An Adequate While-Language for Hybrid Computation

Sergey Goncharov
sergey.goncharov@fau.de
Friedrich-Alexander Universität Erlangen-Nürnberg, Germany

Renato Neves
nevrenato@di.uminho.pt
INESC TEC (HASLab) & University of Minho, Portugal

ABSTRACT

Hybrid computation harbours discrete and continuous dynamics in the form of an entangled mixture, inherently present in various natural phenomena and in applications ranging from control theory to microbiology. The emergent behaviours bear signs of both computational and physical processes, and thus present difficulties not only in their analysis, but also in describing them adequately in a structural, well-founded way.

In order to tackle these issues and, more generally, to investigate *hybridness* as a dedicated computational phenomenon, we introduce a while-language for hybrid computation inspired by the *fine-grain call-by-value* paradigm. We equip it with *operational* and *computationally adequate denotational* semantics. The latter crucially relies on a *hybrid monad* supporting an (Elgot) iteration operator that we developed elsewhere. As an intermediate step, we introduce a more lightweight *duration semantics* furnished with analogous results and based on a new *duration monad* that we introduce as a lightweight counterpart to the hybrid monad.

CCS CONCEPTS

• **Theory of computation → Timed and hybrid models**.

KEYWORDS

Elgot iteration, guarded iteration, hybrid monad, Zeno behaviour, hybrid system, operational semantics.

ACM Reference Format:
Sergey Goncharov and Renato Neves. 2019. An Adequate While-Language for Hybrid Computation. In *PPDP'19: 21st International Symposium on Principles and Practice of Declarative Programming, October 07–09, 2019, Porto, Portugal.* ACM, New York, NY, USA, 15 pages.

1 INTRODUCTION

Motivation and context. Hybrid systems are traditionally seen as computational devices that closely interact with physical processes, such as movement, temperature, energy, and time. Their wide range of applications, including e.g. the design of embedded systems [36] and analysis of disease propagation [28], led to an extensive research in the area, with a particular focus on languages and models that suitably accomodate classic computations and physical processes at the same time [7, 21, 36]. Most of this research revolved around the notion of *hybrid automaton*, widely adopted in control theory and in computer science [4, 21], and around certain classes of *(weak) Kleene algebra*, specialised at verifying safety and liveness properties of hybrid systems [22, 36].

A distinctive aspect of these models is the free use of non-determinism, which semantically amounts to *non-deterministic hybrid systems* rather than *(purely) hybrid systems*, the latter abundantly found in the wild. Furthermore, iterative computations in these models are captured by Kleene star (essentially the non-deterministic choice between all possible finite iterates of the system at hand), i.e. non-determinism is also intertwined with recursion. In this paper, we take a different view: we see non-determinism as an *abstraction layer* added on top of *hybridness* for handling uncertainty. This consideration can be made precise by drawing on Moggi's seminal work [31] associating *computational effects* with *(strong) monads*. Both non-determinism and hybridness are computational effects in Moggi's sense: while the former effect is standard, the latter was identified recently [34] and elaborated in the second author's PhD thesis [33].

Our view of (pure) hybridness as a computational effect connects the area of hybrid systems with a vast body of research on monad-based programming theory (e.g. [14, 27, 31]), which potentially eases the development of sophisticated *hybrid programming languages* – a task that arguably belongs to the core of the twenty-first century's technology, due to the increasing presence of software products that interact with physical processes [7, 25]. As a step in the direction of such languages, we introduce a deterministic while-language for hybrid computation HybCore. We do not intend to use it directly in the development of complex hybrid systems, but rather as a basic language on which to study the subject of *(purely) hybrid computation*, possible extensions of the latter, and as a theoretical basis for the development of more complex hybrid programming languages.

Contributions and overview of HybCore. In the development of HybCore, we concentrate on the indispensable features of hybrid computation, namely classic program constructs and primitives for describing behaviours of physical processes. As we will see, the orchestration of these two aspects yields significant challenges, centring on the main construct of HybCore – the *hybrid while-loop* – which captures iterated hybrid behaviour, including both finite and infinite loop unfoldings, most notably *Zeno behaviour*. Figure 1 shows a taxonomy of while-loops emerging from HybCore: besides the standard possibilities of *convergence* and *divergence*, we now distinguish between *progressive*, *non-progressive*, and Zeno behaviour, depending on the *execution time of computations* (in Figure 1 the *wait*(s) command, triggers a delay of s time units).

PPDP'19, October 07–09, 2019, Porto, Portugal

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
DOI: 10.1145/3354166.3354176

$x \coloneqq 5\ \mathsf{while}\ x > 0\ \{x \coloneqq x - 1\}$

(a) Convergent non-progressive loop

$x \coloneqq 5\ \mathsf{while}\ x > 0\ \{\mathit{wait}(1);\ x \coloneqq x - 1\}$

(b) Convergent progressive loop

$x \coloneqq 0\ \mathsf{while\ true}\ \{x \coloneqq x + 1\}$

(c) Divergent non-progressive loop

$x \coloneqq 0\ \mathsf{while\ true}\ \{\mathit{wait}(1);\ x \coloneqq x + 1\}$

(d) Divergent progressive loop

$x \coloneqq 1\ \mathsf{while\ true}\ \{\mathit{wait}(x);\ x \coloneqq x/2\}$

(e) Zeno behaviour

Figure 1: Taxonomy of hybrid while-loops

We emphasise the distinction between our semantics and the more widespread *transition semantics* for hybrid automata [21]. Following Abramsky [1], the latter can be qualified as an *intensional semantics*, while we aim at an *extensional semantics*, meaning that e.g. we regard the two programs

$$x \coloneqq 1\ \mathsf{while\ true}\ \{\mathit{wait}(1)\} \quad \text{and} \quad x \coloneqq 1\ \mathsf{while\ true}\ \{\mathit{wait}(2)\}$$

as equivalent, while they would be distinct under the transition semantics. In a nutshell, by committing to the extensional view, we abstract away from specific choices of control states – a decision that could also be interpreted in hybrid automata setting and thus potentially lead to coarser, more flexible notions of automata equivalence.

One principal design decision we adopt in HybCore is to base it on the *fine-grain call-by-value* paradigm [27] – a refinement of Moggi's computational λ-calculus, suitable for both operational and denotational semantics. Denotationally, we associate a hybrid computation with a *trajectory*, i.e. a map from a downward closed set D of non-negative real numbers (with $D = \emptyset$ for the *empty trajectory* representing non-progressive divergence). The supremum of D is called the *duration* of the hybrid computation. As shown in previous work [16, 17], such trajectories constitute a *hybrid monad* $\mathbb{H}$.

We introduce an operational semantics for HybCore and show that it is sound and (computationally) adequate w.r.t. $\mathbb{H}$. This result relies on a more lightweight operational semantics that keeps track of durations (i.e. execution times) only. We complement this latter semantics with a *duration monad* and prove analogous soundness and adequacy results for it.

Our results do not fit standard semantic frameworks for the following reasons:

- It is not possible to introduce iteration for the duration monad (as well as for the hybrid monad [16, 17]) via the usual least fixpoint argument, essentially due to the lack of canonical choice of divergence; the strategy we employ here, is thus to introduce a certain *coalgebraic* version of this monad equipped with a partial (guarded) Elgot iteration and then to suitably quotient it.
- Progressive divergence and Zeno behaviour, are principally *non-inductive* kinds of behaviour, which we manage to capture by allowing for infinitely many premises in the operational semantics rules.
- Our semantics for HybCore is essentially two-stage, and contains the aforementioned duration semantics as an auxiliary ingredient. The operational semantics rules thus feature two separate evaluation judgements.

Related work. In designing the different constructs of HybCore, we owe inspiration to Witsenhausen's model of differential equations indexed by operations modes, a percursor of the concept of hybrid system [44], the *relational* Kleene algebra underlying *differential dynamic logic* [36] and also hybrid automata [21]. As already mentioned, we restrict ourselves to a purely hybrid setting, in particular we exclude non-determinism. This is not to say that we could not have considered non-determinism (in fact, there is also a non-deterministic hybrid monad [12]) but for the time being we focus on pure hybrid behaviour in isolation from other computational effects, such as non-determinism.

Suenaga and Hasuo [39] develop a somewhat similar while-language for hybrid systems but from a different semantic viewpoint: by unifying both continuous and discrete cyclic behaviour in a single while-construct that makes incrementations by infinitesimals. An interesting, principled approach to the semantics of *timed systems*, via an *evolution comonad*, is proposed by Kick, Power and Simpson [24]. The underlying functor of the evolution comonad is striking close to the functor underlying our hybrid monad, except that the former does not include empty trajectories and hence does not support non-progressive divergence.

As indicated above, in order to interpret hybrid while-loops, we necessarily went beyond standard semantic frameworks [38, 43], in particular because of the indispensable presence of various notions of divergence. This relates our work to the recent work on modelling productively non-terminating programs using a coalgebraic *delay monad* (previously called the *partiality monad*) [8, 13]. Indeed, our construction of the iteration operator on the *duration monad* in Section 5, is inspired by a technically similar procedure of quotienting the delay monad by weak bisimilarity [9], albeit in a different (constructive) setting.

2 PRELIMINARIES AND NOTATION

We will generally work in the category **Set** of sets and functions, but also call on some standard idioms of category theory [5, 29]. By $|\mathbf{C}|$ we refer to the objects of $\mathbf{C}$, and by $\mathsf{Hom}_{\mathbf{C}}(A, B)$ (or $\mathsf{Hom}(A, B)$, if no confusion arises) to the morphisms $f\colon A \to B$ from $A \in |\mathbf{C}|$ to $B \in |\mathbf{C}|$. We often omit indices at natural transformations. By $\mathbb{N}$, $\mathbb{R}$, $\mathbb{R}_+$ and $\bar{\mathbb{R}}_+$ we denote natural numbers (including 0), real numbers, non-negative real numbers and extended non-negative real numbers $\mathbb{R}_+ \cup \{\infty\}$ respectively. By $X + Y$ we denote a coproduct of X and Y, the corresponding coproduct injections $\mathsf{inl}\colon X \to X + Y$ and $\mathsf{inr}\colon Y \to X + Y$, and dually for products: $\mathsf{fst}\colon X \times Y \to X$, $\mathsf{snd}\colon X \times Y \to Y$. A category with finite products and coproducts is *distributive* [10] if the natural transformation $[\mathrm{id}\times\mathsf{inl}, \mathrm{id}\times\mathsf{inr}]\colon X\times Y+X\times Z \to X\times(Y+Z)$ is an isomorphism, whose inverse we denote by dist. By $p \triangleleft b \triangleright q$ we abbreviate the if-then-else operator expanding as follows: $p \triangleleft \top \triangleright q = p$, $p \triangleleft \bot \triangleright q = q$. For time-dependent functions $e\colon \mathbb{R} \to Y$ we use the superscript notation e^t in parallel with $e(t)$ to emphasize orthogonality of the temporal dimension to the spatial one.

Following Moggi [31], we identify a *monad* $\mathbb{T}$ on a category $\mathbf{C}$ with the corresponding *Kleisli triple* $(T, \eta, (-)^\star)$ consisting of an endomap T on $|\mathbf{C}|$, a $|\mathbf{C}|$-indexed class of morphisms $\eta_X\colon X \to TX$, called the *unit* of $\mathbb{T}$, and the *Kleisli lifting* maps $(-)^\star\colon \mathsf{Hom}(X, TY) \to \mathsf{Hom}(TX, TY)$ such that $\eta^\star = \mathrm{id}$, $f^\star\eta = f$, $(f^\star g)^\star = f^\star g^\star$. Provided that $\mathbf{C}$ has finite products, a monad $\mathbb{T}$ on $\mathbf{C}$ is *strong* if it is equipped with *strength*, i.e. a natural transformation $\tau_{X,Y}\colon X \times TY \to T(X \times Y)$ satisfying a number of standard coherence conditions (see e.g. [31]). Every monad $\mathbb{T}$ on **Set** is strong [26] under $\tau_{X,Y} = \lambda\langle x, p\rangle.\, T(\lambda y.\, \langle x, y\rangle)(p)$. Morphisms of the form $f\colon X \to TY$ form the *Kleisli category* of $\mathbb{T}$, which has the same objects as $\mathbf{C}$, units $\eta_X\colon X \to TX$ as identities, and composition $(f, g) \mapsto f^\star g$, also called *Kleisli composition*.

An *(F-)coalgebra* on $X \in |\mathbf{C}|$ for a functor $F\colon \mathbf{C} \to \mathbf{C}$ is a pair $(X, f\colon X \to FX)$. Coalgebras form a category whose morphisms $h\colon (X, f) \to (Y, g)$ satisfy $h \in \mathsf{Hom}_{\mathbf{C}}(X, Y)$ and $(Fh)f = gh$. A final object $(\nu F, \mathsf{out}\colon \nu F \to F\nu F)$ of this category is called a *final coalgebra* (see [42] for more details on coalgebras in semantics of (co)iteration).

3 A WHILE-LANGUAGE FOR HYBRID COMPUTATION

We start off by defining our core language for hybrid computation HybCore, following the *fine-grain call-by-value* paradigm [27]. First, we introduce *types*:

$$A, B, \ldots ::= \mathbb{N} \mid \mathbb{R} \mid 1 \mid 2 \mid A \times B \tag{1}$$

and then postulate a signature Σ of typed operation symbols of the form $f\colon A \to B$ with $B \in \{\mathbb{N}, \mathbb{R}, 2\}$ for integer and real arithmetic, e.g. summation $+\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, and further primitives meant to capture time-dependent solutions of systems of ordinary differential equations: i.e. for every such system $\{\dot{x}_1 = f_1(\bar{x}, t), \ldots, \dot{x}_n = f_n(\bar{x}, t)\}$ where $\bar{x}$ is the vector $\langle x_1, \ldots, x_n\rangle$ and $f_i\colon \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$, the signature Σ contains the symbol $\mathit{int}^i_{f_1,\ldots,f_n}\colon \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$, representing the i-th projection of the corresponding unique (global) solution [35].

Remark 1. *We do not study conditions ensuring existence of the solutions* $\mathit{int}^i_{f_1,\ldots,f_n}$*, and instead work under the assumption that the functions* f_i *come from a sufficiently well-behaved class for which these solutions exist. Clearly, we could instead introduce a specific grammar of differential equations and make them part of the language. For example, we could use the following grammar for the* $f_i(\bar{x}, t)$*:*

$$\begin{aligned} &f(\bar{x}, t), g(\bar{x}, t), \ldots ::= \\ &\qquad c \in \mathbb{R} \mid t \mid x \in \bar{x} \mid f(\bar{x}, t) + g(\bar{x}, t) \mid f(t) \cdot g(\bar{x}, t). \end{aligned}$$

where $f(t)$ *reads as* $f(\bar{x}, t)$ *with empty* $\bar{x}$*. This corresponds to systems of* linear ordinary differential equations*, which provide a sufficiently large stock of trajectories standardly used in hybrid system modelling [3]. A concrete choice of such grammar however would have no bearing on our present results, but we expect to revisit this decision in future work devoted to reasoning and verification in* HybCore*.*

Our language features two kinds of judgement,

$$\Gamma \vdash_{\mathsf{v}} v\colon A \qquad \text{and} \qquad \Gamma \vdash_{\mathsf{c}} p\colon A \tag{2}$$

for *values* and *computations*, respectively. These involve *variable contexts* Γ which are non-repetitive lists of *typed variables* $x\colon A$. We indicate the *empty context* as $-$ (dash), e.g. $- \vdash_{\mathsf{c}} p\colon A$. The term language over these data is given in Figure 2. Programs of HybCore figure as terms p in computation judgements $\Gamma \vdash_{\mathsf{c}} p\colon A$ and are inductively built over value judgements. Intuitively, the latter capture functions from Γ to A in the standard mathematical sense, and computation judgements capture functions from Γ to *trajectories* valued on A; i.e. instead of single values from A, they return *functions* $D \to A$ valued on A with $D = [0, d]$ or $D = [0, d)$, e.g. $\emptyset = [0, 0)$ and $\mathbb{R}_+ = [0, \infty)$. Most of the constructs in Figure 2 are standard. Sequential composition **(seq)** for example reads as “bind the output of p to x and then feed it to q”. The rule **(now)** converts a value v into an instantaneous computation, $[0, 0] \to A$, that returns v. The rules **(if)** and **(wh)** provide basic control structures (cf. [43]). Finally, **(tr)** converts a value into a computation by simultaneously abstracting over time and restricting the duration of the computation to the *union* of all those intervals $[0, d]$ on which v holds throughout. Note that the resulting interval can thus be either open or closed on the right. Intuitively, one can see **(tr)** as a ‘continuous’ while-loop since the trajectory captured by $t.\, v$ runs as long as the condition w is satisfied. For example, the computation judgement $\Gamma \vdash_{\mathsf{c}} x \coloneqq t.\, t \mathbin{\&} t \leq 7\colon \mathbb{R}$ models the passage of time and runs whilst the latter does not surpass seven time units. This behaviour captures the essence of hybrid behaviour [44], which is now accommodated by hybrid automata and Platzer's differential dynamic logic; our notation ‘&’ alludes precisely to this corresponding operator of the latter framework.

In contrast to fine-grain call-by-value [27], we have no conversions from computations to values, for we do not assume existence of higher order types, which potentially provide a possibility to *suspend* a program as a value and subsequently *execute* it, resulting in a computation. Adding these facilities to HybCore can be done standardly, but we refrain from it, as we intend our semantics to be eventually transferable from **Set** to more suitable universes, which need not be Cartesian closed, such as the category of topological spaces **Top**.

Next we introduce some syntactic conventions for HybCore programs:

- We write $x \coloneqq t.\, v \mathbin{\&} r$ with $r \in \mathbb{R}$ for $\langle y, t\rangle \coloneqq (\, x \coloneqq t.\, \langle v, t\rangle \mathbin{\&} \mathsf{snd}\, x \leq r); \lceil y \rceil$ (assuming that the projection $\mathsf{snd}\colon A \times B \to B$ and the predicate $\leq$ are in Σ). Intuitively, this restricts the trajectory encoded by $t.\, v$ to the interval $[0, r]$.
- Sequential composition is right associative, that is, an iterated sequential composition in the form $x_1 \coloneqq p_1; \ldots; x_n \coloneqq p_n; q$ should be parsed as $x_1 \coloneqq p_1; (\ldots; (x_n \coloneqq p_n; q) \ldots)$.
- We allow for programs of the form $x \coloneqq p$ which abbreviate $x \coloneqq p; \lceil x \rceil$.
- We omit the brackets $\lceil - \rceil$ if they can be easily reconstructed. E.g. $x \coloneqq x + 1$ means $x := \lceil x + 1 \rceil$ and thus $x := \lceil x + 1 \rceil; \lceil x \rceil$, by the previous clause (e.g. this is used in Figure 1).

(unit) $$\frac{}{\Gamma \vdash_{\mathsf{v}} \star : 1}$$

(var) $$\frac{x : A \text{ in } \Gamma}{\Gamma \vdash_{\mathsf{v}} x : A}$$

(sig) $$\frac{f : A \to B \in \Sigma \qquad \Gamma \vdash_{\mathsf{v}} v : A}{\Gamma \vdash_{\mathsf{v}} f(v) : B}$$

(true) $$\frac{}{\Gamma \vdash_{\mathsf{v}} \mathsf{true} : 2}$$

(false) $$\frac{}{\Gamma \vdash_{\mathsf{v}} \mathsf{false} : 2}$$

(prod) $$\frac{\Gamma \vdash_{\mathsf{v}} v : A \qquad \Gamma \vdash_{\mathsf{v}} w : B}{\Gamma \vdash_{\mathsf{v}} \langle v, w \rangle : A \times B}$$

(tr) $$\frac{\Gamma, t : \mathbb{R} \vdash_{\mathsf{v}} v : A \qquad \Gamma, x : A \vdash_{\mathsf{v}} w : 2}{\Gamma \vdash_{\mathsf{c}} x \mathrel{:=} t.\, v \mathbin{\&} w : A}$$

(seq) $$\frac{\Gamma \vdash_{\mathsf{c}} p : A \qquad \Gamma, x : A \vdash_{\mathsf{c}} q : B}{\Gamma \vdash_{\mathsf{c}} x \mathrel{:=} p; q : B}$$

(pm) $$\frac{\Gamma \vdash_{\mathsf{v}} v : A \times B \qquad \Gamma, x : A, y : B \vdash_{\mathsf{c}} p : C}{\Gamma \vdash_{\mathsf{c}} \langle x, y \rangle \mathrel{:=} v; p : C}$$

(now) $$\frac{\Gamma \vdash_{\mathsf{v}} v : A}{\Gamma \vdash_{\mathsf{c}} \lceil v \rceil : A}$$

(if) $$\frac{\Gamma \vdash_{\mathsf{v}} v : 2 \qquad \Gamma \vdash_{\mathsf{c}} p : A \qquad \Gamma \vdash_{\mathsf{c}} q : A}{\Gamma \vdash_{\mathsf{c}} \mathsf{if}\ v\ \mathsf{then}\ p\ \mathsf{else}\ q : A}$$

(wh) $$\frac{\Gamma \vdash_{\mathsf{c}} p : A \qquad \Gamma, x : A \vdash_{\mathsf{v}} w : 2 \qquad \Gamma, x : A \vdash_{\mathsf{c}} q : A}{\Gamma \vdash_{\mathsf{c}} x \mathrel{:=} p\ \mathsf{while}\ w\ \{q\} : A}$$

Figure 2: HybCore’s term formation rules for values (top) and computations (bottom)

Let us examine two examples of hybrid systems programmable in HybCore.

Example 3.1 (Bouncing ball). Consider the system of differential equations $\{\dot{u} = v,\ \dot{v} = \mathrm{g}\}$ describing continuous movement of a ball in terms of its *height* u and *velocity* v with g being Earth’s gravity. We postulate the corresponding solutions $\mathit{ball}_u : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}$ and $\mathit{ball}_v : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}$ in Σ so that $\mathit{ball}_u(x, y, t)$ and $\mathit{ball}_v(x, y, t)$ are the height and the velocity of the ball at time t, assuming x and y as the corresponding initial values (at time 0). Let ball be $\langle \mathit{ball}_u, \mathit{ball}_v \rangle$, and then e.g. $\langle u, v \rangle \mathrel{:=} t.\, \mathit{ball}(5, 0, t) \mathbin{\&} u \geq 0$ encodes the behaviour of the ball starting from the moment when it is dropped from height 5 until it hits the ground. The HybCore program describing the resulting bouncing behaviour with infinitely many iterations is as follows:

```
$\langle u, v \rangle := \langle 5, 0 \rangle$
while true {
  $\langle u, v \rangle := t.\, \mathit{ball}(u, v, t)\ \&\ u \geq 0;$
  $\langle u, v \rangle := \langle u, -0.5v \rangle$
}
```

The movement thus programmed is depicted in Figure 3a. Note that the height is decreasing, tending to zero in the limit. This requires infinitely many iterations, but the process is finite in terms of physical time, because each iteration gets progressively shorter. This yields an example of Zeno behaviour [16, 23, 32], which is often dismissed in the hybrid systems literature, as justified by the fact that such behaviour is on the one hand not computationally realisable, and on the other hand notoriously difficult to analyse. Here, we commit ourselves to the task of faithful modelling Zeno behaviour as an inherent feature of physical, biological and other systems.

Example 3.2 (A simplistic cruise controller). The following program implements a simplistic cruise controller programmed in HybCore:

```
$\langle u, v \rangle := \langle 0, 0 \rangle$
while true {
  if $v \leq 120$ then $\langle u, v \rangle := t.\, \mathit{accel}(u, v, t)\ \&\ 3$
                 else $\langle u, v \rangle := t.\, \mathit{brake}(u, v, t)\ \&\ 3$
}
```

Here, $\mathit{accel} = \langle \mathit{accel}_u, \mathit{accel}_v \rangle$ and $\mathit{brake} = \langle \mathit{brake}_u, \mathit{brake}_v \rangle$ refer to the solutions of the systems of differential equations $\{\dot{u} = v, \dot{v} = 1\}$, $\{\dot{u} = v, \dot{v} = -1\}$ correspondingly. As the names suggest, accel is responsible for the acceleration, while brake is responsible for braking. Each iteration in the loop has the duration of three time units (due to the expression ‘& 3’); at the beginning of each iteration the controller checks the current velocity, leading the vehicle to the target velocity of 120 km/h. This produces the oscillation pattern depicted in Figure 3b.

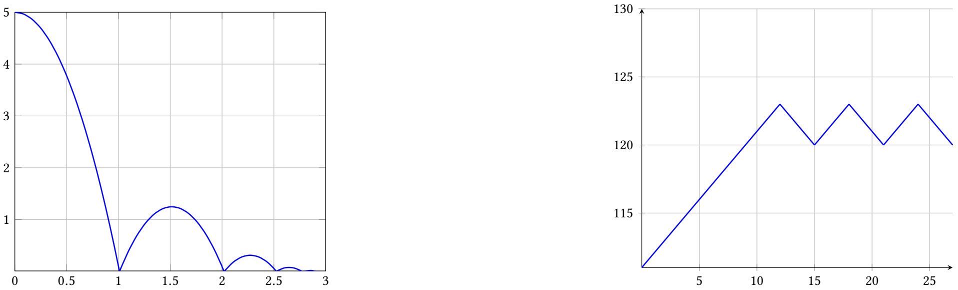
Example 3.3 (Signal sampling). The program below discretises a continuous-time signal $\mathit{signal} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. The variable u, which is valued on natural numbers $\mathbb{N}$, represents the sampled signal, and the operation $\mathsf{round} : \mathbb{R} \to \mathbb{N}$ rounds the input up to the closest natural number.

```
$\langle u, v \rangle := \langle 0, 0 \rangle$
while true {
  $\langle u, v \rangle := t.\, \langle \mathsf{round}(v), \mathit{signal}(v, t) \rangle\ \&\ 1$
}
```

The resulting computation when projected to the second Cartesian factor yields the original signal, while the same computation projected to the first Cartesian factor yields the corresponding discretised, or *sampled* signal. Here, we used the unit 1 as a *sampling interval.*

4 OPERATIONAL DURATION SEMANTICS

We now introduce an *operational duration semantics* of HybCore, both in small-step and big-step styles. In order to help build intuitions, we begin with the former. Duration semantics abstract away from the run-time behaviour of the program (i.e. its trajectory), and only provide the execution time and possibly the final result



(a) Bouncing ball height

(b) Vehicle velocity

Figure 3: Trajectories produced by hybrid programs

(i.e. the endpoint of the trajectory). The small-step judgements are defined on closed computations terms and have the form,

$$p \xrightarrow{d} q \qquad \text{and} \qquad p \xrightarrow{e} \qquad\qquad (d \in \mathbb{R}_+, e \in \bar{\mathbb{R}}_+)$$

meaning that p one-step reduces to q in time d, or p one-step diverges in time e respectively, where only in the latter case the relevant time interval may be infinite. The derivation rules for these judgements are shown in Figure 4. These rules are essentially obtained by extending the standard call-by-value small-step semantics [43] by decorating the transitions with durations. The most distinctive rules are the ones addressing the abstraction construct **(tr)**: intuitively, the resulting durations d are computed as the lengths of the largest interval $[0, t]$ on which the trajectory produced by v satisfies the predicate w throughout. The terminal value $\lceil v[d/t] \rceil$ is available precisely when the trajectory produced by v satisfies the predicate w at d. For example, for $\mathit{line}\colon \mathbb{R}\times\mathbb{R} \to \mathbb{R}$ the global solution of the differential equation $\dot{x} = 1$, the program $x := t.\, \mathit{line}(0, t) \ \&\ x \leq 1$ one-step reduces to $\lceil 1 \rceil$ in time 1, and on the other hand, $x := t.\, \mathit{line}(0, t) \ \&\ x < 1$ one-step diverges in time 1.

Note a certain arbitrariness in defining *instantaneous transitions* $\xrightarrow{0}$. E.g. we could just as well use the following slightly more economical rule for reducing $\texttt{if true then } p \texttt{ else } q$:

$$\frac{p \xrightarrow{d} p'}{\texttt{if true then } p \texttt{ else } q \xrightarrow{d} p'}$$

This would however not change the principal fact of pervasiveness of intermediate instantaneous transitions, as they are also triggered by unfolding while-loops, and cannot be completely eliminated from the corresponding rules.

We proceed to introduce the aforementioned big-step duration semantics. It associates a closed computation term p either with a pair d, v, consisting of a *duration* $d \in \mathbb{R}_+$ and a (closed) *terminal value* v, or with a (possibly infinite) duration $d \in \bar{\mathbb{R}}_+$. The latter case occurs e.g. if p exhibits Zeno behaviour or more generally if p is a while-loop that unfolds infinitely many times. The rules for big-step judgements are given in Figure 5.

Again, we focus only on the least standard rules. In both $(\mathbf{tr}^{\mathbf{d}}_1)$ and $(\mathbf{tr}^{\mathbf{d}}_2)$ the duration is identified with the length of the largest interval $[0, d]$ on which the condition w holds throughout, and subsequently each rule verifies if v is true at $t = d$. The rules $(\mathbf{seq}^{\mathbf{d}}_1)$, $(\mathbf{seq}^{\mathbf{d}}_2)$ and $(\mathbf{seq}^{\mathbf{d}}_3)$ capture the fact that durations of subsequent computations add up, unless the terminal value of the first computation is undefined, in which case the total duration of the program is the duration of the first subprogram. This behaviour is inherited inductively by while-loops via $(\mathbf{wh}^{\mathbf{d}}_1)$–$(\mathbf{wh}^{\mathbf{d}}_4)$, and in addition we have a *non-inductive case* of infinitely many unfoldings of while-loops, as captured by the (infinitary) rule $(\mathbf{wh}^{\mathbf{d}}_5)$. This allows for computing the duration of the program as the infinite sum $\sum_i d_i$.

Remark 2. *The rules $(\mathbf{wh}^{\mathbf{d}}_3)$ and $(\mathbf{wh}^{\mathbf{d}}_4)$ directly capture the inductive intuition in the semantics of while-loops, but notably these are the only rules which are not structural, in the sense that the programs in the premises are not structurally simpler than the corresponding programs in the conclusions. It is easy to see that $(\mathbf{wh}^{\mathbf{d}}_3)$ and $(\mathbf{wh}^{\mathbf{d}}_4)$ can be equivalently replaced by the families of structural rule schemes presented in Figure 6 in the style of $(\mathbf{wh}^{\mathbf{d}}_5)$.*

Proposition 4.1 (Determinacy). *The semantics in Figure 5 is deterministic, that is for every closed computation term p, there is (exclusively!) at most one judgement of the form $p \Downarrow d, v$ or $p \Downarrow d$ derivable using the rules of Figure 5.*

Proof. The proof runs routinely by induction over the number of computation constructs (bottom part of Figure 2) in p and by inspecting the rules in Figure 5, except for $(\mathbf{wh}^{\mathbf{d}}_3)$ and $(\mathbf{wh}^{\mathbf{d}}_4)$, which must be replaced by their structural analogues per Remark 2. □

In order to relate small-step with big-step semantics, we define $p \overset{d}{\Longrightarrow} q$ and $p \overset{d}{\Longrightarrow}$ inductively by the following small-step judgements and closure rules:

$$\frac{}{\lceil v \rceil \overset{0}{\Longrightarrow} \lceil v \rceil} \qquad \frac{p \xrightarrow{d} q}{p \overset{d}{\Longrightarrow} q} \qquad \frac{p \overset{d}{\Longrightarrow} p' \quad p' \xrightarrow{e} q}{p \overset{d+e}{\Longrightarrow} q} \qquad \frac{p \xrightarrow{d}}{p \overset{d}{\Longrightarrow}}$$

$$\frac{p \overset{d}{\Longrightarrow} p' \qquad p' \xrightarrow{e}}{p \overset{d+e}{\Longrightarrow}} \qquad \frac{p \xrightarrow{d_1} p' \qquad p' \xrightarrow{d_2} p'' \qquad \ldots}{p \overset{d_1 + d_2 + \ldots}{\Longrightarrow}}$$

The binary relation $\overset{d}{\Longrightarrow}$ is then essentially the standard transitive closure, modulo the fact that durations are added along the way. The unary relation $\overset{d}{\Longrightarrow}$ is derivable both from finite and infinite numbers of premises, e.g. in the Zeno behaviour case, as in Example 3.1.

Closed values, Closed computations:

$$v, w ::= x \mid \star \mid \mathsf{true} \mid \mathsf{false} \mid \langle v, w\rangle \mid f(v) \qquad (f \in \Sigma)$$

$$p, q ::= \langle x, y\rangle := \langle v, w\rangle; p \mid \mathsf{if}\ v\ \mathsf{then}\ p\ \mathsf{else}\ q \mid \lceil v\rceil \mid x := p; q \mid x := p\ \mathsf{while}\ v\ \{q\} \mid x := t.\, v \ \&\ w$$

Rules:

$$\dfrac{}{\langle x, y\rangle := \langle v, w\rangle; q \xrightarrow{0} q[v/x, w/y]} \qquad \dfrac{p \xrightarrow{d} p'}{x := p; q \xrightarrow{d} x := p'; q} \qquad \dfrac{p \xrightarrow{d}}{x := p; q \xrightarrow{d}}$$

$$\dfrac{}{x := \lceil v\rceil; q \xrightarrow{0} q[v/x]} \qquad \dfrac{}{\mathsf{if\ true\ then}\ p\ \mathsf{else}\ q \xrightarrow{0} p} \qquad \dfrac{}{\mathsf{if\ false\ then}\ p\ \mathsf{else}\ q \xrightarrow{0} q}$$

$$\dfrac{\forall s \leq d.\, w[v[s/t]/x] = \mathsf{true} \qquad \forall e > 0.\, \exists s \in (d, d+e).\, w[v[s/t]/x] = \mathsf{false}}{x := t.\, v \ \&\ w \xrightarrow{d} \lceil v[d/t]\rceil} \qquad \dfrac{p \xrightarrow{d}}{x := p\ \mathsf{while}\ v\ \{q\} \xrightarrow{d}}$$

$$\dfrac{\forall s < d.\, w[v[s/t]/x] = \mathsf{true} \qquad w[v[d/t]/x] = \mathsf{false}}{x := t.\, v \ \&\ w \xrightarrow{d}} \qquad \dfrac{w[v/x] = \mathsf{false}}{x := \lceil v\rceil\ \mathsf{while}\ w\ \{q\} \xrightarrow{0} \lceil v\rceil}$$

$$\dfrac{p \xrightarrow{d} p'}{x := p\ \mathsf{while}\ v\ \{q\} \xrightarrow{d} x := p'\ \mathsf{while}\ v\ \{q\}} \qquad \dfrac{w[v/x] = \mathsf{true}}{x := \lceil v\rceil\ \mathsf{while}\ w\ \{q\} \xrightarrow{0} x := q[v/x]\ \mathsf{while}\ w\ \{q\}}$$

Figure 4: Small-step duration operational semantics

Theorem 4.2. *For every closed computation term p, $p \xRightarrow{d} \lceil v\rceil$ iff $p \Downarrow d, v$ and $p \xRightarrow{d}$ iff $p \Downarrow d$.*

The claimed equivalence is entailed by the following Lemmas 4.3-4.7.

Lemma 4.3. *For any two closed programs p, q,*

(1) unless $p \neq \lceil v\rceil$, the judgement $p \xRightarrow{d} q$ is derivable iff there exists a finite chain $p \xrightarrow{d_1} \dots \xrightarrow{d_n} q$ such that $\sum d_i = d$;
(2) the judgement $p \xRightarrow{e}$ is derivable iff
  (a) either $p \xrightarrow{d_1} p' \dots \xrightarrow{d_n} q \xrightarrow{d_{n+1}}$ (with $n = 0$ meaning $p \xrightarrow{e}$),
  (b) or $p \xrightarrow{d_1} p' \xrightarrow{d_2} \dots$,
  with suitable $p', \dots$ and q where $\sum_i d_i = d$.

Note that $\lceil v\rceil$ is not one-step reducible, and hence $\lceil v\rceil \xRightarrow{0} \lceil v\rceil$ is the only provable judgement of the form $\lceil v\rceil \xRightarrow{d} q$.

Lemma 4.4. *For every closed program p, $p \Downarrow d, v$ implies $p \xRightarrow{d} \lceil v\rceil$.*

Proof. We proceed by induction over the derivation of $p \Downarrow d, v$. Note that every rule in Figure 5, whose conclusion is of the form $p \Downarrow d, v$, has as premise big-step judgements only the judgements in the same form. In order to obtain a proof, we thus need to check the induction step for every rule. This is straightforward for all the rules, except for the ones for sequential composition and for while-loops. We consider the ones that apply, in detail.

($\mathbf{seq}_3^\mathbf{d}$) The premises read as $p \Downarrow d, v$ and $q[v/x] \Downarrow e, w$, hence, by induction, $p \xRightarrow{d} \lceil v\rceil$, $q[v/x] \xRightarrow{e} \lceil w\rceil$. If $p = \lceil v\rceil$ then $d = 0$ and $x := p; q \xrightarrow{0} q[v/x] \xRightarrow{e} \lceil w\rceil$, and hence $x := p; q \xRightarrow{d+e} \lceil w\rceil$, as desired. Otherwise, by Lemma 4.3,

$$x := p; q \xrightarrow{d_1} \dots \xrightarrow{d_n} x := \lceil v\rceil; q \xrightarrow{0} q[v/x] \xRightarrow{e} \lceil w\rceil$$

where $p \xrightarrow{d_1} \dots \xrightarrow{d_n} \lceil v\rceil$, $d = \sum_i d_i$. Hence, again $x := p; q \xRightarrow{d+e} \lceil w\rceil$. **($\mathbf{wh}_2^\mathbf{d}$)** The premises imply by induction hypothesis $p \xRightarrow{d} \lceil w\rceil$. The case $p = \lceil w\rceil$ is as in the previous clause. Otherwise, we have

$$x := p\ \mathsf{while}\ v\ \{q\} \xrightarrow{d_1} \dots \xrightarrow{d_n} x := \lceil w\rceil\ \mathsf{while}\ v\ \{q\}$$
$$\xrightarrow{0} \lceil w\rceil$$

with $p \xrightarrow{d_1} \dots \xrightarrow{d_n} \lceil w\rceil$, $d = \sum_i d_i$, which yields the derivation $x := p\ \mathsf{while}\ v\ \{q\} \xRightarrow{d} \lceil w\rceil$.

($\mathbf{wh}_4^\mathbf{d}$) We obtain $p \xRightarrow{d} \lceil w\rceil$ and $x := q[w/x]\ \mathsf{while}\ v\ \{q\} \xRightarrow{r} \lceil u\rceil$ from the rule premises. After disregarding the obvious case $p = \lceil w\rceil$, we obtain similarly to the previous clause

$$x := p\ \mathsf{while}\ v\ \{q\} \xrightarrow{d_1} \dots \xrightarrow{d_n} x := \lceil w\rceil\ \mathsf{while}\ v\ \{q\}$$
$$\xRightarrow{r} \lceil u\rceil$$

Hence $x := p\ \mathsf{while}\ v\ \{q\} \xRightarrow{d+r} \lceil u\rceil$, and we are done. □

Lemma 4.5. *For every closed program p, $p \xRightarrow{d} \lceil v\rceil$ implies $p \Downarrow d, v$.*

Proof. It suffices to show that $p \xrightarrow{d} p'$ with $p' \Downarrow r, v$ imply $p \Downarrow d + r, v$, from which the claim follows by obvious induction. The latter statement in turn follows by induction over the derivation of $p \xrightarrow{d} p'$ using the rules in Figure 4. □

We consider the proofs of the analogous results in the non-terminating cases in more detail.

Lemma 4.6. *For every closed program p, $p \Downarrow d$ implies $p \xRightarrow{d}$.*

Proof. The proof runs analogously to the proof of Lemma 4.4 and occasionally calls the latter. Again, we restrict ourselves to

$$(\mathbf{tr}^{\mathbf{d}}_1)\quad \frac{\forall s \leq d.\, w[v[s/t]/x] = \mathsf{true} \qquad \forall e > 0.\, \exists s \in (d, d+e).\, w[v[s/t]/x] = \mathsf{false}}{x \coloneqq t.\, v \ \&\ w \Downarrow d, v[d/t]} \qquad (\mathbf{pm}^{\mathbf{d}}_1)\quad \frac{p[v/x, w/y] \Downarrow d}{\langle x, y\rangle \coloneqq \langle v, w\rangle;\, p \Downarrow d}$$

$$(\mathbf{tr}^{\mathbf{d}}_2)\quad \frac{\forall s < d.\, w[v[s/t]/x] = \mathsf{true} \qquad w[v[d/t]/x] = \mathsf{false}}{x \coloneqq t.\, v \ \&\ w \Downarrow d} \qquad (\mathbf{pm}^{\mathbf{d}}_2)\quad \frac{p[v/x, w/y] \Downarrow d, u}{\langle x, y\rangle \coloneqq \langle v, w\rangle;\, p \Downarrow d, u}$$

$$(\mathbf{seq}^{\mathbf{d}}_1)\quad \frac{p \Downarrow d}{x \coloneqq p;\, q \Downarrow d} \qquad (\mathbf{seq}^{\mathbf{d}}_2)\quad \frac{p \Downarrow d, v \qquad q[v/x] \Downarrow e}{x \coloneqq p;\, q \Downarrow d + e} \qquad (\mathbf{seq}^{\mathbf{d}}_3)\quad \frac{p \Downarrow d, v \qquad q[v/x] \Downarrow e, w}{x \coloneqq p;\, q \Downarrow d + e, w}$$

$$(\mathbf{now}^{\mathbf{d}})\quad \frac{}{\lceil v \rceil \Downarrow 0, v} \qquad (\mathbf{if}^{\mathbf{d}}_1)\quad \frac{p \Downarrow d}{\mathsf{if\ true\ then}\ p\ \mathsf{else}\ q \Downarrow d} \qquad (\mathbf{if}^{\mathbf{d}}_2)\quad \frac{q \Downarrow d}{\mathsf{if\ false\ then}\ p\ \mathsf{else}\ q \Downarrow d}$$

$$(\mathbf{if}^{\mathbf{d}}_3)\quad \frac{p \Downarrow d, v}{\mathsf{if\ true\ then}\ p\ \mathsf{else}\ q \Downarrow d, v} \qquad (\mathbf{if}^{\mathbf{d}}_4)\quad \frac{q \Downarrow d, v}{\mathsf{if\ false\ then}\ p\ \mathsf{else}\ q \Downarrow d, v} \qquad (\mathbf{wh}^{\mathbf{d}}_1)\quad \frac{p \Downarrow d}{x \coloneqq p\ \mathsf{while}\ v\ \{q\} \Downarrow d}$$

$$(\mathbf{wh}^{\mathbf{d}}_2)\quad \frac{p \Downarrow d, w \qquad v[w/x] = \mathsf{false}}{x \coloneqq p\ \mathsf{while}\ v\ \{q\} \Downarrow d, w} \qquad (\mathbf{wh}^{\mathbf{d}}_3)\quad \frac{p \Downarrow d, w \qquad v[w/x] = \mathsf{true} \qquad x \coloneqq q[w/x]\ \mathsf{while}\ v\ \{q\} \Downarrow r}{x \coloneqq p\ \mathsf{while}\ v\ \{q\} \Downarrow d + r}$$

$$(\mathbf{wh}^{\mathbf{d}}_4)\quad \frac{p \Downarrow d, w \qquad v[w/x] = \mathsf{true} \qquad x \coloneqq q[w/x]\ \mathsf{while}\ v\ \{q\} \Downarrow r, u}{x \coloneqq p\ \mathsf{while}\ v\ \{q\} \Downarrow d + r, u}$$

$$(\mathbf{wh}^{\mathbf{d}}_5)\quad \frac{p \Downarrow d_0, w_0 \qquad q[w_0/x] \Downarrow d_1, w_1 \qquad q[w_1/x] \Downarrow d_2, w_2 \qquad \ldots \qquad \forall i \in \omega.\, v[w_i/x] = \mathsf{true}}{x \coloneqq p\ \mathsf{while}\ v\ \{q\} \Downarrow \sum_i d_i}$$

Figure 5: Big-step duration operational semantics

verifying the rules for sequential composition and while-loops and make free use of Lemma 4.3.

($\mathbf{seq}^{\mathbf{d}}_1$) The premise of the rule implies $p \xRightarrow{d}$ by induction hypothesis. We proceed by case distinction.

- $p \xRightarrow{e} p'$ and $p' \xrightarrow{r}$ where $d = e + r$. Then $x \coloneqq p;\, q \xRightarrow{e} x \coloneqq p';\, q \xrightarrow{r}$, and hence $x \coloneqq p;\, q \xRightarrow{d}$
- $p \xrightarrow{d_1} p' \xrightarrow{d_2} \ldots, d = \sum_i d_i$. Then

  $$x \coloneqq p;\, q \xrightarrow{d_1} x \coloneqq p';\, q \xrightarrow{d_2} \ldots,$$

  and hence again $x \coloneqq p;\, q \xRightarrow{d}$.

($\mathbf{seq}^{\mathbf{d}}_2$) Using Lemma 4.4 and the induction hypothesis, the rule premises imply $p \xRightarrow{d} \lceil v \rceil$ and $q[v/x] \xRightarrow{e}$. After disregarding the obvious case $p = \lceil v \rceil$, the former implies $x \coloneqq p;\, q \xRightarrow{d} x \coloneqq \lceil v \rceil;\, q$, and therefore, the requisite reduction is constructed as follows:

$$x \coloneqq p;\, q \xRightarrow{d} x \coloneqq \lceil v \rceil;\, q \xrightarrow{0} q[v/x] \xRightarrow{e} .$$

($\mathbf{wh}^{\mathbf{d}}_1$) The induction hypothesis implies $p \xRightarrow{d}$. By case distinction:

- $p \xRightarrow{e} p'$ and $p' \xrightarrow{r}$ where $d = e + r$; then we obtain $x \coloneqq p\ \mathsf{while}\ v\ \{q\} \xRightarrow{e} x \coloneqq p'\ \mathsf{while}\ v\ \{q\} \xrightarrow{r}$, and hence $x \coloneqq p\ \mathsf{while}\ v\ \{q\} \xRightarrow{d}$
- $p \xrightarrow{d_1} p' \xrightarrow{d_2} \ldots, d = \sum_i d_i$; then

$$x \coloneqq p\ \mathsf{while}\ v\ \{q\} \xrightarrow{d_1} x \coloneqq p'\ \mathsf{while}\ v\ \{q\} \xrightarrow{d_2} \ldots,$$

  and hence again $x \coloneqq p\ \mathsf{while}\ v\ \{q\} \xRightarrow{d}$.

($\mathbf{wh}^{\mathbf{d}}_3$) By applying the induction hypothesis and Lemma 4.4 to the premises, we obtain the judgments $p \xRightarrow{d} \lceil w \rceil$ and $x \coloneqq q[w/x]\ \mathsf{while}\ v\ \{q\} \xRightarrow{r}$. After disregarding the obvious case $p = \lceil w \rceil$, we obtain:

$$x \coloneqq p\ \mathsf{while}\ v\ \{q\} \xRightarrow{d} x \coloneqq \lceil w \rceil\ \mathsf{while}\ v\ \{q\} \xrightarrow{0}$$
$$x \coloneqq q[w/x]\ \mathsf{while}\ v\ \{q\} \xRightarrow{r}$$

($\mathbf{wh}^{\mathbf{d}}_5$) Finally, let us consider the case of Zeno behaviour of the while-loop. By applying Lemma 4.3 to the rule premises, we obtain $p \xRightarrow{d_0} \lceil w_0 \rceil$, and $q[w_i/x] \xRightarrow{d_{i+1}} \lceil w_{i+1} \rceil$ for all $i \in \omega$. Thus we obtain the following infinite chain

$$\begin{aligned} x \coloneqq p\ \mathsf{while}\ v\ \{q\} \xRightarrow{d_0}\ & x \coloneqq \lceil w_0 \rceil\ \mathsf{while}\ v\ \{q\} \xrightarrow{0} \\ & x \coloneqq q[w_0/x]\ \mathsf{while}\ v\ \{q\} \\ \xRightarrow{d_1}\ & x \coloneqq \lceil w_1 \rceil\ \mathsf{while}\ v\ \{q\} \xrightarrow{0} \\ & x \coloneqq q[w_1/x]\ \mathsf{while}\ v\ \{q\} \\ & \vdots \end{aligned}$$

as desired. □

Lemma 4.7. *For every closed program p, $p \xRightarrow{d}$ implies $p \Downarrow d$.*

Proof. By Lemma 4.3 (2), we must consider two cases.

(a) $p \xRightarrow{e} p' \xrightarrow{r}$ with $d = e + r$. It follows by induction over the derivation of $p' \xrightarrow{r}$ that $p' \Downarrow r$. As in the proof of Lemma 4.5, by subsequent induction over the length of the chain $p \xrightarrow{e_1} \ldots \xrightarrow{e_n} p'$, we obtain $p \Downarrow d$.

(b) $p \xrightarrow{d_1} p_1 \xrightarrow{d_2} p_2 \ldots$, with $d = \sum_i d_i$. We proceed by induction over the number of computation constructs in p. Unless, p is a sequential composition or a while-loop, we can apply the induction hypothesis to $p_1 \xrightarrow{d_2} p_2 \ldots$ and thus obtain $p \xrightarrow{d_1} p_1$

$$(\mathbf{wh}_3^{\mathbf{d}}{}') \quad \frac{p \Downarrow d_0, w_0 \qquad (q[w_i/x] \Downarrow d_{i+1}, w_{i+1})_{i<n} \qquad q[w_n/x] \Downarrow d_{n+1} \qquad \forall i \leq n.\, v[w_i/x] = \mathsf{true}}{x \coloneqq p \;\mathsf{while}\; v\ \{q\} \Downarrow d_0 + \ldots + d_{n+1}} \quad (n \in \omega)$$

$$(\mathbf{wh}_4^{\mathbf{d}}{}') \quad \frac{p \Downarrow d_0, w_0 \qquad (q[w_i/x] \Downarrow d_{i+1}, w_{i+1})_{i<n} \qquad v[w_n/x] = \mathsf{false} \qquad \forall i < n.\, v[w_i/x] = \mathsf{true}}{x \coloneqq p \;\mathsf{while}\; v\ \{q\} \Downarrow d_0 + \ldots + d_n, w_n} \quad (n \in \omega)$$

Figure 6: Structural rules for while-loops

with $p_1 \Downarrow \sum_{i>1} d_i$ from which the goal follows as in clause (a). Consider the two remaining cases.

Let p be $x \coloneqq p'; q$. Then, either $p' \xrightarrow{d_1} p'_1 \ldots \xrightarrow{d_n} \lceil w \rceil$, hence

$$p \xrightarrow{d_1} x \coloneqq p'_1; q \ldots \xrightarrow{d_n} x \coloneqq \lceil w \rceil; q \xrightarrow{0} q[w/x]$$

and we are done by applying the induction hypothesis to the derivation $q[w/x] \xrightarrow{d_{n+1}} \ldots$, or there is an infinite chain $p' \xrightarrow{d_1} p'_1 \xrightarrow{d_2} \ldots$, from which we obtain by induction that $p' \Downarrow \sum_i d_i$ and therefore $p \Downarrow \sum_i d_i$, by $(\mathbf{seq}_1^{\mathbf{d}})$.

Finally, consider the case $p = (x \coloneqq p' \;\mathsf{while}\; b\ \{q\})$. If there is an infinite chain $p' \xrightarrow{d_1} p'_1 \xrightarrow{d_2} \ldots$ the proof reduces to the induction hypothesis analogously to the case of sequential composition above, but now using the rule $(\mathbf{wh}_1^{\mathbf{d}})$. Otherwise, $p' \xRightarrow{\sum_{i \leq n_1} d_i} \lceil w_1 \rceil$ for suitable n_1 and w_1. We then iterate this case distinction, which results in a sequence of the form

$$\begin{aligned} x \coloneqq p' \;\mathsf{while}\; b\ \{q\} &\xRightarrow{\sum_{i \leq n_1} d_i} x \coloneqq q[w_1/x] \;\mathsf{while}\; b\ \{q\} \\ &\xRightarrow{\sum_{n_1 < i \leq n_2} d_i} x \coloneqq q[w_2/x] \;\mathsf{while}\; b\ \{q\} \ \ldots \end{aligned}$$

which either terminates if for some i, $q[w_i/x]$ produces an infinite chain, yielding a proof by reduction to the induction hypothesis, or does not terminate. In the latter case, by Lemma 4.5, $p' \Downarrow \sum_{i \leq n_1} d_i, w_1$, and $q[w_k/x] \Downarrow \sum_{n_k < i \leq n_{k+1}} d_i$ for every $k \in \omega$. Therefore, $p \Downarrow \sum_i d_i$ by $(\mathbf{wh}_5^{\mathbf{d}})$. □

Since for every closed computation term p we can build a possibly non-terminating reduction chain $p \xrightarrow{d_1} p_1 \xrightarrow{d_2}$ (by applying the rules in Fig. 4), we immediately obtain

Corollary 4.8 (Totality). *For any closed computation term p, precisely one of the judgements $p \Downarrow d$ or $p \Downarrow d, v$ is derivable with the rules in Figure 5.*

5 DENOTATIONAL DURATION SEMANTICS

Next we build a syntax-independent, *denotational* counterpart to the operational semantics previously introduced. In order to achieve this, we will take advantage of the general principles of fine-grain call-by-value [15, 18, 27], and thus introduce a monad for the duration semantics. Recall that $(-)^\star$ denotes the Kleisli lifting of a monad $\mathbb{T}$, the set $\mathbb{R}_+$ denotes the non-negative real numbers and the set $\bar{\mathbb{R}}_+$ denotes the non-negative real numbers with infinity.

Definition 5.1 (Duration monad). The duration monad $\mathbb{Q}$ is defined by the following data: $QX = \mathbb{R}_+ \times X \cup \bar{\mathbb{R}}_+$, $\eta(x) = \langle 0, x \rangle$, and

$$\begin{aligned} (f : X \to QY)^\star(d, x) &= \langle d + e, y \rangle && \text{if } f(x) = \langle e, y \rangle, \\ (f : X \to QY)^\star(d, x) &= d + e && \text{if } f(x) = e, \\ (f : X \to QY)^\star(d) &= d. \end{aligned}$$

It is straightforward to check that $\mathbb{Q}$ is a monad using the fact that $\mathbb{R}_+$ is a monoid and that $\bar{\mathbb{R}}_+$ is a monoid $\mathbb{R}_+$-module.

Theorem 5.2. *$\mathbb{Q}$ is a monad.*

Proof. In every *distributive category* $\mathbf{C}$ [10], specifically $\mathbf{C} = \mathbf{Set}$, for every monoid $M \in |\mathbf{C}|$ and every monoid M-module $E \in |\mathbf{C}|$ (i.e. an object with a monoid action $M \times E \to E$ satisfying obvious equations), $M \times - + E$ canonically extends to a (strong) monad [11]. We regard this as folklore. With E being the initial object, $M \times - + E \cong M \times -$ is known as the *writer monad* or as the *(monoid) action monad.* In that sense, the duration monad can be identified as a *generalised* writer monad with $\mathbb{R}_+$ viewed as a monoid under addition and $\bar{\mathbb{R}}_+$ (which is not initial) viewed as a monoid module under addition $\mathbb{R}_+ \times \bar{\mathbb{R}}_+ \to \bar{\mathbb{R}}_+$ of possibly extended real numbers. □

Now, to be able to interpret while-loops over $\mathbb{Q}$, we need to equip it with an *iteration operator* $(-)^\dagger$,

$$\frac{f : X \to Q(Y + X)}{f^\dagger : X \to QY}$$

Moreover, we expect $\mathbb{Q}$ to satisfy the classical laws of iteration [6], i.e. to be (completely) Elgot.

Definition 5.3 (Elgot monads [2, 14]). A strong monad $\mathbb{T}$ on a distributive category is a *(complete) Elgot monad* if it is equipped with an iteration operator sending each $f : X \to T(Y + X)$ to $f^\dagger : X \to TY$ and satisfying the following principles:

- *fixpoint law*: $f^\dagger = [\eta, f^\dagger]^\star f$;
- *naturality*: $g^\star f^\dagger = ([(T\,\mathsf{inl})\, g, \eta\,\mathsf{inr}]^\star f)^\dagger$ for $f : X \to T(Y + X)$, $g : Y \to TZ$;
- *codiagonal*: $(T[\mathsf{id}, \mathsf{inr}]\, f)^\dagger = f^{\dagger\dagger}$ for $f : X \to T((Y + X) + X)$;
- *uniformity*: $f\, h = T(\mathsf{id} + h)\, g$ implies $f^\dagger\, h = g^\dagger$ for $f : X \to T(Y + X)$, $g : Z \to T(Y + Z)$ and $h : Z \to X$;
- *strength*: $\tau\,(\mathsf{id}_Z \times f^\dagger) = ((T\,\mathsf{dist})\, \tau\,(\mathsf{id}_Z \times f))^\dagger$ for any $f : X \to T(Y + X)$.

A standard strategy to define an Elgot monad structure on $\mathbb{Q}$ would be to enrich (the Kleisli category of) $\mathbb{Q}$ over complete partial orders, and then compute $f^\dagger$ as the least fixpoint of the continuous map $[\eta, -]^\star f : \mathsf{Hom}(X, QY) \to \mathsf{Hom}(X, QY)$. However, this would not compatible with our duration semantics, because the iteration operator of $\mathbb{Q}$ must capture *non-inductive* behaviours of while-loops

$$\frac{}{\llbracket \Gamma \vdash_{\mathsf{v}} \star\colon 1 \rrbracket = \lambda\bar{x}.\,\star} \qquad \frac{}{\llbracket \Gamma \vdash_{\mathsf{v}} x_i\colon A \rrbracket = \lambda\bar{x}.\,x_i} \qquad \frac{\llbracket \Gamma \vdash_{\mathsf{v}} v\colon A \rrbracket = h \qquad f\colon A \to B \in \Sigma}{\llbracket \Gamma \vdash_{\mathsf{v}} f(v)\colon B \rrbracket = f\,h}$$

$$\frac{}{\llbracket \Gamma \vdash_{\mathsf{v}} \mathsf{true}\colon 2 \rrbracket = \lambda\bar{x}.\,\mathsf{true}} \qquad \frac{}{\llbracket \Gamma \vdash_{\mathsf{v}} \mathsf{false}\colon 2 \rrbracket = \lambda\bar{x}.\,\mathsf{false}} \qquad \frac{\llbracket \Gamma \vdash_{\mathsf{v}} v\colon A \rrbracket = h \qquad \llbracket \Gamma \vdash_{\mathsf{v}} w\colon B \rrbracket = l}{\llbracket \Gamma \vdash_{\mathsf{v}} \langle v, w\rangle\colon A \times B \rrbracket = \langle h, l\rangle}$$

$$\frac{\llbracket \Gamma \vdash_{\mathsf{v}} v\colon A \times B \rrbracket = h \qquad \llbracket \Gamma, x\colon A, y\colon B \vdash_{\mathsf{c}} p\colon C \rrbracket = l}{\llbracket \Gamma \vdash_{\mathsf{c}} \langle x, y\rangle \coloneqq v;\, p\colon C \rrbracket = l\,\langle \mathsf{id}, h\rangle} \qquad \frac{\llbracket \Gamma \vdash_{\mathsf{c}} p\colon A \rrbracket = h \qquad \llbracket \Gamma, x\colon A \vdash_{\mathsf{c}} q\colon B \rrbracket = l}{\llbracket \Gamma \vdash_{\mathsf{c}} x \coloneqq p;\, q\colon B \rrbracket = l^{\star}\,\tau\,\langle \mathsf{id}, h\rangle}$$

$$\frac{\llbracket \Gamma \vdash_{\mathsf{v}} v\colon A \rrbracket = h}{\llbracket \Gamma \vdash_{\mathsf{c}} \lceil v \rceil\colon A \rrbracket = \eta_A\,h} \qquad \frac{\llbracket \Gamma \vdash_{\mathsf{v}} v\colon 2 \rrbracket = b \qquad \llbracket \Gamma \vdash_{\mathsf{c}} p\colon A \rrbracket = h \qquad \llbracket \Gamma \vdash_{\mathsf{c}} q\colon A \rrbracket = l}{\llbracket \Gamma \vdash_{\mathsf{c}} \mathsf{if}\ v\ \mathsf{then}\ p\ \mathsf{else}\ q\colon A \rrbracket = \lambda\bar{x}.\,h(\bar{x}) \triangleleft b(\bar{x}) \triangleright l(\bar{x})}$$

$$\frac{\llbracket \Gamma, t\colon \mathbb{R} \vdash_{\mathsf{v}} v\colon A \rrbracket = h \qquad \llbracket \Gamma, x\colon A \vdash_{\mathsf{v}} w\colon 2 \rrbracket = b \qquad \lambda\bar{x}.\,\sup\{e \mid \forall t \in [0, e].\, b(\bar{x}, h(\bar{x}, t))\} = d}{\llbracket \Gamma \vdash_{\mathsf{c}} x \coloneqq t.v \mathbin{\&} w\colon A \rrbracket = \lambda\bar{x}.\,\langle d(\bar{x}), h(\bar{x}, d(\bar{x}))\rangle \triangleleft b(\bar{x}, h(\bar{x}, d(\bar{x}))) \triangleright d(\bar{x})}$$

$$\frac{\llbracket \Gamma \vdash_{\mathsf{c}} p\colon A \rrbracket = h \qquad \llbracket \Gamma, x\colon A \vdash_{\mathsf{v}} v\colon 2 \rrbracket = b \qquad \llbracket \Gamma, x\colon A \vdash_{\mathsf{c}} q\colon A \rrbracket = l}{\llbracket \Gamma \vdash_{\mathsf{c}} x \coloneqq p\ \mathsf{while}\ v\ \{q\}\colon A \rrbracket = \lambda\bar{x}.\,((\lambda x.\,(Q\,\mathsf{inr})\,l(\bar{x}, x) \triangleleft b(\bar{x}, x) \triangleright \eta\,\mathsf{inl}\,x)^{\dagger})^{\star}(h(\bar{x}))}$$

Figure 7: Denotational duration semantics

postulated in $(\mathbf{wh}^{\mathbf{d}}_{5})$, the latter implying that even *divergent* computations produce meaningful durations. We thus proceed to make $\mathbb{Q}$ into an *Elgot monad* in a different way. To that end we will first introduce a fine-grained version of $\mathbb{Q}$, roughly mimicking the small-step duration semantics.

Definition 5.4 (Layered duration monad, Guardedness). Let $\hat{Q}X = \nu\gamma.\,(X + \mathbb{R}_+ \times \gamma)$, which extends to a monad $\hat{\mathbb{Q}}$ by general results [41]. On **Set** this monad takes a concrete particularly simple form: $\hat{Q}X = \mathbb{R}_+^{\star} \times X \cup \mathbb{R}_+^{\omega}$, $\eta(x) = \langle \epsilon, x\rangle \in \hat{Q}X$, and

$$(f\colon X \to \hat{Q}Y)^{\star}(w, x) = \langle wu, y\rangle \qquad \text{if } f(x) = \langle u, y\rangle \in \mathbb{R}_+^{\star} \times Y,$$

$$(f\colon X \to \hat{Q}Y)^{\star}(w, x) = wu \qquad \text{if } f(x) = u \in \mathbb{R}_+^{\omega},$$

$$(f\colon X \to \hat{Q}Y)^{\star}(w) = w.$$

A morphism $f\colon X \to \hat{Q}(Y + Z)$ is called *guarded* when for every $x \in X$ if $f(x) = (w, \mathsf{inr}\,z)$ then the word w is non-empty. Guardedness of $f\colon X \to \hat{Q}(Y + X)$ intuitively means that the iteration operator applied to f successively produces durations as the loop unfolds, in other words that the loop is progressive. This provides a fine-grained semantics of divergence, for every infinite stream of durations thus obtained contributes as a specific divergent behaviour.

Proposition 5.5. *For a guarded morphism $f\colon X \to \hat{Q}(Y + X)$, there is a unique $f^{\dagger}\colon X \to \hat{Q}Y$, satisfying the fixpoint equation $f^{\dagger} = [\eta, f^{\dagger}]^{\star} f$.*

Proof. In the coalgebraic form, this was shown in previous work for a large class of monads [30, 40] including $\hat{\mathbb{Q}}$. □

We now render $\mathbb{Q}$ as a quotient of $\hat{\mathbb{Q}}$ and derive a (total) Elgot iteration operator on $\mathbb{Q}$ from the partial one on $\hat{\mathbb{Q}}$, as follows. Note that every QX is a quotient of $\hat{Q}X$ under the "weak bisimulation" relation $\approx$ generated by the clauses,

$$\langle r_1 \ldots r_n, x\rangle \approx \langle s_1 \ldots s_m, x\rangle \qquad (r_1 + \ldots + r_n = s_1 + \ldots + s_m)$$

$$r_1 r_2 \ldots \approx s_1 s_2 \ldots \qquad \Big(\sum\nolimits_i r_i = \sum\nolimits_j s_j\Big)$$

Then let $\rho_X\colon \hat{Q}X \to QX$ be the emerging quotienting map:

$$\rho_X(r_1 \ldots r_n, x) = \Big\langle \sum\nolimits_i r_i, x\Big\rangle, \qquad \rho_X(r_1 r_2 \ldots) = \sum\nolimits_i r_i,$$

and let $\upsilon_X : QX \to \hat{Q}X$ be defined in the following way:

$$\upsilon_X(d, x) = \langle d, x\rangle, \quad \upsilon_X(r \in QX \cap \mathbb{R}_+) = r0^{\omega}, \quad \upsilon_X(\infty) = 1^{\omega}.$$

Note that by definition, for any $f\colon X \to Q(Y + X)$, $\upsilon f\colon X \to \hat{Q}(Y + X)$ is guarded.

Theorem 5.6. *The following is true for ρ_X and υ_X:*

1. *every ρ_X is a left inverse of the corresponding υ_X;*
2. *ρ is a monad morphism;*
3. *for any guarded $f\colon X \to \hat{Q}(Y + X)$, $\rho f^{\dagger} = \rho(\upsilon\rho f)^{\dagger}$.*

Proof. (1) is easy to see by definition, and (2) is essentially due to the fact that countable summation of non-negative real numbers is associative. Let us check (3). For every $x_0 \in X$, there are three possibilities:

(a) there is a finite sequence of elements $x_0, \ldots, x_n$ in X such that $f(x_i) = \langle w_i, x_{i+1}\rangle$ for $i < n$, $f(x_n) = \langle w_n, y\rangle \in \mathbb{R}_+^{+} \times Y$ and $f^{\dagger}(x_0) = \langle w_0 \ldots w_n, y\rangle$;

(b) there is a finite sequence of elements $x_0, \ldots, x_n$ in X such that $f(x_i) = \langle w_i, x_{i+1}\rangle$ for $i < n$, $f(x_n) = w_n \in \mathbb{R}_+^{\omega}$ and $f^{\dagger}(x_0) = w_0 \ldots w_n$;

(c) there is an infinite sequence of elements $x_0, \ldots$ in X such that $f(x_i) = \langle w_i, x_{i+1}\rangle \in \mathbb{R}_+^{+} \times X$ for $i \in \omega$ and $f^{\dagger}(x_0) = w_0 w_1 \ldots$

These three cases fully describe how the iteration operator of $\hat{\mathbb{Q}}$ works. In each case, it is easy to check the identity in question: e.g. in (a) $\rho f^{\dagger}(x_0) = \rho\langle w_0 \ldots w_n, y\rangle = \big\langle \sum w_0 + \ldots + \sum w_n, y\big\rangle$ and then $\rho(\upsilon\rho f)^{\dagger}(x_0) = \rho\big\langle \sum w_0 \ldots \sum w_n, y\big\rangle = \big\langle \sum w_0 + \ldots + \sum w_n, y\big\rangle$ where by $\sum w_i$ we mean the sum of elements of the vector $w_i \in \mathbb{R}_+^{+}$. The remaining cases (b) and (c) are handled analogously. □

We immediately obtain the following corollary of Theorem 5.6.

Corollary 5.7. *$\mathbb{Q}$ is an Elgot monad with the Elgot iteration sending $f\colon X \to Q(Y + X)$ to $\rho\,(\upsilon f)^\dagger\colon X \to QY$.*

Proof. In the terminology of [20], Theorem 5.6 implies that the pair (ρ, υ) is an *iteration-congruent retraction*. Thus, by [20, Theorem 21], $\mathbb{Q}$ is indeed an Elgot monad under the iteration operator described above. □

Note that the Elgot iteration of $\mathbb{Q}$ *adds* the durations produced by a map $f\colon X \to Q(Y + X)$ along the respective iteration process, the latter realised by feeding back to f values of type X it produces. For example, if $f(x) = \langle 1, \mathsf{inr}\, x\rangle$ then $\rho(\upsilon f)^\dagger(x) = 1 + 1 + \cdots = \infty$; if $f(n+1) = \langle 1/2, \mathsf{inr}\, n\rangle$ and $f(0) = \langle 0, \mathsf{inl}\, \star\rangle$ then $\rho(\upsilon f)^\dagger(n) = \langle n * (1/2), \star\rangle$.

Using Elgot iteration provided by Corollary 5.7, we can now define a denotational semantics of HybCore in the following way. We identify the types in (1) with what they denote in **Set**, i.e. natural numbers, real numbers, the one element set $\{\star\}$, the two-element set $\{\top, \bot\}$ and Cartesian products respectively. For a variable context $\Gamma = (x_1\colon A_1, \ldots, x_n\colon A_n)$, let $\llbracket\Gamma\rrbracket = A_1 \times \ldots \times A_n$ (specifically, $\llbracket\Gamma\rrbracket = 1$ if Γ is empty). We will use $\bar{x}$ to refer to the tuples $\langle x_1, \ldots, x_n\rangle$ where $\Gamma = (x_1\colon A_1, \ldots, x_n\colon A_n)$. The semantics $\llbracket\Gamma \vdash_{\mathsf{v}} v\colon A\rrbracket$ and $\llbracket\Gamma \vdash_{\mathsf{c}} p\colon A\rrbracket$ are respectively maps of type $\llbracket\Gamma\rrbracket \to A$ and $\llbracket\Gamma\rrbracket \to QA$, inductively defined in Figure 7.

Theorem 5.8 (Soundness and Adequacy). *Given a computation judgement $- \vdash_{\mathsf{c}} p\colon A$,*

$$p \Downarrow d \quad\text{iff}\quad \llbracket - \vdash_{\mathsf{c}} p\colon A\rrbracket = d,$$
$$p \Downarrow d, v \quad\text{iff}\quad \llbracket - \vdash_{\mathsf{c}} p\colon A\rrbracket = \langle d, \llbracket - \vdash_{\mathsf{v}} v\colon A\rrbracket\rangle.$$

Proof. First we establish *soundness*, which means simultaneously that $p \Downarrow d$ implies $\llbracket - \vdash_{\mathsf{c}} p\colon A\rrbracket = d$ and $p \Downarrow d, v$ implies $\llbracket - \vdash_{\mathsf{c}} p\colon A\rrbracket = \langle d, \llbracket - \vdash_{\mathsf{v}} v\colon A\rrbracket\rangle$. By induction over the derivation length in the proof system of Figure 5 this amounts to checking that every rule preserves these implications. This is done routinely for most of the rules. Consider some instances.

($\mathbf{tr}_1^{\mathbf{d}}$) The assumption reads as $x \coloneqq t.\, v \mathbin{\&} w \Downarrow d, v[d/t]$. The premises of the rule imply

$$d \geq \sup\{e \mid \forall s \in [0, e].\, w[v[s/t]/x] = \mathsf{true}\},$$
$$d \leq \sup\{e \mid \forall s \in [0, e].\, w[v[s/t]/x] = \mathsf{true}\},$$

hence $d = \sup\{e \mid \forall s \in [0, e].\, w[v[s/t]/x] = \mathsf{true}\}$. Assuming $\llbracket t\colon \mathbb{R} \vdash_{\mathsf{v}} v\colon A\rrbracket = h$ and $\llbracket x\colon A \vdash_{\mathsf{v}} w\colon 2\rrbracket = b$, the latter can be rewritten as

$$d = \sup\{e \mid \forall t \in [0, e].\, b(h(t))\}.$$

The left premise of **($\mathbf{tr}_1^{\mathbf{d}}$)** also implies that $b(h(d)) = \mathsf{true}$, and hence the corresponding rule in Figure 7 produces $\llbracket - \vdash_{\mathsf{c}} x \coloneqq t.\, v \mathbin{\&} w\colon A\rrbracket = \langle d, h(d)\rangle$ as required.

($\mathbf{tr}_2^{\mathbf{d}}$) The assumption reads as $x \coloneqq t.\, v \mathbin{\&} w \Downarrow d$. Assuming $\llbracket - \vdash_{\mathsf{v}} t\colon \mathbb{R} \vdash_{\mathsf{v}} v\colon A\rrbracket = h$ and $\llbracket - \vdash_{\mathsf{v}} x\colon A \vdash_{\mathsf{v}} w\colon 2\rrbracket = b$, like in the previous clause, the premises of the rule imply $d = \sup\{e \mid \forall t \in [0, e].\, b(h(t))\}$, however, now $b(h(d)) = \mathsf{false}$. By using the same rule from Figure 7, we obtain $\llbracket - \vdash_{\mathsf{c}} x \coloneqq t.\, v \mathbin{\&} w\colon A\rrbracket = d$.

($\mathbf{wh}_4^{\mathbf{d}}$) Suppose that $x \coloneqq p \;\mathsf{while}\; v\; \{q\} \Downarrow d + r, u$ is obtained from the respective premises of **($\mathbf{wh}_4^{\mathbf{d}}$)** and $\llbracket - \vdash_{\mathsf{c}} p\colon A\rrbracket = h$, $\llbracket x\colon A \vdash_{\mathsf{v}} v\colon 2\rrbracket = b$, $\llbracket x\colon A \vdash_{\mathsf{c}} q\colon A\rrbracket = l$. We obtain that $h = \langle d, w\rangle$ for some w (by induction), $b(w) = \mathsf{true}$ and $\llbracket - \vdash_{\mathsf{c}} x \coloneqq q[w/x] \;\mathsf{while}\; v\; \{q\}\colon A\rrbracket = \langle r, \llbracket - \vdash_{\mathsf{v}} u\colon A\rrbracket\rangle$ (by induction). Let

$$f = \lambda x.\, (Q\,\mathsf{inr})(l(x)) \lhd b(x) \rhd \eta(\mathsf{inl}\, x)$$

and note that $\langle r, \llbracket - \vdash_{\mathsf{v}} u\colon A\rrbracket\rangle = (f^\dagger)^\star(l(w))$, which means that $l(w) = \langle e, c\rangle$, $f^\dagger(c) = \langle e', \llbracket - \vdash_{\mathsf{v}} u\colon A\rrbracket\rangle$ and $e + e' = r$ for suitable e, e' and c. Therefore, using the fixpoint identity for $(-)^\dagger$, $\llbracket - \vdash_{\mathsf{c}} x \coloneqq p \;\mathsf{while}\; v\; \{q\}\colon A\rrbracket = (f^\dagger)^\star(h) = (f^\dagger)^\star(d, w) = [\eta, f^\dagger]^\star f^\star(d, w) = (f^\dagger)^\star(d + e,\, c) = \langle d + e + e', \llbracket - \vdash_{\mathsf{v}} u\colon A\rrbracket\rangle = \langle d + r, \llbracket - \vdash_{\mathsf{v}} u\colon A\rrbracket\rangle$ and we are done.

($\mathbf{wh}_5^{\mathbf{d}}$) Suppose that $x \coloneqq p \;\mathsf{while}\; v\; \{q\} \Downarrow \sum_i d_i$ is obtained from the premises $p \Downarrow d_0, w_0$, $q[w_i/x] \Downarrow d_{i+1}, w_{i+1}$ $(i \in \omega)$, and $v[w_i/x] = \mathsf{true}$ $(i \in \omega)$ by **($\mathbf{wh}_5^{\mathbf{d}}$)**. By induction, $\llbracket - \vdash_{\mathsf{c}} p\colon A\rrbracket = \langle d_0, \llbracket - \vdash_{\mathsf{v}} w_0\colon A\rrbracket\rangle$, and for every $i \in \omega$, $\llbracket - \vdash_{\mathsf{c}} q[w_i/x]\colon A\rrbracket = l(\llbracket - \vdash_{\mathsf{v}} w_i\colon A\rrbracket) = \langle d_{i+1}, \llbracket - \vdash_{\mathsf{v}} w_{i+1}\colon A\rrbracket\rangle$ where $l = \llbracket x\colon A \vdash_{\mathsf{c}} q\colon A\rrbracket$. Note also that $b(\llbracket - \vdash_{\mathsf{v}} w_i\colon A\rrbracket) = \mathsf{true}$ for every $i \in \omega$ with $b = \llbracket x\colon A \vdash_{\mathsf{v}} v\colon 2\rrbracket$. Let again

$$f = \lambda x.\, (Q\,\mathsf{inr})(l(x)) \lhd b(x) \rhd \eta(\mathsf{inl}\, x)$$

and observe that $f(\llbracket - \vdash_{\mathsf{v}} w_i\colon A\rrbracket) = (Q\,\mathsf{inr})(l(\llbracket - \vdash_{\mathsf{v}} w_i\colon A\rrbracket)) = \langle d_{i+1}, \mathsf{inr}\llbracket - \vdash_{\mathsf{v}} w_{i+1}\colon A\rrbracket\rangle$. Now, by definition of $(-)^\dagger$ of $\hat{\mathbb{Q}}$, $(\upsilon f)^\dagger$ $(\llbracket - \vdash_{\mathsf{v}} w_0\colon A\rrbracket) = d_1 d_2 \ldots$, and therefore in $\mathbb{Q}$, $f^\dagger(\llbracket - \vdash_{\mathsf{v}} w_0\colon A\rrbracket) = \rho(\upsilon f)^\dagger(\llbracket - \vdash_{\mathsf{v}} w_0\colon A\rrbracket) = \sum_{i>0} d_i$. Therefore, $\llbracket - \vdash_{\mathsf{c}} x \coloneqq p \;\mathsf{while}\; v\; \{q\}\colon A\rrbracket = (f^\dagger)^\star(d_0, \llbracket - \vdash_{\mathsf{v}} w_0\colon A\rrbracket) = d_0 + f^\dagger(\llbracket - \vdash_{\mathsf{v}} w_0\colon A\rrbracket) = \sum_i d_i$, as desired.

The remaining adequacy direction reads as follows: $\llbracket - \vdash_{\mathsf{c}} p\colon A\rrbracket = d$ implies $p \Downarrow d$ and $\llbracket - \vdash_{\mathsf{c}} p\colon A\rrbracket = \langle d, \llbracket - \vdash_{\mathsf{v}} v\colon A\rrbracket\rangle$ implies $p \Downarrow d, v$. In order to obtain it by induction over derivations in Figure 7, we slightly strengthen the induction invariant as follows: for every substitution σ sending variables from Γ to values of corresponding types, $\llbracket\Gamma \vdash_{\mathsf{c}} p\colon A\rrbracket = d$ implies $p\sigma \Downarrow d\sigma$ and $\llbracket\Gamma \vdash_{\mathsf{c}} p\colon A\rrbracket = \langle d, \llbracket\Gamma \vdash_{\mathsf{v}} v\colon A\rrbracket\rangle$ implies $p\sigma \Downarrow d\sigma, v\sigma$. It is then verified routinely that every rule in Figure 7 preserves this invariant. □

6 EVOLUTION SEMANTICS

Building on the duration semantics of the previous section, we will now present a big-step *evolution operational semantics* for hybrid programs, which incorporates the former as a critical ingredient. In particular, both semantics will be needed for providing the aforementioned adequacy result. The new semantics relates a closed computation term p with its *trajectory*, i.e. the range of time-indexed values produced by the execution of p – recall for instance the bouncing ball and cruise controller described in Section 3. More specifically, the corresponding big-step relation $\Downarrow$ connects a closed computation term p and a time instant t with the value v to which p evaluates at t, in symbols $p, t \Downarrow v$. The corresponding derivation rules are presented in Figure 8. These rules recur to the duration semantic judgements governed by the rules in Figure 5. Unlike duration semantics, evolution semantics is not total: for example, for a given computation p, $p, t \Downarrow v$ may not be derivable for any v and t, which yields an *empty trajectory* for p, denotationally a totally undefined function $\emptyset \to A$.

Let us discuss the non-obvious rules of this semantics, starting with **($\mathbf{seq}_1^{\mathbf{e}}$)** which takes infinite families of judgements as premises.

$$\textbf{(seq}_1^{\mathbf{e}}\textbf{)}\quad \frac{(p, s \Ddownarrow v_s)_{s \le t} \qquad (q[v_s/x], 0 \Ddownarrow w_s)_{s \le t}}{x \coloneqq p;\ q, t \Ddownarrow w_t} \qquad \textbf{(seq}_2^{\mathbf{e}}\textbf{)}\quad \frac{p \Downarrow d, v' \qquad (p, s \Ddownarrow v_s)_{s \le d} \qquad (q[v_s/x], 0 \Ddownarrow w_s)_{s \le d} \qquad q[v_d/x], t \Ddownarrow w}{x \coloneqq p;\ q, d + t \Ddownarrow w}$$

$$\textbf{(if}_1^{\mathbf{e}}\textbf{)}\quad \frac{p, t \Ddownarrow v}{\mathsf{if\ true\ then}\ p\ \mathsf{else}\ q, t \Ddownarrow v} \qquad \textbf{(if}_2^{\mathbf{e}}\textbf{)}\quad \frac{q, t \Ddownarrow v}{\mathsf{if\ false\ then}\ p\ \mathsf{else}\ q, t \Ddownarrow v} \qquad \textbf{(pm}^{\mathbf{e}}\textbf{)}\quad \frac{p[v/x, w/y], t \Ddownarrow u}{\langle x, y \rangle \coloneqq \langle v, w \rangle;\ p, t \Ddownarrow u}$$

$$\textbf{(now}^{\mathbf{e}}\textbf{)}\quad \frac{}{\lceil v \rceil, 0 \Ddownarrow v} \qquad \textbf{(tr}_1^{\mathbf{e}}\textbf{)}\quad \frac{x \coloneqq s.\, v\ \&\ w \Downarrow d \qquad t < d}{x \coloneqq s.\, v\ \&\ w, t \Ddownarrow v[t/s]} \qquad \textbf{(tr}_2^{\mathbf{e}}\textbf{)}\quad \frac{x \coloneqq s.\, v\ \&\ w \Downarrow d, u \qquad t \le d}{x \coloneqq s.\, v\ \&\ w, t \Ddownarrow v[t/s]}$$

$$\textbf{(wh}_1^{\mathbf{e}}\textbf{)}\quad \frac{p \Downarrow d \qquad p, t \Ddownarrow w \qquad t < d}{x \coloneqq p\ \mathsf{while}\ v\ \{q\}, t \Ddownarrow w} \qquad \textbf{(wh}_3^{\mathbf{e}}\textbf{)}\quad \frac{p \Downarrow d, w' \qquad p, d \Ddownarrow w \qquad v[w/x] = \mathsf{false}}{x \coloneqq p\ \mathsf{while}\ v\ \{q\}, d \Ddownarrow w}$$

$$\textbf{(wh}_2^{\mathbf{e}}\textbf{)}\quad \frac{p \Downarrow d, w' \qquad p, t \Ddownarrow w \qquad t < d}{x \coloneqq p\ \mathsf{while}\ v\ \{q\}, t \Ddownarrow w} \qquad \textbf{(wh}_4^{\mathbf{e}}\textbf{)}\quad \frac{p \Downarrow d, w' \qquad p \Ddownarrow d, w \qquad v[w/x] = \mathsf{true} \qquad x \coloneqq q[w/x]\ \mathsf{while}\ v\ \{q\}, t \Ddownarrow u}{x \coloneqq p\ \mathsf{while}\ v\ \{q\}, d + t \Ddownarrow u}$$

Figure 8: Big-step evolution operational semantics

This rule is similar to $\textbf{(seq}_1^{\mathbf{d}}\textbf{)}$, but in $\textbf{(seq}_1^{\mathbf{e}}\textbf{)}$ even if the first computation evaluates to v_t at t, we cannot disregard q and use v_t in the conclusion, because the return type of q need not match the return type of p. The evaluation $q[v_t/x], 0 \Ddownarrow w_t$ is required to perform the necessary type conversion. Moreover, the rule premise ensures that for every $s \le t$, $q[v_s/x]$ also yields a value at 0. To see why this is necessary, consider the following instance of $x \coloneqq p;\ q$:

$$x \coloneqq (x \coloneqq t.\, t\ \&\ \mathsf{true});\ \mathsf{if}\ x < 1\ \mathsf{then}\,(x \coloneqq 1\ \&\ \mathsf{false})\ \mathsf{else}\ x$$

First observe that for any given time instant $t \ge 0$ the subprogram $x \coloneqq t.\, t\ \&\ \mathsf{true}$ evaluates to t, intuitively modelling the passage of time *ad infinitum* and thus producing a trajectory of infinite duration. On the other hand, the subprogram $x \coloneqq 1\ \&\ \mathsf{false}$ models non-progressive divergence and *does not* yield a value at any time instant, in particular $q[0/x], 0 \Ddownarrow v$ for no v. Hence the entire program $x \coloneqq p;\ q$ diverges already at 0, as we cannot apply any of the sequential composition rules relative to this time instant. However, $q[1/x], 0 \Ddownarrow 1$ and therefore if we did not check previous time instants for divergence (using $q[v_s/x], 0 \Ddownarrow w_s$ for $s \le 1$) we would obtain $x \coloneqq p;\ q, 1 \Ddownarrow 1$ by the sequential composition rules, meaning that we would not be able to detect divergence of the entire program.

Remark 3. *The evolution semantics subtly differs from the duration semantics. Specifically, the above example indicates that $p \Downarrow d$ does not necessarily imply that $d = \sup\{t \mid \exists w.\, p, t \Ddownarrow w\}$.*

The main subtlety behind our semantics of while-loops via $\textbf{(wh}_1^{\mathbf{e}}\textbf{)}$–$\textbf{(wh}_4^{\mathbf{e}}\textbf{)}$ is that, contrary to standard program semantics, while-loops no longer behave as iterated sequential composition, which would be easily achievable via the rule

$$\frac{(p, s \Ddownarrow v_s)_{s \le t} \qquad v[v_t/x] = \mathsf{true} \qquad (x \coloneqq q[v_s/x]\ \mathsf{while}\ v\ \{q\}, 0 \Ddownarrow w_s)_{s \le t,\, v[v_s/x] = \mathsf{true}}}{x \coloneqq p\ \mathsf{while}\ v\ \{q\}, t \Ddownarrow w_t}$$

instead of $\textbf{(wh}_1^{\mathbf{e}}\textbf{)}$ and similarly for $\textbf{(wh}_2^{\mathbf{e}}\textbf{)}$–$\textbf{(wh}_4^{\mathbf{e}}\textbf{)}$. We argue, however, that our rules are more suitable for hybrid programming: the ones corresponding to iterated sequential composition produce undesirable artefacts, e.g. the program

$$x \coloneqq 0\ \mathsf{while\ true}\ \{x \coloneqq t.\, (t + x)\ \&\ 1\}$$

would not evaluate under these rules at any time instant (we would not be able to apply any of these rules since at every time instant we would be lacking a terminating condition) and the same would apply to the bouncing ball in Example 3.1 and other systems involving infinite while-loops or Zeno behaviour. Intuitively, the reader may see the adopted rules as the ones corresponding to testing the condition of the while-loop only at the end of the input trajectory whilst assuming that the other points evaluate to false; the rules corresponding to iterated sequential composition would, on the other hand, correspond to testing the condition of the while-loop at *all* points of the input trajectory.

The following lemma is obtained by induction over the derivation of the corresponding computation judgement by the rules in Figure 8.

Lemma 6.1. *For every closed computation term p, the set*

$$\{t \in \mathbb{R}_+ \mid \exists v.\, p, t \Ddownarrow v\}$$

is downward closed.

To relate the operational semantics with its denotational counterpart, recall the hybrid monad $\mathbb{H}$ from our previous work [17] (in a slightly reformulated equivalent form). Intuitively, for every set X, the elements of HX are trajectories in the sense discussed above and Kleisli composition is essentially used for concatenating trajectories whilst simultaneously tracking divergence along the computation. In what follows we use the superscript notation e^t explained in Section 2.

Definition 6.2 (Hybrid monad). The *hybrid monad* $\mathbb{H}$ is defined as follows. Let

$$\begin{aligned} HX = {} & \{\langle \mathrm{cc}, d, e : [0, d] \to X \rangle \mid d \in \mathbb{R}_+\} \cup \\ & \{\langle \mathrm{cd}, d, e : [0, d] \to X \rangle \mid d \in \mathbb{R}_+\} \cup \\ & \{\langle \mathrm{od}, d, e : [0, d) \to X \rangle \mid d \in \bar{\mathbb{R}}_+\}, \end{aligned}$$

i.e. the set HX contains three types of trajectory: *closed convergent* $\langle \mathrm{cc}, d, e \rangle$, *closed divergent* $\langle \mathrm{cd}, d, e \rangle$ and *open divergent* $\langle \mathrm{od}, d, e \rangle$ (notably, there are no “open convergent” ones), e.g., HX contains the *empty trajectory* $\langle \mathrm{od}, 0, ! \rangle$, which we denote by $\ominus$. Given $p = \langle \alpha, d, e \rangle \in HX$, let us write $p_{\mathrm{tt}} = \alpha$, $p_{\mathrm{dr}} = d$, $p_{\mathrm{ev}} = e$, $\operatorname{dom} e = [0, d)$

$$\frac{[\![\Gamma, t\colon \mathbb{R} \vdash_{\mathsf{v}} v\colon A]\!] = h \qquad [\![\Gamma, x\colon A \vdash_{\mathsf{v}} w\colon 2]\!] = b \qquad \lambda\bar{x}.\ \sup\{e \in \mathbb{R}_{+} \mid \forall t \in [0,e].\, b(\bar{x}, h(\bar{x}, t))\} = d}{[\![\Gamma \vdash_{\mathsf{c}}\ x \coloneqq t.\, v \ \&\ w\colon A]\!] = \lambda\bar{x}.\, \langle \mathsf{cc} \lhd b(\bar{x}, h(\bar{x}, d(\bar{x}))) \rhd \mathsf{od}, d(\bar{x}), \lambda t.\, h(\bar{x}, t)\rangle}$$

$$\frac{[\![\Gamma \vdash_{\mathsf{c}} p\colon A]\!] = h \qquad [\![\Gamma, x\colon A \vdash_{\mathsf{v}} v\colon 2]\!] = b \qquad [\![\Gamma, x\colon A \vdash_{\mathsf{c}} q\colon A]\!] = l}{[\![\Gamma \vdash_{\mathsf{c}}\ x \coloneqq p \ \mathsf{while}\ v\ \{q\}\colon A]\!] = \lambda\bar{x}.\, ((\lambda\langle x, c\rangle.\, (H\,\mathsf{inr})(\kappa\, l(\bar{x}, x)) \lhd b(\bar{x}, x) \wedge c \rhd \eta(\mathsf{inl}\, x))^{\dagger})^{\star}(\kappa(h(\bar{x})))}$$

Figure 9: Denotational evolution semantics

if $\alpha = \mathsf{od}$ and $\operatorname{dom} e = [0, d]$ otherwise. For a map $f\colon X \to HY$ we will abbreviate the notation $(f(x))_{\mathsf{tt}}$ to $f_{\mathsf{tt}}(x)$, and similarly for the other cases, namely $(f(x))_{\mathsf{dr}}$, $(f(x))_{\mathsf{ev}}$ and $(f(x))^{t}$. The monad structure of $\mathbb{H}$ is then given in the following way: $\eta(x) = \langle \mathsf{cc}, 0, \lambda t.\, x\rangle$; for $f\colon X \to HY$, $\langle \alpha, d, e\rangle \in HX$, let

$$D = \bigcup \big\{[0,t] \subseteq \operatorname{dom} e \mid \forall s \in [0,t].\, f(e^{s}) \neq \ominus\big\},$$

$d' = \sup D$. Then $f^{\star}(\alpha, d, e)$ is defined by the clauses:

$$\begin{aligned}
f^{\star}(\mathsf{cc}, d, e) &= \langle f_{\mathsf{tt}}(e^{d}), d + f_{\mathsf{dr}}(e^{d}),\\
&\qquad \lambda t.\, f^{0}_{\mathsf{ev}}(e^{t}) \lhd t < d \rhd f^{t-d}_{\mathsf{ev}}(e^{d})\rangle && (D = [0,d])\\
f^{\star}(\mathsf{cd}, d, e) &= \langle \mathsf{cd}, d, \lambda t.\, f^{0}_{\mathsf{ev}}(e^{t})\rangle && (D = [0,d])\\
f^{\star}(\alpha, d, e) &= \langle \mathsf{cd}, d', \lambda t.\, f^{0}_{\mathsf{ev}}(e^{t})\rangle && (D = [0,d'], d' < d)\\
f^{\star}(\alpha, d, e) &= \langle \mathsf{od}, d', \lambda t.\, f^{0}_{\mathsf{ev}}(e^{t})\rangle && (D = [0,d'), d' \leq d)
\end{aligned}$$

Intuitively, only the first clause for $f^{\star}(\alpha, d, e)$ describes a successful 'concatenation' scenario, which assumes that the argument trajectory is closed and additionally ensures that $f(e^{s})$ does not yield divergence along the run of $\langle \mathsf{cc}, d, e\rangle$. The remaining clauses exhaustively cover divergence scenarios, caused either by $f(e^{s})$ or by the argument trajectory. As shown in [17], $\mathbb{H}$ is an Elgot monad. We thus reuse the rules from Figure 7 to define the semantics over $\mathbb{H}$ by replacing Q with H throughout, except for the rules for **(tr)** and **(wh)** which are overridden by those in Figure 9. To cope with the non-standard behaviour of while-loops discussed above, we call on an auxiliary natural transformation $\kappa\colon H \to H(-\times 2)$, appending to every point of the trajectory a Boolean flag signalling whether the current point is the endpoint, formally:

$$\kappa(\alpha, d, e) = \langle \alpha, d, u\rangle, \qquad u^{t} = \begin{cases} \langle e^{t}, \top\rangle & \text{if } \alpha = \mathsf{cc},\ t = d,\\ \langle e^{t}, \bot\rangle & \text{otherwise.}\end{cases}$$

Finally, we obtain our main result.

Theorem 6.3 (Soundness and Adequacy). *Given $- \vdash_{\mathsf{c}} p\colon A$,*

(1) $p, t \Downarrow v$ iff $[\![- \vdash_{\mathsf{c}} p\colon A]\!] = \langle \alpha, d, e\rangle$, $t \leq d$ and $e^{t} = v$;
(2) $p, d \Downarrow v$ and $p \Downarrow d, w$ iff $v = w$, $[\![- \vdash_{\mathsf{c}} p\colon A]\!] = \langle \mathsf{cc}, d, e\rangle$ and $e^{d} = v$.

The first clause ensures that the correspondence $t \mapsto v_{t}$ generated by the judgements $p, t \Downarrow v_{t}$ yields a trajectory that agrees with the denotational semantics previously presented. The second clause is needed to relate the duration and evolution semantics: both must agree in the evaluation of the *last point* of the trajectory produced by a program p, and naturally both must agree with the corresponding denotational semantics.

The theorem follows from a sequence of lemmas, which we present next. In the rest of the section, we subscript the semantic brackets with $\mathbb{Q}$ and $\mathbb{H}$ to refer to duration semantics and to evolution semantics correspondingly. The first lemma provides a technical characterization of iterations occurring as the semantics of while-loops.

Lemma 6.4. *Let $f\colon X \to H(Y + X)$ have the property*

$$\forall x \in X.\, (\exists y \in X.\, f^{t}_{\mathsf{ev}}(x) = \mathsf{inr}\, y) \Rightarrow f_{\mathsf{tt}}(x) = \mathsf{cc} \wedge f_{\mathsf{dr}}(x) = t. \quad (3)$$

We define $f_{i}\colon X \to H(Y + X)$ by induction as follows: $f_{0} = \eta\,\mathsf{inr}$, $f_{i+1} = [\eta\,\mathsf{inl}, f_{i}]^{\star} f$. Let $x \in X$, and suppose that $f^{\dagger}(x) = \langle \alpha, d, e\rangle$.

(1) If $t < d$ then there exists $n \in \mathbb{N}$ such that $t < (f_{n})_{\mathsf{dr}}(x)$ and $(f_{n})^{t}_{\mathsf{ev}}(x) = \mathsf{inl}\, e^{t}$.
(2) If $\alpha \neq \mathsf{od}$ then there exists $n \in \mathbb{N}$ such that $d = (f_{n})_{\mathsf{dr}}(x)$ and $(f_{n})^{d}_{\mathsf{ev}}(x) = \mathsf{inl}\, e^{d}$.

Proof. We rely on the fact that $\mathbb{H}$ is characterized as an iteration-congruent retract of a monad $\mathbb{H}_{0}\mathbb{M}$ obtained by composing $\mathbb{H}_{0}$, an iteration-free version of $\mathbb{H}$, with the *maybe-monad* $\mathbb{M}$ (with $MX = X + 1$) [17]. That is, there is a monad morphism $\rho\colon \mathbb{H}_{0}\mathbb{M} \to \mathbb{H}$, whose components ρ_{X} come with right inverses v_{X} and ρ preserves iteration as follows: $\rho f^{\ddagger} = (\rho f)^{\dagger}$. Here, by $(-)^{\ddagger}$ we refer to the iteration of $\mathbb{H}_{0}\mathbb{M}$, which is suitably characterized as the least solution of the fixpoint law. For the sake of the present argument it suffices to know that $H_{0}MX$ is obtained from HX by allowing the trajectories to be partial functions and by imposing the the following constraints:

- $\langle \mathsf{cc}, d, e\rangle \in H_{0}MX$ implies that e^{d} is defined;
- closed divergent trajectories are not in $H_{0}MX$.

The natural transformation $v\colon H \to H_{0}M$ sends $\langle \mathsf{cc}, d, e\rangle \in HX$ to $\langle \mathsf{cc}, d, e\rangle \in H_{0}MX$ and $\langle \alpha, d, e\rangle \in HX$ with $\alpha \in \{\mathsf{cd}, \mathsf{od}\}$ to $\langle \mathsf{od}, \infty, e'\rangle \in H_{0}MX$ where e' is the extension of e to $\bar{\mathbb{R}}_{+}$ with undefined elements. Conversely, $\rho(\alpha, d, e)$ is $\langle \alpha, d, e\rangle$ for total e and $\rho(\alpha, d, e) = \langle \alpha', d', e'\rangle$ for partial e, in which case e' is a restriction of e to the largest downward closed subset of the domain of definiteness of e and d' and $\alpha \in \{\mathsf{cd}, \mathsf{od}\}$ are selected accordingly.

Let $g = vf\colon X \to H_{0}M(Y + X)$. For every i, let $g^{\langle i\rangle}\colon X \to H_{0}MY$ be defined as follows: $g^{\langle 0\rangle} = \ominus$ (the empty trajectory) and $g^{\langle i+1\rangle} = [\eta, g^{\langle i\rangle}]^{\star} g$. We use the following property of $\mathbb{H}_{0}\mathbb{M}$ [17, Lemma 25]:

$$\forall x \in X.\, g^{\ddagger}(x) = \langle \alpha, d, e\rangle \wedge e^{t} \in Y \Rightarrow \exists n.\, (g^{\langle n\rangle})^{t}_{\mathsf{ev}}(x) = \mathsf{inl}\, e^{t} \quad (4)$$

for every $t \leq d$ for which e^{t} is defined. Let us define $g_{0} = \eta\,\mathsf{inr}$, $g_{i+1} = [\eta\,\mathsf{inl}, g_{i}]^{\star} g$ and note that, by induction, for every i, $g^{\langle i\rangle} = [\eta, \ominus]^{\star} g_{i}$. Therefore we can reformulate (4) as follows:

$$\forall x \in X.\, g^{\ddagger}(x) = \langle \alpha, d, e\rangle \wedge e^{t} \in Y \Rightarrow \exists n.\, (g_{n})^{t}_{\mathsf{ev}}(x) = \mathsf{inl}\, e^{t}. \quad (5)$$

In order to obtain the first clause of the lemma, suppose that $f^{\dagger}(x) = \langle \alpha, d, e\rangle$ and $t < d$. Now $f^{\dagger}(x) = \rho g^{\ddagger}(x) = \langle \alpha, d, e\rangle$ and therefore $g^{\ddagger}(x) = \langle \alpha', d', e'\rangle$ with suitable α', d', e' and $(e')^{t} = e^{t}$ because e' is at least as defined as e. By (5), for some n, $(g_{n})^{t}_{\mathsf{ev}}(x) = \mathsf{inl}\, e^{t}$.

Note that by definition g satisfies (3) and for every $y \in X$ the trajectory $g_{\mathsf{ev}}(y)$ is total. It follows by induction that the same is true for g_n, hence $g_n(x) = v(f_n(x))$, in particular, $t < (f_n)_{\mathsf{dr}}(x)$ and $(f_n)^t_{\mathsf{ev}}(x) = \mathsf{inl}\, e^t$. The second clause is shown analogously. □

Next, we check that the duration semantics and the evolution semantics agree in the following sense (cf. Remark 3).

Lemma 6.5. *Given a computation judgement $\Gamma \vdash_{\mathsf{c}} p\colon A$, let $\bar v \in \llbracket\Gamma\rrbracket$. Suppose that $\llbracket\Gamma \vdash_{\mathsf{c}} p\colon A\rrbracket_{\mathbb{H}}(\bar v) = \langle \alpha, d, e\rangle$ and $\llbracket\Gamma \vdash_{\mathsf{c}} p\colon A\rrbracket_{\mathbb{Q}}(\bar v) = \langle d', v\rangle$. Then $\alpha = \mathsf{cc}$ iff $d = d'$ and in either case $e^d = v$.*

Proof Sketch. The proof runs by induction over the structure of p. We restrict to the following three representative cases.

- $p = (x \coloneqq q;\, r)$. Let $f = \llbracket\Gamma \vdash_{\mathsf{c}} q\colon B\rrbracket_{\mathbb{H}}$, $g = \llbracket\Gamma, x\colon B \vdash_{\mathsf{c}} r\colon A\rrbracket_{\mathbb{H}}$, $\hat f = \llbracket\Gamma \vdash_{\mathsf{c}} q\colon B\rrbracket_{\mathbb{Q}}$, $\hat g = \llbracket\Gamma, x\colon B \vdash_{\mathsf{c}} r\colon A\rrbracket_{\mathbb{Q}}$. Suppose that $\alpha = \mathsf{cc}$. Then $\llbracket\Gamma \vdash_{\mathsf{c}} p\colon A\rrbracket_{\mathbb{H}}(\bar v) = g^\star\tau\langle \bar v, f(\bar v)\rangle = \langle \mathsf{cc}, d, e\rangle$, which by definition means that

$$f(\bar v) = \langle \mathsf{cc}, d_1, e_1\rangle, \quad g(\bar v, e_1^{d_1}) = \langle \mathsf{cc}, d_2, e_2\rangle, \quad d = d_1 + d_2,$$

and, analogously

$$\hat f(\bar v) = \langle d_1', v'\rangle, \qquad \hat g(\bar v, v') = \langle d_2', v\rangle, \qquad d' = d_1' + d_2'.$$

By induction, $d_1 = d_1'$ and $e_1^{d_1} = v'$. Hence, again by induction, $d_2 = d_2'$ and $e_2^{d_2} = v$. This implies $d = d'$ and $e^d = e_2^{d_2} = v$. By reversing this argument we also obtain the implication in the right-to-left direction.

- $p = (\, x \coloneqq t\,.\, v \;\&\; w)$. By definition,

$$\begin{aligned}\llbracket\Gamma \vdash_{\mathsf{c}} p\colon A\rrbracket_{\mathbb{H}}(\bar v) &= \langle \mathsf{cc}, u(\bar v), \lambda t.\, h(\bar v, t)\rangle \lhd b(\bar v, h(\bar v, u(\bar v))) \rhd \\ &\qquad \langle \mathsf{od}, u(\bar v), \lambda t.\, h(\bar v, t)\rangle,\\ \llbracket\Gamma \vdash_{\mathsf{c}} p\colon A\rrbracket_{\mathbb{Q}}(\bar v) &= \langle u(\bar v), h(\bar v, u(\bar v))\rangle \lhd b(\bar v, h(\bar v, u(\bar v))) \rhd u(\bar v)\end{aligned}$$

where $h = \llbracket\Gamma, t\colon \mathbb{R} \vdash_{\mathsf{v}} v\colon A\rrbracket$ and $b = \llbracket\Gamma, x\colon A \vdash_{\mathsf{v}} w\colon 2\rrbracket$, $u = \lambda \bar x.\ \sup\{e \in \mathbb{R}_+ \mid \forall t \in [0, e].\, b(\bar x, h(\bar x, t))\}$. The assumption implies $b(\bar v, h(\bar v, u(\bar v))) = \top$, hence we reduce to

$$\begin{aligned}\llbracket\Gamma \vdash_{\mathsf{c}} p\colon A\rrbracket_{\mathbb{H}}(\bar v) &= \langle \mathsf{cc}, u(\bar v), \lambda t.\, h(\bar v, t)\rangle,\\ \llbracket\Gamma \vdash_{\mathsf{c}} p\colon A\rrbracket_{\mathbb{Q}}(\bar v) &= \langle u(\bar v), h(\bar v, u(\bar v))\rangle\end{aligned}$$

and the claim of the lemma holds trivially.

- $p = (\, x \coloneqq q \;\mathsf{while}\; v\; \{r\})$. By definition,

$$\begin{aligned}\llbracket\Gamma \vdash_{\mathsf{c}} p\colon A\rrbracket_{\mathbb{H}}(\bar v) &= ((\lambda\langle x, c\rangle.\, (H\,\mathsf{inr})(\kappa\, l(\bar v, x))\\ &\qquad \lhd b(\bar v, x) \wedge c \rhd \eta(\mathsf{inl}\, x))^\dagger)^\star(\kappa\, h(\bar v)),\\ \llbracket\Gamma \vdash_{\mathsf{c}} p\colon A\rrbracket_{\mathbb{Q}}(\bar v) &= ((\lambda x.\, (Q\,\mathsf{inr})(\hat l(\bar v, x))\\ &\qquad \lhd b(\bar v, x) \rhd \eta(\mathsf{inl}\, x))^\dagger)^\star(\hat h(\bar v))\end{aligned}$$

where $b = \llbracket\Gamma, x\colon A \vdash_{\mathsf{v}} v\colon 2\rrbracket$, and

$$\begin{aligned}h &= \llbracket\Gamma \vdash_{\mathsf{c}} q\colon A\rrbracket_{\mathbb{H}}, & l &= \llbracket\Gamma, x\colon A \vdash_{\mathsf{c}} r\colon A\rrbracket_{\mathbb{H}},\\ \hat h &= \llbracket\Gamma \vdash_{\mathsf{c}} q\colon A\rrbracket_{\mathbb{Q}}, & \hat l &= \llbracket\Gamma, x\colon A \vdash_{\mathsf{c}} r\colon A\rrbracket_{\mathbb{Q}}.\end{aligned}$$

Suppose that $\llbracket\Gamma \vdash_{\mathsf{c}} q\colon A\rrbracket_{\mathbb{H}}(\bar v) = \langle \mathsf{cc}, d, e\rangle$. Then, by definition of $\mathbb{H}$, $d = d_0 + d_\star$ where

$$\begin{aligned}\langle \mathsf{cc}, d_0, e_0\rangle &= h(\bar v),\\ \langle \mathsf{cc}, d_\star, e_\star\rangle &=\\ &\hspace{-3em}(\lambda\langle x, c\rangle.\, (H\,\mathsf{inr})(\kappa\, l(\bar v, x)) \lhd b(\bar v, x) \wedge c \rhd \eta\,\mathsf{inl}\, x)^\dagger(e_0^{d_0}, \top).\end{aligned}$$

Consider the sequence

$$\langle \mathsf{cc}, d_0, e_0\rangle, \langle \mathsf{cc}, d_1, e_1\rangle, \ldots \tag{6}$$

formed as follows: $\langle \mathsf{cc}, d_{i+1}, e_{i+1}\rangle = l(\bar v, e_i^{d_i})$ if $b(\bar v, e_i^{d_i})$. That is, every triple $\langle \mathsf{cc}, d_{i+1}, e_{i+1}\rangle$ is obtained by applying the above expression under dagger to $\langle e_i^{d_i}, \top\rangle$ as long as $b(\bar v, e_i^{d_i})$ is true. Note that this process cannot produce triples with the first element different than cc, because this would mean that the loop diverges after finitely many iterations, returning a triple different from $\langle \mathsf{cc}, d_\star, e_\star\rangle$. By Lemma 6.4 (2), the sequence (6) is necessarily finite. Hence, by induction hypothesis $\llbracket\Gamma, x\colon A \vdash_{\mathsf{c}} r\colon A\rrbracket_{\mathbb{Q}}(\bar v, e_i^{d_i}) = \langle d_{i+1}, e_{i+1}^{d_{i+1}}\rangle$ for every i, from which we obtain $\llbracket\Gamma \vdash_{\mathsf{c}} p\colon A\rrbracket_{\mathbb{Q}}(\bar v) = \langle d, e^d\rangle$, as desired.

For the converse, suppose that $\llbracket\Gamma \vdash_{\mathsf{c}} p\colon A\rrbracket_{\mathbb{Q}}(\bar v) = \langle d, v\rangle$. By definition of iteration in $\mathbb{Q}$, this means that the loop in $\llbracket\Gamma \vdash_{\mathsf{c}} p\colon A\rrbracket_{\mathbb{Q}}(\bar v)$ is unfolded finitely many times, i.e. $\langle d_0, v_0\rangle = \hat h(\bar v)$, $\langle d_1, v_1\rangle = \hat l(\bar v, v_0), \ldots, \langle d_n, v_n\rangle = \hat l(\bar v, v_{n-1})$ and $b(\bar v, v_0) = \top, \ldots, b(\bar v, v_{n-1}) = \top, b(\bar v, v_0) = \bot$ for suitable d_i, v_i and $d = d_0 + \ldots + d_n$, $v = v_n$. By induction hypothesis, $\llbracket\Gamma \vdash_{\mathsf{c}} p\colon A\rrbracket_{\mathbb{H}}(\bar v) = \langle \mathsf{cc}, d_0, e_0\rangle$, $\llbracket\Gamma, x\colon A \vdash_{\mathsf{c}} q\colon A\rrbracket_{\mathbb{H}}(\bar v, v_i) = \langle \mathsf{cc}, d_{i+1}, e_{i+1}\rangle$ for $i = 1, \ldots, n$ and $e_i^{d_i} = v_i$ for every $i = 0, \ldots, n$. Using the above expression for $\llbracket\Gamma \vdash_{\mathsf{c}} p\colon A\rrbracket_{\mathbb{Q}}(\bar v)$ and the assumption that $\llbracket\Gamma \vdash_{\mathsf{c}} p\colon A\rrbracket_{\mathbb{H}}(\bar v) = \langle \alpha, d, e\rangle$, we obtain that $\alpha = \mathsf{cc}$. □

The following lemma essentially captures the soundness direction of Theorem 6.3.

Lemma 6.6. *Given $- \vdash_{\mathsf{c}} p\colon A$,*

(1) if $p, t ⟱ v$ then $\llbracket - \vdash_{\mathsf{c}} p\colon A\rrbracket_{\mathbb{H}} = \langle \alpha, d, e\rangle$, $t \le d$ and $e^t = v$;
(2) if $p, d ⟱ v$ and $p \Downarrow d, w$ then $v = w$, $\llbracket - \vdash_{\mathsf{c}} p\colon A\rrbracket_{\mathbb{H}} = \langle \mathsf{cc}, d, e\rangle$ and $e^d = v$.

Proof. First, let us argue that for a fixed $- \vdash_{\mathsf{c}} p\colon A$, (1) implies (2). Indeed, $p, d ⟱ v$ by the first clause implies that $\llbracket - \vdash_{\mathsf{c}} p\colon A\rrbracket_{\mathbb{H}} = \langle \alpha, d, e\rangle$, $e^d = v$ and by Theorem 5.8, $\llbracket - \vdash_{\mathsf{c}} p\colon A\rrbracket_{\mathbb{Q}} = \langle d, w\rangle$. By Lemma 6.5, $e^d = w = v$ and $\alpha = \mathsf{cc}$.

The proof of (1) is now obtained analogously to the proof of Theorem 5.8, by induction over the derivation $p, t ⟱ v$ in the proof system of Figure 8. We occasionally need to call both (1) and (2) in the induction step, but, as we argued (1) implies (2) for every specific p. Consider, e.g. the rule

$$(\mathbf{seq}^{\mathsf{e}}_2)\quad \frac{\begin{array}{cc} p \Downarrow d, v' & (p, s ⟱ v_s)_{s \le d} \\ (q[v_s/x], 0 ⟱ w_s)_{s \le d} & q[v_d/x], t ⟱ w \end{array}}{x \coloneqq p;\, q, d + t ⟱ w}$$

in detail. By induction hypothesis, $\llbracket - \vdash_{\mathsf{c}} p\colon A\rrbracket_{\mathbb{H}} = \langle \mathsf{cc}, d, e\rangle$, $e^d = v_d$, $\llbracket - \vdash_{\mathsf{c}} q[v_d/x]\colon B\rrbracket_{\mathbb{H}} = \langle \alpha, d_\star, e_\star\rangle$, $t \le d_\star$, $e_\star^t = w$. Now, by definition, and using Lemma 6.1,

$$\begin{aligned}&\llbracket - \vdash_{\mathsf{c}} x \coloneqq p;\, q\colon A\rrbracket_{\mathbb{H}}\\ &\quad = \langle f_{\mathsf{tt}}(e^d), d + f_{\mathsf{dr}}(e^d), \lambda s.\ f^0_{\mathsf{ev}}(e^s) \lhd s < d \rhd f^{s-d}_{\mathsf{ev}}(e^d)\rangle\\ &\quad = \langle f_{\mathsf{tt}}(v_d), d + f_{\mathsf{dr}}(v_d), \lambda s.\ f^0_{\mathsf{ev}}(v_s) \lhd s < d \rhd f^{s-d}_{\mathsf{ev}}(v_d)\rangle\end{aligned}$$

where $f = \llbracket x\colon B \vdash_{\mathsf{c}} q\colon A\rrbracket_{\mathbb{H}}$, and therefore $d + t \le d + d_\star = d + f_{\mathsf{dr}}(v_d)$ and

$$f^0_{\mathsf{ev}}(v_{d+t}) \lhd d + t < d \rhd f^{d+t-d}_{\mathsf{ev}}(v_d) = f^t_{\mathsf{ev}}(v_d) = e_\star^t = w,$$

as desired. □

The following lemma essentially captures the adequacy direction of Theorem 6.3.

Lemma 6.7. *If* $[\![- \vdash_{\mathsf{c}} p \colon A]\!]_{\mathbb{H}} = \langle \alpha, d, e\rangle$ *and* $t \leq d$ *then* $p, t \Downarrow v$ *for some* v.

Proof Sketch. The proof is analogous to the corresponding fragment of the proof of Theorem 5.8. Let us consider the case of while-loops, which is the trickiest one. That is, we have to show that $[\![- \vdash_{\mathsf{c}} x \coloneqq p \;\mathsf{while}\; b\; \{q\} \colon A]\!]_{\mathbb{H}} = \langle \alpha, d, e\rangle$ and $t \leq d$ imply that $x \coloneqq p \;\mathsf{while}\; b\; \{q\}, t \Downarrow v$ for some v. By expanding the assumption we obtain

$$\langle \alpha, d, e\rangle =$$

$$\big((\lambda\langle x, c\rangle.\,(H\,\mathsf{inr})(\kappa(l(x))) \lhd b(x) \wedge c \rhd \eta(\mathsf{inl}\, x))^{\dagger}\big)^{\star}(\kappa(\alpha_0, d_0, e_0))$$

where $b = [\![x \colon A \vdash_{\mathsf{v}} v \colon 2]\!]$, $\langle \alpha_0, d_0, v_0\rangle = [\![- \vdash_{\mathsf{c}} p \colon A]\!]_{\mathbb{H}}$ and $l = [\![x \colon A \vdash_{\mathsf{c}} q \colon A]\!]_{\mathbb{H}}$. If $\alpha_0 = \mathsf{od}$ or $\alpha_0 = \mathsf{cd}$, the above equation yields $\alpha_0 = \alpha$, $d = d_0$ and $e = e_0$ and we obtain the requisite judgement using the induction hypothesis either by $(\mathbf{wh}^{\mathbf{e}}_1)$ or by $(\mathbf{wh}^{\mathbf{e}}_2)$. The same considerations apply if $t < d_0$. We proceed under the assumption that $\alpha_0 = \mathsf{cc}$ and $t \geq d_0$. Consider the sequence

$$\langle \mathsf{cc}, d_0, e_0\rangle, \langle \mathsf{cc}, d_1, e_1\rangle, \ldots \tag{7}$$

iteratively constructed as follows: d_{i+1} and e_{i+1} are defined as long as $\langle \mathsf{cc}, d_{i+1}, e_{i+1}\rangle = l(e_i^{d_i})$ and $b(e_i^{d_i})$ is true. If any of the latter conditions eventually fails, the sequence (7) terminates resulting in $d = d_0 + \ldots + d_n$ for some t. We then construct the requisite derivation of $x \coloneqq p \;\mathsf{while}\; b\; \{q\}, t \Downarrow v$ by calling the induction hypothesis. If the sequence (7) is infinite, still, by Lemma 6.4 (1), for some n, $d_0 + \ldots + d_n \leq t < d_0 + \ldots + d_{n+1}$ and $e^t = e_n^{t-d_0-\ldots-d_n}$. Again, by induction, we can construct a derivation of the judgement $x \coloneqq p \;\mathsf{while}\; b\; \{q\}, t \Downarrow e_n^{t-d_0-\ldots-d_n}$. □

Finally, we can prove the soundness and adequacy theorem that was previously mentioned.

Proof (Theorem 6.3). The Lemmas 6.6, 6.7 and 6.5 jointly imply the theorem as follows. The left-to-right directions for both clauses are already in Lemma 6.6. For the right-to-left direction of the first clause, Lemma 6.7 produces some value v, which, again by the soundness direction must be the one for which $e^t = v$. For the right-to-left direction of the second clause we additionally use Lemma 6.5. □

7 CONCLUSIONS AND FURTHER WORK

Our present work is a result of an ongoing effort to establish solid semantic foundations for hybrid computation. More specifically, here we complement our previous work on denotational models [16, 17, 34] in terms of *hybrid monads*, with a corresponding operational semantics and connect both styles of semantics by a soundness and adequacy theorem. The central ingredient of our framework is a hybrid, call-by-value while-language HybCore, which we put forward as syntactic means for capturing the essence of hybrid computation, subsequently study it, and use as basis of future, more complex hybrid programming languages. The task of formulating an adequate operational semantics turned out to require a sophisticated route via an auxiliary lightweight duration semantics, and we regard the fact of success, i.e. the fact that the term "adequate operational semantics" can actually be sensibly interpreted in the hybrid setting, as a striking outcome of this work.

HybCore provides a minimal framework combining the basic and uncontroversial features of hybrid computation with clear and principled semantic foundations. We plan to use it both as a stepping stone and as a yardstick in further work. The fact that HybCore purposefully combines the classical view of (instantaneous) computations – which either succeed immediately or silently diverge (e.g. variable assignments in a loop) – and processes extended over real-time, makes the operational semantics of HybCore particularly distinct, specifically, the rules generally require *continuum-size* numbers of premises, which is needed to ensure the integrity of the resulting trajectories. This kind of complexity vanishes when restricting to progressive semantics, which is obtained by forbidding empty trajectories. The submonad $\mathbb{H}_{+}$ of $\mathbb{H}$ that arises from this restriction, was studied in [16] and enjoys better properties than $\mathbb{H}$; we expect our adequacy result to restrict accordingly. This task is however not entirely trivial, as it requires a considerable adaptation of the language to keep track of progressiveness of terms, yet a systematic approach to this issue was proposed recently [18].

As indicated in the introduction, the use of monads gives access to various generic scenarios for combining hybridness with other effects, specifically via universal constructions and monad transformers. One generic construction that can be applied to $\mathbb{H}$ is the *generalized coalgebraic resumption monad transformer* [19] sending an endofunctor T to the coalgebra $T_F = \nu\gamma.\, T(-+F\gamma)$ for a given endofunctor F, which can be chosen suitably for modelling *interactive/concurrent* features, e.g. $F = A \times -$ corresponds to automata or process algebra-style actions ranging over A. By the abstract results in [19] for an Elgot monad $\mathbb{T}$, T_F canonically extends to an Elgot monad $\mathbb{T}_F$, which is indeed the case for $\mathbb{T} = \mathbb{H}$. A disciplined way of adding further effects to HybCore is by resorting to Plotkin and Power's *generic effects* [37]. E.g. for binary non-determinism, we would need a coin-tossing primitive $\Gamma \vdash_{\mathsf{c}} \mathsf{toss}() \colon 2$ for choosing a control branch non-deterministically, yielding a non-deterministic choice operator in the following way (a similar approach applies e.g. to probabilistic choice):

$$p + q = (b \coloneqq \mathsf{toss}();\ \mathsf{if}\ b\ \mathsf{then}\ p\ \mathsf{else}\ q).$$

Denotationally, in order to interpret such extensions of HybCore we need to modify $\mathbb{H}$ by applying suitable monad transformers, in particular, we plan to extend previous work on combining iteration-free versions of $\mathbb{H}$ with other monads [12]. Our commitment to the language of categorical constructions raises hopes that the presented developments can be carried over to categories other than **Set** and more suitable for analysing topological aspects of hybrid dynamics, such as *stability* of hybrid behaviour under input perturbations. E.g. the construction of the duration monad as a quotient of the corresponding layered duration monad can be instantly reproduced in the category **Top** of topological spaces, therefore inducing a necessary topologization in a canonical way. It remains an important goal of further work to provide an analogous treatment for the hybrid monad $\mathbb{H}$, possibly by introducing a *layered hybrid monad* whose quotient would be $\mathbb{H}$. Our long-term goal is to extend HybCore to a higher-order hybrid language with general recursion.

ACKNOWLEDGMENTS

The first author would like to acknowledge the support of German Research Foundation under grant GO 2161/1-2.

The second author would like to acknowledge the support of ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme and by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project POCI-01-0145-FEDER-030947.

REFERENCES

[1] Samson Abramsky. 2014. Intensionality, Definability and Computation. In *Johan van Benthem on Logic and Information Dynamics*, Alexandru Baltag and Sonja Smets (Eds.). Springer, 121–142.

[2] Jiří Adámek, Stefan Milius, and Jiří Velebil. 2011. Elgot theories: a new perspective of the equational properties of iteration. *Math. Struct. Comput. Sci.* 21, 2 (2011), 417–480.

[3] Rajeev Alur. 2015. *Principles of Cyber-Physical Systems.* MIT Press.

[4] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho. 1993. Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. In *Workshop on the Theory of Hybrid Systems, Denmark. 1992 (Lecture Notes in Computer Science)*, Robert L. Grossman, Anil Nerode, Anders P. Ravn, and Hans Rischel (Eds.), Vol. 736. Springer, 209–229.

[5] Steve Awodey. 2010. *Category Theory.* OUP Oxford.

[6] Stephen Bloom and Zoltán Ésik. 1993. *Iteration theories: the equational logic of iterative processes.* Springer.

[7] David Broman, Edward A. Lee, Stavros Tripakis, and Martin Törngren. 2012. Viewpoints, formalisms, languages, and tools for cyber-physical systems. In *MPM@MoDELS'12: Multi-Paradigm Modeling, 6th International Workshop, Innsbruck, Austria, October 1-5, 2012*, Cécile Hardebolle, Eugene Syriani, Jonathan Sprinkle, and Tamás Mészáros (Eds.). ACM, 49–54.

[8] Venanzio Capretta. 2005. General recursion via coinductive types. *Log. Meth. Comput. Sci.* 1, 2 (2005).

[9] James Chapman, Tarmo Uustalu, and Niccolò Veltri. 2015. Quotienting the Delay Monad by Weak Bisimilarity. In *Theoretical Aspects of Computing, ICTAC 2015 (LNCS)*, Vol. 9399. Springer, 110–125.

[10] J. Robin B. Cockett. 1993. Introduction to Distributive Categories. *Mathematical Structures in Computer Science* 3, 3 (1993), 277–307.

[11] Dion Coumans and Bart Jacobs. 2013. Scalars, monads, and categories. In *Quantum physics and linguistics. A compositional, diagrammatic discourse.*, Chris Heunen; Mehrnoosh Sadrzadeh and Edward Grefenstette (Eds.). Oxford University Press, 184–216.

[12] Fredrik Dahlqvist and Renato Neves. 2018. Compositional semantics for new paradigms: probabilistic, hybrid and beyond. *arXiv preprint arXiv:1804.04145* (2018).

[13] Nils Anders Danielsson. 2012. Operational Semantics Using the Partiality Monad. In *Proceedings of the 17th ACM SIGPLAN International Conference on Functional Programming (ICFP '12)*. ACM, New York, NY, USA, 127–138.

[14] Calvin Elgot. 1975. Monadic Computation And Iterative Algebraic Theories. In *Logic Colloquium 1973 (Studies in Logic and the Foundations of Mathematics)*, Vol. 80. Elsevier, 175–230.

[15] Bram Geron and Paul Blain Levy. 2016. Iteration and Labelled Iteration. In *Mathematical Foundations of Programming Semantics, MFPS XXXII (ENTCS)*, Vol. 325. Elsevier, 127 – 146.

[16] Sergey Goncharov, Julian Jakob, and Renato Neves. 2018. A Semantics for Hybrid Iteration. In *29th International Conference on Concurrency Theory (CONCUR 2018) (LNCS)*, Sven Schewe and Lijun Zhang (Eds.). Springer.

[17] Sergey Goncharov, Julian Jakob, and Renato Neves. 2018. A Semantics for Hybrid Iteration. *CoRR* abs/1807.01053 (2018). arXiv:1807.01053 http://arxiv.org/abs/1807.01053

[18] Sergey Goncharov, Christoph Rauch, and Lutz Schröder. 2018. A Metalanguage for Guarded Iteration. In *15th International Colloquium on Theoretical Aspects of Computing (ICTAC 2018)*, Bernd Fischer and Tarmo Uustalu (Eds.).

[19] Sergey Goncharov, Lutz Schröder, Christoph Rauch, and Julian Jakob. 2018. Unguarded Recursion on Coinductive Resumptions. *Logical Methods in Computer Science* 14, 3 (2018).

[20] Sergey Goncharov, Lutz Schröder, Christoph Rauch, and Maciej Piróg. 2017. Unifying Guarded and Unguarded Iteration. In *Foundations of Software Science and Computation Structures, FoSSaCS 2017 (LNCS)*, Javier Esparza and Andrzej Murawski (Eds.), Vol. 10203. Springer, 517–533.

[21] Thomas A. Henzinger. 1996. The Theory of Hybrid Automata. In *LICS96': Logic in Computer Science, 11th Annual Symposium, New Jersey, USA, July 27-30, 1996.* IEEE, 278–292.

[22] Peter Höfner and Bernhard Möller. 2009. An algebra of hybrid systems. *The Journal of Logic and Algebraic Programming* 78, 2 (2009), 74 – 97.

[23] Peter Höfner and Bernhard Möller. 2011. Fixing Zeno gaps. *Theoretical Computer Science* 412, 28 (2011), 3303 – 3322. Festschrift in Honour of Jan Bergstra.

[24] Marco Kick, John Power, and Alex Simpson. 2006. Coalgebraic semantics for timed processes. *Inf. Comput.* 204, 4 (2006), 588–609.

[25] K. D. Kim and P. R. Kumar. 2012. Cyber-Physical Systems: A Perspective at the Centennial. *Proc. IEEE* 100, Special Centennial Issue (May 2012), 1287–1308.

[26] Anders Kock. 1972. Strong Functors and Monoidal Monads. *Archiv der Mathematik* 23, 1 (1972), 113–120.

[27] Paul Blain Levy, John Power, and Hayo Thielecke. 2002. Modelling Environments in Call-By-Value Programming Languages. *Inf. & Comp* 185 (2002), 2003.

[28] Xinzhi Liu and Peter Stechlinski. 2017. *Infectious Disease Modeling.* Springer.

[29] Saunders Mac Lane. 1971. *Categories for the Working Mathematician.* Springer.

[30] Stefan Milius. 2005. Completely iterative algebras and completely iterative monads. *Inf. Comput.* 196, 1 (2005), 1–41.

[31] Eugenio Moggi. 1991. Notions of Computation and Monads. *Inf. Comput.* 93 (1991), 55–92.

[32] Katsunori Nakamura and Akira Fusaoka. 2005. On Transfinite Hybrid Automata. In *Hybrid Systems: Computation and Control*, Manfred Morari and Lothar Thiele (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 495–510.

[33] Renato Neves. 2018. *Hybrid programs.* Ph.D. Dissertation. Minho University.

[34] Renato Neves, Luís S. Barbosa, Dirk Hofmann, and Manuel A. Martins. 2016. Continuity as a computational effect. *Journal of Logical and Algebraic Methods in Programming* 85, 5 (2016), 1057–1085.

[35] Lawrence Perko. 2013. *Differential equations and dynamical systems.* Vol. 7. Springer Science & Business Media.

[36] André Platzer. 2008. Differential Dynamic Logic for Hybrid Systems. *J. Automated Reasoning* 41, 2 (01 Aug 2008), 143–189.

[37] Gordon Plotkin and John Power. 2001. Adequacy for Algebraic Effects. In *FoSSaCS'01 (LNCS)*, Vol. 2030. 1–24.

[38] J. Reynolds. 1998. *Theories of Programming Languages.* Cambridge University Press.

[39] Kohei Suenaga and Ichiro Hasuo. 2011. Programming with infinitesimals: A while-language for hybrid system modeling. In *International Colloquium on Automata, Languages, and Programming.* Springer, 392–403.

[40] Tarmo Uustalu. 2002. Generalizing substitution. In *Fixed Points in Computer Science, FICS 2002, Copenhagen, Denmark, 20-21 July 2002, Preliminary Proceedings*, Vol. NS-02-2. University of Aarhus, 9–11.

[41] Tarmo Uustalu. 2003. Generalizing Substitution. *ITA* 37, 4 (2003), 315–336.

[42] Tarmo Uustalu and Varmo Vene. 1999. Primitive (Co)Recursion and Course-of-Value (Co)Iteration, Categorically. *Informatica (Lithuanian Academy of Sciences)* 10, 1 (1999), 5–26.

[43] G. Winskel. 1993. *The Formal Semantics of Programming Languages.* MIT Press, Cambridge, Massachusetts.

[44] Hans Witsenhausen. 1966. A class of hybrid-state continuous-time dynamic systems. *IEEE Trans. Automat. Control* 11, 2 (1966), 161–167.