	Experiment 2
$ptpOff$	0	5
$ptpPeriod$	40	40
D_1	10	10
$audOff$	0	0
$audPeriod$	10	10
D_2	10	10

Table 3.1: Fixed parameter settings

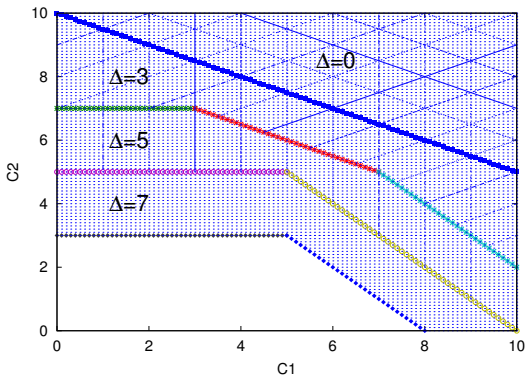
section. For the checker shown in the previous section, the free parameters are the transmission times C_1 and C_2. The values of the fixed parameters for each of the experiments are shown in Table 3.1.

Modelling the time aspect of HCS is possible in both Uppaal [31] and NuSMV [16]. However, capturing a specific instant of time cannot be done in Uppaal as so far it has supported only integer variables. Another important limitation of the Uppaal model is that it only answers yes/no to the verification problem without providing further feedback for the designers regarding how to adjust parameters so that the system remains feasible.

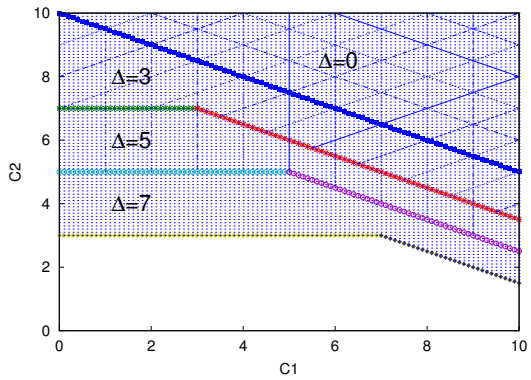
To do parametric analyses on HCS, we have modelled it using NuSMV and by adapting the parametric modelling tool [14] built upon NuSMT [11], we have derived the feasibility (not shaded) and infeasibility (shaded) regions for the PTP checker shown in Figure 3.6. The regions for the Audio checker under a *hard* and *firm* real-time requirement are also shown in the same figure where `driftDelta` (denoted as Δ) is introduced to account for the offset time of the local clock compared to the server clock. By joining the PTP and Audio feasibility regions together, we can obtain the regions which fully describe the assumption on the environment of HCS.



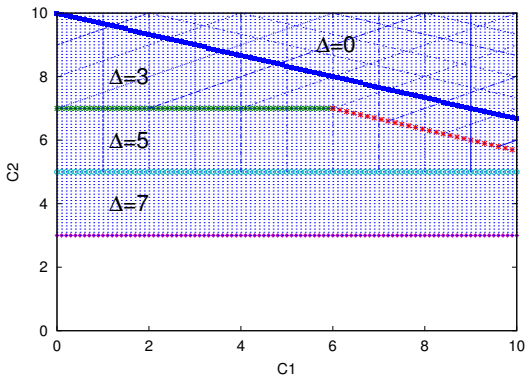
(a) Task audio, hard deadline, exp.1



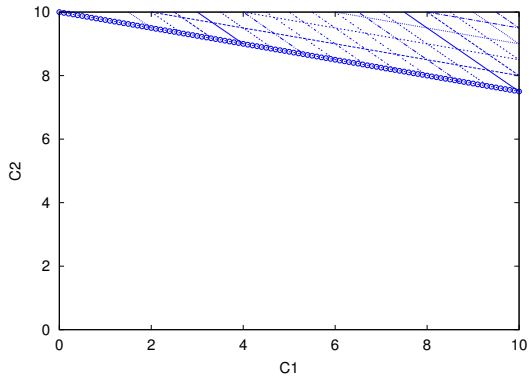
(b) Task audio, hard deadline, exp.2



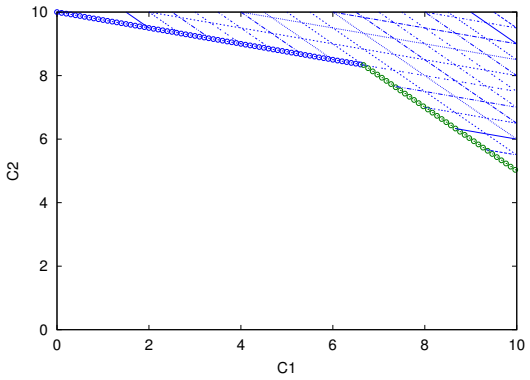
(c) Task audio, firm deadline, exp.1



(d) Task audio, firm deadline, exp.2



(e) Task PTP, exp.1



(f) Task PTP, exp.2

Figure 3.6: Audio (in)feasibility regions for $\Delta = 0, 3, 5, 7$ and PTP (in)feasibility regions in two experiments

Chapter 4

Tag Systems

We use denotational tag systems as our semantic domain [5, 37]. In intuitive terms, a tag system is a representation of the behaviors of a component in terms of sets of events that take place at its interface, intended as a collection of visible ports. Tags, which are associated to every event, characterize the temporal evolution of the behaviors. By changing the structure of tags, one can choose among different notions of time. Formally, a *tag structure* $\mathcal{T}$ is a pair $(T, \leq)$ where T is a set of *tags* and $\leq$ is a partial order on the tags. To distinguish the tag order of $\mathcal{T}$, we refer to it as $\leq_{\mathcal{T}}$ when necessary. The ordering among tags is used to resolve the ordering among events at the system interface. For instance, by using the set of real numbers as tags, with their usual ordering, one can place events anywhere in real time. Conversely, a set of partially ordered symbolic tags can be used to express precedence between events in a branching-time setting.

Events occur at the interface of a component. A component exposes a set V of *variables* (or *ports*) which can take values from a set D. An *event* is a snapshot of a variable state, capturing the variable value at some point in time. Formally, an *event* e on a variable $v \in V$ is a pair (τ, d) of a tag $\tau \in T$ and a value $d \in D$. The simplest way of characterizing a behavior is as a collection of events for each variable. In this work, we are interested

in constructing behaviors incrementally, using an executable model. For this reason, we index the events of a variable into a sequence, with the understanding that events later in the sequence have larger tags [5]. A behavior σ assigns a sequence of events to every variable in V, and is then a function $\sigma \in V \mapsto (\mathbb{N} \mapsto (T \times D))$.

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

4.1 Homogeneous Composition

Combining tag systems over the same tag structure amounts to considering only those behaviors which are consistent with every component. When the sets of variables coincide, this operation corresponds to taking the intersection of the behaviors of all components. When the sets of variables are different, two behaviors are considered consistent if they agree on the shared variables. In this case, we say that the behaviors are *unifiable*. Composition consists in retaining all and only the unifiable behaviors.

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

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

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

Definition 1 ([5])**.** The homogeneous composition P of two tag systems $P_1 = (V_1, \mathcal{T}, \Sigma_1)$ and $P_2 = (V_2, \mathcal{T}, \Sigma_2)$, written $P = P_1 \parallel P_2$, is the tag system $P = (V_1 \cup V_2, \mathcal{T}, \Sigma_1 \wedge \Sigma_2)$, where

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

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

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

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

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

which makes the intersection operator involved with composition explicit.

4.2 Heterogeneous Composition

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

Definition 2 ([5])**.** Let $\mathcal{T}$ and $\mathcal{T}'$ be tag structures. A *tag morphism* from $\mathcal{T}$ to $\mathcal{T}'$ is a total map $\rho : \mathcal{T} \mapsto \mathcal{T}'$ s.t.

$$\forall \tau_1, \tau_2 \in \mathcal{T}, \tau_1 \leq_{\mathcal{T}} \tau_2 \Rightarrow \rho(\tau_1) \leq_{\mathcal{T}'} \rho(\tau_2).$$

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

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

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

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

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

It is convenient, however, to retain some information of the original tag structures in the composition, since they are often referred to in the heterogeneous composition, as we will see in the sequel. To do so, we construct the composition over the fibered product [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:

$$(\tau_1, \tau_2) \leq (\tau_1', \tau_2') \iff (\tau_1 \leq_{\mathcal{T}_1} \tau_1') \wedge (\tau_2 \leq_{\mathcal{T}_2} \tau_2').$$

where $T_1 \,{}_{\rho_1}\!\times_{\rho_2} T_2 = \{(\tau_1, \tau_2) \in T_1 \times T_2 : \rho_1(\tau_1) = \rho_2(\tau_2)\}$. We denote by $\sigma|_{V_1, \mathcal{T}_1}$ the restriction of σ to the variables in V_1 and to the element $\mathcal{T}_1$ of

the fibered product.[1] With this notion, we can define the *heterogeneous composition*.

Definition 3 ([5])**.** Let $P_1 = (V_1, \mathcal{T}_1, \Sigma_1)$ and $P_2 = (V_2, \mathcal{T}_2, \Sigma_2)$ be tag systems and let $\rho_1 : \mathcal{T}_1 \mapsto \mathcal{T}$ and $\rho_2 : \mathcal{T}_2 \mapsto \mathcal{T}$ be tag morphisms. The *heterogeneous composition* $P = P_1 \,{}_{\rho_1}\|_{\rho_2}\, P_2$ is the tag system $P = (V_1 \cup V_2, \mathcal{T}_1 \,{}_{\rho_1}\times_{\rho_2}\, \mathcal{T}_2, \Sigma_1 \,{}_{\rho_1}\wedge_{\rho_2}\, \Sigma_2)$, where

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

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

Chapter 5

Tag Machines

Tag machines (TMs) [6] have been introduced to represent tag systems in a homogeneous context. Since our aim is to provide an operational representation for heterogeneous systems, we extend the TM formalism to encompass the heterogeneous context.

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

Definition 4 ([6])**.** An *algebraic* tag structure is a tag structure $\mathcal{T} = (T, \leq, \cdot)$ where $\cdot$ is a binary operation on T called *concatenation*, such that:

i) $(T, \cdot)$ is a monoid with identity element $\hat{\imath}$,

ii) $\forall \tau, \tau', \bar{\tau}, \bar{\tau}' \in T : (\tau \leq \tau') \wedge (\bar{\tau} \leq \bar{\tau}') \Rightarrow \tau \cdot \bar{\tau} \leq \tau' \cdot \bar{\tau}'$,

iii) $\exists \epsilon \in T : \forall \tau \in T : (\epsilon \leq \tau) \wedge (\epsilon \cdot \tau = \tau \cdot \epsilon = \epsilon)$.

Tags are organized in *tag vectors* $\vec{\tau} = (\tau^{v_1}, \ldots, \tau^{v_n})$, where n is the number of variables in V. During transitions, tag vectors evolve according to a matrix $\mu : V \times V \mapsto T$ called a *tag piece* [6]. Given a tag vector $\vec{\tau}$ and a tag piece μ, the new tag vector is $\vec{\tau}_\mu = \vec{\tau} \cdot \mu$ given by

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

where 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. 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. We denote by $\mathfrak{dom}(\nu)$ the domain of such ν and by $L(V, \mathcal{T})$ the universe of all labeled tag pieces defined over a variable set V and tag structure $\mathcal{T}$. In the following, we assume that tag pieces are always labeled and implicitly associate a labeling function ν to a tag piece μ.

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

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

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

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

A tag machine M is a finite automaton where transitions are marked by labeled tag pieces, or simply *labels*. Our definition below differs from that proposed by Benveniste et al. [6] for certain simplifications and for the addition of a set of accepting states.

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

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

A run r of a TM is a sequence of states and transitions

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

such that $(s_{i-1}, \mu_i, s_i) \in E$ for $1 \leq i \leq m$. Intuitively, a TM is used to construct a behavior by following its labeled transitions over a run, and applying the tag pieces sequentially to the initial tag vector of the TM. 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 $\hat{\imath}_{\mathcal{T}}$, while the index vector is initialized to 0. The behavior $\sigma(r)$[1] of a run r is constructed incrementally by starting from an empty behavior σ_0 and computing:

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

A run r of M is *valid* if the concatenation is always defined along the run, and if $s_m \in F$. 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.

5.1 Composition of Tag Machines

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

5.1.1 Homogeneous Composition

Let $M_1 = (V_1, \mathcal{T}_1, S_1, s_{01}, F_1, E_1)$ and $M_2 = (V_2, \mathcal{T}_2, S_2, s_{02}, F_2, E_2)$ be TMs. We first examine the composition of homogeneous TMs, by adapting the original definition [6] and assuming that $\mathcal{T}_1 = \mathcal{T}_2 = \mathcal{T}$. Two labeled tag pieces $\mu_1 \in L(V_1, \mathcal{T})$ and $\mu_2 \in L(V_2, \mathcal{T})$ are *unifiable*, written $\mu_1 \bowtie \mu_2$, if and only if they are the same on the shared variables. That is, if we denote the set of shared variables with $W = V_1 \cap V_2$, then for all pairs

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

$(w, v) \in W \times W$:

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

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

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

Homogeneous composition can then be defined as follows.

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

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

5.1.2 Heterogeneous Composition

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

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

The newly-composed TM is then defined on $\mathcal{T}_1 {}_{\rho_1}\!\times_{\rho_2} \mathcal{T}_2 = (T_1 {}_{\rho_1}\!\times_{\rho_2} T_2, \leq, \cdot)$, where $\leq \stackrel{\text{def}}{=} (\leq_{\mathcal{T}_1}, \leq_{\mathcal{T}_2})$ and $\cdot \stackrel{\text{def}}{=} (\cdot_{\mathcal{T}_1}, \cdot_{\mathcal{T}_2})$. This fibered tag structure is shown to be algebraic in the next lemma.

Lemma 1. *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* $\cdot \stackrel{\text{def}}{=} (\cdot_{\mathcal{T}_1}, \cdot_{\mathcal{T}_2})$ *is algebraic as defined in Definition 4.*

Proof. i) Let $(\tau_1, \tau_2), (\tau_1', \tau_2') \in T_1{}_{\rho_1}\!\times_{\rho_2} T_2$, then

$$(\tau_1, \tau_2) \cdot (\tau_1', \tau_2') = (\tau_1 \cdot_{\mathcal{T}_1} \tau_1', \tau_2 \cdot_{\mathcal{T}_2} \tau_2').$$

By Definition 7, we have that:

$$\rho_1(\tau_1 \cdot_{\mathcal{T}_1} \tau_1') = \rho_1(\tau_1) \cdot_{\mathcal{T}} \rho_1(\tau_1'),$$
$$\rho_2(\tau_2 \cdot_{\mathcal{T}_2} \tau_2') = \rho_2(\tau_2) \cdot_{\mathcal{T}} \rho_2(\tau_2').$$

Besides, the membership assumption of (τ_1, τ_2) means $\rho_1(\tau_1) = \rho_2(\tau_2)$ and that of (τ_1', τ_2') means $\rho_1(\tau_1') = \rho_2(\tau_2')$. Therefore

$$\rho_1(\tau_1 \cdot_{\mathcal{T}_1} \tau_1') = \rho_2(\tau_2 \cdot_{\mathcal{T}_2} \tau_2')$$

and this implies $(\tau_1 \cdot_{\mathcal{T}_1} \tau_1', \tau_2 \cdot_{\mathcal{T}_2} \tau_2') \in T_1{}_{\rho_1}\!\times_{\rho_2} T_2$. Hence $(T_1{}_{\rho_1}\!\times_{\rho_2} T_2, \cdot)$ is a monoid with the identity element $(\hat{\imath}_{\mathcal{T}_1}, \hat{\imath}_{\mathcal{T}_2})$ as

$$(\tau_1, \tau_2) \cdot (\hat{\imath}_{\mathcal{T}_1}, \hat{\imath}_{\mathcal{T}_2}) = (\tau_1 \cdot_{\mathcal{T}_1} \hat{\imath}_{\mathcal{T}_1}, \tau_2 \cdot_{\mathcal{T}_2} \hat{\imath}_{\mathcal{T}_2}) = (\tau_1, \tau_2).$$

ii) Let $(\tau_1, \tau_2), (\tau_1', \tau_2'), (\bar{\tau}_1, \bar{\tau}_2), (\bar{\tau}_1', \bar{\tau}_2') \in T_1{}_{\rho_1}\!\times_{\rho_2} T_2$ such that

$$(\tau_1, \tau_2) \leq (\tau_1', \tau_2') \text{ and } (\bar{\tau}_1, \bar{\tau}_2) \leq (\bar{\tau}_1', \bar{\tau}_2').$$

We then have the following:

$$(\tau_1, \tau_2) \cdot (\bar{\tau}_1, \bar{\tau}_2) = (\tau_1 \cdot_{\mathcal{T}_1} \bar{\tau}_1, \tau_2 \cdot_{\mathcal{T}_2} \bar{\tau}_2),$$
$$(\tau_1', \tau_2') \cdot (\bar{\tau}_1', \bar{\tau}_2') = (\tau_1' \cdot_{\mathcal{T}_1} \bar{\tau}_1', \tau_2' \cdot_{\mathcal{T}_2} \bar{\tau}_2').$$

By assumption,

$$(\tau_1, \tau_2) \leq (\tau_1', \tau_2') \text{ means } (\tau_1 \leq_{\mathcal{T}_1} \tau_1') \wedge (\tau_2 \leq_{\mathcal{T}_2} \tau_2'),$$
$$(\bar{\tau}_1, \bar{\tau}_2) \leq (\bar{\tau}_1', \bar{\tau}_2') \text{ means } (\bar{\tau}_1 \leq_{\mathcal{T}_1} \bar{\tau}_1') \wedge (\bar{\tau}_2 \leq_{\mathcal{T}_2} \bar{\tau}_2').$$

Therefore, $(\tau_1 \cdot_{\mathcal{T}_1} \bar{\tau}_1) \leq_{\mathcal{T}_1} (\tau_1' \cdot_{\mathcal{T}_1} \bar{\tau}_1')$ and $(\tau_2 \cdot_{\mathcal{T}_2} \bar{\tau}_2) \leq_{\mathcal{T}_2} (\tau_2' \cdot_{\mathcal{T}_2} \bar{\tau}_2')$ implying

$$(\tau_1 \cdot_{\mathcal{T}_1} \bar{\tau}_1, \tau_2 \cdot_{\mathcal{T}_2} \bar{\tau}_2) \leq (\tau_1' \cdot_{\mathcal{T}_1} \bar{\tau}_1', \tau_2' \cdot_{\mathcal{T}_2} \bar{\tau}_2'), \text{ or}$$
$$(\tau_1, \tau_2) \cdot (\bar{\tau}_1, \bar{\tau}_2) \leq (\tau_1', \tau_2') \cdot (\bar{\tau}_1', \bar{\tau}_2').$$

iii) The least element is $(\epsilon_{\mathcal{T}_1}, \epsilon_{\mathcal{T}_2})$ as for any $(\tau_1, \tau_2) \in T_1 \,{}_{\rho_1}\!\times_{\rho_2} T_2$, it is true that $(\epsilon_{\mathcal{T}_1} \leq_{\mathcal{T}_1} \tau_1) \wedge (\epsilon_{\mathcal{T}_2} \leq_{\mathcal{T}_2} \tau_2)$, hence $(\epsilon_{\mathcal{T}_1}, \epsilon_{\mathcal{T}_2}) \leq (\tau_1, \tau_2)$. In addition, $(\epsilon_{\mathcal{T}_1}, \epsilon_{\mathcal{T}_2}) \cdot (\tau_1, \tau_2) = (\epsilon_{\mathcal{T}_1} \cdot_{\mathcal{T}_1} \tau_1, \epsilon_{\mathcal{T}_2} \cdot_{\mathcal{T}_2} \tau_2) = (\epsilon_{\mathcal{T}_1}, \epsilon_{\mathcal{T}_2})$.

□

Referring to the previous notation, two tag pieces μ_1 and μ_2 are *unifiable* under morphisms ρ_1 and ρ_2, written $\mu_1 \,{}_{\rho_1}\!\bowtie_{\rho_2} \mu_2$, whenever for all pairs $(w, v) \in W \times W$:

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

When unifiable, their unification $\mu = \mu_1 \,{}_{\rho_1}\!\sqcup_{\rho_2} \mu_2$ defined over algebraic tag structure $\mathcal{T}_1 \,{}_{\rho_1}\!\times_{\rho_2} \mathcal{T}_2$ 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 labeling function is the same as in the homogeneous case:

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

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

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

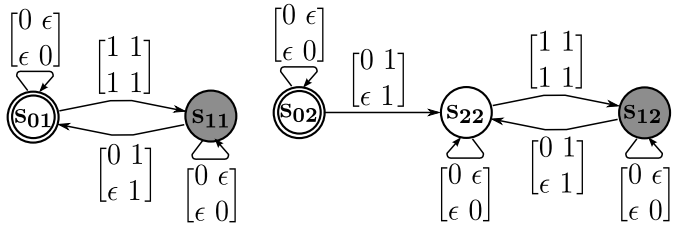
- $V = V_1 \cup V_2$,
- $\mathcal{T}_1 \ {}_{\rho_1}\times_{\rho_2} \mathcal{T}_2 = (T_1 \ {}_{\rho_1}\times_{\rho_2} T_2, \leq, \cdot)$ where $\leq = (\leq_{\mathcal{T}_1}, \leq_{\mathcal{T}_2})$ and $\cdot = (\cdot_{\mathcal{T}_1}, \cdot_{\mathcal{T}_2})$,
- $S = S_1 \times S_2$, $s_0 = (s_{01}, s_{02})$, $F = F_1 \times F_2$,
- $E = \{((s_1, s_2), \mu_1 \ {}_{\rho_1}\sqcup_{\rho_2} \mu_2, (s_1', s_2')) : (s_1, \mu_1, s_1) \in E_1 \wedge (s_2, \mu_2, s_2) \in E_2 \wedge \mu_1 \ {}_{\rho_1}\bowtie_{\rho_2} \mu_2\}$ where $\mu_1 \ {}_{\rho_1}\sqcup_{\rho_2} \mu_2$ extends to all the members of the unification set.

It is noticeable here that homogeneous composition is a special case of the heterogeneous one with identity morphisms.

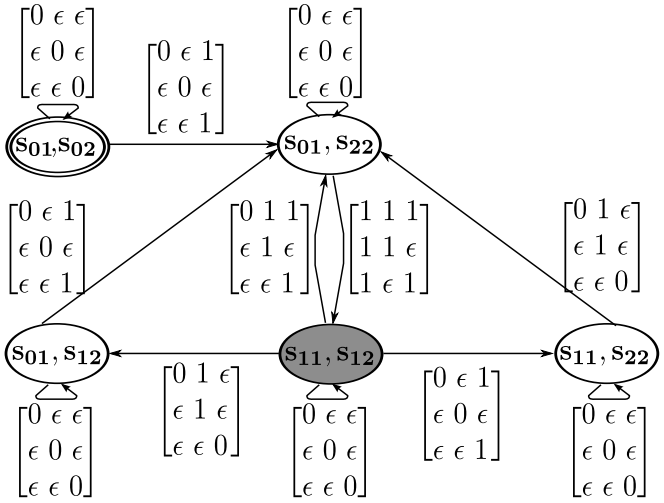
5.2 Interoperable TMs and Composition Soundness

Ideally, we would like there to be a direct correspondence between tag systems and TMs. Let Σ_i and Σ be the behavioral semantics of M_i and composition $M_1 \ {}_{\rho_1}\|_{\rho_2} \ M_2$ respectively, we expect that every behavior of Σ be obtained by composing some pair of behaviors from Σ_i. When this is the case, we say that composition is sound. Example 2 shows that this property generally does not hold even for homogeneous systems.

Example 2. We consider two sets of behaviors Σ_1 and Σ_2 defined on two sets, $V_1 = \{x, y\}$ and $V_2 = \{x, z\}$ respectively, of variables with values in $D = \{true\}$. Since D is a singleton set, we shall omit mentioning



(a) TM accepting Σ_1 (b) TM accepting Σ_2



(c) TM accepting $\Sigma_1 \wedge \Sigma_2$

Figure 5.1: Non-interoperable TMs

the variable value in the rest of the example. These behavioral sets are expressed formally as follows. Let $\sigma_i \in \Sigma_i$ and $\mathbf{enum}_{\sigma_i}(v_i)$ be the total number of events on variable $v_i \in V_i$ in behavior σ_i where $i \in \{1, 2\}$ and let $k \geq 1$:

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

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

Let *reaction* be a maximal set of events with identical tags in a behavior [6],

these behavioral sets can be organized in terms of successive reactions:

Σ_1 :

x :	1		3		5		...
y :	1	2	3	4	5	6	...

Σ_2 :

x :	2		4		6	...
z :	1	2	3	4	5	6 ...

We use algebraic tag structure $(\mathbb{N} \cup \{\epsilon\}, \leqslant, +)$ and the tag piece structure described in Example 1 to model the behaviors in Σ_i as TMs where $\epsilon \stackrel{\text{def}}{=} -\infty$, the row and column designation orders are (x, y) in Figure 5.1(a), (x, z) in Figure 5.1(b) and (x, y, z) in Figure 5.1(c). The initial states are double-circled and the accepting states are shaded.

Tagging the shared variable x can depend on tagging non-shared variables y or z even though there is no real dependence between their tags. Going from s_{01} to s_{11}, TM 5.1(a) tags x and y simultaneously and equally. It then can go back to s_{01}, tagging only y and subsequently repeating the tagging cycle at this state. TM 5.1(b) instead tags only z initially and goes to s_{22}. It then tags both x and z at the same time and goes to s_{12} where it again tags only z before returning to s_{22}. It is easy to verify that TM 5.1(a) and 5.1(b) accept the behavioral sets Σ_1 and Σ_2 respectively. When composing them, the composed TM should not to accept any behavior since $\Sigma_1 \wedge \Sigma_2 = \emptyset$. Its set of accepted behavior is, however, not empty as shown in Figure 5.1(c) because TM 5.1(a) can stay silent while TM 5.1(b) is tagging z. The two TMs then synchronize and tag all variables simultaneously, after which they can go on tagging their own internal variable.

Remarkably, the fact that TM composition is not sound was not previously observed in the homogeneous context [6]; since homogeneous composition is not sound, the same is true for heterogeneous composition. The consequence is that the operational model overestimates the behaviors of

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

The intuition behind TM interoperability is that tagging shared variables should not depend on tagging non-shared variables. Because such dependency is visible only internally inside components, it cannot be taken into account in the composition. A set $\overline{V} \subseteq V$ is said to be *locally independent* in tag machine $M = (V, \mathcal{T}, S, s_0, F, E)$, written $\mathfrak{lind}(M, \overline{V})$, if tagging $v \in \overline{V}$ depends only on tagging variables in $\overline{V}$. If we define the following predicate

$$\mathfrak{lind}(\mu, \overline{V}) \stackrel{\text{def}}{=} (\forall w \in V \setminus \overline{V}, \forall v \in \overline{V} : \mu(w, v) = \epsilon),$$

then the local independence of $\overline{V}$ in M is defined as

$$\mathfrak{lind}(M, \overline{V}) \stackrel{\text{def}}{=} (\forall (s, \mu, s') \in E : \mathfrak{lind}(\mu, \overline{V})).$$

The interoperability between M_1 and M_2 is then formally defined as below.

Definition 9. Two TMs M_1 and M_2 are said to be *interoperable*, written $M_1 \bowtie M_2$, if their shared variables are locally independent in both TMs:

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

Interoperability behaves well under multiple composition.

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

Proof. We prove the lemma by inductive reasoning.

- The base case $n = 2$ is trivial.
- For the step case, we assume the lemma holds for $k = n-1$ and prove that it also holds for $k+1 = n$. Let V_i be the variable set, μ_i a label of M_i, μ the unification of $\mu_1, \mu_2 \ldots, \mu_{n-1}$, $\mathcal{T}$ the fibered product of $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_{n-1}$ and $V = V_1 \cup V_2 \ldots \cup V_{n-1}$.

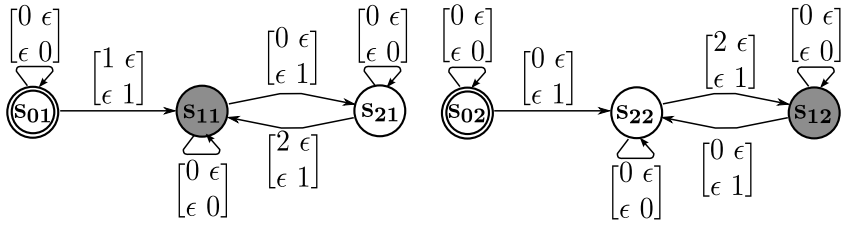
  It is easy to see that $\mathfrak{lind}(\mu, V \cap V_n)$ holds. Let $w \in V \setminus (V \cap V_n)$, then it must be that $w \in V$ and $w \notin V_n$ and so $w \in V_j \setminus V_n$ for some $1 \leq j \leq n-1$. Likewise, let $v \in V \cap V_n$, then $v \in V_r \cap V_n$ for some $1 \leq r \leq n-1$. We show that $\mu(w, v) = \epsilon_{\mathcal{T}}$ is true despite the choice of j and r.

  i) If j and r can coincide, i.e., $j = r$, then w and v can be in the same variable set. Since $M_r \bowtie M_n$, by the interoperability definition it is true that $\mu_r(w, v) = \epsilon_{\mathcal{T}_r}$ and thus $\mu(w, v) = \epsilon_{\mathcal{T}}$.

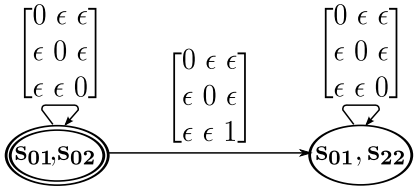
  ii) Otherwise, i.e., $j \neq r$, then w and v cannot be in the same variable set and the $wv-$entries of the composed label μ are set to $\epsilon_{\mathcal{T}}$, by the label composition rule in Section 5.1.

  To show that $\mathfrak{lind}(\mu_n, V \cap V_n)$ holds, we consider $w \in V_n \setminus (V \cap V_n)$ and $v \in V \cap V_n$. The latter means $v \in V_r \cap V_n$ for some $1 \leq r \leq n-1$. The former means $w \in V_n$ and $w \notin V_j$ for all $1 \leq j \leq n-1$ which implies $w \in V_n \setminus V_r$. This together with the interoperability $M_n \bowtie M_r$ implies $\mu_n(w, v) = \epsilon_{\mathcal{T}_n}$.

□



(a) Interoperable TM accepting Σ_1 (b) Interoperable TM accepting Σ_2



(c) Interoperable Composition accepting $\Sigma_1 \wedge \Sigma_2$

Figure 5.2: Interoperable TMs

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

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

Interoperable TMs representing Σ_i are depicted in Figure 5.2(a) and 5.2(b). Tagging the shared interface variable x is now made locally independent in both machines, hence they can be composed interoperably. The composed TM is shown in Figure 5.2(c) where no behavior can be accepted since the accepting state is not reachable from the initial state. This is because TM 5.2(a) has to stutter in s_{01} while TM 5.2(b) tags its internal variable and moves to s_{22}. The two TMs then have to stutter there forever since only the stuttering labels $\left[\begin{smallmatrix}0 & \epsilon\\ \epsilon & 0\end{smallmatrix}\right]$ are unifiable.

Example 4. We consider a simplified version of the water controlling system proposed by Benvenuti et al. [8]. It consists of two components: a

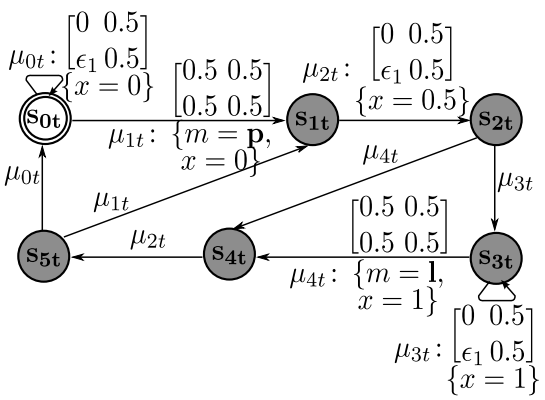
water tank and a water level controller, connected in a closed-loop fashion. We assume that the water level $x(t)$ is changed linearly as follows:

$$x(t) \stackrel{\text{def}}{=} \begin{cases} \Delta_t * (\mathbf{f_i} - \mathbf{f_o}) & \text{when command is Open} \\ \mathbf{H} - \Delta_t * \mathbf{f_o} & \text{when command is Close} \end{cases} \tag{5.1}$$

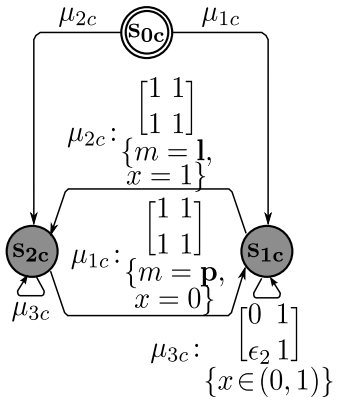
where $\mathbf{f_i}$ and $\mathbf{f_o}$ denote the constant inlet and outlet flow, respectively, $\mathbf{H}$ denotes the height when the tank is full of water and Δ_t denotes the time elapsed since t_0 at which the tank reaches the maximum/minimum water level $\mathbf{H}$, i.e. $\Delta_t = t - t_0$.

Let $P_1 = (V_1, \mathcal{T}_1, \Sigma_1)$ and $P_2 = (V_2, \mathcal{T}_2, \Sigma_2)$ be two tag systems representing the tank and the water controller, respectively, where $\mathcal{T}_1 = (\mathbb{R}_+ \cup \{\epsilon_1\}, +)$, $\mathcal{T}_2 = (\mathbb{N} \cup \{\epsilon_2\}, +)$, $\epsilon_1 = \epsilon_2 = -\infty$, and $V_1 = V_2 = \{m, x\}$. Variable m denotes the command values, which can be Open ($\mathbf{p}$) or Close ($\mathbf{l}$), and variable x denotes the water level, which is of positive real type, i.e. $D_m = \{\mathbf{p}, \mathbf{l}\}$ and $D_x = \mathbb{R}_+$. Assume that $\mathbf{f_i} = 2, \mathbf{f_o} = 1, \mathbf{H} = 1$, we model in this example a linear water level evolution of a water controlling system. The tank component shown in Figure 5.3(a) depicts the water level linear evolution as specified in (5.1). Upon knowing of the tank emptiness/fullness, the controller component in Figure 5.3(b) will issue an Open/Close command. Intuitively, the controller behaviors ensure that controlling commands are always issued timely (i.e., Open when the tank is empty and Close when it is full), no matter how the water level evolves, while the tank behaviors accept untimely controlling commands and allow water spillages or shortages, given that the water level evolves linearly.

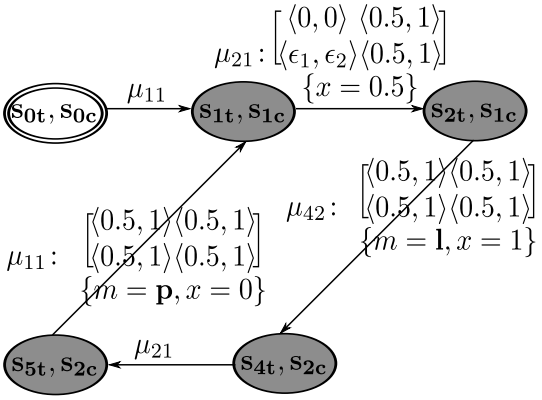
For the sake of simplicity, the events described by the tank component are timestamped periodically every 0.5 time unit. While the tank system uses physical time to stamp its behaviours, the controller system instead timestamps its events logically, described by the integer tag set $\mathbb{N}$. For the sake of expressiveness, some of the labels can be represented symbolically. For example, to capture any event of variable x happening at a specific



(a) TM representing the behaviors of the tank



(b) TM representing the behaviors of the controller



(c) TM representing the behaviors of the tank-controller composition

Figure 5.3: A simple water tank system

time point, we attach expressions such as $x \in D_x$ to the tag piece capturing that time point, meaning that in such an event x can take any value in its domain. Since the tank and the controller capture different behaviors, composing them is only possible under the presence of morphisms such as $\rho_1 : \mathcal{T}_1 \mapsto \mathcal{T}_1$ and $\rho_2 : \mathcal{T}_2 \mapsto \mathcal{T}_1$ given by $\rho_1(\tau_1) = \tau_1$, $\rho_2(\tau_2) = 0.5 * \tau_2$. Figure 5.3(c) shows their composition, where accepted behaviors ensure timely controlling commands and linear water evolution.

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

Theorem 1. *Any behavior $\sigma_{\|}$ of the composition $M_1 \,{}_{\rho_1}\|_{\rho_2}\, M_2$, where M_1 and M_2 are interoperable, is obtained by composing some behavior σ_1 of M_1 and σ_2 of M_2, i.e., $\Sigma \subseteq \Sigma_1 \,{}_{\rho_1}\wedge_{\rho_2}\, \Sigma_2$.*

Proof. Since behavior $\sigma_{\|}$ is accepted by the composition $M_1 \,{}_{\rho_1}\|_{\rho_2}\, M_2$, there must exist a valid run r in the composition

$$r : s_0 \xrightarrow{\mu_1} s_1 \xrightarrow{\mu_2} s_2 \ldots \xrightarrow{\mu_n} s_n$$

such that $\sigma(\mu_1\mu_2 \ldots \mu_n) = \sigma_{\|}$. In addition, it must be that $s_k = (s_{k1}, s_{k2})$ where $s_{ki} \in S_i$, for $0 \leq k \leq n$ and for $1 \leq i \leq 2$. Thus there must exist a valid run r_i in M_i:

$$r_i : s_{0i} \xrightarrow{\mu_{1i}} s_{1i} \xrightarrow{\mu_{2i}} s_{2i} \ldots \xrightarrow{\mu_{ni}} s_{in}$$

such that $\mu_{k1} \,{}_{\rho_1}\bowtie_{\rho_2}\, \mu_{k2}$ and $\mu_k = \mu_{k1} \,{}_{\rho_1}\sqcup_{\rho_2}\, \mu_{k2}$. Let $\sigma_i = \sigma(\mu_{1i}\mu_{2i} \ldots \mu_{ni})$ and $\vec{\tau}_k/\vec{\tau}_{ki}$ be tag vectors obtained at states s_k/s_{ki} in run r/r_i, in order to prove $\sigma_{\|} = \sigma_1 \,{}_{\rho_1}\sqcup_{\rho_2}\, \sigma_2$, we show by inductive reasoning that

i) the composed tag vector $\vec{\tau}_k$ is defined on the fibered product tag structure: $\tau_k^w \in \mathcal{T}_1 \,{}_{\rho_1}\times_{\rho_2}\, \mathcal{T}_2$ for $w \in V_1 \cup V_2$, and

ii) the restriction of the composed tag vector $\vec{\tau}_k$ to variables V_i and tag structure $\mathcal{T}_i$ is exactly $\vec{\tau}_{ki}$, i.e: $\vec{\tau}_k|_{V_1,\mathcal{T}_1} = \vec{\tau}_{k1}$ and $\vec{\tau}_k|_{V_2,\mathcal{T}_2} = \vec{\tau}_{k2}$.

Let $w \in V_1 \cup V_2, v' \in V_1 \cap V_2, v_i \in V_i \setminus (V_1 \cap V_2)$, μ^{vw} denote $\mu(v, w)$ and $\max / \max_{\mathcal{T}} / \max_{\mathcal{T}_i}$ be the maximum operator performed respectively on $\mathcal{T}_1 \,{}_{\rho_1}\times_{\rho_2}\, \mathcal{T}_2/\mathcal{T}/\mathcal{T}_i$.

- The base case $k = 0$ is trivial: $\tau_k^w = (\hat{\imath}_{\mathcal{T}_1}, \hat{\imath}_{\mathcal{T}_2}) \in \mathcal{T}_1 \,{}_{\rho_1}\times_{\rho_2}\, \mathcal{T}_2$ by Lemma 1, and $\vec{\tau}_0|_{V_i,\mathcal{T}_i} = \vec{\tau}_{0i}$ is true since $\vec{\tau}_{0i}$ is initialized to $\hat{\imath}_{\mathcal{T}_i}$ by the TM definition.

- For the step case, we assume the statements hold for $k \leq n - 1$ and prove it for $k = n$.

First, each tag element of the composed vector is computed as follows:

$$\tau_n^w = \max(\tau_{n-1}^{v'} \cdot \mu_n^{v'w}, \tau_{n-1}^{v_1} \cdot \mu_n^{v_1 w}, \tau_{n-1}^{v_2} \cdot \mu_n^{v_2 w}) \tag{5.2}$$

Because $\mathcal{T}_1 \,{}_{\rho_1}\!\times_{\rho_2} \mathcal{T}_2$ is algebraic (by Lemma 1), concatenating its elements is another element of itself. In addition, the max computation is well-defined since run r is valid. Therefore, τ_n^w must belong to $\mathcal{T}_1 \,{}_{\rho_1}\!\times_{\rho_2} \mathcal{T}_2$.

Second, restricting vector $\vec{\tau}_n$ to variables V_1 and tag structure $\mathcal{T}_1$ is a vector composed of the first tag components of τ_n^w where $w \in V_1$. By construction, the $v_2 w-$entries of the composed label μ_n are set to the least tag $(\epsilon_{\mathcal{T}_1}, \epsilon_{\mathcal{T}_2})$ of $\mathcal{T}_1 \,{}_{\rho_1}\!\times_{\rho_2} \mathcal{T}_2$ which are then canceled out from the tag concatenation (5.2).

i) Indeed, if variable w is shared, i.e., $w \in V_1 \cap V_2$, then by the interoperability assumption, $\mu_{n2}^{v_2 w} = \epsilon_{\mathcal{T}_2}$ and this implies $\mu_n^{v_2 w} = (\epsilon_{\mathcal{T}_1}, \epsilon_{\mathcal{T}_2})$.

ii) Otherwise, $w \in V_1 \setminus V_2$, then w and v_2 are not in the same variable set and by the label composition rule, the $v_2 w-$entries of the composed label μ_n are set to the least tag $(\epsilon_{\mathcal{T}_1}, \epsilon_{\mathcal{T}_2})$.

Thus, for $w \in V_1$, the tag concatenation (5.2) becomes:

$$\begin{aligned} \tau_n^w &= \max(\tau_{n-1}^{v'} \cdot \mu_n^{v'w}, \tau_{n-1}^{v_1} \cdot \mu_n^{v_1 w}, \tau_{n-1}^{v_2} \cdot (\epsilon_{\mathcal{T}_1}, \epsilon_{\mathcal{T}_2})) \\ &= \max(\tau_{n-1}^{v'} \cdot \mu_n^{v'w}, \tau_{n-1}^{v_1} \cdot \mu_n^{v_1 w}) \end{aligned}$$

By the hypothesis: $\tau_{n-1}^{v'} = (\tau_{(n-1)1}^{v'}, \tau_{(n-1)2}^{v'})$ and $\tau_{n-1}^{v_1} = (\tau_{(n-1)1}^{v_1}, \tau_2)$, for some tag $\tau_2 \in \mathcal{T}_2$. If $w \in V_1 \cap V_2$, then $\mu_n^{v'w} = (\mu_{n1}^{v'w}, \mu_{n2}^{v'w})$ and $\mu_n^{v_1 w} = (\mu_{n1}^{v_1 w}, \tau_2')$. Hence (5.2) becomes:

$$\begin{aligned} \tau_n^w &= \max((\tau_{(n-1)1}^{v'} \cdot_{\mathcal{T}_1} \mu_{n1}^{v'w}, \tau_{(n-1)2}^{v'} \cdot_{\mathcal{T}_2} \mu_{n2}^{v'w}), (\tau_{(n-1)1}^{v_1} \cdot_{\mathcal{T}_1} \mu_{n1}^{v_1 w}, \tau_2 \cdot_{\mathcal{T}_2} \tau_2')) \\ &= \begin{pmatrix} \max_{\mathcal{T}_1}(\tau_{(n-1)1}^{v'} \cdot_{\mathcal{T}_1} \mu_{n1}^{v'w}, \tau_{(n-1)1}^{v_1} \cdot_{\mathcal{T}_1} \mu_{n1}^{v_1 w}), \\ \max_{\mathcal{T}_2}(\tau_{(n-1)2}^{v'} \cdot_{\mathcal{T}_2} \mu_{n2}^{v'w}, \tau_2 \cdot_{\mathcal{T}_2} \tau_2') \end{pmatrix} \end{aligned}$$

If $w \in V_1 \setminus V_2$, then $\mu_n^{v'w} = (\mu_{n1}^{v'w}, \tau_2'')$ and $\mu_n^{v_1 w} = (\mu_{n1}^{v_1 w}, \tau_2')$. Hence (5.2) becomes:

$$\begin{aligned}\tau_n^w &= \max((\tau_{(n-1)1}^{v'} \cdot_{\mathcal{T}_1} \mu_{n1}^{v'w}, \tau_{(n-1)2}^{v'} \cdot_{\mathcal{T}_2} \tau_2'', (\tau_{(n-1)1}^{v_1} \cdot_{\mathcal{T}_1} \mu_{n1}^{v_1 w}, \tau_2 \cdot_{\mathcal{T}_2} \tau_2')) \\ &= \begin{pmatrix} \max_{\mathcal{T}_1}(\tau_{(n-1)1}^{v'} \cdot_{\mathcal{T}_1} \mu_{n1}^{v'w}, \tau_{(n-1)1}^{v_1} \cdot_{\mathcal{T}_1} \mu_{n1}^{v_1 w}), \\ \max_{\mathcal{T}_2}(\tau_{(n-1)2}^{v'} \cdot_{\mathcal{T}_2} \tau_2'', \tau_2 \cdot_{\mathcal{T}_2} \tau_2') \end{pmatrix}\end{aligned}$$

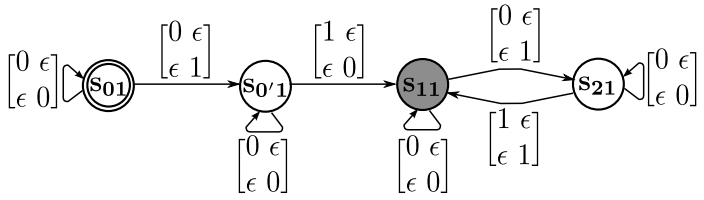
The first component of τ_n^w is always $\max_{\mathcal{T}_1}(\tau_{(n-1)1}^{v'} \cdot_{\mathcal{T}_1} \mu_{n1}^{v'w}, \tau_{(n-1)1}^{v_1} \cdot_{\mathcal{T}_1} \mu_{n1}^{v_1 w})$ in either case which is exactly equivalent to the tag concatenation of τ_{n1}^w, hence $\vec{\tau}_n|_{V_1,\mathcal{T}_1} = \vec{\tau}_{n1}$. Likewise, we can prove also $\vec{\tau}_n|_{V_2,\mathcal{T}_2} = \vec{\tau}_{n2}$.

□

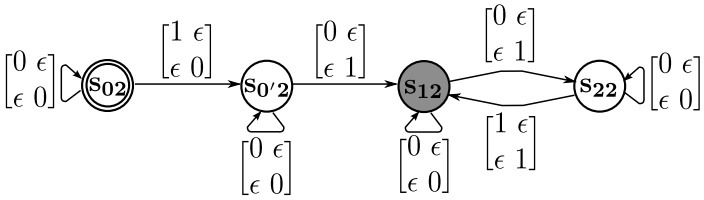
5.3 Self-synchronizing TMs and Composition Completeness

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

Example 5. Figure 5.4 shows two different TMs representing the same set of behaviors as Σ_1, yet their composition does not accept Σ_1. This is because the TMs cannot synchronize on updating their variable tags.



(a) TM accepting Σ_1



(b) A different TM accepting Σ_1

Figure 5.4: Interoperable TMs accepting Σ_1

As demonstrated in Example 5, the TM composition represents incompletely the tag system composition when M_i fails to include all representations of some behavior. Therefore, it is a natural condition for completeness that M_i be *self-synchronizing*. The notion of self-synchronization for homogeneous composition [6] requires the label semantics $L(M)$ of $M = (V, \mathcal{T}, S, s_0, F, E)$ to be $\sim$-closed, where $\sim$ is a binary relation such that $\forall \omega, \omega' \in L(V, \mathcal{T})^* : \omega \sim \omega' \Leftrightarrow \sigma(\omega) = \sigma(\omega')$.

In order to make machine M_i self-synchronize, all possible runs of any behavior of M_i must be added to the machine. In order for the TM semantics to remain unaffected, any sub-run's behavior should be excluded from the machine language. This, however, is not guaranteed by the original TM definition [6] because TMs there do not have accepting states. Our TM definition (Definition 5) cares for such a need and thus can preserve the TM semantics under the self-synchronizing operation.

Self-synchronization is however not sufficient to guarantee completeness of composition, as two unifiable behaviors do not always have unifiable representations, which in turn is caused by the choice of tag structures and morphisms. We recall that events of a variable v are indexed into a sequence

of $(v, 1, \tau_1, d_1), (v, 2, \tau_2, d_2) \ldots, (v, n, \tau_n, d_n)$ where $\tau_1 \leq \tau_2 \ldots \leq \tau_n$. If the tag increments between two successive events can always be mapped into the same increment under morphisms ρ_i, then completeness of composition is ensured by the following theorem.

Theorem 2. *Let σ_1 and σ_2 be accepted respectively by M_1 and M_2 s.t.:*

i) M_1 and M_2 are self-synchronizing,

ii) $\forall i \in \{1, 2\}, \forall (v, j, \tau_{ji}, d_{ji}) \in \sigma_i, \exists \delta_{ji} \in \mathcal{T}_i$:

$$\tau_{ji} = \tau_{(j-1)i} \cdot_{\mathcal{T}_i} \delta_{ji} \text{ where } \tau_{0i} \stackrel{\text{def}}{=} \hat{\imath}_{\mathcal{T}_i},$$

iii) $\forall i \in \{1, 2\}, \forall \delta_i \in \mathcal{T}_i, \forall (\tau_1, \tau_2), (\tau_1', \tau_2') \in \mathcal{T}_1 \,{}_{\rho_1}\!\times_{\rho_2} \mathcal{T}_2$:

$$\tau_i' = \tau_i \cdot \delta_i \Rightarrow \exists \delta_{3-i} \in \mathcal{T}_{3-i} : \begin{pmatrix} (\delta_1, \delta_2) \in \mathcal{T}_1 \,{}_{\rho_1}\!\times_{\rho_2} \mathcal{T}_2 \ \wedge \\ (\tau_1', \tau_2') = (\tau_1, \tau_2) \cdot (\delta_1, \delta_2) \end{pmatrix}.$$

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

Proof. We define the greatest number n_i to be $\max(\mathfrak{enum}_{\sigma_i}(v_i))$ where $\mathfrak{enum}_{\sigma_i}(v_i)$ is the total number of events on variable $v_i \in V_i$ of behavior σ_i.

We then build a sequence of labels $\omega_1 = \mu_{11}\mu_{21} \ldots \mu_{n_1 1}$ where the diagonal entries of μ_{j1} are δ_{j1} specified in item (ii) and the entries outside the diagonal are all $\epsilon_{\mathcal{T}_1}$. At the end, we pad ω_1 with $\max(n_1, n_2) - n_1$ stuttering labels where the diagonal entries are the identity element $\hat{\imath}_{\mathcal{T}_1}$ and entries outside the diagonal are $\epsilon_{\mathcal{T}_1}$.

We next build $\omega_2 = \mu_{12}\mu_{22} \ldots \mu_{n_2 2}$ where the diagonal entries of μ_{j2} are δ_{j2} specified in item (iii) and other entries are all $\epsilon_{\mathcal{T}_2}$. We also pad ω_2 with $\max(n_1, n_2) - n_2$ stuttering labels where the diagonal entries are $\hat{\imath}_{\mathcal{T}_2}$ and other entries are $\epsilon_{\mathcal{T}_2}$.

Since $\sigma(\omega_i) = \sigma_i$, the self-synchronizing condition guarantees that $\omega_i \in L(M_i)$ and ensures the existence of a valid run r_i in M_i over ω_i. By Theorem 2. (iii), it is obvious that $\mu_{j1} \ {}_{\rho_1}\bowtie_{\rho_2} \ \mu_{j2}$ is true, and this implies that there exists a run r in $M_1 \ {}_{\rho_1}\|_{\rho_2} \ M_2$ over the following sequence of labels $\omega = (\mu_{11} \ {}_{\rho_1}\sqcup_{\rho_2} \mu_{12}) \ldots (\mu_{\max(n_1,n_2)1} \ {}_{\rho_1}\sqcup_{\rho_2} \mu_{\max(n_1,n_2)2})$ where $\sigma(r)|_{V_1 \cap V_2} = \sigma|_{V_1 \cap V_2}$ since $\sigma(r)|_{V_1,\mathcal{T}_1} = \sigma_1$ and $\sigma(r)|_{V_2,\mathcal{T}_2} = \sigma_2$, $\sigma(r)|_{V_1 \setminus V_2} = \sigma|_{V_1 \setminus V_2}$ and $\sigma(r)|_{V_2 \setminus V_1} = \sigma|_{V_2 \setminus V_1}$ because of condition (iii), thus $\sigma(r) = \sigma$. □

5.4 An Automotive Case Study

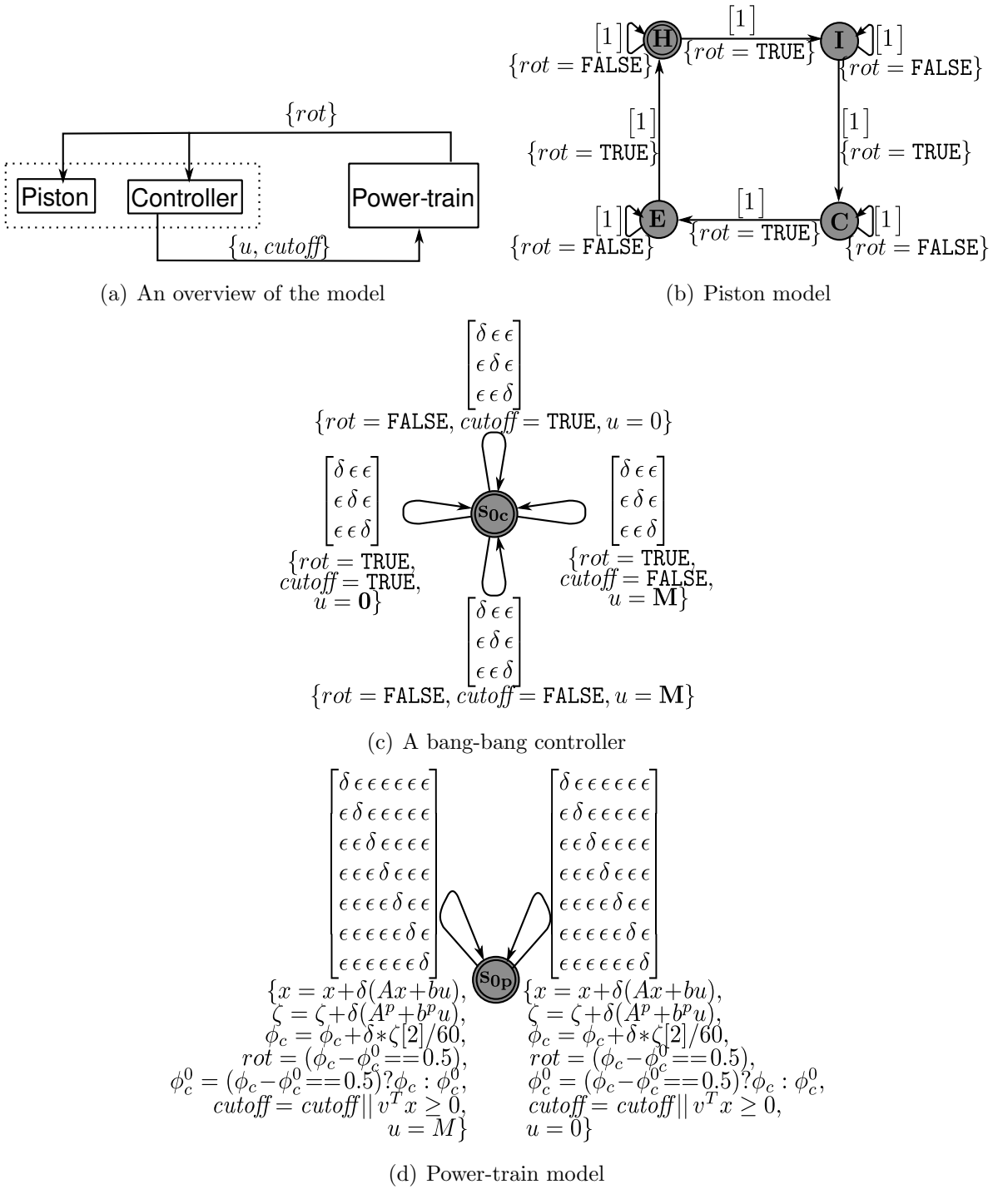
In this section, we demonstrate the application of TM composition in dealing with the problem of heterogeneity in a practical use case of an automotive system.

5.4.1 Description

We consider a simplified version of the automotive engine control model proposed by Balluchi et al. [3]. It consists of a sub-model capturing the piston sequential behavior and another capturing the power-train behavior under a cut-off control policy. The goal of the control policy is to reduce the unpleasant oscillations when the driver releases the gas pedal and requests no torque to the engine. The piston model is naturally expressed by a finite state machine while the power-train model can be represented by a continuous time equation. Thus, in order to understand the behavior of the car, it is important for designers to be able to compose these heterogeneous models.

A piston abstractly cycles through four phases:

i) the *intake* (I) phase during which the piston loads the air-fuel mix $q \in \mathbb{R}_+$;



(a) An overview of the model

(b) Piston model

(c) A bang-bang controller

(d) Power-train model

Figure 5.5: An automotive engine control model

ii) the *compression* (C) phase in which the loaded mix is compressed;

iii) the *expansion* (E) phase in which the compressed mix is combusted,

producing the spark ignition;

iv) the *exhaust* (H) phase during which the piston expels combustion exhaust gases.

We assume the torque evolution $u(t)$ to be a piece-wise constant function which is $u(t) = 0$ everywhere except in the E-phase where $u(t) = G * q$ and when the spark ignition is set with G is the mix-to-torque gain. The continuous time power-train behavior is modeled by the following linear system:

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

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

In this use case, we model the bang-bang control law [3] where the fuel injection is cut when $v^T x \geq 0$. The reduced state $x = [x_1, x_2]^T$ represents the system's oscillation component and is obtained by applying the state transformation $\dot{x}(t) = Ax(t) + bu(t)$ where $A = \left[\begin{smallmatrix} \lambda & -\mu \\ \mu & \lambda \end{smallmatrix}\right]$ and $\lambda \pm j\mu$ are the conjugate complex poles of A^p. The oscillation acceleration can then be computed based on x as $a(t) = cx(t)$. The starting point of the cut-off control horizon is the time at which all the loaded cylinders' potential torques are at the steady value $M = G * q^0$.

The piston model needs only information about the sequencing of events, while the power-train model requires the exact timing of events. Thus we can use $\mathcal{T}_1 = (\mathbb{N} \cup \{\epsilon_1\}, +)$ and $\mathcal{T}_2 = (\mathbb{R}_+ \cup \{\epsilon_2\}, +)$ where $\epsilon_1 = \epsilon_2 = -\infty$, to model them. Figure 5.5(b) shows the piston behavior where the transitions of the automaton occur when a piston reaches the bottom or top dead point, i.e. when the flag *rot* is set. Figure 5.5(c) and 5.5(d) describe the

bang-bang control policy and the evolution of the power-train, respectively. We have approximated the power-train using the forward integration Euler method with a step of δ. The state variables are fixed as follows [3]:

$$A^p = \begin{bmatrix} 0 & 1 & -7.556 \\ -448.1 & -5.186 & 30.87 \\ 3.042 & 0.02773 & -0.2105 \end{bmatrix}, b^p = \begin{bmatrix} 0 \\ 15.05 \\ 0 \end{bmatrix},$$

$$A = \begin{bmatrix} -2.671 & -21.54 \\ 21.54 & -2.671 \end{bmatrix}, b = \begin{bmatrix} 1.92339 \\ -14.32309 \end{bmatrix}, v = \begin{bmatrix} 0.01 \\ -1 \end{bmatrix},$$

$$c = \begin{bmatrix} 0.0379945 & -0.00257 \end{bmatrix}, M = 12.41, \delta = 0.001181.$$

5.4.2 Tag Machine Script Language

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

(a) A piston TM

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

(b) Its morphism

Figure 5.6: High-level description of a piston TM and its morphism

We have implemented a prototype tool to simulate the heterogeneous TM composition under morphisms. Our simulator, written in approximately 5000 lines of C++ code, supports a high level script language to specify TMs. Each TM is described as a module consisting of a set of

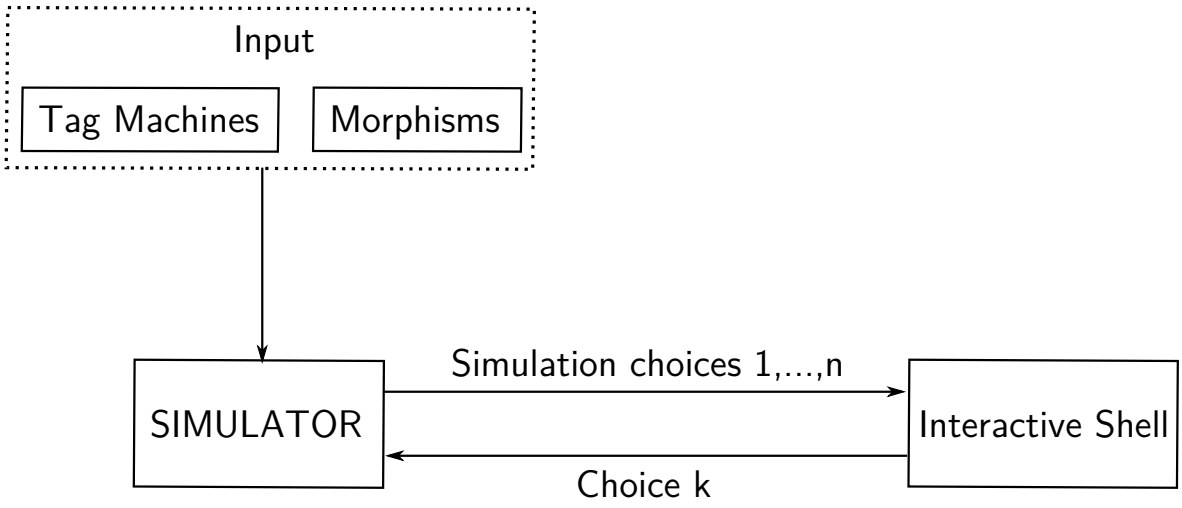


Figure 5.7: Basic schema of TM simulator

constraints on the declared variables. In particular, a module must contain declarations about the machine tag structure (`TAGSTRUCT`), the state (`MSTATE`) and label (`MLABEL`) variable. Declarations about the machine variables (`MVAR`), initial state (`INIT`), accepting states (`FINAL`) and transitions (`EDGE`) are optional. A morphism associated with a machine can be declared likewise. Figure 5.6 shows an example of our script language.

5.4.3 Tag Machine Simulator

The inputs to our simulator are TMs and morphisms under which the TMs can be composed as shown in Figure 5.7. Our simulator performs step-by-step exploration on composition of TM transitions. Specifically, the TM simulator keeps track of the tag vector and value assignment of all variables of the TMs. At every simulation step, the simulator looks for transitions where the tag vectors and value assignments agree with each other over the shared variables as detailed in Section 5.1. It then exposes through an interactive shell to users all the possible choices for the next transitions which can be carried out randomly or interactively. In a random mode, the choice of transition is done by the simulator whereas such choice is

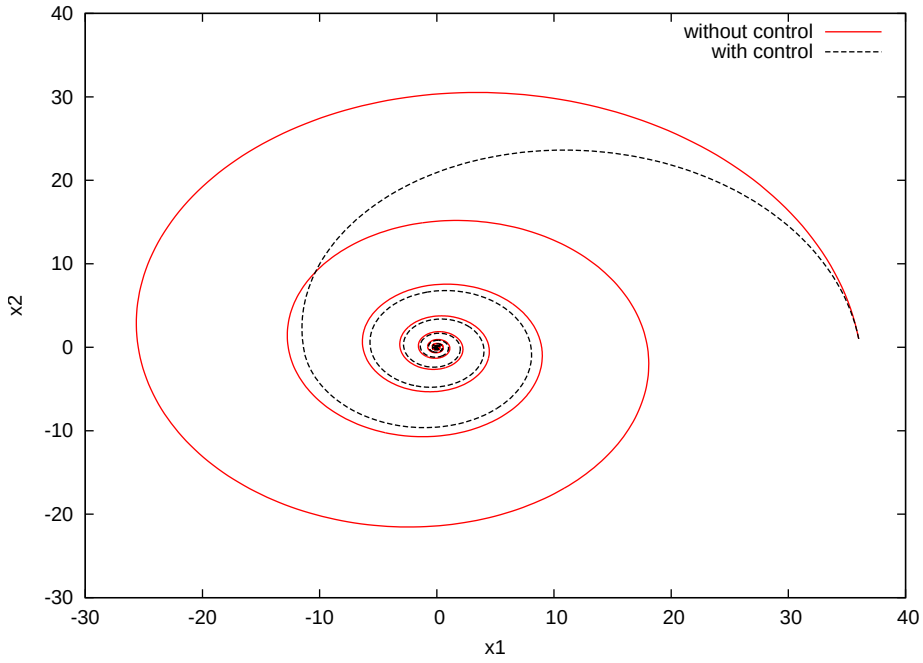
```
>>>>> Simulation starting...

************* AVAILABLE CHOICES *************

    ============================ CHOICE 0 ============================
      TM Torque :
                | 0.001181 EP       EP       EP       EP       EP       EP       EP       EP       EP       EP       |
                | EP       0.001181 EP       EP       EP       EP       EP       EP       EP       EP       EP       |
                | EP       EP       0.001181 EP       EP       EP       EP       EP       EP       EP       EP       |
                | EP       EP       EP       0.001181 EP       EP       EP       EP       EP       EP       EP       |
                | EP       EP       EP       EP       0.001181 EP       EP       EP       EP       EP       EP       |
                | EP       EP       EP       EP       EP       0.001181 EP       EP       EP       EP       EP       |
                | EP       EP       EP       EP       EP       EP       0.001181 EP       EP       EP       EP       |
                | EP       EP       EP       EP       EP       EP       EP       0.001181 EP       EP       EP       |
                | EP       EP       EP       EP       EP       EP       EP       EP       0.001181 EP       EP       |
                | EP       EP       EP       EP       EP       EP       EP       EP       EP       0.001181 EP       |
                | EP       EP       EP       EP       EP       EP       EP       EP       EP       EP       0.001181 |
                           {x11=x11+0*(0*x11+0*x21+0*torque);
                            x21=x21+0*(0*x11+0*x21+0*torque);
                            sm11=sm11+0*(0*sm11+0*sm21+0*sm31)+0*torque;
                            sm21=sm21+0*(0*sm11+0*sm21+0*sm31)+0*torque;
                            sm31=sm31+0*(0*sm11+0*sm21+0*sm31)+0*torque;
                            fai=fai+0*sm21/60;
                            pfai=(fai-pfai>=0&&fai-pfai<=0)*fai+(1-(fai-pfai>=0&&fai-pfai<=0))*pfai;
                            rot=fai-pfai>=0&&fai-pfai<=0;
                            cutoff=cutoff||0*x11+0*x21>=0;
                            torque=0;
                            aw=0*x11+0*x21}
           s0-------------------------------------------------------------------------------------------------------------------->s0
         + tag structure - MAXPLUSFLOAT
         + tag vector - [0,0,0,0,0,0,0,0,0,0,0]
         + variable vector - {x11:36;x21:1;sm11:-4.5;sm21:2.7e+03;sm31:3.5e+02;fai:0;pfai:0;rot:FALSE;cutoff:FALSE;torque:12;aw:1.4}
      TM Control :
                  | 1  EP EP |
                  | EP 1  EP |
                  | EP EP 1  |
                 {rot=FALSE;
                  cutoff=FALSE;
                  torque=0}
           s0------------------->s0
         + tag structure - MAXPLUSINTEGER
         + tag vector - [0,0,0]
         + variable vector - {rot:FALSE;cutoff:FALSE;torque:12}
      TM Piston :
                   | 1 |
                {rot=FALSE}
           H----------------->H
         + tag structure - MAXPLUSINTEGER
         + tag vector - [0]
         + variable vector - {rot:FALSE}


      Pick a choice from the above (0-0): 
```

Figure 5.8: Basic schema of TM simulator

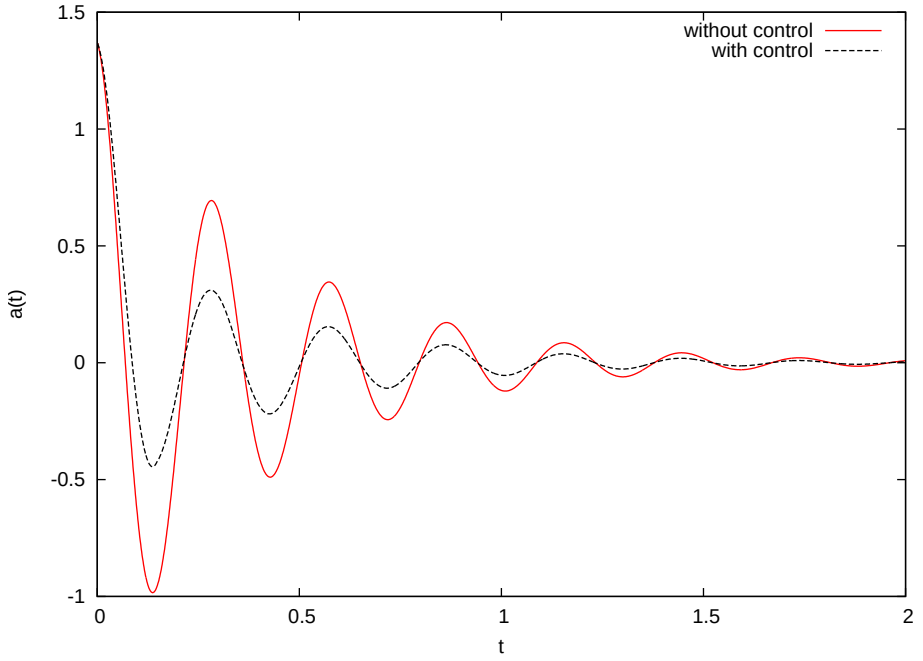
provided by users in an interactive mode. Figure 5.8 shows an example of the initial simulation choices for the automotive model in an interactive mode.

5.4.4 Evaluation

We have described the cut-off problem using our prototype TM-simulator and performed simulations of 2000 steps to evaluate the effect of the cut-off control on the oscillation acceleration. We present here the results of two



(a) $x(t)$



(b) $a(t)$

Figure 5.9: The evolutions of $x(t)$ and $a(t)$ without and with control

such simulations on the above set of parameters and under the presence of morphisms such as $\rho_i : \mathcal{T}_i \mapsto \mathcal{T}_2$ given by $\rho_1(\tau_1) = \delta * \tau_1$ and $\rho_2(\tau_2) = \tau_2$.

Consistent with the previous result [3], Figure 5.9 shows that better acceleration peaks and driving comfort are obtained when cut-off control is enforced. This consistent result can assist automotive engineers and designers in estimating the impact of the control implementation on the engine performance, thereby justifying and selecting suitable system parameter values to obtain a desirable performance.

Another important advantage offered by the heterogeneity methodology is in terms of component modeling. While the components in [3] had to be modeled in the same domain in order to solve the cut-off control problem, they can now be expressed in their natural domain. The component interaction can then be precisely quantified and modeled by means of morphisms. This advantage becomes much more crucial when designers have to deal with large and complex systems.

Chapter 6

Tag Contracts

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 extra operators and relations such as refinement, quotient and conjunction to relate their sets of behaviors (Section 6.1). 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 (Section 6.2).

6.1 Tag Machine Operators

6.1.1 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 $\rho_i : \mathcal{T}_i \mapsto \mathcal{T}$ be algebraic tag morphisms, where $i \in \{1, 2\}$. The TM refinement is defined as follows.

Definition 10. 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 exist $(s_2, \mu_2, s_2') \in E_2$ such that

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

The following theorem shows that our TM theory supports (homogenous) *independent implementability*[29]: refinement is preserved when composing components.

Theorem 3. *Let 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 3 only holds for *homogenous* 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 6. We consider an example where

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

Let M_i, M_i' be defined on $\mathcal{T}_i$ and V_i where $i \in \{1, 2\}$. For the sake of simplicity, assume all TMs have a single state which is both initial and accepting state. In addition, there is only one self-loop at this state annotated with 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 $M_1 \ {}_{\rho_1}\preceq_{\rho_2} M_2$ since $\mu_1 \ {}_{\rho_1}\bowtie_{\rho_2} \mu_2$ and $M_1' \ {}_{\rho_1}\preceq_{\rho_2} M_2'$ since $\mu_1' \ {}_{\rho_1}\bowtie_{\rho_2} \mu_2'$. However, $(M_1 \parallel M_1') \ {}_{\rho_1}\not\preceq_{\rho_2} (M_2 \parallel M_2')$ since the right composition is empty while the left is not.

6.1.2 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 computes the most general specification that retains all unifiable behaviors of two TMs. The dual operator to TM composition is TM quotient which computes the maximal specification as follows.

Definition 11. The quotient $M_1 \, {}_{\rho_1}/_{\rho_2} M_2$ is $M = (V, \mathcal{T}_{12}, S, s_0, F, E)$, where:

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

The dual relation between composition and quotient is presented in the next theorem.

Theorem 4. *Quotient M satisfies refinement $(M \ {}_{\mathsf{proj}_2}\|_{\mathsf{id}_2} M_2) \ {}_{\mathsf{proj}'_1}\preceq_{\mathsf{id}_1} M_1$ where:*

$$\forall i \in \{1,2\}, \forall \tau_i \in \mathcal{T}_i : \mathsf{id}_i(\tau_i) = \tau_i \tag{6.1}$$

$$\forall i \in \{1,2\}, \forall (\tau_1,\tau_2) \in \mathcal{T}_{12} : \mathsf{proj}_i((\tau_1,\tau_2)) = \tau_i \tag{6.2}$$

$$\forall (\tau_{12},\tau_2) \in \mathcal{T}_{12} \ {}_{\mathsf{proj}_2}\times_{\mathsf{id}_2} \mathcal{T}_2 : \mathsf{proj}'_1((\tau_{12},\tau_2)) = \mathsf{proj}_1(\tau_{12}) \tag{6.3}$$

$$\forall (\tau_{12},\tau_1) \in \mathcal{T}_{12} \ {}_{\mathsf{proj}_1}\times_{\mathsf{id}_1} \mathcal{T}_1 : \mathsf{proj}'_2((\tau_{12},\tau_1)) = \mathsf{proj}_2(\tau_{12}) \tag{6.4}$$

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

$$(M' \ {}_{\mathsf{proj}_2}\|_{\mathsf{id}_2} M_2) \ {}_{\mathsf{proj}'_1}\preceq_{\mathsf{id}_1} M_1 \Rightarrow M' \preceq M. \tag{6.5}$$

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.

Initially, $(((s_{01}, s_{02}), s_{02}), s_{01}) \in R$. If there is a transition from 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.

Assuming there exists some runs r' in M' where the last transition $s'_n \xrightarrow{\mu_n}$ cannot be matched by M. There are two cases that can happen, r' can unify fully with some run r_2 in M_2 or partially with every such run. In the first case, the composition of r' and r_2 then refines some run r_1 in M_1. The existence of r_1 and r_2 together implies the existence of a run r 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 in Theorem 4. This universal property is generally expected of quotients [4], and it alone implies that the quotient is uniquely defined up to two-sided homogeneous refinement [28]. As an example, Figure 6.2(c) shows a homogeneous quotient.

6.1.3 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.

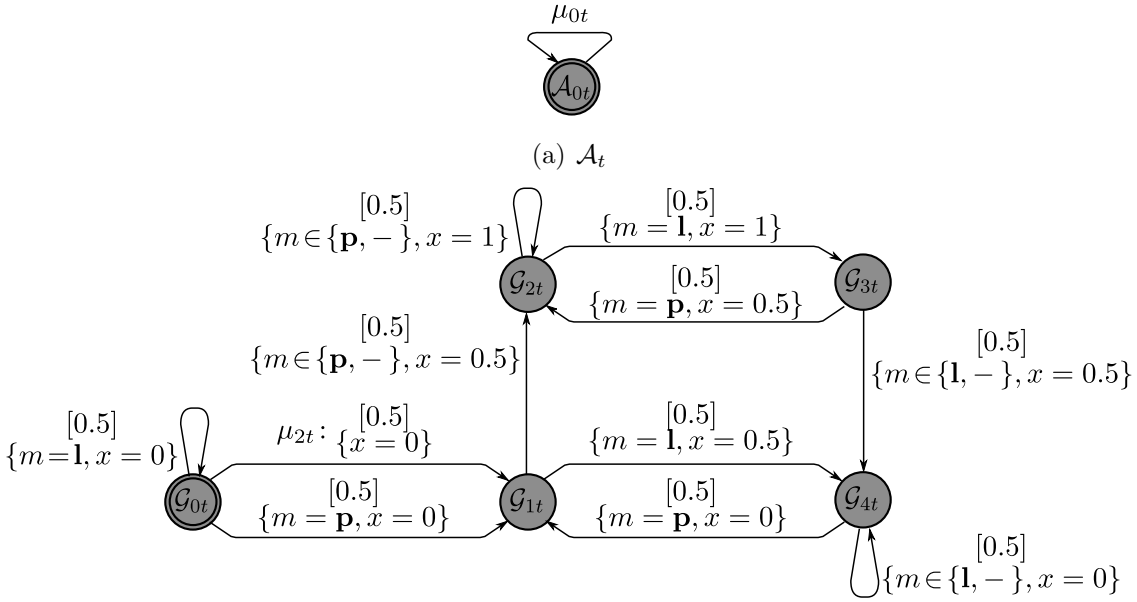
6.2 Tag Contracts

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 12. 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 7. We consider the simplified water controlling system in Example 4 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.

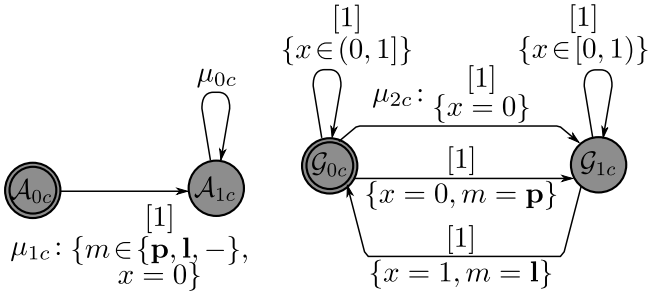
Figure 6.1 depicts the tank contract $\mathcal{C}_t = (\mathcal{A}_t, \mathcal{G}_t)$ which guarantees a linear evolution of the water level $x(t)$ upon receiving controlling commands. The controller contract is shown Figure 6.2, where it assumes the tank to be empty initially (Figure 6.2(a)), i.e., $x = 0$ and places no requirement on its output which is the command signal. As long as such assumption is satisfied, the controller guarantees (Figure 6.2(b)) to send a proper command upon knowing of the tank emptiness or fullness.



(a) $\mathcal{A}_t$

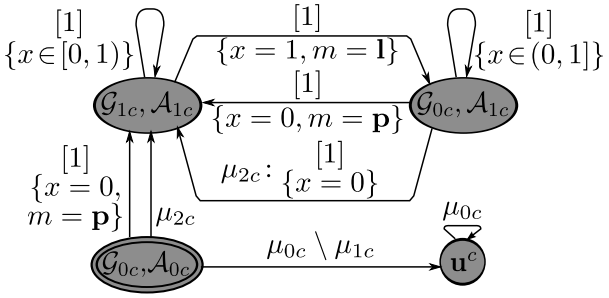
(b) $\mathcal{G}_t$

Figure 6.1: The tank contract



(a) $\mathcal{A}_c$

(b) $\mathcal{G}_c$



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

Figure 6.2: The controller contract

Similar to Example 4, 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 $\mathcal{T}_2 = (\mathbb{N} \cup \{\epsilon_2\}, +)$, in Example 4 to describe the tank and controller contracts respectively. We also use the same morphisms $\rho_1 : \mathcal{T}_1 \mapsto \mathcal{T}_1$ and $\rho_2 : \mathcal{T}_2 \mapsto \mathcal{T}_1$ given by $\rho_1(\tau_1) = \tau_1$, $\rho_2(\tau_2) = 0.5 * \tau_2$ 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 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 12. 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 $\forall \mathcal{E} \in [\![\mathcal{C}]\!]_{\mathsf{e}} : \mathcal{I} \parallel \mathcal{E} \preceq \mathcal{G} \parallel \mathcal{E}$. The set of implementations is similarly denoted by $[\![\mathcal{C}]\!]_{\mathsf{p}}$. Hence, the implementation checking is done based on instantiating all possible environments of a contract. 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

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

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

Proof. By contraposition, assuming that $\mathcal{G} \not\preceq \mathcal{G}/\mathcal{A}$. There are two possible cases. In one case, there must exist run r_g in $\mathcal{G}$ and r in $\mathcal{G}/\mathcal{A}$:

$$r_g : s_{0g} \xrightarrow{\mu_1} s_{1g} \dots s_{(n-1)g} \xrightarrow{\mu_n} s_{ng}$$
$$r : s_0 \xrightarrow{\mu_1} s_1 \dots s_{n-1} \xrightarrow{\mu_n}$$

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, $s_i = (s_{ig}, s_{ia})$ by the definition of quotient. However, $s_{n-1} \xrightarrow{\mu_n}$ implies that $s_{(n-1)g} \xrightarrow{\mu_n}$ and $s_{(n-1)a} \xrightarrow{\mu_n}$. The former implication obviously contradicts the existence of the last transition of r_g. In the other 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 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 5. *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 $r_{\bar{\mathcal{G}}}$.

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

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, $s_{k\bar{\mathcal{G}}} \equiv s'_{k\bar{\mathcal{G}}}$ 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) $s_{nq} \equiv \mathfrak{u}_q$: this implies $s_{n\bar{\mathcal{G}}} \equiv \mathfrak{u}_{\bar{\mathcal{G}}}$ 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, $(\mathcal{G} \| \mathcal{E}) \preceq (\bar{\mathcal{G}} \| \mathcal{E})$ as $\mathcal{G} \preceq (\mathcal{G}/\mathcal{A}) \equiv \bar{\mathcal{G}}$. Second, $(\bar{\mathcal{G}} \| \mathcal{E}) \preceq (\mathcal{G} \| \mathcal{E})$ 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 8. We use the tag contracts in Example 7 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., $\mathcal{G}_t/\mathcal{A}_t \equiv \mathcal{G}_t$. Figure 6.2(c), on the other hand, shows the normalized controller guarantee having more behaviors than the unnormalized one. Relying on the special clock $\mathfrak{inc}$ to restructure the tank and controller machines in Example 4, it is easy to see that the tank machine refines $\mathcal{G}_t$ while the controller machine does not refine $\mathcal{G}_t/\mathcal{A}_t$. Therefore, the tank machine is an implementation of the tank contract while the controller machine is not an implementation of the controller 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.

6.2.1 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 $\rho_i : \mathcal{T}_i \mapsto \mathcal{T}$ be algebraic tag morphisms where $i \in \{1, 2\}$.

Definition 14. 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 6. $\mathcal{C}_1 \ {}_{\rho_1}\preceq_{\rho_2} \mathcal{C}_2 \Leftrightarrow (\mathcal{A}_2 \ {}_{\rho_2}\preceq_{\rho_1} \mathcal{A}_1) \wedge (\mathcal{G}_1^{\mathfrak{n}} \ {}_{\rho_1}\preceq_{\rho_2} \mathcal{G}_2^{\mathfrak{n}})$

Proof.

- $\Rightarrow$: $\mathcal{C}_1 \ {}_{\rho_1}\preceq_{\rho_2} \mathcal{C}_2$ and $\mathcal{G}_1^{\mathfrak{n}} \in [\![\mathcal{C}_1]\!]_{\mathsf{p}}$ together implies

  $$\exists \mathcal{I}_2 \in [\![\mathcal{C}_2]\!]_{\mathsf{p}} : \mathcal{G}_1^{\mathfrak{n}} \ {}_{\rho_1}\preceq_{\rho_2} \mathcal{I}_2,$$

  by the second condition of Definition 14. Since $\mathcal{I}_2 \preceq \mathcal{G}_2^{\mathfrak{n}}$, we can infer that $\mathcal{G}_1^{\mathfrak{n}} \ {}_{\rho_1}\preceq_{\rho_2} \mathcal{G}_2^{\mathfrak{n}}$. Using a similar line of reasoning, we can also infer that $\mathcal{A}_2 \ {}_{\rho_2}\preceq_{\rho_1} \mathcal{A}_1$.

- $\Leftarrow$: Since $\mathcal{C}_1$ is assumed to be normalized, it follows that

  $$\forall \mathcal{I}_1 \in [\![\mathcal{C}_1]\!]_{\mathsf{p}} : \mathcal{I}_1 \preceq \mathcal{G}_1^{\mathfrak{n}}.$$

  Together with the fact of $\mathcal{G}_1^{\mathfrak{n}} \ {}_{\rho_1}\preceq_{\rho_2} \mathcal{G}_2^{\mathfrak{n}}$, this implies $\mathcal{I}_1 \ {}_{\rho_1}\preceq_{\rho_2} \mathcal{G}_2^{\mathfrak{n}}$. Using a similar line of reasoning, we can also deduce that $\mathcal{E}_2 \ {}_{\rho_2}\preceq_{\rho_1} \mathcal{A}_1$ for any environment $\mathcal{E}_2$ of contract $\mathcal{C}_2$.

□

6.2.2 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 [4].

Definition 15. 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 (6.1), (6.2), (6.3), (6.4) of Theorem 4.

These dominance conditions are shown to be equivalent to simpler formulas in the following theorem.

Theorem 7. *Condition* (i) *is equivalent to condition* ***hDC-1)*** *and condition* (ii) *equivalent to the conjunction of* ***hDC-2a)*** *and* ***hDC-2b)*** *:*

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

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

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

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

- $\Rightarrow$: Let $\mathcal{I}_i = \mathcal{G}_i^{\mathfrak{n}}$, then:

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

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

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

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

□

6.2.3 Tag Contract Composition

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

Definition 16. 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 $\mathsf{swap} : \mathcal{T}_2 \ {}_{\rho_2}\times_{\rho_1} \mathcal{T}_1 \mapsto \mathcal{T}_1 \ {}_{\rho_1}\times_{\rho_2} \mathcal{T}_2$ is such that $\mathsf{swap}((\tau_2, \tau_1)) = ((\tau_1, \tau_2))$ and M_{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 8. *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 16.

b) $\mathcal{I}_i \in [\![\mathcal{C}_i]\!]_{\mathsf{p}} \Rightarrow \mathcal{I}_i \preceq \mathcal{G}_i^{\mathfrak{n}}$ (by Theorem 5). 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 4. 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 9. *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 15 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 8. □

6.2.4 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* $\pi = (V^{\mathsf{i}}, V^{\mathsf{o}})$ which is a partition of its variables into inputs and outputs, i.e. $V = V^{\mathsf{i}} \cup V^{\mathsf{o}}$ and $V^{\mathsf{i}} \cap V^{\mathsf{o}} = \emptyset$. 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., $V_1^{\mathsf{o}} \cap V_2^{\mathsf{o}} = \emptyset$. The composite contract profile is then $\pi = ((V_1^{\mathsf{i}} \cup V_2^{\mathsf{i}}) \setminus (V_1^{\mathsf{o}} \cup V_2^{\mathsf{o}}), V_1^{\mathsf{o}} \cup V_2^{\mathsf{o}})$.

Example 9. The tank and controller contracts in Example 7 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:

- $\mathcal{A}_e \equiv M_{\mathfrak{u}}$, c.f. Figure 6.3, 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 $(\mathcal{A}_e/\mathcal{G}^{\mathfrak{n}}) \between (\mathcal{A}/\mathcal{G}_e^{\mathfrak{n}}) \equiv M_{\mathfrak{u}}$ 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 $\mathcal{A}_e \equiv M_{\mathfrak{u}}$, thus the condition of $(\mathcal{A}_e/\mathcal{G}^{\mathfrak{n}}) \between (\mathcal{A}/\mathcal{G}_e^{\mathfrak{n}}) \equiv M_{\mathfrak{u}}$ 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 10. We consider again the water tank controlling problem in Example 7 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

Figure 6.3: M_{u}

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.

Chapter 7

Contract Synthesis

Component-based and contract-based design has been shown to be a rigorous and effective approach for designing concurrent systems [43, 4, 42, 21]. Different components of the same system can be developed by different teams in an independent and concurrent manner provided that their associated contracts can synchronize and satisfy predefined properties. The separation between assumptions and guarantees allows an efficient reuse of already-designed components, thereby supporting the distributed development of complex systems effectively.

Components can be formed by a bottom-up composition of simpler predefined components. They can alternatively be formed by a top-down decomposition into sub-components defined by a set of sub-contracts, as long as the composition of the sub-contracts satisfies or refines the contract of the intended component. When this condition is not satisfied, i.e., the sub-contract composition does not refine the overall contract, designers must refine the sub-contract specifications until the system is proved correct. In this chapter, we deal with the problem of checking if a contract $\mathcal{C}$ can be decomposed into a set of n homogeneous contracts or of 2 heterogeneous contracts. We also address the problem of synthesizing the contract set in order to make their composition refine $\mathcal{C}$ when necessary. In particular,

we study *decomposing conditions* under which the contract decomposition can be verified, and thereby proposing a generic *synthesis strategy* for fixing wrong decompositions. We present in this chapter contract synthesis strategies for homogeneous and heterogeneous systems.

7.1 Homogeneous Contract Synthesis

For our formalization we follow the notation introduced by Bauer et al. [4] which is built on top of a specification theory equipped with a refinement ($\preceq$) and a composition ($\|$) operator. Note that these operators are *meta-theoretical* or uninterpreted operators, meaning that we do not need to know their exact semantics as long as they satisfy certain properties [10]. In particular, monotonicity:

$$(S' \preceq S) \wedge (T' \preceq T) \Rightarrow (S' \parallel T') \preceq (S \parallel T).$$

In addition, composition is commutative and associative while refinement is reflexive and transitive. Two other operators that can be defined on top of composition and refinement are quotient ($/$):

$$((S \parallel (T/S)) \preceq T) \wedge ((S \parallel R) \preceq T \Rightarrow R \preceq T/S)$$

and conjunction ($\curlywedge$):

$$((S \curlywedge T) \preceq S) \wedge ((S \curlywedge T) \preceq T) \wedge (R \preceq S \wedge R \preceq T \Rightarrow R \preceq (S \curlywedge T))$$

While the refinement operator can relate *concrete* and *abstract* specifications, the composition and quotient, which are dual to each other, can combine specifications to create new ones. In particular, the conjunction operator computes the greatest lower bound in the refinement order of the original specifications.

Assuming the existence of such underlying specification theory, we recall that a contract of a component can be defined formally as a pair of specifications, i.e., *assumptions* and *guarantees*: $\mathcal{C} = (\mathcal{A}, \mathcal{G})$. The specification

$\mathcal{A}$ expresses what is constrained on the environments of the component and the specification $\mathcal{G}$ describes what the component can guarantee given the assumption satisfaction. An implementation of the component satisfies its contract whenever it satisfies the contract guarantee, subject to the contract assumption. The contract semantics is therefore defined through the notions of such environments and implementations. An environment $\mathcal{E}$ satisfies contract $\mathcal{C}$ when $\mathcal{E} \preceq \mathcal{A}$. Let $[\![\mathcal{C}]\!]_{\mathsf{e}}$ be the set of environments of $\mathcal{C}$, an implementation $\mathcal{I}$ satisfies contract $\mathcal{C}$ if $\forall \mathcal{E} \in [\![\mathcal{C}]\!]_{\mathsf{e}} : \mathcal{I} \parallel \mathcal{E} \preceq \mathcal{G} \parallel \mathcal{E}$ holds. We denote the set of all possible implementation similarly by $[\![\mathcal{C}]\!]_{\mathsf{p}}$.

Two contracts have identical semantics and are *equivalent* if they possess the same set of environments and implementations. Without loss of generality [4], we assume that for every contract $\mathcal{C} = (\mathcal{A}, \mathcal{G})$, there exists contract $\mathcal{C}^{\mathsf{n}} = (\mathcal{A}, \mathcal{G}^{\mathsf{n}})$ which is equivalent to $\mathcal{C}$ and where the implementation check can be done independently of the assumption presence. We call $\mathcal{C}^{\mathsf{n}}$ the normalized form of $\mathcal{C}$ and derive $\mathcal{G}^{\mathsf{n}}$ using the normalization operator $\rhd$ which can be defined on top of the basic operators $\preceq$, $\parallel$, $/$, $\curlywedge$: $\mathcal{G}^{\mathsf{n}} = \mathcal{G} \rhd \mathcal{A}$. In addition, the following holds:

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

A refinement relation between contracts can then be established based on that between their environment sets and implementation sets. Formally, contract $\mathcal{C}$ is said to refine $\mathcal{C}'$, written $\mathcal{C} \preceq \mathcal{C}'$, when it can accept more environments and fewer implementations than contract $\mathcal{C}'$:

$$[\![\mathcal{C}']\!]_{\mathsf{e}} \subseteq [\![\mathcal{C}]\!]_{\mathsf{e}} \wedge [\![\mathcal{C}]\!]_{\mathsf{p}} \subseteq [\![\mathcal{C}']\!]_{\mathsf{p}}.$$

7.1.1 Contract Composition

Composing contracts is formalized so that the compositionality between their implementations can be respected, i.e., composing such implementations results in an implementation of the composite contract. In addition,

every environment of the composite contract should be able to work with any implementation of an individual contract in a way that their composition does not violate the other contract assumption. In fact, there exists a class of contracts, including the composite contract, able to provide such desirable consequences. These are referred to as *dominating* contracts [4] and the composite contract is the least in the refinement order of all dominating contracts, as we shall see in Section 7.1.1.

This notion of dominance thus enables the compositionality of the implementation relation, an important principle in reusing components and decomposing systems into existing components. Before studying contract decomposition (Section 7.1.2), we first generalize the notion of dominance and composition from two contracts [4] to a set of n contracts.

Definition 17. A contract $\mathcal{C} = (\mathcal{A}, \mathcal{G})$ is said to *dominate* the contract set $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ if the following conditions hold:

i) $\forall \mathcal{I}_1 \in [\![\mathcal{C}_1]\!]_{\mathsf{p}}, \ldots, \forall \mathcal{I}_n \in [\![\mathcal{C}_n]\!]_{\mathsf{p}} : \parallel_{1 \leq i \leq n} \mathcal{I}_i \in [\![\mathcal{C}]\!]_{\mathsf{p}}$,

ii) $\forall \mathcal{E} \in [\![\mathcal{C}]\!]_{\mathsf{e}}, \forall \mathcal{I}_1 \in [\![\mathcal{C}_1]\!]_{\mathsf{p}}, \ldots, \forall \mathcal{I}_n \in [\![\mathcal{C}_n]\!]_{\mathsf{p}}, \forall 1 \leq i \leq n :$

$$\mathcal{E} \parallel \Big|\Big|_{1 \leq j \neq i \leq n} \mathcal{I}_j \preceq \mathcal{A}_i.$$

The following theorem reduces checking the two conditions in Definition 17 to checking simpler formulas.

Theorem 10. *Checking condition* (i) *is equivalent to checking*

$$\Big|\Big|_{1 \leq i \leq n} \mathcal{G}_i^{\mathfrak{n}} \in [\![\mathcal{C}]\!]_{\mathsf{p}},$$

and checking condition (ii) *is equivalent to checking*

$$\forall 1 \leq i \leq n : \mathcal{A} \parallel \Big|\Big|_{1 \leq j \neq i \leq n} \mathcal{G}_j^{\mathfrak{n}} \preceq \mathcal{A}_i.$$

Proof.

i) $\Rightarrow$: Consider $\mathcal{I}_i = \mathcal{G}_i^{\mathfrak{n}}$.

$\Leftarrow$: By normalization, $\mathcal{I}_i \preceq \mathcal{G}_i^{\mathfrak{n}}$ and therefore:

$$\begin{aligned} & \forall \mathcal{E} \in [\![\mathcal{C}]\!]_{\mathsf{e}} : (\mathcal{E} \parallel \bigparallel_{1 \leq i \leq n} \mathcal{I}_i) \preceq (\mathcal{E} \parallel \bigparallel_{1 \leq i \leq n} \mathcal{G}_i^{\mathfrak{n}}) \\ \Rightarrow & \forall \mathcal{E} \in [\![\mathcal{C}]\!]_{\mathsf{e}} : (\mathcal{E} \parallel \bigparallel_{1 \leq i \leq n} \mathcal{I}_i) \preceq (\mathcal{E} \parallel \mathcal{G}) \\ \Rightarrow & \bigparallel_{1 \leq i \leq n} \mathcal{I}_i \in [\![\mathcal{C}]\!]_{\mathsf{p}}. \end{aligned}$$

ii) $\Rightarrow$: Consider $\mathcal{E} = \mathcal{A}, \mathcal{I}_j = \mathcal{G}_j^{\mathfrak{n}}$.

$\Leftarrow$: By definition of environments, normalization and the composition-refinement relation:

$$(\mathcal{E} \parallel \bigparallel_{1 \leq j \neq i \leq n} \mathcal{I}_j) \preceq (\mathcal{A} \parallel \bigparallel_{1 \leq j \neq i \leq n} \mathcal{G}_j^{\mathfrak{n}}) \preceq \mathcal{A}_i.$$

□

The composition of a set of contracts can then be defined as follows.

Definition 18. The composition of a set of contracts $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$, written $\parallel_{1 \leq i \leq n} \mathcal{C}_i$, is the contract $\mathcal{C} = (\mathcal{A}, \mathcal{G}) = (\bigwedge_{1 \leq i \leq n} (\mathcal{A}_i / \parallel_{1 \leq k \neq i \leq n} \mathcal{G}_k^{\mathfrak{n}}), \parallel_{1 \leq j \leq n} \mathcal{G}_j^{\mathfrak{n}})$.

Let contracts $\mathcal{C}_i, \mathcal{C}_i'$ be such that $\mathcal{C}_i' \preceq \mathcal{C}_i$. The following theorem states that the composition of a set of contracts dominates the individual contracts and is the *least*, in the refinement order, of all contracts dominating them.

Theorem 11. *Let $\mathcal{C}$ be the composition of $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$, then:*

i) $\mathcal{C}$ dominates the contract set $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$.

ii) $\forall \mathcal{C}' : \mathcal{C}'$ dominates $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\} \Leftrightarrow \mathcal{C} \preceq \mathcal{C}'$.

iii) If $\mathcal{C}'$ dominates $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ then:

a) it dominates also $\{\mathcal{C}'_1, \ldots, \mathcal{C}'_n\}$,

b) $(\underset{1\leq i\leq n}{\|} \mathcal{C}'_i) \preceq (\underset{1\leq i\leq n}{\|} \mathcal{C}_i)$.

Proof. Let $\mathcal{A}_h^{/} \stackrel{\text{def}}{=} \mathcal{A}_h / \underset{1\leq k\neq h\leq n}{\|} \mathcal{G}_k^{\mathfrak{n}}$, then $\mathcal{A} = \underset{1\leq h\leq n}{\curlywedge} \mathcal{A}_h^{/}$.

i) $\mathcal{C}$ dominates $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ because:

a) $\underset{1\leq i\leq n}{\|} \mathcal{I}_i \in [\![\mathcal{C}]\!]_{\mathsf{p}}$, by Theorem 10 and $\mathcal{G} \in [\![\mathcal{C}]\!]_{\mathsf{p}}$.

b) By Theorem 10 and by the quotient property:

$$\begin{aligned}\mathcal{A} \parallel \underset{1\leq j\neq i\leq n}{\Big\|} \mathcal{G}_j^{\mathfrak{n}} &\preceq \mathcal{A}_i^{/} \parallel \underset{1\leq j\neq i\leq n}{\Big\|} \mathcal{G}_j^{\mathfrak{n}} \\ &\preceq (\mathcal{A}_i / \underset{1\leq k\neq i\leq n}{\Big\|} \mathcal{G}_k^{\mathfrak{n}}) \parallel \underset{1\leq j\neq i\leq n}{\Big\|} \mathcal{G}_j^{\mathfrak{n}} \\ &\preceq \mathcal{A}_i.\end{aligned}$$

ii) $\Rightarrow$: $\mathcal{C} \preceq \mathcal{C}'$ because of the following:

– By the dominance of $\mathcal{C}'$ over $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ and by Theorem 10:

$$\begin{aligned}\mathcal{A}' \parallel \underset{1\leq j\neq i\leq n}{\Big\|} \mathcal{G}_j^{\mathfrak{n}} \preceq \mathcal{A}_i &\Rightarrow \mathcal{A}' \preceq \mathcal{A}_i / \underset{1\leq j\neq i\leq n}{\Big\|} \mathcal{G}_j^{\mathfrak{n}} \\ &\Rightarrow \mathcal{A}' \preceq \underset{1\leq i\leq n}{\curlywedge} (\mathcal{A}_i / \underset{1\leq j\neq i\leq n}{\Big\|} \mathcal{G}_j^{\mathfrak{n}}) \\ &\Rightarrow \mathcal{A}' \preceq \mathcal{A} \\ &\Rightarrow [\![\mathcal{C}']\!]_{\mathsf{e}} \subseteq [\![\mathcal{C}]\!]_{\mathsf{e}}.\end{aligned}$$

– By this result and $\mathcal{G} \in [\![\mathcal{C}']\!]_{\mathsf{p}}$, we have:

$$\begin{aligned}\mathcal{I} \in [\![\mathcal{C}]\!]_{\mathsf{p}} &\Rightarrow \forall \mathcal{E} \in [\![\mathcal{C}]\!]_{\mathsf{e}} : \mathcal{I} \parallel \mathcal{E} \preceq \mathcal{G} \parallel \mathcal{E} \\ &\Rightarrow \forall \mathcal{E}' \in [\![\mathcal{C}']\!]_{\mathsf{e}} : \mathcal{I} \parallel \mathcal{E}' \preceq \mathcal{G} \parallel \mathcal{E}' \preceq \mathcal{G}' \parallel \mathcal{E}' \\ &\Rightarrow \mathcal{I} \in [\![\mathcal{C}']\!]_{\mathsf{p}} \\ &\Rightarrow [\![\mathcal{C}]\!]_{\mathsf{p}} \subseteq [\![\mathcal{C}']\!]_{\mathsf{p}}.\end{aligned}$$

$\Leftarrow$: The refinement relation $\mathcal{C} \preceq \mathcal{C}'$ means $[\![\mathcal{C}]\!]_{\mathsf{p}} \subseteq [\![\mathcal{C}']\!]_{\mathsf{p}}$ and $[\![\mathcal{C}']\!]_{\mathsf{e}} \subseteq [\![\mathcal{C}]\!]_{\mathsf{e}}$. By Theorem 10, $\mathcal{C}'$ then dominates $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ because of the following:

- $\underset{1\leq i\leq n}{\|} \mathcal{G}_i^{\mathfrak{n}} \in [\![\mathcal{C}']\!]_{\mathsf{p}}$ as $\mathcal{G} \in [\![\mathcal{C}]\!]_{\mathsf{p}}$ and $[\![\mathcal{C}]\!]_{\mathsf{p}} \subseteq [\![\mathcal{C}']\!]_{\mathsf{p}}$.
- In addition,

$$\begin{aligned}[\![\mathcal{C}']\!]_{\mathsf{e}} \subseteq [\![\mathcal{C}]\!]_{\mathsf{e}} &\Rightarrow \mathcal{A}' \preceq \mathcal{A} \\ &\Rightarrow \mathcal{A}' \preceq \mathcal{A}_i^{/} \\ &\Rightarrow (\mathcal{A}' \parallel \underset{1\leq j\neq i\leq n}{\Big\|} \mathcal{G}_j^{\mathfrak{n}}) \preceq \mathcal{A}_i.\end{aligned}$$

iii) a) By Theorem 10, $\mathcal{C}'$ dominates $\{\mathcal{C}'_1, \ldots, \mathcal{C}'_n\}$ because of the following:

- First, $\mathcal{C}'_i \preceq \mathcal{C}_i \Rightarrow [\![\mathcal{C}'_i]\!]_{\mathsf{p}} \subseteq [\![\mathcal{C}_i]\!]_{\mathsf{p}} \Rightarrow \mathcal{I}'_i \in [\![\mathcal{C}_i]\!]_{\mathsf{p}} \Rightarrow \underset{1\leq i\leq n}{\|} \mathcal{I}'_i \in [\![\mathcal{C}']\!]_{\mathsf{p}}$ (the last implication is because of the dominance of $\mathcal{C}'$ over $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$).
- Second,

$$\begin{aligned}\mathcal{C}'_i \preceq \mathcal{C}_i &\Rightarrow \mathcal{G}'^{\mathfrak{n}}_i \preceq \mathcal{G}^{\mathfrak{n}}_i \\ &\Rightarrow \mathcal{A}' \parallel \underset{1\leq j\neq i\leq n}{\Big\|} \mathcal{G}'^{\mathfrak{n}}_j \preceq \mathcal{A}' \parallel \underset{1\leq j\neq i\leq n}{\Big\|} \mathcal{G}^{\mathfrak{n}}_j \preceq \mathcal{A}_i \preceq \mathcal{A}'_i.\end{aligned}$$

b) A direct consequence of items (i), (ii), (iiia) of Theorem 11.

□

7.1.2 Contract Decomposition

As a direct consequence of Theorem 11, a contract $\mathcal{C}$ refined by the composition of a set of contracts $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ will dominate that contract set and provide desirable compositional consequences formalized in items (i) and (ii) of Defition 17. This contract set is then considered to be a *decomposition* of $\mathcal{C}$, allowing the components associated with the contract set or

their refinements to be plugged into a system satisfying contract $\mathcal{C}$ without breaking the contract satisfaction.

Verifying if $\mathcal{C}$ can be decomposed into $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ is therefore equivalent to checking the dominance of $\mathcal{C}$ over $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ which, by Theorem 10, corresponds to the two decomposing conditions (**DC**s):

$$\textbf{DC-1)} \quad \Big|\Big| \limits_{1 \leq i \leq n} \mathcal{G}_i^{\mathfrak{n}} \in [\![\mathcal{C}]\!]_{\mathsf{p}}, \text{ or equivalently } \Big|\Big| \limits_{1 \leq i \leq n} \mathcal{G}_i^{\mathfrak{n}} \preceq \mathcal{G}^{\mathfrak{n}}$$

$$\textbf{DC-2)} \; \forall 1 \leq i \leq n : \mathcal{A} \parallel \Big|\Big| \limits_{1 \leq j \neq i \leq n} \mathcal{G}_j^{\mathfrak{n}} \preceq \mathcal{A}_i.$$

Moreover, our extension on the dominance notion is more generic than that of Cimatti et al. [15] and can support the construction of property-based proof systems such as that proposed by the same authors. In fact, we built our system in a generic way using a set of meta-theoretical operators including composition, refinement, quotient and conjunction. By adapting these operators suitably, our extension can be applied to different contract frameworks. For example, trace-based contract system development [15] can be derived by instantiating the composition and refinement between specifications as the intersection and set inclusion as follows:

i) $\bigcap \limits_{1 \leq i \leq n} \mathcal{G}_i^{\mathfrak{n}} \in [\![\mathcal{C}]\!]_{\mathsf{p}}$, or equivalently $\bigcap \limits_{1 \leq i \leq n} \mathcal{G}_i^{\mathfrak{n}} \subseteq \mathcal{G}^{\mathfrak{n}}$

ii) $\forall 1 \leq i \leq n : \mathcal{A} \cap \bigcap \limits_{1 \leq j \neq i \leq n} \mathcal{G}_j^{\mathfrak{n}} \subseteq \mathcal{A}_i.$

Likewise, modal contract system development can be based on the modal alternating refinement $\leq_m$ and the modal composition $\|_m$ on shared actions [4].

7.1.3 Contract Synthesis

When a set of contracts does not satisfy the decomposing conditions established in Section 7.1.2, we must adjust the specification of some of them.

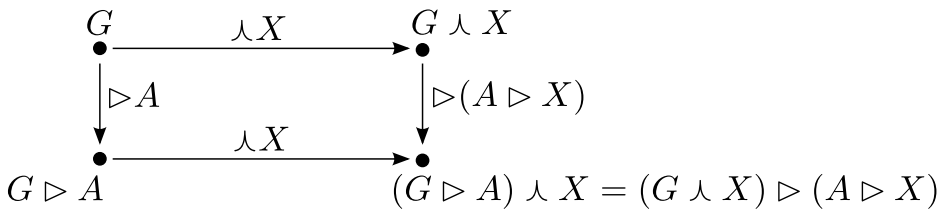


Figure 7.1: Commutative diagram for $\triangleright$ and $\curlywedge$

We propose a synthesis strategy based on the following condition which says the conjunction operator can be distributed over the normalizing operator $\triangleright$ as follows:

$$(\mathcal{G} \triangleright \mathcal{A}) \curlywedge X = (\mathcal{G} \curlywedge X) \triangleright (\mathcal{A} \triangleright X) \tag{7.1}$$

Although this condition poses certain limitations on contract systems, it is a desirable property because it shows that the semantics of a model is invariant when commuting (appropriately) normalization $\triangleright$ and conjunction $\curlywedge$ (Figure 7.1). Better flexibility in the design process can also be gained when these operators are commutative. Since conjunction and normalization amount to strengthening and weakening operations respectively, strengthening X causes a semantic reduction in the two sides of equation (7.1). Thus, when this property does not hold, we can keep strengthening X until we reach a fixed point in semantic equivalence as we shall see later in Section 7.1.5.

Contract synthesis consists of finding suitable refinements for the individual contracts. Our synthesis strategy is based on strengthening the normalized guarantees, which can be reduced to strengthening the unnormalized guarantees and weakening the corresponding assumptions. Because such operations either strengthen the left sides or weaken the right sides of the decomposing conditions, their refinement relation are either maintained or changed from false to true.

To satisfy **DC-1**, we select a guarantee $\mathcal{G}_k^{\mathfrak{n}}$ to be strengthened. By

taking advantage of the quotient, we can find the least specification

$$X = \mathcal{G}^{\mathfrak{n}} / (\bigparallel_{1 \leq i \neq k \leq n} \mathcal{G}_i^{\mathfrak{n}})$$

which ensures the satisfaction of **DC-1**. The newly strengthened normalized guarantee $\bar{\mathcal{G}}_k^{\mathfrak{n}}$ is then:

$$\bar{\mathcal{G}}_k^{\mathfrak{n}} = \mathcal{G}_k^{\mathfrak{n}} \curlywedge X = (\mathcal{G}_k \rhd \mathcal{A}_k) \curlywedge X = (\mathcal{G}_k \curlywedge X) \rhd (\mathcal{A}_k \rhd X) \tag{7.2}$$

Since conjunction and normalization amount to strengthening and weakening operations respectively, the above equation shows that strengthening a normalized guarantee amounts to strengthening its un-normalized version and weakening its coupled assumption. Overall, it amounts to refining the contract $\mathcal{C}_k$. It is also important to notice that strengthening $\mathcal{G}_k^{\mathfrak{n}}$ as above either maintains the refining property established in **DC-2** or may change it from false to true, but not vice-versa because:

$$\begin{aligned} & \mathcal{A}_k \preceq \mathcal{A}_k \rhd X, \text{ for } i = k, \\ \mathcal{A} \parallel \bar{\mathcal{G}}_k^{\mathfrak{n}} \parallel \bigparallel_{1 \leq j \neq k, i \leq n} \mathcal{G}_j^{\mathfrak{n}} & \preceq \mathcal{A} \parallel \bigparallel_{1 \leq j \neq i \leq n} \mathcal{G}_j^{\mathfrak{n}}, \text{ for } i \neq k. \end{aligned}$$

In order to satisfy the i-th clause of **DC-2**, we select a guarantee $\mathcal{G}_{k_i}^{\mathfrak{n}}$ to be strengthened where $k_i \neq i$. Similarly, we can also find the least specification $Y_i = \mathcal{A}_i / (\mathcal{A} \parallel \parallel_{1 \leq j \neq i, k_i \leq n} \mathcal{G}_j^{\mathfrak{n}})$ which ensures the satisfaction of the i-th clause. As done for condition 1, $\mathcal{G}_{k_i}^{\mathfrak{n}}$ is strengthened to $\bar{\mathcal{G}}_{k_i}^{\mathfrak{n}} \stackrel{\text{def}}{=} \mathcal{G}_{k_i}^{\mathfrak{n}} \curlywedge Y_i$:

$$\bar{\mathcal{G}}_{k_i}^{\mathfrak{n}} = \mathcal{G}_{k_i}^{\mathfrak{n}} \curlywedge Y_i = (\mathcal{G}_{k_i} \rhd \mathcal{A}_{k_i}) \curlywedge Y_i = (\mathcal{G}_{k_i} \curlywedge Y_i) \rhd (\mathcal{A}_{k_i} \rhd Y_i) \tag{7.3}$$

Synthesis Strategy:

Based on equation (7.1) and the above analysis, we propose a strategy for synthesizing $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ in order to make it a decomposition of $\mathcal{C}$ as follows:

1. If **DC-1** is not satisfied, select a contract $\mathcal{C}_k$ to be refined and apply (7.2).

2. While **DC-2** is not satisfied:

   (a) Let i be the index of an unsatisfied clause, select contract $\mathcal{C}_{k_i}$ to be refined and apply (7.3).

   (b) Repeat step (2a) until **DC-2** is satisfied.

Our conditions and synthesis strategy for homogeneous systems can be applied to generic homogeneous contract frameworks equipped with specification operators (e.g., composition, refinement) including popular frameworks like trace-based or modal contract frameworks. We next demonstrate our strategy in synthesizing trace-based and modal contract sets.

7.1.4 Trace-based Contract Synthesis

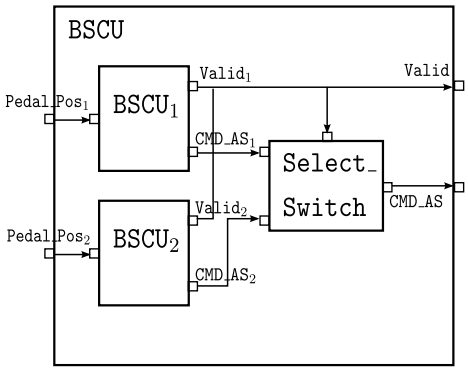
In trace-based contract system where conjunction is simply intersection and normalization is identical to quotient, i.e. $\mathcal{G} \rhd \mathcal{A} = \mathcal{G}/\mathcal{A} = \mathcal{G} \cup \neg\mathcal{A}$, it is easy to prove that equation (7.1) is satisfied:

$$(\mathcal{G} \cup \neg\mathcal{A}) \cap X = (\mathcal{G} \cap X) \cup \neg(\mathcal{A} \cup \neg X).$$

Therefore, we can apply the synthesis strategy proposed above directly. It is also interesting to notice that for trace-based models, to satisfy the i-th clause of **DC-2**, an alternative is to weaken $\mathcal{A}_i$ to $\bar{\mathcal{A}}_i \stackrel{\text{def}}{=} \mathcal{A}_i \cup Z_i$ where $\mathcal{A} \cap \bigcap_{1 \le j \ne i \le n} \mathcal{G}_j^{\mathfrak{n}} \subseteq Z_i$. This operation has a nice consequence in strengthening the corresponding normalized guarantee which is

$$\bar{\mathcal{G}}_i^{\mathfrak{n}} = \mathcal{G}_i \cup \neg(\mathcal{A}_i \cup Z_i) = \mathcal{G}_i \cup (\neg\mathcal{A}_i \cap \neg Z_i)$$

since $(\neg\mathcal{A}_i \cap \neg Z_i) \subseteq \neg\mathcal{A}_i \Rightarrow \bar{\mathcal{G}}_i^{\mathfrak{n}} \subseteq \mathcal{G}_i^{\mathfrak{n}}$.



(a) High-level view of BSCU

$$\mathcal{A} \stackrel{\text{def}}{=} \texttt{always}\left(\neg\texttt{fault\_Monitor}_1 \wedge \neg\texttt{fault\_Command}_1 \wedge \neg\texttt{fault\_Monitor}_2\right) \vee$$
$$\texttt{always}\left(\neg\texttt{fault\_Monitor}_1 \wedge \neg\texttt{fault\_Command}_1 \wedge \neg\texttt{fault\_Command}_2\right) \vee$$
$$\texttt{always}\left(\neg\texttt{fault\_Monitor}_1 \wedge \neg\texttt{fault\_Command}_2 \wedge \neg\texttt{fault\_Monitor}_2\right) \vee$$
$$\texttt{always}\left(\neg\texttt{fault\_Command}_1 \wedge \neg\texttt{fault\_Command}_1 \wedge \neg\texttt{fault\_Monitor}_2\right)$$

$$\mathcal{G} \stackrel{\text{def}}{=} \texttt{always}\left(\texttt{Valid}_1 \vee \texttt{Valid}_2\right)$$

$$\mathcal{A}_i \stackrel{\text{def}}{=} \texttt{always}\left(\neg\texttt{fault\_Monitor}_i\right) \wedge \texttt{always}\left(\neg\texttt{fault\_Command}_i\right)$$

$$\mathcal{G}_i \stackrel{\text{def}}{=} \texttt{always}\left(\neg\texttt{fault\_Monitor}_i\right) \wedge \texttt{always}\left(\neg\texttt{fault\_Command}_i\right) \implies \texttt{always}\left(\texttt{Valid}_i\right)$$

(b) Contract specification of BSCU and $\texttt{BSCU}_i$

Figure 7.2: Structure and contract models of BSCU

Example 11. We consider a variant of the contract model of the Brake System Control Unit (BSCU) described in [19] and shown in Figure 7.2(a). The BSCU takes as inputs the positions of the two brake pedals $\texttt{Pedal\_Pos}_1$ and $\texttt{Pedal\_Pos}_2$ and outputs two control signals Valid and CMD_AS to control the braking process of a wheel-brake system.

The BSCU component is further decomposed into a Select_Switch and two smaller control units: a primary $\texttt{BSCU}_1$ and a backup $\texttt{BSCU}_2$. When $\texttt{BSCU}_1$ fails, the Select_Switch puts the backup signal from $\texttt{BSCU}_2$ through. The signal failure in a control unit $\texttt{BSCU}_i$ is indicated by its signal $\texttt{Valid}_i$ going down and is caused by a basic fault which is either a monitor fault $\texttt{fault\_Monitor}_i$ or a command fault $\texttt{fault\_Command}_i$ with $i \in \{1, 2\}$. A safety requirement on the BSCU is to ensure that $\texttt{Valid}_1 \vee \texttt{Valid}_2$ is always true when at most one of the basic faults $\texttt{fault\_Monitor}_i$ or $\texttt{fault\_Command}_i$ can occur. This is specified as contract $\mathcal{C} = (\mathcal{A}, \mathcal{G})$ in Figure 7.2(b). The safety contract specification $\mathcal{C}_i = (\mathcal{A}_i, \mathcal{G}_i)$ on $\texttt{BSCU}_i$ in the same figure guarantees that signal $\texttt{Valid}_i$ remains true when neither of its basic faults occurs. The contracts are specified in symbolic logic [15] where sets of traces are represented by logical formulas. Thus checking the two **DC**s amounts to checking the following formulas in symbolic logic:

i) $\bigwedge\limits_{1 \leq i \leq n} \mathcal{G}_i^{\mathfrak{n}} \Rightarrow \mathcal{G}^{\mathfrak{n}}$

ii) $\forall 1 \leq i \leq n : \mathcal{A} \wedge \bigwedge\limits_{1 \leq j \neq i \leq n} \mathcal{G}_j^{\mathfrak{n}} \Rightarrow \mathcal{A}_i.$

where $=^{\mathfrak{n}} \mathcal{G} \vee \neg \mathcal{A}$ and $\mathcal{G}_i^{\mathfrak{n}} = \mathcal{G}_i \vee \neg \mathcal{A}_i$. To verify if $\mathcal{C}$ can be decomposed into $\mathcal{C}_1$ and $\mathcal{C}_2$, we verify the satisfaction of the two **DC**s. It is obvious that the contracts $\mathcal{C}_i$ are in normal form, thus $\mathcal{G}_i^{\mathfrak{n}} \equiv \mathcal{G}_i$. Moreover, $\mathcal{G}_1 \wedge \mathcal{G}_2 \Rightarrow \mathcal{G}^{\mathfrak{n}}$ is correct (**DC-1** is satisfied) while $\mathcal{A} \wedge \mathcal{G}_1 \Rightarrow \mathcal{A}_2$ and $\mathcal{A} \wedge \mathcal{G}_2 \Rightarrow \mathcal{A}_1$ are not (**DC-2** is not satisfied). Applying step (2a) of our synthesis strategy twice, we refine $\mathcal{C}_1$ w.r.t. $Y_2 \stackrel{\text{def}}{=} (\mathcal{A} \Rightarrow \mathcal{A}_2)$ and $\mathcal{C}_2$ w.r.t. $Y_1 \stackrel{\text{def}}{=} ((\mathcal{A} \wedge \mathcal{A}_2) \Rightarrow \mathcal{A}_1)$.

Table 7.1: Rules for combing modal specification using modal operators $\rhd_m, \|_m, /_m, \curlywedge_m$

$\rhd_m$	$s_2 \overset{\alpha}{\dashrightarrow} s_2'$	$s_2 \overset{\alpha}{\dashrightarrow\!\!\!\!\!\!\rightarrow}$
$s_1 \overset{\alpha}{\rightarrow} s_1'$	$(s_1, s_2) \overset{\alpha}{\rightarrow} (s_1', s_2')$	$(s_1, s_2) \overset{\alpha}{\dashrightarrow} \mathfrak{u}$
$s_1 \overset{\alpha}{\dashrightarrow} s_1'$	$(s_1, s_2) \overset{\alpha}{\dashrightarrow} (s_1', s_2')$	$(s_1, s_2) \overset{\alpha}{\dashrightarrow} \mathfrak{u}$
$s_1 \overset{\alpha}{\dashrightarrow\!\!\!\!\!\!\rightarrow}$		$(s_1, s_2) \overset{\alpha}{\dashrightarrow} \mathfrak{u}$

$\|_m$	$s_2 \overset{\alpha}{\rightarrow} s_2'$	$s_2 \overset{\alpha}{\dashrightarrow} s_2'$
$s_1 \overset{\alpha}{\rightarrow} s_1'$	$(s_1, s_2) \overset{\alpha}{\rightarrow} (s_1', s_2')$	$(s_1, s_2) \overset{\alpha}{\dashrightarrow} (s_1', s_2')$
$s_1 \overset{\alpha}{\dashrightarrow} s_1'$	$(s_1, s_2) \overset{\alpha}{\rightarrow} (s_1', s_2')$	$(s_1, s_2) \overset{\alpha}{\dashrightarrow} (s_1', s_2')$

$/_m$	$s_2 \overset{\alpha}{\rightarrow} s_2'$	$s_2 \overset{\alpha}{\dashrightarrow} s_2'$	$s_2 \overset{\alpha}{\dashrightarrow\!\!\!\!\!\!\rightarrow}$
$s_1 \overset{\alpha}{\rightarrow} s_1'$	$(s_1, s_2) \overset{\alpha}{\rightarrow} (s_1', s_2')$	(s_1, s_2) is inconsistent	(s_1, s_2) is inconsistent
$s_1 \overset{\alpha}{\dashrightarrow} s_1'$	$(s_1, s_2) \overset{\alpha}{\dashrightarrow} (s_1', s_2')$	$(s_1, s_2) \overset{\alpha}{\dashrightarrow} (s_1', s_2')$	$(s_1, s_2) \overset{\alpha}{\dashrightarrow} \mathfrak{u}$
$s_1 \overset{\alpha}{\dashrightarrow\!\!\!\!\!\!\rightarrow}$			$(s_1, s_2) \overset{\alpha}{\dashrightarrow} \mathfrak{u}$

$\curlywedge_m$	$s_2 \overset{\alpha}{\rightarrow} s_2'$	$s_2 \overset{\alpha}{\dashrightarrow} s_2'$	$s_2 \overset{\alpha}{\dashrightarrow\!\!\!\!\!\!\rightarrow}$
$s_1 \overset{\alpha}{\rightarrow} s_1'$	$(s_1, s_2) \overset{\alpha}{\rightarrow} (s_1', s_2')$	$(s_1, s_2) \overset{\alpha}{\rightarrow} (s_1', s_2')$	(s_1, s_2) is inconsistent
$s_1 \overset{\alpha}{\dashrightarrow} s_1'$	$(s_1, s_2) \overset{\alpha}{\rightarrow} (s_1', s_2')$	$(s_1, s_2) \overset{\alpha}{\dashrightarrow} (s_1', s_2')$	
$s_1 \overset{\alpha}{\dashrightarrow\!\!\!\!\!\!\rightarrow}$	(s_1, s_2) is inconsistent		

Alternatively, we can weaken $\mathcal{A}_i$ w.r.t. any Z_i such that $(\mathcal{A} \wedge \mathcal{G}_{3-i}) \Rightarrow Z_i$ is correct. The simplest option could be $Z_i = \texttt{TRUE}$ and this derives the original safety contracts [15]. Our approach therefore provides a wider set of options which allows designers to explore the refinement space.

7.1.5 Modal Contract Synthesis

Modal contracts are defined over modal transition systems (MST) where transitions are annotated with action labels and with *may* or *must* modalities modeling behaviors which can be (optionally) or must be (compulsorily) implemented respectively. Formally, an MST is a tuple $M = (S, s_0, \Sigma, \dashrightarrow, \rightarrow)$ where S is the set of states, $s_0 \in S$ is the initial state, Σ is the set of actions and $\dashrightarrow, \rightarrow \subseteq S \times \Sigma \times S$ are the *may*,*must* transition relation respectively such that $\rightarrow \subseteq \dashrightarrow$ [4].

For the sake of comprehension, we use our notations with $m-$subscripts when referring to modal operators. The modal operators for combining modal transitions are described in Table 7.1 where $\mathfrak{u}$ denotes a new universal state in which there is a looping *may* transition for every action. In addition, a pruning procedure is applied to the newly combined system in order to prune away inconsistent states [4]. The modal refinement is defined as follows [4]. An MST $M_1 = (S_1, s_{01}, \Sigma_1, \dashrightarrow_1, \rightarrow_1)$ refines another MST $M_2 = (S_2, s_{02}, \Sigma_2, \dashrightarrow_2, \rightarrow_2)$, written $M_1 \leq_m M_2$, if there exists a 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 $\alpha \in \Sigma$:

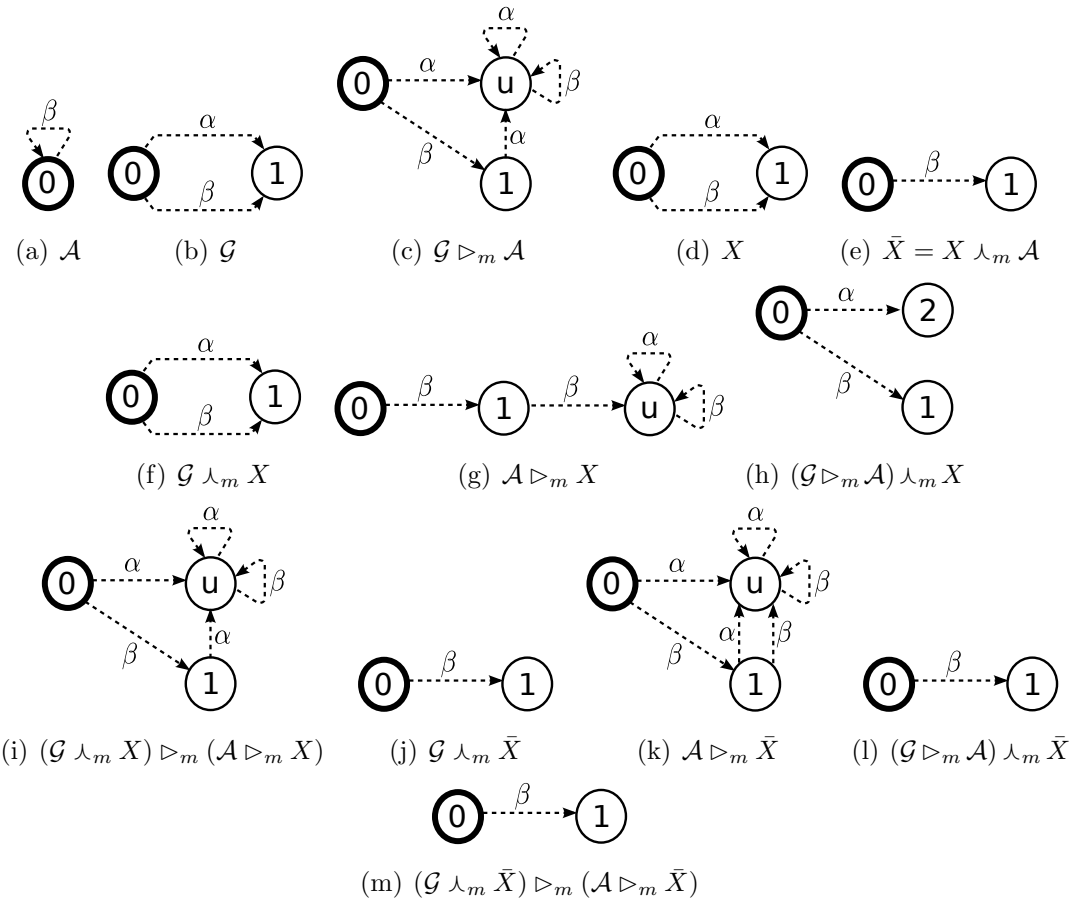
$$((s_1, \alpha, s_1') \in \dashrightarrow_1 \Rightarrow \exists (s_2, \alpha, s_2') \in \dashrightarrow_2 : (s_1', s_2') \in R) \wedge$$
$$((s_2, \alpha, s_2') \in \rightarrow_2 \Rightarrow \exists (s_1, \alpha, s_1') \in \rightarrow_1 : (s_1', s_2') \in R)$$

Consider a simple modal contract $\mathcal{C} = (\mathcal{A}, \mathcal{G})$ specified in Figure 7.3(a) and Figure 7.3(b) and a specification X in Figure 7.3(d) where the initial states are marked by bold circles. Equation (7.1) is shown to be violated as demonstrated in Figure 7.3(h) and Figure 7.3(i). The reason is that normalization may introduce a universal state with a looping *may* transition for every action. Whereas, during conjunction, such universal state could be pruned away. To avoid such inconsistency, $\mathcal{A} \rhd_m X$ should contain all *may* transitions appearing in X. This can be obtained by tightening X to $\bar{X} \stackrel{\text{def}}{=} X \curlywedge_m \mathcal{A}$ as shown in Figure 7.3(l) and Figure 7.3(m).

Theorem 12. $(\mathcal{G} \rhd_m \mathcal{A}) \curlywedge_m \bar{X} = (\mathcal{G} \curlywedge_m \bar{X}) \rhd_m (\mathcal{A} \rhd_m \bar{X})$.

Proof. To prove the satisfaction of Equation (7.1), we show that every path in $(\mathcal{G} \rhd_m \mathcal{A}) \curlywedge_m \bar{X}$ can be simulated by $(\mathcal{G} \curlywedge_m \bar{X}) \rhd_m (\mathcal{A} \rhd_m \bar{X})$ and vice versa.

- Let $p_l : ((g_0, a_0), \bar{x_0}) \stackrel{\alpha_0}{\dashrightarrow} ((g_1, a_1), \bar{x_1}) \ldots \stackrel{\alpha_n}{\dashrightarrow} ((g_n, a_n), \bar{x_n})$ be a path in $(\mathcal{G} \rhd_m \mathcal{A}) \curlywedge_m \bar{X}$. Then by definition of $\curlywedge_m$, there exist p_{ga} in $(\mathcal{G} \rhd_m \mathcal{A})$,



(a) $\mathcal{A}$ (b) $\mathcal{G}$ (c) $\mathcal{G} \rhd_m \mathcal{A}$ (d) X (e) $\bar{X} = X \swarrow_m \mathcal{A}$

(f) $\mathcal{G} \swarrow_m X$ (g) $\mathcal{A} \rhd_m X$ (h) $(\mathcal{G} \rhd_m \mathcal{A}) \swarrow_m X$

(i) $(\mathcal{G} \swarrow_m X) \rhd_m (\mathcal{A} \rhd_m X)$ (j) $\mathcal{G} \swarrow_m \bar{X}$ (k) $\mathcal{A} \rhd_m \bar{X}$ (l) $(\mathcal{G} \rhd_m \mathcal{A}) \swarrow_m \bar{X}$

(m) $(\mathcal{G} \swarrow_m \bar{X}) \rhd_m (\mathcal{A} \rhd_m \bar{X})$

Figure 7.3: A modal contract

$p_{\bar{x}}$ in $\bar{X}$ and p_a in $\mathcal{A}$:

$$p_{ga} : (g_0, a_0) \overset{\alpha_0}{\dashrightarrow} (g_1, a_1) \ldots \overset{\alpha_n}{\dashrightarrow} (g_n, a_n)$$
$$p_{\bar{x}} : \bar{x}_0 \overset{\alpha_0}{\dashrightarrow} \bar{x}_1 \ldots \overset{\alpha_n}{\dashrightarrow} \bar{x}_n$$
$$p_a : a_0 \overset{\alpha_0}{\dashrightarrow} a_1 \ldots \overset{\alpha_n}{\dashrightarrow} a_n.$$

By definition of $\rhd_m$, the existence of p_{ga} and p_a implies that of path p_g in $\mathcal{G}$:

$$p_g : g_0 \overset{\alpha_0}{\dashrightarrow} g_1 \ldots \overset{\alpha_n}{\dashrightarrow} g_n.$$

Next $p_g, p_{\bar{x}}$ and p_a implies the existence of path p_r in $(\mathcal{G} \swarrow_m \bar{X}) \rhd_m (\mathcal{A} \rhd_m \bar{X})$:

$$p_r : ((g_0, \bar{x}_0), (a_0, \bar{x}_0)) \overset{\alpha_0}{\dashrightarrow} \ldots \overset{\alpha_n}{\dashrightarrow} ((g_n, \bar{x}_n), (a_n, \bar{x}_n)).$$

In addition, assume there is a must transition

$$((g_i, a_i), \bar{x}_i) \xrightarrow{\alpha_i} ((g_{i+1}, a_{i+1}), \bar{x}_{i+1})$$

somewhere in p_l. By definition of $\curlywedge_m$, either $(g_i, a_i) \xrightarrow{\alpha_i} (g_{i+1}, a_{i+1})$ or $\bar{x}_i \xrightarrow{\alpha_i} \bar{x}_{i+1}$ holds and implies $(g_i, \bar{x}_i) \xrightarrow{\alpha_i} (g_{i+1}, \bar{x}_{i+1})$. Thus there is also a must transition in p_r:

$$((g_i, \bar{x}_i), (a_i, \bar{x}_i)) \xrightarrow{\alpha_i} ((g_{i+1}, \bar{x}_{i+1}), (a_{i+1}, \bar{x}_{i+1})).$$

- Let $p_r : ((g_0, \bar{x}_0), (a_0, \bar{x}'_0)) \overset{\alpha_0}{\dashrightarrow} \ldots \overset{\alpha_n}{\dashrightarrow} ((g_n, \bar{x}_n), (a_n, \bar{x}'_n))$ be a path in $(\mathcal{G} \curlywedge_m \bar{X}) \rhd_m (\mathcal{A} \rhd_m \bar{X})$. By induction, we prove that for $0 \leq i \leq n$, $((g_i, \bar{x}_i), (a_i, \bar{x}'_i))$ is not the universal state and $\bar{x}_i \equiv \bar{x}'_i$.
  - Base case $i = 0$: trivial.
  - Step case: assume the induction holds up to the i-th state of p_r. By contraposition, assume the $(i+1)$-th state which is state $((g_{i+1}, \bar{x}_{i+1}), (a_{i+1}, \bar{x}'_{i+1}))$ is universal. Then it must be that the fact $(a_i, \bar{x}_i) \overset{\alpha_i}{\not\dashrightarrow}$ holds by definition of $\rhd_m$. This implies

    $$(a_i \overset{\alpha_i}{\not\dashrightarrow}) \quad \wedge \quad (\bar{x}_i \overset{\alpha_i}{\dashrightarrow} \bar{x}_{i+1}).$$

    As $\bar{X} = X \curlywedge_m \mathcal{A}$, the latter then implies $a_i \overset{\alpha_i}{\dashrightarrow} a_{i+1}$ by definition of $\curlywedge_m$, contradicting with the former. Thus the $(i+1)$-th state of p_r is not universal and this implies, by definition of $\rhd_m$, that

    $$((a_i, \bar{x}_i) \overset{\alpha_i}{\dashrightarrow} (a_{i+1}, \bar{x}'_{i+1})) \quad \wedge \quad ((g_i, \bar{x}_i) \overset{\alpha_i}{\dashrightarrow} (g_{i+1}, \bar{x}_{i+1})$$

    which then implies $(a_i, \bar{x}_i) \overset{\alpha_i}{\dashrightarrow} (a_{i+1}, \bar{x}_{i+1}))$. Hence, $\bar{x}_{i+1} \equiv \bar{x}'_{i+1}$ by the deterministic assumption on modal automata.

The induction also infers the existence of p_g in $\mathcal{G}$, $p_{\bar{x}}$ in $\bar{X}$, p_a in $\mathcal{A}$:

$$p_g : g_0 \overset{\alpha_0}{\dashrightarrow} \ldots \overset{\alpha_n}{\dashrightarrow} g_n$$
$$p_{\bar{x}} : \bar{x}_0 \overset{\alpha_0}{\dashrightarrow} \ldots \overset{\alpha_n}{\dashrightarrow} \bar{x}_n$$
$$p_a : a_0 \overset{\alpha_0}{\dashrightarrow} \ldots \overset{\alpha_n}{\dashrightarrow} a_n$$

which together implies that of p_l in $(\mathcal{G} \rhd_m \mathcal{A}) \swarrow_m \bar{X}$:

$$p_l : ((g_0, a_0), \bar{x_0}) \overset{\alpha_0}{\dashrightarrow} \ldots \overset{\alpha_n}{\dashrightarrow} ((g_n, a_n), \bar{x_n}).$$

In addition, if there is a must transition

$$((g_i, \bar{x}_i), (a_i, \bar{x}_i)) \overset{\alpha_i}{\to} ((g_{i+1}, \bar{x}_{i+1}), (a_{i+1}, \bar{x}_{i+1}))$$

somewhere in p_r, then there must be $(g_i, \bar{x}_i) \overset{\alpha_i}{\to} (g_{i+1}, \bar{x}_{i+1})$ by definition of $\rhd_m$. Thus either $g_i \overset{\alpha_i}{\to} g_{i+1}$ or $\bar{x}_i \overset{\alpha_i}{\to} \bar{x}_{i+1}$ holds and implies that there is a must transition in p_l:

$$((g_i, a_i), \bar{x}_i) \overset{\alpha_i}{\to} ((g_{i+1}, a_{i+1}), \bar{x}_{i+1}).$$

□

Example 12. We consider the simple message system `System` with contract specification $(\mathcal{A}_{System}, \mathcal{G}_{System})$ studied by Bauer et al. [4] and shown in Figure 7.4(a) and Figure 7.4(b) where *may* transitions underlying *must* transitions are not drawn for simplicity. In addition, we retain the `User` component and make a minor modification to the assumption of the `Server` component by disallowing the authentication code reception after a message is sent to the user (Figure 7.4(f)). Decomposing the message system into these two components is only possible when the system contract can also be decomposed into their associated contracts. However, the composition of the `Server` and `User` normalized guarantees, i.e., $\mathcal{G}^{\mathfrak{n}}_{Server} \parallel_m \mathcal{G}^{\mathfrak{n}}_{User}$ does not refine $\mathcal{G}^{\mathfrak{n}}_{System}$ since the authentication code reception is allowed by the former and forbidden by the latter. We next apply our synthesis strategy in Sect. 7.1.3 to synthesize the `Server` contract w.r.t. $\bar{X}$ shown in Figure 7.4(i). The newly-synthesized `Server` contract provides the same guarantee under a more general assumption (Figure 7.4(j)). It is then easy to verify that its composition with the `User` contract refines the overall `System` contract.

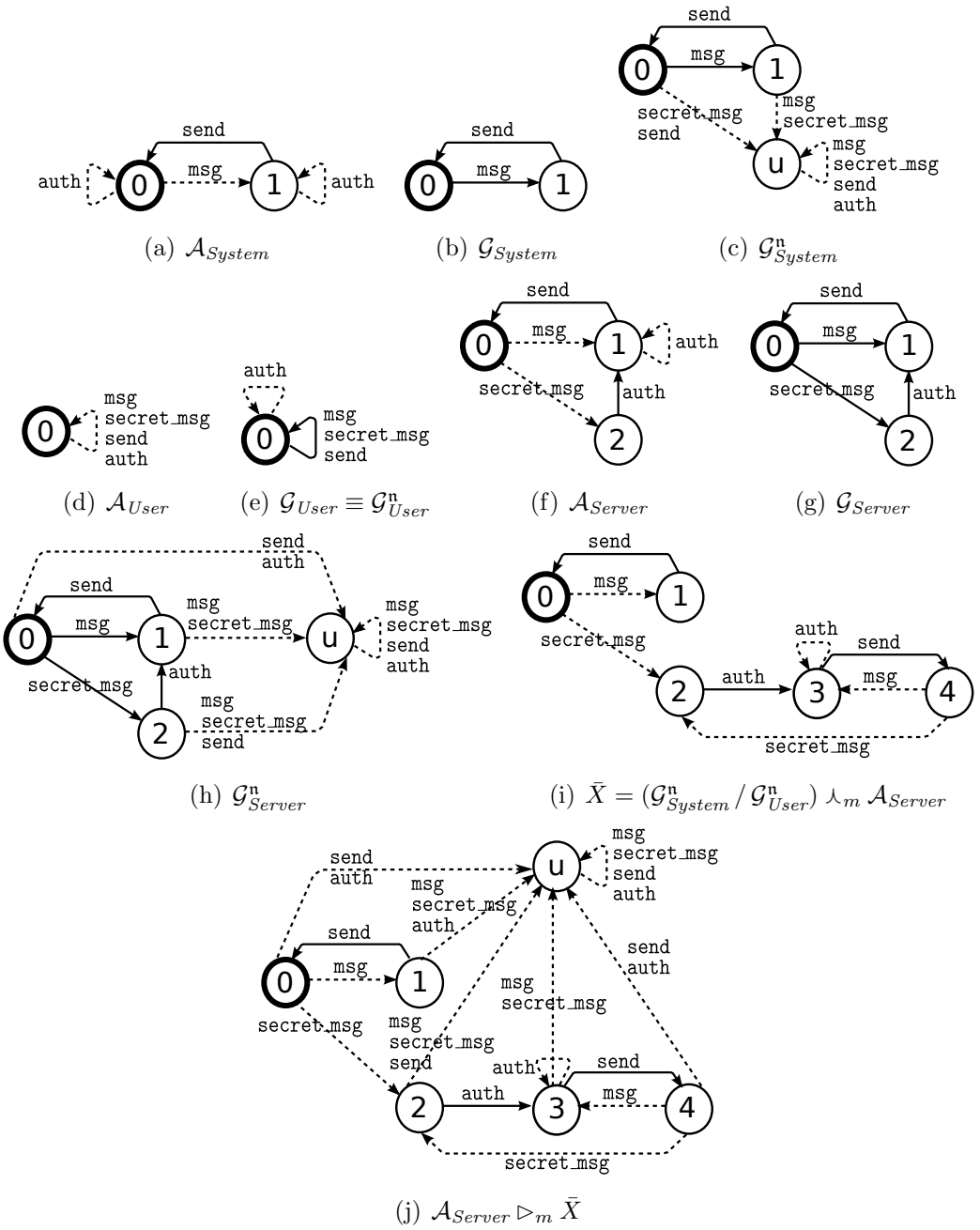


Figure 7.4: Modal contracts for a simple message system

7.2 Heterogeneous 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 Theorem 10 is not satisfied, we must adjust or *synthesize* the individual contracts in order to gain the dominance satisfaction.

To obtain the satisfaction of **hDC-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 13, 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 13. We consider an example where:

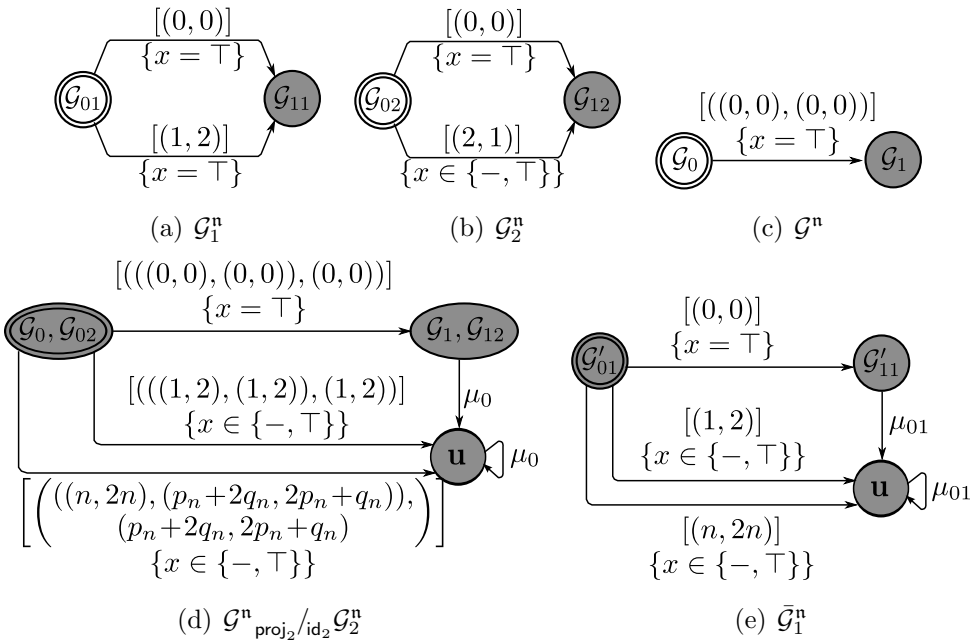
- $T_1 = \{(-\infty, -\infty), (0,0), (k, 2k)\}$ with $k \in \mathbb{N} \wedge k \geq 1$.
- $T_2 = \{(-\infty, -\infty), (0,0), (i+2j, 2i+j)\}$ with $i, j \in \mathbb{N} \wedge i, j \geq 0$.
- $\leq_1 \equiv \leq_2$ and is defined such that

$$(\tau_1, \tau_2) \leq_1 (\tau_1', \tau_2') \Leftrightarrow (\tau_1 \leq \tau_1') \wedge (\tau_2 \leq \tau_2').$$

- $+_1 \equiv +_2$ 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. Assuming that we have



(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 7.5: Synthesis based on heterogeneneous quotient and projection

the algebraic tag morphisms $\rho_1 : \mathcal{T}_1 \mapsto \mathcal{T}$ and $\rho_2 : \mathcal{T}_2 \mapsto \mathcal{T}$ 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 7.5(a), 7.5(b) and 7.5(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 $V_1 \equiv V_2 \equiv V = \{x\}$ 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 agreeing 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 7.5(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 7.5(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. The following procedure demonstrates how this can be done.

$\mathcal{G}_1^{\mathfrak{n}} = (V_1, \mathcal{T}_1, S_{g_1}, s_{0g_1}, F_{g_1}, E_{g_1})$;
Input: $\mathcal{G}_2^{\mathfrak{n}} = (V_2, \mathcal{T}_2, S_{g_2}, s_{0g_2}, F_{g_2}, E_{g_2})$;
$\mathcal{G}^{\mathfrak{n}} = (V, \mathcal{T}_1 \,{}_{\rho_1}\!\times_{\rho_2} \mathcal{T}_2, S_g, s_{0g}, F_g, E_g)$;
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 **hDC-1**

Lemma 4. $\bar{\mathcal{G}}_1^{\mathfrak{n}} \preceq \mathcal{G}_1^{\mathfrak{n}}$.

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

Proof. By contraposition, assuming 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:

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

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 $r : s_{0g} \xrightarrow{\mu_1} s_{1g} \ldots \xrightarrow{\mu_n} s_{ng}$ 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 $r : s_0 \xrightarrow{\mu_1} s_1 \ldots \xrightarrow{\mu_n} s_n$ 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 6. *Algorithm 1 finally terminates in finite time.*

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

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 using the following algorithm.

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
    $Q_{g_1} = L(V_1, \mathcal{T}_1) \setminus \{\mu_1 | \exists (s_{kg_1}, \mu_1, s_{(k+1)g_1}) \in \bar{E}_{g_1}\}$;
    $Q_{a_1} = L(V_1, \mathcal{T}_1) \setminus \{\mu_1 | \exists (s_{ka_1}, \mu_1, s_{(k+1)a_1}) \in \bar{E}_{a_1}\}$;
    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

Lemma 7. $\mathcal{A}_1 \preceq \bar{\mathcal{A}}_1$.

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

Lemma 8. $\bar{\mathcal{G}}_1^{\mathfrak{n}}/\bar{\mathcal{A}}_1 = \bar{\mathcal{G}}_1^{\mathfrak{n}}$.

Proof. It is obvious that $\bar{\mathcal{G}}_1^{\mathfrak{n}}/\bar{\mathcal{A}}_1 \preceq \bar{\mathcal{G}}_1^{\mathfrak{n}}$ by Algorithm 2 and $\bar{\mathcal{G}}_1^{\mathfrak{n}} \preceq \bar{\mathcal{G}}_1^{\mathfrak{n}}/\bar{\mathcal{A}}_1$ by Lemma 3. □

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.

Lemma 9. *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 that i) $\bar{\mathcal{G}}_1^{\mathfrak{n}}/\bar{\mathcal{A}}_1 \preceq \bar{\mathcal{G}}_1/\bar{\mathcal{A}}_1$, ii) and $\bar{\mathcal{G}}_1/\bar{\mathcal{A}}_1 \preceq \bar{\mathcal{G}}_1^{\mathfrak{n}}/\bar{\mathcal{A}}_1$. First, the refinement $\bar{\mathcal{G}}_1^{\mathfrak{n}} \preceq \bar{\mathcal{G}}_1/\bar{\mathcal{A}}_1$ holds by the assumption of $\bar{\mathcal{G}}_1^{\mathfrak{n}} \parallel \bar{\mathcal{A}}_1 \preceq \bar{\mathcal{G}}_1$ and by the Quotient Property (6.5). Second, the refinement $\bar{\mathcal{G}}_1^{\mathfrak{n}}/\bar{\mathcal{A}}_1 \preceq \bar{\mathcal{G}}_1^{\mathfrak{n}}$ also holds by Lemma 8. Hence, item i) follows immediately. By construction, $\bar{\mathcal{G}}_1 \preceq \bar{\mathcal{G}}_1^{\mathfrak{n}}$ holds and by the Quotient Property (6.5), the refinement $(\bar{\mathcal{G}}_1/\bar{\mathcal{A}}_1) \parallel \bar{\mathcal{A}}_1 \preceq \bar{\mathcal{G}}_1$ also holds, implying $(\bar{\mathcal{G}}_1/\bar{\mathcal{A}}_1) \parallel \bar{\mathcal{A}}_1 \preceq \bar{\mathcal{G}}_1^{\mathfrak{n}}$. Hence item ii) follows immediately. □

It is important to notice that strengthening $\mathcal{G}_1^{\mathfrak{n}}$ as above either maintains the refining property established in **hDC-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 **hDC-2a** and **hDC-2b**, we can strengthen the normalized guarantees by following respectively Algorithm 3 and 4 which are

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

$\mathcal{G}_1^{\mathfrak{n}} = (V_1, \mathcal{T}_1, S_{g_1}, s_{0g_1}, F_{g_1}, E_{g_1})$;
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
            if $(s_{(k+1)a} \in F_a) \wedge (s_{(k+1)g_1} \in \bar{F}_{g_1}) \wedge (s_{(k+1)a_2} \notin F_{a_2})$ **then**
              Remove $s_{(k+1)g_1}$ from $\bar{F}_{g_1}$;
            end
            Add $((s_{(k+1)a}, s_{(k+1)g_1}), s_{(k+1)a_2})$ to R;
          end
        end
      end
    end
  end
end

Algorithm 3: Refining $\mathcal{G}_1^{\mathfrak{n}}$ so as to satisfy **hDC-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
            if $(s_{(k+1)a} \in F_a) \wedge (s_{(k+1)g_2} \in \bar{F}_{g_2}) \wedge (s_{(k+1)a_1} \notin F_{a_1})$ **then**
              Remove $s_{(k+1)g_2}$ from $\bar{F}_{g_2}$;
            end
            Add $((s_{(k+1)a}, s_{(k+1)g_2}), s_{(k+1)a_1})$ to R;
          end
        end
      end
    end
  end
end

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

Lemma 10. $(\mathcal{A} \,{}_{\mathsf{proj}_1}\|_{\mathsf{id}_1}\, \bar{\mathcal{G}}_1^{\mathfrak{n}}) \,{}_{\mathsf{proj}_2'}\preceq_{\mathsf{id}_2} \mathcal{A}_2$ *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. Similar to the proof of Lemma 4. □

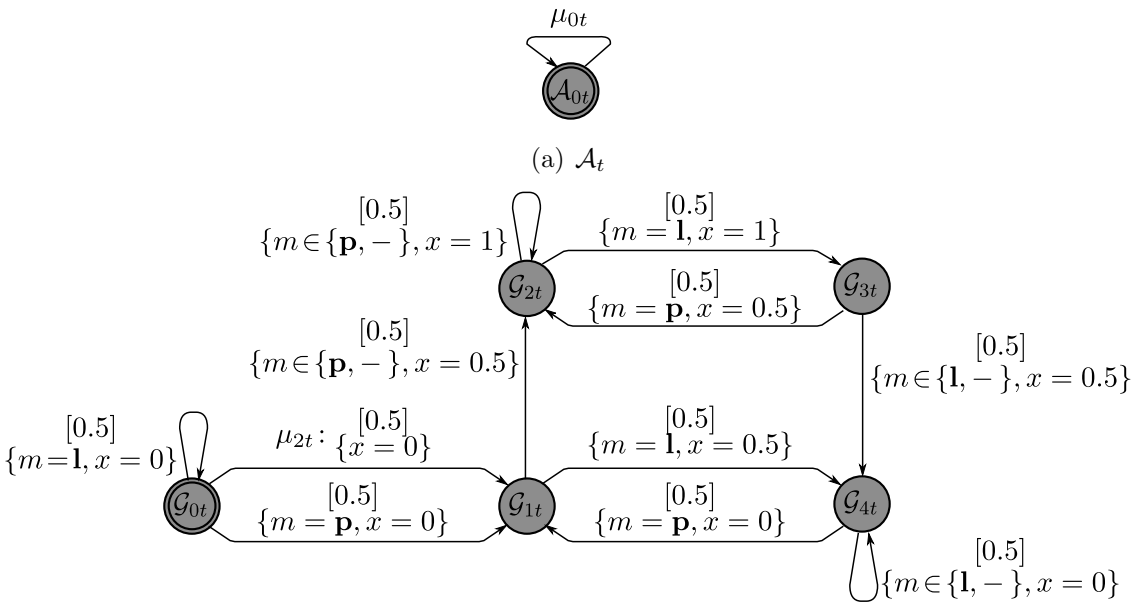


Figure 7.6: The tank contract

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 **hDC-1** is satisfied.

2. Repeat applying Algorithm 3 or Algorithm 4 until **hDC-2** is satisfied.

Example 14. We consider again the simplified water controlling system in Example 4 as shown in Figure 6.1. Figure 7.6 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 controller contract is shown Figure 7.7, where it assumes the tank to be empty initially (Figure 7.7(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 (Figure 7.7(b)) can send a proper command upon knowing of the tank emptiness or fullness. Intuitively, the controller contract ensures timely control

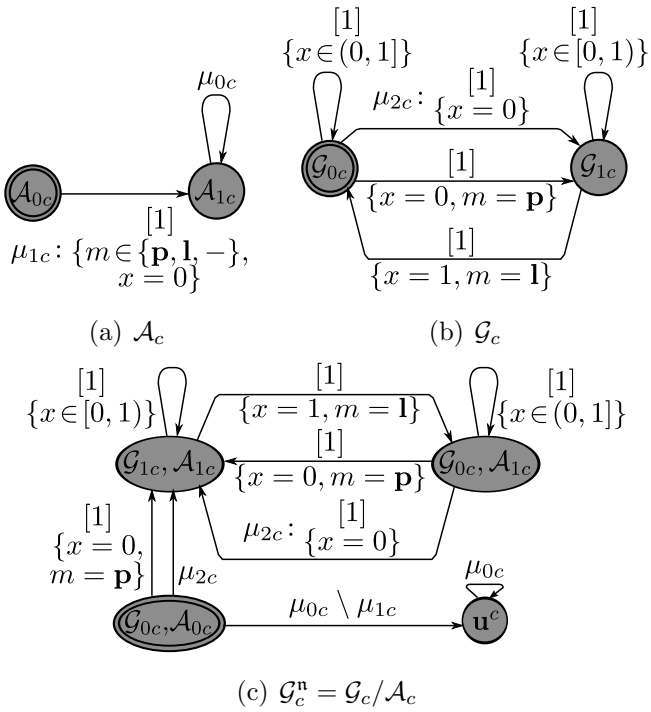
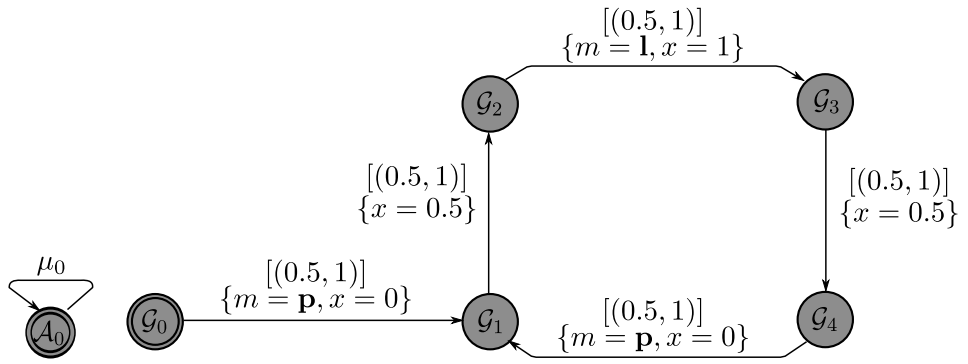


Figure 7.7: The controller contract

over the water evolution while the tank contract accepts untimely control and allow water spillages or shortages.

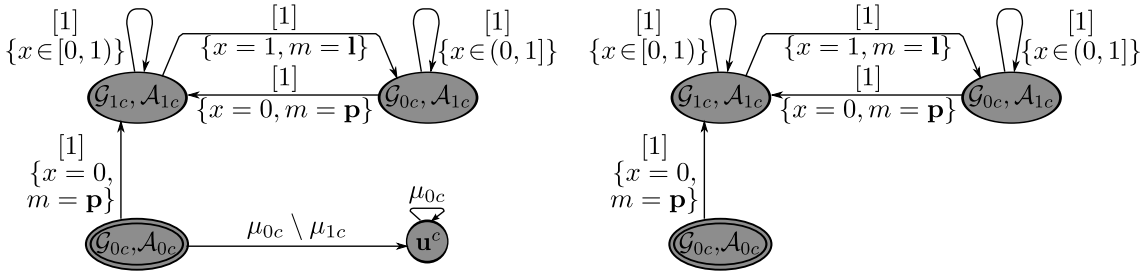
We use the same tag structures, which are $\mathcal{T}_1 = (\mathbb{R}_+ \cup \{\epsilon_1\}, +)$ and $\mathcal{T}_2 = (\mathbb{N} \cup \{\epsilon_2\}, +)$, in Example 4 to describe the tank and controller contracts respectively. We also use the same morphisms $\rho_1 : \mathcal{T}_1 \mapsto \mathcal{T}_1$ and $\rho_2 : \mathcal{T}_2 \mapsto \mathcal{T}_1$ given by $\rho_1(\tau_1) = \tau_1$, $\rho_2(\tau_2) = 0.5 * \tau_2$ 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 $L(V_2, \mathcal{T}_2)$.

We consider the specification $\mathcal{C} = (\mathcal{A}, \mathcal{G})$ where it makes no assump-



(a) $\mathcal{A}$ (b) $\mathcal{G}^{\mathfrak{n}} \equiv \mathcal{G}$

Figure 7.8: The desirable water control behavior



(a) $\bar{\mathcal{G}}_c^{\mathfrak{n}}$ (b) $\bar{\mathcal{G}}_c$

Figure 7.9: Controller synthesis

tions, 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 7.8. 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 assumption made to their variables. Meanwhile, the controller normalized guarantee specifies more behaviors than its un-normalized version as the controller does have some assumption on its input.

Composing $\mathcal{C}_t$ and $\mathcal{C}_c$ under morphisms ρ_1 and ρ_2, however, does not satisfy contract $\mathcal{C}$. This is because $\mathcal{G}_t^{\mathfrak{n}} \,{}_{\rho_1}\|_{\rho_2}\, \mathcal{G}_c^{\mathfrak{n}} \not\preceq \mathcal{G}^{\mathfrak{n}}$ 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 Figure 7.9(a). The controller assumption needs not be weaken since $\bar{\mathcal{G}}_c^{\mathfrak{n}}/\mathcal{A}_c \equiv \bar{\mathcal{G}}_c^{\mathfrak{n}}$ and a simpler un-normalized version of $\bar{\mathcal{G}}_c^{\mathfrak{n}}$ can then be computed as in Figure 7.9(b). The new composition of the tank and controller contracts can now satisfy the desirable specification $\mathcal{C}$.

Chapter 8

Conclusion

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 for specifying components in terms of operational behaviors and extend TMs, which were introduced to represent only homogeneous tag systems, to the heterogeneous context. We subsequently propose a contract methodology for synchronizing heterogeneous components based on a set of useful operations on TMs such as composition, quotient and refinement. In addition, it is often desirable to verify if a general requirement $\mathcal{C}$ can be decomposed into a set of requirements $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$. To this end, we have presented a set of decomposing conditions for verifying such decomposition of a contract into a set of contracts in the homoge-

neous context and a pair of contracts in the heterogeneous context. To provide for a complete design methodology, we have also proposed synthesis strategies which can correct wrong contracts causing the condition failure in both contexts.

Our future work includes the implementation of our proposed theoretical contract framework and perform extensive evaluation on analysis and verification performance.

Bibliography

[1] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[2] Rajeev Alur, Thomas A. Henzinger, Orna Kupferman, and Moshe Y. Vardi. Alternating refinement relations. In *CONCUR '98*, pages 163–178, London, UK, 1998. Springer-Verlag.

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

[4] Sebastian S. Bauer, Alexandre David, Rolf Hennicker, Kim Guldstrand Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wasowski. Moving from specifications to contracts in component-based design. In *FASE*, pages 43–58, 2012.

[5] Albert Benveniste, Benoît Caillaud, Luca P. Carloni, Paul Caspi, and Alberto L. Sangiovanni-Vincentelli. Composing heterogeneous reactive systems. *ACM Trans. Embed. Comput. Syst.*, 7:43:1–43:36, 2008.

[6] Albert Benveniste, Benoît Caillaud, Luca P. Carloni, and Alberto Sangiovanni-Vincentelli. Tag machines. In *EMSOFT*, pages 255–263, 2005.

[7] Albert Benveniste, Benoît Caillaud, Alberto Ferrari, Leonardo Mangeruca, Roberto Passerone, and Christos Sofronis. Multiple viewpoint contract-based specification and design. In *FMCO*, pages 200–225, 2007.

[8] Luca Benvenuti, Alberto Ferrari, Leonardo Mangeruca, Emanuele Mazzi, Roberto Passerone, and Christos Sofronis. A contract-based formalism for the specification of heterogeneous systems. In *Proceedings of the Forum on Specification, Verification and Design Languages*, pages 142–147, Stuttgart, Germany, September 23–25, 2008.

[9] Simon Bliudze and Joseph Sifakis. The algebra of connectors: Structuring interaction in BIP. *IEEE Transactions on Computers*, 57(10):1315–1330, 2008.

[10] Matteo Bordin and Tullio Vardanega. Correctness by construction for high-integrity real-time systems: A metamodel-driven approach. In *Reliable Software Technologies Ada-Europe 2007*, volume 4498 of *LNCS*, pages 114–127. Springer Berlin Heidelberg, 2007.

[11] Roberto Cavada, Alessandro Cimatti, Anders Franzén, Krishnamani Kalyanasundaram, Marco Roveri, and R. K. Shyamasundar. Computing predicate abstractions by integrating bdds and smt solvers. In *FMCAD'07*, pages 69–76, Washington, DC, USA, 2007. IEEE Computer Society.

[12] Krishnendu Chatterjee and ThomasA. Henzinger. Assume-guarantee synthesis. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4424 of *Lecture Notes in Computer Science*, pages 261–275. Springer Berlin Heidelberg, 2007.

[13] John Cheesman and John Daniels. *UML components: a simple process for specifying component-based software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[14] A. Cimatti, L. Palopoli, and Y. Ramadian. Symbolic computation of schedulability regions using parametric timed automata. In *Real-Time Systems Symposium, 2008*, pages 80–89, Dec. 3 2008.

[15] A. Cimatti and S. Tonetta. A property-based proof system for contract-based design. In *38th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, pages 21–28, 2012.

[16] Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. Nusmv: A new symbolic model checker. *STTT*, 2(4):410–425, 2000.

[17] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. Hydi: A language for symbolic hybrid systems with discrete interaction. In *EUROMICRO-SEAA*, pages 275–278, 2011.

[18] Antonio Coronato, Antonio d'Acierno, Diego D'Ambrosio, and Giuseppe De Pietro. Supporting tools for designing-by-contract in component-based applications. In *Metainformatics*, pages 1–13, 2004.

[19] Werner Damm, Hardi Hungar, Bernhard Josko, Thomas Peikenkamp, and Ingo 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*, Grenoble, France, 2011.

[20] Abhijit Davare, Douglas Densmore, Liangpeng Guo, Roberto Passerone, Alberto L. Sangiovanni-Vincentelli, Alena Simalatsar, and

Qi Zhu. METROII: A design environment for cyber-physical systems. *ACM Trans. on Embed. Comput. Syst.*, 12(1s):49:1–49:31, March 2013.

[21] Luca de Alfaro and Thomas A. Henzinger. Interface automata. *ACM SIGSOFT*, 26:109–120, 2001.

[22] Willem-Paul 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.

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

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

[25] Edsger W. Dijkstra. Guarded commands, non-determinancy and a calculus for the derivation of programs. In *Language Hierarchies and Interfaces*, pages 111–124, 1975.

[26] Desmond Francis D'Souza and Alan Cameron Wills. *Objects, Components, and Frameworks with UML: The Catalysis Approach*. Addison-Wesley Professional, 1998.

[27] EADS Innovation Works. Case study on distributed heterogeneous communication systems, 2009, http://www.combest.eu/home/?link=Application1.

[28] Uli Fahrenberg, Axel Legay, and Andrzej Wasowski. Make a difference! (semantically). In *MoDELS*, pages 490–500, 2011.

[29] Thomas A. Henzinger and Dejan Ničković. Independent implementability of viewpoints. In *Proceedings of the 17th Monterey conference on Large-Scale Complex IT Systems: development, operation and management*, pages 380–395, Berlin, Heidelberg, 2012. Springer-Verlag.

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

[31] K.G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1):134–152, 1997.

[32] Kim G. Larsen, Ulrik Nyman, and Andrzej Wasowski. Modal i/o automata for interface and product line theories. In *Proceedings of the 16th European conference on Programming*, ESOP'07, pages 64–79, Berlin, Heidelberg, 2007. Springer-Verlag.

[33] Kim Guldstrand Larsen. Modal specifications. In *Automatic Verification Methods for Finite State Systems*, pages 232–246, 1989.

[34] Thi Thieu Hoa Le, Luigi Palopoli, Roberto Passerone, Yusi Ramadian, and Alessandro Cimatti. Parametric analysis of distributed firm real-time systems: A case study. In *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*, pages 1–8, 2010.

[35] Thi Thieu Hoa Le, Roberto Passerone, Uli Fahrenberg, and Axel Legay. A tag contract framework for heterogeneous systems. In Carlos Canal and Massimo Villari, editors, *Advances in Service-Oriented and Cloud Computing*, volume 393 of *Communications in Computer and Information Science*, pages 204–217. Springer Berlin Heidelberg, 2013.

[36] Thi Thieu Hoa Le, Roberto Passerone, Uli Fahrenberg, and Axel 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, July 8–10, 2013.

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

[38] Shang-Wei Lin and Pao-Ann Hsiung. Counterexample-guided assume-guarantee synthesis through learning. *IEEE Transactions on Computers*, 60(5):734–750, 2011.

[39] Xiaojun Liu, Yuhong Xiong, and Edward A. Lee. The Ptolemy II framework for visual languages. In *HCC*, pages 50–51. IEEE, 2001.

[40] Nancy A. Lynch and Mark R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *PODC*, pages 137–151, 1987.

[41] A. Marchand and M. Silly-Chetto. Qos scheduling components based on firm real-time requirements. In *Computer Systems and Applications, 2005. The 3rd ACS/IEEE International Conference on*, page 141, 2005.

[42] Bertrand Meyer. Applying "Design by contract". *Computer*, 25(10):40–51, 1992.

[43] Marco Natale. Design and development of component-based embedded systems for automotive applications. In *Reliable Software Technologies Ada-Europe 2008*, volume 5026 of *LNCS*, pages 15–29. Springer Berlin Heidelberg, 2008.

[44] A precision clock synchronization protocol for networked measurement and control systems. Ieee standard 1588-2002, November 2002.

[45] Jean-Baptiste Raclet, Eric Badouel, Albert Benveniste, Benoît Caillaud, Axel Legay, and Roberto Passerone. A modal interface theory for component-based design. *Fundam. Inform.*, 108(1-2):119–149, 2011.