Assume-Guarantee Reasoning for Additive Hybrid Behaviour

Pieter J. L. Cuijpers[1,2], Jonas Hansen[3](✉), and Kim G. Larsen[3]

[1] Technische Universiteit Eindhoven, Eindhoven, The Netherlands
p.j.l.cuijpers@tue.nl
[2] Radboud Universiteit, Nijmegen, The Netherlands
[3] Aalborg University, Aalborg, Denmark
{jonash,kgl}@cs.aau.dk

Abstract. Hybrid Automata describe dynamical systems where continuous behaviour interacts with discrete events. Resource Timed Automata (RTA), a subset of Hybrid Automata, adopt an additive composition scheme, in which discrete behaviour of components is executed concurrently, time is synchronized, and the evolution of continuous variables is arithmetically added up. Additive composition facilitates modelling and analysis of cumulative properties of continuous variables, such as conservation laws, typically manifested as the balancing of real-valued variables. In this paper, we present and exemplify an assume-guarantee framework aimed at additive compositional reasoning in the setting of hybrid systems. Crucially, we introduce a notion of refinement on so-called Resource Hybrid Automata (RHA), and show that it is a pre-congruence for additive composition. Furthermore - crucial for our assume-guarantee framework – we show that RHAs are closed under conjunction and admit a so-called quotient constructions (a dual operator to parallel composition). Finally, we demonstrate how the Statistical Model Checking (SMC) engine of the tool UPPAAL may be used to efficiently falsify refinements.

Keywords: Assume-Guarantee Reasoning · Hybrid Specification Theory · Resource Hybrid Automata · Additive Composition

1 Introduction

Hybrid Automata (HA) [22] are an extension of Timed Automata (TA) [2] combining timed discrete events with the continuous evolution of real-valued variables. Resource-, Priced-, Energy- and Weighted- Timed Automata [3,8,16,29,31] all define strict subsets of HA in which continuous dynamics cannot affect the timed discrete semantics of a system. These formalisms differ in their mechanics for dealing with composition. Of particular note are Resource Timed Automata (RTA) which adopt a so-called additive composition scheme aimed at simplifying the analysis of conservation laws e.g. resource balancing.

In this paper we consider component-wise reasoning for additive composition over hybrid variables, henceforth referred to as resources, and study the hybrid extension of RTA, namely Resource Hybrid Automata (RHA).

Additive composition as studied in [16] extends the usual concurrent execution of discrete behaviour in synchronized time with a notion of resource accumulation both in terms of evolution and flow. This particular composition scheme, has, to the best of our knowledge, not received much attention in previous literature on hybrid systems. We argue that conservation laws follow naturally from additive composition and show that accumulation of shared resources across components directly corresponds to the problem of balancing said resources. We consider how behavioural requirements of a composite system can be expressed as a number of local and concise requirements on open additive parallel components. This kind of compositional reasoning is known as specification theory.

A *specification theory* interprets specifications as abstract under-specified descriptions of behaviour, which can generally be thought of as requirements to implementations. Formally, this is captured by a *satisfaction relation* between implementations and specifications, inducing for each specification S the set $[S]$ of the implementations satisfying it. Crucial to a specification theory is a notion of *refinement* between specifications. Refining a specification should result in a new specification which is stricter in terms of its implementation space. Now, a specification theory should also allow for both logical and structural *composition* of specifications.

In order to support step-wise refinement and compositional reasoning, it is vital that the notion of refinement is a pre-order over specifications (we write $S \sqsubseteq T$ to say that specification S refines T) and it must be a pre-congruence with respect to a composition of interest (we write $S \parallel T$ for a composition of two specifications). A fully expressive specification theory should ideally contain a *quotient* operator, a dual operator to the composition of specifications. If T is an overall specification of a composite system $P_1 \parallel P_2$ and S is a component specification for P_1, then the quotient specification $T \mathbin{\backslash\!\!\backslash} S$ is the weakest requirement to the component P_2 in order for the composite system to satisfy T.

Mathematically, we have $S \parallel (T \mathbin{\backslash\!\!\backslash} S) \sqsubseteq T$ for the quotient $T \mathbin{\backslash\!\!\backslash} S$ of specifications S and T, and for every specification Q with $S \parallel Q \sqsubseteq T$ we have $Q \sqsubseteq T \mathbin{\backslash\!\!\backslash} S$. In the setting of sequential, imperative programs Dijkstra's celebrated notions of *weakest precondition* and *strongest postcondition* effectively provide quotient constructs for sequential composition with respect to pre-and-post-condition specification pairs. Here He Jifeng has made seminal work with C.A.R. Hoare [24,26].

For concurrent systems, there have historically been two schools of specification theories: *Process Algebra* [6,23,34] and *Temporal Logic* [36]. In Process Algebra, specifications are process expressions with a variety of proposals for refinement orderings: e.g. trace-inclusion, bisimulation equivalence, ready-simulation, simulation, and failure-trace inclusion as reported in the linear-branching time spectrum [15]. In the late 80-ties, He Jifeng played a central role in improving the failure semantics for CSP [17,18]. In later work He Jifeng has proposed refinements for more complex settings including real-time [37], service- and object-oriented systems [19,21]. Moreover, Process Algebra has strong support for structural composition of specification, but lack general support for logical composition. In Temporal Logic, specifications are logical formula and refinement is simply logical implication. Temporal Logic has by nature full support for logical composition of specifications, but lack in general support for structural composition. What we seek is a specification oriented theory that unifies logical and process algebra frameworks, a theme that Hoare and He have developed in the unified theory of programming [20,25].

Also, in the late 80-ties, the notion of Modal Transition Systems [7,30,32,33] was introduced by Larsen and Thomsen as a means to provide a specification theory supporting both logical and structural composition. This theory of Modal Transition Systems has later been extended to the setting of timed as well as probabilistic systems [9,10,14]. The contributions of this paper may be seen as an extension of Modal Transition Systems to resource-aware concurrent systems. A particularly useful type of specification theories are those based on the notion of *contract*, first developed and promoted in the community of software engineering. So-called "design-by-contract", popularized by Bertrand Meyer, has roots in the classical work by Owicki-Gries [35] extending Floyd-Hoare logic (for sequential imperative programs) to the setting of concurrently executing programs, where interference on shared variables may occur. In the concurrent setting, contract specifications come as *Assume-Guarantee* (or Rely-Guarantee) pairs, where the Assumption states conditions on the effect on the shared variables by the system's environment, and the Guarantee are obligations of the systems operation on the shared variables. Early contributions were made by Jones [27], Abadi, Lamport and Wolper [28]. Later, He Jifeng together with Qiwen Xu and Willem-Paul de Roever gave a sound and complete proof system for rely-guarantee assertions [38] providing a compositional reformulation of the non-compositional Owicki-Gries method. More recently the notion of *interface automata* was introduced by Alfaro and Henzinger [1] and since then a number of frameworks has been proposed that can be seen as instances of contracts theories, with [5] providing a recent "meta-theory" of contracts and its application to software and systems.

Returning to general specification theories, we note that the quotient operator defines how a component helps to achieve a target behaviour T *for the system as a whole*, given an assumption S on its environment. In fact, it is shown in [4,12] how a contract framework can be built in a generic way on top of any specification theory which supports refinement, composition and quotienting of specifications. The resulting contract framework lifts refinement to the level of contracts and proposes a notion of contract composition on the basis of dominating contracts. In particular, it has been shown in [12] that one can weaken a guarantee G of an assume-guarantee contract, under assumption A: the weakened guarantee, denoted $G \gg A$, is simply $(A \parallel G) \setminus\!\!\setminus A$ and provides a combined specification equivalent to the original (A, G) pair. In this paper, we develop a complete specification theory aimed at the additive hybrid setting.

We characterize compositional reasoning within this domain and introduce a notion of refinement. We define appropriate products for the crucial operations logical and structural composition, together with its dual, namely the quotient product. Using this theory we show how assume-guarantee reasoning aimed at hybrid additivity is possible over specifications described by RHAs. In addition, we show that simulation based methods can be used to refute the existence of certain refinements.

Section 2 introduces our modelling language. Section 3 characterizes compositional reasoning and formally defines an additive hybrid specification theory. Section 4 formally introduces the assume-guarantee aspect and exemplifies it. Section 5 discusses practical computation methods for ascertaining refinement. Section 6 concludes our findings and discuss potential future research directions.

2 Resource Hybrid Automata

We first define our modelling language, followed by the characterization of an additive hybrid specification theory. We introduce Resource Hybrid Automata (RHA), which fundamentally define Linear Hybrid Automata [22] subject to additive composition as adopted by Resource Timed Automata (RTA) [16].

By $\mathcal{V} = \mathcal{V}^G \uplus \mathcal{V}^L$ we denote a partitioned set of real-valued global and local variables, which we think of as (non-)shared resources. We write $\sigma : \mathcal{V} \to \mathbb{R}$ or alternatively $\sigma \in \mathbb{R}^{\mathcal{V}}$ for a valuation of such variables. These are split into $\sigma_L : \mathcal{V}^L \to \mathbb{R}$ and $\sigma_G : \mathcal{V}^G \to \mathbb{R}$ in the obvious way. Furthermore, we define the following arithmetic over $\mathbb{R}^{\mathcal{V}}$: Let $\sigma, \sigma' \in \mathbb{R}^{\mathcal{V}}$ and $v \in \mathcal{V}$ then $(\sigma + \sigma')(v) = \sigma(v) + \sigma'(v)$ and dually for its inverse $-$. Additionally, let $v \in \mathcal{V}$ then $\mathbf{0}(v) = 0$. Thus valuations define the abelian group $(\mathbb{R}^{\mathcal{V}}, +, \mathbf{0})$.

Resources are subject to discrete updates and can be tested in constraints. To allow reasoning about resource additivity we require both our update- and constraint-algebra to be closed under negation, conjunction, addition $\oplus$ and quotient $\oslash$. Furthermore, we require that constraints are closed under side-effects $\triangleright$ which, for the sake of intuition and brevity, we characterize as updates.

Much like we can think of a conjunction of two updates/constraints as the join of their respective influence, i.e. what they have in common, an addition $\oplus$ is what they can do together, i.e. their arithmetic accumulated influence. A quotient $\oslash$ is the dual of addition. Here the question is what makes it possible to fulfil our goal (left operand) if we add it to something that already exists (right operand). We will see later how these operations naturally provide mechanisms for resource additivity.

Definition 1 (Resource Update). *We characterize the set of updates $\mathcal{U}(\mathcal{V})$ over $\mathcal{V}$ by the following abstract syntax:*

$$u ::== TT \mid \neg u \mid \epsilon \mid R \mid u \wedge u \mid u \oplus u \mid u \oslash u$$

where $R \subseteq \mathcal{V}$. Let $u_1, u_2 \in \mathcal{U}(\mathcal{V})$ be updates, then the evaluation of valuations $\sigma, \sigma' \in \mathbb{R}^{\mathcal{V}}$ in u denoted $(\sigma, \sigma') \models u$ is defined inductively on the structure of u:

- $(\sigma, \sigma') \models \mathit{TT}$:
- $(\sigma, \sigma') \models \neg u \Leftrightarrow (\sigma, \sigma') \not\models u$:
- $(\sigma, \sigma') \models \epsilon \Leftrightarrow \sigma = \sigma'$:
- $(\sigma, \sigma') \models R \Leftrightarrow \sigma'(v) = \begin{Bmatrix} 0, & v \in R, \\ \sigma(v), & v \notin R \end{Bmatrix}$:
- $(\sigma, \sigma') \models u \wedge u' \Leftrightarrow (\sigma, \sigma') \models u \wedge (\sigma, \sigma') \models u'$:
- $(\sigma, \sigma') \models u \oplus u' \Leftrightarrow \exists \varsigma, \varsigma' \in \mathbb{R}^{\mathcal{V}} : (\sigma, \varsigma) \models u \wedge (\sigma, \varsigma') \models u' \wedge \sigma' = \varsigma + \varsigma' - \sigma$:
- $(\sigma, \sigma') \models u \oslash u' \Leftrightarrow \forall \varsigma \in \mathbb{R}^{\mathcal{V}} : (\sigma, \varsigma) \models u' \Rightarrow (\sigma, \sigma' + \varsigma - \sigma) \models u$.

Definition 2 (Resource Constraint). *We characterize the set of constraints $\mathcal{C}(\mathcal{V})$ over $\mathcal{V}$ by the following abstract syntax:*

$$c ::== \mathit{TT} \mid \neg c \mid \sum_{i=1}^{n} v_i \cdot r_i \bowtie r \mid u \triangleright c \mid c \wedge c \mid c \oplus c \mid c \oslash c$$

where $r, r_i \in \mathbb{Q}$, $v_i \in \mathcal{V}$, $u \in \mathcal{U}(\mathcal{V})$ and $\bowtie \in \{\leq, \geq, ==, <, >\}$. Let u be an update, c' be a constraint, then the evaluation of valuation $\sigma \in \mathbb{R}^{\mathcal{V}}$ in c denoted $\sigma \models c$ is defined inductively on the structure of c:

- $\sigma \models \mathit{TT}$:
- $\sigma \models \neg c \Leftrightarrow \sigma \not\models c$:
- $\sigma \models \sum_{i=1}^{n} v_i \cdot r_i \bowtie r \Leftrightarrow \sum_{i=1}^{n} \sigma(v_i) \cdot r_i \bowtie r$:
- $\sigma \models u \triangleright c \Leftrightarrow \exists \sigma' \in \mathbb{R}^{\mathcal{V}} : (\sigma, \sigma') \models u \wedge \sigma' \models c$:
- $\sigma \models c \wedge c' \Leftrightarrow \sigma \models c \wedge \sigma \models c'$:
- $\sigma \models c \oplus c' \Leftrightarrow \exists \varsigma, \varsigma' \in \mathbb{R}^{\mathcal{V}} : \varsigma \models c \wedge \varsigma' \models c' \wedge \sigma = \varsigma + \varsigma'$:
- $\sigma \models c \oslash c' \Leftrightarrow \forall \sigma' \in \mathbb{R}^{\mathcal{V}} : \sigma' \models c' \Rightarrow \sigma + \sigma' \models c$.

As such, updates can reset sets of resources to zero and constraints define systems of linear inequalities over resources.

Definition 3 (Resource Hybrid Automata). *We define a resource hybrid automaton (RHA) as a tuple:*

$$H = \langle \mathrm{L}, \mathrm{l}_0, \mathcal{V}, E, \mathrm{inv}, \mathrm{rate} \rangle$$

where L *is a finite set of modes,* $\mathrm{l}_0 \in \mathrm{L}$ *is the initial mode,* $\mathcal{V} = \mathcal{V}_L \uplus \mathcal{V}_G$ *is a finite partitioned set of local and global variables,* $E \subseteq \mathrm{L} \times \mathcal{C}(\mathcal{V}) \times \mathcal{U}(\mathcal{V}) \times \mathrm{L}$ *is a finite set of edges,* $\mathrm{inv} : \mathrm{L} \to \mathcal{C}(\mathcal{V})$ *assigns an invariant constraint on resources to each mode and* $\mathrm{rate} : \mathrm{L} \to \mathcal{C}(\mathcal{V})$ *assigns a constraint to the first derivative of resources to each mode.*

An example RHA can be seen in Fig. 2.

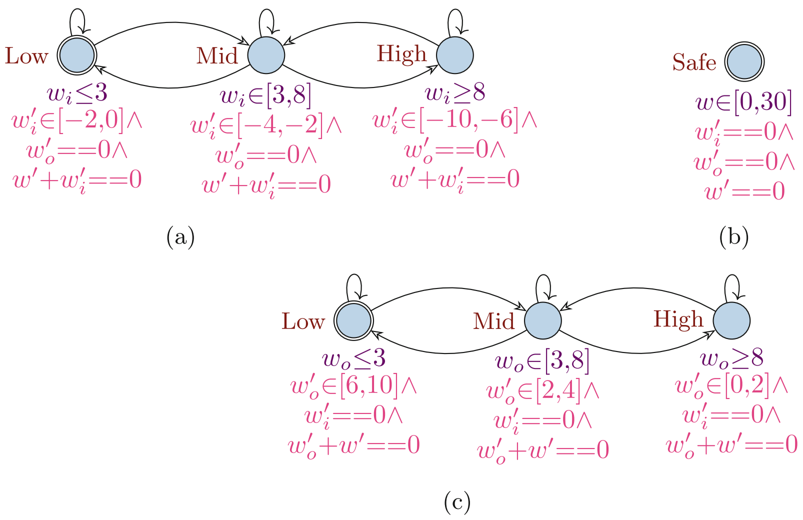


Fig. 1. (a) An RHA specification modelling a water intake valve. Depending on reservoir capacity (captured by global resource w_i), the water flow allowed by the valve is either in the interval $[-2, 0]$, $[-4, -2]$ or $[-10, -6]$. (b) An RHA specification modelling a water output valve. Depending on reservoir capacity (captured by global resource w_o), the water flow allowed by the valve is either in the interval $[6, 10]$, $[2, 4]$ or $[0, 2]$. (c) An RHA specification modelling the safe capacity of the pump's private water tank (captured by global resource w).

Let's consider a physical system, a water pump controller, governing two flow valves. One valve is connected to an input reservoir (characterized by resource w_i), which captures how the environment can make water available to the pump. Another valve is connected to an output reservoir (characterized by resource w_o), which captures how the pump can make water available to the environment. Both are connected to an internal water tank (characterised by resource w). Our system simply keeps track of capacity by regulating water flow from/to the environment using an internal tank for temporary storage. Figure 1 shows three RHAs, each modelling a component of our water pump controller.

Syntactically, RHA's are hybrid automata defined over additive oriented update and constraint algebra. Semantically their hybrid dynamics are interpreted as timed semantics. Specifically, the semantics of an RHA H denoted $[\![H]\!]$ is defined by a Resource Timed Transition System.

Definition 4 (Resource Timed Transition System). *We define a resource timed transition system (RTS) as a tuple* $S = \langle \mathrm{X}, \chi_L, \rightarrow \rangle$ *where* $\mathrm{X} = \mathrm{X}_L \times \mathrm{X}_G$ *is a set of pairs of local and global states,* $\chi_L \in \mathrm{X}_L$ *is the initial local state, and* $\rightarrow \subseteq \mathrm{X} \times (\{\tau\} \cup \mathbb{R}_{\geq 0}) \times \mathrm{X}$ *is a timed transition relation, where discrete transitions are denoted by* τ*. We write* $(x_L, x_G) \xrightarrow{\gamma} (x_L', x_G')$ *whenever* $((x_L, x_G), \gamma, (x_L', x_G')) \in \rightarrow$*. Similarly, we write* $(x_L, x_G) \overset{\gamma}{\not\rightarrow}$ *whenever* $\forall x' \in \mathrm{X} : ((x_L, x_G), \gamma, x') \notin \rightarrow$*. Furthermore, we require that states form an abelian group*

$(\mathrm{X}, +, \boldsymbol{0})$. *For convenience we refer to the inverse of* $+$ *as* $-$. *We call a local state* $x_L \in \mathrm{X}_L$ *reachable if there exists a sequence* $x^i \xrightarrow{\gamma} x^{i+1}$ *with* $0 \leq i \leq n$ *of transitions such that* $x_L^0 = \chi_L$ *and* $x_L^n = x_L$.

Definition 5 (RHA Semantics). *Let* $H = \langle \mathrm{L}, \mathrm{l}_0, \mathcal{V}, E, \mathrm{inv}, \mathrm{rate} \rangle$ *be an RHA. We define its semantics as an RTS* $[\![H]\!] = \langle \mathrm{X}, \chi_L, \rightarrow \rangle$ *in which the local states are defined as products of a mode and the valuation of local variables* $\mathrm{X}_L = \mathrm{L} \times \mathbb{R}^{\mathcal{V}_L}$, *the global states are defined as valuations of global variables* $\mathrm{X}_G = \mathbb{R}^{\mathcal{V}_G}$, *the initial local state is given by* $\chi_L = (\mathrm{l}_0, \boldsymbol{0})$, *and* $\rightarrow$ *is the smallest relation satisfying:*

- *If* $(l, c, u, l') \in E$, $\sigma \in \mathbb{R}^{\mathcal{V}}$, $\sigma \models \mathrm{inv}(l)$, $\sigma \models c$, $(\sigma, \sigma') \models u$ *and* $\sigma' \models \mathrm{inv}(l')$, *then* $((l, \sigma_L), \sigma_G) \xrightarrow{\tau} ((l', \sigma'_L), \sigma'_G)$*:*
- *If* $l \in \mathrm{L}$, $\delta \in \mathbb{R}_{\geq 0}$, *and* $\phi : [0, \delta] \rightarrow \mathbb{R}^{\mathcal{V}}$ *is a right-differentiable function with piece-wise constant derivative, such that for all* $t \in [0, \delta]$ *we have* $\phi(t) \models \mathrm{inv}(l)$ *and for* $t \in [0, \delta)$ *we have* $\frac{d}{dt}\phi(t) \models \mathrm{rate}(l)$, *then* $((l, \phi_L(0)), \phi_G(0)) \xrightarrow{\delta} ((l, \phi_L(\delta)), \phi_G(\delta))$.

Discrete transitions in the automata defines τ transitions in its semantics, which we generally think of as internal.

Note that restricting flow behaviour to right-differential piece-wise constant solutions is a standard way to avoid problems with finite-set refutability when considering hybrid dynamics over a timed semantics [11]. Furthermore, any RTS generated by an RHA are time-reflexive and time-additive.

Theorem 1. *Let* H *be an RHA. The following holds for* $[\![H]\!] = \langle \mathrm{X}, \chi_L, \rightarrow \rangle$*: For all* $x, x' \in \mathrm{X}$, $\delta, \delta' \in \mathbb{R}_{\geq 0}$ *we have:* $[\![H]\!]$ *is* ***time reflexive*** $x \xrightarrow{0} x$, *and* $[\![H]\!]$ *is* ***time additive*** $x \xrightarrow{\delta+\delta'} x' \Rightarrow \exists x'' \in \mathrm{X} : x \xrightarrow{\delta} x'' \wedge x'' \xrightarrow{\delta'} x'$.

3 Compositional Reasoning

With RHA and its semantic interpretation RTS we have a formal characterization of hybrid behaviour. We now turn our attention to component-wise abstraction and realization in terms of behavioural requirements. We start by motivating component-wise design and refinement with additivity using a simple example.

Our goal is to design a light controller, the kind that might be used to control the alternating blinking pattern of a warning light on top of an antenna or maybe a control console. There are a few rules we need to follow: the controller must provide an alternating light pattern, i.e. it must facilitate blinking mechanics for our light determined by some intervals, the light itself is limited in how much power it can consume depending on its brightness level, and it must interact with the electrical grid indirectly through a single battery. As such we are designing an open component, since we are only concerned with saturation. How the battery charges is handled by a different controller.

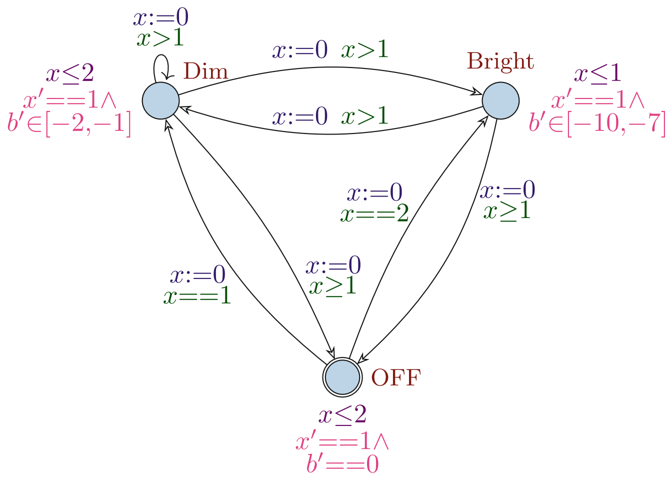


Fig. 2. An RHA, modelling a warning light controller. It has three modes; Dim, Bright and the initial mode OFF, each describes a distinct brightness. It is defined over two resources x (local) and b (global). The former characterizes a clock with constant rate 1 in all modes and is used to model timing behaviour. The latter describes a battery. It is used to model the light controllers interaction with its environment. Each mode (or rather each brightness level) define distinct battery saturation rates. Observe that when we interpret the model as blueprint for timed discrete requirements, infinitely many distinct blinking- and saturation-patterns are possible.

A system architect has provided us with a blueprint defined as an RHA that formally captures the rules we just discussed, which is depicted in Fig. 2. Of course, the light controller is only a single component of a larger system, which we collectively think of as the environment. Now, from the environment's perspective, the blueprint captures exactly what is assumed about our controller. On top of that, potential conservation rules in terms of energy for our battery is handled by the environment as well. Looking at Fig. 2 we immediately see the benefit of additive reasoning, since no consideration for how resource b i.e. our battery interacts with other components is necessary, it is simply assumed that the collective interaction over b across all components is handled in an additive manner.

Figure 2 defines abstract requirements. We now desire to create a concrete controller that behaves according to these requirements. Because it needs to be concretely realizable a few precautions needs to be considered; The controller must be specific in its battery saturations, it is not allowed to stop time, and any discrete behaviour whenever enabled must occur. To that end we introduce the model depicted in Fig. 3.

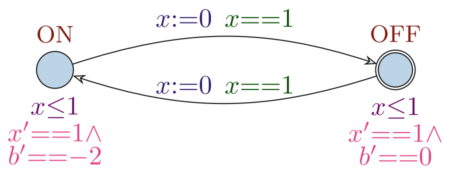


Fig. 3. An RHA, modelling a concrete realization of the light controller depicted in Fig. 2. It consists of two modes On and the initial mode OFF. Like Fig. 2, it is defined over the two resources x and b. Observe that this concrete light controller simply defines an altering pattern in which the light is turned off for 1 time unit, after which it is turned on for 1 time unit, during which it consumes exactly 2 units of power per time unit.

Now the question is, can we substitute our abstract model of Fig. 2 with the concrete model of Fig. 3 whenever we consider the light controller in other components of our system. To do so, we must ascertain whether the abstract controller can mimic any an all timed-discrete behaviour allowed by the concrete controller in all environments (in terms of the capacity of b and how it is charged). If this is indeed the case, then we do not have to consider the cumbersome abstract model whenever we desire to reason about our light controller, the simple concrete model is sufficient. This property is called refinement.

Adopting the usual terminology from specification theory we refer to RTS's as specifications. In this paper, we consider specifications that model the reaction of a system to changes in its environment. This means that a refinement of one specification into another should preserve those reactions, given a particular environment. (Note that in other works, other interpretations of transition systems lead to different notions of refinement. E.g. in [5], changes in the environment are explicitly modeled as input transitions of the RTS's, leading to a two-way type of simulation refinement).

Definition 6 (Refinement). *For $i \in \{1,2\}$, let $S^i = \langle \mathrm{X}^i, \chi_L^i, \rightarrow_i \rangle$ be a specification. We assume that their global states are shared $\mathrm{X}_G^1 = \mathrm{X}_G^2$. We say that S^1 refines S^2, denoted $S^1 \sqsubseteq S^2$ iff there exists a binary relation $R \subseteq \mathrm{X}_L^1 \times \mathrm{X}_L^2$ defined over the local states such that $\chi_L^1 R \chi_L^2$ and for all $x^1 \in \mathrm{X}^1$ and $x^2 \in \mathrm{X}^2$ with $x_G^1 = x_G^2$ we find:*

- *If $x_L^1 R x_L^2$ and $x^1 \xrightarrow{\tau}_1 y^1$, then there exists $y^2 \in \mathrm{X}^2$ such that $x^2 \xrightarrow{\tau}_2 y^2$ with $y_L^1 R y_L^2$ and $y_G^1 = y_G^2$.*
- *If $x_L^1 R x_L^2$ and $x^1 \xrightarrow{\delta}_1 y^1$ for some $\delta \in \mathbb{R}_{\geq 0}$, then there exists $y^2 \in \mathrm{X}^2$ such that $x^2 \xrightarrow{\delta} y^2$ with $y_L^1 R y_L^2$ and $y_G^1 = y_G^2$.*

Crucially, refinements between specifications form a pre-order.

Theorem 2. *The refinement relation $\sqsubseteq$ is a pre-order over the set of all specifications.*

In specification theory, specifications for which there is a concrete realization in practice are called implementations. These implementations may occur at any place in the pre-order, as it is often possible to further refine behaviour of an already existing implementation, hence creating a refinement of that implementation. Without fixing the implementation mechanisms, we cannot determine which specifications are realizable precisely. However, we can rule out any specifications that are self-contradictory, block the progress of time, or contain unresolved non-deterministic choices. A specification that has these properties, such as Fig. 3, we call an implementation in this paper.

Definition 7 (Implementation). *Let $S = \langle \mathrm{X}, \chi_L, \rightarrow \rangle$ be a specification, we say that S is an implementation if furthermore the following holds:*

- ***Independent progression**: For every reachable state $x \in \mathrm{X}$ there exists $\gamma \in \{\tau\} \cup \mathbb{R}_{\geq 0}$ and $x' \in \mathrm{X}$ such that $x \xrightarrow{\gamma} x'$;*
- ***Discrete-determinism**: For every reachable state $x, x', x'' \in \mathrm{X}$, with $x \xrightarrow{\tau} x'$ and $x \xrightarrow{\tau} x''$, we find $x = x''$;*
- ***Time-determinism**: For every reachable state $x, x', x'' \in \mathrm{X}$ and every $\delta \in \mathbb{R}_{\geq 0}$ with $x \xrightarrow{\delta} x'$ and $x \xrightarrow{\delta} x''$ we find $x' = x''$;*
- ***Urgency**: For every reachable state $x \in \mathrm{X}$, if there exists $x' \in \mathrm{X}$ with $x \xrightarrow{\tau} x'$, then there does not exists an $x'' \in \mathrm{X}$ and $\delta \in \mathbb{R}_{\geq 0}$ with $\delta > 0$ and $x \xrightarrow{\delta} x''$.*

The syntactic notion of implementations naturally follows.

Definition 8 (Implementation RHA). *Let H be an RHA. We say that H is an implementation RHA whenever $[\![H]\!]$ is an implementation.*

The concrete light controller of Fig. 3 defines an implementation, another example is shown in Fig. 4.

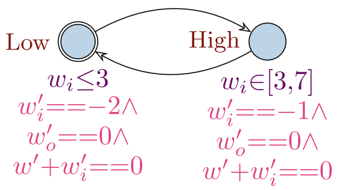


Fig. 4. An implementation RHA implementing the intake valve specification of Fig. 1a. Much like Fig. 3, the intake valve implementation is in a sense a restriction of behaviour.

An implementation is said to satisfy a specification if it only admits discrete/continuous behaviour allowed by the specification. The general notion of satisfaction is formally captured by refinement.

Definition 9 (Specification Satisfaction). *Let S be a specification and I an implementation. If $I \sqsubseteq S$ we say that I satisfies S. By $[S] = \{I \mid I \sqsubseteq S\}$ we denote the set of all implementations that satisfy S.*

This gives us a natural way of thinking about specification equivalence.

Definition 10 (Specification Equivalence). *We say that specifications S and S' are equivalent, denoted $S \approx S'$ iff $[S] = [S']$.*

It is not difficult to see that we could define specifications that could never be satisfied, e.g. if they contain contradictions. In practice, only those admitting implementations are of interest. This notion is usually referred to as consistency, and we characterize it using implementation spaces.

Definition 11 (Consistency). *We say that a specification S is consistent iff $[S] \neq \emptyset$.*

Compositional reasoning in terms of component-wise refinement and abstraction crucially relies on a well defined notion of refinement within which substitutability is guaranteed. This means that whenever we have a composition of two specifications, by replacing one of the constituents with a refining component, it results in a refinement of the original composition. We now define two such compositions that pertain substitutability over refinement, namely logical- and structural-composition.

Fig. 5. (a) A Specification, modelling the battery capacity of the light controller, whose timing behaviour is defined by the specification depicted in Fig. 2. It consists of a single mode, the initial mode, and is defined over the global resource b. It invariantly requires that the battery capacity never exceeds 1 unit of energy and is never saturated beyond depletion. Note that any and all discrete transitions are allowed, captured by the self loop. (b) A Specification, modelling a power supply. It consists of a single mode, the initial mode, and is defined over the global resource b. It does not admit any discrete behaviour, however it defines a constant charging of 1 unit of power per time unit.

Before we dive into the formal definitions, we first motivate their existence and usefulness. Going back to our light controller specification of Fig. 2, together with a possible implementation thereof, shown in Fig. 3. Our goal is to introduce mechanisms for reasoning about compositions of specifications. Consider the specification modelled in Fig. 5a, which captures battery capacity requirements. We would like to capture the notion of joint refinement, in the sense that Fig. 3 should both refine the timing behaviour of Fig. 2 and the capacity restrictions imposed by Fig. 5a. One can think of the specifications as two distinct aspects of the same component. Their joined requirements is exactly captured by their logical composition.

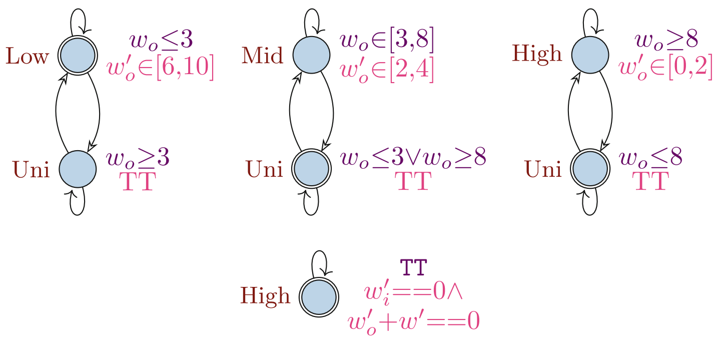


Fig. 6. Four RHA specifications whose logical composition refines the output valve specification of Fig. 1c, each modelling distinct responsibilities. All of them define a so-called universal mode Uni admitting an arbitrary flow of resource w_o up until some bound for which it must make a transition or define a flow that shift w_o away from said bound. In other words, each component do not care about w_o until its top mode invariant holds. Even though the output valve specification of Fig. 1c is relatively small, these three components are even smaller and arguably more intuitive.

Logical composition defines the joined behaviour of its constituents. As such an implementation satisfies the composition of two specifications if and only if both of these specifications are satisfied by said implementation.

Definition 12 (Logical Product). *Let* $S^j = \langle \mathrm{X}^j, \chi_L^j, \rightarrow_j \rangle$ *for* $j \in \{1,2\}$ *be a specification, where* $\mathrm{X}_G^1 = \mathrm{X}_G^2$. *We define the logical product of* S^1 *and* S^2, *denoted* $S^1 \wedge S^2$ *as a new specification:* $S^1 \wedge S^2 = \langle \mathrm{X}, \chi_L, \rightarrow \rangle$, *where* $\mathrm{X}_L = \mathrm{X}_L^1 \times \mathrm{X}_L^2$, $\mathrm{X}_G = \mathrm{X}_G^1$, $\chi_L = (\chi_L^1, \chi_L^2)$ *and* $\rightarrow$ *is the smallest relation satisfying:*

$$\frac{(x_L^1, x_G) \xrightarrow{\gamma}_1 (y_L^1, y_G) \qquad (x_L^2, x_G) \xrightarrow{\gamma}_2 (y_L^2, y_G)}{\big((x_L^1, x_L^2), x_G\big) \xrightarrow{\gamma} \big((y_L^1, y_L^2), y_G\big)} \gamma \in \{\tau\} \cup \mathbb{R}_{\geq 0}$$

The logical product admits a transition if and only if its constituents both admits it. On the syntactic level we can compute the logical composition by the following RHA construction.

Definition 13 (Logical Composition). *Let* $H^j = \langle \mathrm{L}^j, \mathrm{l}_0^j, \mathcal{V}, E^j, \mathrm{inv}^j, \mathrm{rate}^j \rangle$ *for* $j \in \{1,2\}$ *be an RHA, where* $\mathcal{V}_G^1 = \mathcal{V}_G^2$ *and* $\mathcal{V}_L^1 \cap \mathcal{V}_L^2 = \emptyset$. *We define the logical composition of* H^1 *and* H^2, *denoted* $H^1 \wedge H^2$ *as a new RHA:* $H^1 \wedge H^2 = \langle \mathrm{L}, \mathrm{l}_0, \mathcal{V}, E, \mathrm{inv}, \mathrm{rate} \rangle$, *where* $\mathrm{L} = \mathrm{L}^1 \times \mathrm{L}^2$, $\mathrm{l}_0 = (\mathrm{l}_0^1, \mathrm{l}_0^2)$, $\mathcal{V}_L = \mathcal{V}_L^1 \uplus \mathcal{V}_L^2$, $\mathcal{V}_G = \mathcal{V}_G^1$, $\mathrm{inv}\big((l^1, l^2)\big) = \mathrm{inv}^1(l^1) \wedge \mathrm{inv}^2(l^2)$, *rate* $\big((l^1, l^2)\big) = \mathrm{rate}^1(l^1) \wedge \mathrm{rate}^2(l^2)$ *and* E *is defined by the following rule:*

– *If* $(l^1, c^1, u^1, k^1) \in E^1$ *and* $(l^2, c^2, u^2, k^2) \in E^2$*, then* $\big((l^1, l^2), c^1 \wedge c^2, u^1 \wedge u^2, (k^1, k^2)\big) \in E$

Figure 6 shows an example of the of type reasoning possible using logical composition.

As expected, the logical composition and product coincide.

Theorem 3. *Let* H *and* H' *be RHA then:* $[\![H]\!] \wedge [\![H']\!] \approx [\![H \wedge H']\!]$.

Referring back to the light controller, clearly, no positive battery requirements can ever be fulfilled by any implementation of Fig. 2, because only saturation is admitted. We are missing a power supply, something that charges our battery. Consider the specification of a power supply depicted in Fig. 5b. The question is whether the joined requirements of capacity and timed behaviour put in parallel with the power supply is captured by our implementation put in parallel with the power supply. As such the flow of resource b becomes the sum of the saturation provided by the controller and the charge induced by the power supply. This interaction is exactly captured by structural composition.

The structural composition defines the time synchronized and resource additive product behaviour of its constituents. As such given two implementations, each satisfying a distinct constituent, their parallel execution results in an implementation of their composition.

Definition 14 (Structural product). *Let* $S^j = \langle \mathrm{X}^j, \chi_L^j, \rightarrow_j \rangle$ *for* $j \in \{1, 2\}$ *be a specification, where* $\mathrm{X}_G^1 = \mathrm{X}_G^2$*. We define the structural product of* S^1 *and* S^2*, denoted* $S^1 \parallel S^2$ *as a new specification:* $S^1 \parallel S^2 = \langle \mathrm{X}, \chi_L, \rightarrow \rangle$*, where* $\mathrm{X}_L = \mathrm{X}_L^1 \times \mathrm{X}_L^2$*,* $\mathrm{X}_G = \mathrm{X}_G^1$*,* $\chi_L = (\chi_L^1, \chi_L^2)$ *and* $\rightarrow$ *is the smallest relation satisfying:*

$$\frac{(x_L^1, x_G) \xrightarrow{\gamma}_1 (y_L^1, y_G^1) \qquad (x_L^2, x_G) \xrightarrow{\gamma}_2 (y_L^2, y_G^2)}{\big((x_L^1, x_L^2), x_G\big) \xrightarrow{\gamma} \big((y_L^1, y_L^2), y_G^1 + y_G^2 - x_G\big)} \gamma \in \{\tau\} \cup \mathbb{R}_{\geq 0}$$

$$\frac{(x_L^1, x_G) \xrightarrow{\tau}_1 (y_L^1, y_G) \qquad (x_L^2, x_G) \not\xrightarrow{\tau}_2}{\big((x_L^1, x_L^2), x_G\big) \xrightarrow{\tau} \big((y_L^1, x_L^2), y_G\big)}$$

$$\frac{(x_L^2, x_G) \xrightarrow{\tau}_2 (y_L^2, y_G) \qquad (x_L^1, x_G) \not\xrightarrow{\tau}_1}{\big((x_L^1, x_L^2), x_G\big) \xrightarrow{\tau} \big((x_L^1, y_L^2), y_G\big)}$$

As noted earlier, refinement indeed defines a precongruence over specifications in terms of the structural product.

Theorem 4. *If* S, S'' *and* T *are specifications such that* $S \sqsubseteq S'$*, then* $S \parallel T$ *exists iff* $S' \parallel T$ *exists, and given existence of these we find* $S \parallel T \sqsubseteq S' \parallel T$.

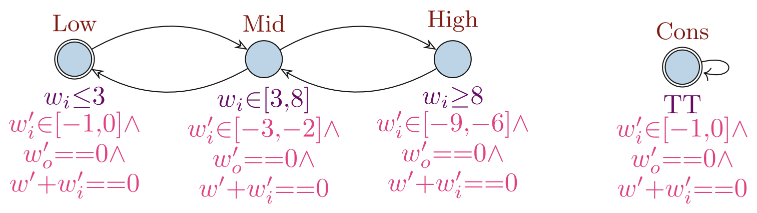


Fig. 7. (a) An RHA modelling modelling an intake valve. It is similar to the one depicted in Fig. 1a, however, it does not allow internal discrete behaviour in its operational modes, i.e. whenever a transition is made it must change mode. Additionally, it defines slightly different flow rates in Low, Mid and High. (b) Another RHA modelling an intake valve. This one is simple as all it does regardless of capacity is to continuously consume between -1 and 0 water per time unit while allowing any discrete behaviour. The structural composition of these two, however, do in fact refine the intake valve specification depicted in Fig. 1a. Note how resource additivity makes it possible to intuitively "add up" flows in a component-wise manner.

The structural product admits a transition if an only if either of its constituents allow a discrete transition or if both admit the same delay. Resource additivity is captured by our treatment of the global state-space in the target state, i.e. changes are added up. On the syntactic level we can compute the structural composition by the following RHA construction.

Definition 15 (Structural Composition).
Let $H^j = \langle \mathrm{L}^j, \mathrm{l}_0^j, \mathcal{V}^j, E^j, \mathrm{inv}^j, \mathrm{rate}^j \rangle$ *for* $j \in \{1,2\}$ *be an RHA, where* $\mathcal{V}_G^1 = \mathcal{V}_G^2$ *and* $\mathcal{V}_L^1 \cap \mathcal{V}_L^2 = \emptyset$. *We define the structural composition of* H^1 *and* H^2*, denoted* $H^1 \parallel H^2$ *as a new RHA:* $H^1 \parallel H^2 = \langle \mathrm{L}, \mathrm{l}_0, \mathcal{V}_L \uplus \mathcal{V}_G, E, \mathrm{inv}, \mathrm{rate} \rangle$*, where* $\mathrm{L} = \mathrm{L}^1 \times \mathrm{L}^2$, $\mathrm{l}_0 = (\mathrm{l}_0^1, \mathrm{l}_0^1)$, $\mathcal{V}_L = \mathcal{V}_L^1 \uplus \mathcal{V}_L^2$, $\mathcal{V}_G = \mathcal{V}_G^1$, $\mathrm{inv}\big((l^1, l^2)\big) = \mathrm{inv}^1(l^1) \wedge \mathrm{inv}^2(l^2)$, *rate* $\big((l^1, l^2)\big) = \mathrm{rate}^1(l^1) \oplus \mathrm{rate}^2(l^2)$*, and E is defined by the following rules:*

- *If* $(l^1, c^1, u^1, k^1) \in E^1$ *and* $(l^2, c^2, u^2, k^2) \in E^2$*, then* $\big((l^1, l^2), c^1 \wedge c^2, u^1 \oplus u^2, (k^1, k^2)\big) \in E$*;*
- *If* $(l^1, c, u, k) \in E^1$ *and* $\forall k^2 \in \mathrm{L}^2 : (l^2, c^2, u^2, k^2) \notin E^2$*, then* $\big((l^1, l^2), c, u, (k, l^2)\big) \in E$*;*
- *If* $(l^2, c, u, k) \in E^2$ *and* $\forall k^1 \in \mathrm{L}^1 : (l^1, c^1, u^1, k^1) \notin E^1$*, then* $\big((l^1, l^2), c, u, (l^1, k)\big) \in E$*.*

Figure 7 shows an example of the type of reasoning possible using structural composition.

Note that RHAs are closed under additive structural composition as defined by Definition 15, this is generally not the case for Linear Hybrid Automata (LHA) under synchronized composition [22]. As expected, the structural composition and product coincide.

Theorem 5. *Let H and H′ be RHA then:* $[\![H]\!] \parallel [\![H']\!] \approx [\![H \parallel H']\!]$*.*

Both logical and structural composition provide concise ways for capturing divided responsibilities of open components. A large specification can therefore be reasoned about in a component-wise manner using an intuitive notion of additivity over globally available resources.

The quotient composition of two specifications T, referred to as the 'target' and S, referred to as the 'existing component' results in a new specification X which for any implementation I where $S \parallel I \sqsubseteq T$ we have $I \sqsubseteq X$. In other words, the quotient defines the most permissive specification that characterizes the missing behaviour of the existing component in order to refine the target. In order to capture the quotient we make use of two new state types: $\bot$, which characterizes deadlock states and $\top$ which characterizes universal states. Deadlock states allow no behaviour, while universal states allow arbitrary behaviour.

Definition 16 (Quotient product). *Let $S^j = \langle \mathrm{X}^j, \chi_L^j, \rightarrow^j \rangle$ for $j \in \{1,2\}$ be a specification, where $\mathrm{X}_G^1 = \mathrm{X}_G^2$. We define the quotient product of S^1 and S^2, denoted $S^2 \backslash\backslash S^1$ as a new specification: $S^2 \backslash\backslash S^1 = \langle \mathrm{X}, \chi_L, \rightarrow \rangle$ where: $\mathrm{X}_L = (\mathrm{X}_L^1 \times \mathrm{X}_L^2) \uplus \{\bot, \top\}$, $\mathrm{X}_G = \mathrm{X}_G^1$, $\chi_L = (\chi_L^1, \chi_L^2)$, and $\rightarrow$ is the smallest relation satisfying:*

$$\frac{(x_L^1, x_G) \xrightarrow{\gamma}_1 (y_L^1, y_G^1) \qquad (x_L^2, x_G) \xrightarrow{\gamma}_2 (y_L^2, y_G^2)}{\big((x_L^1, x_L^2), x_G\big) \xrightarrow{\gamma} \big((y_L^1, y_L^2), y_G^2 + x_G - y_G^1\big)} \gamma \in \{\tau\} \cup \mathbb{R}_{\geq 0}$$

$$\frac{(x_L^2, x_G) \xrightarrow{\tau}_2 (y_L^2, y_G^2) \qquad (x_L^1, x_G) \not\xrightarrow{\tau}_1}{\big((x_L^1, x_L^2), x_G\big) \xrightarrow{\tau} \big((x_L^1, y_L^2), y_G^2\big)}$$

$$\frac{(x_L^1, x_G) \not\xrightarrow{\gamma}_1}{\big((x_L^1, x_L^2), x_G\big) \xrightarrow{\gamma} (\top, y_G)} \gamma \in \mathbb{R}_{\geq 0}$$

$$\frac{(x_L^1, x_G) \xrightarrow{\gamma}_1 (y_L^1, y_G^1) \qquad (x_L^2, x_G) \not\xrightarrow{\gamma}_2}{\big((x_L^1, x_L^2), x_G\big) \xrightarrow{\gamma} (\bot, y_G)} \gamma \in \{\tau\} \cup \mathbb{R}_{\geq 0}$$

$$\frac{(x_L^1, x_G) \not\xrightarrow{\tau}_1 \qquad (x_L^2, x_G) \not\xrightarrow{\tau}_2}{\big((x_L^1, x_L^2), x_G\big) \xrightarrow{\tau} \big((x_L^1, x_L^2), x_G\big)}$$

$$\frac{}{(\top, x_G) \xrightarrow{\gamma} (\top, y_G)} \gamma \in \{\tau\} \cup \mathbb{R}_{\geq 0}$$

Crucially, The dual of the structural product corresponds exactly to the quotient.

Theorem 6. *Let S and T be specifications. If $T \backslash\backslash S$ exists then for all implementations I we have $S \parallel I$ exists and $S \parallel I \sqsubseteq T$ iff $I \sqsubseteq T \backslash\backslash S$.*

On the syntactic level we can compute the quotient by the following RHA construction.

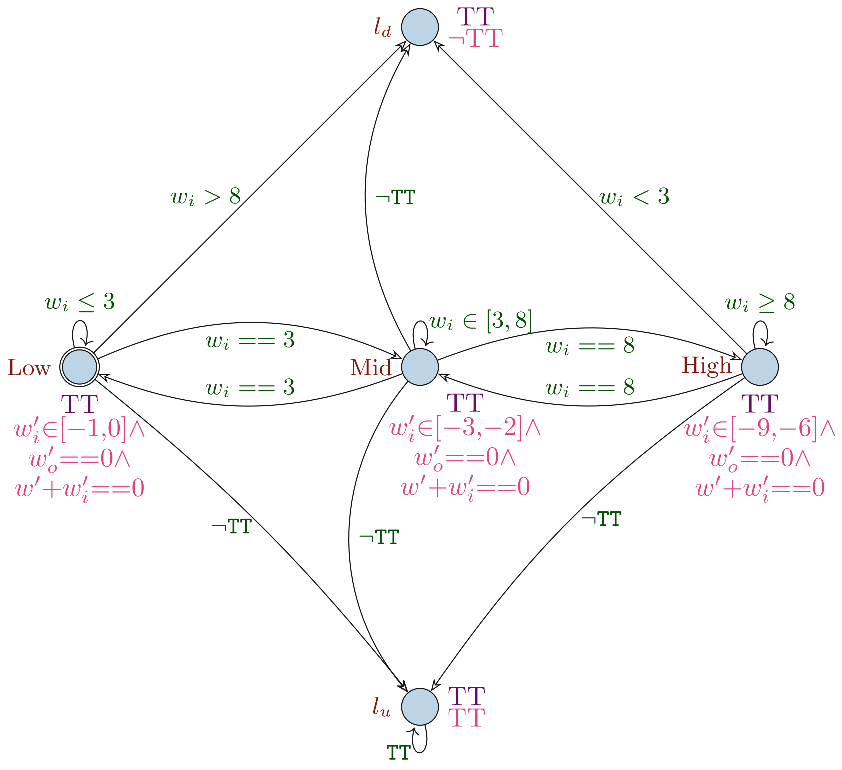


Fig. 8. The resulting RHA generated by the quotient construction by using the intake valve specification of Fig. 1a as its target and the simple consumer valve of Fig. 7b as its existing component. As expected, the intake valve specification of Fig. 7a indeed refines the quotient, the formal proof of which is omitted for the sake of brevity.

Definition 17 (Quotient composition).
Let $H^j = \langle \mathrm{L}^j, \mathrm{l}_0^j, \mathcal{V}^j, E^j, \mathrm{Inv}^j, \mathrm{rate}^j \rangle$ *for* $j \in \{1,2\}$ *be an RHA, where* $\mathcal{V}_G^1 = \mathcal{V}_G^2$*, such that* $\mathcal{V}_L^1 \cap \mathcal{V}_L^2 = \emptyset$*. We define the quotient composition of* H^1 *and* H^2 *as a new RHA* $H^2 \backslash\backslash H^1 = \langle \mathrm{L}, \mathrm{l}_0, \mathcal{V}, E^j, \mathrm{Inv}, \mathrm{rate} \rangle$ *where:* $\mathrm{L} = (\mathrm{L}^1 \times \mathrm{L}^2) \cup \{\mathrm{l}_u, \mathrm{l}_d\}$, $\mathrm{l}_0 = (\mathrm{l}_0^1, \mathrm{l}_0^2)$, $\mathcal{V}_L = \mathcal{V}_L^1 \uplus \mathcal{V}_L^2$, $\mathcal{V}_G = \mathcal{V}_G^1$, $\mathrm{Inv}\big((l^1,l^2)\big) = \mathrm{Inv}(l_u) = \mathrm{Inv}(l_d) = \mathit{TT}$, $\mathrm{rate}\big((l^1,l^2)\big) = \mathrm{rate}^2(l_2) \oslash \mathrm{rate}^1(l_1)$, $\mathrm{rate}(l_u) = \mathit{TT}$, $\mathrm{rate}(l_d) = \neg \mathit{TT}$, *and* E *is defined by the following rules:*

- *If* $(l^1,c^1,u^1,k^1) \in E^1$ *and* $(l^2,c^2,u^2,k^2) \in E^2$*, then* $\big((l^1,l^2),c,u,(k^1,k^2)\big) \in E$ *where* $u = u^2 \oslash u^1$ *and* $c = c^1 \wedge \mathrm{Inv}^1(l^1) \wedge u \triangleright \mathrm{Inv}^1(k^1) \wedge c^2 \wedge \mathrm{Inv}^2(l^2) \wedge u \triangleright \mathrm{Inv}^2(k^2)$*;*
- *If* $(l^2,c^2,u^2,k^2) \in E^2$ *and* $\forall k^1 \in \mathrm{L}^1 : (l^1,c^1,u^1,k^1) \notin E^1$*, then* $\big((l^1,l^2),c,\alpha,u^2,(l^1,k^2)\big) \in E$ *where* $l^1 \in \mathrm{L}^1$ *and* $c = \mathrm{Inv}^1(l^1) \wedge c^2 \wedge \mathrm{Inv}^2(l^2) \wedge u^2 \triangleright \mathrm{Inv}^2(k^2)$*;*
- *If* $l^1 \in \mathrm{L}^1$ *and* $l^2 \in \mathrm{L}^2$*, then* $\big((l^1,l^2),c,\mathit{TT},l_u\big) \in E$ *where* $c = \big(\neg\mathrm{Inv}^1(l^1) \vee \mathrm{Inv}^2(l^2)\big) \wedge \bigwedge_{(l^1,c^1,u^1,k^1)\in E^1} (\neg c^1 \vee \neg u^1 \triangleright \mathrm{Inv}^1(k^1))$*;*

– *If* $l^1 \in \mathrm{L}^1$, $l^2 \in \mathrm{L}^2$, *then* $\big((l^1, l^2), c, \epsilon, (l^1, l^2)\big) \in E$ *where* $c = c^1 \wedge = \mathrm{Inv}^1(l^1) \wedge \bigwedge_{(l^1,c^1,u^1,k^1) \in E^1} (\neg c^1 \vee \neg u^1 \rhd \mathrm{Inv}^1(k^1)) \wedge \mathrm{Inv}^2(l^2) \wedge \bigwedge_{(l^2,c^2,u^2,k^2) \in E^2} (\neg c^2 \vee \neg u^2 \rhd \mathrm{Inv}^2(k^2))$;
– *If* $(l^1, c^1, u^1, k^1) \in E^1$ *and* $l^2 \in \mathrm{L}^2$, *then* $\big((l^1, l^2), c, TT, l_d\big) \in E$ *where* $c = c^1 \wedge \mathrm{Inv}^1(l^1) \wedge u^1 \rhd \mathrm{Inv}^1(k^1) \bigwedge_{(l^2,c^2,u^2,k^2) \in E^2} (\neg c^2 \vee \neg u^2 \rhd \mathrm{Inv}^2(k^2))$;
– $(l^u, TT, TT, l^u) \in E$.

An example quotient can be seen in Fig. 8.

The quotient product refines the underlying semantics defined by the quotient composition, and whenever the product is consistent then so is the composition.

Theorem 7. *Let H and H' be RHA then $[\![H]\!] \backslash\backslash [\![H']\!] \sqsubseteq [\![H \backslash\backslash H']\!]$.*

Theorem 8. *Let H and H' be RHA then $[\![H \backslash\backslash H']\!]$ is consistent iff $[\![H]\!] \backslash\backslash [\![H']\!]$ is consistent.*

Unfortunately, the syntactic construction and the semantic product do not fully coincide. In fact the syntactic construction is an abstraction of the semantic product. This is because the product insists that a state after some delay can act as a deadlock- or universal- state, which cannot be mimicked in the syntactic construction without introducing complex rate rules and appropriate mechanisms for handling universal and deadlock behaviour directly in the semantics of RHAs. For the sake of brevity and because it has no impact on practical applications (however still vital to a full characterization of the theory), this aspect is left as a topic for future research. Note also, that all three compositions always exists. This is because RHAs are defined over essentially internal discrete actions, as such, the signature of all RHA is the same. Indeed any RHA is defined over the set of all global resources. The notion of environment and component is solely dictated by the model. Intuitively, one can think of environments as components that provides a positive resource flow and vice versa for components. A more powerful extension of the theory with discrete inputs and outputs would complicate this aspect however. In such an extension, the notion of compatibility in terms of signature becomes relevant. We leave this aspect of compatibility as a topic for future research in the full discrete I/O characterization of the theory.

4 Assume-Guarantee Reasoning

With our specification theory of RHA, we have a robust and complete framework, suitable for component-wise design and refinement in the additive hybrid setting. We now show that our theory facilitates component-wise assume-guarantee reasoning. The main idea is to use the notion of pre- and post-conditions in order to characterize intended behaviour. Usually pre- and post-conditions define properties of sequential processes that must hold before respectively after some behaviour is encountered. Since we are dealing with systems that consists of real-time concurrent components the notion of 'before' and 'after' is better captured by structural compositional reasoning. As such a pre-condition defines an environment that affects our system, and a post-condition defines how a system should act whenever such an environment is within our sphere of influence. Mathematically, we are simply dealing with implications, i.e. for the pair (P, Q) consisting of a pre-condition and post-condition (as mentioned earlier, this is called a contract), a system upholds the pair if and only if whenever P holds then so does Q. We adopt the usual terminology used in the real-time setting, that is, pre- and post- conditions are referred to as assumptions and guarantees.

Let's first formally capture our notion of an assume-guarantee pair and satisfaction thereof. In our theory, assumptions and guarantees are defined by RHAs. We need to characterize an RHA that exactly describes the assume-guarantee implication. In the concurrent real-time setting this is known as a weakening and captured by a so-called weaken operation as defined in [12].

Definition 18 (Weaken). *Let A and G be RHAs. We define the weakening of G in A, as: $G \gg A \sqsubseteq (A \parallel G) \backslash\backslash A$*

Let's design a water pump based on a specification defined by a weakening. Our water pump's environment consists of two reservoirs; an input reservoir, for which the pump itself can only draw from, and an output reservoir, for which the pump can only provide to. We characterize the capacity of the input and output reservoirs by the global resources w_i and w_o. Our assumption on the environment and our guarantee on the water pump whenever that assumption holds is shown in Fig. 9.

We define our proposed water pump system S as the structural composition of the three RHAs shown in Fig. 1. We hypothesize that S refines $G \gg A$ thereby making it possible to use an implementation of S whenever we need an implementation of $G \gg A$. As such we retain the simple and intuitive model defined by the weakening when considering the water pump in a larger context while providing certainty that an actual system can be implemented using the more specialised but less intuitive model. We should note here that the reserve question is just as useful from a system design perspective. Looking at Fig. 1, clearly the composition results in a large RHA, in fact even the component-wise representation is large and cumbersome. If one finds that $G \gg A$ refines S then, from a design perspective, we can reason about the water pump specification simply by using the weakening instead.

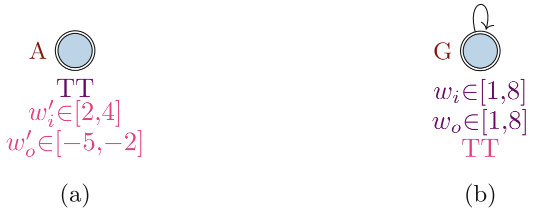


Fig. 9. (a) An RHA modelling our assumption on the environment. We assume that the input reservoir gets filled by a rate in the interval $[2, 4]$ and that the output reservoir gets saturated by rate in the interval $[-5, -2]$. Additionally, we assume that the environment never saturates the usage reservoir, thereby completely delegating that responsibility to the system from a modelling perspective. (b) Another RHA modelling our guarantee on the system if the assumption is fulfilled. We guarantee that the capacity of both the input and output reservoir stays in the interval $[1, 8]$. Additionally, we guarantee that at any time, a discrete event can occur, as long as it performs no resets.

Now we have a system, a weakening and well defined set of operations on our language, all we need now is to ascertain whether the refinement holds.

5 Refinement

Unfortunately, checking refinement for general RHAs is undecidable. Indeed if it were, then that would imply that reachability for general LHA is decidable, which it is not [22]. Instead we explore how statistical model checking can be used to refute the existence of refinement. To that end, we utilize the verification engine of UPPAAL SMC [13] to conduct simulation based validation by translating the RHA models into Stochastic Hybrid Automata (SHA) [13]. Essentially, these are hybrid automata defined over a stochastic timed semantics, refining the non-deterministic mechanics of edge transitions and time delays into probabilistic occurrences based on some probability distribution. Much like RHAs, SHAs allows us to define linear differential equations on variable rates and also to consider such variables in guards and invariants. Hence can use resources of

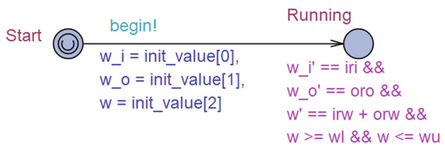


Fig. 10. The SHA agent responsible for computing the continuous rates of all resources in the composition. Here resources w_i, w_o and w are first initialized to their respective initial values; $\{1, 1, 10\}$ signified by the firing of action begin after which the mode Running in each time step sets the appropriate rate of each resource according to the real-valued variables iri, oro, irw and orw. Annotation $\smile$ signifies that time cannot pass in a mode.

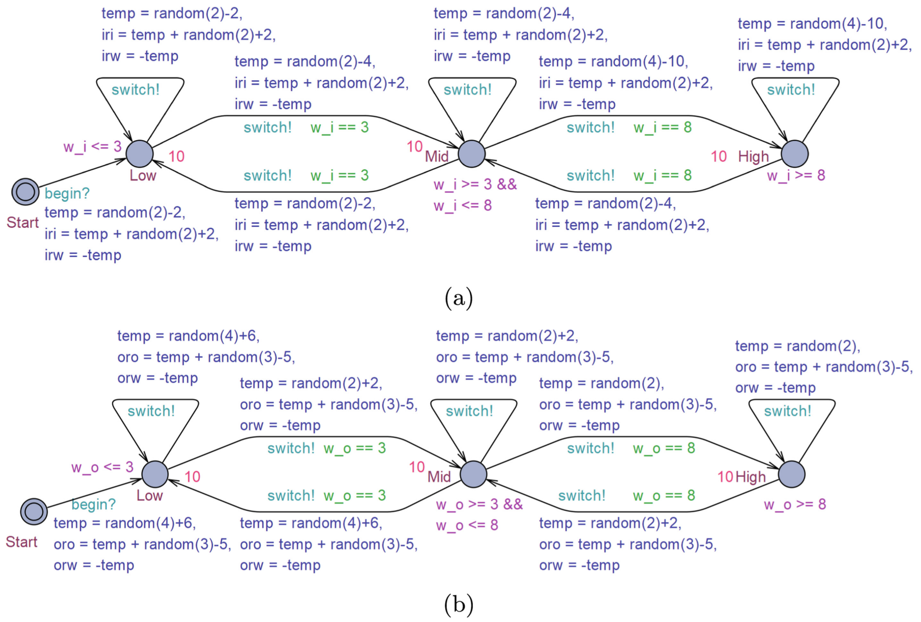


Fig. 11. The SHA interpretations of the intake valve (a) and the output valve (b) specifications of Fig. 1a and 1c. The discrete action switch is used to communicate that a transition has been fired. This aspect is crucial because we need to know when a possible refinement target must be able to do a transition. The red scalars next to each mode is a probability parameter and just an implementation detail, suffice it to say that larger numbers results in higher preference for taking a transition whenever one is enabled.

RHAs as is in SHAs. We can specify a convex interval of real numbers by using the function $random(max) = [0, max)$. SHAs are defined over a discrete action set of inputs (characterized by a question mark ?) and outputs (characterized by an exclamation point !). Sadly, SHAs are not natively additive, so this needs to be simulated. To do so is a simple matter of introducing a real-valued variable for each automata specific occurrence of a resource and define an additivity agent responsible for adding up the different rates into the actual rates. Our implementation of the agent model is shown in Fig. 10.

All that remains is to translate the capacity monitor, and the intake/output valve of Fig. 1 into SHAs. For convenience we have dedicated the responsibility of the capacity monitor to the additivity agent. Additionally, the rates defined by the assumption shown in Fig. 9 have been put directly on the intake and output valve. Their SHA interpretations are shown in Fig. 11

With that we have a stochastic realization of our water pump controller. To answer the question of whether the water pump refines the weakening $G \gg A$, we investigate whether the water pump controller in parallel with the assumption refines the composition of the assumption and the guarantee. One could also

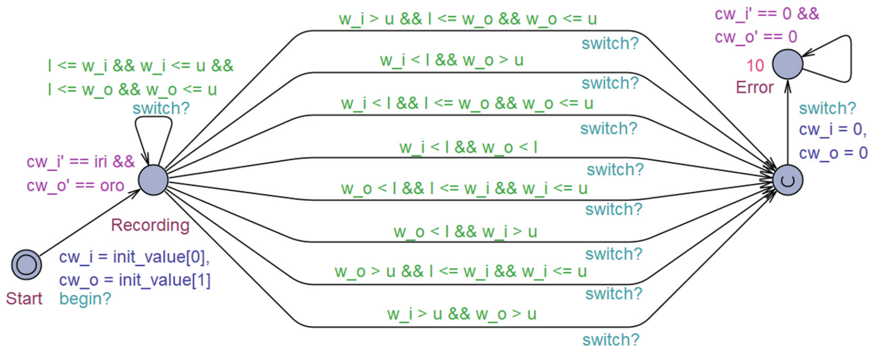


Fig. 12. The unfolded SHA interpretation of $A \parallel G$. Resources cw_i and cw_o are defined as clones of resources w_i and w_o. The mode Recording directly models the behaviour of $A \parallel G$. Whenever a transition is made, it is checked whether the invariants of $A \parallel G$ holds. Otherwise the invariant of Recording is unbounded, capturing values of w_i and w_o not admitted by the original composition. This is because we aim to explicitly capture illegal behaviour in order to direct it into the Error mode. The eight transitions leading to it simply captures the negation of the invariant defined by $A \parallel G$. Note that we need one for each case because stochastic semantics requires input-determinism.

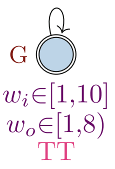


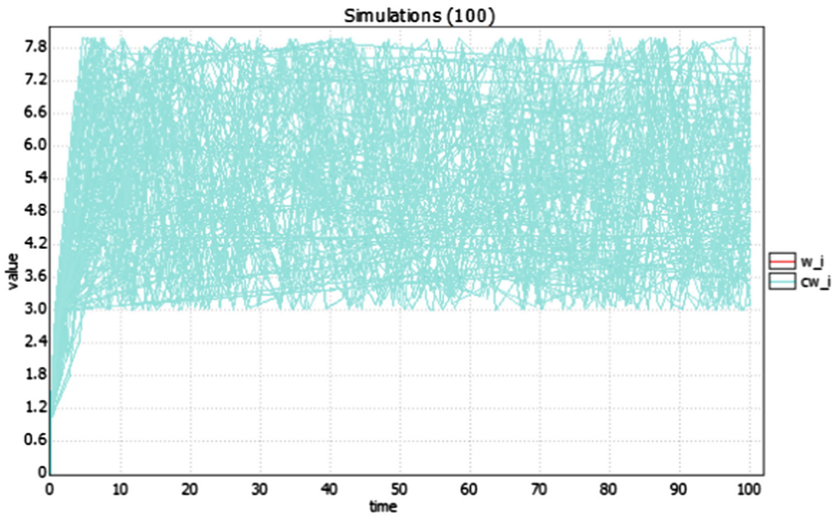
Fig. 13. An RHA modelling a slightly more strict guarantee.

just show the refinement to the quotient defined by $G \gg A$ directly, however the composition of A and G is arguable more intuitive.

The last step in our translation effort is to obtain a stochastic interpretation of our refinement target, namely the structural composition of A and G. This is shown in Fig. 12.

Using the SHA interpretation of the composition of our water pump controller and the assumption, we intend to drive the SHA model of $A \parallel G$. Our target property is supported by the following reasoning. If the error mode is ever entered and the clones of w_i and w_o get assigned the rate 0, then we know with absolute certainty that the original water pump controller cannot refine the weakening $G \gg A$. Since both SHAs are abstractions, this holds. We validate our property using the simulation capabilities of UPPAAL SMC.

We are now ready to conduct testing on our setup through simulations. Simulations shown in this paper all depict a number of sample runs as solid coloured lines. The metrics of interest are the evolution of the actual resources w_i and w_o together with the evolution of their clones cw_i and cw_o (y-axis) over time (x-axis).



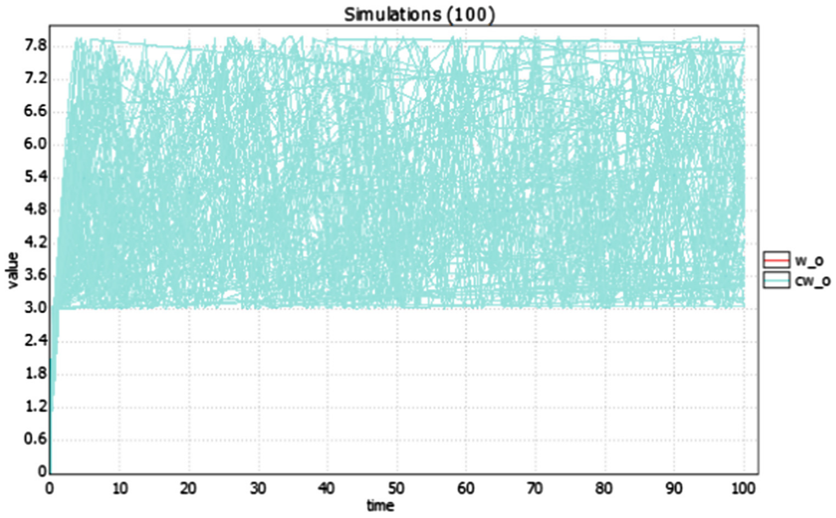


Fig. 14. Two sample simulation results obtained by queries $simulate[\Leftarrow 100]\{w_i, cw_i\}$ and $simulate[\Leftarrow 100]\{w_o, cw_o\}$ capturing the values of resource w_i and its clone respectively w_o and its clone. As may be expected, even with 100 simulations, the water pump controller seems to not be able to break the behaviour of $A \parallel G$. This is why we see only one variable in all traces, the clone perfectly matches the actual resource.

Let's start by simulating our water pump controller driving $A \parallel G$. The results of which are shown in Fig. 14. As can be seen, the controller seems to have difficulties breaking the invariant of $A \parallel G$. Of course we cannot conclude whether it is impossible, there might exist some execution that renders the invariant false, it just so happens that we have not found it.

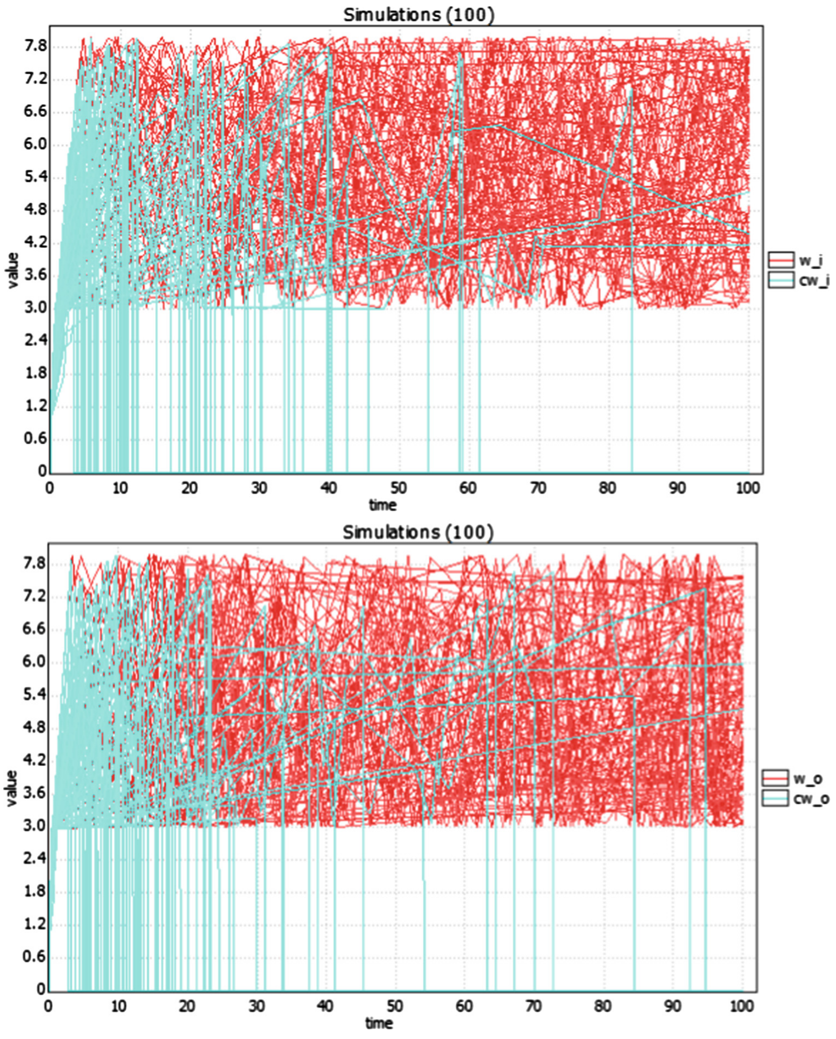


Fig. 15. Two sample simulation results obtained by queries $simulate[\Leftarrow 100]\{w_i, cw_i\}$ and $simulate[\Leftarrow 100]\{w_o, cw_o\}$ capturing the values of resource w_i and its clone respectively w_o and its clone. These results show that the water pump controller can force $A \parallel G$ into the Error mode, hence refuting the existence of a refinement into $G \gg A$.

Next we try to modify G slightly, maybe we can identify an interesting frontier for the upper limit of w_i and w_o. For that purpose, we use the slightly modified guarantee shown in Fig. 13. The simulation results are shown in Fig. 15. Now we clearly see that a capacity strictly below 8 cannot be guaranteed by our water pump controller under assumption A. Using the same reasoning as above, we can say with certainty that the composite of Fig. 1 under assumption A does not refine $G \gg A$.

6 Concluding Remarks

In this paper, we have proposed the complete specification theory of RHA, suitable for reasoning about step-wise refinement in the domain of additive resource-aware concurrent systems. As far as we know, this is the first such theory considering additive composition. We have shown how assume-guarantee reasoning is possible within the theory and exemplified a relevant sample case thereof. Furthermore, by translating RHAs into SHAs we have shown how one can refute the existence of refinement using a simulation based validation method.

In terms of further validation, a proper case study still remains to be conducted. Energy-aware systems, such as load-balancing and smart-grid analysis are prime candidates. Additionally, an intuitive and robust tool implementation through automated translation into SHA would significantly decrease the entry-level knowledge required to use the method presented. For that purpose, the discrete input/output extension of RHA would be required, including the complete characterization of the quotient construction in this setting. This full characterization of the theory could serve as a general meta theory in the additive hybrid domain, which would significantly push state-of-the-art in the real-time analysis setting if a useful and decidable instance of the theory is identified. Furthermore, showing how the weaken operation can be used to handle scalability issues is also desirable, i.e. lifting this result on Timed Input/Output Automata as shown in e.g. [12] to the additive hybrid setting.

References

1. de Alfaro, L., Henzinger, T.A.: Interface automata. In: Tjoa, A.M., Gruhn, V. (eds.) Proceedings of the 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering 2001, Vienna, Austria, 10–14 September 2001, pp. 109–120. ACM (2001). https://doi.org/10.1145/503209.503226
2. Alur, R., Dill, D.L.: A theory of timed automata. Theoret. Comput. Sci. **126**(2), 183–235 (1994). https://doi.org/10.1016/0304-3975(94)90010-8
3. Bacci, G., Bouyer, P., Fahrenberg, U., Larsen, K.G., Markey, N., Reynier, P.-A.: Optimal and robust controller synthesis. In: Havelund, K., Peleska, J., Roscoe, B., de Vink, E. (eds.) FM 2018. LNCS, vol. 10951, pp. 203–221. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-95582-7_12
4. Bauer, S.S., et al.: Moving from specifications to contracts in component-based design. In: de Lara, J., Zisman, A. (eds.) FASE 2012. LNCS, vol. 7212, pp. 43–58. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28872-2_3
5. Benveniste, A., et al.: Contracts for system design. Found. Trends Electron. Des. Autom. **12**(2–3), 124–400 (2018). https://doi.org/10.1561/1000000053
6. Bergstra, J.A., Klop, J.W.: Algebra of communicating processes with abstraction. Theor. Comput. Sci. **37**, 77–121 (1985). https://doi.org/10.1016/0304-3975(85)90088-X
7. Boudol, G., Larsen, K.G.: Graphical versus logical specifications. In: Arnold, A. (ed.) CAAP 1990. LNCS, vol. 431, pp. 57–71. Springer, Heidelberg (1990). https://doi.org/10.1007/3-540-52590-4_40

8. Bouyer, P., Colange, M., Markey, N.: Symbolic optimal reachability in weighted timed automata. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9779, pp. 513–530. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41528-4_28
9. Caillaud, B., Delahaye, B., Larsen, K.G., Legay, A., Pedersen, M.L., Wasowski, A.: Constraint Markov chains. Theor. Comput. Sci. **412**(34), 4373–4404 (2011). https://doi.org/10.1016/j.tcs.2011.05.010
10. Čerāns, K., Godskesen, J.C., Larsen, K.G.: Timed modal specification—theory and tools. In: Courcoubetis, C. (ed.) CAV 1993. LNCS, vol. 697, pp. 253–267. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-56922-7_21
11. Cuijpers, P.J.L., Reniers, M.A.: Lost in translation: hybrid-time flows vs. real-time transitions. In: Egerstedt, M., Mishra, B. (eds.) HSCC 2008. LNCS, vol. 4981, pp. 116–129. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78929-1_9
12. David, A., et al.: Compositional verification of real-time systems using Ecdar. Int. J. Softw. Tools Technol. Transf. **14**(6), 703–720 (2012). https://doi.org/10.1007/s10009-012-0237-y
13. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B.: UPPAAL SMC tutorial. Int. J. Softw. Tools Technol. Transf. **17**(4), 397–415 (2015). https://doi.org/10.1007/s10009-014-0361-y
14. David, A., Larsen, K.G., Legay, A., Nyman, U., Wasowski, A.: Timed I/O automata: a complete specification theory for real-time systems. In: Johansson, K.H., Yi, W. (eds.) Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2010, Stockholm, Sweden, 12–15 April 2010, pp. 91–100. ACM (2010). https://doi.org/10.1145/1755952.1755967
15. van Glabbeek, R.J.: The linear time - branching time spectrum I: the semantics of concrete, sequential processes. In: Bergstra, J.A., Ponse, A., Smolka, S.A. (eds.) Handbook of Process Algebra, chap. 1, pp. 3–99. Elsevier Science, Amsterdam (2001). https://doi.org/10.1016/B978-044482830-9/50019-9
16. Hansen, J., Larsen, K.G., Cuijpers, P.J.L.: Balancing flexible production and consumption of energy using resource timed automata. In: 2022 11th Mediterranean Conference on Embedded Computing (MECO), pp. 1–6 (2022). https://doi.org/10.1109/MECO55406.2022.9797191
17. He, J.: Process simulation and refinement. Formal Aspects Comput. **1**(3), 229–241 (1989). https://doi.org/10.1007/BF01887207
18. Jifeng, H.: Various simulations and refinements. In: de Bakker, J.W., de Roever, W.-P., Rozenberg, G. (eds.) REX 1989. LNCS, vol. 430, pp. 340–360. Springer, Heidelberg (1990). https://doi.org/10.1007/3-540-52559-9_70
19. He, J.: Service refinement. In: 15th Asia-Pacific Software Engineering Conference (APSEC 2008), 3–5 December 2008, Beijing, China, p. 5. IEEE Computer Society (2008). https://doi.org/10.1109/APSEC.2008.78
20. He, J., Hoare, C.A.R.: Unifying theories of programming. In: Orlowska, E., Szalas, A. (eds.) Participants Copies for Relational Methods in Logic, Algebra and Computer Science, 4th International Seminar RelMiCS, Warsaw, Poland, 14–20 September 1998, pp. 97–99 (1998)
21. He, J., Liu, Z., Li, X.: Towards a refinement calculus for object systems. In: Proceedings of the 1st IEEE International Conference on Cognitive Informatics (ICCI 2002), 19–20 August 2002, Calgary, Canada, pp. 69–76. IEEE Computer Society (2002). https://doi.org/10.1109/COGINF.2002.1039284

22. Henzinger, T.A., Kurshan, R.P.: The theory of hybrid automata. In: Inan, M.K., Kurshan, R.P. (eds.) Verification of Digital and Hybrid Systems. NATO ASI Series, vol. 170, pp. 265–292. Springer, Heidelberg (2000)
23. Hoare, C.A.R.: Communicating Sequential Processes. International Series in Computer Science. Prentice Hall (1985)
24. Hoare, C.A.R., He, J.: The weakest prespecification. Inf. Process. Lett. **24**(2), 127–132 (1987). https://doi.org/10.1016/0020-0190(87)90106-2
25. Hoare, T., He, J.: Unifying theories for parallel programming. In: Lengauer, C., Griebl, M., Gorlatch, S. (eds.) Euro-Par 1997. LNCS, vol. 1300, pp. 15–30. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0002714
26. Hoare, C.A.R., He, J., Sanders, J.W.: Prespecification in data refinement. Inf. Process. Lett. **25**(2), 71–76 (1987). https://doi.org/10.1016/0020-0190(87)90224-9
27. Jones, C.B.: Developing methods for computer programs including a notion of interference. Ph.D. thesis, University of Oxford, UK (1981). https://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.259064
28. Lamport, L.: Hybrid systems in TLA$^+$. In: Grossman, R.L., Nerode, A., Ravn, A.P., Rischel, H. (eds.) HS 1991-1992. LNCS, vol. 736, pp. 77–102. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-57318-6_25
29. Larsen, K., et al.: As cheap as possible: efficient cost-optimal reachability for priced timed automata. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 493–505. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44585-4_47
30. Larsen, K.G.: Modal specifications. In: Sifakis, J. (ed.) CAV 1989. LNCS, vol. 407, pp. 232–246. Springer, Heidelberg (1990). https://doi.org/10.1007/3-540-52148-8_19
31. Larsen, K.G., Rasmussen, J.I.: Optimal reachability for multi-priced timed automata. Theor. Comput. Sci. **390**(2), 197–213 (2008). https://doi.org/10.1016/j.tcs.2007.09.021. Foundations Software Science and Computational Structures
32. Larsen, K.G., Steffen, B., Weise, C.: The methodology of modal constraints. In: Broy, M., Merz, S., Spies, K. (eds.) Formal Systems Specification. LNCS, vol. 1169, pp. 405–435. Springer, Heidelberg (1996). https://doi.org/10.1007/BFb0024437
33. Larsen, K.G., Thomsen, B.: A modal process logic. In: Proceedings of the Third Annual Symposium on Logic in Computer Science (LICS 1988), Edinburgh, Scotland, UK, 5–8 July 1988, pp. 203–210. IEEE Computer Society (1988). https://doi.org/10.1109/LICS.1988.5119
34. Milner, R.: A Calculus of Communicating Systems. Lecture Notes in Computer Science, vol. 92. Springer, Cham (1980). https://doi.org/10.1007/3-540-10235-3
35. Owicki, S.S., Gries, D.: An axiomatic proof technique for parallel programs I. Acta Inform. **6**, 319–340 (1976). https://doi.org/10.1007/BF00268134
36. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science, pp. 46–57 (1977). https://doi.org/10.1109/SFCS.1977.32
37. Scholefield, D., Zedan, H., Jifeng, H.: Real-time refinement: semantics and application. In: Borzyszkowski, A.M., Sokołowski, S. (eds.) MFCS 1993. LNCS, vol. 711, pp. 693–702. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-57182-5_60
38. Xu, Q., de Roever, W.P., He, J.: The rely-guarantee method for verifying shared variable concurrent programs. Formal Aspects Comput. **9**(2), 149–174 (1997). https://doi.org/10.1007/BF01211617