Abstract

Hybrid systems combine discrete and continuous dynamics. We introduce a semantics for such systems consisting of a coalgebra together with a monoid action. The coalgebra captures the (discrete) operations on a state space that can be used by a client (like in the semantics of ordinary (non-temporal) object-oriented systems). The monoid action captures the influence of time on the state space, where the monoids that we consider are the natural numbers monoid \((\mathbb{N}, 0, +)\) of discrete time, and the positive reals monoid \((\mathbb{R}_+, 0, +)\) of real time. Based on this semantics we develop a hybrid specification formalism with timed method applications: it involves expressions like \(s.meth@z\), with the following meaning: in state \(s\) let the state evolve for \(z\) units of time (according to the monoid action), and then apply the (coalgebraic) method \(meth\). In this formalism we specify various (elementary) hybrid systems, investigate their correctness, and display their behaviour in simulations. We further define a suitable notion of homomorphism between our hybrid models (of coalgebras plus monoid actions), in such a way that minimal realizations (of the specified behaviour) appear as terminal models. We identify the terminal models of our example specifications, and give general constructions. This leads to an investigation of various topics related to terminality: bisimilarity, invariance, refinement and behaviour-realization adjunctions. In a final section we briefly discuss non-homogeneous hybrid systems (with continuous inputs).

1. Introduction

Hybrid systems combine discrete and continuous dynamics. Roughly, they involve a combination of automata theory and differential equations. Hybrid systems are essential in embedded software, monitoring and controlling the outside world (which is
assumed to be continuous). For example, a chemical process may be described via the (continuous) solutions of differential equations, and it may be (discretely) influenced by the (instantaneous) addition of chemical ingredients (e.g. via opening and closing of valves).

In this paper we develop semantics and specification for (deterministic) hybrid systems, building on earlier work on semantics and specification for object-oriented systems (see [35, 21, 23, 15, 25]). Object orientation means that the domain of application is represented as a collection of autonomous entities, called objects, each dealing with a specific task. Coordination and communication takes place via sending of messages. Objects are grouped into classes, and these classes exist in a (sub)class hierarchy which reflects a particular organization of the application domain. Objects have private data, which are accessible only via specified operations, called methods. These methods are defined in a class and are the same for all objects belonging to the class (but the data may be different in different objects). Once a class has been defined, objects can be created as instances of this class. For example, one may have a class of vehicles, with subclasses of cars and of lorries. These subclasses each have additional data and operations which are typical for cars and for lorries. Objects belonging to these subclasses (i.e. particular cars and lorries) can then be used in a specific application.

In this paper we are interested in extending the object-oriented approach from ordinary object-oriented, non-continuous systems to hybrid systems. The object-oriented approach may help one to deal with the enormous complexities that arise in real-world situations, via a suitably modular approach and via inheritance (both for conceptual classification, and for re-use).

The earlier work on object-oriented systems that we extend here is “coalgebraic” in nature, because the underlying models are based on “coalgebras”. These coalgebras are the formal duals of algebras,¹ in which one only has “destructors” (or “observers”) as operations, instead of “constructors” in algebras [24]. Coalgebras may be seen as abstract machines, consisting of a state space together with certain operations acting on this space. But typically, there are no means for (algebraically) constructing elements of the state space. This formalizes the view that a state space is a black box with unknown contents. A somewhat different approach to specification and semantics of object-oriented systems is based on hidden-sorted algebras, see e.g. [12–14, 7, 6]. In this approach, the state space of a class is represented by several “hidden” sorts which are only indirectly accessible via the “visible” sorts. The (product of the) hidden sorts corresponds to our (coalgebraic) state space $X$, see [28] for a comparison.

Here we extend the coalgebraic semantics and specification with temporal aspects.

¹ Category theory provides the proper level of abstraction at which the difference between algebras and coalgebras can be fully appreciated: an algebra is a map of the form $T(X) \to X$ whereas a coalgebra is a map of the (reversed) form $X \to T(X)$, where $T : C \to C$ is a functor on some category $C$. An algebra $T(X) \to X$ allows us to construct elements of $X$ (since it is a map pointing into $X$), and a coalgebra $X \to T(X)$ allows us to decompose elements of $X$. 
At the semantic level we continue to use coalgebras to accommodate for methods (object-oriented terminology for operations) which can be used by a client. What we add is a feature which allows us to deal with time. This feature is a “monoid action” \(\mu : U \times M \rightarrow U\), where \(U\) is a state space and \(M\) is a monoid (usually the positive integers \(\mathbb{N} = \{0, 1, 2, \ldots\}\) for discrete time, or the positive reals \(\mathbb{R}_{\geq 0} = \{x \in \mathbb{R} \mid x \geq 0\}\) for real time). Such a monoid action \(\mu\) describes for a state \(s \in U\) and a time duration \(\alpha \in M\) a new state \(\mu(s, \alpha) \in U\), obtained by letting \(s\) evolve for \(\alpha\) units of time. Typically, the state of a timer evolves under the influence of time. A monoid action \(\mu\) should satisfy certain linearity conditions, see equations (6) in Section 4.1. Monoid actions typically arise from state transition functions, or from differential equations (as solutions). They are fundamental in system theory, and are known under various alternative names: flows, motions, solutions or trajectories. These (coalgebra plus monoid action)-models will be presented as (mathematical) realizations of hybrid specifications.

At the syntactic level we introduce a notation which allows us to indicate that a method in a class will be applied after a certain time delay: we introduce expressions \(s\text{.meth}@\alpha\), where \(\alpha\) is a time parameter, with the following meaning: in state \(s\) let the state evolve for \(\alpha\) units of time (according to the monoid action), and then apply the (coalgebraic) method \(\text{meth}\). Such expressions will be used in assertions in specifications of (hybrid) classes. These assertions impose behavioural constraints on the elements of the state space. But these assertions also allow us to reason both about time and about (observations of) states. We thus use assertional methods (in contrast to process algebraic methods) to describe and reason about hybrid systems.

A subtle point is what definition should be taken for “homomorphism of hybrid models”. The obvious idea of pasting together a homomorphism of monoid actions and a homomorphism of coalgebras does not work, in the sense that it does not yield the terminal characterization of the intended minimal models. Therefore we introduce a different notion of homomorphism which allows us to suitably hide the internal time-steps (given by the monoid action), see Definition 6 below. It tells us what a “terminal model” is, because a terminal model is characterized by the property — dual to the property that determines initial models — that from an arbitrary model there is a unique homomorphism to the terminal model. We will show in various examples that terminal models are “optimal” models in the sense that they have the minimal set of states. They form minimal realizations, in system-theoretic terminology, and the terminal model is usually the intended model of a specification. Terminal models are special because they identify all observationally indistinguishable (bisimilar) states (see e.g. [37]). We find that forcing oneself to identify the terminal model of a hybrid specification is a useful way to get one’s specification right: in writing out the details of the terminal model — and in proving that it really is terminal — it will become clear whether the specification is incomplete and extra assertions should be added.

In our (current) specification format we do not use differential equations, like e.g. in [31, 2], but solutions to differential equations. This may be seen as restrictive, e.g. in situations where there is no solution at hand. However, the systems which are
currently verified in the (computer science) literature seldom have complicated, non-linear differential equations.\(^2\)

Our specification format is deterministic: we can only produce a single new state (from a given state) via a procedure and via a monoid action. But a client has a choice which procedure to apply next. Often in automata theory, these external choices are handled via non-determinism. But coalgebraically, these choices are handled deterministically: there are multiple procedures, acting as separate operations on a state space. Each of these operations changes the state space in its own way. The client’s choice which of these procedures to apply, is handled externally, so that the specific procedure that is selected can operate deterministically. (It is another matter whether or not it is useful to have these individual procedures operating non-deterministically. This can also be described in a coalgebraic setting, but it is outside the scope of the present paper.)

A hybrid coalgebraic specification typically contains a great number of assertions, many of which are trivial. This is because every possible (observable) consequence of a procedure changing a state has to be laid down. Often, for example, a particular procedure only affects one attribute, so that we have to say explicitly for each other attribute that its value after the procedure application is the same as before. Our class specifications are thus on a low level, involving many details. This forces us to keep class specifications of reasonably small size (involving only a few methods), and thus to work in a very modular way. But limiting the size of classes is usually recommended in object-oriented design, and a higher level specification formalism may then be used for handling more general requirements.

This paper introduces a specific approach to hybrid systems, extending the coalgebraic approach to object-oriented systems. It shows that many notions for object-oriented systems (like minimal realization, bisimilarity, refinement, inheritance) extend to hybrid systems, via similar definitions and results. It thus establishes a conceptual unity, at a mathematical level. As such it follows the tradition of [4, 26, 34, 11].

Two points are worth mentioning in this context.

- A language Coalgebraic Class Specification Language (CCSL) is introduced in [15], together with a tool which serves as a front-end to a proof tool. It translates (object-oriented) class specifications in CCSL into logical theories, as input for a proof tool. Reasoning about the CCSL class specification proceeds via this translation, using the support of the proof tool. In principle, this approach can also be used for the hybrid specifications as described in this paper. They would require a superclass, containing a monoid action \(\mu\), from which all hybrid classes would inherit. The CCSL language and its tool are still under development, and serious studies using hybrid specifications have not been carried out yet. But in principle, the language and the tool from [15] are also useful for hybrid specifications.

\(^2\)In principle, one can replace the explicit solutions as used in the current specification format by differential equations. The resulting notion of model would then probably involve an action that forms a solution to the equation. This alternative specification style is still to be investigated.
The monoid actions $\mu$ as studied below can also be used in actual programming. Objects of a class are then regularly updated (via a special “advance” method $\mu$), with a time duration – which depends on how much time has elapsed since the last update. For “continuous” systems, the update computation may, for example, involve approximation of a differential equation. Also, external and internal events can be handled within this approach. This leads to a systematic organization and handling of the dynamics of objects. It has been tested in a few applications, including games and simulations (e.g. of two pendulums – of equal length, attached to the same point – whose behaviour is determined by differential equations, together with discrete events). \footnote{This approach of using a specific operation for updating the state is not really new. It can, for example, be found in the code of the BouncingHead demo, included in the Java Development Kit 1.0.2. In this paper it is studied systematically for specifying systems.}

We briefly summarize the contents. In Section 2 we start by recalling coalgebraic specification and semantics of object-oriented systems from [35, 21]. This specification formalism is extended with temporal information to handle hybrid systems in Section 3. We illustrate the new hybrid specification formalism by putting several, mostly standard, examples in coalgebraic form. The subsequent Section 4 introduces hybrid models (or implementations) of specifications, and homomorphisms of them. The terminal models of the specification examples from the earlier Section 3 are described (as minimal realizations). Section 5 investigates the model theory of hybrid systems in more detail, focussing on terminal models. First, it gives two different – but isomorphic – descriptions of terminal hybrid models of operations only. The two basic logical notions in the theory of coalgebras, namely bisimilarity and invariance are discussed next. Hybrid bisimilarity is characterized – in the standard way – in terms of equality on terminal hybrid models. Invariants are used for two purposes: first, to construct terminal hybrid models satisfying assertions (from a specification), and second, to obtain refinements between hybrid specifications. Finally, terminality also leads to a behaviour-realization adjunction between hybrid models and hybrid behaviours. We conclude in Section 6 with a brief sketch of how our formalism could handle non-homogeneous systems with continuous input.

2. Discrete coalgebraic specifications and their models

What distinguishes coalgebraic specification from algebraic specification (with initial semantics) is the use of “destructors” instead of “constructors” as atomic operations. Typically, if $X$ is an unknown type that we are specifying and $A$ is a given set, then a map of the form $A \to X$ is a constructor, since it tells us how to form elements of $X$, and a map $X \to A$ is a destructor (also called an observer, or a selector) since it gives us some observations about what is in $X$. In the coalgebraic specification format introduced in this paper we shall restrict ourselves to two kinds of destructors, of the form at : $X \to A$, and proc : $X \times B \to X$ (sometimes written as proc : $X \to X^B$).
The first of these is an attribute giving us some information about states in $X$, and the second one is a procedure which allows us to produce a new state from a given state and a parameter element out of a given set $B$. Attributes correspond to (instance) variables, whose values may be changed by procedures, see the example below. We mostly use the dot-notation (derived from field selection in records) instead of the functional notation. Hence for a state $s \in X$ we write $s\text{.at}$ for $\text{at}(s)$ and $s\text{.proc}(b)$ for $\text{proc}(s,b)$. Thus $s\text{.proc}(b)\text{.at}$ is the result of applying in state $s$ the procedure $\text{proc}$ with parameter $b$, and then applying the attribute $\text{at}$ to its outcome. Functionally, this would be written as $\text{at}(\text{proc}(s,b))$.

We present a typical example of a coalgebraic class specification in Fig. 1, provided with some comments after the #-sign. Such a class specification consists of three sections, describing methods (attributes plus procedures), assertions and creation conditions. It is very similar to an abstract, deferred class in Eiffel [29], in which all methods are deferred (i.e. not interpreted) and in which assertions describe the behaviour. In this paper we do not consider visibility modifiers (allowing methods to be private), like in [21].

The typically coalgebraic aspect of a class specification, such as the one in Fig. 1, is that it tells nothing about what is inside the state space $X$; it only describes the operations on $X$, and certain (behavioural) constraints they satisfy. We restrict equations (here and below) to be exclusively between attribute values, and not between states. This is in line with the coalgebraic philosophy whereby states are not directly accessible. Hence, one cannot speak about equality (=) of states, but only about bisimilarity ($\leftrightarrow$) of states.\footnote{One could also write “equations” with bisimilarity $\leftrightarrow$ instead of equality $=$ between terms inhabiting states, see e.g. [21,23], but we shall not do so in this paper.} Bisimilarity captures indistinguishability by any outside client,
who has only limited access to the state space via the coalgebraic operations. Further
example class specifications in object-oriented languages may be found in [35, 20–23].

Multiple attributes atі : X → Aі , . . . , atm : X → Am can be combined into a single at-
tribute at : X → (A1 × . . . × An). Multiple procedures proc1 : X × B1 → X , . . . , procп : 
X × (B1 + . . . + Bп ) → X , where + is disjoint union (or coproduct). This is convenient since it (formally) al-


dows us to consider systems with only one attribute at : X → A and one procedure proc : X × B → X .

A (mathematical) model of the flip-flop specification in Fig. 1 consists of three parts. First, it consists of an interpretation U = [X] of the unknown type X as a set
(of states). Second, the methods are interpreted as functions [val] : U → {0, 1}, [on] : 
U → U and [off] : U → U acting on the state space U , which should be such that the
above assertions are satisfied. Usually we omit the interpretation braces [−]. Third,
there should be an initial state u0 ∈ U satisfying the creation condition, i.e. satisfying
val(u0) = 0. The three (interpretations of the) methods can be combined into a single function U → {0, 1} × U × U , giving us a coalgebra on U of the functor
X → {0, 1} × X × X , see [24, 21, 36, 37] for more (categorical) background informa-
tion on coalgebras.

Such a model ⟨U → {0, 1} × U × U , u0 ∈ U⟩ of the FF class specification is called
terminal if for every model ⟨V → {0, 1} × V × V , v0 ∈ V⟩ there is a unique function
f : V → U preserving the operations and the initial state:
valU ◦ f = valV , onU ◦ f = f ◦ onV , offU ◦ f = f ◦ offV , f(v0) = u0.

Terminal models are “minimal realizations” (in system-theoretic terminology, see e.g.
[10]); namely, they feature the minimal set of states needed to exhibit the required
behaviour. For example, the terminal model of the above flip-flop specification has the
set U = {0, 1} of attribute values as state space, with operations:

\[
\begin{align*}
\{0, 1\} & \xrightarrow{\text{val}} \{0, 1\} \\
\{0, 1\} & \xrightarrow{\text{on}} \{0, 1\} \\
\{0, 1\} & \xrightarrow{\text{off}} \{0, 1\}
\end{align*}
\]

x → x + 0
x → 1
x → 0

and with 0 ∈ {0, 1} as initial state. It is clear that these operations satisfy the assertions
in Fig. 1, and for an arbitrary model ⟨V → {0, 1} × V × V , v0 ∈ V⟩ there is a unique ho-
omorphism f : V → U = {0, 1} satisfying the above requirements, namely f = valV .
This terminal model captures the way one would implement flip-flops in an efficient
way. There are plenty of other models of this class specification; for example, any set
V with at least two elements can be turned into a model of this specification. But termi-
nal models of coalgebraic specifications distinguish themselves as “optimal” models, in
much the same sense in which initial (term) models of algebraic specifications are “opti-
mal”. See [38, 8] for more information on initial semantics of algebraic specifications.

Although we have described the notion of model only for one particular coalgebraic
class specification, it should be clear what a model is for an arbitrary coalgebraic
specification:
• a carrier set U = [X] serving as state space;
functions \( \text{at}_i : U \rightarrow A_i \) and \( \text{proc}_j : U \times B_j \rightarrow U \), referred to as methods, that act on \( U \); these functions interpret the attributes and procedures, and satisfy the assertions in the specification;

- an initial state \( u_0 \in U \) satisfying the creation conditions in the specification.

A homomorphism of such models is a function \( f : U \rightarrow V \) between state spaces which commutes with the methods (on \( U \) and on \( V \)) and preserves the initial state. In object-oriented notation: \( f(x).\text{at}_i = x.\text{at}_i \), \( f(x).\text{proc}_j(b) = f(x.\text{proc}_j(b)) \) and \( f(u_0) = v_0 \). This allows us to identify terminal models as those models \( Z \) to which there is precisely one homomorphism \( U \rightarrow Z \) from an arbitrary model \( U \). Terminal models, if they exist, are unique up to isomorphism. They are minimal realizations of the required behaviour in a specification.

A class as used in object-oriented languages is identified here as a model of a class specification. An object belonging to a class is simply an element of the underlying state space of the class. The result of creating a new object of a class is the initial state. Method invocation on an object involves applying a coalgebraic operation to the object. In this way we explain some of the basic features of the object-oriented paradigm.

The following result gives an explicit description of terminal coalgebraic models. The first item describes terminal models for specifications with methods only, following [35]. The second item additionally deals with assertions. This result will be generalized later to hybrid models.

**Theorem 1.** (1) The terminal model of a specification with an attribute \( \text{at}: X \rightarrow A \) and a procedure \( \text{proc}: X \times B \rightarrow X \) is given by the state space

\[
Z \overset{\text{def}}{=} A^B^* 
\]

of functions \( \phi : B^* \rightarrow A \) mapping finite input sequences \( (b_1, \ldots, b_n) \in B^* \) to the observable output \( \phi((b_1, \ldots, b_n)) \in A \). The attribute and procedure interpretations are

\[
\begin{align*}
Z & \overset{\text{at}}{\rightarrow} A \\
Z \times B & \overset{\text{proc}}{\rightarrow} Z \\
\phi & \mapsto \phi(\lambda) \\
(\phi, b) & \mapsto \lambda \in B^* \mapsto \phi(b \cdot \lambda)
\end{align*}
\]

where \( \cdot \) denotes the prefixing of an element from \( B \) to a sequence from \( B^* \). Hence, for \( \phi \in Z = A^B^* \) one can write \( \phi((b_1, \ldots, b_n)) = \phi.\text{proc}(b_1) \cdot \ldots \cdot \text{proc}(b_n).\text{at} \).

(2) If we additionally have assertions in our specifications (involving a single state variable \( s \) as above), then these determine a subset

\[
E = \{ \phi \in Z \mid \phi \text{ satisfies the assertions} \} \subseteq Z.
\]

The terminal model satisfying the assertions is given by the submodel \( E \rightarrow Z \) with state space

\[
E \overset{\text{def}}{=} \{ \phi \in Z \mid \forall (b_1, \ldots, b_n) \in B^* \phi.\text{proc}(b_1) \cdot \ldots \cdot \text{proc}(b_n) \in E \}
\]

and with method interpretations inherited from \( Z \).
The subset $E \subseteq E$ can be characterized as the greatest invariant contained in $E$, see [23].

3. Hybrid coalgebraic specifications

In this section we extend coalgebraic class specifications as above, with temporal aspects, and present several examples of the resulting “hybrid coalgebraic class specifications”. These are further illustrated by correctness proofs and by simulations (using the OmSim simulator of the object-oriented hybrid language Omola [3]). The semantics on which this hybrid specification formalism is based will be described in the next section.

A “hybrid” coalgebraic class specification is, like in the previous section, given by a collection of methods consisting of attributes and procedures, but the crucial difference lies in the formulation of the assertions. They will now contain temporal information. For an arbitrary method $\text{meth}$ and a state $s$ we shall use the new notation:

$$s.meth@x \quad \text{for} \quad \begin{cases} \text{the result of applying method } \text{meth} \text{ to the state } s' \\ \text{resulting from } s \text{ in } x \text{ units of time.} \end{cases} \quad (1)$$

Or, more operationally, $s.meth@x$ means: let state $s$ evolve for $x$ units of time and then apply method $\text{meth}$. In untimed coalgebraic specification, as in the previous section, a state-change of an object could only result from a procedure application. But, in the hybrid setting, objects are “active” and may evolve autonomously in time. For example, the state of a kettle (water cooker) may change under the influence of time. Of course, a client can still cause a state-change via a procedure application (e.g. switching the kettle on or off).

(Later in Section 6 we shall further extend the notation $s.meth@x$ to $s.meth@(x,u)$ where $u$ is a continuous input function with the interval $[0,x]$ as domain; this $u$ additionally influences the evolution of the state $s$ during the interval $[0,x]$ preceding our method application.)

We shall consider examples where the time parameter $x$ ranges over natural numbers $\mathbb{N}$ (discrete time) and also over positive real numbers $\mathbb{R}_{>0}$ (real time). We allow $x$ to be 0, so that $s.meth_1@x.meth_2@0$ means that $meth_2$ is applied immediately after $meth_1$ (which is applied after a delay of $x$ time units to state $s$). We assume that messages arrive in sequential order; if we write $s.meth@x$, then it is assumed that $meth$ is the first method to be applied in state $s'$, resulting from $s$ after $x$ units of time, and that no other method was applied in the meantime. If $meth$ is a method that takes a parameter $b \in B$, then we shall write $s.meth(b)@x$ for the result of applying $meth(b)$ after $x$ units of time.

Hybrid coalgebraic class specifications will be presented in a format similarly to that of (ordinary, untimed) coalgebraic class specifications (see Fig. 1), except that we start with the keyword “DT-class spec” or “RT-class spec”, instead of just “class spec”. “DT” and “RT” stand for “discrete time” and “real time”. The time variables $x, \beta, \ldots$ range over the natural numbers $\mathbb{N} = \{0, 1, 2, \ldots\}$ in discrete time, and over the positive real numbers $\mathbb{R}_{>0} = \{x \in \mathbb{R} | x \geq 0\}$ in real time.
In the rest of this section we shall consider examples of hybrid coalgebraic specifications, involving timers, chemical reactions and thermostats. In some of these examples we illustrate the specified behaviour via explicit correctness statements and/or via simulations.

### 3.1. Timers

We start with an elementary example, building on the flip-flop specification from the previous section. We now wish to specify flip-flops which can be switched on by a client, and will automatically switch off after 10 units of discrete time. See Fig. 2 for the specification. Notice that the `off` procedure is no longer needed. We explain the meaning of the assertions, since they contain the new temporal aspects. We use the turnstile \( \vdash \) to describe conditional assertions. The first “monotony” assertion tells that if at any time \( x \) the value in state \( s \) is 0, then this value is still 0 at some later time \( x + \beta \). Hence the flip-flop does not simply switch on (get value 1) by itself. In this hybrid coalgebraic format one has to indicate explicitly what the values of attributes are as a function of time. The second assertion tells us that no matter in what state our flip-flop is, if we wait at least 10 units of time, then its value will be 0. Finally, if we switch it on at some time \( x \), and then inspect it at some time \( \beta \) less than 10 units later, then it will have value 1. This formally captures our informally described timer. The creation clause tells us that newly created instances of this class have value 0 immediately after their creation. Then we can deduce `new.val[0] = 0` for any \( x \), from the first assertion.

Notice that objects for this DTFF specification (i.e. actual timed flip-flops) are active objects: if they are switched on by a client, then they switch off autonomously. Nothing is specified about this internal activity during the time period between (externally) switching on and (internally) switching off. There are various ways to realize the specified behaviour, as we shall see in Section 4.2.
The time of 10 units that these flip-flops will remain “on” is of course completely arbitrary. We could, more generally, write a parametrized specification DTFF(dur: N) where the parameter dur (instead of the constant 10) describes the time the flip-flops remain “on”. Also, we could allow a client to set this time dur via an explicit procedure (as in the ART class specification in Fig. 10 below).

In order to familiarize the reader with this hybrid specification formalism, we consider a few variations. Notice that a timed flip-flop satisfying the above specification can be switched on (again) if it has value 1. In this way we can keep it with value 1 for a longer time than 10. Suppose we wish to alter this and stipulate that the flip-flop can only be switched on if it has value 0. We can achieve this by taking the following two assertions, instead of the above third assertion:

\[
\begin{align*}
&\text{s.val}@x = 0, \beta < 10 \vdash \text{s.on}@x.\text{val}@\beta = 1 \\
&\text{s.val}@x = 1 \vdash \text{s.on}@x.\text{val}@\beta = \text{s.val}@x + \beta).
\end{align*}
\] (2)

The first assertion in (2) is like above, except that it now has an extra assumption that the value is 0 at the moment x when the “on-event” happens. This reflects our modification. The second assertion tells us that at a moment x when the value is 1, an “on-event” has no effect on the value; that is, the value at \(\beta\) time units later is the same as the value \(x + \beta\) at time units after the original starting point. One can further modify this example by requiring that after the timer has had value 1, it must remain with value 0 for at least 20 units (say) of time. This comes close to the (single) traffic light specification for pedestrians in [16] with value 0 standing for “red light” and 1 for “green light”. An auxiliary (possibly private) attribute \(\text{waiting}: X \rightarrow \{\text{yes, no}\}\) is needed, telling whether one has waited long enough in a state with value 0 (to switch the flip-flop on again). Details of such a specification are left to the reader. Similarly one can coalgebraically specify standard timer examples like an automated railway crossing where the times needed to open and to close the gate are explicitly taken into account.

We now consider a real-time version RTFF of the above discretely timed flip-flop DTFF. Its specification is the same as the discrete time specification in Fig. 2, except that in order to deal with boundary conditions we add an extra “denseness” assertion

\[
\text{s.val}@x = 1, \exists \beta > 0 \text{ s.val}@x(\alpha + \beta) = 1.
\] (3)

It tells us that if the value at time \(x\) is 1, then we can always find a (possibly very small) non-zero positive real number \(\beta\) such that \(\beta\) units of time later the value is still 1. As a consequence, if \(\text{s.val}@x = 1\), then the set \(\{\beta | \text{s.val}@x(\alpha + \beta) = 1\}\) is an upwardly open interval \([0, \gamma) \subseteq \mathbb{R}_{>0}\).

We conclude this subsection on hybrid timers with a more complicated example, which is well-known in the literature, see e.g. [27, 39]). It involves a “watch-dog” surveying a number of processes and expecting signals from them confirming at regular intervals that everything is all-right, see Fig. 3. For convenience, our specification only involves two processes, where the first process should give a confirmation signal every
RT-class spec: WD($\text{dur}_1$: $\mathbb{R}_{>0}$, $\text{dur}_2$: $\mathbb{R}_{>0}$)

methods:

time$_1$: $X \rightarrow [0, \text{dur}_1)$
time$_2$: $X \rightarrow [0, \text{dur}_2)$
time: $X \rightarrow \mathbb{R}$
alarm: $X \rightarrow \{0, 1\}$
conf$_1$: $X \rightarrow X$
conf$_2$: $X \rightarrow X$
reset: $X \rightarrow X$

assertions:

\begin{align*}
\text{s.time}@x &= \min\{\text{s.time}_1}@x, \text{s.time}_2}@x\} \\
\text{s.time}@x &= 0 \land \text{s.alarm}@x = 1 \\
\text{s.time}@x &= 0 \land \text{s.alarm}@x = 0 \\
\beta < \text{s.time}@x &\land \text{s.time}_1}@x + \beta = (\text{s.time}_1}@x) - \beta \\
\beta < \text{s.time}@x &\land \text{s.time}_2}@x + \beta = (\text{s.time}_2}@x) - \beta \\
\beta \geq \text{s.time}@x &\land \text{s.time}_1}@x + \beta = 0 \\
\beta \geq \text{s.time}@x &\land \text{s.time}_2}@x + \beta = 0 \\
\text{s.conf}_1}@x.\text{time}_2}@0 &= \text{s.time}_2}@x \\
\text{s.conf}_2}@x.\text{time}_1}@0 &= \text{s.time}_1}@x \\
\text{s.alarm}@x &= 0 \land \text{s.conf}_1}@x.\text{time}_1}@0 = \text{dur}_1 \\
\text{s.alarm}@x &= 0 \land \text{s.conf}_2}@x.\text{time}_2}@0 = \text{dur}_2 \\
\text{s.alarm}@x &= 1 \land \text{s.conf}_1}@x.\text{time}_1}@0 = \text{s.time}_1}@x \\
\text{s.alarm}@x &= 1 \land \text{s.conf}_2}@x.\text{time}_2}@0 = \text{s.time}_2}@x \\
\text{s.reset}@x.\text{time}_1}@0 &= \text{dur}_1 \\
\text{s.reset}@x.\text{time}_2}@0 &= \text{dur}_2
\end{align*}

creation:

\begin{align*}
\text{new.time}_1}@0 &= \text{dur}_1 \\
\text{new.time}_2}@0 &= \text{dur}_2
\end{align*}

end class spec

Fig. 3. A specification of a watchdog surveying two processes, by raising alarm unless confirmation signals are given regularly.

dur$_1$ time-units, and the second process should give a signal every dur$_2$ units. These (non-zero, real) times dur$_1$, dur$_2$ are parameters in the specification. The specification contains two clocks, indicating a value in the intervals $[0, \text{dur}_1), [0, \text{dur}_2] \subseteq \mathbb{R}$. The first clock s.time$_1}@x$ indicates the remaining time for the first process to confirm. And s.time$_2}@x$ has the same rôle for the second process. If a confirmation signal conf$_1$ comes in from the first process, then time$_1$ will be set to dur$_1$, without affecting time$_2$. Similarly for conf$_2$ signals. In order to improve the readability we use two additional attributes as abbreviations: time is the minimum of time$_1$ and time$_2$, and alarm is 1.
(corresponding to the alarm status) if and only if time is 0. If one of the processes fails to deliver a confirmation signal in time, then both clocks, given by time$_1$ and time$_2$, are set to 0 and alarm is raised (i.e. the alarm attribute gets value 1). The system stays in this state of alarm until it is reset, via the reset procedure.

We wish to formally show that this watchdog behaves as it should. This turns out to be a non-trivial task. First, it involves a precise formulation of the required behaviour, in response to timed sequences of confirmation signals. Such a sequence of signals may be described in terms of elements \( \langle i, \beta \rangle \) where \( i \in \{1, 2\} \) tells whether the confirmation signal is from process 1 or from process 2, and \( \beta \in \mathbb{R}_{\geq 0} \) says at what time the confirmation signal arrived. A sequence of confirmation inputs can thus be identified with a sequence \( \sigma \in \{1, 2\} \times \mathbb{R}_{\geq 0}^* \). For such a \( \sigma \) and for an arbitrary state \( s \) we shall use the ad hoc notation \( s.\text{conf}(\sigma) \) for the state resulting from \( s \) by the confirmation signals in \( \sigma \). This is defined as follows:

\[
\begin{align*}
s.\text{conf}(\sigma) & \stackrel{\text{def}}{=} \\
& \begin{cases} 
  s & \text{if } \sigma \text{ is the empty sequence } \langle \rangle, \\
  s.\text{conf}_1 @ \beta.\text{conf}(\sigma') & \text{if } \sigma = \langle 1, \beta \rangle \cdot \sigma', \\
  s.\text{conf}_2 @ \beta.\text{conf}(\sigma') & \text{if } \sigma = \langle 2, \beta \rangle \cdot \sigma'. 
\end{cases}
\end{align*}
\]

We wish to characterize the occurrence of the alarm state \( s.\text{reset}@\beta.\text{conf}(\sigma).\text{alarm}@0 \) = 1, resulting after first resetting an arbitrary state \( s \) and then presenting it with confirmation signals as in \( \sigma \), in terms of this sequence \( \sigma \in \{1, 2\} \times \mathbb{R}_{\geq 0}^* \) of signals. If the specification is correct, we should be able to derive this occurrence of alarm if and only if the confirmation signals \( \langle 1, \beta \rangle \) and \( \langle 2, \beta \rangle \) in \( \sigma \) are “too far apart”. We shall make this idea of being “too far apart” mathematically precise by defining two derived sequences \( \sigma_1, \sigma_2 \in (\mathbb{R}_{\geq 0})^* \) describing the times at which confirmations signals of process 1 and of process 2 are received. We therefore define

\[
\sigma_1 \equiv 0 \odot \sigma \quad \text{and} \quad \sigma_2 \equiv 0 \odot \sigma
\]

where \( \odot \) and \( \oplus \) are auxiliary “filter” operations of type \( \mathbb{R}_{\geq 0} \times (\{1, 2\} \times \mathbb{R}_{\geq 0})^* \rightarrow (\mathbb{R}_{\geq 0})^* \) defined by

\[
\begin{align*}
\text{if } \sigma = \langle \rangle \text{ and } x = 0: \ x \odot \sigma = \langle \rangle \quad \text{and} \quad x \oplus \sigma = \langle \rangle \\
\text{if } \sigma = \langle \rangle \text{ and } x > 0: \ x \odot \sigma = \langle x \rangle \quad \text{and} \quad x \oplus \sigma = \langle x \rangle \\
\text{if } \sigma = \langle 1, \beta \rangle \cdot \sigma': \ x \odot \sigma = (x + \beta) \cdot (0 \odot \sigma') \quad \text{and} \quad x \oplus \sigma = (x + \beta) \oplus \sigma' \\
\text{if } \sigma = \langle 2, \beta \rangle \cdot \sigma': \ x \odot \sigma = (x + \beta) \odot \sigma' \quad \text{and} \quad x \oplus \sigma = (x + \beta) \cdot (0 \odot \sigma').
\end{align*}
\]

Then, for example,

\[
\begin{align*}
(\langle 2, 3 \rangle \cdot \langle 2, 6 \rangle \cdot \langle 1, 1 \rangle \cdot \langle 1, 3 \rangle \cdot \langle 2, 5 \rangle \cdot \langle 2, 3 \rangle) & \equiv 1 \\
= 0 \odot (\langle 2, 3 \rangle \cdot \langle 2, 6 \rangle \cdot \langle 1, 1 \rangle \cdot \langle 1, 3 \rangle \cdot \langle 2, 5 \rangle \cdot \langle 2, 3 \rangle) \\
= 3 \odot (\langle 2, 6 \rangle \cdot \langle 1, 1 \rangle \cdot \langle 1, 3 \rangle \cdot \langle 2, 5 \rangle \cdot \langle 2, 3 \rangle) \\
= 9 \odot (\langle 1, 1 \rangle \cdot \langle 1, 3 \rangle \cdot \langle 2, 5 \rangle \cdot \langle 2, 3 \rangle)
\end{align*}
\]
\[ 10 \cdot (0 \odot (1, 3) \cdot (2, 5) \cdot (2, 3)) \]
\[ = 10 \cdot 3 \cdot (0 \odot (2, 5) \cdot (2, 3)) \]
\[ = 10 \cdot 3 \cdot (5 \odot (2, 3)) \]
\[ = 10 \cdot 3 \cdot 8. \]

The resulting sequence \( 10 \cdot 3 \cdot 8 \) incrementally describes the times at which \( \text{conf}_1 \) signals (given by tuples with first component equal to 1) occur in the original (mixed) sequence \( (2, 3) \cdot (2, 6) \cdot (1, 1) \cdot (1, 3) \cdot (2, 5) \cdot (2, 3) \) of both \( \text{conf}_1 \) and \( \text{conf}_2 \) signals. (The last element 8 of the sequence \( 10 \cdot 3 \cdot 8 \) collects the time since the last \( \text{conf}_1 \) signal.)

We are now in a position to formulate and prove the correctness of the watchdog specification in Fig. 3. We use the notation \( a \sigma \), for \( a \in A \) and \( \sigma \in A^* \), to mean occurrence of \( a \) in \( \sigma \).

**Proposition 2** (Watchdog correctness). For a sequence of confirmation signals \( \sigma \in (\{1, 2\} \times \mathbb{R}_{\geq 0})^* \) and for an arbitrary state \( s \), the alarm state after the occurrence of \( s : \text{reset} \odot \text{conf}(\sigma) \odot \text{alarm} @ 0 \) occurs if and only if one of the confirmation signals in \( \sigma \) is too late, i.e.

\[ \exists \alpha \in \sigma \mid \alpha > \text{dur}_1 \quad \text{or} \quad \exists \alpha \in \sigma \mid \alpha > \text{dur}_2. \]

**Proof.** We first mention two easy auxiliary results, whose proof we leave to the reader.

(a) \( s \cdot \text{time}_1 @ x = 0 \Leftrightarrow s \cdot \text{time}_2 @ x = 0 \Leftrightarrow s \cdot \text{time} @ x = 0 \Leftrightarrow s \cdot \text{alarm} @ x = 1. \)

(b) \( s \cdot \text{alarm} @ 0 = 1 \Rightarrow s \cdot \text{conf}(\sigma) \cdot \text{alarm} @ x = 1. \)

We start with the (only if)-case of the statement in the proposition. Therefore we use the following (suitably loaded) auxiliary statement (A): for all sequences of confirmation signals \( \sigma \in (\{1, 2\} \times \mathbb{R}_{\geq 0})^* \) and for all non-alarm states \( s \) (i.e. \( s \) with \( s \cdot \text{alarm} @ 0 = 0 \), if all confirmation signals in \( \sigma \) arrive in time:

\[ \forall \alpha \in ((\text{dur}_1 - s \cdot \text{time}_1 @ 0) \odot \sigma) \alpha < \text{dur}_1 \]

and

\[ \forall \alpha \in ((\text{dur}_2 - s \cdot \text{time}_2 @ 0) \odot \sigma) \alpha < \text{dur}_2 \]

then after \( \sigma \) there is no alarm:

\[ s \cdot \text{conf}(\sigma) \cdot \text{alarm} @ 0 = 0. \]

This statement (A) is proved by induction on \( \sigma \).

- If \( \sigma = () \), then \( s \cdot \text{conf}(\sigma) \cdot \text{alarm} @ 0 = s \cdot \text{alarm} @ 0 = 0 \) holds by assumption.
- If \( \sigma = (\beta, \cdot \sigma') \), then our aim is to apply the induction hypothesis to the (smaller) sequence \( \sigma' \) with state \( s' = s \cdot \text{conf}_1 @ \beta \). This will yield the required conclusion \( 0 = s' \cdot \text{conf}(\sigma') \cdot \text{alarm} @ 0 = s \cdot \text{conf}(\sigma) \cdot \text{alarm} @ 0. \)
We thus have to check that \( \sigma' \) with \( s' \) satisfies the assumptions of (A). We start by calculating
\[
(dur_1 - s.time_1) \odot \sigma = (dur_1 - s.time_1 @ 0 + \beta) \cdot (0 \odot \sigma')
\]
using the definition of \( \odot \). Hence, by the assumptions about \( \sigma \) we get \( dur_1 - s.time_1 @ 0 + \beta < dur_1 \), i.e. \( \beta < time_1 @ 0 \), and \( x < dur_1 \) for all \( x \in 0 \odot \sigma' = ((dur_1 - s'.time_1 @ 0) \odot \sigma') \).

Similarly we have
\[
(dur_2 - s.time_2) \odot \sigma = (dur_2 - s.time_2 @ 0 + \beta) \odot \sigma'
\]
yielding \( \beta < s.time_2 @ 0 \) (and thus \( \beta < s.time@0 \), so that
\[
(dur_2 - s.time_2) \odot \sigma = (dur_2 - (s.time_2 @ 0 - \beta)) \odot \sigma'
\]
\[
= (dur_2 - s.time_2 @ 0) \odot \sigma'
\]
\[
= (dur_2 - s'.time_2 @ 0) \odot \sigma'.
\]
Hence we also have \( x < dur_2 \) for all \( x \in ((dur_2 - s'.time_2 @ 0) \odot \sigma') \).

We still have to check that \( s'.alarm@0 = 0 \). But this follows from \( \beta < s.time_2 @ 0 \) by
\[
s'.alarm@0 = 0 \iff s'.time@0 = \min\{s.conf_1 @ \beta, time_1 @ 0,\}
\]
\[
s.conf_1 @ \beta, time_2 @ 0\} > 0
\]
\[
\iff \min\{dur_1, s.time_2 @ \beta\} > 0
\]
\[
\iff s.time_2 @ \beta = s.time_2 @ 0 - \beta > 0.
\]
* The case \( \sigma = (2, \beta) \cdot \sigma' \) is handled symmetrically.

For the proof of the (only if)-statement in the proposition: \( s.reset@\gamma.conf(\sigma).alarm@0 = 1 \) implies \( \exists x \in \sigma_1 | x \geq dur_1 \) or \( \exists x \in \sigma_2 | x \geq dur_2 \), assume that the conclusion does not hold, i.e. that \( \forall x \in \sigma_1 | x < dur_1 \) and \( \forall x \in \sigma_2 | x < dur_2 \). We can then apply statement (A) with the sequence of signals \( \sigma \) and with state \( s.reset@\gamma.time(\sigma) > 0 \), and so \( s.reset@\gamma.conf(\sigma).alarm(\sigma) = 0 \). And secondly \((dur_1 - s.reset@\gamma.time_1(\sigma) @ 0) \odot \sigma\) = \( 0 \odot \sigma\) = \( \sigma_1 \), and similarly for \( \sigma_2 \). Hence, we can conclude \( s.reset@\gamma.conf(\sigma).alarm(\sigma) = 0 \) by (A), which contradicts the original assumption.

Conversely, for the (if)-part of the proposition, we assume that at some stage we do not have a confirmation signal in time, i.e. that \( \exists x \in \sigma_1 | x \geq dur_1 \) or \( \exists x \in \sigma_2 | x \geq dur_2 \). We consider the first stage where there is no such confirmation signal, say from process 1. Then we can write
\[
\sigma = \tau \cdot (2, \beta_1) \cdots (2, \beta_n) \cdot \rho
\]
where
* \( (2, \beta_1) \cdots (2, \beta_n) \) is the sequence of all \( conf_2 \) signals that arrived (in time) before the time ran out for the first process. Hence \( \beta_i < dur_2 \).
contains the earlier confirmation signals that arrived in time. Either \( \tau \) the empty sequence \( \langle \rangle \), and then \( \text{s.reset} @ \gamma . \text{conf} (\tau) . \text{time} @ 0 = \text{s.reset} @ \gamma . \text{time} @ 0 = \text{dur}_1 \). Or \( \tau \) is of the form \( \tau = \tau' \cdot \langle 1, \beta \rangle \). The (only if)-part that we have just proved now yields \( \text{s.reset} @ \gamma . \text{conf}(\tau') \). \text{alarm} @ 0 = 0 \) and so we can conclude \( \text{s.reset} @ \gamma . \text{conf}(\tau) . \text{time} @ 0 = \text{dur}_1 \).

- \( \rho \) is either the empty sequence \( \langle \rangle \) so that \( \beta_1 + \cdots + \beta_n \geq \text{dur}_1 \), or \( \rho = \langle 1, \delta \rangle \cdot \rho' \) with \( \beta_1 + \cdots + \beta_n + \delta \geq \text{dur}_1 \). In the second case a confirmation signal from the first process still arrived, but too late.

For a \( \beta_{n+1} \in \mathbb{R}_{\geq 0} \) with \( \beta_{n+1} < \text{dur}_2 \) we can compute, by induction on \( i \leq n \)

\[
\begin{align*}
\text{s.reset} @ \gamma . \text{conf}(\tau \cdot \langle 2, \beta_1 \rangle \cdots \langle 2, \beta_i \rangle) . \text{time} @ \beta_{i+1} &= \text{s.reset} @ \gamma . \text{conf}(\tau) . \text{conf}_2 @ \beta_1 \cdots \text{conf}_2 @ \beta_i . \text{time} @ \beta_{i+1} \\
&= \left\{ \begin{array}{ll}
\text{dur}_1 - (\beta_1 + \cdots + \beta_{i+1}) & \text{if } \beta_1 + \cdots + \beta_{i+1} < \text{dur}_1 \\
0 & \text{otherwise}
\end{array} \right.
\end{align*}
\]

In case \( \rho = \langle \rangle \), this yields the required alarm state \( \text{s.reset} @ \gamma . \text{conf}(\sigma) . \text{alarm} @ 0 = 1 \) by taking \( \beta_{n+1} = 0 \). And if \( \rho = \langle 1, \delta \rangle \cdot \rho' \), we take \( \beta_{n+1} = \delta \) so that we may conclude

\[
\begin{align*}
\text{s.reset} @ \gamma . \text{conf}(\tau \cdot \langle 2, \beta_1 \rangle \cdots \langle 2, \beta_n \rangle) . \text{conf}_1 @ \delta . \text{time} @ 0 &= \text{s.reset} @ \gamma . \text{conf}(\tau \cdot \langle 2, \beta_1 \rangle \cdots \langle 2, \beta_n \rangle) . \text{time} @ \delta \\
&= 0.
\end{align*}
\]

Hence also this second case leads to alarm: \( \text{s.reset} @ \gamma . \text{conf}(\sigma) . \text{alarm} @ 0 = 1 \), by statement (b) in the beginning of this proof. \( \square \)

### 3.2. Chemical processes

In chemical process theory, one finds the typical hybrid combination of continuous and discrete behaviour. That is to say, the evolution of chemical processes may be described by differential equations, and the control of such processes via discrete actions, like opening or closing of valves and (instantaneous) addition of extra substances, may be described via procedures acting on the state space (see for example [33]). In this subsection we shall consider two such simple situations, involving chemical substances in a reaction container. The second example specification in Fig. 5 will be illustrated with a simulation, see Fig. 6. The link between the specification and the simulation will be provided by Definition 3, describing an alternative formulation for the observable behaviour in a particular state.

In a first example we assume we have control over a confined reaction space into which we can inject a chemical substance \( A \). In this space, \( A \) will start reacting and transforming itself to another substance, with a reaction speed proportional to the available amount of \( A \). If we write this amount as a function \( A = A(z) \) depending on a time parameter \( z \in \mathbb{R}_{\geq 0} \), then we have a differential equation

\[
\frac{dA}{dz} = -kA \quad \text{where } k \in \mathbb{R}_{\geq 0} \text{ is a reaction constant.} \quad (4)
\]
**RT-class spec:** REACT\(_A(k : \mathbb{R}_{>0})\)

**methods:**
- amount: \(X \rightarrow \mathbb{R}_{>0}\)
- add: \(X \times \mathbb{R}_{>0} \rightarrow X\)
- clear: \(X \rightarrow X\)

**assertions:**
- \(s\).add(\(x\))\_at\_x\_amount@0 = (s\_amount@x) + x
- \(s\).clear\_at\_x\_amount@0 = 0
- \(s\_amount@(x + \beta) = (s\_amount@x) \cdot e^{-k\beta}\)

**creation:**
- new.\_amount@0 = 0

*Fig. 4. A specification of a reaction space with a single substance \(A\).*

The solution of this equation is the function \(A(x) = A(0) \cdot e^{-kx}\). It is used in the class specification in Fig. 4. This specification involves an amount attribute which tells us how much \(A\) there is (in our confined reaction space). And it has two procedures add and clear which allow us to inject a certain amount of \(A\) (using the parameter of the add method), and to clear the space in which we are working. This explains the first two assertions. The third assertion incorporates the solution of the differential equation: it tells the amount of \(A\) at any time \(\beta\) after \(x\), in terms of the amount of \(A\) at \(x\) and the elapsed time \(\beta\).

Our specification formalism allows us to compute the amount of \(A\) at a specific time after a series of actions. For instance, in an arbitrary state \(s\), we first clear our working space, 1 time-unit later we inject 10 units of \(A\), then 8 time-units later we decide to inject another 5 units of \(A\), and then we check 3 time units later. The result can be computed as

\[
\begin{align*}
\text{s.clear@0.add(10)@1.add(5)@8.amount@3} &= \text{s.clear@0.add(10)@1.add(5)@8.amount@0 + 3} \\
&= \text{(s.clear@0.add(10)@1.add(5)@8.amount@0) \cdot e^{-3k}} \\
&= \text{((s.clear@0.amount@1 + 10) \cdot e^{-8k} + 5) \cdot e^{-3k}} \\
&= \text{((s.clear@0.amount@0) \cdot e^{-k} + 10) \cdot e^{-8k} + 5) \cdot e^{-3k}} \\
&= \text{(10 \cdot e^{-8k} + 5) \cdot e^{-3k}} \\
&= 10 \cdot e^{-11k} + 5 \cdot e^{-3k}.
\end{align*}
\]
The first factor $10 \cdot e^{-11 \cdot t}$ in the outcome captures the amount of $A$ after inserting 10 units of $A$ and waiting 11 time-units, whereas the second factor $5 \cdot e^{-3 \cdot t}$ captures the amount after waiting 3 time-units starting from 5 units of $A$.

A more interesting example arises when we can (independently) inject two substances $A$ and $B$, which can engage in reactions $A + B$, say, both with reaction speed proportional to the amount of transforming substance, and such that an $x$-amount of $A$ (resp. $B$) is transformed into an $x$-amount of $B$ (resp. $A$). This leads to the differential equation

$$\frac{dA}{dx} = -kA + \ell B \quad \text{where } A(x) + B(x) = A(0) + B(0).$$

In the first equation, $k$, $\ell$ are constants (in $\mathbb{R} > 0$). The second equation tells that the total amount of $A$ plus $B$ must be constant (and equal to the sum in the initial state). The solution of this equation,

$$A(x) = \frac{1}{k + \ell}((kA(0) - \ell B(0)) \cdot e^{-(k+\ell) \cdot x} + \ell(A(0) + B(0))),$$

$$B(x) = A(0) + B(0) - A(x)$$

is incorporated into the class specification in Fig. 5. It has two attributes $\text{amount}_A$ and $\text{amount}_B$ describing the amounts of the substances $A$ and $B$, and two procedures $\text{add}_A$ and $\text{add}_B$ allowing us to add amounts of $A$ and of $B$ independently. Notice that this independence is enforced explicitly by the assertions $\text{add}_B(x)@x.\text{amount}_A@0 = \text{amount}_A@x$ and $\text{add}_A(x)@x.\text{amount}_B@0 = \text{amount}_B@x$.

Let $s$ be an arbitrary state, and put $s_s = s.\text{clear}@0.\text{add}_A(x)@0$. Then one can show that, as the time $\beta$ goes to infinity, the amount $s_s.\text{amount}_A@\beta$ of $A$ in state $s_s$ at time $\beta$ goes to $\ell/(k+\ell) \cdot x$, and the amount $s_s.\text{amount}_B@\beta$ of $B$ goes to $k/(k+\ell) \cdot x$. Hence the eventual ratio $A/B$ is $\ell/k$, see the simulation in Fig. 6. It illustrates the behaviour of such a chemical system via the simulator OmSim [3]. The output describes both the (observable) attributes $\text{amount}_A$ (solid line) and $\text{amount}_B$ (dashed line). But this output is not given by function expressions

$$x \mapsto s.\text{proc}(b_1)@\beta_1, \ldots, \text{proc}(b_n)@\beta_n, \text{at}@x$$

like in the computation (5), since these expressions describe the attribute value after the procedure applications $\text{proc}(b_1)@\beta_1, \ldots, \text{proc}(b_n)@\beta_n$. The simulation in Fig. 6 gives the attribute values right from the beginning when we start applying these procedures. This requires some new notation.

**Definition 3.** Consider a state $s$ in a specification with attribute $\text{at}$ and procedure $\text{proc}$. We use the (underline) $\text{at}$ notation

$$s.\text{at}@x[\text{proc}(b_1)@\beta_1, \ldots, \text{proc}(b_n)@\beta_n]$$

for the attribute value of state $s$ after $x$ time-units, while the procedures $\text{proc}(b_1)@\beta_1, \ldots, \text{proc}(b_n)@\beta_n$ are being applied to the state $s$. Formally we define $s.\text{at}@x[\sigma]$ by
RT-class spec: \textsc{React}_{A,B}(k: \mathbb{R}_{\geq 0}, \ell' : \mathbb{R}_{\geq 0})

methods:

\texttt{amount}_A : X \rightarrow \mathbb{R}_{\geq 0}
\texttt{add}_A : X \times \mathbb{R}_{\geq 0} \rightarrow X
\texttt{amount}_B : X \rightarrow \mathbb{R}_{\geq 0}
\texttt{add}_B : X \times \mathbb{R}_{\geq 0} \rightarrow X
\texttt{clear} : X \rightarrow X

assertions:

\texttt{s.add}_A(x)@x.\texttt{amount}_A@0 = (s.\texttt{amount}_A@x) + x
\texttt{s.add}_B(x)@x.\texttt{amount}_B@0 = s.\texttt{amount}_B@x
\texttt{s.clear}@x.\texttt{amount}_A@0 = 0
\texttt{s.amount}_A@(x + \beta) = \frac{1}{k + \ell'})((k(s.\texttt{amount}_A@x)) - \ell((s.\texttt{amount}_B@x)) \cdot e^{-(k + \ell')x}
+ \ell((s.\texttt{amount}_A@x) + (s.\texttt{amount}_B@x)))
\texttt{s.add}_B(x)@x.\texttt{amount}_B@0 = (s.\texttt{amount}_B@x) + x
\texttt{s.add}_A(x)@x.\texttt{amount}_B@0 = s.\texttt{amount}_B@x
\texttt{s.clear}@x.\texttt{amount}_B@0 = 0
\texttt{s.amount}_B@(x + \beta) = (s.\texttt{amount}_A@x) + (s.\texttt{amount}_B@x)
- (s.\texttt{amount}_A@(x + \beta))

creation:

\texttt{new.\texttt{amount}_A@0 = 0}
\texttt{new.\texttt{amount}_B@0 = 0}

end class spec

Fig. 5. A specification of a reaction space with two interacting substances \textit{A} and \textit{B}.

---

Fig. 6. Initially, \textit{A} = 0 and \textit{B} = 0. Additions: 10 units of \textit{A} at time = 8, and 5 units of \textit{B} at time = 25. (The values of the constants in this simulation are: \textit{k} = 0.2 and \textit{\ell'} = 0.3. Hence the eventual ratio \textit{A}/\textit{B} is 3/2.)
induction on the length of the sequence \( \sigma \) of procedures:

\[
\begin{align*}
\text{s. at @z[()]} &= \text{s. at @} \\
\text{s. at @z[(\text{proc}(b)@\beta) \cdot \sigma]} &= 
\begin{cases} 
\text{s. at @} & \text{if } z < \beta, \\
\text{s. proc(b)@\beta. at @} (\sigma) & \text{if } z \geq \beta.
\end{cases}
\end{align*}
\]

The output of the simulation in Fig. 6 can now be described formally by the pair of functions

\[
\alpha \mapsto \text{new. amount}_a@z[\text{add}_a(10)@8. \text{add}_a(5)@15] \\
\alpha \mapsto \text{new. amount}_b@z[\text{add}_a(10)@8. \text{add}_b(5)@15]
\]

on the interval \([0, 50] \subseteq \mathbb{R}_{\geq 0}\).

Later, in Section 5.1, we shall see that the two ways of describing observations (after the procedure applications, like in (5), and during the procedure applications, like in the above definition) both give rise to a terminal model.

### 3.3. Thermostats

This final subsection with example specifications describes two thermostats. These are adapted from [32, 2] and are put in coalgebraic format. We shall describe a “passive” and an “active” version. The passive thermostat PTERM lets the user regulate the temperature in a room, via ‘on’ and ‘off’ switches of a heater (like for the earlier flip-flops in Fig. 1). There are two attributes, namely ‘val’ describing whether the heater is on or off, and ‘temp’ describing the temperature in the room. We have to consider the following two cases.

- When the heater is off, the temperature in the room is determined by “Newton’s law of cooling”: the rate of change \( dT/dz \) of the temperature \( T = T(z) \) in the room is proportional to the difference between the temperature \( T \) in the room and the temperature of its surroundings. For convenience, we assume the latter to be constantly 0, so that we have a differential equation

\[
\frac{dT}{dz} = -kT, \quad \text{with solution } T(z) = T(0) \cdot e^{-kz}.
\]

- If the heater is switched on, we assume that the rate of change of temperature due to heating is constant. Hence, we have an extra constant \( \ell \in \mathbb{R}_{\geq 0} \) in our differential equation:

\[
\frac{dT}{dz} = -kT + \ell, \quad \text{with solution } T(z) = \left( T(0) - \frac{\ell}{k} \right) \cdot e^{-kz} + \frac{\ell}{k}.
\]

(These solutions are also used in [32, 2].) We thus arrive at the class specification in Fig. 7.\footnote{In this PTERM specification we essentially repeat the flip-flop specification from Fig. 1 (in hybrid form). With inheritance one could import such a (hybrid) flip-flop specification so that only the additional temp attribute should be added.} What is interesting about this passive thermostat example is that different
RT-class spec: `PTERM(k: \mathbb{R}_{\geq 0}, \ell: \mathbb{R}_{\geq 0})`

methods:
- `val:X \rightarrow \{0, 1\}`
- `temp:X \rightarrow \mathbb{R}_{\geq 0}`
- `on:X \rightarrow X`
- `off:X \rightarrow X`

assertions:
- `s.val@ (x + \beta) = s.val@ x`
- `s.on@ x. val@ 0 = 1`
- `s.off@ x. val@ 0 = 0`
- `s.on@ x. temp@ 0 = s.temp@ x`
- `s.off@ x. temp@ 0 = s.temp@ x`
- `s.val@ x = 0 \models s.temp@ (x + \beta) = (s.temp@ x) \cdot e^{-k\beta}`
- `s.val@ x = 1 \models s.temp@ (x + \beta) = ((s.temp@ x) - \frac{\ell}{k}) \cdot e^{-k\beta} + \frac{\ell}{k}`

creation:
- `new. val@ 0 = 0`
- `new. temp@ 0 = 0`

end class spec

Fig. 7. A specification of "passive thermostats", which allow the user to regulate the temperature in a room by switching a heater on or off.

(discrete) states have different dynamic control laws: different formulas are used for the temperature in the room (as a function of the elapsed time) whether the heater is on (value 1) or off (value 0). If the heater is off at \( x \), only the natural loss of temperature is described, that is, if \( \beta \rightarrow -\infty \), then the temperature at time \( x + \beta \) goes to 0. But if the heater is on at \( x \) there is an extra factor raising the temperature; if \( \beta \rightarrow \infty \), then the temperature at \( x + \beta \) goes to the ratio \( \ell/k \); this is the highest temperature that we can achieve by heating the room – it forms an equilibrium between heating and cooling.

Notice that the newly created thermostats have their heater off, and have a temperature equal to 0 (which is the temperature of the environment).

We show two example calculations in `PTERM`. First, we assume that we have an arbitrary state \( s \) with value \( s.val@ 0 = 0 \) (heater is off). If we switch the heater on after \( x \) time units, and then read the temperature \( \beta \) units, then we get as observable outcome:

\[
\begin{align*}
\text{s.on@ x. temp@}\beta &= \text{s.on@ x. temp@}(0 + \beta) \\
&= \left(s.on@ x. temp@ 0 - \frac{\ell}{k}\right) \cdot e^{-k\beta} + \frac{\ell}{k} \\
&\text{since } s.on@ x. val@ 0 = 1
\end{align*}
\]
\[
= \left( s\,\text{temp}@x - \frac{\ell}{k} \right) \cdot e^{-k\beta} + \frac{\ell}{k}
\]
\[
= \left( s\,\text{temp}@0 + x - \frac{\ell}{k} \right) \cdot e^{-k\beta} + \frac{\ell}{k}
\]
\[
= \left( s\,\text{temp}@0 \cdot e^{-kx} - \frac{\ell}{k} \right) \cdot e^{-k\beta} + \frac{\ell}{k} \quad \text{since } s\,\text{val}@0 = 0
\]
\[
= \left( s\,\text{temp}@0 \right) \cdot e^{-k(x+\beta)} + \frac{\ell}{k} \cdot (1 - e^{-k\beta}).
\]

Secondly we present (the outcome of) a calculation using the formalism of Definition 3. We start from a newly created instance \texttt{new} of PTERM, and consider the evolution of its temperature if we switch the heater on at time 1 and switch it off 5 time-units later. This can be written as a function
\[
x \mapsto \text{new}\,\text{temp}@x(\text{on}@1, \text{off}@5) = \begin{cases} 
0 & \text{if } 0 \leq x < 1 \\
\frac{\ell}{k} \cdot (1 - e^{-(x-1)k}) & \text{if } 1 \leq x < 6 \\
\frac{\ell}{k} \cdot (1 - e^{-5k}) \cdot e^{-(x-6)k} & \text{if } 6 \leq x.
\end{cases}
\]

This function can be plotted (like in Fig. 6).

We call this PTERM a “passive” hybrid system because the heater will be switched on or off only as a result of an external action of a client. A more user-friendly system allows a client to set the goal temperature, whereupon the system “actively” regulates the temperature. We shall specify such a system (see Fig. 8) in which the temperature (after some time for adjustment) is kept in the interval \([z-1, z+1] \subseteq \mathbb{R}_{\geq 0}\) around the client’s choice temperature \(z\). Therefore, we assume that the highest possible temperature \(\ell/k\) in the room is bigger than 2, and that the client’s choice \(z\) lies in the open interval \((1, \ell/k - 1) \subseteq \mathbb{R}_{\geq 0}\).

The class specification ATHERM of such thermostats in Fig. 8 has three attributes \(\text{val}, \text{temp}, \text{goal}\) for respectively the value of the heater (\(0 = \text{off}, 1 = \text{on}\)), the actual temperature in the room, and the goal temperature as set by the client. (Initially this goal will be set to \(\ell/2k\), i.e. to half of the maximal temperature.) There is one procedure \texttt{set}, which allows a client to feed the desired goal temperature into the system. We shall use the abbreviations
\[
\sup_x \{ \beta \left| \right. (s\,\text{temp}@x) - \frac{\ell}{k} \cdot e^{-k\beta} + \frac{\ell}{k} (s\,\text{goal}@x) + 1 \} = \frac{1}{k} \ln \left( \frac{\ell - k (s\,\text{temp}@x)}{\ell - k (s\,\text{goal}@x) + 1} \right)
\]
for a state \(s\) with \(s\,\text{temp}@x < s\,\text{goal}@x + 1\), and
\[
\inf_x \{ \beta \left| \right. (s\,\text{temp}@x) \cdot e^{-k\beta} (s\,\text{goal}@x) - 1 \} = \frac{1}{k} \ln \left( \frac{s\,\text{temp}@x}{(s\,\text{goal}@x) - 1} \right)
\]
Fig. 8. A specification of “active thermostats”, keeping the temperature in the region ±1 around a goal temperature, which can be set by a user.

for a state \( s \) with \( s.\text{temp}@x > \text{goal}@x - 1 \) (where \( \ln \) is the natural logarithm \( \log_e \)). Thus, \( \lceil (s, x) \rceil \) is the time needed in state \( s \) at \( x \) to reach the maximum \( (s.\text{goal}@x) + 1 \) by heating, and \( \lfloor (s, x) \rfloor \) is the time needed to reach the minimum \( (s.\text{goal}@x) - 1 \).

The correctness of the ATERM specification involves two points: (1) the temperature will always go to the region ±1 around the goal temperature, and (2) once the temperature has reached this region it will stay there and become a periodic function. Proving (1) is not hard, so we shall concentrate on (2).

---

6 These abbreviations \( \lceil (s, x) \rceil \) and \( \lfloor (s, x) \rfloor \) will be used in the ATERM specification in Fig. 8 for so-called “time can proceed” (tcp) predicates – like in [31].
Proposition 4. Assume that \( s \) is a state in the active thermostat specification \( \text{ATHERM} \) in Fig. 8, which at time \( z \) has reached the intended region \( \pm 1 \) around the goal, i.e. which satisfies \( (s.\text{goal}(z) - 1 \leq s.\text{temp}(z) \leq (s.\text{goal}(z)) + 1 \). The temperature will then remain in this region: it will oscillate between \( s.\text{goal}(z) - 1 \) and \( s.\text{goal}(z) + 1 \) with periodicity \( |(s.\text{goal}(z)) + (s.\text{goal}(z))| \), where \( |z| \) (resp. \( \downarrow z \)) is the time that is required for the temperature to rise from \( z - 1 \) to \( z + 1 \), (resp. to fall from \( z + 1 \) to \( z - 1 \)). Further, the heater will be switched on and off with the same periodicity.

The periodicity mentioned in this result is clearly visible in the simulation in Fig. 9.

Proof. We shall prove that for \( s \) and \( z \) as above,

\[
\begin{align*}
\text{s.temp}(z) &= \text{s.temp}(x + \uparrow (s.\text{goal}(z)) + \downarrow (s.\text{goal}(z))), \\
\text{s.val}(z) &= \text{s.val}(x + \uparrow (s.\text{goal}(z)) + \downarrow (s.\text{goal}(z))).
\end{align*}
\]

There are two cases, depending on whether the heater is off or on at \( z \), i.e. on whether \( \text{s.val}(z) = 0 \) or \( \text{s.val}(z) = 1 \). We shall do the first case, and leave the (similar) second case to the reader.

So let \( \text{s.val}(z) = 0 \). We shall compute the value \( \text{s.val}(x + \uparrow (s.\text{goal}(z)) + \downarrow (s.\text{goal}(z))) \) and the temperature \( \text{s.temp}(x + \uparrow (s.\text{goal}(z)) + \downarrow (s.\text{goal}(z))) \) in three steps.

1. For \( \beta < \|(s, z)\| \leq \|(s.\text{goal}(z))\) the (falling) temperature is given by \( \text{s.temp}(x + \beta) = (\text{s.temp}(z) \cdot e^{-k\beta}) \). And at the border time \( \|(s, z)\) we get

\[
\begin{align*}
\text{s.val}(x + \|(s, z)\)) &= 1, \\
\text{s.temp}(x + \|(s, z)\)) &= s.\text{goal}(z) - 1.
\end{align*}
\]
2. The time from $z + \downarrow(s, z)$ to the next border can be computed as

$$
\uparrow(s, z + \downarrow(s, z)) = \frac{1}{k} \ln \left( \frac{\ell - k(s \text{temp}(z + \downarrow(s, z)))}{\ell - k(s \text{goal}(z + \downarrow(s, z)) + 1)} \right)
$$

$$
= \frac{1}{k} \ln \left( \frac{\ell - k(s \text{goal}(z - 1))}{\ell - k(s \text{goal}(z + 1))} \right)
$$

$$
= \uparrow(s \text{goal}(z))
$$

so that

$$
s \text{val}(z + \downarrow(s, z) + \uparrow(s \text{goal}(z))) = 0
$$

$$
s \text{temp}(z + \downarrow(s, z) + \uparrow(s \text{goal}(z))) = s \text{goal}(z) + 1.
$$

3. One further border occurs at

$$
\downarrow(s, z + \downarrow(s, z) + \uparrow(s \text{goal}(z)))
$$

$$
= \frac{1}{k} \ln \left( \frac{s \text{temp}(z + \downarrow(s, z)) + \uparrow(s \text{goal}(z))}{s \text{goal}(z + \downarrow(s, z) + \uparrow(s \text{goal}(z)) - 1)} \right)
$$

$$
= \frac{1}{k} \ln \left( \frac{s \text{goal}(z + 1)}{s \text{goal}(z - 1)} \right)
$$

$$
= \downarrow(s \text{goal}(z))
$$

And so for $\beta < \downarrow(s \text{goal}(z))$ we get

$$
s \text{temp}(z + \downarrow(s, z) + \uparrow(s \text{goal}(z) + \beta)) = (s \text{goal}(z) + 1) \cdot e^{-k\beta}
$$

$$
s \text{val}(z + \downarrow(s, z) + \uparrow(s \text{goal}(z) + \beta)) = 0.
$$

Especially for $\beta = \downarrow(s \text{goal}(z)) - \downarrow(s, z)$ this gives us the required outcome:

$$
s \text{temp}(z + \downarrow(s, z) + \uparrow(s \text{goal}(z)))
$$

$$
= s \text{temp}(z + \downarrow(s, z) + \uparrow(s \text{goal}(z)) + \downarrow(s \text{goal}(z)) - \downarrow(s, z))
$$

$$
= (s \text{goal}(z) + 1) \cdot e^{-k(\downarrow(s \text{goal}(z)) - \downarrow(s, z))}
$$

$$
= (s \text{goal}(z) + 1) \cdot \frac{s \text{goal}(z - 1)}{s \text{goal}(z) + 1} \cdot \frac{s \text{temp}(z)}{s \text{goal}(z) - 1}
$$

$$
= s \text{temp}(z)
$$

$$
s \text{val}(z + \downarrow(s, z) + \uparrow(s \text{goal}(z)))
$$

$$
= s \text{val}(z + \downarrow(s, z) + \uparrow(s \text{goal}(z)) + \downarrow(s \text{goal}(z)) - \downarrow(s, z))
$$

$$
= 0
$$

$$
= s \text{val}(z). \quad \Box
$$
4. Hybrid models and their homomorphisms: terminality as minimal realization

In this section we shall study the semantics of hybrid class specifications, as introduced in the previous section. We shall define notions of “hybrid model” and of “homomorphism of hybrid models”. In brief,

hybrid model = coalgebra + monoid action,

where the coalgebra handles the discrete dynamics (of external operations) and the monoid action captures the (discrete-time or real-time) internal dynamics. The notion of homomorphism of hybrid models determines “terminal” models, namely as those models to which there is precisely one homomorphism out of an arbitrary model. Terminal models are determined up to isomorphism. The larger part of this section will be devoted to describing the terminal models of the specification examples (timers, chemical processes and thermostats) from the previous section. It will make clear that a terminal model of a hybrid class specification is a “minimal realization” with a minimal state space realizing the intended behaviour. The next section contains a more systematic investigation of terminality.

4.1. Coalgebras plus monoid actions

We recall that a monoid is a 3-tuple \((M; 0; +)\) consisting of a set \(M\) with a distinguished “zero” element \(0 \in M\), and with a binary “addition” operation \(+: M \times M \rightarrow M\) which is associative, \(x + (\beta + \gamma) = (x + \beta) + \gamma\), and has \(0 \in M\) as neutral element, \(x + 0 = x = 0 + x\). Often we write \(M\) for the 3-tuple \((M; 0; +)\) when the operations \(0; +\) are understood from the context. We shall mainly use the (commutative) monoids \((\mathbb{N}; 0; +)\) of discrete time and \((\mathbb{R}_{\geq 0}; 0; +)\) of real time, where \(\mathbb{R}_{\geq 0} = \{x \in \mathbb{R} \mid x \geq 0\}\) is the set of positive reals. Actually, we shall also use are linearly ordered monoids (i.e. monoids in the category of linear posets).

Let \((M; 0; +)\) be an arbitrary monoid. An action of this monoid on a set \(U\) consists of a function \(\mu: U \times M \rightarrow U\) satisfying the following two requirements:

\[
\mu(x, 0) = x \quad \text{and} \quad \mu(x, x + \beta) = \mu(\mu(x, x), \beta).
\]  

(6)

In the examples below, the set \(U\) will be the set of states of a certain dynamical system, and the function \(\mu: U \times M \rightarrow U\) may be seen as giving for a state \(x \in U\) and amount of time \(\alpha \in M\) a new state \(\mu(x, \alpha) \in U\) obtained by letting the system run for \(\alpha\) units of time starting in state \(x\). The above two conditions express a certain linearity\(^7\) of this action: \(\mu(x, 0) = x\) says that letting the machine run for 0 units of time does not change the state, and \(\mu(x, x + \beta) = \mu(\mu(x, x), \beta)\) expresses that the effect of letting the machine run \(x + \beta\) units of time is the same as first letting it run \(x\) units of time, and then \(\beta\) units of time.

\(^7\)The first equation in (6) is sometimes referred to as consistency and the second as composition property, see e.g. [26, Definition (1.1)]. These monoid actions appear as “evolution functions” in [31, Definition 2.1] (and in many other places).
There are many more (non-temporal) instances of monoid-actions.

- modules and vector spaces are monoid-actions, given by the application \((a, v) \mapsto a \cdot v\) of a scalar \(a\) to a vector \(v\); it satisfies \(1 \cdot v = v\) and \((ab) \cdot v = a \cdot (b \cdot v)\) and is thus a monoid action with respect to the (multiplicative) monoid structure on the scalars.

- For a deterministic automaton with alphabet \(A\) and transition function \(\delta : X \times A \rightarrow X\) there is (by induction) an extended transition function \(\delta^* : X \times A^* \rightarrow X\) given by

\[
\delta^*(x, \lambda) = x \quad \text{and} \quad \delta^*(x, a \cdot \sigma) = \delta^*(\delta(x, a), \sigma).
\]

This \(\delta^*\) forms a monoid action with respect to the (free) monoid \(A^*\) of words (since one easily proves \(\delta^*(x, \sigma \cdot \tau) = \delta^*(\delta^*(x, \sigma), \tau)\), by induction on \(\sigma\).

- A standard result is that unique solutions to differential equations give rise to monoid actions, see e.g. [17, 8.7]. In this context monoid actions are sometimes called flows.

- The first projection function \(\pi : U \times M \rightarrow U\) is a trivial \(M\)-action on an arbitrary set \(U\). It captures a situation where a state space \(U\) is not influenced by time. Hence, via this trivial action, we can regard ordinary, untimed coalgebraic models (as in Section 2) as hybrid models.

- One can alternatively impose an \(M\)-action on an arbitrary state space \(U\) by making time part of the state space: take the extended space \(U' = U \times M\) with action \(U' \times M \rightarrow U'\) given by \(((x, \gamma), a) \mapsto (x, \gamma + a)\).

We mention a concrete paradigmatic (temporal) example of a monoid action. It involves the “minus” function \(\preceq\) (also called truncated subtraction) defined as follows:

\[
x \preceq y = \max(0, x - y) = \begin{cases} x - y & \text{if } x \geq y, \\ 0 & \text{otherwise.} \end{cases}
\]

This monus will be used as a function \(\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}\), and also as a function \(\mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}\). Let \(U = \{s \in \mathbb{R}_{\geq 0} \mid s \leq 10\}\) be the set of states of a (real-time) timer, where the state \(s \in U\) indicates that the timer will give a signal in \(s\) units of time.

There is then an action \(\mu : U \times \mathbb{R}_{\geq 0} \rightarrow U\) given by \(\mu(s, x) = s - x\). Thus, if we have a state \(5.7 \in U\) indicating that a signal will be given in 5.7 units of time, then the state \(\mu(5.7, 3)\) obtained by letting the timer run for 3 units of time, is 2.7 \(\in U\). It is not hard to see that \(\mu\) satisfies the two equations of a monoid action.

We now come to the question: what is a model of a hybrid coalgebraic specification (as used in the previous section)? Such a model will have to consist of a state space \(U\) with functions acting on \(U\) that interpret the methods in the specification. But some additional structure is needed to interpret temporal expressions \(s \cdot \text{meth}@x\) involving a time variable \(x\). This extra structure is a monoid action \(\mu : U \times M \rightarrow U\), so that we can interpret

\[
s \cdot \text{meth}@x \quad \text{as} \quad \text{meth}(\mu(s, x)).
\]

Note that this interpretation precisely captures the meaning (1) of the expression \(s \cdot \text{meth}@x\), namely as: let the state \(s\) evolve for \(x\) units of time to \(\mu(s, x)\), and apply the method \text{meth} to the resulting state \(\mu(s, x)\). Hence, the monoid action describes the (discrete or continuous) evolution of states under influence of time, and the coalgebraic procedures describe the state changes resulting from external actions (by clients).
Monoid actions are an explicit part of hybrid models, but exist only implicitly in hybrid specifications, in the notation s.meth@x. Notice that the first monoid equation in (6) yields s.meth@0 = s.meth.

We recapitulate the above discussion.

**Definition 5.** Consider a (discrete- or real-time) hybrid class specification \( \mathcal{S} \) as in the previous section, with attributes \( X \to A_1, \ldots, X \to A_n \) and procedures \( X \times B_1 \to X, \ldots, X \times B_m \to X \).

A **hybrid model** of this specification \( \mathcal{S} \) consists of four parts:

(a) a “state space” or “carrier set” \( U \), serving as interpretation \( U = [X] \) of the unknown type \( X \) in the specification \( \mathcal{S} \); elements of \( U \) will be called states;

(b) a monoid action \( \mu : U \times M \to U \), where \( M \) is the monoid of discrete or real time (in accordance with whether \( \mathcal{S} \) is a discrete or real time class specification);

(c) functions \( U \to A, U \times B \to U \), where \( A = A_1 \times \cdots \times A_n \) is the product of the sets of attribute values and \( B = B_1 + \cdots + B_m \) is the coproduct (disjoint union) of the procedure parameter sets, giving combined interpretations \( U \to A_i \) of the attributes and of the procedures \( U \times B_j \to U \), in such a way that the assertions of the specification \( \mathcal{S} \) are satisfied;

(d) an initial state \( u_0 \in U \) satisfying the creation conditions in the specification \( \mathcal{S} \).

We notice that the interpretations of the attributes and procedures together with the monoid action form a **coalgebra** \( U \to A \times U \times U \) on the state space \( U \).

Having seen the notion of a hybrid model, we proceed to describe the notion of a “homomorphism” of hybrid models, preserving the relevant structure. One may expect such a homomorphism to be at the same time a homomorphism of coalgebras and of monoid actions\(^8\) (i.e. to be a homomorphism of coalgebras of the functor \( X \mapsto A \times X \times XM \)). But this turns out not to be the right definition since it does not yield the intended minimal realizations as terminal models. The following definition does work, as we shall show in the examples in the remainder of this section.

**Definition 6.** Consider two hybrid models \( \mathcal{U} = \langle U, U \times B \text{ \text{ at } } A, U \times M \text{ \text{ at } } U, u_0 \in U \rangle \) and \( \mathcal{V} = \langle V, V \times B \text{ \text{ at } } A, V \times M \text{ \text{ at } } V, v_0 \in V \rangle \) of the same class specification. A **homomorphism** \( f : \mathcal{U} \to \mathcal{V} \) between these models consists of a function \( f : U \to V \) between the underlying state spaces making the following three diagrams commute:

\[
\begin{array}{ccc}
U \times M & \xrightarrow{\mu_U} & U \\
\downarrow \quad \quad \downarrow & & \quad \quad \downarrow \\
V \times M & \xrightarrow{\mu_V} & V \\
\end{array}
\]

\[
\begin{array}{ccc}
U & \xrightarrow{\text{at}_U} & A \\
\downarrow \quad \quad \downarrow & & \quad \quad \downarrow \\
V & \xrightarrow{\text{at}_V} & A \\
\end{array}
\]

\[
f \times \text{id}
\]

\[^8\text{A homomorphism of monoid actions is usually called an equivariant mapping, see e.g. [5, 3.2.1, 3.2.2].}\]
That is, \( f \) satisfies for \( x \in U, \ z \in M \) and \( b \in B \):

\[
\begin{align*}
  f(x) \cdot \text{at}_\gamma @ z &= x \cdot \text{at}_\gamma @ z \\
  f(x) \cdot \text{proc}_\gamma (b) @ z &= f(x \cdot \text{proc}_\gamma (b) @ z)
\end{align*}
\]

and \( f(u_0) = v_0 \).

In this setting we call a hybrid model \( \mathcal{V} \) terminal if for an arbitrary hybrid model \( \mathcal{U} \) (of the same specification) there is precisely one homomorphism \( \mathcal{U} \to \mathcal{V} \).

In the notion of homomorphism used in the definition, the internal time steps are not preserved directly, but only indirectly via their observable effect. As we shall see later in Corollary 15(ii), the time-steps are preserved up to bisimilarity. This gives a form of hiding. The three operations \( \text{at} : U \to A \), \( \text{proc} : U \times B \to U \) and \( \mu : U \times M \to U \) can be combined into a coalgebra

\[
U \to A^M \times U^{B \times M}
\]

given by \( x \mapsto \langle \lambda x \in M. x \cdot \text{at}@z, \lambda (b, \beta) \in B \times M. x \cdot \text{proc}(b)@\beta \rangle 
\).

Commutativity of the first two diagrams in the previous definition means that \( f : U \to V \) is a homomorphism between such induced coalgebras (of the functor \( X \mapsto A^M \times X^{B \times M} \)). Hence technically, this hiding of time-steps happens by considering homomorphisms with respect to a derived (more abstract) functor \( X \mapsto A^M \times X^{B \times M} \), rather than to the functor \( X \mapsto A \times X^B \times X^M \) describing the structure of the model.

4.2. Timers

In models of hybrid class specifications the elements of the carrier set are internal (invisible) states which are used to realize the specified behaviour of objects belonging to a class. A useful heuristics for determining the terminal model is to identify the minimal set of states that realizes the required behaviour. This will be illustrated in several examples.
We start with the discrete time flip-flop DTFF from Fig. 2. Such flip-flops can be switched on by a user, and will then automatically switch off after 10 time-units. In an implementation one should somehow keep track of the amount of time before the flip-flop must switch off. The easiest way to do this is to take natural numbers \( s \leq 10 \) as internal states, where the state \( s \in \mathbb{N} \) captures the state of the flip-flop in which it will switch off in \( s \) time-units. This explains the maximum 10. We thus take \( U = [0, 10] = \{0, 1, \ldots, 10\} \subseteq \mathbb{N} \) as underlying state space. This, by the way, seems yield the minimal set of states needed for such a timer, so we expect this \( U \) to be the terminal model. But first we have to see what the operations on \( U \) are. The monoid action \( \mu : [0, 10] \times \mathbb{N} \to [0, 10] \) should capture the influence of (discrete) time \( \mathbb{N} \) on the state space. Waiting 5 time-units starting in state \( s \in [0, 10] \) means that the time to switch off is decreased by 5, if possible, yielding \( s = s - 5 \) as new state. Hence, in general, we define

\[
\mu(s, \alpha) = s - \alpha.
\]

The interpretations of the (coalgebraic) methods val, on are then

\[
\begin{align*}
[0, 10] &\xrightarrow{\text{val}} \{0, 1\} \\
\quad s &\mapsto \begin{cases} 0 & \text{if } s = 0 \\ 1 & \text{otherwise} \end{cases} \\
[0, 10] &\xrightarrow{\text{on}} [0, 10] \\
\quad s &\mapsto 10.
\end{align*}
\]

And as initial state in this model we take \( 0 \in [0, 10] \). We shall verify in detail that the assertions of the DTFF specification in Fig. 2 hold in this model.

1. The first assertion \( s\.\text{val}@x = 0 \vdash s\.\text{val}@\alpha + \beta = 0 \) holds, since for \( s \in [0, 10] \),

\[
s\.\text{val}@x = 0 \Rightarrow \mu(s, x) = s - x = 0 \Rightarrow x \geq s \Rightarrow x + \beta \geq s \Rightarrow s\.\text{val}@\alpha + \beta = 0.
\]

2. The second assertion \( \beta \geq 0 \vdash s\.\text{val}@\alpha = 0 \) obviously holds, since for \( s \in [0, 10] \) and \( \beta \geq 0 \) one has \( s - \beta = 0 \).

3. Also the last assertion \( \beta < 10 \vdash s\.\text{on}@x\.\text{val}@\beta = 1 \) holds since

\[
\beta < 10 \Rightarrow 10 - \beta > 0 \Rightarrow (s\.\text{on}@x) - \beta > 0 \Rightarrow s\.\text{on}@x\.\text{val}@\beta = 1.
\]

4. Finally, the initial state \( 0 \in [0, 10] \) satisfies the creation condition, because

\[
0\.\text{val}@0 = 0, \quad \text{since} \quad \mu(0, 0) = 0 - 0 = 0.
\]

There are many other models of this DTFF specification besides \( [0, 10] = \{0, 1, \ldots, 10\} \subseteq \mathbb{N} \). One can also take the closed intervals \( [0, 10] \subseteq \mathbb{Q} \) and \( [0, 10] \subseteq \mathbb{R} \) of (positive) rational and real numbers below 10. The definitions of the action and methods are as above. But in these models there are “too many” states. However, for a client who can only use the specified methods, these differences of internal state structure are not noticeable.

Another model can be defined with the subset \( V = \{(n, m) | n \geq m\} \subseteq \mathbb{N} \times \mathbb{N} \) as state space. We think of state \((n, m)\) as consisting of the current time \( n \) together with the time \( m \leq n \) at which the flip-flop was last switched on by a client. This leads to
As initial state in this model we take $(10, 0) \in V$. (In fact, any pair $(n_0, m_0) \in \mathbb{N} \times \mathbb{N}$ with $n_0 > m_0 + 10$ can be taken as initial state: all these states are "bisimilar", i.e. indistinguishable by a client on the outside, as will be explained later in Section 5.2.) We leave it to the reader to verify the validity of the DTFF assertions in this model on $V$.

The minimality of our first DTFF model with state space $[0, 10]_\mathbb{N}$ can be expressed mathematically using terminality. This will be shown next.

**Proposition 7.** (1) The terminal hybrid model of the discrete-time flip-flop (DTFF) specification in Fig. 2 is the model with state space $[0, 10]_\mathbb{N}$ described above.

(2) The terminal model of the real-time flip-flop specification (RTFF), with additional assertion (3), has the interval $[0, 10]_\mathbb{R}$ as state space (with operations as on $[0, 10]_\mathbb{N}$ in (1)).

**Proof.** (1) We have to show that for an arbitrary model, say with carrier set $V$, action $v: V \times \mathbb{N} \rightarrow V$, methods $\text{val}: V \rightarrow \{0, 1\}$, $\text{on}: V \rightarrow V$ and initial state $v_0 \in V$, there is a unique homomorphism of hybrid models $f: V \rightarrow [0, 10]_\mathbb{N}$. This function is given by

$$f(x) = \inf \{ \beta \in [0, 10]_\mathbb{N} | x.\text{val}@\beta = 0 \}.$$ 

Thus $f$ maps a state $x \in V$ to the first time-unit where the value of state $x$ is 0. We show that $f$ is a homomorphism.

(a) Commutation with $\text{val}$:

$$f(x).\text{val}@x = 0 \iff x.\text{val}@x = f(x) \Downarrow x = 0$$

$$\iff x \geq f(x) = \inf \{ \beta \in [0, 10]_\mathbb{N} | x.\text{val}@\beta = 0 \}$$

$$\iff x.\text{val}@x = 0.$$

The direction ($\Rightarrow$) of the last step is easy, by definition of infimum. For ($\Rightarrow$), assume $x \geq f(x)$, say $x = f(x) + \beta$. Since $x.\text{val}@f(x) = 0$ because $f(x)$ is the first time that the value in state $x$ is 0, we get $x.\text{val}@x = x.\text{val}@f(x) + \beta = 0$ by the monotony assertion in the DTFF specification.

(b) Commutation with $\text{on}$:

$$f(x.\text{on}@x)$$

$$= \inf \{ \beta \in [0, 10]_\mathbb{N} | x.\text{on}@x.\text{val}@\beta = 0 \}$$

$$= \inf \{ \beta \in [0, 10]_\mathbb{N} | \mu(x.\text{on}@x, \beta) = (x.\text{on}@x) \Downarrow \beta = 10 \Downarrow \beta = 0 \}$$
\[ = \inf \{ \beta \in [0, 10]_\mathbb{N} \mid \beta \geq 10 \} \]
\[ = 10 \]
\[ = f(x).on@x. \]

(c) Preservation of the initial state:

\[ f(v_0) = \inf \{ \beta \in [0, 10]_\mathbb{N} \mid v_0.val@\beta = 0 \} = \inf \{ \beta \in [0, 10]_\mathbb{N} \mid \beta \geq 0 \} = 0. \]

Finally, we have to show that \( f \) is unique with these properties. If also \( g: V \to [0, 10]_\mathbb{R} \)
satisfies \( g(x).val@x = x.val@x, g(x).on@x = g(x.on@x) \) and \( g(v_0) = 0 \), then \( g(x) = f(x) \) since

- \( g(x) \leq f(x) \) because \( g(x) \) is a lower bound of the set \( \{ \beta \in [0, 10]_\mathbb{N} \mid x.val@\beta = 0 \} \):

\[ x.val@\beta = 0 \Rightarrow g(x).val@\beta = 0 \Rightarrow \mu(g(x), \beta) = g(x) \triangleright \beta = 0 \Rightarrow g(x) \leq \beta. \]

- \( g(x) \geq f(x) \) because \( x.val@g(x) = 0 \). This follows since \( g(x).val@g(x) = 0 \) since \( \mu(g(x), g(x)) = g(x) \blacktriangleleft g(x) = 0. \)

(2) We now turn to the real-time version RTFF with the additional “denseness” assertion \( s.val@x = 1 \Leftrightarrow \exists \beta > 0 \ s.val@(x + \beta) = 1 \). The claim is that the terminal model satisfying this RTFF specification is the closed interval \([0, 10]_\mathbb{R} \subseteq \mathbb{R} \) of positive reals less than or equal to 10. The idea is (again) that a state \( s \in [0, 10]_\mathbb{R} \) represents the state of the flip-flop in which the value will be 0 in \( s \) units of real-time. The (real-time) action \( \mu: [0, 10]_\mathbb{R} \times \mathbb{R}_{\geq 0} \to [0, 10]_\mathbb{R} \) is \( (s, x) \mapsto s \blacktriangleleft x \), as before. And also the method interpretations \( \text{val}: [0, 10]_\mathbb{R} \to \{0, 1\} \) and \( \text{on}: [0, 10]_\mathbb{R} \to [0, 10]_\mathbb{R} \) are as in (i). Also, \( 0 \in [0, 10]_\mathbb{R} \) is the initial state. We check the validity of the extra “denseness” assertion. If for some state \( s \in [0, 10]_\mathbb{R} \) and time \( x \in \mathbb{R}_{\geq 0} \) we have \( s.val@x = 1 \), then \( s \blacktriangleleft x \geq 0 \), so that \( x < s \). But then we can find a \( \beta > 0 \) with \( x + \beta < s \) because \( \mathbb{R}_{\geq 0} \) is dense. This means that \( s.val@(x + \beta) = 1 \).

In order to show terminality of the model \([0, 10]_\mathbb{R} \), assume another model consisting of a carrier set \( V \), with action \( v: V \times \mathbb{R}_{\geq 0} \to V \), method interpretations \( \text{val}: V \to \{0, 1\} \), \( \text{on}: V \to V \) and initial state \( v_0 \in V \). Then we can define a function \( f: V \to [0, 10]_\mathbb{R} \) by \( f(x) = \inf \{ \beta \in [0, 10]_\mathbb{R} \mid x.val@\beta = 0 \} \). We show that \( f(x).val@x = x.val@x \).

\[ f(x).val@x = 0 \Leftrightarrow f(x) \blacktriangleleft x = 0 \Leftrightarrow x \geq f(x) \Leftrightarrow x.val@x = 0. \]

The marked implication \((\Rightarrow)\) is easy by definition of infimum. For \((\Leftarrow)\) we use that \( x.val@f(x) = 0 \). Suppose not, i.e. \( x.val@f(x) = 1 \). Then we can find a \( \gamma > 0 \) with \( x.val@(f(x) + \gamma) = 1 \), by the additional assertion mentioned above. But \( f(x) + \gamma \) is a lower bound for \( \beta \in [0, 10]_\mathbb{R} \) with \( x.val@\beta = 0 \). Hence \( f(x) + \gamma \leq f(x) \), because \( f(x) \) is the greatest lower bound (infimum). But this is impossible.

The remaining details that \( f \) is the unique homomorphism \( V \to [0, 10]_\mathbb{R} \) are as in (i), and are left to the reader. □

We turn to the watch-dog specification WD(\( \text{dur}_1: \mathbb{R}_{\geq 0}, \text{dur}_2: \mathbb{R}_{\geq 0} \)) in Fig. 3. An implementation should keep track of the remaining time for each process before alarm
must be raised. The most economical way to do this seems to require as state space the cartesian product $[0, \text{dur}_1]_R \times [0, \text{dur}_2]_R \subseteq R \times R$, so that a pair $(x_1, x_2) \in Z$ represents that there is $x_1$ time left for process 1 to confirm, and $x_2$ time for process 2. But closer scrutiny reveals that if one of the timers reaches 0, then alarms is raised, and the other timer is also set to 0. Hence it suffices to take a subset $^9 Z \subseteq [0, \text{dur}_1]_R \times [0, \text{dur}_2]_R$ consisting of those pairs $(x_1, x_2)$ with $x_1 = 0$ if and only if $x_2 = 0$. This turns out to be the terminal model. The details of the proof are left to the interested reader.

**Proposition 8.** The terminal model of the watch-dog specification $\text{WD}($dur$_1$: $R > 0$, dur$_2$: $R > 0)$ from Fig. 3 has as state space

$$Z = \{(x_1, x_2) \in R \times R \mid 0 \leq x_1 \leq \text{dur}_1 \text{ and } 0 \leq x_2 \leq \text{dur}_2 \text{ and } x_1 = 0 \Leftrightarrow x_2 = 0\}$$

with operations

$$Z \times R_{\geq 0} \xrightarrow{\mu} Z$$

$$(x_1, x_2, z) \mapsto \begin{cases} (x_1 - z, x_2 - z) & \text{if } z < \min\{x_1, x_2\} \\ (0, 0) & \text{otherwise} \end{cases}$$

$$Z \xrightarrow{\text{conf}_1} Z$$

$$(x_1, x_2) \mapsto \begin{cases} (\text{dur}_1, x_2) & \text{if } \min\{x_1, x_2\} > 0 \\ (0, 0) & \text{otherwise} \end{cases}$$

$$Z \xrightarrow{\text{conf}_2} Z$$

$$(x_1, x_2) \mapsto \begin{cases} (x_1, \text{dur}_2) & \text{if } \min\{x_1, x_2\} > 0 \\ (0, 0) & \text{otherwise} \end{cases}$$

$$Z \xrightarrow{\text{alarm}} \{0, 1\}$$

$$(x_1, x_2) \mapsto \begin{cases} 1 & \text{if } \min\{x_1, x_2\} = 0 \\ 0 & \text{otherwise} \end{cases}$$

$$Z \xrightarrow{\text{time}_1} [0, \text{dur}_1]$$

$$(x_1, x_2) \mapsto x_1$$

$$Z \xrightarrow{\text{time}_2} [0, \text{dur}_2]$$

$$(x_1, x_2) \mapsto x_2$$

$$Z \xrightarrow{\text{time}_R} R$$

$$(x_1, x_2) \mapsto \min\{x_1, x_2\}$$

$$Z \xrightarrow{\text{reset}} Z$$

$$(x_1, x_2) \mapsto (\text{dur}_1, \text{dur}_2)$$

and with $$(\text{dur}_1, \text{dur}_2) \in Z$$ as initial state. □

---

$^9$ This subset can be described as a “smash” or “wedge” product $[0, \text{dur}_1]_R \otimes [0, \text{dur}_2]_R$ of pointed spaces.
4.3. Chemical processes

We consider the hybrid class specifications \( \text{REACT}_A \) and \( \text{REACT}_{A,B} \) from Section 3.2, describing reaction containers with one and two chemical substances. In the \( \text{REACT}_A \) case an implementation has to keep track of the amount of the chemical substance \( A \). This is done most economically by taking as state space the set \( \mathbb{R}_{>0} \) of positive reals, elements of which represent this amount of \( A \). And in the \( \text{REACT}_{A,B} \) case both the amounts of \( A \) and of \( B \) have to be tracked. This leads to a minimal state space \( \mathbb{R}_{>0} \times \mathbb{R}_{>0} \).

**Proposition 9.** (1) The terminal model of the \( \text{REACT}_A(k: \mathbb{R}_{>0}) \) hybrid class specification in Fig. 4 describing the decay of one substance \( A \) has the set \( \mathbb{R}_{>0} \) as state space. An element \( x \in \mathbb{R}_{>0} \) in this state space represents the current amount \( x \) of \( A \) in the reaction container.

(2) Similarly, the terminal model of the \( \text{REACT}_{A,B}(k: \mathbb{R}_{>0}, \ell: \mathbb{R}_{>0}) \) hybrid class specification in Fig. 5 involving two substances \( A \) and \( B \) has the set \( \mathbb{R}_{>0} \times \mathbb{R}_{>0} \) as state space, where a state \((x_A, x_B) \in \mathbb{R}_{>0} \times \mathbb{R}_{>0}\) captures the current amounts \( x_A \) of \( A \) and \( x_B \) of \( B \).

**Proof.**

(1) The action \( \mu: \mathbb{R}_{>0} \times \mathbb{R}_{>0} \rightarrow \mathbb{R}_{>0} \) sends a pair \((x, z)\) consisting of the present amount \( x \) of \( A \) and the time \( z \), to the amount \( \mu(x, z) = x \cdot e^{-kz} \) after \( z \) time-units. This is a monoid action, since the action equations (6) hold: \( \mu(x, 0) = x \cdot e^0 = x \cdot 1 = x \), and \( \mu(\mu(x, z), \beta) = \mu(x, z) \cdot e^{-k\beta} = (x \cdot e^{-kz}) \cdot e^{-k\beta} = x \cdot e^{-k(z+\beta)} = \mu(x, z + \beta) \).

The interpretations of the methods amount, add and clear are then simply

\[
\begin{align*}
R_{>0} \times R_{>0} & \xrightarrow{\text{add}} R_{>0} \quad R_{>0} & \xrightarrow{\text{amount}} R_{>0} \quad R_{>0} & \xrightarrow{\text{clear}} R_{>0} \\
(x, y) & \mapsto x + y \quad x & \mapsto x \quad x & \mapsto 0.
\end{align*}
\]

As initial state we have to take \( 0 \in \mathbb{R}_{>0} \). This is the terminal model, since for an arbitrary model with state space \( V \), with action \( \nu: V \times \mathbb{R}_{>0} \rightarrow V \), attribute amount: \( V \rightarrow \mathbb{R}_{>0} \), procedures add: \( V \times \mathbb{R}_{>0} \rightarrow V \), clear: \( V \rightarrow V \) and with initial state \( v_0 \in V \), we get a unique homomorphism \( f: V \rightarrow \mathbb{R}_{>0} \), namely \( f(x) = x \cdot \text{amount} @ 0 \). Then

\[
f(x) \cdot \text{amount} @ z = \mu(f(x), z) \\
= f(x) \cdot e^{-kz} \\
= (x \cdot \text{amount} @ 0) \cdot e^{-kz} \\
= x \cdot \text{amount} @ (0 + z) \\
= x \cdot \text{amount} @ z
\]

\[
f(x) \cdot \text{add}(a) @ z = \mu(f(x), z) + a \\
= x \cdot \text{amount} @ z + a \\
= x \cdot \text{add}(a) @ z \cdot \text{amount} @ 0 \\
= f(x) \cdot \text{add}(a) @ z
\]
\[ f(x).\text{clear}@z = 0 \]
\[ = x.\text{clear}@z.\text{amount}@0 \]
\[ = f(x.\text{clear}@z). \]

And also \( f(v_0) = v_0.\text{amount}@0 = 0 \). Uniqueness is obvious.

(2) For the terminal model \( \mathbb{R}_{>0} \times \mathbb{R}_{>0} \) of the REACT\(_{A,B}\) class specification we only define the monoid action, and leave further details to the reader. This action \( \mu: (\mathbb{R}_{>0} \times \mathbb{R}_{>0}) \times \mathbb{R}_{>0} \rightarrow (\mathbb{R}_{>0} \times \mathbb{R}_{>0}) \) is defined by

\[ \mu((x_A, x_B), z) = (\mu_A((x_A, x_B), z), x_A + x_B - \mu_A((x_A, x_B), z)) \]

where

\[ \mu_A((x_A, x_B), z) = \frac{1}{k + \ell} ((kx_A - \ell x_B) \cdot e^{-(k+\ell)x} + \ell(x_A + x_B)) \]

Notice that

\[ \mu_A((x_A, x_B), 0) = (1/(k + \ell))((\ell(x_A + x_B) + kx_A - \ell x_B) = (1/(k + \ell))(\ell + k)x_A) = x_A, \]

and thus

\[ \mu((x_A, x_B), 0) = (x_A, x_A + x_B - x_A) = (x_A, x_B). \]

The other action-equation \( \mu((x_A, x_B), z), \beta) = \mu((x_A, x_B), z + \beta) \) is left to the interested reader. \( \square \)

4.4. Thermostats

We only sketch the terminal models of the passive and active thermostat specifications.

**Proposition 10.** (1) The terminal model of the passive thermostat \( \text{PTERM}(k; \mathbb{R}_{>0}, \ell; \mathbb{R}_{>0}) \) hybrid class specification in Fig. 7 has as state space the set

\[ \{0, 1\} \times \mathbb{R}_{>0} \]

consisting of elements \((z, x)\) where \( z \in \{0, 1\} \) tells us whether the heater is currently on or off, and \( x \in \mathbb{R}_{>0} \) tells the current temperature in the room.

(2) The terminal model of the active thermostat \( \text{ATHERM}(k; \mathbb{R}_{>0}, \ell; \mathbb{R}_{>2k}) \) hybrid class specification in Fig. 8 has as underlying state the set

\[ U = \{(x, y, z) \in \{0, 1\} \times [0, \frac{\ell}{k}] \times (1, \frac{\ell}{k} - 1) | y < z - 1\} \Rightarrow x = 1 \] and
\[ y > z + 1 \Rightarrow x = 0 \]

describing whether the heater is on or off (in its first component), the current temperature in the room (in its second component) and the current goal temperature (in its third component).

**Proof.** (1) The action \( \mu: (\{0, 1\} \times \mathbb{R}_{>0}) \times \mathbb{R}_{>0} \rightarrow (\{0, 1\} \times \mathbb{R}_{>0}) \) is given by

\[ \mu((0, x), z) = (0, x \cdot e^{-kx}) \] and
\[ \mu((1, x), z) = \left(1, \left(x - \frac{\ell}{k}\right) \cdot e^{-kx} + \frac{\ell}{k}\right). \]
It is not hard to check that \( \mu \) is an action. The interpretations of the methods on the state space \( \{0,1\} \times \mathbb{R}_{\geq 0} \) are given by

\[
\begin{align*}
\{0,1\} \times \mathbb{R}_{\geq 0} & \xrightarrow{\text{val}} \{0,1\} \\
(z,x) & \mapsto z \\
\{0,1\} \times \mathbb{R}_{\geq 0} & \xrightarrow{\text{temp}} \mathbb{R}_{\geq 0} \\
(z,x) & \mapsto x
\end{align*}
\]

\[
\begin{align*}
\{0,1\} \times \mathbb{R}_{\geq 0} & \xrightarrow{\text{on}} \{0,1\} \times \mathbb{R}_{\geq 0} \\
(z,x) & \mapsto (1,x) \\
\{0,1\} \times \mathbb{R}_{\geq 0} & \xrightarrow{\text{off}} \{0,1\} \times \mathbb{R}_{\geq 0} \\
(z,x) & \mapsto (0,x).
\end{align*}
\]

As initial state we take \((0,0) \in \{0,1\} \times \mathbb{R}_{\geq 0}\), as prescribed in the class specification. We leave it to the reader to verify that the assertions in the PTHERM-specification hold in this model.

If we have another PTHERM-model with carrier set \( V \), action \( V \times \mathbb{R}_{\geq 0} \to V \), method interpretations \( \text{val} : V \to \{0,1\}, \text{temp} : V \to \mathbb{R}_{\geq 0}, \text{on} : V \to V, \text{off} : V \to V \) and initial state \( v_0 \in V \). Then there is a unique homomorphims \( f : V \to \{0,1\} \times \mathbb{R}_{\geq 0} \), namely \( f(x) = (x, \text{val}(x), x, \text{temp}(x)) \). We only check that \( f \) commutes with the temperature attributes. If \( x, \text{val} = 0 \), then

\[
f(x).\text{temp}@x = \text{snd}(f(x), x) = (\text{snd} f(x)) \cdot e^{-kz} = (x.\text{temp}@0) \cdot e^{-kz} = x.\text{temp}@x.
\]

And if \( x, \text{val} = 1 \), then

\[
f(x).\text{temp}@x = (\text{snd} f(x) - \frac{i}{\ell}) \cdot e^{-kz} + \frac{i}{\ell} = ((x,\text{temp}@0) - \frac{i}{\ell}) \cdot e^{-kz} + \frac{i}{\ell} = x.\text{temp}@x.
\]

(2) We first notice that the restrictions \( y < z - 1 \Rightarrow x = 1 \) and \( y > z + 1 \Rightarrow x = 0 \) in the above definition of the state space \( U \) deal with the states of adjustment, when the temperature \( y \) in the room is outside the region \([z - 1, z + 1]\) around the goal temperature \( z \). The method interpretations on \( U \) are as follows:

\[
\begin{align*}
U & \xrightarrow{\text{val}} \{0,1\} \\
(x,y,z) & \mapsto x \\
U & \xrightarrow{\text{temp}} \{0,1\} \\
(x,y,z) & \mapsto y \\
U & \xrightarrow{\text{goal}} (1, \frac{\ell}{k} - 1) \\
(x,y,z) & \mapsto z \\
U & \xrightarrow{\text{set}} U \\
(x,y,z) & \mapsto ((x,y,z), a) \begin{cases} (0,y,a) & \text{if } y \geq a \\ (1,y,a) & \text{if } y < a \end{cases}
\end{align*}
\]

The action \( \mu : U \times \mathbb{R}_{\geq 0} \to U \) is more difficult, because it is responsible for the internal autonomous activity. We first define, for a goal temperature \( z \in (1,\ell/k - 1) \), a history function \( h : \mathbb{R}_{\geq 0} \to [0,\ell/k) \) describing the periodic oscillation of the temperature in the room around the goal temperature \( z \), as function of time \( x \in \mathbb{R}_{\geq 0} \). Therefore we first need the times

\[
\uparrow z \overset{\text{def}}{=} \frac{1}{k} \ln \left( \frac{\ell - k(z - 1)}{\ell - k(z + 1)} \right) \quad \text{and} \quad \downarrow z \overset{\text{def}}{=} \frac{1}{k} \ln \left( \frac{z + 1}{z - 1} \right)
\]
that it takes for the temperature in the room to rise from \( z - 1 \) to \( z + 1 \), respectively to fall from \( z + 1 \) to \( z - 1 \). The periodicity of \( h_z \) is then \( \lceil z + \downarrow \rceil \), through the definition

\[
h_z(x) = \begin{cases} 
(z + 1) \cdot e^{-kx} & \text{if } x \in [0, \lceil z \rceil) \\
(z + 1) \cdot e^{-kx} & \text{if } x \in [\lceil z \rceil, \lceil z + \downarrow \rceil) \\
h_z(x - n(\lceil z + \downarrow \rceil)) & \text{otherwise, where } n \in \mathbb{N} \text{ is greatest with } x \geq n(\lceil z + \downarrow \rceil).
\end{cases}
\]

Now we define the action \( \mu : U \times \mathbb{R}_{\geq 0} \to U \) as follows. We first deal with the adjustment phases: if \( y \leq z - 1 \), then

\[
\mu((x, y, z), x) = \begin{cases} 
(x, (y - \frac{1}{k} \cdot e^{-kx} + \frac{1}{k} \cdot z), x - \frac{1}{k} \ln(\frac{e^{ky}}{e^{ky - 1}})) & \text{if } x < \frac{1}{k} \ln(\frac{e^{ky}}{e^{ky - 1}}) \\
\mu((x, z - 1, z), x - \frac{1}{k} \ln(\frac{e^{ky}}{e^{ky - 1}})) & \text{otherwise}.
\end{cases}
\]

And if \( y > z + 1 \), then

\[
\mu((x, y, z), x) = \begin{cases} 
(x, y \cdot e^{-kz}, z) & \text{if } x < \frac{1}{k} \ln(\frac{e^{ky}}{e^{ky - 1}}) \\
\mu((x, z + 1, z), x - \frac{1}{k} \ln(\frac{e^{ky}}{e^{ky - 1}})) & \text{otherwise}.
\end{cases}
\]

Finally, if we are in the “stability” phase \( z - 1 \leq y \leq z + 1 \), then we can use the history function \( h_z \) to define \( \mu \):

\[
\mu((0, y, z), x) = (x, h_z(h_z^{-1}(y) + x), z) \quad \text{where} \\
h_z^{-1}(y) \in [\lceil z \rceil, \lceil z + \downarrow \rceil) \text{ is unique with } h_z(h_z^{-1}(y)) = y, \text{ and} \\
x = 0 \text{ if the derivative } h'_z(h_z^{-1}(y) + x) < 0, \text{ and } x = 1 \text{ else.}
\]

\[
\mu((1, y, z), x) = (x, h_z(h_z^{-1}(y) + x), z) \quad \text{where} \\
h_z^{-1}(y) \in [0, \lceil z \rceil) \text{ is unique with } h_z(h_z^{-1}(y)) = y, \text{ and} \\
x = 0 \text{ if the derivative } h'_z(h_z^{-1}(y) + x) < 0, \text{ and } x = 1 \text{ else.}
\]

It is laborious, but in essence straightforward, to check that \( U \) with this action is a model of the active thermostat specification \( \text{ATHERM} \); and also that it is the terminal model: for an arbitrary \( \text{ATHERM} \) model \( V \) there is a unique homomorphism \( f : V \to U \) given by \( f(x) = \langle x\.\text{val}@0, x\.\text{temp}@0, x\.\text{goal}@0 \rangle \).

5. Terminality and its applications

In this section we shall systematically investigate terminality of coalgebraic models of hybrid class specifications. First we describe terminal models of degenerate class specifications with methods only in terms of so-called “sampling observations”. This leads to a (standard) characterization of bisimilarity in terms of equality on terminal models. Then, in Section 5.3 we use invariants to describe terminal models of (proper) hybrid class specifications with assertions. Invariants are also used for refinements between hybrid specifications in Section 5.4. Finally, in Section 5.5 we show how terminality gives rise to a so-called behaviour-realization adjunction (a common situation in system theory).
5.1. Terminal hybrid models of sampling observations

For the time being, we fix a monoid \( M = (M, 0, +) \). We shall prove the existence of terminal hybrid models of a hybrid coalgebraic specification \( \text{at}: X \to A, \text{proc}: X \times B \to X \) over \( M \) (with operations only). We shall actually give two different – but isomorphic – descriptions of such terminal models, corresponding to two different ways of observing the behaviour of a state \( s \) via samples. In the first model we shall consider observable outputs in \( A \) arising as outcome of experiments of the form

\[
\text{s.proc}(b_1)@\beta_1, \ldots, \text{proc}(b_n)@\beta_n, \text{at}@x
\]

And in the second case, of the form

\[
\text{s.at}@x[\text{proc}(b_1)@\beta_1, \ldots, \text{proc}(b_n)@\beta_n].
\]

In the first case we record the observable output \( x \) time-units after the end of the sequence of procedure applications \( \text{proc}(b_1)@\beta_1, \ldots, \text{proc}(b_n)@\beta_n \). And in the second case we record the attribute value \( x \) time-units after the beginning of applying the sequence of procedures \( \text{proc}(b_1)@\beta_1, \ldots, \text{proc}(b_n)@\beta_n \), see Definition 3.

At first we ignore initial states.

**Proposition 11.** Consider a hybrid specification with an attribute \( \text{at}: X \to A \) and a procedure \( \text{proc}: X \times B \to X \) over the (fixed) monoid \( M \). The terminal hybrid model in this situation can be described as the set

\[
Z \overset{\text{def}}{=} A((B \times M)^* \times M)
\]

of functions from \((B \times M)^* \times M\) to \( A \). These functions may be seen as “sampling observations”. The interpretations of the methods are

\[
\begin{align*}
\text{at}: Z & \to A \\
\varphi & \mapsto \varphi(\langle, 0 \rangle) \\
\text{proc}: Z \times B & \to Z \\
(\varphi, b) & \mapsto \hat{\lambda}(\sigma, x) \in (B \times M)^* \times M \varphi((b, 0) \cdot \sigma, x).
\end{align*}
\]

The action \( \mu: Z \times M \to Z \) is

\[
\mu(\varphi, \beta) = \hat{\lambda}(\sigma, x) \in (B \times M)^* \times M \begin{cases} 
\varphi(\langle, \beta + x \rangle) & \text{if } \sigma = \langle \rangle \\
\varphi((c, \beta + \gamma) \cdot \sigma', x) & \text{if } \sigma = (c, \gamma) \cdot \sigma'.
\end{cases}
\]

We shall write \( Z = \langle Z, \text{at}, \text{proc}, \mu \rangle \) for this model.

Recall from Section 2 how a single attribute and procedure can be combinations of several attributes and of several procedures. Notice that the monoid \( M \) is a parameter in this result. The standard examples are the monoids \((\mathbb{N}, 0, +)\) of discrete time, and \((\mathbb{R}_{\geq 0}, 0, +)\) of real time. But the result is not confined to these instances. If we take the trivial singleton-set monoid \( M = \{0\} \), then the proposition reduces to Theorem 1(i).

In general, elements of terminal coalgebras are “observations”. This applies here as well, since a function \( \varphi \in Z = A((B \times M)^* \times M \) describes the observable outcome of
experiments:

\[
\phi((b_1, \beta_1), \ldots, (b_n, \beta_n), x) = \phi.\text{proc}(b_1)@\beta_1. \cdots .\text{proc}(b_n)@\beta_n. \text{at}@x \in A.
\]

This intuition may be helpful in the proof.

**Proof.** We first have to check that \(\mu\) is indeed a monoid action, see Eq. (6). The equation \(\mu(\varphi, 0) = \varphi\) follows because \(0 \in M\) is neutral element. Associativity of addition + in \(M\) yields the other equation \(\mu(\varphi, \beta_1 + \beta_2) = \mu(\mu(\varphi, \beta_1), \beta_2)\) in

\[
\mu(\varphi, \beta_1 + \beta_2)((\cdot), x) = \varphi((\cdot), (\beta_1 + \beta_2) + x) \\
= \varphi((\cdot), \beta_1 + (\beta_2 + x)) \\
= \mu(\varphi, \beta_1)((\cdot), \beta_2 + x) \\
= \mu(\mu(\varphi, \beta_1), \beta_2)((\cdot), x)
\]

and similarly

\[
\mu(\varphi, \beta_1 + \beta_2)((c, \gamma) \cdot \sigma, x) = \varphi((c, (\beta_1 + \beta_2) + \gamma) \cdot \sigma, x) \\
= \varphi((c, \beta_1 + (\beta_2 + \gamma)) \cdot \sigma, x) \\
= \mu(\varphi, \beta_1)((c, \beta_2 + \gamma) \cdot \sigma, x) \\
= \mu(\mu(\varphi, \beta_1), \beta_2)((c, \gamma) \cdot \sigma, x).
\]

So let us assume another model with state space \(U\), with action \(\mu : U \times M \to U\), attribute \(\text{at} : U \to A\) and procedure \(\text{proc} : U \times B \to U\). We can then define a function \(! : U \to Z = A^{(B \times M)^*} \times M\) by

\[
!(x)((b_1, \beta_1), \ldots, (b_n, \beta_n), x) \overset{\text{def}}{=} x.\text{proc}(b_1)@\beta_1. \cdots .\text{proc}(b_n)@\beta_n. \text{at}@x,
\]

where \text{proc} and \text{at} on the right-hand side are in \(U\). We show that the function \(!\) commutes with the operations \((-).\text{at}@x\) and \((-).\text{proc}(c)@\gamma\).

\[
!(x).\text{at}@x = \text{at}(\mu(!(x), x)) = \mu(\mu(x), x)((\cdot), 0) = !(x)(\cdot, x) = x.\text{at}@x.
\]

Also

\[
(!(x).\text{proc}(c)@\gamma)((b_1, \beta_1), \ldots, (b_n, \beta_n), x) \\
= \text{proc}(\mu(!(x), \gamma), \cdot)((b_1, \beta_1), \ldots, (b_n, \beta_n), x) \\
= \mu(\mu(x), \gamma)((c, 0) \cdot (b_1, \beta_1), \ldots, (b_n, \beta_n), x) \\
= !x)(\cdot, x) \cdot (b_1, \beta_1), \ldots, (b_n, \beta_n), x) \\
= x.\text{proc}(c)@\gamma.\text{proc}(b_1)@\beta_1. \cdots .\text{proc}(b_n)@\beta_n. \text{at}@x \\
= !(x.\text{proc}(c)@\gamma)((b_1, \beta_1), \ldots, (b_n, \beta_n), x).
\]

The proof of uniqueness of such a homomorphism \(! : U \to Z\) is left to the reader. □
Earlier in Definition 3 we have seen alternative notation for the observations that we can make about a state \( s \). It involves expressions \( s @ x_\sigma \), where \( \sigma \) is a sequence \( \text{proc}(b_1)@b_1, \ldots, \text{proc}(b_n)@b_n \) of procedure applications. We shall identify such a sequence with the input-time pairs \((b_1, \beta_1), \ldots, (b_n, \beta_n) \in (B \times M)^*\). This alternative notation describes the output function that we used in our simulations in Section 3. The observable output value \( s @ [\sigma] \) at \( x \) is defined as follows.

\[
\begin{aligned}
\text{s.at}[\sigma] &= \text{s.at} \quad \text{and} \quad \text{s.at}[((b, \beta) \cdot \sigma)] \\
&= \begin{cases}
\text{s.at} & \text{if } x < \beta \\
\text{s.proc}(b)@\beta. \text{at}[@(x - \beta)[\sigma]} & \text{if } x \geq \beta.
\end{cases}
\end{aligned}
\]

For this definition we actually assume that \( M \) is \( \mathbb{N} \) or \( \mathbb{R}^\geq 0 \) (in order to be able to compare \( x \) and \( \beta \)).

Interestingly, these alternative observations can also be organized into a terminal hybrid model \( Z' \), as will be shown next. But of course, we then have an isomorphism \( Z \cong Z' \) of hybrid models, since terminal objects in a category are only determined up to isomorphism.

**Proposition 12.** Take a hybrid specification \( \text{at} : X \to A, \text{proc} : X \times B \to X \) over \( M \) as above. Consider the subset \( Z' \subseteq Z = A^{\mathbb{N} \times M} \times M \) given by

\[
Z' = \{ \varphi \in Z \mid \forall (\sigma, x) \in (B \times M)^* \times M \, \varphi(\sigma, x) = \varphi(\sigma_{|_x}, x) \}
\]

where

\[
\sigma_{|_x} = \begin{cases}
\{\} & \text{if } \sigma = \{\} \\
\{\} & \text{if } \sigma = (b, \beta) \cdot \sigma' \text{ and } x < \beta \\
(b, \beta) \cdot (\sigma'_{|(x - \beta)}) & \text{if } \sigma = (b, \beta) \cdot \sigma' \text{ and } x \geq \beta.
\end{cases}
\]

This \( Z' \) then also forms a terminal hybrid model \( Z' \), with method interpretations on \( \varphi \in Z' \) as in \( Z \):

\[
\varphi. \text{at} \overset{\text{def}}{=} \varphi(\{\}, 0) \quad \text{and} \quad \varphi. \text{proc}(b) \overset{\text{def}}{=} \lambda(\sigma, x) \varphi((b, 0) \cdot \sigma, x)
\]

but with a slightly different monoid action \( \mu' : Z' \times M \to Z' \) given by

\[
\mu'(\varphi, \beta) = \lambda(\sigma, x) \begin{cases}
\varphi(\{\}, \beta + x) & \text{if } \sigma = \{\} \\
\varphi((c, \beta + \gamma) \cdot \sigma', \beta + x) & \text{if } \sigma = (c, \gamma) \cdot \sigma'.
\end{cases}
\]

The proof of this result consists of a straightforward verification of the terminality property, like in the previous proof. It uses the fact that \( s.\text{at}[\sigma] = s.\text{at}[\sigma_{|_x}] \).

We conclude by remarking that the resulting isomorphism \( f : Z' \iso Z \) of models is given by

\[
f(\varphi)(\sigma, x) = \varphi(\sigma, \text{sum}(\pi'*(\sigma)) + x)
\]

and

\[
f^{-1}(\varphi)(\sigma, x) = \varphi(\sigma_{|_x}, x - \text{sum}(\pi'*(\sigma_{|_x})))
\]

where \( \text{sum}(\pi'*(\sigma)) \) is the sum of the \( \beta_i \in M \) occurring as \( (b_i, \beta_i) \in \sigma \in (B \times M)^* \).
5.2. Hybrid bisimilarity

Bisimilarity is a fundamental notion in state-based computation and in process theory [30]. It describes indistinguishability of states, from the perspective of an observer (or client) on the outside. It typically arises in a situation with limited access to a state space, considered as a black box. Bisimilarity is a standard notion in coalgebra, and captures indistinguishability of states via the coalgebraic operations (acting on the state space). Terminal coalgebras generally play an important rôle because they identify all observationally indistinguishable (bisimilar) states (see e.g. [1, 37, 21]). Along the same lines we shall in this subsection first define an appropriate relation $\rightarrow$ of “hybrid bisimilarity”, and then show that this relation $\rightarrow$ corresponds to equality on terminal hybrid models.

Definition 13. Consider two hybrid models $\mathcal{U} = (\text{at}_U : U \to A, \text{proc}_U : U \times B \to U, \mu_U : U \times M \to U)$ and $\mathcal{V} = (\text{at}_V : V \to A, \text{proc}_V : V \times B \to V, \mu_V : V \times M \to V)$ over some monoid $M$. A relation $R \subseteq U \times V$ on the state spaces of these models is called a bisimulation if it satisfies, for all $x \in U$ and $y \in V$,

$$R(x, y) \Rightarrow \begin{cases} x \text{ at } U = y \text{ at } V \\
R(x \text{ proc}_U (b), y \text{ proc}_V (b)) & \text{ for all } b \in B, \text{ and} \\
R(\mu_U (x, z), \mu_V (y, z)) & \text{ for all } z \in M. 
\end{cases}$$

We shall say that two states $x \in U, y \in V$ are bisimilar, and write this as $x \Uparrow \rightarrow y$, if there is a bisimulation $R \subseteq U \times V$ with $R(x, y)$.

It is not hard to see that bisimilarity $\Uparrow \rightarrow$ can be characterized as the greatest bisimulation, and the bisimilarity relational $\Uparrow \rightarrow \subseteq U \times U$ on a single model $\mathcal{U}$ is an equivalence relation. This is because the identity relation on $U$, the converse relation $\Uparrow \rightarrow^\text{op}$ and the composite relation $\Uparrow \rightarrow \circ \Uparrow \rightarrow$ are bisimulations, and thus contained in $\Uparrow \rightarrow$.

As an example, consider the hybrid model $\mathcal{V}$ of the discrete time flip-flop DTFF with the set $V = \{(n, m) \mid n \geq m\} \subseteq \mathbb{N} \times \mathbb{N}$ as state space (as described in Section 4.2). The two states $(10, 6) \in V$ and $(25, 21) \in V$ are observationally indistinguishable: their values $(10, 6).\text{val}$ and $(25, 21).\text{val}$ are the same, namely 1, and for both of them this value becomes 0 after 6 time-units. And also, after applying the on method to them they will behave in the same way. Hence, we cannot see a difference, using the operations $(-).\text{val}@x$ and $(-).\text{on}@x$ at our disposal. These states $(10, 6), (25, 21) \in V$ are bisimilar. It is not hard to show that for arbitrary states $(n_1, m_1), (n_2, m_2) \in V$

$$(n_1, m_1) \rightarrow \rightarrow (n_2, m_2) \Leftrightarrow (m_1 + 10) \triangleright n_1 = (m_2 + 10) \triangleright n_2.$$ 

We recall from Section 4.2 that the terminal model of this discrete time flip-flop is the set of natural numbers $[0, 10]_\triangleright \subseteq \mathbb{N}$. The unique homomorphism of models
\[ !: V \to [0, 10]_\mathbb{N} \] is the function \( !(n, m) = \min\{\beta \in [0, 10]_\mathbb{N} \mid (n, m).\text{val}\beta = 0\} = (m+10) \triangleleft n. \] We see that the two bisimilar elements \((10, 6), (25, 21)\) in \( V \) considered above are equal when mapped to the terminal coalgebra: \( !(10, 6) = 6 = !(25, 21) \). This turns out to be a general phenomenon.

**Theorem 14.** Consider two hybrid models \( \mathcal{U} \) and \( \mathcal{V} \) as in the previous definition (with state spaces \( U \) and \( V \) respectively), together with the two resulting homomorphisms of models \( !_\mathcal{U} : U \to Z \) and \( !_\mathcal{V} : V \to Z \) to the terminal model \( Z \) with state space \( Z = A^{(B \times M)^*} \times M \) as in Proposition 11. Then, for states \( x \in U \) and \( y \in V \)

\[ x \overset{!\mathcal{U}}{\leftrightarrow} y \iff !_\mathcal{U}(x) = !_\mathcal{V}(y). \]

States are thus bisimilar if and only if they are equal when mapped to the terminal model \( Z \). In particular, bisimilarity \( \overset{!\mathcal{U}}{\leftrightarrow} \) on the terminal model \( Z \) is the identity relation (since the function \( ! \) is the identity homomorphism, by uniqueness). Actually, this result holds by virtue of \( Z \) being the terminal model; it also holds for \( Z' \) in Proposition 12.

Recall from the proof of Proposition 11 that the unique map \( ! \) to the terminal model \( Z \) is defined by \( !(x) ((b_1, \beta_1), \ldots, (b_n, \beta_n)), z) = x.\text{proc} (b_1) @ \beta_1 \cdots.\text{proc} (b_n) @ \beta_n. \text{at} @ z \). This will be used in the next proof.

**Proof.** (\( \Rightarrow \)) For \( n \in \mathbb{N} \) define

\[ K_n = \{(x, y) \in U \times V \mid \forall (b_1, \beta_1), \ldots, (b_n, \beta_n) \in (B \times M) \forall z \in M \]

\[ x.\text{proc} \ s @ \beta_1 \cdots.\text{proc} \ s @ \beta_n. \text{at} @ z \]

\[ = y.\text{proc} \ s @ \beta_1 \cdots.\text{proc} \ s @ \beta_n. \text{at} @ z \}. \]

Then one proves \( R \subseteq K_n \) for an arbitrary bisimulation \( R \subseteq U \times V \), by induction on \( n \). In particular, \( \overset{!\mathcal{U}}{\leftrightarrow} \subseteq \bigcap_n K_n = \{(x, y) \mid !_\mathcal{U}(x) = !_\mathcal{V}(y)\} \).

(\( \Leftarrow \)) One shows that the “kernel” relation \( K = \{(x, y) \mid !_\mathcal{U}(x) = !_\mathcal{V}(y)\} \subseteq U \times V \) is a bisimulation, using the explicit formulation of \( ! \) given above. Hence \( K \subseteq \overset{!\mathcal{U}}{\leftrightarrow} \), since \( ! \) is by definition the greatest bisimulation. \( \square \)

**Corollary 15.** Let \( \mathcal{U}, \mathcal{V} \) be hybrid models as above, and let \( f : \mathcal{U} \to \mathcal{V} \) be a homomorphism of models. Then for \( x, y \in U \) and \( \gamma \in M \)

1. \( x \overset{!\mathcal{U}}{\leftrightarrow} y \Rightarrow f(x) \overset{!\mathcal{V}}{\leftrightarrow} f(y) \).
2. \( f(\mu_{\mathcal{U}}(x, \gamma)) \overset{!\mathcal{V}}{\leftrightarrow} \mu_{\mathcal{V}}(f(x), \gamma) \).

**Proof.**

1. By uniqueness of maps to a terminal model we get \( !_{\mathcal{V}} \circ f = !_\mathcal{U} \). Hence

\[ x \overset{!\mathcal{U}}{\leftrightarrow} y \Rightarrow !_\mathcal{V}(f(x)) = !_\mathcal{V}(f(y)) \Rightarrow f(x) \overset{!\mathcal{V}}{\leftrightarrow} f(y). \]
2. Since
\[ f(\mu_U(x, \gamma)).\ Proc_{\gamma}(b_1)@\beta_1, \ldots, \ Proc_{\gamma}(b_n)@\beta_n, \ at_{\gamma}@x \]
\[ = f(\mu_U(x, \gamma)).\ Proc_{\gamma}(b_1)@\beta_1, \ldots, \ Proc_{\gamma}(b_n)@\beta_n, \ at_{\gamma}@x \]
\[ = \mu_U(x, \gamma).\ Proc_{\gamma}(b_1)@\beta_1, \ldots, \ Proc_{\gamma}(b_n)@\beta_n, \ at_{\gamma}@x \]
\[ = \begin{cases} 
  x.at_{\gamma}(\gamma + z) & \text{if } n = 0 \\
  x.proc_{\gamma}(b_1)@((\gamma + \beta_1) + \cdots + \beta_n).\ Proc_{\gamma}(b_n)@\beta_n, \ at_{\gamma}@x & \text{if } n > 0 
\end{cases} \]
\[ = \begin{cases} 
  f(x).\ Proc_{\gamma}(b_1)@\gamma + z & \text{if } n = 0 \\
  f(x).\ Proc_{\gamma}(b_1)@((\gamma + \beta_1) + \cdots + \beta_n).\ Proc_{\gamma}(b_n)@\beta_n, \ at_{\gamma}@x & \text{if } n > 0 
\end{cases} \]
\[ = \begin{cases} 
  \mu_{\gamma}(f(x), \gamma). at_{\gamma}@x & \text{if } n = 0 \\
  \mu_{\gamma}(f(x), \gamma).\ Proc_{\gamma}(b_1)@((\gamma + \beta_1) + \cdots + \beta_n).\ Proc_{\gamma}(b_n)@\beta_n, \ at_{\gamma}@x & \text{if } n > 0 
\end{cases} \]
\[ = \mu_{\gamma}(f(x), \gamma).\ Proc_{\gamma}(b_1)@\gamma + \beta_1, \ldots, \ Proc_{\gamma}(b_n)@\beta_n, \ at_{\gamma}@x. \]

The second point shows that homomorphisms \( f \) of hybrid models commute with monoid actions up to bisimilarity (i.e. as far as a client can observe). For the special case when the homomorphism \( f \) is the map \( !: U \to \mathcal{Z} \to \mathcal{Z} \), we get an actual equality \( !((\mu_U(x, \gamma))) = \mu_{\mathcal{Z}}(!(x), \gamma) \), since \( \mathcal{Z} \subseteq \mathcal{Z} \times \mathcal{Z} \) is the equality relation.

### 5.3. Invariants and terminal hybrid models satisfying assertions

In Section 5.1 we have considered terminal hybrid models of class specifications consisting of operations only. In the present subsection we consider terminal hybrid models for class specifications with additional assertions (as used in Section 3). We show that terminal models for such specifications also exist, via an explicit description. The approach is as in [18, 23] (which deal with the non-temporal case), and extends Theorem 1(ii). It can be described in terms of (greatest) invariants. The latter are predicates on the state space of a system, which, once they are true of a state \( s \), will remain true no matter which operations are applied to \( s \). The notion of invariant for hybrid models will be introduced first.

**Definition 16.** Consider a hybrid model \( \mathcal{U} = (\mathcal{A}, \mathcal{B}, \mathcal{M}, \mu : U \times \mathcal{M} \to U) \). A predicate \( P \subseteq U \) on the state space will be called an invariant if \( P \) is closed under the operations, i.e. if for all \( s \in U \),

\[ P(x) \Rightarrow \begin{cases} 
  P(x, \mathcal{B}(b)) & \text{for all } b \in \mathcal{B} \\
  P(\mu(x, z)) & \text{for all } z \in \mathcal{M}. 
\end{cases} \]

Notice that we do not include the requirement that invariants should hold of initial states. An invariant on a model \( \mathcal{U} \), considered as a subset of the state space, inherits the operations of \( \mathcal{U} \), and thus forms a submodel of \( \mathcal{U} \).
We turn to the construction of models satisfying assertions. These assertions, say in a class specification with methods $\text{at} : X \rightarrow A$, $\text{proc} : X \times B \rightarrow X$ over a monoid $M$, involve (possibly conditional) statements of the form

$$s.\text{proc}(b)@b_i.\text{at}@x = s.\text{proc}(d_j)@d_j.\text{at}@\gamma$$

(plus inequalities $x<\beta$, $x\leq \beta$ as assumptions).\(^10\) Such assertions typically involve a single state variable $s$. They can be interpreted in the terminal model $Z = A^{(B \times M)^*} \times M$, and lead to a subset

$$E = \{ \phi \in Z | \phi \text{ satisfies the assertions} \} \subseteq Z.$$  

We shall construct the terminal model satisfying the assertions via the “greatest invariant” of $Z$, contained in $E$.

**Proposition 17.** Consider a hybrid class specification with assertions as above, and let $E \subseteq Z$ be the resulting set of states of the terminal model $\mathcal{Z}$ for which the assertions hold. The terminal hybrid model satisfying the assertions is given by the greatest invariant $E \hookrightarrow E$ with state space

$$E = \{ \phi \in Z | \forall ((b_1, \beta_1), \ldots, (b_n, \beta_n)) \in (B \times M)^* \forall \gamma \in M \mu(\phi.\text{proc}(b_1)@\beta_1, \ldots, \text{proc}(b_n)@\beta_n, \gamma) \in E \}$$

(7)

with methods inherited from $Z$.

**Proof.** We first show that $E$ is indeed an invariant, i.e. that $E$ is closed under $\mathcal{Z}$’s procedure $\text{proc}$ and under its action $\mu$: if $\phi \in E$, then $\phi.\text{proc}(b) \in E$, since for $(b_1, \beta_1), \ldots, (b_n, \beta_n) \in (B \times M)^*$ and $\gamma \in M$ we have an equation $\mu(\phi.\text{proc}(b).\text{proc}(b_1)@\beta_1, \ldots, \text{proc}(b_n)@\beta_n, \gamma) = \mu(\phi.\text{proc}(b)@0.\text{proc}(b_1)@\beta_1, \ldots, \text{proc}(b_n)@\beta_n, \gamma)$, where the right-hand side is in $E$, by assumption. Similarly for $\mu$. Hence $E$ forms the state space of a (sub)model of the operations of the specification.

Next, $E$ is a model of the assertions, since $E \subseteq E$. This follows easily by taking $n = 0 \in \mathbb{N}$ and $\gamma = 0 \in M$ in the definition of $E$.

Finally, $E$ is the terminal model satisfying the assertions: if $\mathcal{U}$ is an arbitrary model satisfying the assertions, then we get of course a unique homomorphism $! : \mathcal{U} \rightarrow \mathcal{Z}$.

We have to show that $!$ factors through $E \hookrightarrow Z$. Firstly, $!$ factors through $E \hookrightarrow Z$ since the assertions hold for states $x$ in $\mathcal{U}$, and hence also for $!(x) \in Z$, because $!$ is a homomorphism and thus commutes with the operations $\text{proc}(b)@\beta$, $\text{at}@x$ which build up the assertions. Secondly, $!$ factors through $E \hookrightarrow Z$ using the definition of $E$: the image

\(^{10}\) The statements that we use in this paper only involve actual equations between observable output values. One may also wish to use “equations” between states, in the form of bisimilarity $\equiv$ (like in [21, 23]). This gives additional statements of the form $s.\text{proc}(b_i)@b_i \equiv s.\text{proc}(d_j)@d_j$. The approach that we are about to describe applies to such “equations” as well, since they are preserved by homomorphisms, see Corollary 15(i). This is needed in the proof of Proposition 17.
of \( f \) is closed under \( \text{proc} \) (by definition of homomorphism) and under \( \mu \) (by Corollary 15(ii), since \( \not\equiv \)) is equality). \( \Box \)

We shall apply this abstract description in the concrete case of the discrete time flip-flop DTFF from Fig. 2. In this case we have as monoid \( M = \mathbb{N} \), as attribute output set \( A = \{0,1\} \), and as (trivial) procedure input set \( B = \{0\} \). Hence, the terminal model of operations has as state space the set

\[ Z = A^{(B \times M)^*} \times M = \{0,1\}^{\mathbb{N}^*} \times \mathbb{N} = \{0,1\}^{\mathbb{N}^*} \]

(where \( \mathbb{N}^+ \rightarrow \mathbb{N}^* \) is the set of non-empty finite sequences) with operations

\[ \varphi \xrightarrow{\text{val}} \varphi(0), \quad \varphi \xrightarrow{\text{on}} \lambda \sigma \in \mathbb{N}^+ \varphi(0) \cdot \sigma, \]

\[ (\varphi, \gamma) \xrightarrow{\mu} \lambda \sigma \in \mathbb{N}^+ \varphi(\gamma + \text{head}(\sigma)) \cdot \text{tail}(\sigma). \]

The three assertions \( \text{s.val@x0} \vdash \text{s.val@}(x+\beta) = 0 \), \( \beta \geq 10 \vdash \text{s.val@}\beta = 0 \) and \( \beta < 10 \vdash \text{s.on@x.val@}\beta = 1 \) in the DTFF-specification determine the subset \( E \subseteq Z = \{0,1\}^{\mathbb{N}^*} \) as

\[ E = \{ \varphi \in \{0,1\}^{\mathbb{N}^*} | \forall x, \beta \in \mathbb{N} \varphi(\langle x \rangle) = 0 \Rightarrow \varphi(\langle x + \beta \rangle) = 0 \]

and \( \beta \geq 10 \Rightarrow \varphi(\langle \beta \rangle) = 0 \) and \( \beta < 10 \Rightarrow \varphi(\langle x, \beta \rangle) = 1 \}. \]

Then the subset \( E \subseteq E \) is, according to (7),

\[ E = \{ \varphi \in \{0,1\}^{\mathbb{N}^*} | \forall \sigma \in \mathbb{N}^* \forall x, \beta \in \mathbb{N} \varphi(\sigma \cdot \langle x \rangle) = 0 \Rightarrow \varphi(\sigma \cdot \langle x + \beta \rangle) = 0 \]

and \( \beta \geq 10 \Rightarrow \varphi(\sigma \cdot \langle \beta \rangle) = 0 \) and \( \beta < 10 \Rightarrow \varphi(\sigma \cdot \langle x, \beta \rangle) = 1 \}. \]

If we look at this description of \( E \) for a moment, then we see that for functions \( \varphi \in E \), only the outcomes \( \varphi(\langle 0 \rangle), \ldots, \varphi(\langle 9 \rangle) \in \{0,1\} \) are not fully determined. But we do not have \( 2^{10} \) possibilities, since if \( \varphi(\langle i \rangle) = 0 \), then also \( \varphi(\langle i+1 \rangle) = 0 \). Hence \( E \) has precisely 11 elements: there is an isomorphism

\[ \{0,1,\ldots,10\} = [0,10]_\mathbb{N} \xrightarrow{\sim} E \quad \text{naming} \quad i \mapsto (\varphi \text{ for which } i \text{ is least with } \varphi(\langle i \rangle) = 0). \]

Via this isomorphism \( [0,10]_\mathbb{N} \xrightarrow{\sim} E \) the state space \( [0,10]_\mathbb{N} \) inherits its operations as described in Section 4.2 from \( E \). This concludes the example.

The above Proposition 17 gives a general description \( E \) of the terminal model satisfying a class specification. In many concrete cases – like above – it can be simplified considerably. Another way to find the terminal model is to follow the (intuitive) idea that a terminal model is a minimal realization, and to describe the minimal set of states needed for the required behaviour. Then one still needs to check terminality explicitly. This is the approach that we followed in Section 4.

As a final point we briefly consider the (interpretation of the) initial state \( \text{new} \) in terminal models. The specification should determine \( \text{new} \) up to bisimilarity. That is, one
should know the value \texttt{new.proc}(b_1)@b_1, \ldots, \texttt{proc}(b_n)@b_n, at @z for each sequence of inputs \((b_1, b_1), \ldots, (b_n, b_n)\) and attribute time \(z\). These outcomes determine a function \((B \times M)^* \times M \to A\), which, as an element of the terminal coalgebra \(Z = A((B \times M)^* \times M)\), yields the interpretation of \texttt{new} in \(Z\).

In most situations the immediate attribute values \texttt{new.at}@0 are specified, and the outcomes \texttt{new.proc}(b_i)@b_i, at @z are determined by the assertions. This should guarantee that the interpretation of \texttt{new} in \(Z\) is in the subset \(E \subset \mathcal{Z}\). But this need not be the case in an arbitrary specification: for example, there may, by mistake, be an inconsistency in the combination of the assertions with the creation conditions.

In the above example of the discrete time flip-flop, the interpretation of the initial state \texttt{new} in the state space \([0, 10]_\mathbb{N}\) is the element \(0 \in [0, 10]_\mathbb{N}\), since the \texttt{val} attribute is defined as \(\texttt{val}(s) = 0 \iff s = 0\). But of course we could have chosen a different creation condition, say with the value of the initial state \texttt{new} equal to \(1\), and with this \texttt{new} keeping this value for the next 5 time-units. This would yield \(5 \in [0, 10]_\mathbb{N}\) as interpretation for \texttt{new}. (Another possibility, which we do not consider in this paper, is to have a parametrized initialization of the form \texttt{new}(n), where \(n \leq 10\) determines the first time-unit where the value of \texttt{new} will be \(0\).)

5.4. Refinement of hybrid specifications via invariants

The idea of a refinement of a specification \(\mathcal{T}\) by another specification \(\mathcal{S}\) is that \(\mathcal{S}\) is in a sense more concrete, more detailed, or more easily available than \(\mathcal{T}\). Typically in an object-oriented setting with a library of classes at hand, one tries to refine towards existing classes, for example because (reliable) implementations of these are available.

The approach to refinement of hybrid specifications that we present here is a straightforward extension of the approach for ordinary coalgebraic specifications, as set forward in [23]. Therefore we will only give a sketch, using an example.

In order to refine a specification \(\mathcal{A}\) (for abstract) by a specification \(\mathcal{C}\) (for concrete) via an invariant one needs to
1. express the abstract methods in terms of the concrete ones;
2. construct for each model of \(\mathcal{C}\) an \(\mathcal{A}\)-invariant \(P\) on the state space and an
   \(\mathcal{A}\)-reachable state \(r\) in the model (serving as initial state for \(\mathcal{A}\)) such that:
   (a) \(P\) holds for \(r\);
   (b) \(r\) satisfies the creation conditions from \(\mathcal{A}\), expressed in terms of the translated methods of the \(\mathcal{C}\)-model;
   (c) the assertions from \(\mathcal{A}\) (expressed via the methods of the \(\mathcal{C}\)-model) are implied by \(P\).

Together these requirements guarantee that all the states that one ever reaches (using the \(\mathcal{A}\)-methods) in the \(\mathcal{C}\)-model from \(r\), form a model of the abstract specification, since one never gets outside the invariant \(P\), on which the abstract assertions hold.

Suppose, as an example, that we wish to refine the class specification RTFF of the real-time flip-flop from Section 3.1. Also suppose that we have a specification ART
at hand of a fancy adjustable timer in real time, see Fig. 10. It has a time attribute describing the current time indicated by the timer, and a dur attribute describing the time to which a client can set the duration dur of the timer (by the set procedure; it ensures that the current time is always less than the duration.) The reader may wish to verify that the terminal model of this specification has as state space the set of pairs of reals \( \{(x, y) \mid 0 \leq x \leq y\} \).

We wish to refine RTFF by ART. First of all, we have to express the RTFF-methods val and on in terms of ART-methods. We define

\[
\text{val} : X \rightarrow \{0, 1\} \quad \text{by } x.\text{val} = \begin{cases} 
1 & \text{if } x.\text{time} > 0 \\
0 & \text{otherwise,}
\end{cases}
\]

and take on as translation of on. The reachable state \( r \) is defined as the (ART-) initial state with duration set to 10 – as used in the RTFF-specification: \( r = \text{new.set}(10)@0 \).

The invariant \( P \) that we shall use is

\[
P = \{ x \mid x.\text{dur}@0 = 10 \}.
\]

We prove the above requirements of a refinement. First we have to see that \( P \) is an invariant. This requires a proof that \( P \) is closed under on and under the action \( \mu \). Assume
therefore that \( P(x) \) holds. Then also \( P(x, \text{on}) \), since \( x.\text{on} \cdot \text{dur}@0 = x.\text{on}@0 \cdot \text{dur}@0 = x.\text{dur}@0 = 10 \). And similarly, \( P(\mu(x, z)) \) holds for each \( x: \mu(x, z) \cdot \text{dur}@0 = x.\text{dur}@z = x.\text{dur}@0 = 10 \). We continue with the above three conditions (a)–(c).

(a) The predicate \( P \) holds of \( r \), since \( r.\text{dur}@0 = \text{new.set}(10) @0.\text{dur} = 10 \).

(b) The state \( r \) satisfies the RTFF creation condition: \( r.\text{val}@0 = 0 \). This follows from the definition of \( \text{val} \) and from: \( r.\text{time}@0 = \text{new.set}(10) @0.\text{time} = \text{new.time}@0 = 0 \).

(c) The RTFF assertions hold on states satisfying \( P \). Assuming \( P(x) \) we get

- \( x.\text{val}@x = 0 \Rightarrow x.\text{time}@x = 0 \Rightarrow x.\text{time}@x(\alpha + \beta) = x.\text{time}@x \uparrow \beta = 0 \Rightarrow x.\text{val}@ (\alpha + \beta) = 0 \);
- for \( \beta \geq 10 \), \( x.\text{val}@\beta = 0 \) since \( x.\text{time}@\beta = x.\text{time}@0 \uparrow \beta = 0 \), because \( x.\text{time}@0 \leq x.\text{dur}@0 = 10 \) by assumption;
- for \( \beta < 10 \), \( x.\text{on}@x.\text{val}@\beta = 1 \), since \( x.\text{on}@x.\text{time}@\beta = x.\text{on}@x.\text{time}@0 \uparrow \beta = x.\text{dur}@x \uparrow \beta = 10 \uparrow \beta > 0 \);
- finally, the denseness assertion holds: if \( x.\text{val}@x = 1 \), i.e. if \( x.\text{time}@x > 0 \), then we can find a \( \beta > 0 \) with \( \beta < x.\text{time}@x \). But then \( x.\text{time}@x(\alpha + \beta) = x.\text{time}@x \uparrow \beta > 0 \), so that \( x.\text{val}@ (\alpha + \beta) = 1 \).

As a second example one can refine the watch-dog specification in Fig. 3 via two ARTs (as components in of the refining concrete class).

5.5. A behaviour-realization adjunction

In this subsection we return to Proposition 11 describing terminal models of operations only (without assertions). This result will be crucial in establishing a so-called “behaviour-realization” adjunction in our hybrid setting. Such an adjunction captures the fundamental relation between machines which can perform certain behaviour, and behaviours which can be realized in a certain (universal) way. Such behaviour-realization adjunctions are typical in mathematical system theory, following work [9–11] of Goguen. We adapt this approach in a minor way, by taking the morphisms between input sets in contravariant direction in order to avoid some unnecessary restrictions.

For the remainder of this subsection, we fix a monoid \( M = (M, 0, +) \). As mentioned before, a hybrid model \( \mathcal{U} \) consists of a state space \( U \) with an initial state \( u_0 \in U \), an attribute \( : U \rightarrow A \), a procedure \( \text{proc}: U \times B \rightarrow U \) and a monoid action \( \mu: U \times M \rightarrow U \). For convenience, we write this information as \( (u_0, U \rightarrow A \times U^B \times U^M) \). In this setting, such a structure will also be called a (hybrid) machine (in which the procedure and action are seen as transition functions).

We start by organizing these machines in a category \( \mathbf{HM} \). Its objects are thus hybrid machines \( (u_0, U \rightarrow A \times U^B \times U^M) \). And a morphism \( (u_0, U \rightarrow A \times U^B \times U^M) \rightarrow (v_0, V \rightarrow C \times V^D \times V^M) \) in \( \mathbf{HM} \) between two such machines will be defined as a 3-tuple \( (f, g, \phi) \) of functions

\[
A \xrightarrow{f} D \quad B \xrightarrow{g} D \quad U \xrightarrow{\phi} V
\]
satisfying
\[
\phi(\mu_0) = \nu_0, \quad \phi(\mu) \cdot \text{at}\alpha = f(\mu \cdot \text{at}\alpha),
\]
\[
\phi(\mu) \cdot \text{proc}(d) \cdot \text{at}\alpha = \phi(\mu \cdot \text{proc}(g(d)) \cdot \text{at}\alpha).
\]
for all \(x \in U\), \(\alpha \in M\) and \(d \in D\). Diagrammatically this reads as

\[
\begin{array}{cccc}
1 & \longrightarrow & 1 \\
\downarrow & & \downarrow \\
U & \phi & V \\
\downarrow & & \downarrow \\
\downarrow & & \downarrow \\
U \times M & \phi \times \text{id} & V \times M \\
\downarrow & & \downarrow \\
\downarrow & & \downarrow \\
U \times M & \phi \times \text{id} & V \times M \\
\downarrow & & \downarrow \\
\downarrow & & \downarrow \\
U \times M & \phi \times \text{id} & V \times M \\
\downarrow & & \downarrow \\
\downarrow & & \downarrow \\
U \times M & \phi \times \text{id} & V \times M \\
\downarrow & & \downarrow \\
\downarrow & & \downarrow \\
U \times M & \phi \times \text{id} & V \times M \\
\end{array}
\]

where \(\phi^\theta : U^B \rightarrow V^D\) is the function that sends \(h : B \rightarrow U\) to \(\phi \circ h \circ g : D \rightarrow V\).

This kind of morphism between machines (or models) generalizes the kind described in Definition 6 in the sense that it involves additional variation in the procedure-input and attribute-output sets (via the functions \(f, g\)). Notice the reverse direction of the function \(g\) between procedure-input sets; this differs from Goguen’s approach.

In a next step we define a category \(\text{HB}\) of “hybrid behaviours”. Objects of \(\text{HB}\) are 3-tuples \((A, B, s)\), where \(s\) is a function \(s : (B \times M)^* \times M \rightarrow A\). A morphism \((s : (B \times M)^* \times M \rightarrow A) \rightarrow (t : (D \times M)^* \times M \rightarrow C)\) in \(\text{HB}\) between two such behaviours will be defined as a pair of functions \(f : A \rightarrow C\), \(g : D \rightarrow B\) making the following diagram commute:

\[
\begin{array}{cccc}
(B \times M)^* \times M & \longrightarrow & (D \times M)^* \times M \\
\downarrow & & \downarrow \\
A & \longrightarrow & C \\
\end{array}
\]

This means that \(t = f^*(g \times \text{id}) \times \text{id}(s)\), i.e. that

\[
f(s((g(d_1), \beta_1), \ldots, (g(d_n), \beta_n)), \alpha)) = t((d_1, \beta_1), \ldots, (d_n, \beta_n)), \alpha)
\]
for all sequences \(\langle (d_1, \beta_1), \ldots, (d_n, \beta_n) \rangle \in (D \times M)^*_\ast\) and times \(x \in M\). These behaviours \(s: (B \times M)^*_\ast \times M \to A\) in \(HB\) are actually elements of the terminal model \(Z\) described in Proposition 11. Alternatively, one may use behaviours as in Proposition 12.

**Theorem 18.** There is a “behaviour” functor \(\mathcal{B}: HM \to HB\) from machines to behaviours, and in the reverse direction a “realisation” functor \(\mathcal{R}: HB \to HM\), forming an adjunction \((\mathcal{R} \dashv \mathcal{B})\) with \(\mathcal{B} \circ \mathcal{R} = \text{id}\).

In brief, the behaviour functor \(\mathcal{B}: HM \to HB\) sends a machine to the behaviour of its initial state, and the realization functor \(\mathcal{R}: HB \to HM\) sends a behaviour to a machine which realizes this behaviour. The fact that we have an adjunction tells us that this behaviour is realized in a minimal way. The adjunction involves for a machine \(m \in HM\) and a behaviour \(s \in HB\), a (natural) bijective correspondence

\[
\mathcal{B}(m) \to s \\
m \mapsto \mathcal{R}(s)
\]

between morphisms of behaviours \(\mathcal{B}(m) \to s\) and morphisms of machines \(m \to \mathcal{R}(s)\).

**Proof.** A machine (or model) \(\langle u_0, U \to A \times U^B \times U^M \rangle\) in \(HM\) yields a unique homomorphism \(\bar{U}: U \to A^{(B \times M)^*_\ast \times M}\) to the terminal model, see Proposition 11. Hence, we get a behaviour \(\bar{U}(u_0): (B \times M)^*_\ast \times M \to A\) in \(HB\) by applying this function \(\bar{U}\) to the initial state \(u_0 \in U\). Explicitly, this behaviour is given by

\[
\bar{U}(u_0)(\langle (b_1, \beta_1), \ldots, (b_n, \beta_n) \rangle, x) = u_0.\text{proc}(b_1)\beta_1.\ldots.\text{proc}(b_n)\beta_n.\text{at} x.
\]

This defines the behaviour functor \(\mathcal{B}: HM \to HB\) on objects. On morphisms \(\mathcal{B}\) is defined by \((f, g, \phi) \mapsto (f, g)\).

The realization functor \(\mathcal{R}: HB \to HM\) sends a behaviour \(s: (B \times M)^*_\ast \times M \to A\) to the machine made up from the terminal model in Proposition 11 with \(s\) as initial state:

\[
\mathcal{R}(s) = \langle s.A^{(B \times M)^*_\ast \times M} \to A \times (A^{(B \times M)^*_\ast \times M})^B \times (A^{(B \times M)^*_\ast \times M})^M \rangle.
\]

On morphisms one defines \(\mathcal{R}\) by \((f, g) \mapsto (f, g, f'(\cdot)\times\text{id})\).

In order to establish the adjunction \((\mathcal{B} \dashv \mathcal{R})\) we show that for a machine \(m = \langle u_0, U \to A \times U^B \times U^M \rangle \in HM\) and a behaviour \((t: (D \times M)^*_\ast \times M \to C) \in HB\) there is a bijective correspondence between morphisms

\[
\bar{U}(u_0) \xrightarrow{(f, g)} t
\]

in \(HB\), and morphisms

\[
\langle u_0, U \to A \times U^B \times U^M \rangle \xrightarrow{(f, g, \phi)} \mathcal{R}(t)
\]

in \(HM\). In the latter situation the function \(\phi: U \to C^{(B \times M)^*_\ast \times M}\) between state spaces is determined (by terminality) as \(\phi = f'(\cdot)\times\text{id} \circ \bar{U}: U \to A^{(B \times M)^*_\ast \times M} \to C^{(D \times M)^*_\ast \times M}\).
Hence \((f, g)\) is a morphism in (8) \(\Leftrightarrow f (g \times \text{id})^* \times \text{id}(\tilde{U}(u_0)) = t \Leftrightarrow \phi(u_0) = t \Leftrightarrow (f, g, \phi)\) is a morphism in (9). This gives us the adjunction \((\mathcal{B} \dashv \mathcal{H})\). □

6. Non-homogeneous specifications and models

In mathematical system theory, non-homogeneous differential equations model situations with a certain (continuous) input from the environment. So far we have only considered systems in which only discrete input from the environment (or from a client) can be processed, via procedures \text{proc} in a coalgebra. In this final section we briefly discuss “non-homogeneous” hybrid systems, in which there may be such a continuous input from the environment.

Our description of non-homogeneous hybrid systems will involve the following two changes.

1. At the syntactic level we shall change our notation: so far we used expressions like \(s: \text{at} @ s\) and \(s: \text{proc} (b) @ s\) describing a method application after a certain time delay \(s\). What we now need are expressions which also take the continuous input into account. We shall use new notation

\[
s: \text{at} @ (s, u) \quad \text{and} \quad s: \text{proc} (b) @ (s, u)
\]

where \(s\) is a time parameter as before, and \(u\) is a suitable input function with domain \([0, z] \subseteq \mathbb{R}_{\geq 0}\). It gives the input over the time interval \([0, z]\) during which we let the state \(s\) evolve, before applying the method at or \(\text{proc}(b)\).

2. At the model theoretic level we have been using monoid actions \(\mu: U \times \mathbb{R}_{\geq 0} \rightarrow U\) to describe the influence of time on a state space \(U\). In non-homogeneous systems we also have to accommodate for the influence of the continuous input \(u\) during the interval that we consider. This leads us to use “non-homogeneous” monoid actions

\[
\mu: \prod_{z \in \mathbb{R}_{\geq 0}} (U \times B^{[0, z]} \rightarrow U)
\]

(of three variables) satisfying

\[
\mu(0, x, u) = x \quad \text{and} \quad \mu(z + \beta, x, u) = \mu(\beta, \mu(x, x, u|_{[0, z]}), u(x + (-))|_{[0, \beta]}).
\]

(10)

In this situation the set \(B\) is a suitable subset \(B \subseteq \mathbb{R}^n\) describing the output type of the continuous (real-valued) input function \(u\).

The meaning of a “non-homogeneous” expression

\[
s: \text{meth} @ (s, u) \quad \text{is} \quad \text{meth}(\mu(x, s, u)).
\]

As an example, we shall describe a system like \(\text{REACT}_A\) in Fig. 4 involving a (decaying) chemical substance \(A\) in a reaction space, but this time with additional continuous input given by a function \(u(x): \mathbb{R}_{\geq 0}\) depending on a time parameter \(x \in \mathbb{R}_{\geq 0}\).

In this situation we have a non-homogeneous differential equation

\[
dA/dx = -kA + u
\]

(11)
RT-class spec: NH-REACT₄

methods:
amount: X → ℝ⁺
add: X × ℝ⁺₀ → X
clear: X → X

assertions:
s.add(a)@((x, u).amount@(0, v) = s.amount@(x, u) + a
s.clear@(x, u).amount@(0, v) = 0
s.amount@(x + β, u) = e⁻ᵏβ ⋅ (s.amount@(x, u) + ∫₀ β u(α + γ) ⋅ eᵏγ dγ)

creation:
new.amount@(0, u) = 0

end class spec

Fig. 11. Decay of chemical substance A with continuous input.

which, unlike the homogeneous equation (4), involves an extra factor u describing the input. The solution to Eq. (11) can be described explicitly as

\[ A(x) = e^{-kx} \left( A(0) + \int_0^x u(\gamma) \cdot e^{k\gamma} d\gamma \right). \]

(Assuming that u is integrable.) This equation is incorporated into the non-homogeneous class specification NH-REACT₄ in Fig. 11.

We describe a model of this specification NH-REACT₄ with a state space \( U = ℝ⁺ \) describing the present amount of the substance A in the reaction container. The interpretations of the methods amount, add and clear are as in the proof of Proposition 9. That is, \( x,.amount = x \), \( x,.add(a) = x + a \) and \( x,.clear = 0 \), for a state \( x \in U = ℝ⁺₀ \). Most interestingly is the non-homogenous monoid action

\[ \mu : \prod_{x \in ℝ⁺₀} \left( ℝ⁺₀ \times ℝ⁺₀ \rightarrow ℝ⁺₀ \right) \text{ given by } \]

\[ \mu(x, x, u) = e^{-kx} \left( x + \int_0^x u(\gamma) \cdot e^{k\gamma} d\gamma \right). \]

Clearly, we shall have to restrict ourselves to integrable input functions u. We check that \( \mu \) satisfies the non-homogenous monoid equations (10). The first equation \( \mu(0, x, u) = x \) is obvious, and

\[ \mu(β, μ(x, x, u), u(x + (-))) \]

\[ = e^{-kβ} \left( μ(x, x, u) + \int_0^β u(x + γ) \cdot e^{kγ} dγ \right) \]
\[ 
= e^{-k\beta} \left( e^{-kx} \cdot \left( x + \int_0^x u(\gamma) \cdot e^{k\gamma} \, d\gamma \right) + \int_x^{x+\beta} u(\gamma) \cdot e^{k(\gamma-x)} \, d\gamma \right) 
\]

\[ 
= e^{-k\beta} \left( e^{-kx} \left( x + \int_0^x u(\gamma) \cdot e^{k\gamma} \, d\gamma \right) + e^{-kx} \int_x^{x+\beta} u(\gamma) \cdot e^{k\gamma} \, d\gamma \right) 
\]

\[ 
= e^{-k(x+\beta)} \left( x + \int_0^{x+\beta} u(\gamma) \cdot e^{k\gamma} \, d\gamma \right) 
\]

\[ 
= \mu(x+\beta,x,u). 
\]

It is not hard to check that this structure \(\mu\), amount, add, clear on the state space \(\mathbb{R}_{\geq 0}\) satisfies the assertions in the specification in Fig. 11.

These non-homogeneous systems will be further studied elsewhere. This study will involve a suitable notion of homomorphism of such systems, for which one can show, for example, that this NH-REACT model \(\mathbb{R}_{\geq 0}\) is in fact the terminal model. And also it will involve connections between the continuous outputs of one system and continuous inputs of another system. This should extend the modular object-oriented approach to more complex non-homogeneous hybrid systems.

References


