RENATO NEVES

HYBRID PROGRAMS

HYBRID PROGRAMS

RENATO NEVES

Minho Aveiro Porto - Doctoral Programme in Computer Science

Winter of 2017/2018

This work was financed by FCT – Fundação para a Ciência e a Tecnologia – with the grant SFRH/BD/52234/2013. Additional support was provided by the PT-FLAD Chair on Smart Cities & Smart Governance and by the project Dalí (POCI-01-0145-FEDER-016692), the latter funded by ERDF – European Regional Development Fund – through COMPETE 2020 – Operational Programme for Competitiveness and Internationalisation – together with FCT.

RESUMO

Esta tese estuda sistemas híbridos, uma família emergente de dispositivos que envolvem diferentes interações entre computações digitais e processos físicos. Estes sistemas estão rapidamente a tornar-se elementos-chave da engenharia de software, o que é explicado pela necessidade de desenvolver produtos que interagem com os atributos físicos do seu ambiente *e. g.* velocidade, tempo, energia, e temperatura – exemplos típicos variam de micro-sensores e pacemakers, a veículos autónomos, infra-estruturas de transporte, e redes eléctricas distritais. Mas ainda que amplamente usados, estes sistemas são geralmente desenvolvidos de forma pouco sistemática nas prácticas de programação atuais.

O objetivo deste trabalho é isolar as interações básicas entre computações digitais e processos físicos, e subsequentemente desenvolver o paradigma de programação subjacente. Para fazer isto de forma precisa, a nossa base de trabalho irá ser a teoria das mónadas, uma estrutura categórica para o desenvolvimento sistemático de semânticas na programação. A partir desta base, provamos a existência de uma mónada que capta as interações acima mencionadas, e usamo-la para desenvolver e examinar os fundamentos do paradigma de programação correspondente a que chamamos *programação híbrida*: mostramos como construir, de maneira metódica, diferentes linguagens de programação que acomodam amplificadores, equações diferenciais, e atribuições - os ingredientes básicos dos sistemas híbridos - caracterizamos todas as operações sobre programas disponíveis, introduzimos construções if-then-else, operações para lidar com excepções, e diferentes tipos de feedback.

Os sistemas híbridos trazem vários aspectos da teoria de controlo para a ciência da computação. Um destes é a noção de *estabilidade*, que se refere à capacidade de um sistema de evitar mudanças drásticas no seu output se pequenas variações no seu estado ou input ocorrerem. Neste trabalho, desenvolvemos uma noção *composicional* de estabilidade para a programação híbrida. Introduzimos também programas híbridos com memória interna, que formam a base de uma disciplina de *desenvolvimento de software baseado em componentes.* Desenvolvemos a sua teoria coalgébrica, nomeadamente linguagens, noções de comportamento e bisimulação. Neste processo, introduzimos também novos resultados teóricos sobre coalgebra, incluindo melhorias a resultados conhecidos e provas acerca da existência de noções de comportamento para sistemas de transição não determinísiticos com espaço de estados infinitos.

ABSTRACT

This thesis studies hybrid systems, a family of devices that intertwine digital computations and physical processes. They are very quickly becoming a main concern in software engineering, because more and more often software products closely interact with physical aspects of the environment *e. g.* movement, time, energy, temperature – examples range from micro-sensors and pacemakers, to transport infrastructures and district-wide electric grids. But even if already widespread, the ability of these systems to combine programs and physical processes, renders their development a challenging task that is still largely unmet by the current programming practices.

Our goal is to address this challenge at its core: we wish to isolate the possible, basic interactions between discrete computations and physical processes, and bring forth the *programming paradigm* that naturally underlies them. In order to do so in a precise and clean way, we resort to monad theory, a well established categorical framework to develop program semantics in a systematic way. We prove the existence of a monad that naturally encodes the aforementioned interactions, and use it to develop and examine the foundations of the paradigm alluded above, which we call *hybrid programming.* We show how to build in a methodical way different programming languages that accommodate amplifiers, differential equations, and discrete assignments – the basic ingredients of hybrid systems – we list all program operations available in the paradigm, introduce if-then-else constructs, abort operations, and different types of feedback.

Hybrid systems bring several important aspects of control theory into computer science. One of them is the notion of *stability* – the system's capacity of avoiding significant changes in its output if small variations in its state or input occur. We introduce a notion of stability to hybrid programming, explore it, and show how to analyse hybrid programs with respect to it in a *compositional* manner.

We also introduce hybrid programs with internal memory and show that they form the basis of a *component-based software development* discipline in hybrid programming. We develop their coalgebraic theory, namely languages, notions of behaviour, and bisimulation. In the process we introduce new theoretical results on coalgebra, including improvements to well-known results and proofs on the existence of suitable notions of behaviour for non-deterministic transition systems with infinite state spaces.

ACKNOWLEDGMENTS

"Man is a social animal". At some point this was the opening sentence of my sister's thesis – it was great choice of words but she later changed it. I am still not sure why. The late artist Eugène Delacroix, wrote in his diary what might seem an opposing statement: *"Nourish yourself with grand and austere ideas of beauty that feed the soul... Seek solitude"*. This idea was emphasised in the past millenia by many illuminated minds, popularised by different religions, and in many cases took to extreme lengths. But it is perhaps the case that both statements depend on each other. To truly enjoy solitude one needs a strong circle of friends, and friendship only flourishes if there is space for solitude, similarly to the belief that happiness and sadness (or good and evil) are mutually dependent. The current page, however, takes a one side view. It is completely devoted to what my sister wrote, not the symbiosis between solitude and friendship, but to the latter, to all friends and colleagues that made this thesis possible.

Nothing more could be expected of my supervisors, Luís and Manuel. Their knowledge, wisdom, patience, and, above all, friendship were greatly appreciated. I hope I can enjoy these beautiful traits of them for many more years. Dirk was, for all practical purposes, a third supervisor and also a friend. I owe to him many of the things I know about mathematics and I hope he is willing to teach me much more.

Some truly productive times of solitude were made possible by Alexandra and Ichiro, who so warmly welcomed me in London, Nijmegen, and Tokyo, and helped me with so many cool ideas and insights. I am very grateful to them. A very big thanks to my colleagues and friends, in particular, Alexandre, Ana, Carlos, Cláudio, Eros, Fred, Guillermina, Jorge, José, Leandro, Matteo, Pedro, Ricardo, and Vera. They are the proof that man is indeed a social animal.

Finally, this thesis is dedicated to my family, in particular, my parents, sister, and of course my partner in life: Carina, I do not believe there are words to suitably express what I feel about you; my only is hope is that this can be done with actions in some way or another. The great artist Louise Borgeois wrote: *"You are born alone. You die alone. The value of the space in between is trust and love"*. You, and all people mentioned in this note, are my own space in between.

Renato Neves,
Porto, 12th March 2018

CONTENTS

1 PROLOGUE

1.1 HYBRID SYSTEMS AND THE CHALLENGE THEY SET FORTH

Programs and *physical processes* are combined more and more often in programming practice though a lot of times implicitly. This tendency follows from the need to deliver software products that closely interact with physical processes, like velocity, movement, energy, and time. Examples of such systems are numerous, ranging from small medical devices, like pacemakers and infusion pumps, to surgical robots, autonomous vehicles, and district-wide electric grids. The general belief is that many more of these systems will be introduced to society in the coming years, and will become a crucial part of the twenty-first century's technology [Raj+10; KK12; Alu15; LS16].

Their rigorous development, however, is still an extremely difficult challenge. Adding up to the intrinsic difficulties in correct program design, which are already arduous to handle *per se*, the engineer now faces an intricate interaction between the digital and physical worlds. His/her view of a system must encompass not only the behaviour of software, but also the evolution of physical processes over time, which requires skills from computer science, analysis, and control theory.

Systems that require this global perspective are traditionally designated as *hybrid* and constitute the *raison d'être* of this thesis.

1.2 OUR GOAL

Our goal is to provide a detailed and rigorous account on the combination of programs and physical processes: we wish to know if both can be organised within a single unifying framework, and to discern and analyse the possible types of interaction between them.

- *Can both concepts be considered systematically in a single programming language ?*
- *And if so, which operations and programming features can(not) such a language support ?*
- *Can central notions of control theory, like stability, convergence, and feedback, be embedded in the language ?*

Consider for example the following program, written in an imperative style.

```
int cool_or_heat() {
  double target = 300;
  if (temp <= target) {
      (dtemp = 1 & 3);       // Heating up
      return 0;              // Success
```

```
    }  else {
       (dtemp = -1 & 3);      // Cooling down
       return 0;              // Success
    }
  }
```

Its purpose is to manage the temperature of a reactor. There exists a global variable (`temp`) that registers the current temperature, the expression `(dtemp = 1 & 3)` dictates how the temperature is going to evolve for the next three miliseconds, and similarly for `(dtemp = -1 & 3)`. *Can we provide a suitable semantics for programs of this kind ? And if so, which programming features will such a semantics support ?*

Imagine now that at very high temperatures the program raises exceptions or even starts to behave non-deterministically. *Can the previous semantics be extended to accomodate those features ?* Let us now consider the following variation of the previous program.

```
  void cool_or_heat() {
    double target = 300;
    while (true) {
      if (temp <= target) {
         (dtemp = 1 & 3);      // Heating up
      }  else {
         (dtemp = -1 & 3);     // Cooling down
      }
    }
  }
```

Instead of waiting for a call to either increase or decrease the reactor's temperature, this program autonomously checks the current temperature every three milliseconds, and manages it according to the variable `target`. *Can we interpret while loops such as one used in this program ? And now putting the control theorist's hat, will the variable `temp` converge to the target temperature ? If not, can we at least show that it converges to the target temperature up to a reasonable margin of error ?*

1.3 CONTEXTUALISATION

A rigorous formulation of the questions above demands that we fix from the outset suitable notions of what a physical process and a program are. Regarding the former, we follow the perspective advocated by H. Poincaré in the late nineteenth century [Hol90], which is now standard in control theory and analysis: physical processes are *continuous dynamical systems for the additive monoid of non-negative reals*, the usual examples being the solutions of differential equations. Regarding the latter, we adopt the notion proposed by E. Moggi around thirty years ago [Mog89]: programs are *morphisms in the Kleisli category of a monad and their sequential composition is simply Kleisli composition.*

1.4 RESEARCH SUMMARY

Moggi's interpretation of a program is highly generic: different types of program are captured by different monads which propounds the question *is there a monad that allows to interpret both discrete assignments and physical processes as programs ?* This is the genesis of the thesis' research, which shows that one such monad indeed exists. We call it hybrid monad.

The programming languages field has a long history on programming paradigms whose development and analysis unavoidably required studying (if it exists) their underlying monadic structure. In the same spirit, the hybrid monad is here taken as *the basic element of hybrid programming* and this will allow us to answer the questions asked in Section 1.2 in a rigourous manner. The hybrid monad takes therefore an important role in the three research lines that the thesis explores.

Research line 1. *Foundations of hybrid programming (Chapter 3)*

Our goal is to investigate the basic features of hybrid programming. We will examine program operations, if-then-else constructs, programming languages, and different types of iteration. Overall, the study of these aspects will provide very fine grained results on what to expect of hybrid programs and corresponding composition operators. For example, hybrid programs do not support any 'failure operation' whatsoever, but they do admit non-terminating loops. The main contributions of this research line are summarised next.

1. *Kleisli representations* (Definition 3.1.14). A generic, monadic framework to build and analyse *compositional semantics for programming languages.* We will use it to generate hybrid programming languages, analyse their operations, and investigate the possibility of adding classical programming constructs such as abort, exception-handling, and non-deterministic operations.

2. *The hybrid monad* (Theorem 3.2.6). The main ingredient in our analysis of hybrid programming. Among other things, it allows to consider discrete assignments, differential equations, and amplifiers all in the same programming language.

3. *Program operations* (Theorem 3.3.1). We show how to fully describe all n-ary natural transformations $(FU)^n \to FU$ where F is a functor of a certain type and U is a right adjoint. This will allow us to list all program operations that the hybrid paradigm supports, and also how to build them in a simple manner.

4. *Iteration* (Section 3.4). We introduce and analyse two types of iteration for hybrid programs.

5. *Combination with non-determinism* (Theorem 3.5.2). We prove the existence of a distributive law between the non-deterministic and hybrid paradigms. This yields a new

monad, which, among other things, brings forth programming languages that accomodate non-deterministic assignments, differential predicates, and new types of iteration.

These results were achieved in collaboration with the following people: L. Barbosa, F. Dahlqvist, D. Hofmann, and M. Martins. The hybrid monad and a brief study of its Kleisli category were published in a journal paper [Nev+16b]:

- Renato Neves, Luis S. Barbosa, Dirk Hofmann, and Manuel A. Martins. "Continuity as a computational effect". In: Journal of Logical and Algebraic Methods in Programming 85.5 (2016), pp. 1057–1085.

The author and F. Dahlqvist also submitted a paper on Kleisli representations and on the combination with non-determinism to an international conference. It is currently under review.

Research line 2. *Hybrid programs with internal memory (Chapter 4)*

Our next goal is to examine hybrid programs with internal memory, which, as we will show, can be encoded by different types of *coalgebra*. Our interest in such programs comes from their ability to naturally represent a digital device (seen as a *black-box system*) that interacts with a physical process from time to time. In this context, the discrete behaviour of the digital device is hidden from the environment but the continuous behaviour of the physical process is observable from the outside. The cruise control system is a classical example: the computations of its digital device are *not* directly observable, but the vehicle's velocity and movement are. In fact, one can consider a component `crs_cntrl` that is only possible to interact with via two methods: one for setting the desired speed,

```
crs_cntrl.set_target_speed(double);
```

and the other for moving the car forward during a given amount of time,

```
crs_cntrl.move_forw_during(double);
```

This perspective complements that of the previous research line. Whilst in the latter *discrete and continuous behaviour is mixed* by seeing hybrid programs *algebraically*, *i. e.* emphasis is put on data and data manipulation, in the former *discrete and continuous behaviour is kept separated* by seeing hybrid programs *coalgebraically*, the emphasis being put on observational behaviour, evolution, and interaction with the environment. Under this perspective, in which internal states and digital computations are not directly observable, notions of bisimulation, behaviour, and behavioural description languages become central elements in the game. We show how to lift them to the hybrid programming context using standard results of coalgebra. The main contributions of this research line are summarised below.

1. *Semantics* (Section 4.2 and Section 4.4). Each endofunctor $F : \mathsf{Set} \to \mathsf{Set}$ induces a category of hybrid programs with internal memory and whose discrete behaviour is 'shaped' by F. If F is the powerset functor we recover hybrid automata (the standard formalism

of hybrid systems) and if it is the distribution functor we recover probabilistic hybrid automata instead (another well-known formalism of hybrid systems).

2. *Languages* (Section 4.2). Using documents [BRS09; Sil10], we show how to generate for free canonical languages for different types of hybrid program with internal memory, including hybrid automata. This is illustrated in the deterministic setting, where the endofunctor F is assumed to be the identity.

3. *Bisimulation* (Theorem 4.4.6 and Theorem 4.4.7). We introduce a coalgebraic notion of bisimulation for hybrid programs with internal memory and show that it generalises the notions of bisimulation adopted by classical and probabilistic hybrid automata.

These results were achieved in collaboration with L. Barbosa and published in [NB16; NB17]:

- Renato Neves and Luis S. Barbosa. "Hybrid Automata as Coalgebras". In: Theoretical Aspects of Computing - ICTAC 2016 - 13th International Colloquium, Taipei, Taiwan. Lecture Notes in Computer Science. Springer, 2016, pp. 385–402.

- Renato Neves and Luis S. Barbosa. "Languages and models for hybrid automata: A coalgebraic perspective". In: Theoretical Computer Science, 2017. (In Press)

Research line 3. *When sets do not suffice (Chapter 5)*

Even if perhaps not clear now, in the two previous research lines a number of difficulties appeared that result from the adoption of Set as the working category (sometimes the latter does not possess a rich enough structure to handle the possible interactions between computational devices and physical processes). Two cases stand out: i) the inability to reason about hybrid programs' *stability*, and ii) the absence of meaningful notions of observational behaviour for certain types of hybrid program with internal memory. The direct implications of the latter case are obvious, but for the former a few remarks are in order since the notion of stability is rarely a concern in computer science.

Studied by A. Lyapunov in his doctoral thesis *"The general problem of the stability of motion"* [Lya92], stability refers to a system's capacity of avoiding significant changes in its output in response to small variations in its intended state or input. For example, a *stable* cruise controller must *not* drastically alter the vehicle's velocity due to small perturbations in the atmosphere, such as small wind changes caused by a truck passing by. In the verification process, one might show that a hybrid program is safe for a initial configuration, but it is *unreasonable* to assume that it always starts with this exact configuration, in the same way that one does not expect a speed sensor to have infinite precision, a thermostat to keep the exact right temperature, or a robot to land at precisely the intended spot. For this reason it is often important to show that the hybrid program being verified is stable.

The goal of this research line is to tackle the two aforementioned issues. Our strategy will be to move from Set to the category Top of topological spaces and continuous maps, and recast part of the theory previously developed in this new setting. This approach will allow us not only to reason about stability *in a compositional manner*, but also to enlarge the class of hybrid programs with internal memory that possess a notion of observational behaviour. In the process we will also obtain several new theoretical results on coalgebra. The main contributions of this research line are summarised below.

1. *Topological coalgebras* (Theorem 5.1.22). Using the notion of topological functor, we show that all categories of (sub)polynomial coalgebras over Top are complete and that their limits are computed as in the analogous category of coalgebras over Set.
2. *Coreflections* (Theorem 5.1.31). We prove that for a subfunctor $F \hookrightarrow G$ the category of F-coalgebras is a *coreflective subcategory* of the category of G-coalgebras. Among other things, this shows that the former has all limits that the latter has.
3. *Limits in categories of Vietoris coalgebras* (Section 5.2). We introduce several new results about limits in categories of coalgebras whose underlying functor is a Vietoris polynomial one. These will be used to enlarge the class of hybrid programs with internal memory that possess a meaningful notion of observational behaviour.
4. *Hybrid monad on* Top (Theorem 5.3.5). The basis of stability in hybrid programming. It will allow us not only to reason about hybrid programs' stability in a compositional manner, but also to generate different hybrid languages in which all programs are ensured to be stable.

These results were achieved in collaboration with L. Barbosa, D. Hofmann, and P. Nora. An article on the stability of hybrid programs is currently in preparation, and the other results are published in [HNN18]:

- Dirk Hofmann, Renato Neves, and Pedro Nora. "Limits in Categories of Vietoris Coalgebras". In: Mathematical Structures in Computer Science, 2018. (In Press)

1.5 READING INFORMATION AND NOTATION

We already mentioned that the research lines described above are covered in Chapters 3, 4, and 5. In Chapter 2 we will review the current formalisms for hybrid systems, particularly program semantics and automata, due to their close connection with the thesis. In order to make the text as clear as possible, a few conventions on notation will be followed. In particular,

- Categories will be denoted in serif font, for example C, D, Set ...
- Natural transformations will be denoted by greek letters, for example α, β, γ ...
- Diagram functors will be denoted in calligraphic font, for example $\mathscr{D}$, $\mathscr{K}$, $\mathscr{L}$ ...

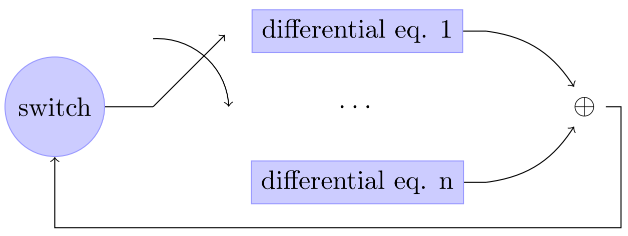
- Monads will be denoted in roman font, for example $\mathbb{T}$, $\mathbb{S}$, $\mathbb{U}$ ...
- Monomorphisms will often be denoted by a tailed arrow $A \rightarrowtail B$ and epimorphisms by a two-headed one $A \twoheadrightarrow B$. Inclusions also receive special notation by being denoted as an hooked arrow $A \hookrightarrow B$ and being labelled by ι_A.
- Consider a locally small category C and a C-object X. Hom functors will be denoted by $\mathsf{C}(X, -) : \mathsf{C} \to \mathsf{Set}$ and $\mathsf{C}(-, X) : \mathsf{C}^{\mathsf{op}} \to \mathsf{Set}$. In some cases we will also use the alternative notation $\hom(X, -) : \mathsf{C} \to \mathsf{Set}$ and $\hom(-, X) : \mathsf{C}^{\mathsf{op}} \to \mathsf{Set}$.
- The right transpose of a function $f : A \times B \to C$ will be denoted by $\overline{f} : A \to C^B$.
- The category of functors from A to B will be denoted by $[\mathsf{A}, \mathsf{B}]$.
- An injection $X_l \rightarrowtail \coprod_{k \in I} X_k$ into a coproduct will often be labelled by i_l, and a projection $\prod_{k \in I} X_k \to X_l$ from a product by π_l.
- The singleton set will be denoted by 1 and similarly for any finite set with cardinality n.
- Given an element $x \in X$ and a set Y the function $Y \to X$ that constantly outputs x will be labelled by $\underline{x}$.

Finally, we will assume that the reader has basic knowledge of category theory and topology. The reader is referred to [Kel55; Gou13; ML98; AHS09] for detailed textbooks and materials on the aforementioned topics.

2 CURRENT FORMALISMS FOR HYBRID SYSTEMS

2.1 A BIRD'S EYE VIEW

The original notion of a hybrid system was introduced by H. Witsenhausen in the sixties [Wit66]. He proposed a formalism based on the idea of differential equations indexed by *operation modes* – which is essentially the key ingredient of hybrid automata, currently the standard formalism of hybrid systems. Briefly, Witsenhausen regarded a hybrid system as a transition device whose operation modes are labelled by systems of differential equations, which specifies the device's behaviour at each particular mode; a jump between two modes occurs *as soon as* the device satisfies a predefined condition. Witsenhausen's idea is often illustrated with the diagram below, which depicts a *switch* that commutes between systems of differential equations when certain conditions are satisfied – the reader familiar with automata theory might also find helpful to see it as a deterministic automaton whose states are labelled by differential equations and whose edges are labelled by guards. Systems with this pattern are nowadays qualified as *switched*.



With the exception of [Tav87], hybrid systems were readdressed only in the nineties, a decade that became vintage for the topic due to the introduction of several models, languages, and tools [MMP91; CRH93; Hen96; Dav97]. A possible reason for this is that most results were built on the field of timed systems, which advanced significantly in the eighties (*cf.* [MMP91; Mal10]). For example, extended duration calculus [CRH93] and hybrid automata [Hen96], two well-known formalisms of hybrid systems, are simple extensions of the duration calculus [CHR91] and timed automata [AD94].

In the following decade, hybrid systems were put under a broader context, qualified as *cyber-physical* and advocated as a *crucial ingredient of the twenty-first century's technology* [Lee06; Lee08; Raj+10; KK12; Gun+14; LS16]. A cyber-physical system is often characterised as a *network* of computational devices that closely interact with physical processes so that a specific goal can be reached. Thus, in contrast to hybrid systems, cyber-physical ones put emphasis not only on the interaction between computational devices and physical processes, but also on concurrency, coordination, and communication.

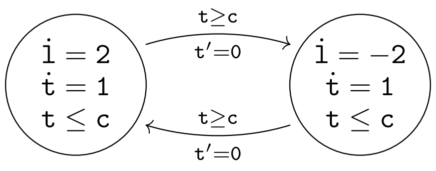
The ensuing section surveys some of the aforementioned formalisms for hybrid systems in more detail; a particular focus is given to automata and program semantics due to their close connection with the thesis. The interested reader will find in [DN00; Pla10; Höf09; Bro+12; Kha08; Ban+15] clear and detailed surveys on other topics, such as process algebras – *e. g.* ϕ-calculus [RS03], hybrid-χ [Bee+06], HCSP [CJR96] – specification languages – *e. g.* the language of linear hybrid action systems [RL01], the infinitesimal hybrid language [SH11], Hybrid Event-B [Ban+15] – and industrial tools – *e. g.* Simulink [Kle07], Modellica [Fri14].

2.2 AUTOMATA AND PROGRAM SEMANTICS

2.2.1 *Automata*

The formal specification and analysis of hybrid systems typically resorts to the theory of hybrid automata [Alu+93; Hen96]. Their distinguishing feature is the ability of variables to evolve continuously while in a state, and to instantaneously change values in a transition between states.

Example 2.2.1. Consider a leaking tank with a valve. The latter allows water to flow in at the rate of 2cm/s during intervals of $\mathtt{c}$ seconds; between these periods it is shut (also) for $\mathtt{c}$ seconds, and the water level drops at 2cm/s. We can describe this behaviour via the hybrid automaton below.



The variable $\mathtt{l}$ denotes the water level, which rises when the valve is open (differential equation $\dot{\mathtt{l}} = 2$) and lowers when the valve is closed (differential equation $\dot{\mathtt{l}} = -2$). The differential equation $\dot{\mathtt{t}} = 1$ defines the passage of time, which, along with invariant $\mathtt{t} \leq \mathtt{c}$, forces the current mode to be active for at most $\mathtt{c}$ seconds. On the other hand, the guard $\mathtt{t} \geq \mathtt{c}$ and assignment $\mathtt{t}' = 0$ force the current mode to be active at least $\mathtt{c}$ seconds before a switch occurs. Finally, note that the guard $\mathtt{t} \geq \mathtt{c}$ does not force a transition to happen, but only makes it possible. This means that, if not for invariant $\mathtt{t} \leq \mathtt{c}$, the valve could be open (or shut) indefinitely.

Being the standard model of hybrid systems, hybrid automata form an active research area that encompasses diverse topics, from decidability issues [Hen96] to extensions that cater for input mechanisms [AH97; Liu+99] and uncertainty [Spr00a; Spr00b]. To create a new extension, however, frequently entails a return to the drawing board in order to redesign or adapt whatever definitions, notions or techniques are deemed relevant. The notion of bisimulation, which we will address later in the thesis, is a prime example of this, as it usually takes an apparently different form in each extension of the theory.

2.2.2 *Program semantics*

There are two well-known program semantics for hybrid systems. We start with one that emerged in the context of dynamic logic, where it serves as the interpretation domain of a famous logic's actions. In order to understand how it came to be, we make a small detour to the topic of logics for hybrid systems.

At around the same time that hybrid automata were introduced, C. Zhou *et al.* developed the so-called extended duration calculus [CRH93]. As hinted by its name, it is an extension of the duration calculus [CHR91; HC97], a logic that features time-dependent predicates and syntax for reasoning about durations; it allows to specify, for example, "predicate p holds for no longer than ten seconds". The extension adds derivative expressions and operators to handle continuous behaviour. However, the notion of derivative falls out of the logic's semantics, which entails external mathematical machinery to handle some aspects of trajectories. Actually, this was also noted by the authors; in [CRH93] they wrote: "*. . . even though differential equation was introduced in the model, we had no means in the logic to reason about such equations*".

Some years later, J. Davoren and A. Nerode proposed μ-calculus to specify and reason about trajectories [Dav97; DN00]: their main argument was that fixpoint operators can be used to handle properties of trajectories (such as invariance, reachability, and Zeno behaviour) quite naturally. They also illustrated approximate specification and verification within the framework by assuming that the states of hybrid systems were endowed with a metric.

More recently, A. Platzer introduced differential dynamic logic [Pla08; Pla10; Pla12] – the one alluded to in this subsection's beginning. Technically speaking, differential dynamic logic, in short $d\mathcal{L}$, is a dynamic first-order logic with real arithmetic support. The corresponding actions form a Kleene algebra with tests [Koz97] whose basic elements are discrete assignments and systems of differential equations – this puts in the same framework the notion of derivative, used to specify the behaviour of physical processes, as was the intention in [CRH93], and fixpoint operators, advocated in [Dav97; DN00] as an important tool for reasoning about trajectories.

Platzer's algebra of actions is an instance of Kleene algebra with tests for the powerset construction: the operators $+$, $;$, and $(-)^*$ are the union, relational composition, and reflexive-transitive closure operators, respectively. And its elements are interpreted as non-deterministic programs. In particular, a system of differential equations s is interpreted as a program $[\![s]\!] : \mathbb{R}^n \to \mathrm{P}(\mathbb{R}^n)$ that for a given input, regarded as the initial value, it outputs *all points of the evolution* that occurs under s.

Remark 2.2.2. Platzer's interpretation $[\![s]\!] : \mathbb{R}^n \to \mathrm{P}(\mathbb{R}^n)$ of a system of differential equations is, therefore, an *abstraction* of the standard interpretation in analysis and control theory: among other things, it abstracts from slopes, notions of convergence, and certain aspects of periodic orbits, concentrating mainly on the notions of safety – a safe region is never left – and liveness – a certain region is eventually reached.

The logic $d\mathcal{L}$ is seen as a landmark: among other things, it has a rich calculus that provides symbolic rules for decomposing complex hybrid programs into simpler ones, as opposed to hybrid automata which are typically harder (and sometimes impossible) to decompose in a suitable manner [Pla10]. Moreover, it has a (semi) automatic verification tool that establishes a powerful verification environment for specifications written in $d\mathcal{L}$ [PQ08]. Cases of success include the discovery of potentially disastrous errors in the design of aircraft collision avoidance manoeuvres [PC09], and the proof that a specific control algorithm of a surgical robot was, in general, unsafe [Kou+13].

The other program semantics that we want to mention is the *weak Kleene algebra* of hybrid systems, introduced by P. Höfner and B. Möller [Höf09; HM09] and clearly another landmark. It has a rich palette of operators, and, notably, it serves as a semantics for both hybrid automata and Platzer's algebra of actions. In its foundations, however, the algebra differs significantly from Platzer's approach: its elements are *not* non-deterministic programs but sets of trajectories, and what is regarded as sequential composition is not relational composition but a form of concatenation of trajectories. Notwithstanding, the algebra [Höf09] is also fundamentally non-deterministic.

Our work in Chapter 3 has some points in common with these projects on program semantics, so let us clarify in which ways the former differs from the latter. Three crucial points should be highlighted:

- we restrict ourselves to a deterministic context whereas they assume non-determinism by default. The latter makes difficult to achieve one of the thesis' goals, namely to isolate the interaction between programs and physical processes, or in other words to isolate *purely hybrid behaviour*. As mentioned in the introduction, our restriction will allow a precise and rigorous analysis of the features supported by the hybrid paradigm and will provide firm answers on how to extend it systematically when new features are in need.

- Our goal is *not to introduce a new semantics* for hybrid programs, but rather to develop and investigate an interpretation domain on which to build new semantics for them. This is illustrated in Chapter 3, where using Kleisli representations and the hybrid monad, we generate *different* hybrid programming languages. In particular, we show that sequential composition in one of these languages captures the deterministic version of sequential composition in [Höf09].

- Our research is performed in a monadic context, which is key for the ensuing chapters' contributions and allows to capitalise on existing literature on monads (*e. g.* [Mog91; BO03; MB06; PP03; Sea13]). The monadic context brings up a number of canonical constructions and smooths the integration with other behavioural effects, such as faulty, non-deterministic, weighted, or probabilistic behaviour – in fact, it promotes a modular view in which the engineer selects the relevant computational effects and composes the corresponding monads (if possible), obtaining as a result a programming paradigm whose properties are completely determined by those of the constituent monads (see [HPP06; DPS17]). Such an integration, however, requires first of all a detailed study of the hybrid monad, which further justifies our restriction to the deterministic setting.

3 FOUNDATIONS OF HYBRID PROGRAMMING

Overview. We introduce the hybrid monad and investigate its basic properties. We show that it is the basis of a programming paradigm in which discrete assignments and physical processes are objects of the same type, allowing the engineer to combine both in a single programming language and thus establishing a connection between the digital and physical worlds.

As mentioned in the introduction we want to analyse this paradigm in detail. Therefore, we develop a *generic*, monadic framework that allows to *systematically* investigate not only which programming features a given paradigm supports, but also how it can be extended with new constructs. By applying this method to the hybrid case, we will generate different hybrid programming languages, list all program operations available, and show precisely when and if important axioms such as commutativity and idempotency hold. We also analyse if-then-else constructs, different types of iteration, and possible combinations with other monads.

Roadmap. We start by reviewing the basic theory of monads and then introduce the monadic framework mentioned above (Section 3.1). After this, we develop the hybrid monad, some of its programming languages, and show how to extend the latter with if-then-else constructs and tests in a systematic manner (Section 3.2). In order to endow these programming languages with additional features, we list all binary program operations supported by the hybrid paradigm (Section 3.3), introduce two types of iteration and investigate their corresponding properties (Section 3.4). Finally, we conclude by highlighting some challenges that emerged from this research and that we think deserve further study, including monad combination and the incorporation of notions of stability in programming languages (Section 3.5).

3.1 PRELIMINARIES: MONADS AND KLEISLI REPRESENTATIONS

The notion of monad, which serves as a concise generalisation of an *algebraic theory*, has been extensively studied in universal algebra since the sixties [Lin66]. In the late eighties, when Moggi proposed to use it as a uniform semantics for programming languages [Mog89; Mog91], it began to be understood also as an important concept of computer science – the idea was later introduced into the programming practice by P. Wadler, which lead to a rigorous style of combining purely functional programs that can mimic side-effects [Wad95]. The core observation in [Mog89; Mog91] is that monads encode in abstract terms several kinds of computational effect, including exceptions, state updates, non-determinism, and probabilistic behaviour. Under this view, a computational effect is given by a type constructor T (technically, an endofunctor) and computations are elements of TY for some type Y. The denotation of a program `p` is then a morphism $[\![\mathsf{p}]\!] : X \to TY$ and sequential composition of programs, as mentioned before, boils down to Kleisli composition.

In documents [PP01b; PP01a; PP03] G. Plotkin and J. Power extended this basic semantics with program operations using the notion of algebraic operation – *i. e.* a natural transformation $T^n \to T$ that respects the multiplication of T in a certain way. However, as already noted in [PP01b; PP01a; PP03], the condition concerning the multiplication of T is frequently too strict: it forbids, for example, exception-handling operations, the dual operator of game logic [HKL14], and *non* right-distributive operations [PP01b; PP01a; PP03]. This is the reason why more recently the authors of [HKL14; HK15] do not require that natural transformations $T^n \to T$ respect the multiplication of T. In the current work we also do *not* force programs operations $T^n \to T$ to be algebraic.

In order to further detail these aspects let us, first of all, recall the basic theory of monads.

3.1.1 *The basic theory of monads*

Definition 3.1.1. A monad is a triple $\mathbb{T} = (T, \eta, \mu)$ such that $T : \mathsf{C} \to \mathsf{C}$ is a functor, and $\eta : \mathrm{Id} \to T$, $\mu : TT \to T$ are natural transformations that make the diagrams below commute.

$$\begin{tikzcd}
T \arrow[r, "\eta_T"] \arrow[dr, "\mathrm{Id}"'] & TT \arrow[d, "\mu"] & T \arrow[l, "T\eta"'] \arrow[dl, "\mathrm{Id}"] \\
 & T &
\end{tikzcd}
\qquad
\begin{tikzcd}
TTT \arrow[r, "\mu_T"] \arrow[d, "T\mu"'] & TT \arrow[d, "\mu"] \\
TT \arrow[r, "\mu"'] & T
\end{tikzcd}$$

Examples 3.1.2. Consider a category C. In the following examples of a monad let X and Y be C-objects and $f : X \to Y$ be a C-morphism.

1. Take a set E. The exception monad $(\mathrm{M} : \mathsf{Set} \to \mathsf{Set}, \eta, \mu)$ is defined by,

$$\mathrm{M}X = X + E, \qquad \mathrm{M}f = f + \mathrm{id}$$

The unit η is defined at each X as the injection of X into $X + E$, and the multiplication μ is defined at each X as,

$$(\mathrm{id} + \triangledown) \cdot \mathrm{assocr} : (X + E) + E \to X + E$$

where $\mathrm{assocr} : (X + E) + E \to X + (E + E)$ is the associativity map and $\triangledown : E + E \to E$ is the codiagonal map. When E is the singleton set the exception monad is more commonly called maybe monad.

2. The list monad $(\mathrm{L} : \mathsf{Set} \to \mathsf{Set}, \eta, \mu)$ is defined by,

$$\mathrm{L}X = \coprod_{n \in \mathbb{N}} X^n, \qquad \mathrm{L}f = \coprod_{n \in \mathbb{N}} f^n : \mathrm{L}X \to \mathrm{L}Y$$

The unit η is defined at each X as $\eta_X(x) = [x]$, and the multiplication μ at each X is the concatenation of lists.

3. The powerset monad $(\mathrm{P} : \mathsf{Set} \to \mathsf{Set}, \eta, \mu)$ is defined by,

$$\mathrm{P}X = \{U \mid U \subseteq X\}, \qquad \mathrm{P}f = f[-]$$

The unit η is defined at each X by $\eta_X(x) = \{x\}$, and the multiplication μ at each X is given by $\mu_X(U) = \bigcup U$.

The non-empty powerset functor $\mathrm{Q} : \mathsf{Set} \to \mathsf{Set}$ together with the unit and multiplication of P also forms a monad. The compact Vietoris functor $\mathrm{V} : \mathsf{Top} \to \mathsf{Top}$ defined by,

$$\mathrm{V}X = (\{U \mid U \text{ compact on } X\}, \text{ hit-and-miss topology on } X), \qquad \mathrm{V}f = f[-]$$

and equipped with these two operations is also a monad [Mic51, Theorems 2.5.2 and 5.7.2][1]. The reader will find in Chapter 5 more details about the compact Vietoris functor and its topology.

4. The free rectangular bands monad $((- \times -) : \mathsf{Set} \to \mathsf{Set}, \eta, \mu)$. The unit η is defined at each X as the diagonal map, and the multiplication at each X as,

$$\mu_X(a, b, c, d) = (a, d)$$

5. A semiring $(S, +, \cdot, 1, 0)$ induces a weight functor $\mathrm{W}_S : \mathsf{Set} \to \mathsf{Set}$ defined by,

$$\mathrm{W}_S X = \left\{\phi \in S^X \mid \mathrm{supp}(\phi) \text{ finite}\right\}, \qquad \mathrm{W}_S f(\phi)(y) = \sum_{x \in f^{-1}(y) \cap \mathrm{supp}(\phi)} \phi(x)$$

This functor forms a monad whose unit is defined at each X by, $\eta_X(x)(y) = 1$ if $x = y$ and 0 otherwise; multiplication is defined at each X by,

$$\mu_X(\Phi)(x) = \sum_{\phi \in \mathrm{supp}(\Phi)} \Phi(\phi) \cdot \phi(x)$$

where $\cdot$ is the semiring multiplication.

Consider the unit $([0, 1], +, \cdot, 1, 0)$ semiring. The subfunctor $\mathrm{D} : \mathsf{Set} \to \mathsf{Set}$ of $\mathrm{W}_{[0,1]} : \mathsf{Set} \to \mathsf{Set}$ that only considers probability distributions equipped with the unit and multiplication of $\mathrm{W}_{[0,1]}$ is also a monad [Sok05, page 167]. The same applies to the subfunctor of $\mathrm{W}_{[0,1]} : \mathsf{Set} \to \mathsf{Set}$ that only considers subprobability distributions [HJS07]. The powerset functor P is the monad W_2 in disguise where 2 corresponds to the lattice $\{0 \leq 1\}$.

The following theorem shows that monads are closely related to adjunctions [ML98, page 138].

1 [Mic51] and most classical references concerning the compact Vietoris functor exclude the empty set, although in the coalgebraic community this is more unusual (see *e. g.* [KKV04; VV14; BBH12]). In the current work we do consider the empty set, simply because we found no technical reason for not doing so.

Theorem 3.1.3. *Every adjunction,*

$$\begin{tikzcd} \mathsf{A} \arrow[r, bend left, "F"] \arrow[r, phantom, "\perp"] & \mathsf{B} \arrow[l, bend left, "G"] \end{tikzcd}$$

induces a monad $(GF, \eta, G\epsilon_F)$ *where* $\eta : \mathrm{Id} \to GF$, $\epsilon : FG \to \mathrm{Id}$ *are, respectively, the unit and counit of the adjunction.*

A natural question to ask is whether the converse also holds. The answer is positive and extremely interesting: every monad induces a *Kleisli adjunction* and an *Eilenberg-Moore adjunction*. The former is closely related to Moggi's denotations, and the latter establishes the aforementioned connection with algebraic theories.

Definition 3.1.4. The Kleisli category $\mathsf{C}_{\mathbb{T}}$ of a monad $\mathbb{T}$ has as objects those of C and as hom-sets those defined by the equation,

$$\mathsf{C}_{\mathbb{T}}(X, Y) = \mathsf{C}(X, TY)$$

For each $\mathsf{C}_{\mathbb{T}}$-object X the identity is $\eta_X : X \to TX$, and the composition $g \bullet f$ of two morphisms $f : X \to TY$ and $g : Y \to TZ$ is $\mu_Z \cdot Tg \cdot f$.

There exists a functor $\mathsf{C} \to \mathsf{C}_{\mathbb{T}}$ that acts as the identity on objects and that post-composes C-morphisms with the monad's unit. There also exists a functor $\mathsf{C}_{\mathbb{T}} \to \mathsf{C}$ that acts like T on objects and that maps $\mathsf{C}_{\mathbb{T}}$-morphisms $f : X \to TY$ to C-morphisms $\mu_Y \cdot Tf : TX \to TY$. Both functors form the aforementioned Kleisli adjunction,

$$\begin{tikzcd} \mathsf{C} \arrow[r, bend left] \arrow[r, phantom, "\perp"] & \mathsf{C}_{\mathbb{T}} \arrow[l, bend left] \end{tikzcd}$$

Definition 3.1.5. The Eilenberg-Moore category $\mathsf{C}^{\mathbb{T}}$ of a monad $\mathbb{T}$ has as objects T-algebras (X, a) that make the diagrams below commute, and as morphisms T-algebra morphisms.

$$\begin{tikzcd} X \arrow[r, "\eta_X"] \arrow[dr, "\mathrm{id}"'] & TX \arrow[d, "a"] \\ & X \end{tikzcd} \qquad \begin{tikzcd} TTX \arrow[r, "Ta"] \arrow[d, "\mu_X"'] & TX \arrow[d, "a"] \\ TX \arrow[r, "a"'] & X \end{tikzcd}$$

The Eilenberg-Moore adjunction is the free-forgetful adjunction,

$$\begin{tikzcd} \mathsf{C} \arrow[r, bend left] \arrow[r, phantom, "\perp"] & \mathsf{C}^{\mathbb{T}} \arrow[l, bend left] \end{tikzcd}$$

whose free algebras are the pairs (TX, μ_X).

These two adjunctions are the two 'extreme solutions' for the problem of finding an adjunction that yields a given monad $\mathbb{T}$ [ML98, pages 142 and 148]. To be concrete,

Theorem 3.1.6. *Consider an adjunction,*

$$\begin{tikzcd}\mathsf{A} \arrow[r, bend left] \arrow[r, phantom, "\perp"] & \mathsf{B} \arrow[l, bend left]\end{tikzcd}$$

that yields a monad $\mathbb{T}$. There exist unique functors $\mathsf{A}_{\mathbb{T}} \to \mathsf{B}$ and $\mathsf{B} \to \mathsf{A}^{\mathbb{T}}$ that make the diagram below commute.

$$\begin{tikzcd}\mathsf{A}_{\mathbb{T}} \arrow[r] \arrow[dr, bend left] \arrow[dr, phantom, "\top"] & \mathsf{B} \arrow[r] \arrow[d, bend left] \arrow[d, phantom, "\dashv"] & \mathsf{A}^{\mathbb{T}} \arrow[dl, bend left] \\ & \mathsf{A} \arrow[ul, bend left] \arrow[u, bend left] \arrow[ur, bend left] \arrow[ur, phantom, "\perp"] & \end{tikzcd}$$

Eilenberg-Moore categories $\mathsf{A}^{\mathbb{T}}$ have many well-known properties. For example, their forgetful functor $\mathsf{A}^{\mathbb{T}} \to \mathsf{A}$ is always faithful and always creates limits. Therefore, if the 'comparison' functor $\mathsf{B} \to \mathsf{A}^{\mathbb{T}}$ is either an isomorphism or an equivalence, then it is generally a good idea to examine the category B by looking at $\mathsf{A}^{\mathbb{T}}$. This is in fact particularly relevant for varieties: the Eilenberg-Moore category of the monad induced by the free-forgetful adjunction of a variety is equivalent to the variety itself [ML98, page 156].

Examples 3.1.7. The following table provides some correspondences between monads $\mathbb{T}$ and categories that are equivalent to $\mathsf{C}^{\mathbb{T}}$.

Monad	**Categories**	**Reference**
Maybe (M)	Pointed sets and point-preserving maps	[HPP06]
List (L)	Monoids and monoid maps	[AHS09]
Multiset ($\mathrm{W}_{\mathbb{N}}$)	Commutative monoids and monoid maps	[Jac10]
Powerset ($\mathrm{P} \simeq \mathrm{W}_2$)	Complete lattices and join-preserving maps	[AHS09]
Rectangular bands ($(-)^2$)	Rectangular bands and semigroup maps	[MM07]
Distribution (D)	Convex algebras and convex algebra maps	[Jac10]

Definition 3.1.8. A morphism $\alpha : \mathbb{T} \to \mathbb{S}$ between two monads $\mathbb{T} = (T, \eta, \mu)$ and $\mathbb{S} = (S, \kappa, \nu)$ is a natural transformation $\alpha : T \to S$ that makes the diagrams below commute.

$$\begin{tikzcd}\mathrm{Id} \arrow[r, "\eta"] \arrow[dr, "\kappa"'] & T \arrow[d, "\alpha"] \\ & S\end{tikzcd} \qquad \begin{tikzcd}TT \arrow[rr, "\mu"] \arrow[d, "T\alpha"'] & & T \arrow[d, "\alpha"] \\ TS \arrow[r, "\alpha_S"'] & SS \arrow[r, "\nu"'] & S\end{tikzcd}$$

We say that $\mathbb{T}$ is a *submonad* of $\mathbb{S}$ if there exists a monad monomorphism $\alpha : T \rightarrowtail S$ that witnesses T as a subfunctor of S. In this case, there exists an inclusion functor,

$$\mathsf{C}_{\mathbb{T}} \to \mathsf{C}_{\mathbb{S}}$$

that acts as the identity on objects and that post-composes a $\mathsf{C}_{\mathbb{T}}$-morphism $X \to TY$ with the map $\alpha_Y : TY \rightarrowtail SY$. Intuitively, this tells that $\mathbb{T}$ captures a fragment of the program semantics induced by $\mathbb{S}$.

The following notion provides a natural way of combining monads.

Definition 3.1.9. Consider two monads $\mathbb{T} = (T, \eta, \mu)$ and $\mathbb{S} = (S, \kappa, \nu)$. A *distributive law* of $\mathbb{S}$ over $\mathbb{T}$ is a natural transformation $\delta : ST \to TS$ that makes the following diagrams commute.

$$\begin{tikzcd}
& T \arrow[dl, "\kappa_T"'] \arrow[dr, "T\kappa"] & \\
ST \arrow[rr, "\delta"'] & & TS \\
& S \arrow[ul, "S\eta"] \arrow[ur, "\eta_S"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
STT \arrow[r, "\delta_T"] \arrow[d, "S\mu"'] & TST \arrow[r, "T\delta"] & TTS \arrow[d, "\mu_S"] \\
ST \arrow[rr, "\delta"'] & & TS \\
SST \arrow[u, "\nu_T"] \arrow[r, "S\delta"'] & STS \arrow[r, "\delta_S"'] & TSS \arrow[u, "T\nu"']
\end{tikzcd}$$

Theorem 3.1.10. *A distributive law $\delta : ST \to TS$ for two monads $\mathbb{T} = (T, \eta, \mu)$ and $\mathbb{S} = (S, \kappa, \nu)$, induces another monad $(ST, \kappa_T \cdot \eta, \nu_\kappa \cdot SS\mu \cdot S\lambda_T)$.*

The maybe monad M has an interesting property about distributive laws [LG02, Section 2.3].

Theorem 3.1.11. *Consider a monad $\mathbb{T} = (T, \eta, \mu)$ on* Set*. There exists a distributive law $\delta : \mathrm{M}T \to T\mathrm{M}$ defined at each set X by,*

$$\delta_X = [Ti_1, \eta_{\mathrm{M}X} \cdot i_2]$$

Hence, for a monad $\mathbb{T}$ on Set there is always a monad structure on $T\mathrm{M}$. In the following section, we show that this allows to extend a programming paradigm (given by $\mathbb{T}$) with notions of failure. In the hybrid case, it gives rise to partial hybrid computations HM which includes abort operations, tests, and exception-handling constructs.

3.1.2 *Kleisli representations*

We saw in Section 2.2 that both differential dynamic logic [Pla10] and the weak Kleene algebra of hybrid systems [HM09] assume a single-sorted setting. In the monadic perspective, this leads to the interpretation of a program p as an endomorphism $[\![\mathsf{p}]\!] : X \to TX$ for a monad $\mathbb{T}$ – as exemplified in [HKL14; HK15], where the goal is the systematic generation of dynamic logics from a monad $\mathbb{T}$ with actions interpreted as morphisms $X \to TX$.

Our illustrations and results on hybrid programming also adopt the single-sorted setting: not only this clarifies their connection with the works on hybrid systems [Pla10; HM09] and dynamic logic [HKL14; HK15] that were previously mentioned, but it also helps us to focus on the essential ingredients of hybrid programming and provide rich examples whilst keeping simplicity. In order for our investigation to proceed smoothly, we will also need a framework for the systematic development and analysis of program semantics. This is what the current subsection aims to achieve.

Program denotations *à la* Moggi assume the existence of an interpretation map $\Pi \to \mathrm{End}_{\mathbb{T}}(X)$ from programs Π to endomorphisms over X in the Kleisli category of $\mathbb{T}$. Going a bit further, we require that the interpretation map is also a monoid homomorphism,

$$(\Pi,\ ;\ , \mathsf{skip}) \to (\mathrm{End}_{\mathbb{T}}(X),\ \bullet\ , \eta_X)$$

where ; denotes sequential composition, and skip is the 'do nothing' program. A monoid homomorphism with the monoid $(\mathrm{End}_{\mathbb{T}}(X), \bullet, \eta_X)$ as codomain is here called a *Kleisli* $\mathbb{T}$*-representation* or simply Kleisli representation if it is clear which monad is involved.

Remark 3.1.12. The name 'Kleisli representation' is inspired by Representation theory, a vast field of research where the elements of certain algebraic structures, such as groups and Lie algebras, are *represented* by endomorphisms of vector spaces [Ste11; FH13], with applications ranging from the classification of finite groups to quantum field theory.

Kleisli representations unify the syntactic and semantic parts of a programming language into a single, well-known concept – that of monoid homomorphism. Among other things, this feature allows us to build programming languages systematically. The following case is the simplest example of this aspect, since we always assume that a programming language is (at least) a monoid.

Example 3.1.13. Let $\mathrm{F} : \mathsf{Set} \to \mathsf{Mon}$ be the free monoid functor, let At be a set of atomic programs, and consider an interpretation map $f : \mathsf{At} \to \mathrm{End}_{\mathbb{T}}(X)$ for a monad $\mathbb{T}$. The free extension of f is a Kleisli representation,

$$\mathrm{F}(\mathsf{At}) \to (\mathrm{End}_{\mathbb{T}}(X),\ \bullet\ , \eta_X)$$

For example,

1. An interpretation map $f : 1 \to \mathrm{End}_{\mathbb{T}}(X)$ generates a basic 'iteration' programming language, given by the free extension of f,

   $$(\mathbb{N}, +, 0) \simeq (1^*,\ ;\ , \mathsf{skip}) \to \mathrm{End}_{\mathbb{T}}(X)$$

   The program '10', for instance, reads "execute $f(*)$ ten times".

2. Consider a deterministic automaton $t : A \times X \to X$, and an interpretation map $f : A \to \mathrm{End}_{\mathrm{Id}}(X)$ defined by $a \mapsto t(a, -) : X \to X$. The free extension of f,

   $$(A^*,\ ;\ , \mathsf{skip}) \to (\mathrm{End}_{\mathrm{Id}}(X),\ \bullet\ , \eta_X)$$

   is a very simple programming language for running the automaton. The program $\mathsf{a}_1; \mathsf{a}_2$, for instance, reads "trigger event a_1 and then event a_2". Programming languages for non-deterministic $A \times X \to \mathrm{P}X$ and probabilistic automata $A \times X \to \mathrm{D}X$ are obtained analogously by replacing the identity monad with the powerset P and the distribution monad D, respectively.

In order to generate richer programming languages, we need to incorporate program operations in the notion of a Kleisli representation. Consider an object of a finitary quasi-variety[2] V defined by a signature $\Sigma = \Phi \cup \{\mathsf{skip}, ;\}$ with arity map $\mathrm{ar} : \Sigma \to \mathbb{N}$ and a set of quasi-equations E that contains the monoid laws with respect to $\{\mathsf{skip}, ;\}$ – we will regard objects of this type as programming languages, with Σ as the set of program operations that they come equipped with. Consider also a monad $\mathbb{T}$ on a category C with products. We then proceed in footsteps of [HKL14]: for each n-ary program operation $\sigma \in \Phi$, we choose a natural transformation[3],

$$\alpha^{\sigma} : T^{(\mathrm{ar}(\sigma))} \to T$$

where $T^0 = 1$, and define $[\![\sigma]\!] : \mathrm{End}_{\mathbb{T}}(X)^{(\mathrm{ar}(\sigma))} \to \mathrm{End}_{\mathbb{T}}(X)$ by,

$$[\![\sigma]\!](a_1, \ldots, a_{\mathrm{ar}(\sigma)}) = \alpha^{\sigma}_X \cdot \langle a_1, \ldots, a_{\mathrm{ar}(\sigma)} \rangle$$

Each choice α^{σ} of natural transformation $T^{\mathrm{ar}(\sigma)} \to T$ will be used to fix the interpretation of σ in the programming language. Using these ingredients we can now state the general definition of Kleisli $\mathbb{T}$-representation.

Definition 3.1.14. Let $\mathbb{T}$ be a monad on a category C with products and A an object of a quasi-variety V as defined above. A Kleisli $\mathbb{T}$-representation of A in $\mathsf{C}_{\mathbb{T}}$ is an assignment of every $\sigma \in \Phi$ to a natural transformation $\alpha^{\sigma} : T^{\mathrm{ar}(\sigma)} \to T$ together with a V-algebra morphism,

$$(A,\ ;\ , \mathsf{skip}, \Phi) \to (\mathrm{End}_{\mathbb{T}}(X),\ \bullet\ , \eta_X, ([\![\sigma]\!])_{\sigma \in \Phi})$$

In the formalism of Kleisli representations several interesting questions can be asked in rigorous terms. For example,

(i) *For a monad $\mathbb{T}$ which programming languages can be given a Kleisli $\mathbb{T}$-representation, or in other words which kinds of program and program operation can $\mathbb{T}$-computations support ?*

(ii) *Conversely, given a programming language Π what are the good choices of monads $\mathbb{T}$ to define Kleisli $\mathbb{T}$-representations of Π ?*

(iii) *More generally, given a programming language Π and a monad $\mathbb{T}$ what are the Kleisli representations $\Pi \to \mathrm{End}_{\mathbb{T}}(X)$?*

(iv) *Given a monad $\mathbb{T}$ that is able to interpret only a fragment of a language, how should we suitable extend it so that it is able to interpret the whole language?*

To concretely answer these questions we need to determine which additional algebraic structures can be placed on monoids of the form $\mathrm{End}_{\mathbb{T}}(X)$ for a given monad $\mathbb{T}$. Frequently, $\mathrm{End}_{\mathbb{T}}(X)$ inherits the algebraic structure of $\mathbb{T}$, for example $\mathrm{End}_{\mathrm{P}}(X)$, for P the powerset monad, is always a

2 Given a signature Σ, a finitary quasi-variety is a class V of Σ-algebras that satisfy a set of implications (quasi-equations) of the form $(\tau_1 = \tau_1') \wedge \cdots \wedge (\tau_n = \tau_n') \Rightarrow (\tau = \tau')$. See [AR94, Section 3.B] for more details.

3 Recall that we do *not* force programs operations $T^n \to T$ to be algebraic [PP01b; PP01a; PP03].

complete lattice, and $\mathrm{End}_{\mathrm{D}}(X)$, for D the distribution monad, is always a convex algebra. In this case, the questions above can be informally rephrased as: *which algebraic structures are hidden within the algebraic structure of* $\mathbb{T}$ *?* Ultimately, we want to know which natural transformations $T^n \to T$ a given monad $\mathbb{T}$ supports, as they are the basis of all program operations (with the exception of Kleisli composition). Let us illustrate these remarks with some examples.

Example 3.1.15. We saw how free monoids of atomic programs are given a Kleisli $\mathbb{T}$-representation for any monad $\mathbb{T}$. This set-up can now be extended in several ways.

1. Consider the signature $\{\mathsf{skip},\ ;\ ,\mathsf{0}\}$ and recall that a pointed set is simply a set X together with a map $1 \to X$. Recall also that the Eilenberg-Moore category $\mathsf{Set}^{\mathrm{M}}$ of the maybe monad is equivalent to the category of pointed sets and point-preserving maps. This equivalence renders evident the existence of a natural transformation $1 \to \mathrm{M}$ that constantly outputs 'failure' (it is given by the free pointed set construction). We can use this natural transformation to interpret $\mathsf{0}$ as an abort operation since it is not hard to show that $1 \to \mathrm{M}$ and Kleisli composition endow $\mathrm{End}_{\mathrm{M}}(X)$ with the structure of a monoid equipped with an absorbing element[4]. Given an interpretation map $\mathsf{At} \to \mathrm{End}_{\mathrm{M}}(X)$ we freely extend it,
$$(\Pi,\ ;\ ,\mathsf{skip},\mathsf{0}) \to (\mathrm{End}_{\mathrm{M}}(X),\ \bullet\ ,\eta_X,[\![\mathsf{0}]\!])$$
into the category of monoids with an absorbing element. In the language, $\mathsf{0}$ is the program that 'always fails'; its composition with any program always yields $\mathsf{0}$.

2. Consider now idempotent semirings, with signature $\{\mathsf{skip},\ ;\ ,\mathsf{0},+\}$. For the powerset monad P, the monoid $\mathrm{End}_{\mathrm{P}}(X)$ can be extended to an idempotent semiring by taking $\alpha^0 : 1 \to \mathrm{P}$ to be the natural transformation that outputs $\emptyset$ at every set, and $\alpha^+ : \mathrm{P}^2 \to \mathrm{P}$ to be the natural transformation that given two sets outputs their union (these transformations are given by the free complete lattices). Now given a set of atomic programs At, every map $\mathsf{At} \to \mathrm{End}_{\mathrm{P}}(X)$ can be freely extended to an idempotent semiring morphism from the corresponding freely generated language to $\mathrm{End}_{\mathrm{P}}(X)$. The reader might recall that the actions of propositional dynamic logic are seen as non-deterministic programs [HKT00]. If the Kleene star is not considered, then their semantics is given precisely by the free idempotent semiring extension of $\mathsf{At} \to \mathrm{End}_{\mathrm{P}}(X)$.

3. We already know how to generate a basic programming language $(\Pi,\ ;\ ,\mathsf{skip})$ for an automaton $A \times X \to X$ (see Example 3.1.13). So suppose now that we wish to enrich the language with a probabilistic choice operation $+_\lambda$, $\lambda \in [0,1]$. This new ingredient suggests that we replace the interpretation domain $\mathrm{End}_{\mathrm{Id}}(X)$ by $\mathrm{End}_{\mathrm{D}}(X)$, which can be easily achieved with the composition of monoid morphisms,
$$(\Pi,\ ;\ ,\ \mathsf{skip}) \to (\mathrm{End}_{\mathrm{Id}}(X),\ \bullet\ ,\eta_X) \rightarrowtail (\mathrm{End}_{\mathrm{D}}(X),\ \bullet\ ,\eta_X)$$

4 A monoid $(M,\cdot,1)$ with an absorbing element is a monoid with an element $0 \in M$ such that $0\cdot m = 0 = m \cdot 0$ for every $m \in M$.

where the one on the right is given by the inclusion $\mathsf{Set} \to \mathsf{Set}_{\mathrm{D}}$ (see Definition 3.1.8 and note that the identity monad is a submonad of the distribution D monad). The task now boils down to finding suitable natural transformations $\mathrm{D}^2 \to \mathrm{D}$ to interpret the probabilistic choice operation.

A set $\mathrm{D}X$ is always a convex algebra [AH17]. In particular, for each $\lambda \in [0,1]$ there exists a natural transformation $\alpha_\lambda : \mathrm{D}^2 \to \mathrm{D}$ defined at each X by,

$$(\alpha_\lambda)_X(\mu_1, \mu_2) = \lambda\mu_1 + (1-\lambda)\mu_2$$

Let us interpret probabilistic choice using these natural transformations and endow $\mathrm{End}_{\mathrm{D}}(X)$ with this algebraic structure in the usual way. Consider the variety V generated by the signature $\{\mathsf{skip},\ ;\ ,(+_\lambda)_{\lambda\in[0,1]}\}$ and the monoid laws for $\{\mathsf{skip},\ ;\}$. The tuple,

$$(\mathrm{End}_{\mathrm{D}}(X),\ \bullet\ , \eta_X, (\llbracket +_\lambda \rrbracket)_{\lambda\in[0,1]})$$

is an element of V. Given an interpretation map $A \to \mathrm{End}_{\mathrm{Id}}(X)$ one obtains $A \to \mathrm{End}_{\mathrm{Id}}(X) \rightarrowtail \mathrm{End}_{\mathrm{D}}(X)$. The last step is to freely extend this composition into a V-homomorphism,

$$(\Pi,\ ;\ , \mathsf{skip}, (+_\lambda)_{\lambda\in[0,1]}) \to (\mathrm{End}_{\mathrm{D}}(X),\ \bullet\ , \eta_X, (\llbracket +_\lambda \rrbracket)_{\lambda\in[0,1]})$$

and the desired programming language is obtained. The program $(\mathsf{a}_1 +_{0.3} \mathsf{a}_2)\ ;\ \mathsf{a}_3$, for example, reads "trigger with probability 0.3 the event a_1 and with probability 0.7 the event a_2. After this, trigger the event a_3".

Remark 3.1.16. The programming language in the last example can be refined with additional laws: *e. g.* one can safely assume idempotency $\mathsf{p} +_\lambda \mathsf{p}$ and a weak form of commutativity $\mathsf{p} +_\lambda \mathsf{q} = \mathsf{q} +_{1-\lambda} \mathsf{p}$.

The examples above tell us that programming paradigms can be thoroughly examined by characterising natural transformations $T^n \to T$ and their interaction with Kleisli composition. This is the strategy that we will employ to study the hybrid paradigm, and in this mission Yoneda lemma [ML98, page 61] will be an essential tool.

Lemma 3.1.17 (Yoneda lemma)**.** *Let C be a locally small category. For every functor $F : \mathsf{C} \to \mathsf{Set}$ and C-object X there exists a bijection,*

$$[\mathsf{C}, \mathsf{Set}]\,(\mathsf{C}(X, -), F) \simeq FX$$

that is natural both in X and F.

In order to exemplify the high relevance of Yoneda lemma in the context of Kleisli representations, consider again the signature $\{\mathsf{skip},\ ;\ , \mathsf{0}\}$. We already know that the maybe monad M supports programming languages with signature $\{\mathsf{skip},\ ;\ , \mathsf{0}\}$ (see Example 3.1.15). So we may ask ourselves *"are there any others monads that support programming languages of this type ?"* Or equivalently, *"are there any other monads $\mathbb{T}$ that possess a natural transformation $1 \to T$?".* Since the constant functor $1 : \mathsf{Set} \to \mathsf{Set}$ is representable as $\mathsf{C}(\emptyset, -) : \mathsf{Set} \to \mathsf{Set}$, Yoneda lemma reduces the answers to these questions to a triviality: we have the following chain of bijections,

$$[\mathsf{Set}, \mathsf{Set}](1, T) \simeq [\mathsf{Set}, \mathsf{Set}](\mathsf{C}(\emptyset, -), T) \simeq T\emptyset$$

and hence a monad $\mathbb{T}$ supports a programming language with signature $\{\mathsf{skip},\ ;\ , \mathsf{0}\}$ iff its underlying functor does not preserve the empty set. We can actually go a bit further and state that,

Corollary 3.1.18. *The maybe monad, the list monad, and the powerset monad have* exactly one *possible interpretation for the construct* $\mathsf{0}$*. The rectangular bands monad and the distribution monad have* no possible *interpretation for the construct* $\mathsf{0}$*, and therefore they cannot support an abort operation.*

We conclude this subsection by considering *quotient representations* which further justify the naturality condition imposed on program operations.

Often one will need to abstract away details of a representation $\rho : A \to \mathrm{End}_{\mathbb{T}}(X)$ in a large, fine-grained space X by building a representation $\rho' : A \to \mathrm{End}_{\mathbb{T}}(Q)$ in a coarser one, but in such a way that both representations 'agree' with each other. Formally, for an epimorphism $q : X \twoheadrightarrow Q$, we need that the equation,

$$Tq \cdot \rho(a) = \rho'(a) \cdot q \tag{1}$$

holds for all programs $a \in A$. This property tells that abstracting and then interpreting is the same as interpreting and then abstracting. The naturality of program operations promotes a *compositional* construction of quotient representations, because it allows to prove that Equation (1) holds just by showing that it holds for atomic programs. For sequential composition, this follows from the naturality of μ,

$$\begin{tikzcd}
X \arrow[r, "\rho(a)"] \arrow[d, "q"'] & TX \arrow[r, "T\rho(b)"] \arrow[d, "Tq"] & TTX \arrow[r, "\mu_X"] \arrow[d, "TTq"] & TX \arrow[d, "Tq"] \\
Q \arrow[r, "\rho'(a)"'] & TQ \arrow[r, "T\rho'(b)"'] & TTQ \arrow[r, "\mu_Q"'] & TQ
\end{tikzcd}$$

and for other operations the naturality requirement on $\alpha : T \times T \to T$ makes the following diagram also commute.

$$\begin{tikzcd}
X \arrow[r, "\langle\rho(a),\rho(b)\rangle"] \arrow[d, "q"'] & TX \times TX \arrow[r, "\alpha_X"] \arrow[d, "Tq\times Tq"] & TX \arrow[d, "Tq"] \\
Q \arrow[r, "\langle\rho'(a),\rho'(b)\rangle"'] & TQ \times TQ \arrow[r, "\alpha_Q"'] & TQ
\end{tikzcd}$$

So naturality allows to freely extend an abstraction from atomic programs to all programs in the language.

3.2 THE HYBRID MONAD

In Chapter 1 and 2 we saw that physical processes, like time, velocity, and radioactive decay, are traditionally specified via systems of differential equations [KH96; BS02; Per13]. For example, the water level of a tank can be described by the equation,

$$\dot{l} = k \tag{2}$$

where the constant k is the subtraction of the rate of water flowing out from the rate of water flowing in. Another example is the position and velocity of a vehicle, which in a one-dimensional perspective can be specified by,

$$\dot{p} = v, \dot{v} = a \tag{3}$$

where a denotes the vehicle's acceleration.

If a system of ordinary differential equations specifies a physical process then its solution is the physical process itself [KH96; BS02; Per13]. A solution of a system of ordinary differential equations is a dynamical system for the additive monoid of non-negative reals [Per13, Section 3.1]; more concretely, a map $\phi : \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ that makes the following diagrams commute .

$$\begin{tikzcd}
\mathbb{R}^n \arrow[r, "\langle \mathrm{id}, 0 \rangle"] \arrow[dr, "\mathrm{id}"'] & \mathbb{R}^n \times \mathbb{R}_{\geq 0} \arrow[d, "\phi"] \\
& \mathbb{R}^n
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathbb{R}^n \times \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \arrow[r, "\mathrm{id} \times (+)"] \arrow[d, "\phi \times \mathrm{id}"'] & \mathbb{R}^n \times \mathbb{R}_{\geq 0} \arrow[d, "\phi"] \\
\mathbb{R}^n \times \mathbb{R}_{\geq 0} \arrow[r, "\phi"'] & \mathbb{R}^n
\end{tikzcd}$$

Intuitively, a physical process ϕ controls *ad infinitum* the evolution (in some contexts, called trajectory) $\phi(v, -) : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ of a given value $v \in \mathbb{R}^n$. In the presence of a digital device, however, physical processes can only act for specific time intervals – a consequence of a computer's ability to dictate which physical process is active at a certain time (recall Chapter 2, where switched systems and their visual interpretation were introduced). For example, in the water tank, Equation (2) must frequently be updated in order to reflect changes in the valves that let water flow in/out, the latter being typically controlled by a computer. In regard to the moving vehicle, Equation (3) must also change but this time to reflect different choices of acceleration made by the cruise controller.

In the current section we will see that the hybrid monad captures this basic interaction between physical processes and digital devices.

Definition 3.2.1. Define $\mathrm{H} : \mathsf{Set} \to \mathsf{Set}$ as the coproduct of hom functors,

$$\coprod_{d \in \mathbb{R}_{\geq 0}} \hom([0, d], -)$$

Definition 3.2.2. Let the unit $\eta : \mathrm{Id} \to \mathrm{H}$ be defined at each set X by $\eta_X(x) = (\underline{x}, 0)$ where $\underline{x} : [0,0] \to X$ is the map constant on x. It is natural because it can be written as the composition,

$$\mathrm{Id} \simeq \hom(1, -) \xrightarrow{!_*} \hom([0,0], -) \xrightarrow{i_0} \mathrm{H}$$

where the natural transformation $!_* : \hom(1, -) \to \hom([0,0], -)$ is the one induced via Yoneda embedding by the universal map $! : [0,0] \to 1$.

The unit has two obvious inverses $\theta : \mathrm{H} \to \mathrm{Id}$ and $\lambda : \mathrm{H} \to \mathrm{Id}$: the first one sends an evolution (f,d) to the image $f(0)$ and the second one to $f(d)$.

The multiplication requires the following operation for concatenating evolutions.

Definition 3.2.3. Define $(+\!\!+) : \mathrm{H} \times \mathrm{H} \to \mathrm{H}$ as the natural transformation such that for every set X the equation,

$$(f,d) +\!\!+_X (g,e) = (h, d+e)$$

holds with $h(x) = f(x)$ if $x \leq d$ and $h(x) = g(x-d)$ otherwise. The transformation is natural because it can be written as the following composition,

$$\begin{aligned}
\mathrm{H} \times \mathrm{H} &\simeq \coprod_{d \in \mathbb{R}_{\geq 0}} \hom([0,d], -) \times \coprod_{e \in \mathbb{R}_{\geq 0}} \hom([0,e], -) \\
&\simeq \coprod_{d,e \in \mathbb{R}_{\geq 0}} \hom([0,d], -) \times \hom([0,e], -) \\
&\simeq \coprod_{d,e \in \mathbb{R}_{\geq 0}} \hom([0,d] + [0,e], -) \\
&\xrightarrow{\coprod f_*} \coprod_{d,e \in \mathbb{R}_{\geq 0}} \hom([0,d+e], -) \\
&\xrightarrow{[i_{d+e}]} \mathrm{H}
\end{aligned}$$

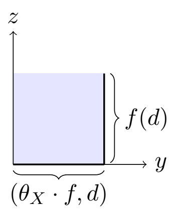
where $f : [0, d+e] \to [0,d] + [0,e]$ is defined as $f(x) = i_1(x)$ if $x \leq d$ and $f(x) = i_2(x-d)$ otherwise.

Definition 3.2.4. Define the multiplication $\mu : \mathrm{HH} \to \mathrm{H}$ at each set X by,

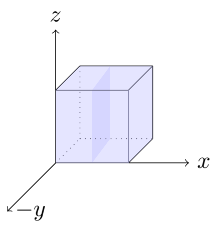
$$\mu_X(f,d) = (\theta_X \cdot f, d) +\!\!+_X (f(d))$$

It is natural because it can be written as the composition $\mathrm{HH} \xrightarrow{\langle \mathrm{H}\theta, \lambda_{\mathrm{H}} \rangle} \mathrm{H} \times \mathrm{H} \xrightarrow{(+\!\!+)} \mathrm{H}$.

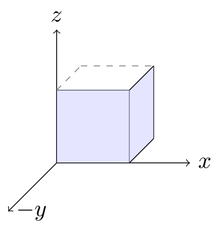
Remark 3.2.5. The multiplication has an intuitive visual interpretation: an element of $(f,d) \in \mathrm{HH}X$ can be seen as a 'square' whose columns are elements of $\mathrm{H}X$. Under this view, the multiplication sends (f,d) to the concatenation of the square's bottom line $(\theta_X \cdot f, d)$ with column $f(d)$, as illustrated below.

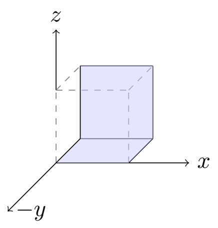


This perspective also offers a visual sketch of the proof that $\mu \cdot \mathrm{H}\mu = \mu \cdot \mu_{\mathrm{H}}$. First, an element of $\mathrm{HHH}X$ can be seen as the cube depicted below, where a projection on the x-axis yields an element of $\mathrm{HH}X$.

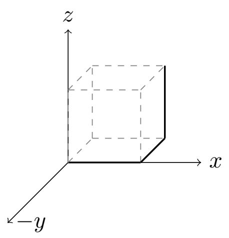


Through multiplication, the cube is flattened into a square in two different ways: via μ_{H} : $\mathrm{HHH}X \to \mathrm{HH}X$ or $\mathrm{H}\mu : \mathrm{HHH}X \to \mathrm{HH}X$. In the former case, only the front and right surfaces are kept (depicted below on the left). In the latter case, the function $\mathrm{H}\mu$ applies μ to each projection on the x-axis, and thus only the bottom and back surfaces are kept (depicted below on the right).





Finally, to apply $\mu : \mathrm{HH}X \to \mathrm{H}X$ to the resulting squares yields the same result,



Theorem 3.2.6. *The triple* $\mathbb{H} = (\mathrm{H}, \eta, \mu)$ *is a monad.*

Proof. In Appendix A. □

Let us explore the Kleisli composition of the hybrid monad and compare it against the remarks made at the section's beginning. Consider two maps $f : X \to \mathrm{H}Y$, $g : Y \to \mathrm{H}Z$ and for every element $x \in X$ denote $f(x)$ by $(f(x, -), d)$. Then calculate,

$$\begin{aligned} g \bullet f(x) &= \mu \cdot \mathrm{H}g \cdot f(x) \\ &= \mu(g \cdot f(x, -), d) \\ &= (\theta \cdot g \cdot f(x, -), d) +\!\!+ (g\,(f(x, d))) \end{aligned}$$

If the equation $\theta_Z \cdot g = \mathrm{id}$ holds then the last expression is equivalent to,

$$(f(x, -), d) +\!\!+ (g\,(f(x, d)))$$

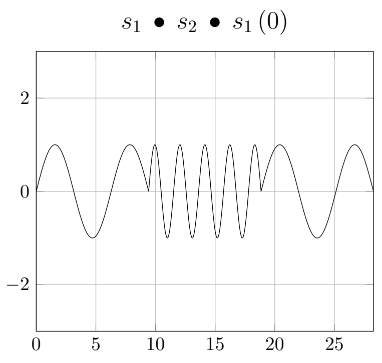
which is simply the concatenation of $(f(x, -), d)$ with $g\,(f(x, d))$. Otherwise, up to completion of $[0, d]$, g alters the evolution given by f and then proceeds with its own evolution. These notions are illustrated in the following example, where we will see that Kleisli composition can be used to switch between physical processes; as mentioned in Chapter 2, this was the idea that gave rise to the whole concept of hybrid system [Wit66].

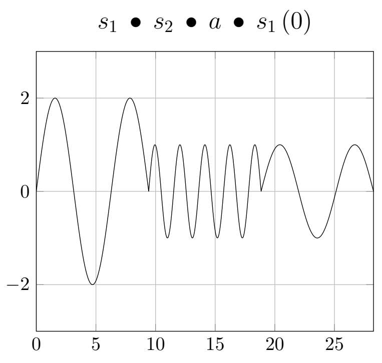
Example 3.2.7. Consider two signal generators $s_1, s_2 : \mathbb{R} \to \mathrm{H}\mathbb{R}$ defined by,

$$s_1(x) = (x + \sin(-), 3\pi), \qquad s_2(x) = (x + \sin(3 \times -), 3\pi)$$

The evolution $s_1 \bullet s_2 \bullet s_1\,(0)$ is depicted in the plot below on the left. This type of signal is commonly seen in frequency modulation, where the varying frequency is used to encode information for electromagnetic transmission. In the hybrid systems perspective, the map $s_1 \bullet s_2 \bullet s_1\,(0)$ can be regarded as the result of a computer giving control of 0's evolution to s_1, after some time to s_2, and then to s_1 again.

In order to amplify signals, one can use the map $a : \mathbb{R} \to \mathrm{H}\mathbb{R}$ defined by $a(x) = (\underline{x \times 2}, 0)$. Note that it does not respect $\theta_{\mathbb{R}} \cdot a = \mathrm{id}$ and therefore it can alter the evolutions of other maps. Given the input 0, the system $s_1 \bullet s_2 \bullet a \bullet s_1$ returns the evolution below on the right.





Definition 3.2.8. Consider a map $f : X \to \mathrm{H}X$. We call it a *unit-action* if the equation $\theta_X \cdot f = \mathrm{id}$ holds.

3.2.1 *Hybrid programming languages*

Using both the hybrid monad and the framework of Kleisli representations, we can start to develop hybrid programming languages systematically. Let us analyse a very simple case: take a finite set of real-valued variables $X = \{x_1, \ldots, x_n\}$ and denote by $\mathsf{At}(X)$ the set given by the grammar,

$$\varphi \ni (x_1 := t, \ldots, x_n := t) \mid (\dot{x}_1 = t, \ldots, \dot{x}_n = t \,\&\, d), \qquad\qquad t \ni r \mid r \cdot x \mid t + t$$

where d and r are real numbers and $x \in X$.

Denote the usual interpretation of a term t over a valuation $(v_1, \ldots, v_n) \in \mathbb{R}^n$ by $[\![t]\!]_{(v_1,\ldots,v_n)} \in \mathbb{R}$, or simply $[\![t]\!]$ if the valuation is clear from the context. Since linear systems of ordinary differential equations have always unique solutions [Per13, page 17] there exists an interpretation map $\mathsf{At}(X) \to \mathrm{End}_{\mathrm{H}}(\mathbb{R}^n)$ that sends the expression $(x_1 := t_1, \ldots, x_n := t_n)$ to the function $\mathbb{R}^n \to \mathrm{H}(\mathbb{R}^n)$ defined by,

$$(v_1, \ldots, v_n) \mapsto \eta\,([\![t_1]\!], \ldots, [\![t_n]\!])$$

and that sends the expression $(\dot{x}_1 = t_1, \ldots, \dot{x}_n = t_n \,\&\, d)$ to the respective solution $\mathbb{R}^n \to (\mathbb{R}^n)^{\mathbb{R}_{\geq 0}}$ but restricted to $\mathbb{R}^n \to (\mathbb{R}^n)^{[0,d]}$.

Remark 3.2.9. Solutions of systems of ordinary differential equations are dynamical systems for the additive monoid $(\mathbb{R}_{\geq 0}, +, 0)$. Therefore, the interpretation $\phi : \mathbb{R}^n \to (\mathbb{R}^n)^{[0,d]}$ of an expression $(\dot{x}_1 = t, \ldots, \dot{x}_n = t \,\&\, d)$ is always a unit-action.

The free monoidal extension of the interpretation map induces a programming language,

$$\mathsf{p} = \mathsf{a} \in \mathsf{At}(X) \mid \mathsf{skip} \mid \mathsf{p} \,;\, \mathsf{p}$$

which includes assignments and differential equations.

Examples 3.2.10. Let us explore the language with a series of examples.

1. Recall that the water level of a tank can be described by the equation $\dot{l} = k$, and suppose that a valve connected to it opens at precisely ten seconds. The program,

   $$(\dot{\mathtt{l}} = \mathtt{k} \,\&\, 10) \,;\, (\dot{\mathtt{l}} = \mathtt{k} + \mathtt{m} \,\&\, \mathtt{n})$$

   describes this behaviour: it makes the water level change at the rate of $\mathtt{k}$ cm/s for ten seconds and then at $\mathtt{k} + \mathtt{m}$ cm/s for $\mathtt{n}$ seconds.

2. Consider a heating program $(\dot{\mathtt{t}} = 2 \,\&\, 10)$ that makes the temperature go up at the rate of $2°\mathrm{C}/s$, and a (simplistic) program $(\dot{\mathtt{t}} = 0 \,\&\, 10)$ that maintains a given temperature for ten seconds. The composition,

   $$(\dot{\mathtt{l}} = 2 \,\&\, 10) \,;\, (\dot{\mathtt{l}} = 0 \,\&\, 10)$$

   makes the current temperature rise for ten seconds, and then keeps it stable for another ten seconds.

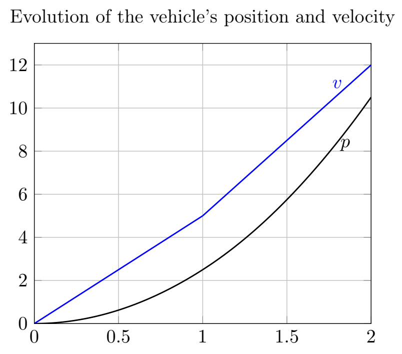
3. Recall that the movement of a vehicle can be described by the system of equations,

$$\dot{p} = v, \dot{v} = a$$

where a denotes an acceleration chosen by the controller. The program,

$$\mathtt{a} := 5 \,;\, (\dot{\mathtt{p}} = \mathtt{v}, \dot{\mathtt{v}} = \mathtt{a} \,\&\, 1) \,;\, \mathtt{a} := \mathtt{a} + 2 \,;\, (\dot{\mathtt{p}} = \mathtt{v}, \dot{\mathtt{v}} = \mathtt{a} \,\&\, 1)$$

sets the acceleration to $5m/s^2$, lets position and velocity evolve for one second, increases the acceleration, and finally lets position and velocity evolve again for one second. Assuming that position and velocity are equal to zero, this program yields the following plot with respect to the vehicle's position and velocity.



The language illustrated by the previous examples is *time-triggered*: *i. e.* a program ($\dot{x}_1 = t_1, \ldots, \dot{x}_n = t_n \,\&\, d$) terminates precisely when the instant of time d is achieved. It is also possible to consider *event-triggered* languages by forcing a program to terminate *as soon as* a certain event occurs. To do this, however, we need to be very strict on the kinds of event allowed: consider for example the expression ($\dot{x} = 1 \,\&\, x > 7$). We want to interpret it as a program that terminates as soon as the condition $x > 7$ is satisfied, *but given an initial value for x, can we find the earliest time at which the condition is satisfied ?*

Remark 3.2.11. Already in [Wit66], Witsenhausen considered the possibility of hybrid systems being event-triggered by forcing a physical process to terminate as soon as it intersects a given *closed* set $C \subseteq \mathbb{R}^n$. As explained below, the assumption of C being closed ensures that if C is eventually reached then there exists an earliest time at which this occurs. We adopt the same strategy for the event-triggered language presented below.

Consider a finite set of real-valued variables $X = \{x_1, \ldots, x_n\}$ and denote by $\mathsf{At}^{\mathsf{e}}(X)$ the set given by the grammar,

$$\varphi \ni (x_1 := t, \ldots, x_n := t) \mid (\dot{x}_1 = t, \ldots, \dot{x}_n = t \,\&\, \psi), \qquad t \ni r \mid r \cdot x \mid t + t$$
$$\psi \ni t \leq t \mid t \geq t \mid \psi \wedge \psi \mid \psi \vee \psi$$

where $x \in X$. Then, for a predicate ψ define $[\![\psi]\!] \subseteq \mathbb{R}^n$ by,

$$[\![t_1 \leq t_2]\!] = \{(v_1, \ldots, v_n) \in \mathbb{R}^n \mid [\![t_1]\!] \leq [\![t_2]\!]\} \qquad [\![\psi_1 \wedge \psi_1]\!] = [\![\psi_1]\!] \cap [\![\psi_2]\!]$$
$$[\![t_1 \geq t_2]\!] = \{(v_1, \ldots, v_n) \in \mathbb{R}^n \mid [\![t_1]\!] \geq [\![t_2]\!]\} \qquad [\![\psi_1 \vee \psi_1]\!] = [\![\psi_1]\!] \cup [\![\psi_2]\!]$$

Theorem 3.2.12. *For every predicate ψ the set $[\![\psi]\!]$ is closed in $\mathbb{R}^n$.*

Proof. Start with $[\![t_1 \leq t_2]\!]$. We will show that for every family of real numbers $(a_i, b_i)_{i \in n}$ and $(c, d) \in \mathbb{R} \times \mathbb{R}$ the set,

$$A = \left\{(v_1, \ldots, v_n) \in \mathbb{R}^n \mid \sum\nolimits_{i \in n} a_i v_i + c \leq \sum\nolimits_{i \in n} b_i v_i + d\right\}$$

is closed in $\mathbb{R}^n$. Recall that the order relation $R_{\leq} \subseteq \mathbb{R} \times \mathbb{R}$ is closed and consider two maps $f, g : \mathbb{R}^n \to \mathbb{R}$ defined by,

$$f(v_1, \ldots, v_n) = \sum\nolimits_{i \in n} a_i v_i + c, \qquad g(v_1, \ldots, v_n) = \sum\nolimits_{i \in n} b_i v_i + d$$

Clearly both f and g are continuous since they can be written as compositions of multiplication and addition maps. Moreover, $A = \langle f, g \rangle^{-1}(R_{\leq})$ and therefore A must be closed in $\mathbb{R}^n$.

An analogous reasoning applies to $[\![t_1 \geq t_2]\!]$ since the order relation $R_{\geq} \subseteq \mathbb{R} \times \mathbb{R}$ is also closed. For the cases that involve conjunction and disjunction apply the property of closed sets being closed under intersections and finite unions. □

Corollary 3.2.13. *Consider a program $(\dot{x}_1 = t_1, \ldots, \dot{x}_n = t_n \;\&\; \psi)$, its solution $\phi : \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathbb{R}^n$, and a valuation $(v_1, \ldots, v_n) \in \mathbb{R}^n$. If there exists a time instant $t \in \mathbb{R}_{\geq 0}$ such that $\phi(v_1, \ldots, v_n, t) \in [\![\psi]\!]$ then there exists a* smallest *time instant that also satisfies this condition.*

Proof. By Theorem 3.2.12 the set $\phi(v_1, \ldots, v_n, -)^{-1}([\![\psi]\!]) \cap [0, t]$ is compact and thus it has a minimum. □

We can now introduce an event-triggered programming language in the following manner. Let us define the interpretation map $\mathsf{At}^{\mathsf{e}}(X) \to \mathrm{End}_{\mathrm{H}}(\mathbb{R}^n)$ as the one that sends the expression $(x_1 := t_1, \ldots, x_n := t_n)$ to the function $\mathbb{R}^n \to \mathrm{H}(\mathbb{R}^n)$ defined by,

$$(v_1, \ldots, v_n) \mapsto \eta\left([\![t_1]\!], \ldots, [\![t_n]\!]\right)$$

and the expression $(\dot{x}_1 = t_1, \ldots, \dot{x}_n = t_n \;\&\; \psi)$ to the function $\mathbb{R}^n \to \mathrm{H}(\mathbb{R}^n)$ defined by,

$$(v_1, \ldots, v_n) \mapsto (\phi(v_1, \ldots, v_n, -), d)$$

where d is the smallest time instant that intersects $[\![\psi]\!]$ if $\phi^{-1}(v_1, \ldots, v_n, -)([\![\psi]\!]) \neq \emptyset$ (Corollary 3.2.13) and 0 otherwise. The free monoid extension of this interpretation map provides a hybrid programming language,

$$\mathsf{p} = \mathsf{a} \in \mathsf{At}^{\mathsf{e}}(X) \mid \mathsf{skip} \mid \mathsf{p} \,;\, \mathsf{p}$$

with events.

Examples 3.2.14. Let us analyse it with a few examples.

1. Consider a heating program $(\dot{\mathtt{t}} = 2 \,\&\, \mathtt{t} = 20)$ that terminates when the temperature of $20°$C is reached, and a maintenance program $(\dot{\mathtt{t}} = 0, \dot{\mathtt{u}} = 1 \,\&\, \mathtt{u} = 10)$ that keeps the current temperature, terminating when the instant of time $\mathtt{10}$ reached. If the initial temperature is below $20°$C, the composition,

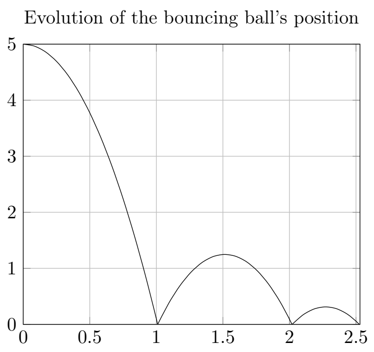
$$(\dot{\mathtt{t}} = 2 \,\&\, \mathtt{t} = 20) \;;\; \mathtt{u} := 0 \;;\; (\dot{\mathtt{t}} = 0, \dot{\mathtt{u}} = 1 \,\&\, \mathtt{u} = 10)$$

makes the current temperature rise to $20°$C, and then keeps it stable for ten seconds. In this context, the variable $\mathtt{u}$ marks the passage of time.

2. Consider a bouncing ball dropped at a positive height $\mathtt{p}$ and with no initial velocity $\mathtt{v}$. Due to the gravitational acceleration $\mathtt{g}$, it falls into the ground and then bounces back up, losing part of its kinetic energy in the process. Consider the program,

$$(\dot{\mathtt{p}} = \mathtt{v}, \dot{\mathtt{v}} = \mathtt{g} \,\&\, \mathtt{p} \leq 0 \wedge \mathtt{v} \leq 0) \;;\; (\mathtt{v} := \mathtt{v} \times -0.5)$$

denoted by $\mathtt{b}$. The composition $(\mathtt{v} := 0, \mathtt{p} := 5) \;;\; \mathtt{b} \;;\; \mathtt{b} \;;\; \mathtt{b}$ encodes the action of dropping the ball at the height of five meters and letting it bounce exactly three times. The projection on $\mathtt{p}$ of this program yields the plot below.



Our next programming language brings evolutions with discontinuities into the scene. As explained in [GST09, Chapter 1], this feature is useful for modelling (i) abrupt changes in a trajectory (resulting *e. g.* from collisions, chemical or biological reactions), and (ii) the evolution of discrete variables over time. Classical examples of (i) range from stopwatches, which along an execution can reset their time variables, to chemical solutions in which one can 'instantaneously' add new quantities of a chemical ingredient that is being gradually consumed [Jac00]. In regard to (ii), typical examples include semaphores (one wants to see which lights are on during a given period of time), digital sound synthesizers, and, more generally, any computational device that needs to provide certain outputs at certain instants of time.

An evolution $(e, d) \in \mathrm{H}X$ induces another evolution $(\langle e, m_d \rangle, d) \in \mathrm{H}(X \times 2)$ where $m_d : [0, d] \to 2$ is defined by $m_d(x) = \bot$ if $x \neq d$ and $m_d(x) = \top$ otherwise. This construction allows to extend the codomain of a morphism $f : X \to \mathrm{H}X$ into $f_2 : X \to \mathrm{H}(X \times 2)$ in such a way that the new program can explicit announce when it terminates.

Recall the interpretation map $\mathsf{At}^{\mathsf{e}}(X) \to \mathrm{End}_{\mathrm{H}}(\mathbb{R}^n)$ that generated the event-triggered programming language introduced before. Let us compose it with the function,

$$\mathrm{End}_{\mathrm{H}}(\mathbb{R}^n) \to \mathrm{End}_{\mathrm{H}}(\mathbb{R}^n \times 2)$$

that sends an endomorphism $f \in \mathrm{End}_{\mathrm{H}}(\mathbb{R}^n)$ to the map g defined by,

$$g(v, x) = \begin{cases} \eta_{\mathbb{R}^n}(v, \perp) & \text{if } x = \perp \\ f_2(v) & \text{otherwise} \end{cases}$$

The free monoidal extension of the composition,

$$\mathsf{At}^{\mathsf{e}}(X) \to \mathrm{End}_{\mathrm{H}}(\mathbb{R}^n \times 2)$$

provides another Kleisli representation for the event-triggered programming language,

$$\mathsf{p} = \mathsf{a} \in \mathsf{At}^{\mathsf{e}}(X) \mid \mathsf{skip} \mid \mathsf{p} \,;\, \mathsf{p}$$

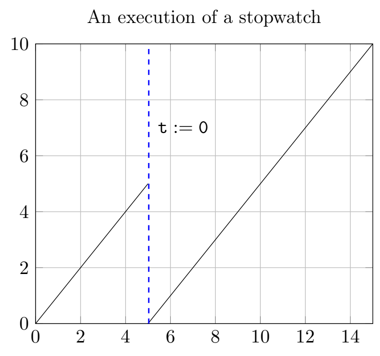
In this language, discrete assignments are applied precisely at the end of an evolution, and evolutions produced by two programs are concatenated.

Examples 3.2.15. Let us analyse some simple cases.

1. Consider a stopwatch that counts time during five seconds, resets, and then counts time during ten seconds. This is given by the composition,

$$\mathtt{t} := 0 \,;\, (\dot{\mathtt{t}} = 1 \,\&\, \mathtt{t} = 5) \,;\, \mathtt{t} := 0 \,;\, (\dot{\mathtt{t}} = 1 \,\&\, \mathtt{t} = 10)$$

which yields the plot below.

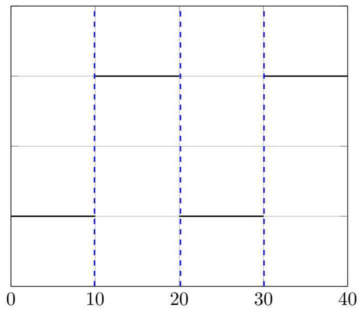


2. Suppose that we want to implement a semaphore that every ten seconds switches between a red light and a green one. It can be obtained by taking the composition,

$$(\mathtt{t} := 0, \mathtt{l} := 0) \,;\, (\dot{\mathtt{t}} = 1 \,\&\, \mathtt{t} = 10) \,;\, (\mathtt{t} := 0, \mathtt{l} := 1) \,;\, (\dot{\mathtt{t}} = 1 \,\&\, \mathtt{t} = 10)$$

here abbreviated to $\mathtt{s}$. The projection on $\mathtt{l}$ of the program $\mathtt{s} \,;\, \mathtt{s}$, for example, yields the plot below.

An execution of a semaphore



3.3 PROGRAM OPERATIONS

3.3.1 *A general theorem for the characterisation of program operations*

Having introduced the hybrid monad and some of its programming languages, our next task is to list all natural transformations of the type $\mathrm{H} \times \mathrm{H} \to \mathrm{H}$. As discussed in Section 3.1, these are the basis of program operations in hybrid programming, and so by detailing them we will be able generate hybrid programming languages more sophisticated than the ones introduced so far.

In order to list the natural transformations above, we will resort to the following theorem which provides a detailed account on natural transformations $TU \times TU \to TU$ for T a coproduct of hom functors[5] and $U : \mathsf{C} \to \mathsf{Set}$ a right adjoint.

Theorem 3.3.1. *Consider an adjunction,*

$$\begin{tikzcd} \mathsf{Set} \arrow[r, bend left, "F"] \arrow[r, phantom, "\perp"] & \mathsf{C} \arrow[l, bend left, "U"] \end{tikzcd}$$

If $T : \mathsf{Set} \to \mathsf{Set}$ is a functor expressible as a coproduct of hom *functors,* i. e. *if there exists a non-empty family $(X_i)_{i \in I}$ of sets such that $T \simeq \coprod_{i \in I} \hom(X_i, -)$, then,*

$$[\mathsf{C}, \mathsf{Set}]\,(TU \times TU, TU) \simeq \prod_{i,j \in I} TUF(X_i + X_j)$$

5 Coproducts of hom functors correspond to the non-indexed version of container [AAG03]. They can also be seen as polynomial functors in Set [GK13].

Proof. In Set binary products distribute over arbitrary coproducts. It follows that,

$$\begin{aligned}
TU \times TU \simeq & (\coprod_{i \in I} \hom(X_i, U)) \times (\coprod_{j \in I} \hom(X_j, U)) \\
\simeq & \coprod_{i,j \in I} \hom(X_i, U) \times \hom(X_j, U) \\
\simeq & \coprod_{i,j \in I} \hom(X_i + X_j, U) \\
\simeq & \coprod_{i,j \in I} \hom(F(X_i + X_j), -)
\end{aligned}$$

where the one but last step follows from contravariant hom functors sending colimits to limits, and the last step follows from U being the right adjoint of F. Then,

$$\begin{aligned}
[\mathsf{C}, \mathsf{Set}]\,(TU \times TU, TU) \simeq & [\mathsf{C}, \mathsf{Set}]\,(\coprod_{i,j \in I} \hom(F(X_i + X_j), -), TU) \\
\simeq & \prod_{i,j \in I} [\mathsf{C}, \mathsf{Set}]\,(\hom(F(X_i + X_j), -), TU) \\
\simeq & \prod_{i,j \in I} TUF(X_i + X_j)
\end{aligned}$$

where the last step is an application of the Yoneda lemma. Given an element $s \in \prod_{i,j \in I} TUF(X_i + X_j)$ with components $s_{i,j} \in TUF(X_i + X_j)$, the respective natural transformation α^s is defined at each X by,

$$(a, b) \in \hom(X_i, UX) \times \hom(X_j, UX) \mapsto TU\overline{[a, b]}(s_{ij})$$

where $\overline{[a, b]}$ is the left adjoint transpose of $[a, b]$. □

Corollary 3.3.2. *If $T : \mathsf{Set} \to \mathsf{Set}$ is a functor expressible as a coproduct of* hom *functors,* i. e. *if there exists a non-empty family $(X_i)_{i \in I}$ of sets such that $T \simeq \coprod_{i \in I} \hom(X_i, -)$, then,*

$$[\mathsf{Set}, \mathsf{Set}]\,(T \times T, T) \simeq \prod_{i,j \in I} T(X_i + X_j)$$

Given an element $s \in \prod_{i,j \in I} T(X_i + X_j)$ with components $s_{i,j} \in T(X_i + X_j)$, the respective natural transformation α^s is defined at each X by,

$$(a, b) \in \hom(X_i, X) \times \hom(X_j, X) \mapsto T\,[a, b]\,(s_{ij})$$

Examples 3.3.3. Let us apply this corollary to some well-known monads, thus allowing us to identify the program operations that their underlying programming paradigms support.

1. The functor of the rectangular bands monad can be written as $\hom(2, -) : \mathsf{Set} \to \mathsf{Set}$, which means that it has precisely $16 \simeq (2 + 2)^2$ natural transformations,

$$(-)^2 \times (-)^2 \to (-)^2$$

Every element $s \in (2+2)^2$ is a list of size two over $2+2$ and it dictates what the transformation α^s does to all tuples $((a,b),(c,d)) \in X^2 \times X^2$; basically, it tells which values given at input will appear in the output and how many times. For example, $i_1(0)i_2(0) \in (2+2)^2$ yields (a,c), $i_1(1)i_2(0)$ yields (b,c), and $i_1(0)i_1(0)$ yields (a,a). Thus, these transformations can also be seen as maps,

$$(\,-\,)^2 \times (\,-\,)^2 \xrightarrow{\langle \pi_{\epsilon(1)} \cdot \pi_{\epsilon(2)}, \pi_{\epsilon(3)} \cdot \pi_{\epsilon(4)} \rangle} (\,-\,)^2$$

where $\epsilon : 4 \to 2$ is an arbitrary map.

2. The functor of the maybe monad can be written as $\mathrm{hom}(\emptyset, -) + \mathrm{hom}(1, -)$. It follows from Corollary 3.3.2 that the natural transformations $\mathrm{M} \times \mathrm{M} \to \mathrm{M}$ are in bijective correspondence with the set $\mathrm{M}(2) \times \mathrm{M}(1) \times \mathrm{M}(1) \times \mathrm{M}(0)$, in particular there are exactly $3 \times 2 \times 2 \times 1 = 12$ natural transformations $\mathrm{M} \times \mathrm{M} \to \mathrm{M}$. The $\mathrm{M}(2)$-coordinate specifies what a transformation does to pairs (x,y) with $x, y \neq *$, *i. e.* projecting to the left, to the right or mapping to $*$, the first $\mathrm{M}(1)$-coordinate specifies what happens to pairs $(x,*), x \neq *$, *i. e.* projecting to the left or the right, and similarly for the last coordinate and pairs $(*,y), y \neq *$.

3. The functor of the list monad can be written as the coproduct $\coprod_{n \in \mathbb{N}} \mathrm{hom}(n, -)$. Thus, the natural transformations $\mathrm{L} \times \mathrm{L} \to \mathrm{L}$ are in one-to-one correspondence with $\prod_{i,j \in \mathbb{N}} \mathrm{L}(i+j)$. Similarly to the rectangular bands monad, each (i,j)-coordinate specifies what a transformation does to pairs of lists of length i and j. In particular, it tells how the values in a given pairs of lists are distributed in the new list. For example, if the $(1,2)$-coordinate is given by the word $i_1(1)i_2(1)i_1(1)i_2(1)i_2(2)$ then the natural transformation will send a pair of words (a, bc) of length 1 and 2, respectively, to $ababc$.

4. The functor of the hybrid monad is the coproduct $\coprod_{d \in \mathbb{R}_{\geq 0}} \mathrm{hom}([0,d], \,-\,)$, therefore the natural transformations $\mathrm{H} \times \mathrm{H} \to \mathrm{H}$ are in bijective correspondence with the set $\prod_{i,j \in \mathbb{R}_{\geq 0}} \mathrm{H}([0,i] + [0,j])$. Similarly to the list monad, each (i,j)-coordinate s_{ij} dictates what a transformation does to pairs of evolutions with duration $[0,i]$ and $[0,j]$. In particular, it tells how the values in a given pair of evolutions (f,g) of duration $[0,i]$ and $[0,j]$ are distributed in the new evolution: one has the composition,

$$\begin{tikzcd}[0,k] \arrow[r, "s_{ij}"] \arrow[rr, bend right, "\alpha^s_X(f,g)"'] & {[0,i] + [0,j]} \arrow[r, "{[f,g]}"] & X\end{tikzcd}$$

which makes clear that for every element $a \in [0,k]$ the value $\alpha^s_X(f,g)\,(a)$ arises from one of the two starting functions f or g. For example, if $i_1 = s_{ij} : [0,i] \to [0,i] + [0,j]$ then $\alpha^s_X(f,g) = f$, and if $s_{ij} : [0,i] \to [0,i] + [0,j]$ is defined by,

$$s_{ij}(a) = \begin{cases} i_1(a) & \text{if } a \in \mathbb{Q} \\ i_2(a) & \text{otherwise} \end{cases}$$

then $\alpha^s_X(f,g)(a) = f(a)$ if $a \in \mathbb{Q}$ and $\alpha^s_X(f,g)(a) = g(a)$ otherwise.

5. Using the isomorphisms $\mathrm{Id} \simeq \hom(1, -)$, $A \simeq \coprod_{a \in A} \hom(\emptyset, -)$, and induction it is routine to prove that Corollary 3.3.2 applies to *all* polynomial functors on Set.

Remark 3.3.4. [AAG03, Theorem 3.4] provides a powerful representation theorem that can also be used to prove Corollary 3.3.2 (a particular case of Theorem 3.3.1). This, however, involves the notions of container and fibration which is not required in our case.

3.3.2 *Axioms*

Corollary 3.3.2 provides a detailed description of natural transformations of the type $\mathrm{H} \times \mathrm{H} \to \mathrm{H}$, which as mentioned before, are the basis of hybrid program operations. In order to study program operations' axiomatics, we will further enrich this description by characterising the natural transformations $\mathrm{H} \times \mathrm{H} \to \mathrm{H}$ that satisfy common axioms, namely commutativity, idempotence, associativity, and units.

Commutativity. We start with the following proposition which tells that commutative natural transformations $T \times T \to T$ (*i. e.* those whose components are commutative) for coproducts of hom functors T must adhere to rather harsh conditions.

Proposition 3.3.5. *If $T \simeq \coprod_{i \in I} \hom(X_i, -)$, a natural transformation $\alpha : T^2 \to T$, given by an element $(s_{ij})_{i,j \in I} \in \prod_{i,j \in I} T(X_i + X_j)$ via Corollary 3.3.2, is commutative iff for all $i, j \in I$, the equation below holds.*

$$[i_2, i_1] \cdot s_{ij} = s_{ji} : X_k \to X_j + X_i$$

In particular, for all $i \in I$, s_{ii} must be the map with empty domain, which means that if there is no set $X_i = \emptyset$ then there is also no commutative natural transformation $T^2 \to T$.

Proof. If a natural transformation α^s is commutative then for every $a : X_i \to X$, $b : X_j \to X$ the equation,

$$\alpha^s(a, b) = [a, b] \cdot s_{ij} = \alpha^s(b, a) = [b, a] \cdot s_{ji}$$

must hold. In particular, the equation,

$$[i_2, i_1] \cdot s_{ij} = [i_1, i_2] \cdot s_{ji} = s_{ji} \tag{4}$$

holds for the injections $i_2 : X_i \to X_j + X_i$, $i_1 : X_j \to X_j + X_i$.

Let us now assume that Equation (4) holds. The goal is to show that for every $a : X_i \to X$, $b : X_j \to X$ the equation $[a, b] \cdot s_{ij} = [b, a] \cdot s_{ji}$ holds. Thus reason,

$$[b, a] \cdot s_{ji} = [b, a] \cdot [i_2, i_1] \cdot s_{ij} = [a, b] \cdot s_{ij}$$

□

Examples 3.3.6. Let us analyse the monads in Examples 3.3.2 under the light of this proposition.

1. Since $(\,-\,)^2 \simeq \hom(2,\,-\,)$ and $2 \not\simeq \emptyset$, by Proposition 3.3.5 there is no commutative natural transformation $(\,-\,)^2 \times (\,-\,)^2 \to (\,-\,)^2$.

2. A natural transformation $\mathrm{M} \times \mathrm{M} \to \mathrm{M}$ for the maybe monad can be commutative only if its $(1,1)$-coordinate lies in the third summand of $\mathrm{M}(1+1)$. In other words, if two elements different than failure are mapped to failure. A commutative natural transformation $\mathrm{M} \times \mathrm{M} \to \mathrm{M}$ is thus completely determined by the coordinates $\mathrm{M}(\emptyset+1)$ and $\mathrm{M}(1+\emptyset)$. Actually, even just $\mathrm{M}(\emptyset+1)$ is enough, since the equation $[i_2, i_1] \cdot s_{\emptyset 1} = s_{1\emptyset}$ holds and therefore $s_{\emptyset 1}$ completely determines $s_{1\emptyset}$. This means that there are precisely two commutative natural transformations $\mathrm{M} \times \mathrm{M} \to \mathrm{M}$.

3. Commutative natural transformations $\mathrm{L} \times \mathrm{L} \to \mathrm{L}$ must always map pairs of lists with the same length to the empty list.

4. Since $\mathrm{H} = \coprod_{d \in \mathbb{R}_{\geq 0}} \hom([0,d],\,-\,)$ and $[0,d] \not\simeq \emptyset$ for every $d \in \mathbb{R}_{\geq 0}$, there cannot exist commutative natural transformations $\mathrm{H} \times \mathrm{H} \to \mathrm{H}$.

As illustrated in the examples above, commutativity is a problematic axiom for the class of functors treated by Corollary 3.3.2. It necessarily leads to trivial natural transformations, in the sense that no pair of instructions in the same set $\hom(X_i, X)$ for $\coprod_{i \in I} \hom(X_i,\,-\,) \simeq T$ can be non-trivially combined. As shown below, commutativity also does not get along with idempotence.

Idempotence. We start with a result in the same spirit than Proposition 3.3.5.

Proposition 3.3.7. *If $T \simeq \coprod_{i \in I} \hom(X_i, -)$ a natural transformation $\alpha : T^2 \to T$, given by an element $(s_{ij})_{i,j \in I} \in \prod_{i,j \in I} T(X_i + X_j)$ via Corollary 3.3.2, is idempotent iff for each $i \in I$, $s_{ii} \in T(X_i + X_i)$ is a map $s_{ii} : X_i \to X_i + X_i$ such that $\nabla \cdot s_{ii} = \mathrm{id}$ where $\nabla : X_i + X_i \to X_i$ is the codiagonal.*

Proof. If a transformation α^s is idempotent then for every $i \in I$, $a : X_i \to X$, the equation $\alpha^s(a,a) = [a,a] \cdot s_{ii} = a : X_i \to X$ holds. In particular, we have

$$\nabla \cdot s_{ii} = [\mathrm{id}, \mathrm{id}] \cdot s_{ii} = \mathrm{id} : X_i \to X_i \tag{5}$$

Now assume that Equation (5) holds and reason,

$$a = a \cdot \mathrm{id} = a \cdot [\mathrm{id}, \mathrm{id}] \cdot s_{ii} = [a,a] \cdot s_{ii}$$

□

Examples 3.3.8. The idempotent natural transformations $T \times T \to T$ for the functors in Examples 3.3.2 are described in the following manner.

1. Idempotent natural transformations $\alpha^s : (-)^2 \times (-)^2 \to (-)^2$ are in bijective correspondence with maps $s : 2 \to 2 + 2$ such that $\nabla \cdot s = \mathrm{id}$. The latter correspond to the words $i_1(0)i_1(1)$, $i_1(0)i_2(1)$, $i_2(0)i_1(1)$, and $i_2(0)i_2(1)$. Therefore, there are exactly four idempotent natural transformations for the rectangular bands monad.

2. A natural transformation $\mathrm{M} \times \mathrm{M} \to \mathrm{M}$ is idempotent iff its $(1,1)$-coordinate is either in the first or second summand of $\mathrm{M}(1+1)$. Hence, there exist precisely eight idempotent natural transformations for the maybe monad.

3. The idempotent natural transformations $\mathrm{L} \times \mathrm{L} \to \mathrm{L}$ are the ones which take pairs of words (w^1, w^2) of length n to a word $w_1^{\epsilon(1)} \dots w_n^{\epsilon(n)}$, where ϵ is a map $n \to \{1,2\}$ and w_i^j is the i-th letter in the word $w^j, j = 1,2$. More succintly, the i-th letter of the new word is either the i-th letter of w^1 or the i-th letter of w^2.

4. The idempotent natural transformations $\alpha^s : \mathrm{H} \times \mathrm{H} \to \mathrm{H}$ for the hybrid monad follow a reasoning analogous to the one for the list monad: the transformation α^s is idempotent iff for every two evolutions (f, g) with the same duration $[0, d]$, $\alpha^s(f, g)$ has domain $[0, d]$, and for every element $a \in [0, d]$ either $\alpha^s(f, g)(a) = f(a)$ or $\alpha^s(f, g)(a) = g(a)$. Thus, the restriction of an idempotent natural transformation $\alpha^s : \mathrm{H} \times \mathrm{H} \to \mathrm{H}$ to pairs of evolutions with duration $[0, d]$ can be seen as a map,

$$(-)^{[0,d]} \times (-)^{[0,d]} \xrightarrow{\langle \pi_r \cdot \pi_{\rho(r)} \rangle_{r \leq d}} (-)^{[0,d]}$$

where $\rho : [0, d] \to 2$ is a function.

It follows from Propositions 3.3.5 and 3.3.7 that the combination of commutativity and idempotence is impossible when T is a coproduct of hom functors with an $X_i \neq \emptyset$ in its presentation, since then Proposition 3.3.5 requires that s_{ii} be of type $\emptyset \to X_i + X_i$ but Proposition 3.3.7 requires that s_{ii} be of type $X_i \to X_i + X_i$. Thus none of the monads in Examples 3.3.8 can support a programming language with a commutative and idempotent binary operation on programs.

Associativity. In general, we have not found any useful characterisation of associativity, and unravelling the definition seems to be the surest way of checking whether associativity holds for a given natural transformation $\mathrm{H} \times \mathrm{H} \to \mathrm{H}$. We can, however, make the following observations: there exist associative natural transformations $\mathrm{H} \times \mathrm{H} \to \mathrm{H}$, the obvious projections being two prime examples. There also exist non-associative natural transformations $\mathrm{H} \times \mathrm{H} \to \mathrm{H}$. Take, for example, the natural transformation that sends the pair $((f, d), (g, e)) \in \mathrm{H}X \times \mathrm{H}X$ to $\eta_X \cdot f(0)$ if $d < e$ and to $\eta_X \cdot g(0)$ otherwise.

Units. We now examine the possibility of representing units for natural transformations of the type $\mathrm{H} \times \mathrm{H} \to \mathrm{H}$. At first sight, there are two obvious possibilities: a unit can either be a constant, represented by a natural transformation $1 \to \mathrm{H}$, or it can be the unit of $\mathbb{H}$. However,

Corollary 3.3.9. *The hybrid monad has no natural transformation* $1 \to \mathrm{H}$.

Proof. Observe that $\mathrm{H}\emptyset = \emptyset$ and $1 \simeq \hom(\emptyset, -)$. Then apply Yoneda lemma to obtain $[\mathsf{Set}, \mathsf{Set}](1, \mathrm{H}) \simeq [\mathsf{Set}, \mathsf{Set}](\hom(\emptyset, -), \mathrm{H}) \simeq \mathrm{H}\emptyset \simeq \emptyset$. □

So the first possibility is unviable. The following theorem shows that the second possibility is unviable as well.

Theorem 3.3.10. *Consider a monad* $\mathbb{T}$ *with the unit* $\eta : \mathrm{Id} \to T$ *injective and with* $T \simeq \coprod_{i \in I} \hom(X_i, -)$. *Under these conditions,* η *can never be the unit of a natural transformation* $\alpha^s : T \times T \to T$.

Proof. Using Yoneda lemma and the fact that $\mathrm{Id} \simeq \hom(1, -)$, we can show that $\eta : \mathrm{Id} \to T$ factorises through an injection $\hom(X_k, -) \to T$. Then since η is injective, thet set X_k cannot be empty, otherwise $\hom(X_k, -) = \hom(\emptyset, -) \simeq 1$. Now consider the two-point set 2, observe that $\eta_2(0) \in \hom(X_k, 2)$, and define $a = \phi \cdot (\eta_2(0))$ where $\phi : 2 \to 2$ is the 'bit flip' operation. Assuming that η is the monad's unit entails that the equation,

$$[a, \eta_2(0)] \cdot s_{kk} = [\eta_2(0), a] \cdot s_{kk} : X_k \to 2$$

holds. However, this cannot happen when $X_k \neq \emptyset$. □

Corollary 3.3.11. *All monads* $\mathbb{T}$ *in Examples 3.3.8 cannot have* $\eta : \mathrm{Id} \to T$ *as the unit of a natural transformation* $T \times T \to T$.

3.3.3 *Program operations over algebraic structures*

Recall that natural transformations $T \times T \to T$ for a monad $\mathbb{T}$ are traditionally the basis for $\mathbb{T}$-program operations [Mog91; PP01b; HKL14]. To conclude this section, we propose a more generic approach: to adopt instead natural transformations of the type $TU \times TU \to TU$ where U is the forgetful functor of a variety. In principle, these transformations allow to use different algebraic structures on program operations, which ultimately leads to more powerful ways of combining programs. This is particularly clear in the context of hybrid programming, since the set $\mathbb{R}^n$ – which lies at the center of our hybrid programming languages – has very rich algebraic structures.

To illustrate our point, let us characterise these transformations in the same way that we did for those of the type $T \times T \to T$. To start, observe that the functor U mentioned above is always a right adjoint, and this means that Theorem 3.3.1 can be used in our quest.

Examples 3.3.12. Let $\mathrm{U} : \mathsf{Mon} \to \mathsf{Set}$ be the forgetful functor of monoids.

1. Consider the functor of the rectangular bands monad. The natural transformations of the type $\mathrm{U}^2 \times \mathrm{U}^2 \to \mathrm{U}^2$ are in bijective correspondence with the set $(\mathrm{L}(2+2))^2$, whose elements are pairs of lists over $2 + 2$. One such element s dictates what the transformation

α^s_M does to all tuples $((a,b),(c,d)) \in (\mathrm{U}M)^2 \times (\mathrm{U}M)^2$ where M abbreviates $(X, \cdot, 1)$. In particular, the left value in the pair $\alpha^s_M((a,b),(c,d))$ is the M-product of a list over $\{a,b,c,d\}$, which is obtained from the left list in s, and similarly for the right value. For example, if s is (ϵ, ϵ) then $\alpha^s_M((a,b),(c,d)) = (1,1)$ and if s is $(i_1(1)i_2(1), i_1(2)i_2(2))$ then $\alpha^s_M((a,b),(c,d)) = (a \cdot c, b \cdot d)$.

2. Consider now the maybe monad. The natural transformations of the type $\mathrm{MU} \times \mathrm{MU} \to \mathrm{MU}$ are in bijective correspondence with $\mathrm{ML}\emptyset \times \mathrm{ML}1 \times \mathrm{ML}1 \times \mathrm{ML}2$. The set $\mathrm{ML}\emptyset$ specifies what a transformation does to the pair $(*,*)$, *i. e.* sending to $*$ or to the unit of the monoid; the set $\mathrm{ML}1$ tells what happens to pairs $(x,*)$, *i. e.* sending to $*$ or to,

$$\underbrace{x \cdot \ldots \cdot x}_{n \text{ times}}$$

for some natural number n, and similarly for pairs $(*,x)$. The ML2-coordinate tells what happens to pairs (x,y), *i. e.* sending to $*$ or to the M-product of a list over $\{x,y\}$.

3. Consider the hybrid monad. The natural transformations of the type $\mathrm{HU} \times \mathrm{HU} \to \mathrm{HU}$ are in bijective correspondence with the set,

$$\coprod_{i,j \in \mathbb{R}_{\geq 0}} \mathrm{HL}([0,i] + [0,j])$$

Each (i,j)-coordinate tells what a transformation α^s_M does to pairs of evolutions $((f,i),(g,j))$ with duration $[0,i]$ and $[0,j]$. In particular, one has the composition,

$$\begin{tikzcd}[0,k] \arrow[r, "s_{ij}"] \arrow[rr, bend right, "\alpha^s_M(f,g)"'] & \mathrm{L}([0,i]+[0,j]) \arrow[r, "\overline{[f,g]}"] & \mathrm{U}M\end{tikzcd}$$

where $\overline{[f,g]}$ is the free monoid extension (*i. e.* left adjunct) of $[f,g]$. This means that for every $a \in [0,k]$ the value $\alpha^s_M(f,g)(a)$ is the M-product of a list over $\operatorname{img} f \cup \operatorname{img} g$ which is specified by $s_{ij}(a)$. For example, if $s_{ij} : [0, \max(i,j)] \to \mathrm{L}([0,i] + [0,j])$ is defined by,

$$s_{ij}(a) = i_1(\min(a,i))i_2(\min(a,j))$$

then the respective natural transformation $(+) : \mathrm{HU} \times \mathrm{HU} \to \mathrm{HU}$ M-multiplies the evolutions f and g pointwise: in particular, if M is the monoid $(\mathbb{R}, +, 0)$ and $i = j$ then $(f,i) +_M (g,i)) = (f+g, i)$.

4. The natural transformations for the list monad follow a reasoning similar to the one used in the previous case.

The natural transformations $\mathrm{HU} \times \mathrm{HU} \to \mathrm{HU}$ with U the forgetful functor for groups can be described in a manner similar to above; the only difference is that whilst in the case of monoids a natural transformation always outputs evolutions whose values are M-products of lists, in the case of groups the transformations output evolutions whose values are M-products of lists that contain inverses. For example, take the components $s_{ij} : [0, \max(i, j)] \to [0, i] + [0, j]$ defined by,

$$s_{ij}(a) = i_1(\min(a, i))i_2(\min(a, j)) - i_1(0)$$

Then the induced natural transformation $(\|) : \mathrm{HU} \times \mathrm{HU} \to \mathrm{HU}$ sends two evolutions $((f, i), (g, j))$ to the map $(h, \max(i, j))$ defined by,

$$h(x) = f(\min(x, i)) \cdot g(\min(x, j)) - f(0)$$

Recall from Examples 3.3.3 (4) that no natural transformation $\mathrm{H} \times \mathrm{H} \to \mathrm{H}$ behaves as any of the two natural transformations $(+)$ and $(\|)$, defined above. Therefore, one indeed gains more expressive power by adopting natural transformations of the type $TU \times TU \to TU$ instead of the ones used in the classical approach. As a further attestment to this, note that $(+)$ is commutative and we previously showed that there is no commutative natural transformation $\mathrm{H} \times \mathrm{H} \to \mathrm{H}$.

Examples 3.3.13. Let us now explore natural transformations of the type $\mathrm{HU} \times \mathrm{HU} \to \mathrm{HU}$ in the context of hybrid programming.

1. A harmonic oscillator [Blo13] is traditionally specified by the differential equation $\ddot{x} = -\omega^2 x$, where $\omega/2\pi$ is the angular frequency. The solution to this equation is equal to the solution of the system,

   $$\dot{p} = v, \dot{v} = -\omega^2 p$$

   projected on p, *i. e.* the map $\phi : \mathbb{R}^2 \times \mathbb{R}_{\geq 0} \to \mathbb{R}$ defined by,

   $$\phi(p, v, t) = p\cos(\omega t) + (v/\omega)\sin(\omega t)$$

   Thus, the projection on $\mathtt{p}$ of the program,

   $$(\mathtt{p} := \mathtt{a}, \mathtt{v} := 0, \mathtt{u} := 0) \; ; \; (\dot{\mathtt{p}} = \mathtt{v}, \dot{\mathtt{v}} = -\omega^2\mathtt{p}, \dot{\mathtt{u}} = 1 \;\&\; \mathtt{u} = 20)$$

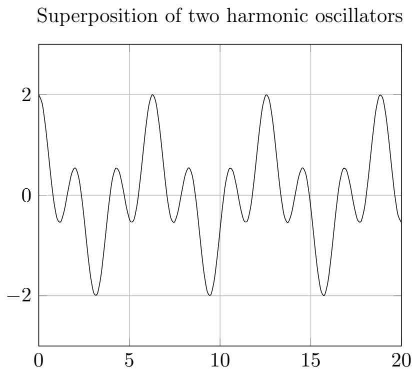
   yields the function $t \mapsto \mathtt{a}\cos(\omega t)$ with duration twenty. Now let us consider the natural transformation $(+) : \mathrm{HU} \times \mathrm{HU} \to \mathrm{HU}$ of Examples 3.3.12 which M-multiplies two evolutions pointwise. It is associative and it has a unit given by the natural transformation $1 \to \mathrm{HU}$ that picks the evolution with duration 0 and constant on the unit of the monoid M. The set of endomorphisms $\mathrm{End}_{\mathrm{H}}(\mathbb{R}^n)$ can thus be equipped with a double monoid structure (*i. e.* two monoids over the same carrier), as shown in the definition of Kleisli representation (Definition 3.1.14). Now recall the interpretation map $\mathsf{At}^{\mathsf{e}}(X) \to \mathrm{End}_{\mathrm{H}}(\mathbb{R}^n)$ respective to the event-triggered language of Examples 3.2.14. Its free bimonoid extension provides a hybrid programming language,

   $$\mathsf{p} = \mathsf{a} \in \mathsf{At}^{\mathsf{e}}(X) \mid \mathsf{skip} \mid \mathsf{p} \; ; \; \mathsf{p} \mid 1 \mid \mathsf{p} + \mathsf{p}$$

that brings the principle of *superposition* into play [Fre12]. For example, we can now sum two harmonic oscillators,

$$
\begin{aligned}
&(\mathtt{p} := 1, \mathtt{v} := 0, \mathtt{u}_1 := 0, \mathtt{u}_2 := 0) \ ; \\
&((\dot{\mathtt{p}} = \mathtt{v}, \dot{\mathtt{v}} = -\mathtt{p}, \dot{\mathtt{u}_1} = 1 \ \& \ \mathtt{u}_1 = 20) + (\dot{\mathtt{p}} = \mathtt{v}, \dot{\mathtt{v}} = -9\mathtt{p}, \dot{\mathtt{u}_2} = 1 \ \& \ \mathtt{u}_2 = 20))
\end{aligned}
$$

The projection on $\mathtt{p}$ of this program yields the following plot.



2. Recall that the water level of a tank can be described by the differential equation $\dot{l} = k$ and consider the natural transformation $(\|) : \mathrm{HU} \times \mathrm{HU} \to \mathrm{HU}$ of Examples 3.3.12, which is defined at each group G by,

$$(f, i) \parallel_G (g, j) = (h, \max(i, j))$$

with $h(x) = f(\min(x, i)) + g(\min(x, j)) - f(0)$. This operation is associative, and thus the set of endomorphisms $\mathrm{End}_{\mathrm{H}}(\mathbb{R}^n)$ can be equipped with a monoid and a semigroup structure, as shown in the definition of Kleisli representation (Definition 3.1.14). Consider the interpretation map $\mathsf{At}^{\mathsf{e}}(X) \to \mathrm{End}_{\mathrm{H}}(\mathbb{R}^n)$ that was used in the previous example. The corresponding free extension now provides a hybrid programming language whose programs can contribute to (or compete over) shared resources.

$$\mathsf{p} = \mathsf{a} \in \mathsf{At}^{\mathsf{e}}(X) \mid \mathsf{skip} \mid \mathsf{p} \ ; \ \mathsf{p} \mid \mathsf{p} \parallel \mathsf{p}$$

For example, the effect of two open valves that let water flow in can be modelled by the composition,

$$(\mathtt{u}_1 := 0, \mathtt{u}_2 := 0) \ ; ((\dot{\mathtt{l}} = \mathtt{k}, \dot{\mathtt{u}_1} = 1 \ \& \ \mathtt{u}_1 = 10) \parallel (\dot{\mathtt{l}} = \mathtt{m}, \dot{\mathtt{u}_2} = 1 \ \& \ \mathtt{u}_2 = 10))$$

which is equivalent to the program below.

$$(\mathtt{u}_1 := 0, \mathtt{u}_2 := 0) \ ; (\dot{\mathtt{l}} = \mathtt{k} + \mathtt{m}, \dot{\mathtt{u}_1} = 1, \dot{\mathtt{u}_2} = 1 \ \& \ \mathtt{u}_1 = 10)$$

3.4 CLASSICAL PROGRAMMING CONSTRUCTS

3.4.1 *If-then-else*

In non-deterministic programming, if-then-else constructs emerge from two well-known natural transformations [Koz97], namely $\underline{\emptyset} : 1 \to \mathrm{P}$, constant on the empty set, and $\alpha^+ : \mathrm{P} \times \mathrm{P} \to \mathrm{P}$ which sends two sets to their union. The first one introduces tests: given a condition ψ, the test program $[\![\psi]\!] : X \to \mathrm{P}X$ is defined by $[\![\psi]\!](x) = \{x\} = \eta_X(x)$ if x satisfies ψ and $[\![\psi]\!](x) = \emptyset = \underline{\emptyset}_X \cdot !_X(x)$ otherwise. The second combines the two possible execution paths of an if-then-else statement. Both transformations together give rise to if-then-else statements by defining $\mathsf{if}\ \psi\ \mathsf{then}\ \mathsf{p}\ \mathsf{else}\ \mathsf{q}$, or more shortly $\mathsf{p} +_\psi \mathsf{q}$, as,

$$(\psi\ ;\ \mathsf{p}) + ((\neg\psi)\ ;\ \mathsf{q})$$

This procedure also applies to faulty programs by taking the natural transformation $1 \to \mathrm{M}$, constant on 'failure', and the natural transformation $\alpha : \mathrm{M} \times \mathrm{M} \to \mathrm{M}$ defined at each set X by $\alpha_X(*, *) = \alpha_X(x, y) = *$ and $\alpha_X(*, x) = \alpha(x, *) = x$. It does *not* apply, however, in a general setting, since it presupposes the existence of two natural transformations that might not always exist. This is in fact the case for the distribution and hybrid monads, which do not have a natural transformation $1 \to T$ (Corollary 3.1.18 and Corollary 3.3.9).

In order to solve this issue, we will 'embed' if-then-else constructs directly in the notion of Kleisli representation by replacing monoids, as the basic structure of programming languages (see Section 3.1), with *conditional monoids.*

Definition 3.4.1. Let Ψ be a set of conditions. A Ψ-conditional monoid is a tuple $(X, \cdot, 1, (m_\psi)_{\psi \in \Psi})$ such that $(X, \cdot, 1)$ is a monoid and for every condition $\psi \in \Psi$, $m_\psi : X \times X \to X$ is a binary map.

Consider the family of maps $s_{\psi \in \Psi} : X \to X + X$ defined by,

$$s_\psi(x) = \begin{cases} i_1(x) & \text{if } x \text{ satisfies } \psi \\ i_2(x) & \text{otherwise} \end{cases}$$

A monoid of endomorphisms $(\mathrm{End}_{\mathbb{T}}(X), \bullet, \eta)$ can be extended into a Ψ-conditional monoid by defining,

$$m_\psi(f, g) = [f, g] \cdot s_\psi \tag{6}$$

Intuitively, if an input $x \in X$ satisfies ψ then the program $m_\psi(f, g) : X \to TX$ tells f to execute with x as input, otherwise g executes also with x as input. Such is the classical behaviour of if-then-else statements and moreover this construction applies to *all* monads on Set.

Definition 3.4.2. Let V be a finitary quasi-variety defined by a signature $\Sigma = \{\mathsf{skip}, ;\} \cup \Psi \cup \Phi$ and a set of quasi-equations such that the $\{\mathsf{skip}, ;, \Psi\}$-fragment is the variety of Ψ-conditional monoids. Consider also a Set-monad $\mathbb{T}$ equipped with a family of maps $s_{\psi \in \Psi} : X \to X + X$.

A *conditional Kleisli $\mathbb{T}$-representation* of a V-object A is an assignment of every $\sigma \in \Phi$ to a natural transformation $\alpha^\sigma : T^{\mathrm{ar}(\sigma)} \to T$ together with a V-algebra morphism,

$$A \to (\mathrm{End}_{\mathbb{T}}(X), \bullet, \eta_X, (m_\psi)_{\psi \in \Psi}, (\llbracket \sigma \rrbracket)_{\sigma \in \Phi})$$

with m_ψ defined as in Equation (6).

Recall the event-triggered programming language of Examples 3.2.14. In order to extend it with if-then-else constructs, we just need a suitable set of conditions Ψ and a family of satisfaction maps $s_{\psi \in \Psi} : \mathbb{R}^n \to \mathbb{R}^n + \mathbb{R}^n$. Let Ψ be the set generated by the grammar,

$$\psi \ni t \leq t \mid t \geq t \mid \psi \wedge \psi \mid \psi \vee \psi \mid \neg \psi, \qquad t \ni r \mid r \cdot x \mid t + t$$

where $x \in X$ and let $s_\psi : \mathbb{R}^n \to \mathbb{R}^n + \mathbb{R}^n$ be the canonical extension of the map,

$$\begin{aligned}
(t_1 \leq t_2, (v_1, \dots, v_n)) &\mapsto i_1(v_1, \dots, v_n) \text{ if } \llbracket t_1 \rrbracket_{(v_1,\dots,v_n)} \leq \llbracket t_2 \rrbracket_{(v_1,\dots,v_n)} \\
(t_1 \leq t_2, (v_1, \dots, v_n)) &\mapsto i_2(v_1, \dots, v_n) \text{ if } \llbracket t_1 \rrbracket_{(v_1,\dots,v_n)} \not\leq \llbracket t_2 \rrbracket_{(v_1,\dots,v_n)} \\
(t_1 \geq t_2, (v_1, \dots, v_n)) &\mapsto i_1(v_1, \dots, v_n) \text{ if } \llbracket t_1 \rrbracket_{(v_1,\dots,v_n)} \geq \llbracket t_2 \rrbracket_{(v_1,\dots,v_n)} \\
(t_1 \geq t_2, (v_1, \dots, v_n)) &\mapsto i_2(v_1, \dots, v_n) \text{ if } \llbracket t_1 \rrbracket_{(v_1,\dots,v_n)} \not\geq \llbracket t_2 \rrbracket_{(v_1,\dots,v_n)}
\end{aligned}$$

Denote the forgetful functor of the variety of Ψ-conditional monoids by U, and recall the interpretation map $\mathsf{At}^{\mathsf{e}}(X) \to \mathrm{End}_{\mathrm{H}}(\mathbb{R}^n)$ of the event-triggered programming language just mentioned. The free extension of the interpretation map,

$$\mathsf{At}(X) \to \mathrm{U}(\mathrm{End}_{\mathbb{T}}(X), \bullet, \eta_X, (m_\psi)_{\psi \in \Psi})$$

provides an event-triggered hybrid programming language with if-then-else constructs.

$$\mathsf{p} = \mathsf{a} \in \mathsf{At}^{\mathsf{e}}(X) \mid \mathsf{skip} \mid \mathsf{p} \,;\, \mathsf{p} \mid \mathsf{p} +_\psi \mathsf{p} \qquad\qquad (\psi \in \Psi)$$

Examples 3.4.3. Let us explore this new language.

1. Consider a tank with water flowing out at one *cm/s*. Then suppose a computer checks every ten seconds if the water level is lower than two *cm*; if such is the case, it opens a valve that lets water flow in at two *cm/s*, otherwise it either keeps the valve closed or closes it. Denote the program,

   $$(\mathtt{e} := 2 \ +_{\mathtt{l}<2} \ \mathtt{e} := 0) \,;\, \mathtt{u} := 0 \,;\, (\dot{\mathtt{l}} = \mathtt{k} + \mathtt{e}, \dot{\mathtt{u}} = 1 \,\&\, (\mathtt{l} = 0 \wedge \mathtt{k} + \mathtt{e} < 0) \vee \mathtt{u} = 10) \,;$$
   $$(\dot{\mathtt{l}} = 0 \,, \dot{\mathtt{u}} = 1 \,\&\, \mathtt{u} = 10)$$

   by $\mathtt{w}$. The composition,

   $$\underbrace{\mathtt{w} \,;\, \dots \,;\, \mathtt{w}}_{6 \text{ times}} = \mathtt{w}^6$$

   encodes the behaviour of the system described above for six iterations. Note that the left side of the disjunction $(\mathtt{l} = 0 \wedge \mathtt{k} + \mathtt{e} \leq 0) \vee \mathtt{u} = 10$ forbids the water level to go below zero, which is a physical constraint of our system.

2. Consider a (simplistic) convoy system comprised of a truck that moves at constant velocity and a *follower truck* equipped with a computer that either orders the truck to accelerate with force $\mathtt{a} > 0$ or brake with force $\mathtt{b} < 0$. Now let $\mathtt{d}$ be the desired minimum distance between the leader and the follower, and assume that the periodicity of the computer's orders is exactly one second. The condition $\mathtt{p_l} + \mathtt{v_l} - \mathtt{p_f} - \mathtt{v_f} - \frac{1}{2}\mathtt{a} \geq \mathtt{d}$ – which we will abbreviate to ψ – tells whether the minimum distance is going to be violated in the next second if acceleration $\mathtt{a}$ is chosen: to see why, note that $\mathtt{p_l} + \mathtt{v_l}$ is the leader's position at the next second and $\mathtt{p_f} + \mathtt{v_f} + \frac{1}{2}\mathtt{a}$ is the follower's position at the next second if acceleration $\mathtt{a}$ is chosen.

   Denote the program,

$$(\mathtt{a_f} := \mathtt{a} \ +_\psi \ \mathtt{a_f} := \mathtt{b}) \ ; \mathtt{u} := 0 \ ; (\dot{\mathtt{p_l}} = \mathtt{v_l},\ \dot{\mathtt{p_f}} = \mathtt{v_f}, \dot{\mathtt{v_f}} = \mathtt{a_f}, \dot{\mathtt{u}} = 1 \ \& \ \mathtt{u} = 1)$$

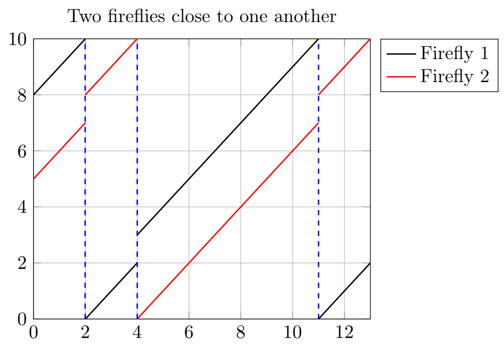
   by $\mathtt{c}$. The composition $\mathtt{c}^{10}$ encodes the possible executions of the convoy system for ten iterations.

3. Consider the following *biological* system, which was addressed in [GST09]. Every firefly has an internal clock which helps it to know when to flash: when the clock reaches a threshold the firefly flashes and the clock's value is reset to zero. The flash of a firefly increases the internal clock's value of all other fireflies nearby.

   In order to model this biological system, we will need the programming language with discontinuities that was explored in Examples 3.2.15. We can easily extend it with if-then-else constructs using conditional Kleisli representations, and this allows us to build the program below.

$$\begin{array}{l}(\dot{\mathtt{f}_1} = 1, \dot{\mathtt{f}_2} = 1 \ \& \ \mathtt{f}_1 \geq 10 \vee \mathtt{f}_2 \geq 10) \ ; \{ \\ \quad (\mathtt{f}_1 := 0, \mathtt{f}_2 := \mathtt{f}_2 + 1) \ \ +_{\mathtt{f}_1 \geq 10 \,\wedge\, \mathtt{f}_2 < 10} \ \ \{ \\ \quad\quad (\mathtt{f}_2 := 0, \mathtt{f}_1 := \mathtt{f}_1 + 1) \ \ +_{\mathtt{f}_2 \geq 10 \,\wedge\, \mathtt{f}_1 < 10} \ \ (\mathtt{f}_1 := 0, \mathtt{f}_2 := 0) \ \} \\ \} \end{array}$$

   Let us denote it by $\mathtt{f}$. It models the internal clocks of two fireflies which are close to one another. The program $(\mathtt{f}_1 := 8, \mathtt{f}_2 := 5) \ ; \mathtt{f}^4$, for example, yields the plot below.



The plot discloses the existence of cyclic behaviour in the biological system: the evolution in the interval $[2, 4]$ is equal to the evolution in the interval $[11, 13]$, which means that the whole behaviour of the system can be unfolded by analysing the first eleven seconds.

Remark 3.4.4. Using conditional Kleisli representations, one can systematically endow programming languages with if-then-else statements. Another approach for this is to combine the hybrid monad with the maybe using the distributive law $\mathrm{MH} \to \mathrm{HM}$ described in Theorem 3.1.11. This gives rise to the partial hybrid monad HM which inherits useful properties from M: (i) it has a natural transformation $1 \to \mathrm{HM}$ that picks the evolution with duration zero and constant on failure – thus the partial hybrid paradigm supports abort operations and tests. (ii) It has a natural transformation $\mathrm{HM} \times \mathrm{HM} \to \mathrm{HM}$ defined at each set X by,

$$((f, d), (g, e)) \mapsto (\alpha_X \cdot \langle f, g \rangle, \max(d, e))$$

with $\alpha : \mathrm{M} \times \mathrm{M} \to \mathrm{M}$, $\alpha_X(*, *) = \alpha_X(x, y) = *$ and $\alpha_X(*, x) = \alpha(x, *) = x$ – together with the transformation $1 \to \mathrm{HM}$, it brings if-then-else constructs to the partial hybrid paradigm. (iii) These results emerge without the need for conditional Kleisli representations.

So why did we chose the latter approach, the reader might ask. The reason is that HM adds a new layer of complexity to the hybrid paradigm, and this makes difficult to accomplish the current chapter's goal, namely to examine purely hybrid computations. Our main topic for future work, however, lies precisely in the possible combinations of the hybrid monad with other ones (*e. g.* maybe, powerset, distribution . . .) and on the investigation of the resulting programming paradigms.

3.4.2 *Feedback and while loops*

We will now analyse the existence of feedback operators in hybrid programming. In the monadic setting, feedback operators on programs are usually obtained by endowing the hom sets $\mathrm{End}_{\mathbb{T}}(X)$ of a monad $\mathbb{T}$ with a suitable order [Koz97; GS13; HK15]: intuitively, for a map $f : X \to TX$, its feedback $f^* : X \to TX$ is defined by,

$$f^* = \bigvee \{\eta_X,\ f,\ f^2,\ f^3, \dots\}$$

which requires certain completeness properties on the set of endomorphisms $\mathrm{End}_{\mathbb{T}}(X)$. The widely famous *Kleene star* [Koz97] is one feedback operator that arises in this way: the complete partial order $(\mathrm{P}X, \subseteq)$ induces an order on $\mathrm{End}_{\mathrm{P}}(X)$ via pointwise extension, and the Kleene star of a program $f : X \to \mathrm{P}X$ is given by,

$$f^*(x) = \bigcup \{\eta_X(x),\ f(x),\ f^2(x),\ f^3(x), \dots\}$$

Our approach for analysing feedback operators in hybrid programming will follow a path similar to this case; to start we introduce a natural order on the sets $\mathrm{H}X$.

Definition 3.4.5. Consider a set X and two evolutions $(f,d),(g,e) \in \mathrm{H}X$. Then define, $(f,d) \sqsubseteq (g,e)$ if $d \leq e$ and the diagram below commutes.

$$\begin{tikzcd}
{[0,d]} \arrow[r, hook] \arrow[dr, "f"'] & {[0,e]} \arrow[d, "g"] \\
& X
\end{tikzcd}$$

Consider a unit-action $f : X \to \mathrm{H}X$ defined as $f(x) = (\underline{x}, 1)$. It is straightforward to show that each $x \in X$ induces a chain,

$$\eta_X(x) \sqsubseteq f(x) \sqsubseteq f^2(x) \sqsubseteq \dots$$

This chain, however, does not have a colimit, otherwise there would exist evolutions with infinite duration in $\mathrm{H}X$. To fix this issue we need to extend the hybrid monad with precisely this type of evolution.

Definition 3.4.6. Let $\mathrm{H}_* : \mathsf{Set} \to \mathsf{Set}$ be the coproduct of functors $\mathrm{H} + \hom(\mathbb{R}_{\geq 0}, -)$ and $\eta : \mathrm{Id} \to \mathrm{H}_*$ be the composition,

$$\begin{tikzcd}
\mathrm{Id} \arrow[r, "\eta^{\mathrm{H}}"] & \mathrm{H} \arrow[r, tail] & \mathrm{H}_*
\end{tikzcd}$$

Let also $\theta : \mathrm{H}_* \to \mathrm{Id}$ be the natural transformation that sends an evolution (f,d) to $f(0)$. Then observe that the operation for concatenating evolutions (see Section 3.2) can be straightforwardly extended to a natural transformation of the type $\mathrm{H} \times \mathrm{H}_* \to \mathrm{H}_*$. This gives rise to the natural transformation $\mu_1 : \mathrm{H}\mathrm{H}_* \to \mathrm{H}_*$ that is defined as the composition,

$$\begin{tikzcd}
\mathrm{H}\mathrm{H}_* \arrow[r, "\langle \mathrm{H}\theta, \lambda_{\mathrm{H}_*} \rangle"] & \mathrm{H} \times \mathrm{H}_* \arrow[r, "(+\!\!+)"] & \mathrm{H}_*
\end{tikzcd}$$

Definition 3.4.7. Let I denote the hom functor $\hom(\mathbb{R}_{\geq 0}, -) : \mathsf{Set} \to \mathsf{Set}$ and $\mu_2 : \mathrm{I}\mathrm{H}_* \to \mathrm{H}_*$ be the natural transformation defined at each set X by,

$$\mu_{2X}(f) = (\theta_X \cdot f, \infty)$$

It is natural because it can be written as the composition,

$$\begin{tikzcd}
\mathrm{I}\mathrm{H}_* \arrow[r, "\simeq"] & \hom(\mathbb{R}_{\geq 0}, \mathrm{H}_*) \arrow[r, "\theta_*"] & \hom(\mathbb{R}_{\geq 0}, -) \arrow[r, tail] & \mathrm{H}_*
\end{tikzcd}$$

Finally,

Definition 3.4.8. Define $\mu : \mathrm{H}_*\mathrm{H}_* \to \mathrm{H}_*$ as the mediating map in the diagram below.

$$\begin{tikzcd}
\mathrm{H}\mathrm{H}_* \arrow[r, tail] \arrow[dr, "\mu_1"'] & \mathrm{H}_*\mathrm{H}_* \arrow[d, dotted] & \mathrm{I}\mathrm{H}_* \arrow[l, tail] \arrow[dl, "\mu_2"] \\
& \mathrm{H}_* &
\end{tikzcd}$$

Theorem 3.4.9. *The triple* $(\mathrm{H}_*, \eta, \mu)$ *forms a monad.*

Proof. In Appendix A. □

The Kleisli composition of $\mathbb{H}_*$ behaves exactly like the Kleisli composition of $\mathbb{H}$ if restricted to evolutions with finite duration. In order to analyse what happens to those with infinite duration, consider two maps $f : X \to \mathrm{H}_*Y$, $g : Y \to \mathrm{H}_*Z$ and an element $x \in X$ such that $f(x) = (h, \infty)$. According to the multiplication of $\mathbb{H}_*$, the equation,

$$g \bullet f(x) = (\theta_Z \cdot g \cdot h, \infty)$$

holds. If g is a unit-action (Definition 3.2.8) the equation is simplified into $g \bullet f(x) = f(x)$. This means that g never executes if it is the solution of a system of ordinary differential equations, since then it needs to wait for the evolution $f(x)$ to finish which will clearly never happen.

It is easy to show that $\mathbb{H}$ is a submonad of $\mathbb{H}_*$, because the latter behaves exactly in the same way than the former when only finite evolutions are considered. This fact provides an inclusion functor $\mathsf{Set}_{\mathbb{H}} \rightarrowtail \mathsf{Set}_{\mathbb{H}_*}$ which for every set X gives rise to a monoid monomorphism,

$$\big(\mathrm{End}_{\mathrm{H}}(X), \bullet^{\mathrm{H}}, \eta^{\mathrm{H}}_X\big) \rightarrowtail \big(\mathrm{End}_{\mathrm{H}_*}(X), \bullet^{\mathrm{H}_*}, \eta^{\mathrm{H}_*}_X\big)$$

Consequently, the hybrid programming languages that were derived in the previous sections can also be seen as (conditional) Kleisli H_*-representations. Adding to this, the extended hybrid monad $\mathbb{H}_*$ has better completeness properties than the hybrid one, as attested by the theorem below.

Theorem 3.4.10. *Consider a set X and let $(\mathrm{H}_*X, \sqsubseteq)$ be the partially ordered set that canonically extends $(\mathrm{H}X, \sqsubseteq)$. Every directed subset $(f_i, d_i)_{i \in I}$ of H_*X with $\bigvee_{i\in I} d_i = \infty$ has a colimit.*

Proof. In order to construct the colimit of $(f_i, d_i)_{i\in I}$, observe that for every $a \in \mathbb{R}_{\geq 0}$ the set,

$$\{f_i(a) \mid i \in I \text{ such that } a \leq d_i\}$$

must be a singleton $\{x_a\}$, otherwise there would exist two elements $i, j \in I$ such that $f_i(a) \neq f_j(a)$ which violates the directedness property. So consider the function (f, ∞) defined by $f(a) = x_a$ for every $a \in \mathbb{R}_{\geq 0}$.

It follows from the definition of f that $(f_i, d_i) \sqsubseteq (f, \infty)$ for all $i \in I$. Every other function (g, ∞) that respects this property must also satisfy the equation $g(a) = f_i(a) = f(a)$ for every $i \in I$ with $a \leq d_i$ and $a \in [0, d_i]$. Since $\bigvee_{i\in I} d_i = \infty$, the equation $f = g$ holds and the condition $(f, \infty) \sqsubseteq (g, \infty)$ arises. □

Remark 3.4.11. The condition $\bigvee_{i\in I} d_i = \infty$ is essential for the previous theorem. Consider, for example, the chain,

$$(\underline{x}, 1) \sqsubseteq (\underline{x}, 1 + {}^1\!/\!_2) \sqsubseteq (\underline{x}, 1 + {}^1\!/\!_2 + {}^1\!/\!_4) \sqsubseteq \dots$$

Its supremum needs to have $[0, 2]$ as the duration and moreover it must output x for every element $a \in [0, 2)$. However, the set of all such functions in H_*X does not have a smallest element which forbids the existence of a colimit for the chain.

Hybrid systems whose executions yield this sort of chain are traditionally called Zeno and are known to be quite thorny [Zha+01; HM11]. We will address them more throughly later on.

Consider a diagram functor $\mathscr{D} : (\mathbb{N}_0, \leq) \to (\mathrm{H}_* X, \sqsubseteq)$. According to Theorem 3.4.10, it has a colimit if the equation,

$$\mathrm{colim}\ \big(\pi_2 \cdot \mathscr{D} : (\mathbb{N}_0, \leq) \to ([0,\infty], \leq)\big) = \infty \tag{7}$$

holds. If the $\mathscr{D}$-image has a final object, *i. e.* if there exists a natural number $i \in \mathbb{N}_0$ such that the condition,

$$\mathscr{D}(j) \sqsubseteq \mathscr{D}(i) \qquad (j \in \mathbb{N}_0) \tag{8}$$

holds then by definition $\mathscr{D}$ has a colimit. Finally, observe that if $\pi_2 \cdot \mathscr{D}$ has a final object then Condition (8) necessarily holds. These conditions will help us build a feedback map for $f : X \to \mathrm{H}_* X$ and are illustrated in the examples below.

Definition 3.4.12. Consider a unit-action $f : X \to \mathrm{H}_* X$ such that for every element $x \in X$ the diagram functor,

$$f^{(-)}(x) : (\mathbb{N}_0, \leq) \to (\mathrm{H}_* X, \sqsubseteq)$$

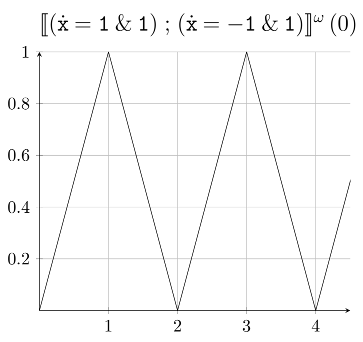
with $f^0 = \eta_X$ and $f^{n+1} = f^n \bullet f$ satisfies either Condition (7) or Condition (8). Then f *supports a feedback map* $f^\omega : X \to \mathrm{H}_* X$ which is given by,

$$f^\omega(x) = \mathrm{colim}\ f^{(-)}(x)$$

Intuitively, it corresponds to the *infinite* composition of f with itself.

Examples 3.4.13. Let us explore this operator using the time-triggered programming language of Examples 3.2.10.

1. Consider the composition $(\dot{\mathtt{x}} = 1 \ \&\ 1) \ ; \ (\dot{\mathtt{x}} = -1 \ \&\ 1)$ and observe that its interpretation always satisfies Condition (7) since it adds two time units at each iteration. For each $r \in \mathbb{R}$, the corresponding feedback map returns a triangular wave of infinite duration, as illustrated below.



2. Consider now the program $(\dot{\mathsf{x}} = -1 \mathbin{\&} 1) \; +_{\mathsf{x}>0} \; \mathsf{skip}$. It is straightforward to show that its interpretation always satisfies Condition (8), since after a finite number of iterations only skip will execute. Thus, a feedback map exists; and essentially it continuously decreases a given real number through successive iterations until it becomes negative.

   This example directs us to while loops in hybrid programming: for every program p that supports feedback one can define,

   $$[\![\mathsf{while}\ \psi\ \mathsf{do}\ \mathsf{p}]\!] = [\![\mathsf{p}\ +_{\psi}\ \mathsf{skip}]\!]^{\omega}$$

   which unfolds into a finite iteration of p if ψ is falsified at the end of some iteration, and to an infinite iteration of p otherwise. Note that these while loops do not behave exactly as the ones in classical programming. In fact, they differ in a fundamental aspect: in the hybrid case, a terminating while loop does not output a single point but a whole evolution and only the last point of this evolution necessarily satisfies the negation of ψ. One prime example of this aspect is the program,

   $$\mathsf{while}\ \ \mathsf{x} > 0\ \ \mathsf{do}\ (\dot{\mathsf{x}} = -1 \mathbin{\&} 1)$$

   which was presented above.

3. Recall that the movement of a vehicle can be described by the systems of equations,

   $$\dot{p} = v, \dot{v} = a$$

   where a denotes an acceleration chosen by a digital controller. Consider a positive acceleration $\mathsf{a} > 0$, a negative one $\mathsf{b} < 0$, and a desired speed s. The interpretation of the program,

   $$(\dot{\mathsf{p}} = \mathsf{v}, \dot{\mathsf{v}} = \mathsf{a} \mathbin{\&} 1) \;\; +_{\mathsf{v} \leq \mathsf{s}} \;\; (\dot{\mathsf{p}} = \mathsf{v}, \dot{\mathsf{v}} = \mathsf{b} \mathbin{\&} 1)$$

   clearly supports feedback since it adds one time unit at each iteration and thus satisfies Condition (7). In this case, the feedback map can be seen as a (simplistic) cruise controller that tries to maintain a desired speed (s).

In the sequel, we will examine some theoretical properties of the feedback operator $(-)^{\omega}$ (Definition 3.4.12). The main result is that the feedback map f^{ω} can be characterised as the smallest fixed point of $(- \;\; \bullet \; f) : \mathrm{End}_{\mathrm{H}_*}(X) \to \mathrm{End}_{\mathrm{H}_*}(X)$.

We start with the following observation which provides a natural order to $\mathrm{End}_{\mathrm{H}_*}(X)$: the ordered set $(\mathrm{H}_* X, \sqsubseteq)$ induces the product $\prod_X (\mathrm{H}_* X, \sqsubseteq)$ and since there exists a bijection,

$$\prod_X \mathrm{H}_*(X) \simeq \mathrm{End}_{\mathrm{H}_*}(X)$$

the set of endomorphisms $\mathrm{End}_{\mathrm{H}_*}(X)$ automatically inherits an order: explicitly, $f \sqsubseteq g$ iff $f(x) \sqsubseteq g(x)$ for every $x \in X$. The order on $\mathrm{End}_{\mathrm{H}_*}(X)$ arises canonically from $(\mathrm{H}_* X, \sqsubseteq)$, and, among other things, this allows to prove the following theorem quite easily.

Theorem 3.4.14. *Let $f : X \to \mathrm{H}_*X$ be a map that supports feedback and consider the diagram functor,*

$$f^{(\,-\,)} : (\mathbb{N}_0, \leq) \to (\mathrm{End}_{\mathrm{H}_*}(X), \sqsubseteq)$$

The feedback of f is the colimit of this functor.

Proof. We just need to observe that the diagram below commutes, recall Definition 3.4.12 and that isomorphisms preserve colimits.

$$\begin{tikzcd}
(\mathbb{N}_0, \leq) \arrow[r, "f^{(\,-\,)}"] \arrow[dr, bend right, "\langle f^{(\,-\,)}(x) \rangle_{x \in X}"'] & (\mathrm{End}_{\mathrm{H}_*}(X), \sqsubseteq) \arrow[d, tail, two heads, "\langle \pi_x \rangle_{x \in X}"] \\
& \prod_X (\mathrm{H}_*(X), \sqsubseteq)
\end{tikzcd}$$

□

The following two results show that the map $(- \bullet f) : \mathrm{End}_{\mathrm{H}_*}(X) \to \mathrm{End}_{\mathrm{H}_*}(X)$ is monotone and that it preserves the colimit of the chain,

$$f^{(\,-\,)} : (\mathbb{N}_0, \leq) \to (\mathrm{End}_{\mathrm{H}_*} X, \sqsubseteq)$$

These are two useful points for showing that f^ω is a smallest fixed point.

Lemma 3.4.15. *Every map $f : X \to \mathrm{H}_*X$ induces a functor,*

$$(- \bullet f) : (\mathrm{End}_{\mathrm{H}_*}(X), \sqsubseteq) \to (\mathrm{End}_{\mathrm{H}_*}(X), \sqsubseteq)$$

Proof. We need to show that for every two elements $a \sqsubseteq b \in \mathrm{End}_{\mathrm{H}_*}(X)$ the condition $a \bullet f \sqsubseteq b \bullet f$ holds. Observe that $a \sqsubseteq b$ entails $\theta_X \cdot a \sqsubseteq \theta_X \cdot b$, and denote $f(x)$ by $(f(x, -), d)$. If the latter is infinite,

$$\begin{aligned}
a \bullet f(x) &\sqsubseteq \theta_X \cdot a \cdot f(x, -) \\
&\sqsubseteq \theta_X \cdot b \cdot f(x, -) \\
&\sqsubseteq b \bullet f(x)
\end{aligned}$$

Otherwise,

$$\begin{aligned}
a \bullet f(x) &\sqsubseteq \theta_X \cdot a \cdot f(x, -) +\!\!+ a \cdot f(x, d) \\
&\sqsubseteq \theta_X \cdot b \cdot f(x, -) +\!\!+ a \cdot f(x, d) \\
&\sqsubseteq \theta_X \cdot b \cdot f(x, -) +\!\!+ b \cdot f(x, d) && (a \sqsubseteq b) \\
&\sqsubseteq b \bullet f(x)
\end{aligned}$$

□

Theorem 3.4.16. *The functor* $(- \bullet f) : (\mathrm{End}_{\mathrm{H}_*}(X), \sqsubseteq) \to (\mathrm{End}_{\mathrm{H}_*}(X), \sqsubseteq)$ *preserves the colimit of the chain* $f^{(-)} : (\mathbb{N}_0, \leq) \to (\mathrm{End}_{\mathrm{H}_*}(X), \sqsubseteq)$.

Proof. In appendix A. □

Corollary 3.4.17. *Consider a map* $f : X \to \mathrm{H}_* X$ *with feedback* f^ω*. The latter is the smallest fixed point of,*

$$(- \ \bullet \ f) : (\mathrm{End}_{\mathrm{H}_*}(X), \sqsubseteq) \to (\mathrm{End}_{\mathrm{H}_*}(X), \sqsubseteq)$$

Proof. It follows from Theorem 3.4.16 that f^ω is a fixed point. Then by induction, we can show that any fixed point g is a cocone for the chain,

$$f^{(-)} : (\mathbb{N}, \leq) \to (\mathrm{End}_{\mathrm{H}_*}(X), \sqsubseteq)$$

since $\eta_X \sqsubseteq g$ and $f^n \sqsubseteq g$ entails $f^{n+1} \sqsubseteq g \ \bullet \ f \sqsubseteq g$. Finally, the feedback f^ω is a colimit for the chain which proves that $f^\omega \sqsubseteq g$. □

Remark 3.4.18. There exists no least/greatest fixed point for $(f \bullet -) : \mathrm{End}_{\mathrm{H}_*}(X) \to \mathrm{End}_{\mathrm{H}_*}(X)$. The reason is that every constant map $\underline{(\underline{x}, \infty)}$ yields,

$$f \ \bullet \ \underline{(\underline{x}, \infty)} = \underline{(\underline{x}, \infty)}$$

but there does not exist an element in $(\mathrm{End}_{\mathrm{H}_*}(X), \sqsubseteq)$ that is smaller/greater than all these maps.

3.4.3 *Feedback on complete metric spaces*

To conclude the section, we will briefly analyse the existence of feedback operators for Zeno systems. Aside from a few exceptions (*e. g.* [AAS05; Zha+01; HM11]), the latter have not been thoroughly addressed by the hybrid systems community. This is mainly due to two reasons: (*i*) they are not realisable by computational devices, and (*ii*) to analyse them is highly problematic. In order to further detail these aspects, consider the formal definition of Zeno system.

Definition 3.4.19. An endomap $f \in \mathrm{End}_{\mathrm{H}_*}(X)$ is called Zeno if there exists some element $x \in X$ such that the function,

$$f^{(-)} : \mathbb{N}_0 \to \mathrm{End}_{\mathrm{H}_*}(X)$$

yields a bounded family $\pi_2 \cdot f^{(-)}$ with no greatest element.

Thus, if an endomap is Zeno, Conditions (7) and (8) necessarily fail, and consequently the feedback operator $(-)^\omega$ cannot be used. As a further attestment to the obstinacy of Zeno devices, [Pla10, page 55] uses the Kleene star as an iteration operator for hybrid systems. However, any system that comes after a Zeno one will never be reached through a finite number of iterations, which means that the operator is not suitable for Zeno devices. Recall for example the bouncing ball of Examples 3.2.14,

$$(\dot{\mathtt{p}} = \mathtt{v}, \dot{\mathtt{v}} = \mathtt{g} \mathbin{\&} \mathtt{p} \leq 0 \land \mathtt{v} \leq 0) \;;\; (\mathtt{v} := \mathtt{v} \times -0.5)$$

It is Zeno because an infinite number of progressively smaller bounces occur in a finite amount of time. Now suppose that when the ball stops something or someone kicks it. Clearly, this action cannot be reached in a finite number of iterations. So whilst we really want a notion of *infinite composition* for Zeno devices, the case seems hopeless if one just relies on the notion of feedback $(-)^{\omega}$ introduced in Definition 3.4.12.

A way of solving this prickly problem (up to some extent) is to assume that the set H_*X is a complete metric space [Kel55; Gou13]: consider the following proposition [Gou13, page 41].

Proposition 3.4.20. *Let X be a set and $(M, d : M \times M \to [0, \infty])$ be a complete metric space*[6]*. The set of functions M^X can also be equipped with a complete metric defined by,*

$$d^*(f, g) = \sup_{x \in X} d\,(f(x), g(x))$$

Let M be a complete metric space and consider the set $[0, \infty]$ with the usual Euclidean metric. The binary product $M^{\mathbb{R}_{\geq 0}} \times [0, \infty]$ with the sup metric is also complete, and since there exists an inclusion,

$$\mathrm{H}_*M \hookrightarrow M^{\mathbb{R}_{\geq 0}} \times [0, \infty]$$

the set H_*M can be equipped with a complete metric as well. Finally,

Definition 3.4.21. Let M be a complete metric space and consider an endomap $f \in \mathrm{End}_{\mathrm{H}_*}(M)$ such that for every $x \in M$ the family $(f^i(x))_{i \in \mathbb{N}}$ forms a Cauchy sequence. Then define,

$$f^{\omega}(x) = \lim_{i \to \infty} f^i(x)$$

Example 3.4.22. We already saw that the bouncing ball is a Zeno system. It is modelled by the composition,

$$(\dot{\mathtt{p}} = \mathtt{v}, \dot{\mathtt{v}} = \mathtt{g} \mathbin{\&} \mathtt{p} \leq 0 \land \mathtt{v} \leq 0) \;;\; (\mathtt{v} := \mathtt{v} \times -0.5)$$

which we will abbreviate to $\mathtt{b}$. Let us (co)restrict its interpretation $[\![\mathtt{b}]\!] : \mathbb{R}^2 \to \mathrm{H}_*(\mathbb{R}^2)$ to the set $\mathbb{R}_{\geq 0} \times \mathbb{R}$, since the ball can never be below ground. The set $\mathbb{R}_{\geq 0} \times \mathbb{R}$ with the usual metric is complete and therefore the set $\mathrm{H}_*(\mathbb{R}_{\geq 0} \times \mathbb{R})$ can be equipped with a complete metric as well. Our next task is to show that every element $x \in \mathbb{R}_{\geq 0} \times \mathbb{R}$ induces a Cauchy sequence,

$$([\![\mathtt{b}]\!]^i(x))_{i \in \mathbb{N}}$$

6 The classical notion of a metric space forbids points to be at an infinite distance from each other. However, in the current work this situation is allowed because it eases different constructions of metric spaces whilst preserving important topological properties such as convergence and continuity (see [Gou13, page 19] for more details).

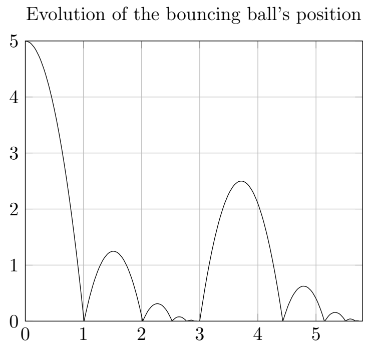
This is laborious but easy to show, because the distance between $[\![\mathsf{b}]\!]^i(x)$ and $[\![\mathsf{b}]\!]^{i+1}(x)$ is the height of the last jump in $[\![\mathsf{b}]\!]^{i+1}(x)$ and, therefore, as the index i increases the distance between $[\![\mathsf{b}]\!]^i(x)$ and $[\![\mathsf{b}]\!]^{i+1}(x)$ becomes progressively smaller by a factor of at least two. It is also easy to show that the magnitude of the ball's velocity becomes progressively smaller with each iteration by a factor of at least two. This means that the feedback $[\![\mathsf{b}]\!]^\omega$ of $[\![\mathsf{b}]\!]$ is well-defined. In particular, the composition,

$$[\![\mathsf{b}]\!]^\omega \bullet [\![\mathtt{p} := 5, \mathtt{v} := 0]\!]$$

encodes the action of letting the ball drop at, five meters and letting it bounce until it stops. Consider also the composition,

$$[\![\mathsf{b}]\!]^\omega \bullet [\![\mathtt{v} := 7]\!] \bullet [\![\mathsf{b}]\!]^\omega \bullet [\![\mathtt{p} := 5, \mathtt{v} := 0]\!]$$

It encodes the action of kicking the ball right after it stops, as shown in the plot below. This last case illustrates that, in contrast to the Kleene star operator [Pla10], feedback over complete metric spaces does not lead to unreachability problems.



3.5 OPEN CHALLENGES

In this chapter we introduced the hybrid monad and showed that it captures the basic interaction between programs and physical processes, initially pointed out in the sixties by Witsenhausen [Wit66] and further examined in the following decades by several people (*e. g.* [Tav87; Hen96; Höf09; SH11]). Using the framework of Kleisli representations and the hybrid monad, we generated different hybrid programming languages, listed all program operations available in this paradigm, and examined the possibility of adding classical programming constructs, such as if-then-else statements and while loops. In this process, several interesting questions emerged for which we still lack a definitive answer. We summarise them next.

The algebraic theory of the hybrid monad. As discussed in Section 3.1, every monad $\mathbb{T}$ induces a category $\mathsf{C}^{\mathbb{T}}$ of Eilenberg-Moore algebras, which in many cases is equivalent to a familiar category of algebras (see Examples 3.1.7). We also saw in Examples 3.1.15 that when building a programming language from a monad $\mathbb{T}$ it is often useful to have an affable description of the corresponding category of Eilenberg-Moore algebras. *Can we find one such description for the category of Eilenberg-Moore algebras of the hybrid monad ?* For now, we only have the following result: the coproduct of hom functors,

$$\mathrm{H}_{\mathbb{N}} = \coprod_{n\in\mathbb{N}} \hom(n+1, -\,) : \mathsf{Set} \to \mathsf{Set}$$

equipped with the operations of the hybrid monad is a monad as well. And the category of Eilenberg-Moore $\mathrm{H}_{\mathbb{N}}$-algebras is equivalent to the variety V generated by a single binary operation $(\cdot)$ that satisfies $(a\cdot b)\cdot c = a\cdot c$. Let us briefly see why. It is straightforward to prove the existence of a free-forgetful adjunction,

$$\begin{tikzcd}\mathsf{Set} \arrow[r, bend left, "\mathrm{F}"] \arrow[r, phantom, "\perp"] & \mathsf{V} \arrow[l, bend left, "\mathrm{U}"]\end{tikzcd}$$

such that the free algebras are the tuples $(\coprod_{n\in\mathbb{N}} \hom(n+1, X), *)$ with the binary operation defined by,

$$(x_1 \ldots x_n) * (y_1 \ldots y_m) = x_1 y_1 \ldots y_m$$

The unit $\mathrm{Id} \to \mathrm{UF}$ of the adjunction is the unit of the hybrid monad, and the counit $\epsilon : \mathrm{FU} \to \mathrm{Id}$ is defined at each algebra $(X, \cdot)$ by the equations,

$$\epsilon_{(X,\cdot)}(x) = x \qquad\qquad \epsilon_{(X,\cdot)}(x_1 \ldots x_n) = x_1 \cdot \epsilon_{(X,\cdot)}(x_2 \ldots x_n)$$

It is then routine to prove that the triple $(\mathrm{UF}, \eta, \mathrm{U}\epsilon_{\mathrm{F}})$ is the monad $\mathrm{H}_{\mathbb{N}}$. An application of the following theorem shows that V is equivalent to the category of Eilenberg-Moore algebras of this monad [AHS09, Proposition 20.20].

Theorem 3.5.1. *Each finitary variety is a monadic construct.*

Stability. Despite being rarely a concern in the programming languages field, the notion of stability is a crucial aspect of control theory. *Can the programming languages generated by the hybrid monad be endowed with a suitable notion of stability, allowing us to reason about the stability of a program in a compositional manner ?* A quick analysis of this question yields a negative answer: the hybrid monad is defined on Set which lacks sufficient structure for defining a suitable notion of stability. However, in Chapter 5 we prove that the hybrid monad also lives in the category of topological spaces, which provides a more suitable setting to address this notion. For example, we will see that a notion of stability naturally emerges from the topological hybrid monad, and that the latter allows to generate programming languages whose operators are closed under this notion. Among other things, we use these results to prove that *all* programs built from the programming language of Examples 3.2.10 are stable.

Feedback. We introduced two types of iteration, but both behest strict conditions. *Can we relax them ? And which types of iteration does the hybrid monad actually supports ?* We still lack definitive answers for these questions, but it should be mentioned that this particular problem is being addressed in [Jak18].

Monad combinations. Having studied purely hybrid computations, a next natural step is to analyse possible combinations of the hybrid monad with other monads. *We already saw that there exists a distributive law* $\mathrm{MH} \to \mathrm{HM}$ *relating the maybe monad over the hybrid one; can we find other interesting cases ?* One positive answer is given by the following theorem.

Theorem 3.5.2. *Let* Q *be the non-empty powerset monad. There exists a distributive law* $\delta : \mathrm{HQ} \to \mathrm{QH}$ *defined at each set* X *by,*

$$\delta_X(f, d) = \{(g, d) \in \mathrm{H}X \mid g \in f\}$$

where $g \in f$ *is shorthand notation for the condition* $\forall t \in [0, d] \,.\, g(t) \in f(t)$.

Proof. In Appendix A[7]. □

The non-deterministic hybrid monad QH, induced by the distributive law above, generates hybrid programming languages with different types of interesting features, from non-deterministic assignments, to differential predicates and programs that *do not* terminate at precisely the prescribed time. Consider for example the following programming language: denote by $\mathsf{At}(X)$ the set given by the grammar,

$$\varphi \ni (x_1 := [t, t], \ldots, x_n := [t, t]) \mid (\dot{x}_1 = t, \ldots, \dot{x}_n = t \,\&\, d), \qquad t \ni r \mid r \cdot x \mid t + t$$

where d and r are real numbers and $x \in X$. It is straightforward to build a Kleisli QH-representation for the programming language below,

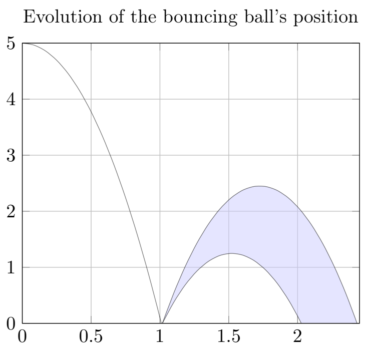
$$\mathsf{p} = \mathsf{a} \in \mathsf{At}(X) \mid \mathsf{skip} \mid \mathsf{p} \,;\, \mathsf{p} \mid \mathsf{p} + \mathsf{p} \mid \mathsf{p}^*$$

which includes differential equations, non-deterministic assignments, non-deterministic choice, and the Kleene star operator. Using this language we can model, for example, a non-deterministic bouncing ball,

$$\begin{aligned}&(\mathsf{p} := 5, \mathsf{v} := 0) \;\; ; \;\; (\dot{\mathsf{p}} = \mathsf{v}, \dot{\mathsf{v}} = \mathsf{g} \,\&\, \mathsf{p} \leq 0 \wedge \mathsf{v} \leq 0) \; ; \; \mathsf{v} := [\mathsf{v} \times -0.5, \mathsf{v} \times -0.7] \; ; \\ &(\dot{\mathsf{p}} = \mathsf{v}, \dot{\mathsf{v}} = \mathsf{g} \,\&\, \mathsf{p} \leq 0 \wedge \mathsf{v} \leq 0)\end{aligned}$$

whose projection on p yields the plot below.

7 It is laborious but easy to show that the analogous natural transformation $\delta : \mathrm{HP} \to \mathrm{PH}$ for the powerset monad P is not distributive.


Evolution of the bouncing ball's position
5
4
3
2
1
0
0
0.5
1
1.5
2

4 BLACK-BOX HYBRID PROGRAMMING

Overview. In this chapter, we develop the theory of hybrid programs with internal memory, which, as discussed in the introduction, form the basis of *component-based software development* in hybrid programming. These programs adopt a black-box, observational perspective in which digital computations are hidden from the environment, and influence physical processes, which are external and make up the observable behaviour. In this observational context, notions of bisimulation, behaviour, and regular expression become central elements of the game, and we provide them to these programs using standard results in coalgebra.

Along the process, we show that the coalgebraic approach permits to uniformly extend these results to hybrid programs with *different types of internal memory*, covering in particular hybrid automata – the standard formalism of hybrid systems – and a number of their variants. We explore this feature by developing a *uniform* framework of hybrid automata, which we use to prove that apparently different notions of bisimulation for these devices are instances of a single, coalgebraic definition, and that the same applies to their semantics.

Roadmap. We start by reviewing the basic notions of coalgebra (Section 4.1). Then, we introduce hybrid programs with internal memory and develop their coalgebraic theory (Section 4.2). After this, we take the generic view mentioned above: starting with a brief detour to hybrid automata's land – so that we can precisely relate them to this work and build useful intuitions (Section 4.3) – we develop a generic framework for hybrid programs with *different* types of internal memory, which covers syntax, semantics, and bisimulation (Section 4.4). We conclude with a discussion on several challenges that emerged from this research and that we think deserve further study, *e. g.* (generic) notions of approximate bisimulation, composition operators, and techniques for handling hybrid systems with infinite state spaces (Section 4.5).

4.1 PRELIMINARIES: BASIC NOTIONS OF COALGEBRA

The theory of coalgebras [Rut00; Adá05; Jac16] provides an abstract framework for state-based transition systems that allows to derive notions and results parametric on a transition type. Examples of this include generic definitions of bisimulation [Sok05], uniform trace semantics [HJS07], and a generalisation of Kleene's theorem to different families of transition systems [Sil10]. This high level of genericity is possible due to the interpretation of a transition type as a functor $F : \mathsf{C} \to \mathsf{C}$ – the corresponding family of transition systems is formed by the C-arrows of the type $X \to FX$, which are what we call F-*coalgebras* or, more simply, coalgebras.

Examples 4.1.1. Let A and B be arbitrary sets.

1. Coalgebras for the powerset functor, *i. e.* P-coalgebras, are Kripke frames [BBW06, Example 143].

2. Coalgebras for $(- \times A) : \mathsf{Set} \to \mathsf{Set}$ are *stream automata* [HKR17], which can be seen as *black-box machines* with two interfaces: a button and a display. When the button is pressed the machine *internally* jumps into a new state and sends a value of type A to the (*external*) display.

3. Coalgebras for $(- \times A)^B : \mathsf{Set} \to \mathsf{Set}$ are *Mealy machines* [Bar03; BRS09], which extend stream automata with an input dimension. In the black-box machine analogy, instead of having a single button the user now has a family of buttons indexed by the elements of B.

4. Coalgebras for $\mathrm{P}(- \times A) : \mathsf{Set} \to \mathsf{Set}$ are labelled transition systems, and coalgebras for $2 \times (-)^A : \mathsf{Set} \to \mathsf{Set}$ are deterministic automata [Rut00, pages 9 and 21].

5. Let Stone be the category of Stone spaces and continuous maps [Joh86]. Let also $\mathrm{V} : \mathsf{Stone} \to \mathsf{Stone}$ be the restriction of the compact Vietoris functor to Stone (see Examples 3.1.2 and [Mic51, Theorem 4.9.9]). Coalgebras for this functor are (up-to isomorphism) descriptive general frames [KKV04].

6. Let $\mathrm{D} : \mathsf{Set} \to \mathsf{Set}$ be the functor of distributions (see Examples 3.1.2). Coalgebras for it are simply discrete Markov chains [Sok05, page 77].

Coalgebras carry several useful constructs for state-based transition systems, from state minimisation and bisimulation techniques to notions of observable behaviour. In the current section we review some of these aspects, focusing specially on those with an important role in our work. Observe that,

Definition 4.1.2. Every functor $F : \mathsf{C} \to \mathsf{C}$ induces a category $\mathsf{CoAlg}\,(F)$ whose objects are F-coalgebras $(X, c : X \to FX)$ and morphisms,

$$f : (X, c : X \to FX) \to (Y, d : Y \to FY)$$

are C-arrows $f : X \to Y$ that make the diagram below commute.

$$\begin{tikzcd} X \arrow[r, "f"] \arrow[d, "c"'] & Y \arrow[d, "d"] \\ FX \arrow[r, "Ff"'] & FY \end{tikzcd}$$

For every functor $F : \mathsf{C} \to \mathsf{C}$ there exists a forgetful functor $\mathrm{U} : \mathsf{CoAlg}\,(F) \to \mathsf{C}$ that sends coalgebras to their carrier.

4.1.1 *Bisimulation and observable behaviour*

Since the notion of coalgebra serves as an abstract definition of transition system, it is perhaps not surprising that bisimulation takes a central place within coalgebraic theory. Its definition, parametric on a transition type and extensively studied in several papers (see [Sta11]), resorts to the notion of coalgebra homomorphism, and it is usually based on Set.

Definition 4.1.3. Consider a functor $F : \mathsf{Set} \to \mathsf{Set}$, two F-coalgebras (X, c), (Y, d), and a relation $R \subseteq X \times Y$. Then R is called an F-bisimulation (or simply, a bisimulation) if there exists a coalgebra (R, r) that makes the following diagram commute.

$$\begin{tikzcd}
X \arrow[d, "c"'] & R \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "r"'] & Y \arrow[d, "d"] \\
FX & FR \arrow[l, "F\pi_1"] \arrow[r, "F\pi_2"'] & FY
\end{tikzcd}$$

We say that two states $x \in X$ and $y \in Y$ are *bisimilar*, in symbols $x \sim y$, if they are related by some F-bisimulation. If the relation $R \subseteq X \times Y$ is both an equivalence and an F-bisimulation then we call it an *F-bisimulation equivalence.*

Examples 4.1.4. Let A and B be arbitrary sets.

1. Bisimulation for P-coalgebras is exactly bisimulation for Kripke frames [BBW06, Example 168].

2. Consider two $(- \times A)$-coalgebras $(X, \langle a, b \rangle)$ and $(Y, \langle c, d \rangle)$. A relation $R \subseteq X \times Y$ is a bisimulation if $x \, R \, y$ entails,

$$a(x) \, R \, c(y) \text{ and } b(x) = d(y)$$

   Recall the black-box machine of Examples 4.1.1 (2). Each state $x \in X$ (resp. $y \in Y$) corresponds to one such machine whose initial state is x (resp. y). Two states x and y are bisimilar if we cannot differentiate one another using just the interfaces (button and display) of the two associated machines.

3. Denote the right transpose of a function $f : A \times B \to C$ by $\overline{f} : A \to C^B$.

   Then, consider two $(- \times A)^B$-coalgebras $(X, \overline{\langle a, b \rangle})$ and $(Y, \overline{\langle c, d \rangle})$. A relation $R \subseteq X \times Y$ is a bisimulation if $x \, R \, y$ entails,

$$a(x, i) \, R \, c(y, i) \text{ and } b(x, i) = d(y, i) \qquad (i \in B)$$

4. A relation $R \subseteq X \times Y$ induces a relation $\asymp_R \subseteq \mathrm{D}X \times \mathrm{D}Y$ on distributions defined by $\mu_1 \asymp_R \mu_2$ iff there exists a distribution $\nu \in \mathrm{D}(X \times Y)$ such that,

$$\nu(x, y) > 0 \text{ entails } x \, R \, y, \qquad \nu(\{x\} \times Y) = \mu_1(x), \qquad \nu(X \times \{y\}) = \mu_2(y)$$

Consider two D-coalgebras (X, c), (Y, d) and a relation $R \subseteq X \times Y$. Then R is a bisimulation if $x \; R \; y$ entails $c(x) \asymp_R d(y)$. In other words, bisimulation for D-coalgebras is exactly bisimulation for discrete Markov chains [Sok05, page 90].

5. Consider an F-coalgebra homomorphism $f : (X, c) \to (Y, d)$. Its graph is a bisimulation for (X, c) and (Y, d) [Rut00, Theorem 2.5].

Another central concept in coalgebra is the notion of observable behaviour, which is given by *final coalgebras*.

Definition 4.1.5. Consider a functor $F : \mathsf{C} \to \mathsf{C}$ and an F-coalgebra (X, c). We call the latter a final coalgebra if for every F-coalgebra (Y, d) there exists a unique coalgebra homomorphism $[\![-]\!] : (Y, d) \to (X, c)$.

Intuitively, the final F-coalgebra collects in its carrier the behaviours of all F-coalgebras' states, and the universal maps associate each state of an F-coalgebra to its behaviour. For example,

Examples 4.1.6. Let A and B be arbitrary sets.

1. The final coalgebra of the category $\mathsf{CoAlg}\,(- \times A)$ is the map $\langle \mathrm{tl}, \mathrm{hd} \rangle : A^\omega \to A^\omega \times A$ induced by the 'tail' and 'head' functions where A^ω is the set of streams whose values are of type A.

   In the black-box machine analogy (see Examples 4.1.1 (2)), to press the button an infinite number of times can be seen as the computation of the machine's observable behaviour, which in this case is a stream of values of type A.

2. The final coalgebra of the category $\mathsf{CoAlg}\,(- + 1)$ is the 'predecessor' function,

$$p : \mathbb{N} \cup \{\infty\} \to (\mathbb{N} \cup \{\infty\}) + 1$$

   which maps every number greater than zero to its predecessor and to 'fail' otherwise [Rut00, Example 10.2]. The observable behaviour of a $(- + 1)$-coalgebra's state s can be intuitively seen as the number of times the coalgebra switches between states (starting in s) before it fails.

3. Let B^+ denote the set of non-empty lists whose values are of type B. The final coalgebra of the category $\mathsf{CoAlg}\left((- \times A)^B\right)$ is the function,

$$\overline{\langle \mathrm{apd}, \mathrm{sng} \rangle} : A^{B^+} \to (A^{B^+} \times A)^B$$

   where $\mathrm{apd}(f, b)(bs) = f(b : bs)$ and $\mathrm{sng}(f, b) = f[b]$ [Rut06].

4. The final coalgebra of the category $\mathsf{CoAlg}\left(A \times (-)^B\right)$ is the function,

$$\langle \overline{\mathrm{apd}}, \mathrm{emp} \rangle : A^{B^*} \to (A^{B^*})^B \times A$$

where $\mathrm{apd}(f, b)(bs) = f(b : bs)$ and $\mathrm{emp}(f) = f(\epsilon)$ [Rut00, Example 10.2].

Now recall from Examples 4.1.1 that $(2 \times (-)^B)$-coalgebras are deterministic automata. The set 2^{B^*} is the powerset of words over B, and in this context every state of a deterministic automaton is associated with the set of words that it accepts.

5. The final coalgebra for the finitary powerset functor $\mathrm{P}_\omega : \mathsf{Set} \to \mathsf{Set}$ is the set of finitely-branching trees modulo bisimilarity equipped with the function that maps a tree into its set of subtrees [Wor05].

To conclude this subsection, let us see why coalgebra homomorphisms are also called 'behaviour-preserving morphisms'.

Consider a functor $F : \mathsf{Set} \to \mathsf{Set}$ that preserves weak pullbacks and that admits a final coalgebra (ν_F, m). Consider also two F-coalgebras (X, c), (Y, d) and a bisimulation (R, r) for them. By uniqueness, the following diagram commutes.

$$\begin{tikzcd}
(R, r) \arrow[r, "\pi_2"] \arrow[d, "\pi_1"'] & (Y, d) \arrow[d, "\llbracket - \rrbracket"] \\
(X, c) \arrow[r, "\llbracket - \rrbracket"'] & (\nu_F, m)
\end{tikzcd}$$

Therefore, for every two bisimilar states $x \in X$ and $y \in Y$ the equation $\llbracket x \rrbracket = \llbracket y \rrbracket$ holds. Now recall from Examples 4.1.4 (5) that the graph of a coalgebra homomorphism $f : (X, c) \to (Y, d)$ is a bisimulation. This entails that $x \sim f(x)$ and therefore $\llbracket x \rrbracket = \llbracket f(x) \rrbracket$.

Remark 4.1.7. It is also true that $\llbracket x \rrbracket = \llbracket y \rrbracket$ entails $x \sim y$. Consider the pullback,

$$\begin{tikzcd}
P \arrow[r, "\pi_2"] \arrow[d, "\pi_1"'] & Y \arrow[d, "\llbracket - \rrbracket"] \\
X \arrow[r, "\llbracket - \rrbracket"'] & \nu_F
\end{tikzcd}$$

and then use the weak pullback preserving property of $F : \mathsf{Set} \to \mathsf{Set}$ to obtain suitable a morphism $P \to FP$.

4.1.2 *Colimits, factorisation structures, and subcoalgebras*

Dually to categories of algebras, categories of coalgebras are very well-behaved with respect to colimits. In particular, every forgetful functor,

$$\mathsf{CoAlg}\,(F) \to \mathsf{C}$$

creates colimits [Che13, Proposition 4.1.3]. Coequalisers of coalgebras, for example, are obtained in the following way.

Definition 4.1.8. Assume that C has coequalisers. Consider two F-coalgebras (X, c) and (Y, d) and a pair of coalgebra homomorphisms,

$$\begin{tikzcd}
(X, c) \arrow[r, shift left, "f"] \arrow[r, shift right, "g"'] & (Y, d)
\end{tikzcd}$$

The latter induce a coalgebra whose carrier is the C-coequaliser Q of the two morphisms $f, g : X \to Y$ and the arrow $Q \to FQ$ is given by the commuting diagram below.

$$\begin{tikzcd}
X \arrow[r, shift left, "f"] \arrow[r, shift right, "g"'] \arrow[d, "c"'] & Y \arrow[r, two heads, "q"] \arrow[d, "d"'] & Q \arrow[d, dotted, "m"] \\
FX \arrow[r, shift left, "Ff"] \arrow[r, shift right, "Fg"'] & FY \arrow[r, "Fq"'] & FQ
\end{tikzcd}$$

The coalgebra (Q, m) is the coequaliser of the morphisms $f, g : (X, c) \to (Y, d)$.

The quest for a suitable notion of subcoalgebra, required later in the thesis, justifies the following incursion on *factorisation structure for morphisms* [AHS09, Section 14].

Definition 4.1.9. Consider two classes E and M of morphisms in a category C. We say that C has an (E, M)-factorisation structure if,

1. every morphism in E or in M is closed under composition with isomorphisms,
2. the category C has (E, M)-factorisations, *i. e.* every C-morphism can be written as a composition $e \cdot m$ such that $e \in E$ and $m \in M$,
3. the category has the unique fill-in property, *i. e.* for every diagram,

$$\begin{tikzcd}
\bullet \arrow[r, "e"] \arrow[d, "f"'] & \bullet \arrow[d, "g"] \arrow[ld, dotted, "d"'] \\
\bullet \arrow[r, "m"'] & \bullet
\end{tikzcd}$$

such that $e \in E$, $m \in M$ and the outer square commutes, there exists a unique morphism d that makes the diagram commutative.

Example 4.1.10. The category Set has an (Epi,Mono)-factorisation structure and the category Top has an (Epi,Subspace)-factorisation structure [AHS09, Examples 14.2].

Categories of coalgebras frequently inherit the factorisation structure of their underlying categories [Che13, Proposition 4.1.15].

Theorem 4.1.11. *Consider a category C with an (E, M)-factorisation structure and a functor $F : \mathsf{C} \to \mathsf{C}$ that preserves M-morphisms. The category $\mathsf{CoAlg}\,(F)$ has an $(\mathrm{U}^{-1}E, \mathrm{U}^{-1}M)$-factorisation structure.*

Via an orchestrated use of factorisation structures and the results below, we can obtain a generic notion of subcoalgebra that is closed under unions and intersections.

Definition 4.1.12. Consider a category C. Let M be a class of C-monomorphisms and X be a C-object. An *M-subobject* of X is a pair $(S, m : S \rightarrowtail X)$ such that S is a C-object and $m : S \rightarrowtail X$ is an M-morphism.

The class of M-subobjects of X forms a preorder category $\mathsf{Sub}(X)$ defined by,

$$S_1 \overset{m_1}{\rightarrowtail} X \leq S_2 \overset{m_2}{\rightarrowtail} X$$

iff there exists a C-morphism $h : S_1 \to S_2$ that makes the diagram below commute.

$$\begin{tikzcd} S_1 \arrow[r, "h"] \arrow[dr, tail, "m_1"'] & S_2 \arrow[d, tail, "m_2"] \\ & X \end{tikzcd}$$

Document [Bor94a, Proposition 4.2.6] tells that the category $\mathsf{Sub}(X)$ is cocomplete (*i. e.* it has unions of M-subobjects) if C has a suitable factorisation structure and coproducts. More concretely,

Theorem 4.1.13. *Let C be a category with an (E, M)-factorisation structure such that $M \subseteq$ Mono(C). Suppose also that it has coproducts. Then for every C-object X, the category $\mathsf{Sub}(X)$ of M-subobjects of X is cocomplete.*

Proof. Consider a family of M-subobjects $(m_i : S_i \rightarrowtail X)_{i \in I}$. Since C has coproducts, one may assume the existence of a C-morphism,

$$[m_i]_{i \in I} : \coprod_{i \in I} S_i \to X$$

which can then be factorised as shown in the diagram below.

$$\begin{tikzcd} \coprod_{i \in I} S_i \arrow[r, two heads, "e"] \arrow[rr, bend left, "{[m_i]_{i \in I}}"] & Y \arrow[r, tail, "m"] & X \\ S_i \arrow[u, "\iota_i"] \arrow[urr, bend right, "m_i"'] & & \end{tikzcd}$$

Clearly Y is an M-subobject of X. Moreover the condition,

$$S_i \overset{m_i}{\rightarrowtail} X \leq Y \overset{m}{\rightarrowtail} X \qquad (i \in I) \tag{9}$$

holds. Finally, take an M-subobject $Z \rightarrowtail X$ of X that also satisfies Condition (9). We need to show that,

$$Y \overset{m}{\rightarrowtail} X \leq Z \rightarrowtail X$$

which follows directly from the diagonal fill-in property,

$$\begin{tikzcd}
\coprod_{i\in I} S_i \arrow[r, two heads] \arrow[d] & Y \arrow[d, tail] \arrow[dl, dotted] \\
Z \arrow[r, tail] & X
\end{tikzcd}$$

□

The category $\mathsf{Sub}(X)$ is, therefore, cocomplete if some mild conditions are satisfied. The following definition [AHS09, Definition 7.82] will help us show that frequently $\mathsf{Sub}(X)$ is also complete, *i. e.* it also has intersections of M-subobjects.

Definition 4.1.14. Recall that for a C-object X, $\mathsf{C}/_X$ denotes the slice category over X. Let M be a class of C-monomorphisms and note that each C-object X induces the full subcategory of $\mathsf{C}/_X$ whose objects are morphisms in M. We say that the category C is *M-wellpowered* if for every C-object X the corresponding category $\mathsf{Sub}(X)$ is equivalent to a small category (in other words, it is essentially small).

If C is M-wellpowered and $\mathsf{Sub}(X)$ is cocomplete $\mathsf{Sub}(X)$ is also complete. This is due to the following theorem, which tells that 'cocompleteness almost implies completeness' [AHS09, Theorem 12.7].

Theorem 4.1.15. *A small category is cocomplete iff it is complete.*

Corollary 4.1.16. *Assume the conditions of Theorem 4.1.13 and let the category C be M-wellpowered. Consider a C-object X and a family of M-subobjects $(m_i : S_i \rightarrowtail X)_{i\in I}$. The intersection $\bigwedge_{i\in I} S_i$ is given by the union,*

$$\bigvee\{Y \mid Y \textit{ is an } M\textit{-subobject of } S_i \textit{ for all } i \in I\}$$

Let us instantiate these results in categories of coalgebras.

Consider a category C with an $(E,\ M)$-factorisation structure such that $M \subseteq \mathrm{Mono}(\mathsf{C})$. The functor $\mathrm{U} : \mathsf{CoAlg}\,(F) \to \mathsf{C}$ is faithful, and, therefore, the inequation $\mathrm{U}^{-1}M \subseteq \mathrm{Mono}(\mathsf{CoAlg}\,(F))$ holds. Via Definition 4.1.12, we obtain a notion of $\mathrm{U}^{-1}M$-subobject in $\mathsf{CoAlg}\,(F)$, which is equivalent to the following notion of subcoalgebra.

Definition 4.1.17. Let C be a category with an (E,M)-factorisation structure such that $M \subseteq \mathrm{Mono}(\mathsf{C})$, and let $F : \mathsf{C} \to \mathsf{C}$ be a functor that preserves M-morphisms. Consider also two F-coalgebras (X, c) and (Y, d). Then (X, c) is an *M-subcoalgebra* of (Y, d) if there exists a coalgebra homomorphism $(X, c) \rightarrowtail (Y, d)$ whose U-image is in M.

An application of Theorem 4.1.11 tells that $\mathsf{CoAlg}\,(F)$ has an $(\mathrm{U}^{-1}E, \mathrm{U}^{-1}M)$-factorisation structure. So if the previous definition assumes that C is cocomplete, M-subcoalgebras become closed under unions (Theorem 4.1.13). According to the following theorem and Corollary 4.1.16, intersections of these objects can also be obtained by forcing C to be M-wellpowered.

Theorem 4.1.18. *Let* C *be a category with an* (E, M)*-factorisation structure. If* C *is* M*-wellpowered,* $\mathsf{CoAlg}\,(F)$ *is* $\mathrm{U}^{-1}M$*-wellpowered as well.*

Proof. For each F-coalgebra (X, c), take the set of $\mathrm{U}^{-1}M$-subobjects of (X, c) whose carrier is in the set of M-subobjects of X. The claim then follows straightforwardly, because for each M-subobject S of X there is only a set of coalgebras $\hom(S, FS)$. □

Examples 4.1.19. In the thesis we will consider coalgebras over different categories, two prime examples being Set and Top. Let us see how the previous results on subcoalgebras are instantiated to these two cases.

1. It is well-known that the category Set has an (Epi,Mono)-factorisation structure, and that it is Mono-wellpowered [AHS09, Theorem 7.88]. It is also cocomplete and most functors $F : \mathsf{Set} \to \mathsf{Set}$ preserves monomorphisms [Rut00, Proposition A.1]. Then, for an F-coalgebra (X, c), the category of Mono-subcoalgebras of (X, c) is essentially the set of all F-coalgebras (S, a) such that $S \subseteq X$ and the inclusion $S \hookrightarrow X$ is a coalgebra homomorphism $(S, a) \hookrightarrow (X, c)$. The union of subcoalgebras $\vee_{i \in I}(S_i, a_i)$ is the coalgebra that is obtained from the (E, M)-factorisation of the coalgebra homomorphism $\coprod_{i \in I}(S_i, a_i) \to (X, c)$. The meet $\wedge_{i \in I}(S_i, a_i)$ is simply the biggest Mono-subcoalgebra of (X, c) that is smaller than all subcoalgebras in $(S_i, a_i)_{i \in I}$.

2. Recall that a monomorphism is called regular if it is the equaliser of two morphisms. Recall also that the category Top has an (Epi, RegMono)-factorisation structure and that it is RegMono-wellpowered [AHS09, Theorem 7.88]. It is also cocomplete and most functors $F : \mathsf{Top} \to \mathsf{Top}$ that we will consider preserve regular monomorphisms. Now, for an F-coalgebra (X, c) the category of regular subcoalgebras of (X, c) is essentially the set of all F-coalgebras (S, a) such that S is a subspace of X and the inclusion map $S \hookrightarrow X$ is a coalgebra homomorphism $(S, a) \hookrightarrow (X, c)$. In the case of F preserving regular monomorphisms, the join of subcoalgebras $\vee_{i \in I}(S_i, a_i)$ is constructed as in Set with the difference that the carrier set is equipped with the subspace topology. The meet of regular subcoalgebras follows as expected.

4.2 HYBRID PROGRAMS WITH INTERNAL MEMORY

4.2.1 *A change of perspectives in hybrid programming*

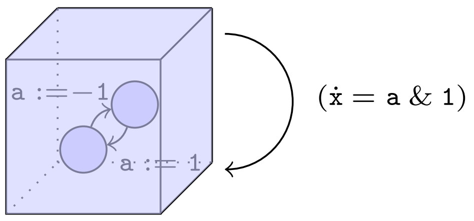
In Chapter 3 we put the spotlight on programming languages that treat assignments and differential equations *in the same manner*, effectively mixing discrete and continuous behaviour. In this chapter, we adopt and explore a complementary perspective for developing hybrid programs with internal memory.

This new perspective brings a *black-box paradigm* to hybrid programming by making a conceptual distinction between discrete computations and continuous, physical processes. In detail, discrete assignments are internal, *hidden* from the environment, whilst continuous evolutions are external, making up the observable behaviour. Think for example of a cruise control system; one cannot directly observe the computations of the digital device, only their effect over the car's velocity and movement which evolve over time. An analogous pattern occurs *e. g.* in thermostats, pacemakers, and water level regulators.

Let us illustrate the black-box paradigm with a very simple case. In Examples 3.4.13, we examined an oscillator defined by the composition,

$$((\dot{\mathtt{x}} = \mathtt{1} \,\&\, \mathtt{1}) \,;\, (\dot{\mathtt{x}} = -\mathtt{1} \,\&\, \mathtt{1}))^{\omega}$$

In the black-box perspective, we see it as the machine below,



which already gives some intuitions on what we mean by hybrid programs with internal memory. They have an internal automaton which discretely changes the computer's memory when certain events occur, and for each mode there exists an observable continuous dynamics.

Remark 4.2.1. Another important advantage of this perspective is that, since discrete devices and physical processes are explicitly separated, the engineer can apply classical automata theory to the discrete part and classical analysis to the continuous counterpart – *e. g.* we will show that in this context notions of bisimulation, observable behaviour, and regular expression can be obtained straightforwardly using coalgebra. Just to get an idea of the things that we will explore, and not getting into many details for now, we will see that the oscillator's *black-box representation* corresponds to the regular expression below,

$$\mu x.\ ((x \mid \mathtt{a} := \mathtt{1}) \owedge (\dot{\mathtt{x}} = \mathtt{a} \,\&\, \mathtt{1}) \mid \mathtt{a} := -\mathtt{1}) \owedge (\dot{\mathtt{x}} = \mathtt{a} \,\&\, \mathtt{1})$$

Intuitively, the construct $\mu x.$ introduces recursion and the expression $(\varphi \mid \mathtt{a} := \mathtt{t})$ denotes a transition to a state specified by φ together with a change in the computer's internal memory $(\mathtt{a} := \mathtt{t})$. The conjunction $\owedge$ serves to put together two pieces of information about the current state, namely the trajectory that it outputs and the information about the next transition.

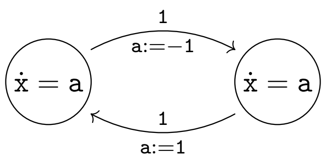
4.2.2 *Component-based software development in hybrid programming*

The black-box perspective and the hybrid monad (Chapter 3) will be key ingredients in introducing hybrid programs with internal memory, but more generally on providing *a solid foundation* *for component-based software development in hybrid programming* [Szy98; BO03; Sif+15] which is the current chapter's main goal: in particular, (i) hybrid programs with internal memory, based on the black-box perspective, are formally seen as components in the sense of [BO03]; (ii) and the hybrid monad will yield different composition operators for them, as explained below.

Component-based software development is often explained with a visual metaphor: a palette of computational units, and a canvas in which they are dropped and interconnected by drawing wires that abstract different composition and synchronisation mechanisms. One generic way of looking at components, proposed in [BO03], emphasises an *observational* semantics, through a signature of observers and methods, that makes them amenable to a coalgebraic characterisation as (generalisations of) Mealy machines. The resulting component calculus [BO03] is parametric on a behavioural model given by a monad, and analogously to Moggi's case [Mog89], this captures partial, non-deterministic, probabilistic, and, using Chapter 3's results, hybrid components.

We will show that black-box *representations*, such as the one for the oscillator, can be naturally interpreted as hybrid programs with internal memory *i. e.* hybrid components in the sense of [BO03], which provides for free different composition operators to them. We will focus on providing a suitable language for these representations, and suitable notions of bisimulation and observational behaviour. This provides the first steps towards component-based software development in hybrid programming.

We start with a simple observation by turning our attention back to the oscillator and its visual representation as a black-box: one can unfold the latter into the automaton below, which is much easier to draw than the black-box, and, more importantly, moves us close to coalgebra, as explained next.



Such automata can be encoded as coalgebras quite easily: to see how, consider a finite set of real-valued variables $X = \{x_1, \dots, x_n\}$ and recall the grammars of terms and events that were used in Chapter 3,

$$t \ni r \mid r \cdot x \mid t + t \qquad\qquad \psi \ni t \leq t \mid t \geq t \mid \psi \wedge \psi \mid \psi \vee \psi$$

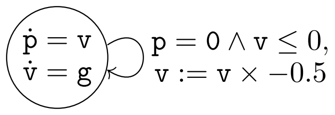
with $r \in \mathbb{R}$ and $x \in X$. Then denote by $\mathrm{Ev}(X)$ the set of 'time-triggered' differential equations $(\dot{x}_1 = t, \dots, \dot{x}_n = t \,\&\, r)$, and by $\mathrm{Tr}(X)$ the set of discrete assignments $(x_1 := t, \dots, x_n := t)$. Coalgebras of the type,

$$M \to M \times \mathrm{Tr}(X) \times \mathrm{Ev}(X)$$

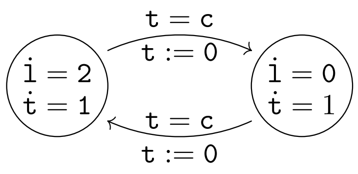
capture the automaton above. Let us denote the set of event-triggered differential equations also by $\mathrm{Ev}(X)$. Then

Examples 4.2.2.

1. Consider a bouncing ball dropped at a specific positive height $\mathtt{p}$ and with no initial velocity $\mathtt{v}$. Due to the gravitational acceleration $\mathtt{g}$, it falls into the ground and then bounces back up, losing part of its kinetic energy in the process. The following automaton sums up this behaviour.



2. Consider now a system comprised of a tank and a valve connected to it. The valve allows water to flow in filling the tank at the rate of 2cm/s during intervals of $\mathtt{c}$ seconds; between these periods the valve is shut also for $\mathtt{c}$ seconds. We can describe this behaviour via the automaton below.



Remark 4.2.3. The reader may have noticed that the automata above are very similar to hybrid automata (Chapter 2). This connection is detailed in Section 4.3. In Section 4.4 we will show that a straightforward generalisation of the coalgebraic theory developed in this section provides a uniform framework for hybrid automata and their variants.

Since we also wish to have an input dimension, we will take a slightly more general perspective by considering finite coalgebras of the type,

$$M \to M \times (\mathrm{Tr}(X) \times \mathrm{Ev}(X))^I$$

where I is a finite set. As alluded before, we will call them *representations*, due to their ability to represent hybrid programs with internal memory.

4.2.3 *Regular expressions*

Using document [Sil10, Chapter 4] as basis, we start to develop the coalgebraic theory of hybrid programs with internal memory and their representations by pursuing a Kleene-like theorem for the latter. In order to avoid a heavy notation we will drop the letter X in the expressions $\mathrm{Tr}(X)$ and $\mathrm{Ev}(X)$.

Let Pos be the category of partially ordered sets and monotone maps. The forgetful functor $\mathsf{Pos} \to \mathsf{Set}$ has a left adjoint $\mathsf{Set} \to \mathsf{Pos}$ which equips a set with its discrete order.

$$\begin{tikzcd} \mathsf{Set} \arrow[r, bend left] \arrow[r, phantom, "\perp"] & \mathsf{Pos} \arrow[l, bend left] \end{tikzcd}$$

Recall that a semilattice X is a partially ordered set with finite joins. Recall also that every object X in Pos has an order-embedding into its Dedekind-Macneille completion $\mathscr{C}X$ [Joh86, page 109]: if X is discrete, this is just the set $X + \{\bot, \top\}$ whose order extends that of X by interpreting the elements $\{\bot, \top\}$ as a bottom and top elements, respectively.

Definition 4.2.4. Consider two partially ordered sets $(X, \leq)$ and $(Y, \leq)$. Their product is the set $X \times Y$ equipped with the product order defined by,

$$(a, b) \leq (c, d) \equiv a \leq c \text{ and } b \leq d$$

Proposition 4.2.5. *The product of two semilattices is also a semilattice.*

Proof. This is a particular case of the following fact: if two categories have limits of a certain type then their product has limits of the same type. □

We are ready to introduce the grammar of regular expressions for representations. In order to be more familiar to the hybrid systems' community, the grammar (and associated notions) will be slightly different than the one introduced in [Sil10, Chapter 4]. The changes, however, do not require substantial modifications to the theory nor to the underlying proofs of most results in the *op. cit.*

Definition 4.2.6. Let I be a finite set of inputs and X a finite set of variables. Then consider the grammars below.

$$\epsilon \ni \underline{\emptyset} \mid x \mid i\ (\epsilon \mid a) \mid i \downarrow b \mid \epsilon \oslash \epsilon \mid \mu x.\, \gamma, \qquad \gamma \ni \underline{\emptyset} \mid i\ (\epsilon \mid a) \mid i \downarrow b \mid \gamma \oslash \gamma \mid \mu x.\, \gamma$$

$(a \in \mathscr{C}\mathrm{Tr}, b \in \mathscr{C}\mathrm{Ev}, i \in I, x \in X)$. An expression ϵ is called *closed* if every variable x in ϵ is under the scope of the binder μx; the set of closed expressions is denoted by Exp. Expressions γ which occur after a binder μx are called *guarded*; and the set of guarded expressions is denoted by Exp_g.

Intuitively, for an input i the expression $i\ (\epsilon \mid a)$ denotes a transition to a state specified by ϵ; the letter a records the assignment associated with the transition. The expression $i \downarrow b$ specifies the evolution that the current mode will produce for an input $i \in I$. Finally, the construct $\mu x.$ introduces recursion and the construct $\oslash$ works as the conjunction. This will be formally detailed in Definition 4.2.9.

Remark 4.2.7. Assume that the set of inputs I is the singleton set. The previous grammar can then be simplified into,

$$\epsilon \ni \underline{\emptyset} \mid x \mid (\epsilon \mid a) \mid b \mid \epsilon \oslash \epsilon \mid \mu x.\, \gamma, \qquad \gamma \ni \underline{\emptyset} \mid (\epsilon \mid a) \mid b \mid \gamma \oslash \gamma \mid \mu x.\, \gamma$$

$(a \in \mathscr{C}\mathrm{Tr}, b \in \mathscr{C}\mathrm{Ev}, x \in X)$. If we also wish to discard assignments, as in the case of switched systems (Chapter 2), the grammar can be further simplified into,

$$\epsilon \ni \underline{\emptyset} \mid x \mid \diamond \epsilon \mid b \mid \epsilon \oslash \epsilon \mid \mu x.\, \gamma, \qquad \gamma \ni \underline{\emptyset} \mid \diamond \epsilon \mid b \mid \gamma \oslash \gamma \mid \mu x.\, \gamma$$

$(b \in \mathscr{C}\mathrm{Ev}, x \in X)$.

Example 4.2.8. Recall the bouncing ball introduced in Example 4.3.3. Its behaviour is formulated by the expression,

$$\mu x.\, (x \mid \mathtt{v} := \mathtt{v} \times -0.5) \ \oslash\ (\dot{\mathtt{p}} = \mathtt{v} \wedge \dot{\mathtt{v}} = \mathtt{g},\ \mathtt{p} \leq 0 \wedge \mathtt{v} \leq 0)$$

The set of closed expressions Exp can be equipped with a coalgebra structure,

$$\mathrm{Exp} \to (\mathrm{Exp} \times \mathrm{Cmd})^I$$

where Cmd is the product of the semilattices $\mathscr{C}\mathrm{Tr}$ and $\mathscr{C}\mathrm{Ev}$. We will see that this provides a natural semantics to expressions and moreover it allows to relate them with representations' internal modes in regard to bisimilarity.

Definition 4.2.9. Define the coalgebra $\overline{\langle \delta_1, \delta_2 \rangle} : \mathrm{Exp} \to (\mathrm{Exp} \times \mathrm{Cmd})^I$ as follows.

$$\begin{aligned}
\delta_1(\underline{\emptyset}, i) &= \underline{\emptyset} & \delta_2(\underline{\emptyset}, i) &= \bot \\
\delta_1(j(\epsilon \mid a), i) &= \epsilon \text{ if } (i = j) \text{ else } \underline{\emptyset} & \delta_2(j(\epsilon \mid a), i) &= (a, \bot) \text{ if } (i = j) \text{ else } \bot \\
\delta_1(j \downarrow b, i) &= \underline{\emptyset} & \delta_2((j \downarrow b), i) &= (\bot, b) \text{ if } (i = j) \text{ else } \bot \\
\delta_1(\epsilon_1 \oslash \epsilon_2, i) &= \delta_1(\epsilon_1, i) \oslash \delta_1(\epsilon_2, i) & \delta_2(\epsilon_1 \oslash \epsilon_2, i) &= \delta_2(\epsilon_1, i) \vee \delta_2(\epsilon_2, i) \\
\delta_1(\mu x.\gamma, i) &= \delta_1(\gamma[\mu x.\gamma/x], i) & \delta_2(\mu x.\gamma, i) &= \delta_2(\gamma[\mu x.\gamma/x], i)
\end{aligned}$$

The expression $\gamma[\mu x.\gamma/x]$ denotes syntactic substitution. It reads: 'in the expression γ replace any free occurrence of x by $\mu x.\gamma$'.

Proposition 4.2.10. *The coalgebra* $\overline{\langle \delta_1, \delta_2 \rangle} : \mathrm{Exp} \to (\mathrm{Exp} \times \mathrm{Cmd})^I$ *is well-defined. More concretely, the equations*

$$\delta_1(\mu x.\gamma, i) = \delta_1(\gamma[\mu x.\gamma/x], i), \qquad \delta_2(\mu x.\gamma, i) = \delta_2(\gamma[\mu x.\gamma/x], i)$$

are well-defined.

Proof. The proof is analogous to the one in [Sil10, page 55]. □

As discussed in Examples 4.1.6 (3), the functor $(-\times \mathrm{Cmd})^I : \mathsf{Set} \to \mathsf{Set}$ admits a final coalgebra,

$$\mathrm{Cmd}^{I^+} \to (\mathrm{Cmd}^{I^+} \times \mathrm{Cmd})^I$$

which means that every expression $\epsilon \in \mathrm{Exp}$ can be mapped into its behaviour $[\![\epsilon]\!] \in \mathrm{Cmd}^{I^+}$. Assume that $I = 1$ and consider the composition,

$$\mathrm{beh}: \ \mathrm{Exp} \xrightarrow{[\![-]\!]} \mathrm{Cmd}^{1^+} \xrightarrow{\simeq} \mathrm{Cmd}^{\omega}$$

We will use it to compute the stream associated with the expression in Example 4.2.8.

Example 4.2.11. Recall the bouncing ball described in Example 4.3.3 and its expression,

$$\mu x.\, (x \mid \mathtt{v} := \mathtt{v} \times -0.5) \ \oslash \ (\dot{\mathtt{p}} = \mathtt{v} \wedge \dot{\mathtt{v}} = \mathtt{g},\ \mathtt{p} \leq 0 \wedge \mathtt{v} \leq 0)$$

The associated stream is $((\mathtt{a},\mathtt{b}),(\mathtt{a},\mathtt{b}),\dots)$ where $\mathtt{a}$ denotes the assignment $\mathtt{v} := \mathtt{v} \times -\mathtt{0.5}$ and $\mathtt{b}$ the pair $(\dot{\mathtt{p}} = \mathtt{v} \wedge \dot{\mathtt{v}} = \mathtt{g},\ \mathtt{p} \leq \mathtt{0} \wedge \mathtt{v} \leq \mathtt{0})$. Abbreviating the expression above into $\mu x.\psi$, this is generated by unfolding,

$$\begin{aligned}
\mathrm{beh}(\mu x.\,\psi) &= ((\mathtt{a},\bot) \vee (\bot,\mathtt{b})) : \mathrm{beh}(\mu x.\,\psi \ \oslash\ \underline{\emptyset}) \\
&= (\mathtt{a},\mathtt{b}) : \mathrm{beh}(\mu x.\,\psi \ \oslash\ \underline{\emptyset}) \\
&= ((\mathtt{a},\mathtt{b}),(\mathtt{a},\mathtt{b}) \vee \bot) : \mathrm{beh}(\mu x.\,\psi \ \oslash\ \underline{\emptyset} \ \oslash\ \underline{\emptyset}) \\
&= ((\mathtt{a},\mathtt{b}),(\mathtt{a},\mathtt{b})) : \mathrm{beh}(\mu x.\,\psi \ \oslash\ \underline{\emptyset} \ \oslash\ \underline{\emptyset}) \\
&= \dots \\
&= ((\mathtt{a},\mathtt{b}),(\mathtt{a},\mathtt{b}),(\mathtt{a},\mathtt{b})\dots)
\end{aligned}$$

Given the set of expressions Exp, the next natural step is to provide a correspondence between representations' internal modes and expressions. For this, observe that the embedding $\mathrm{Tr}\times\mathrm{Ev} \hookrightarrow \mathscr{C}\mathrm{Tr} \times \mathscr{C}\mathrm{Ev}$ allows to interpret without loss of information a $(-\ \times \mathrm{Tr} \times \mathrm{Ev})^I$-coalgebra as a $(-\ \times \mathrm{Cmd})^I$-coalgebra. Thus,

Definition 4.2.12. Take a finite $(-\ \times \mathrm{Cmd})^I$-coalgebra $\overline{\langle \mathrm{nxt}, \mathrm{out}\rangle} : M \to (M \times \mathrm{Cmd})^I$ with $M = \{m_1, \dots, m_n\}$. For every mode $m_l \in M$ define the expression,

$$A_l^0 = \mu m_l.\, \oslash_{i\in I}\ i((m_l)_i \mid a_l) \ \oslash\ i \downarrow b_l$$

where $(m_l)_i = \mathrm{nxt}(m_l, i)$ and $(a_l, b_l) = \mathrm{out}(m_l, i)$. Moreover, $A_l^{k+1} = A_l^k\{A_{k+1}^k / x_{k+1}\}$ with $k \in \{0, \dots, n-1\}$ and where $\{A_{k+1}^k / x_{k+1}\}$ denotes substitution without renaming the free variables in A_{k+1}^k that become bound due to the substitution. Finally, define $\epsilon_l = A_l^n$.

Intuitively, this construction eliminates free variables at each iteration, starting with m_1 and ending with m_n.

Proposition 4.2.13. *Consider an expression A_l^k with $k \in \{1, \dots, n\}$. All variables $m_1 \leq m \leq m_k$ are closed in A_l^k.*

Proof. The proof is analogous to the one in [Sil10, Theorem 4.2.7] □

Example 4.2.14. Recall Examples 4.2.2 (2), which describes the behaviour of a water tank system. Let us compute the expression relative to the left mode of the associated automaton. Denote the assignment $\mathtt{t} := \mathtt{0}$ by $\mathtt{a}$, and the tuples,

$$(\dot{\mathtt{i}} = \mathtt{2}, \dot{\mathtt{t}} = \mathtt{1} \,\&\, \mathtt{t} = \mathtt{c}), \qquad\qquad (\dot{\mathtt{i}} = \mathtt{0}, \dot{\mathtt{t}} = \mathtt{1} \,\&\, \mathtt{t} = \mathtt{c})$$

by $\mathtt{b_1}$ and $\mathtt{b_2}$, respectively. Then we have,

$$\begin{aligned} A_1^0 &= \mu m_1.(m_2 \mid \mathtt{a}) \oslash \mathtt{b}_1 & A_2^1 &= A_2^0\{A_1^0/m_1\} \\ A_2^0 &= \mu m_2.(m_1 \mid \mathtt{a}) \oslash \mathtt{b}_2 & A_1^2 &= A_1^0\{A_2^1/m_2\} \\ A_1^1 &= A_1^0\{A_1^0/m_1\} = A_1^0 & A_2^2 &= A_2^1\{A_2^1/m_2\} = A_2^1 \end{aligned}$$

Hence, the left mode of the automaton corresponds to the expression A_1^2,

$$\mu m_1.\ (\epsilon \mid \mathtt{a}) \oslash \mathtt{b}_1$$

where ϵ abbreviates $\mu m_2.\ (\mu m_1.\ (m_2 \mid \mathtt{a}) \oslash \mathtt{b}_1 \mid \mathtt{a}) \oslash \mathtt{b}_2$.

Let us consider the bisimulation equivalence R generated by the set $\{(\epsilon, \epsilon \oslash \underline{\emptyset}) \mid \epsilon \in \mathrm{Exp}\}$ [Sil10, Theorem 4.3.4]. As discussed in the previous section, this equivalence induces a quotient coalgebra,

$$\mathrm{Exp}/R \to (\mathrm{Exp}/R \times \mathrm{Cmd})^I$$

of the coalgebra of expressions $\delta : \mathrm{Exp} \to (\mathrm{Exp} \times \mathrm{Cmd})^I$. In order to avoid a burdened notation, we will use the latter as if it were the former.

Theorem 4.2.15. *Recall Definition 5.3.1. Let $m_l \in M$ be the internal mode of a representation $(M, \overline{\langle \mathrm{nxt}, \mathrm{out} \rangle})$ and ϵ_l the associated expression. Then we have $m_l \sim \epsilon_l$.*

Proof. In Appendix A. □

Our goal now is to generate a representation from an expression $\epsilon \in \mathrm{Exp}$ – recall that the former are interpreted as finite $(- \times \mathrm{Cmd})^I$-coalgebras and *vice-versa*. In a first analysis, it may seem that the task is done, since there exists a smallest subcoalgebra $\overline{\{\epsilon\}}$ of (Exp, δ) whose carrier contains the expression ϵ. However, as discussed in [Sil10, pages 59 and 60], this coalgebra is not necessarily finite. To overcome this, we need to further quotient the coalgebra of expressions $\delta : \mathrm{Exp} \to (\mathrm{Exp} \times \mathrm{Cmd})^I$ using the equivalences,

$$\epsilon \oslash \epsilon \equiv \epsilon, \qquad \epsilon_1 \oslash \epsilon_2 \equiv \epsilon_2 \oslash \epsilon_1, \qquad (\epsilon_1 \oslash \epsilon_2) \oslash \epsilon_3 \equiv \epsilon_1 \oslash (\epsilon_2 \oslash \epsilon_3)$$

This gives rise to a quotient map $[\_] : \mathrm{Exp} \twoheadrightarrow \mathrm{Exp}/R$, and the associated quotient coalgebra,

$$m : \mathrm{Exp}/R \to (\mathrm{Exp}/R \times \mathrm{Cmd})^I$$

We will prove that for every expression $\epsilon \in \mathrm{Exp}$ the subcoalgebra $\overline{\{[\epsilon]\}}$ of $(\mathrm{Exp}/R, m)$ is finite. The strategy for this will rely on showing the existence of a finite set $\mathrm{C}(\epsilon)$ that contains $[\epsilon]$ and that forms a subcoalgebra of $(\mathrm{Exp}/R, m)$. By the definition of smallest subcoalgebra, its existence entails that $\overline{\{[\epsilon]\}}$ is finite.

Definition 4.2.16. Consider an expression $\epsilon \in \mathrm{Exp}$. Let $\mathrm{cl}(\epsilon)$ be the smallest set generated by the following equations, with a slight abuse of notation in the last one.

$$\begin{aligned} \mathrm{cl}(x) &= \{x\} & \mathrm{cl}(i \downarrow b) &= \{i \downarrow b\} \\ \mathrm{cl}(\underline{\emptyset}) &= \{\underline{\emptyset}\} & \mathrm{cl}(\epsilon_1 \owedge \epsilon_2) &= \mathrm{cl}(\epsilon_1) \cup \mathrm{cl}(\epsilon_2) \cup \{\epsilon_1 \owedge \epsilon_2\} \\ \mathrm{cl}(i(\epsilon \mid a)) &= \mathrm{cl}(\epsilon) \cup \{i(\epsilon \mid a)\} & \mathrm{cl}(\mu x.\gamma) &= \mathrm{cl}(\gamma[\mu x.\gamma/x]) \cup \{\mu x.\gamma\} \end{aligned}$$

Then define the set $\mathrm{C}(\epsilon)$ as,

$$\{\, [\epsilon_1 \owedge \cdots \owedge \epsilon_k] \mid \epsilon_1, \ldots, \epsilon_k \in \mathrm{cl}(\epsilon),\ \text{and } \epsilon_1, \ldots, \epsilon_k \text{ all distinct} \,\}$$

with $[\underline{\emptyset}]$ as the empty sum.

Theorem 4.2.17. *For every $\epsilon \in \mathrm{Exp}$ the associated set $\mathrm{C}(\epsilon)$ is finite and forms a subcoalgebra of* $(\mathrm{Exp}/R, m)$*.*

Proof. Let κ be an expression in Exp and κ_i be its next state for an input i. Since the set $\mathrm{cl}(\epsilon)$ is finite the set $\mathrm{C}(\epsilon)$ must be finite as well. In order to show that the set $\mathrm{C}(\epsilon)$ forms a subcoalgebra of $(\mathrm{Exp}/R, m)$ we will prove that,

$$\text{if } [\kappa] \in \mathrm{C}(\epsilon) \text{ then } [\kappa_i] \in \mathrm{C}(\epsilon)$$

This result follows by induction on a complexity measure for closed expressions [Sil10, Definition 4.2.3]. In particular,

- $[\underline{\emptyset}_i] = [(a \downarrow b)_i] = [\underline{\emptyset}] \in \mathrm{C}(\epsilon)$, by definition of $\mathrm{C}(\epsilon)$.
- If $[i(\psi \mid a)] \in \mathrm{C}(\epsilon)$ then $i(\psi \mid a) \in \mathrm{cl}(\epsilon)$ and therefore $[\psi] \in \mathrm{C}(\epsilon)$.
- If $[\kappa_1 \owedge \kappa_2] \in \mathrm{C}(\epsilon)$ then by the induction hypothesis $[\kappa_{1i}], [\kappa_{2i}] \in \mathrm{C}(\epsilon)$. We may assume that $\kappa_{1i} = [\psi_1 \owedge \cdots \owedge \psi_k]$, $\kappa_{2i} = [\xi_1 \owedge \cdots \owedge \xi_l]$. By definition, we have $[\psi_1 \owedge \cdots \owedge \psi_k \owedge \xi_1 \owedge \cdots \owedge \xi_l] \in \mathrm{C}(\epsilon)$.
- If $[\mu x.\gamma] \in \mathrm{C}(\epsilon)$ then, by definition of $\mathrm{cl}(\epsilon)$, $[\gamma[\mu x.\gamma/x]] \in \mathrm{C}(\epsilon)$. Using the induction hypothesis, $[(\gamma[\mu x.\gamma/x])_i] \in \mathrm{C}(\epsilon)$.

$\square$

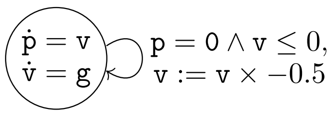
Corollary 4.2.18. *Let $\epsilon \in \mathrm{Exp}$ be an expression. Then there exists a finite $(- \times \mathrm{Cmd})^I$-coalgebra $\overline{\{[\epsilon]\}}$ and $[\epsilon] \sim \epsilon$.*

Examples 4.2.19. We will consider the bouncing ball and the water tank that were described in Examples 4.2.2.

1. The bouncing ball corresponds to the expression,

$$\mu x.\, (x \mid \mathtt{v} := \mathtt{v} \times -0.5) \owedge (\dot{\mathtt{p}} = \mathtt{v} \wedge \dot{\mathtt{v}} = \mathtt{g},\ \mathtt{p} \leq 0 \wedge \mathtt{v} \leq 0)$$

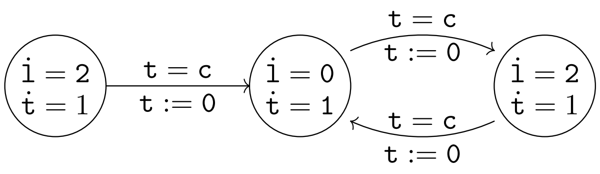
which we abbreviate into $\mu x.\psi$. The subcoalgebra $\overline{\{[\mu x.\psi]\}}$ unfolds into the automaton below.


$\dot{\mathtt{p}} = \mathtt{v}$
$\dot{\mathtt{v}} = \mathtt{g}$
$\mathtt{p} = 0 \wedge \mathtt{v} \leq 0,$
$\mathtt{v} := \mathtt{v} \times -0.5$

2. For the water tank, we saw that the expression $\mu m_1.\, (\epsilon \mid \mathtt{a}) \oslash \mathtt{b}_1$ specifies the automaton's left mode where ϵ is

$$\mu m_2.\, (\mu m_1.\, (m_2 \mid \mathtt{a}) \oslash \mathtt{b}_1 \mid \mathtt{a}) \oslash \mathtt{b}_2$$

Denote the expression $\mu m_1.\, (\epsilon \mid \mathtt{a}) \oslash \mathtt{b}_1$ by $\mu m_1.\, \psi_1$. The subcoalgebra $\overline{\{[\mu m_1.\psi_1]\}}$ unfolds into the automaton below.



In Examples 4.2.2 (2), we specified the water tank via an automaton with two modes. Here we ended up with three, because the subcoalgebra $\overline{\{[\mu m_1.\psi_1]\}}$ is not minimal. To make it so one needs to show that the left and right modes are bisimilar and then identify them.

4.2.4 *Semantics*

Denote the category $\mathsf{CoAlg}\left((- \times \mathrm{H}(\mathbb{R}^n))^I\right)$ of hybrid components by Hyb, and then the category of their representations $\mathsf{CoAlg}\left((- \times \mathrm{Tr} \times \mathrm{Ev})^I\right)$ by RepHyb. We start this subsection by building an interpretation functor,

$$\mathsf{RepHyb} \to \mathsf{Hyb}$$

To achieve this, recall some basic machinery introduced in Chapter 3, namely the interpretation map $[\![-]\!] : \mathsf{At}^{\mathsf{e}}(X) \to \mathrm{End}_{\mathrm{H}}(\mathbb{R}^n)$ used for generating event-triggered programming languages and the natural transformation $\lambda : \mathrm{H} \to \mathrm{Id}$ that sends evolutions to their last point. Consider also a map,

$$\langle a, b, c\rangle : M \times I \to M \times \mathrm{Tr} \times \mathrm{Ev}$$

It induces a new map $M \times \mathbb{R}^n \times I \to M \times \mathbb{R}^n \times \mathrm{H}(\mathbb{R}^n)$ defined by,

$$(m, v, i) \mapsto \big(a(m,i),\ [\![b(m,i)]\!] \cdot \lambda_{\mathbb{R}^n} \cdot [\![c(m,i)]\!](v),\ [\![c(m,i)]\!](v)\big) \tag{10}$$

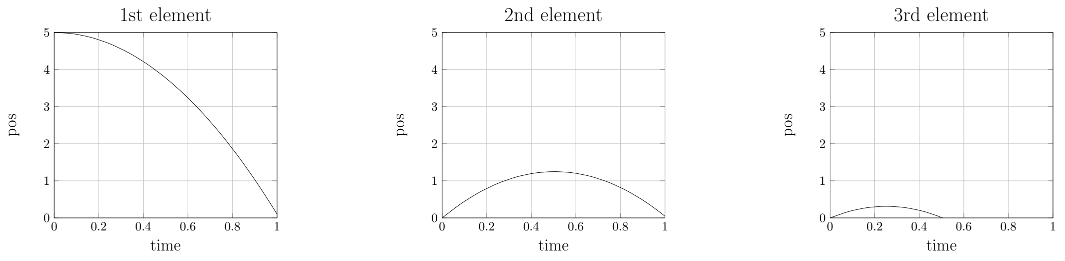
Theorem 4.2.20. *There exists an interpretation functor* $[\![-]\!] : \mathsf{RepHyb} \to \mathsf{Hyb}$ *that acts on objects as (10) and that sends a coalgebra morphism f to* $f \times \mathrm{id}$.

Proof. A direct consequence of Theorem 4.4.2. □

Intuitively, every representation induces a hybrid component such that, for every internal state, $(m, v) \in M \times \mathbb{R}^n$ and external input $i \in I$, it gives rise to an observable, continuous evolution – *i. e.* an element of $\mathrm{H}(\mathbb{R}^n)$ – and an internal, discrete transition to the next state. Let us illustrate this idea with a few examples.

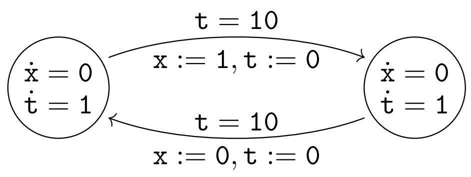
Examples 4.2.21.

1. Consider the bouncing ball from Examples 4.2.2 and its automaton (M, c). The latter's interpretation $[\![(M, c)]\!]$ is clearly a $(- \times \mathrm{H}(\mathbb{R}^n))$-coalgebra which means that for every state $(m, v) \in M \times \mathbb{R}^n$ there is a canonical, observable behaviour $[\![(m, v)]\!] \in (\mathrm{H}(\mathbb{R}^n))^\omega$. For example, hiding the evolutions concerning the ball's velocity to keep our illustration simple, the first three elements of $[\![(m, 5, 0)]\!]$ are shown in the following plots.

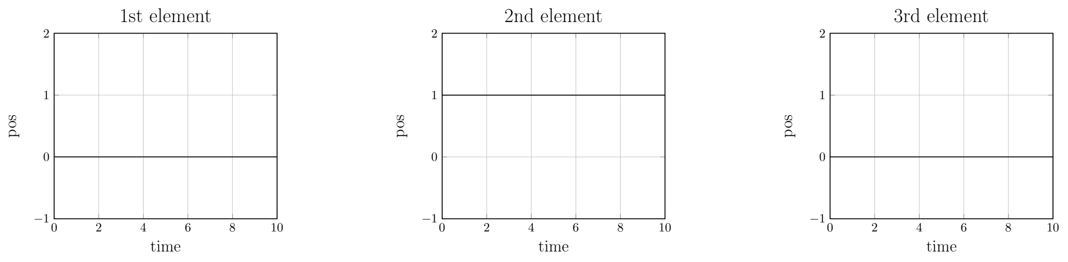


2. Recall the tank-and-valve system described in Examples 4.2.2 and its automaton (M, c). The latter's interpretation $[\![(M, c)]\!]$ is also a $(- \times \mathrm{H}(\mathbb{R}^n))$-coalgebra and thus for every state $(m, v) \in M \times \mathbb{R}^n$ there is a canonical, observable behaviour $[\![(m, v)]\!] \in (\mathrm{H}(\mathbb{R}^n))^\omega$. In this case what we observe is the water level going up at intervals of $\mathtt{c}$ seconds.

3. Recall that every ten seconds the semaphore described in Examples 3.2.15 switches between a red light and a green one. In the black-box perspective, this corresponds the following automaton.



Denote its left mode by $m \in M$ and observe that its interpretation $[\![(M, c)]\!]$ is a $(- \times \mathrm{H}(\mathbb{R}^n))$-coalgebra. The state $(m, 0, 0) \in M \times \mathbb{R}^2$ yields a stream of evolutions $[\![(m, 0, 0)]\!]$ whose first three elements are depicted below.

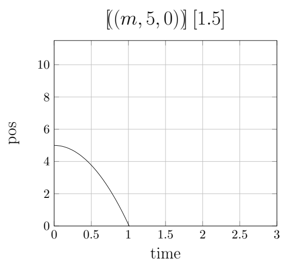


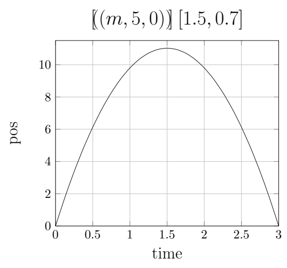
4. Suppose also that one is able to change the dampening factor of the bouncing ball at each bounce. This gives rise to an automaton,

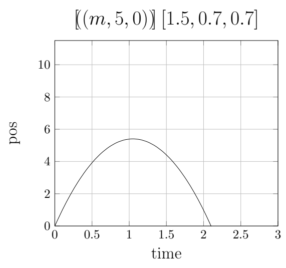
$$(M, m) : M \to (M \times \mathrm{Tr} \times \mathrm{Ev})^I$$

with I as the set of possible dampening factors and with $\mathtt{v} := \mathtt{v} \times -\mathtt{i}$ as the new assignment. The interpretation $[\![(M,c)]\!]$ is now a $(- \times \mathrm{H}(\mathbb{R}^n))^I$-coalgebra which means that for every state $(m,v) \in M \times \mathbb{R}^n$ there exists an observable behaviour $[\![(m,v)]\!] \in (\mathrm{H}(\mathbb{R}^n))^{I^+}$.

For example, the following three expressions $[\![(m,5,0)]\!]\,[1.5]$, $[\![(m,5,0)]\!]\,[1.5, 0.7]$, and $[\![(m,5,0)]\!]\,[1.5, 0.7, 0.7]$ yield the sequence of plots below.







The coalgebraic semantics illustrated in these examples marks a clear frontier between the discrete domain and the continuous one: the elements of $\mathrm{H}(\mathbb{R}^n)$ that we considered live in the continuous domain whilst the structures $(\mathrm{H}(\mathbb{R}^n))^\omega$ and $(\mathrm{H}(\mathbb{R}^n))^{I^+}$ clearly possess a discrete nature. The semantics is thus faithful to the black-box perspective previously discussed since all the discrete behaviour is kept hidden and the only thing that we can observe are the continuous evolutions.

4.2.5 *Bisimulation*

We already know that the coalgebraic notion of bisimulation is parametric on a functor, which provides at once notions of bisimulation for several types of transition system. Two subsections back, when we developed a language for representations, we adopted a notion of bisimulation that was obtained precisely in this way. From now on we will call it *syntactic bisimulation* since it is anchored to the syntactic level (the level of representations) rather than to the semantic one (the level of hybrid components).

As the reader may have already guessed, syntactic bisimulation is often too strong. In the bouncing ball (Example 4.2.2), for example, to replace the differential equation $\dot{\mathtt{p}} = \mathtt{v}$ by $\dot{\mathtt{p}} = \mathtt{v} + \mathtt{0}$ yields a totally different automaton in the sense that the mode of the original one is not syntactically bisimilar to the mode of the modified version. Nevertheless, one knows that at the semantic level both modes will produce exactly the same behaviour. Fortunately, the category Hyb of hybrid components also carries a notion of bisimulation and there exists a functor,

$$[\![-]\!] : \mathsf{RepHyb} \to \mathsf{Hyb}$$

that maps each representation to its corresponding model. These will allows us to compare two internal modes of representations also at the semantic level.

The following result is a sanity check which states that if two internal modes are syntactically bisimilar then they are bisimilar also at the semantic level.

Theorem 4.2.22. *Consider two representations (M, c), (N, d), two modes $m \in M$, $n \in N$ that are syntactically bisimilar $m \sim n$, and a state $(m, v) \in M \times \mathbb{R}^n$. The property $(m, v) \ \sim \ (n, v)$ holds.*

Proof. Since $m \ \sim \ n$, there must exist a span $\{\pi_i : (R, r) \to (M_i, c_i)\}_{i \in 2}$ in RepHyb such that $(m, n) \in R$. The interpretation functor $[\![-]\!] : \mathsf{RepHyb} \to \mathsf{Hyb}$ maps this span into a another span in Hyb, which by an application of Examples 4.1.4 (5) provides $(m, v) \sim ((m, n), v) \sim (n, v)$ □

Previously, we showed that every internal mode m of a representation (M, c) induces an expression ϵ such that m and ϵ are syntactically bisimilar. An application of the previous theorem shows that m and ϵ are bisimilar also at the semantics level. The corollary below is another useful consequence of the previous theorem; intuitively, it states that when computing the observable behaviour of an internal mode $m \in M$ one can either move directly to Hyb and compute its behaviour there, or first compute its behaviour $(\!| m |\!)_{(M,c)}$ in RepHyb and then move to Hyb to compute the behaviour of $(\!| m |\!)_{(M,c)}$.

Corollary 4.2.23. *Let (M, c) be a representation. For every state $(m, v) \in M \times \mathbb{R}^n$ the following property holds.*

$$(m, v) \ \sim \ ((\!| m |\!)_{(M,c)}, v)$$

Proof. Follows directly from Examples 4.1.4 (5) and the previous theorem. □

4.3 A DETOUR TO THE LAND OF HYBRID AUTOMATA

4.3.1 *The general theory*

We will now show how representations connect to hybrid automata – the standard formalism for hybrid systems. As discussed in the following section, this exposition will lay the necessary steps to generalise the previous section's work into a uniform theory of hybrid automata and variants.

We start by introducing the classical definition of hybrid automata and associated notions. We will use document [Hen96] as basis, but with a notation much closer to the coalgebraic perspective so that the connection with our work becomes clear.

Definition 4.3.1. Given a finite set X, the set of predicates φ over X, denoted by $\mathrm{Pr}(X)$, is generated by the grammar below in the left.

$$\varphi \ni \neg\varphi \mid \varphi \wedge \varphi \mid t < t \mid t = t, \qquad t \ni t + t \mid t \cdot t \mid x \mid r \qquad (x \in X, r \in \mathbb{R})$$

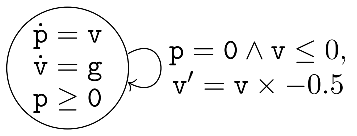
Definition 4.3.2. A hybrid automaton is a tuple $(M, e, X, \mathrm{inv}, \mathrm{dyn}, \mathrm{asg}, \mathrm{grd})$ such that

- M is a finite set of control modes and $e : M \to \mathrm{P}M$ is a relation between them.
- X is a finite set of real-valued variables $\{x_1, \ldots, x_n\}$.

- inv : $M \to \Pr(X)$ is a function that associates to each mode an invariant, the latter being given as a predicate over the variables in X.
- dyn : $M \to \Pr(X \cup \dot{X})$ is a function that associates to each mode a predicate over $X \cup \dot{X}$, where the set $\dot{X} = \{\dot{x}_1, \ldots, \dot{x}_n\}$ represents the first derivatives of the variables in X. This map is used to dictate which evolutions may occur in a given mode.
- Denote by $\mathscr{G}(f)$ be the graph of a function f. Then, asg : $\mathscr{G}(e) \to \Pr(X \cup X')$ is a function that associates to each edge a predicate over $X \cup X'$, where $X' = \{x'_1, \ldots, x'_n\}$ represents the variables in X immediately after a discrete jump. In other words, asg is a function that provides an assignment to each edge.
- Finally, the function grd : $\mathscr{G}(e) \to \Pr(X)$ associates to each edge a guard.

The following example of a bouncing ball may help to illustrate and clarify some aspects of this definition.

Example 4.3.3. Consider a bouncing ball dropped at a specific positive height $\mathtt{p}$ and with no initial velocity $\mathtt{v}$. Due to the gravitational acceleration $\mathtt{g}$, it falls into the ground and then bounces back up, losing part of its kinetic energy in the process. The following hybrid automaton sums up this behaviour.


$\dot{\mathtt{p}} = \mathtt{v}$
$\dot{\mathtt{v}} = \mathtt{g}$
$\mathtt{p} \geq 0$
$\mathtt{p} = 0 \wedge \mathtt{v} \leq 0,$
$\mathtt{v}' = \mathtt{v} \times -0.5$

Observe that only one mode exists; let us call it m. Furthermore there exists exactly one discrete transition: $(m, m) \in \mathscr{G}(e)$. Then $X = \{\mathtt{p}, \mathtt{v}\}$, and inv$(m)$ is $\mathtt{p} \geq 0$. Moreover, we have that grd(m, m) is $\mathtt{p} = 0 \wedge \mathtt{v} \leq 0$, dyn$(m)$ is $\dot{\mathtt{p}} = \mathtt{v} \wedge \dot{\mathtt{v}} = \mathtt{g}$, and asg$(m, m)$ is $\mathtt{v}' = \mathtt{v} \times -0.5 \wedge \mathtt{p}' = \mathtt{p}$. Note that the second conjunct does not appear in the hybrid automaton above, a common practice to avoid a notational burden.

Contrary to document [Hen96], we do not consider initial states nor labels in the definition of hybrid automata. This is because we wish to keep our results simple and intuitive, and both mechanisms can be accommodated later on in a straightforward manner.

Assumptions 4.3.4. We also make the following common assumptions.

1. The function dyn always returns a system of linear differential equations *i. e.* a system $\{\dot{x}_1 = e_1 \ldots \dot{x}_n = e_n\}$ where e_i is a linear combination of variables in $X = \{x_1 \ldots x_n\}$. This is a usual assumption (*e.g.* [Alu+95; Dav97; Jac00]), as the hybrid systems described in literature rarely involve non-linear differential equations which are often untractable. The important point is that this condition allows the function dyn : $M \to \Pr(X \cup \dot{X})$ to induce a map,

$$\text{sol} : M \times \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathbb{R}^n$$

such that given a pair $(m, v) \in M \times \mathbb{R}^n$, the map $\mathrm{sol}\,(m, v, -) : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ is the solution of the system of differential equations associated to a specific mode and valuation $(m, v) \in M \times \mathbb{R}^n$. Its domain $(\mathbb{R}_{\geq 0})$ represents time and n is the cardinality of the set X (*i. e.* the number of real-valued variables). This is possible because linear systems of differential equations always have unique solutions [Per13, page 17].

2. All assignments are deterministic, *i. e.* they take the form $x := t$ with t a term.

For a predicate φ on $\mathbb{R}^n$ abbreviate the condition $(v_1 \dots v_n) \in [\![\varphi]\!]$ by $(v_1 \dots v_n) \models \varphi$. The traditional semantics of hybrid automata is given in terms of labelled transition systems [Hen96]. More concretely,

Definition 4.3.5. A hybrid automaton H induces a transition system (Z_H, L, t_H) such that,

$$Z_H = \{(m, v) \in M \times \mathbb{R}^n \mid v \models \mathrm{inv}(m)\}$$

$L = 1 + \mathbb{R}_{\geq 0}$, and the map $t_H : Z_H \times L \to \mathrm{P}(Z_H)$ is defined by,

- $(m_2, v_2) \in t_H(m_1, v_1, *)$ iff one has $m_2 \in e(m_1)$, $v_1 \models \mathrm{grd}(m_1, m_2)$, and $(v_1, v_2) \models \mathrm{asg}(m_1, m_2)$.
- $(m_2, v_2) \in t_H(m_1, v_1, r)$ iff $m_1 = m_2$, $\mathrm{sol}(m_1, v_1, r) = v_2$, and for all $a \in [0, r]$ $\mathrm{sol}(m_1, v_1, a) \models \mathrm{inv}(m_1)$.

We write $z_2 \in t_H(z_1, l)$ as $z_1 \xrightarrow{l} z_2$, and, whenever found suitable, we omit the subscripts in (Z_H, L, t_H).

Example 4.3.6. Let us recall the hybrid automaton that describes the bouncing ball of Example 4.3.3. The associated labelled transition system (Z, L, t) is defined as $Z = \{m\} \times \mathbb{R}_{\geq 0} \times \mathbb{R}$, and

- $(m, p_1, v_1) \xrightarrow{*} (m, p_2, v_2)$ iff $p_1 = 0 \wedge v_1 \leq 0$ and $v_2 = v_1 \times -0.5 \wedge p_2 = p_1$.
- $(m, p_1, v_1) \xrightarrow{r} (m, p_2, v_2)$ iff $\mathrm{sol}(m, p_1, v_1, r) = (p_2, v_2)$, and for every $a \in [0, r]$ one has $\mathrm{sol}(m, p_1, v_1, a) \models \mathsf{p} \geq 0$.

The function sol, determined by dyn, describes the continuous evolution of the ball's position and velocity between jumps.

Remark 4.3.7. Note that both discrete events and continuous evolutions are embedded into the map t. Not only this makes difficult to adopt the black-box perspective mentioned before, but it also makes the verification of hybrid automata extremely challenging, as a large number of states and edges need to be taken into consideration. The standard technique for overcoming the latter is to quotient the state space by a bisimulation equivalence, *i. e.* to collapse states that possess equivalent behaviour. The resulting states become symbolic representations of (possibly infinite) regions, and verification techniques are applied to the reduced system instead.

Definition 4.3.8. Consider the underlying labelled transition system (Z, L, t) of a hybrid automaton and an equivalence relation $\Phi \subseteq Z \times Z$ over the states. A Φ-*bisimulation* $R \subseteq Z \times Z$ is a relation R such that $x_1 \, R \, y_1$ entails the following cases:

1. $x_1 \, \Phi \, y_1$, and for every label $l \in L$,
2. if $x_1 \xrightarrow{l} x_2$ then there exists a state y_2 such that $y_1 \xrightarrow{l} y_2$ and $x_2 \, R \, y_2$,
3. if $y_1 \xrightarrow{l} y_2$ then there exists a state x_2 such that $x_1 \xrightarrow{l} x_2$ and $x_2 \, R \, y_2$.

Two states $x, y \in Z$ are Φ-*bisimilar* (in symbols, $x \equiv^{\Phi} y$) if they are related by a Φ-bisimulation.

This is the notion of bisimulation that hybrid automata traditionally use.

Let us consider now a probabilistic variant of hybrid automata, often referred to as probabilistic hybrid automata [Spr00a; Spr00b].

Definition 4.3.9. A probabilistic hybrid automaton is a tuple $(M, X, e, \mathrm{inv}, \mathrm{dyn})$ where,

- M is a finite set of control modes.
- X is a finite set of real-valued variables $\{x_1, \dots, x_n\}$.
- $e : M \to \mathrm{PD}\,(M \times \mathrm{Pr}(X \cup X') \times \mathrm{Pr}(X))$ is a function that associates to each mode a set of probability distributions over modes, assignments, and guards.
- $\mathrm{inv} : M \to \mathrm{Pr}(X)$ is a function that associates to each mode a predicate over the variables in X.
- $\mathrm{dyn} : M \to \mathrm{Pr}(X \cup \dot{X})$ is a function that associates to each mode a predicate over the variables in $X \cup \dot{X}$.

Probabilistic hybrid automata capture the likelihood of events associated with digital computations. For example, the probability of occurring a computer malfunction if temperature gets too high, or the probability of a reset after a specific time. Technically, these automata harbour such a behaviour due to their transition maps, which before a jump (or state transition) presents us with set of distributions (over modes, guards, and assignments) to choose from. The chosen distribution is then used to compute the likelihood of a given state to become the next one in the execution process.

Document [Spr00a] provides several examples of probabilistic hybrid automata and explains some of their intricacies. Their semantics is also discussed: it is given in terms of probabilistic transition systems which requires some preliminary definitions that we will recall next.

Definition 4.3.10. Consider a probabilistic hybrid automaton. Given a pair $(m, v) \in Z$, define $N_{(m,v)} \subseteq \mathrm{D}Z$ as the set such that $\mu \in N_{(m,v)}$ iff there exists a distribution $\nu \in e(m)$ that respects the following condition: let $\{(m_1, a_1, g_1), \dots, (m_n, a_n, g_n)\}$ denote the support of ν; for every pair $(m', v') \in Z$ we have,

$$\mu(m', v') = \sum_{i \in I} \nu(m_i, a_i, g_i), \qquad I = \big\{1 \leq i \leq n \;\mid\; m' = m_i, v' = a_i(v), v \models g_i\big\}$$

Intuitively, the summation above is used to add the probabilities of triples in the set $M \times \mathrm{Pr}(X \cup X') \times \mathrm{Pr}(X)$ that lead to the same result from a valuation $(m, v) \in M \times \mathbb{R}^n$.

Remark 4.3.11. The previous definition is slightly simpler than the one given in [Spr00a, page 3], a consequence of Assumptions 4.3.4 (2). Probabilistic hybrid automata traditionally come equipped with a colouring map $M \to C$. This feature can be accommodated in Definition 4.3.9 straightforwardly: recall the embedding of functions into relations,

$$\mathscr{G} : C^M \rightarrowtail \mathrm{P}(M \times C)$$

and observe that a probabilistic hybrid automaton with a colouring map $M \to C$ can be seen as a probabilistic hybrid automaton whose set of modes is a subset of the cartesian product $M \times C$.

Definition 4.3.12. A probabilistic hybrid automaton induces a probabilistic transition system (Z, L, t), such that $L = 1 + \mathbb{R}_{\geq 0}$, and the map $t : Z \times L \to \mathrm{PD}Z$ is defined by,

$$t(m, v, *) = N_{(m,v)}$$
$$t(m, v, r) = \begin{cases} \{\delta\} & \text{if } \mathrm{sol}(m, v, a) \models \mathrm{inv}(m) \quad (a \in [0, r]) \\ \emptyset & \text{otherwise} \end{cases}$$

where δ is the Dirac distribution on $(m, \mathrm{sol}(m, v, r))$. Finally, let us recall the standard notion of bisimulation for probabilistic hybrid automata.

Definition 4.3.13. Recall from Examples 4.1.4 that every relation $R \subseteq X \times Y$ induces a relation on distributions $\asymp_R \subseteq \mathrm{D}X \times \mathrm{D}Y$. Consider a probabilistic hybrid automaton, its transition system (Z, L, t), and let $\Phi \subseteq Z \times Z$ be an equivalence relation. A relation $R \subseteq Z \times Z$ is a probabilistic Φ-bisimulation iff $z_1 \, R \, z_2$ entails:

1. $z_1 \, \Phi \, z_2$,
2. if $\mu_1 \in t(z_1, l)$ then there exists a distribution $\mu_2 \in t(z_2, l)$ such that $\mu_1 \asymp_R \mu_2$,
3. if $\mu_2 \in t(z_2, l)$ then there exists a distribution $\mu_1 \in t(z_1, l)$ such that $\mu_1 \asymp_R \mu_2$.

Two states $x, y \in Z$ are Φ-bisimilar (in symbols, $x \equiv^\Phi y$) if they are related by a Φ-bisimulation.

4.3.2 *Hybrid automata as coalgebras*

Let the set $\mathrm{Pr}(X \cup X') \times \mathrm{Pr}(X)$ of assignments and guards be denoted by Tr, and the set $\mathrm{Pr}(X \cup \dot{X}) \times \mathrm{Pr}(X)$ of differential equations and invariants by Ev.

Remark 4.3.14. The transition map $\langle \mathrm{asg}, \mathrm{grd} \rangle : \mathscr{G}(e) \to \mathrm{Tr}$ of a hybrid automaton can be embedded into the set $\mathrm{P}(M \times \mathrm{Tr})^M$, as witnessed by the following composition.

$$\mathrm{Tr}^{\mathscr{G}(e)} \rightarrowtail (\mathrm{Tr}+1)^{M \times M} \rightarrowtail (\mathrm{P}\mathrm{Tr})^{M \times M} \simeq ((\mathrm{P}\mathrm{Tr})^M)^M \simeq \mathrm{P}(M \times \mathrm{Tr})^M$$

Every hybrid automaton is, therefore, equivalently represented by a coalgebra,

$$M \to \mathrm{P}(M \times \mathrm{Tr}) \times \mathrm{Ev}$$

for the functor $\mathrm{P}(- \times \mathrm{Tr}) \times \mathrm{Ev} : \mathsf{Set} \to \mathsf{Set}$, which means that hybrid automata are essentially a *non-deterministic generalisation of representations* (Section 4.2).

Even if quite simple, this remark has several useful consequences. First, it tells that hybrid automata can be organised in a category of coalgebras and therefore part of their theory comes for free – as shown in Section 4.2 for the deterministic case, this includes notions of bisimulation, observable behaviour, and regular expression languages. Secondly, it provides the basis for a systematic, coalgebraic description of hybrid automata and their variants, in many cases the coalgebraic descriptions being simpler than the standard counterparts. Probabilistic hybrid automata, for example, are simply finite coalgebras of the type $M \to \mathrm{PD}(M \times \mathrm{Tr}) \times \mathrm{Ev}$. Thirdly, the remark is the basis for a uniform framework of hybrid automata by considering finite coalgebras of the type,

$$M \to (F(M \times \mathrm{Tr}) \times \mathrm{Ev})^I \tag{11}$$

where I is a finite set and $F : \mathsf{Set} \to \mathsf{Set}$ is a 'discrete' transition type. We call them F-representations.

In the sequel, we provide a generic, coalgebraic semantics to the latter, which generalises the standard semantics for a subclass of classical and probabilistic hybrid automata. Contrary to the usual semantics of these devices (see Remark 4.3.7), the generic, coalgebraic one is faithful to the black-box perspective introduced in this chapter, and keeps discrete and continuous behaviour separated; not only this supports component-based software development [BO03], but it also makes easy for the engineer to apply classical automata theory to the discrete part and classic analysis to the continuous one (Remark 4.2.1).

4.4 HYBRID MACHINES: WHEN DIFFERENT TYPES OF MEMORY APPEAR

4.4.1 *The general picture*

Denote the right transpose of a function $f : A \times B \to C$ by $\overline{f} : A \to C^B$. In this section we examine coalgebras of the type $\overline{\langle \mathrm{nxt}, \mathrm{out} \rangle} : M \to (F(M \times \mathrm{Tr}) \times \mathrm{Ev})^I$ where $F : \mathsf{Set} \to \mathsf{Set}$ determines a discrete transition type and I is a finite set of inputs. These arrows can be decomposed into,

$$\mathrm{nxt} : M \times I \to F(M \times \mathrm{Tr}), \qquad \mathrm{out} : M \times I \to \mathrm{Ev}$$

Coalgebra	**Functor F**	**Behaviour**
$M \to (\mathrm{Id}(M \times \mathrm{Tr}) \times \mathrm{Ev})^I$	$\mathrm{Id}X = X$	Deterministic [Liu+99]
$M \to (\mathrm{M}(M \times \mathrm{Tr}) \times \mathrm{Ev})^I$	$\mathrm{M}X = X + 1$	Faulty
$M \to (\Delta(M \times \mathrm{Tr}) \times \mathrm{Ev})^I$	$\Delta X = X \times X$	Replicating
$M \to (\mathrm{P}(M \times \mathrm{Tr}) \times \mathrm{Ev})^I$	$\mathrm{P}X = \{A \subseteq X\}$	Nondeterministic [Hen96]
$M \to (\mathrm{D}(M \times \mathrm{Tr}) \times \mathrm{Ev})^I$	$\mathrm{D}X \subseteq \{\mu \in [0,1]^X \mid \mu[X] = 1\}$	Probabilistic [Spr00b]
$M \to (\mathrm{PD}(M \times \mathrm{Tr}) \times \mathrm{Ev})^I$	PD —	Segala [Spr00b]
$M \to (\mathrm{W}(M \times \mathrm{Tr}) \times \mathrm{Ev})^I$	$\mathrm{W}X \subseteq S^X$; S is a semiring.	Weighted [ATP04]

Table 1.: Possible variants of F.

which makes clear that variations in the functor $F : \mathsf{Set} \to \mathsf{Set}$ correspond to variations in the transition relation e and the maps asg and grd (recall Definition 4.3.2). In other words, F dictates how an F-representation discretely jumps to a next (internal) state.

Table 1 lists functors $F : \mathsf{Set} \to \mathsf{Set}$ and respective F-representations. Some of the latter are already well known (*e. g.* the non-deterministic case in the fourth row), and others are new (*e. g.* the replicating case in the third row). This illustrates the high level of genericity that coalgebra brings to the theory of hybrid automata: specific types of automaton are captured in specific instantiations of $F : \mathsf{Set} \to \mathsf{Set}$ and global constructions and results are defined parametric on F once and for all.

Faulty and replicating behaviour. The Maybe functor $\mathrm{M} : \mathsf{Set} \to \mathsf{Set}$ (second row) brings faulty behaviour into the scene by giving rise to coalgebras,

$$M \to (\mathrm{M}(M \times \mathrm{Tr}) \times \mathrm{Ev})^I$$

which can terminate an execution at a discrete transition, as a response, for instance, to a program exception or loss of information.

The diagonal functor $\Delta : \mathsf{Set} \to \mathsf{Set}$ yields coalgebras typed as,

$$M \to (\Delta(M \times \mathrm{Tr}) \times \mathrm{Ev})^I$$

These behave like the representations introduced Section 4.2, but now a discrete transition goes to two different places at the same time, which means that these systems 'replicate' themselves at each discrete transition – for example, in this context the bouncing ball turns into two at each bounce. From a strict computer science point of view, this kind of behaviour may seem rather strange, but in other areas it is quite common: *e. g.* in biology, cells replicate when a specific saturation point is reached.

Nondeterministic, probabilistic, and Segala behaviour. The powerset functor $\mathrm{P} : \mathsf{Set} \to \mathsf{Set}$ leads to coalgebras typed as,

$$M \to (\mathrm{P}(M \times \mathrm{Tr}) \times \mathrm{Ev})^I$$

which clearly subsume classical hybrid automata (see Section 4.3), and similarly for $\mathrm{PD} : \mathsf{Set} \to \mathsf{Set}$ with probabilistic hybrid automata.

Weighted behaviour. Consider a semiring S and recall that it induces a weight functor $\mathrm{W} : \mathsf{Set} \to \mathsf{Set}$ (Chapter 3). The latter yields finite coalgebras typed as,

$$M \to (\mathrm{W}(M \times \mathrm{Tr}) \times \mathrm{Ev})^I$$

which decorate each edge, associated guard and assignment, with a cost. This feature has already been discussed sometimes in the hybrid systems' domain (*e. g.* [ATP04; Bou06]).

4.4.2 *Semantics*

Let us now introduce a generic semantics for a broad class of F-representations, by following essentially the same steps than the ones presented in Section 4.2. We will not consider state invariants, because they can be accomodated later on straightfowardly and we want to keep our exposition clear and intuitive.

We start by placing a restriction on F-representations: assume that there can only exist one guard per each internal mode, and that a transition from a mode must occur *as soon as* the corresponding guard is satisfied. The restriction is achieved by assuming that Ev is the set of event-triggered differential equations $(\dot{x}_1 = t, \ldots, \dot{x}_n = t \;\&\; \psi)$ and Tr is the set of discrete assignments $(x_1 := t, \ldots, x_n := t)$ both introduced in Chapter 3.

Remark 4.4.1. This condition is important so that the models of F-representations can have a meaningful notion of observable behaviour whenever F is bounded [GS02, page12]. This topic is more throughly detailed in the following section, and also in the following chapter.

In the lines below, we will see that the interpretation of an F-representation is naturally defined as a coalgebra of the type,

$$(M \times \mathbb{R}^n) \to (F(M \times \mathbb{R}^n) \times \mathrm{H}(\mathbb{R}^n))^I$$

which clearly generalises the notion of hybrid component presented in Section 4.2. Note that this broader class of coalgebras does *not* give components in general [BO03] since neither $F\mathrm{H}$ nor $\mathrm{H}F$ are necessarily monads. We will thus call them F-hybrid machines, due to their resemblance with Mealy machines (Section 4.1).

Denote the category of $(F(-) \times \mathrm{H}(\mathbb{R}^n))^I$-coalgebras by $\mathsf{Hyb}(F)$, and the category of $(F(- \times \mathrm{Tr}) \times \mathrm{Ev})^I$-coalgebras by $\mathsf{RepHyb}(F)$. We will build an interpretation functor,

$$\mathsf{RepHyb}(F) \to \mathsf{Hyb}(F)$$

To achieve this, we need to recall some basic machinery introduced in Chapter 3, namely the interpretation map $[\![-]\!] : \mathsf{At}^{\mathsf{e}}(X) \to \mathrm{End}_{\mathrm{H}}(\mathbb{R}^n)$ used for generating event-triggered programming languages and the natural transformation $\lambda : \mathrm{H} \to \mathrm{Id}$ that sends evolutions to their last point. Recall also that every functor $F : \mathsf{Set} \to \mathsf{Set}$ is equipped with a natural transformation $\alpha : F \times \mathrm{Id} \to F(\mathrm{Id} \times \mathrm{Id})$ [Jac16, Exercises 2.5.4]. Given a map $M \times I \to F(M \times \mathrm{Tr}) \times \mathrm{Ev}$ consider the composition,

$$\begin{tikzcd}
M \times I \arrow[r] \arrow[rr, bend right, "\langle a,b \rangle"'] & F(M \times \mathrm{Tr}) \times \mathrm{Ev} \arrow[r, "F(\mathrm{id} \times [\![-]\!]) \times [\![-]\!]"] & F(M \times \mathrm{End}_{\mathrm{Id}}(\mathbb{R}^n)) \times \mathrm{End}_{\mathrm{H}}(\mathbb{R}^n)
\end{tikzcd}$$

It induces a new map $M \times \mathbb{R}^n \times I \to F(M \times \mathbb{R}^n) \times \mathrm{H}(\mathbb{R}^n)$ defined by,

$$(m, v, i) \mapsto \Big(F(\mathrm{id} \times \mathrm{ev}) \cdot \alpha\big(a(m,i), \lambda \cdot b(m,i)(v)\big),\ b(m,i)(v)\Big) \tag{12}$$

Theorem 4.4.2. *There exists an interpretation functor $[\![-]\!] : \mathsf{RepHyb}(F) \to \mathsf{Hyb}(F)$ that acts on objects as (12) and that sends a coalgebra morphism f to $f \times \mathrm{id}$.*

Proof. It is straightforward to show that the mapping above preserves identity maps and distributes over composition. So it remains to show that it sends morphisms in $\mathsf{RepHyb}(F)$ to morphisms in $\mathsf{Hyb}(F)$. Take a $\mathsf{RepHyb}(F)$-morphism,

$$f : (M, \overline{\langle a, b \rangle}) \to (N, \overline{\langle c, d \rangle})$$

We will prove that the diagram,

$$\begin{tikzcd}
M \times \mathbb{R}^n \times I \arrow[r, "(f \times \mathrm{id}) \times \mathrm{id}"] \arrow[d, "{[\![\langle a,b \rangle]\!]}"'] & N \times \mathbb{R}^n \times I \arrow[d, "{[\![\langle c,d \rangle]\!]}"] \\
F(M \times \mathbb{R}^n) \times \mathrm{H}(\mathbb{R}^n) \arrow[r, "F(f \times \mathrm{id}) \times \mathrm{id}"'] & F(N \times \mathbb{R}^n) \times \mathrm{H}(\mathbb{R}^n)
\end{tikzcd}$$

commutes by showing that the two diagrams below commute.

$$\begin{tikzcd}
M \times \mathbb{R}^n \times I \arrow[r, "(f \times \mathrm{id}) \times \mathrm{id}"] \arrow[d, "{\pi_1 \cdot [\![\langle a,b \rangle]\!]}"'] \arrow[dr, phantom, "(1)"] & N \times \mathbb{R}^n \times I \arrow[d, "{\pi_1 \cdot [\![\langle c,d \rangle]\!]}"] \\
F(M \times \mathbb{R}^n) \arrow[r, "F(f \times \mathrm{id})"'] & F(N \times \mathbb{R}^n)
\end{tikzcd}
\qquad
\begin{tikzcd}
M \times \mathbb{R}^n \times I \arrow[r, "(f \times \mathrm{id}) \times \mathrm{id}"] \arrow[d, "{\pi_2 \cdot [\![\langle a,b \rangle]\!]}"'] \arrow[dr, phantom, "(2)"] & N \times \mathbb{R}^n \times I \arrow[d, "{\pi_2 \cdot [\![\langle c,d \rangle]\!]}"] \\
\mathrm{H}(\mathbb{R}^n) \arrow[r, "\mathrm{id}"'] & \mathrm{H}(\mathbb{R}^n)
\end{tikzcd}$$

We start with Diagram (2). By assumption, we have $b(m) = d(f(m))$ which entails the commutativity of the diagram. Commutativity of Diagram (1) follows from the commutativity of the diagrams below,

$$\begin{tikzcd}
M \times \mathbb{R}^n \times I \arrow[r, "(f\times \mathrm{id})\times id"] \arrow[d, "g"'] & N \times \mathbb{R}^n \times I \arrow[d, "h"] \\
F(M \times \mathrm{Tr}) \times \mathbb{R}^n \arrow[r, "F(f\times \mathrm{id})\times \mathrm{id}"'] \arrow[d, "\alpha_{M\times \mathrm{Tr},\mathbb{R}^n}"'] & F(N \times \mathrm{Tr}) \times \mathbb{R}^n \arrow[d, "\alpha_{N\times \mathrm{Tr},\mathbb{R}^n}"] \\
F(M \times \mathrm{Tr} \times \mathbb{R}^n) \arrow[r, "F(f\times \mathrm{id}\times \mathrm{id})"'] \arrow[d, "F(\mathrm{id}\times \mathrm{ev})"'] & F(N \times \mathrm{Tr} \times \mathbb{R}^n) \arrow[d, "F(\mathrm{id}\times \mathrm{ev})"] \\
F(M \times \mathbb{R}^n) \arrow[r, "F(f\times \mathrm{id})"'] & F(N \times \mathbb{R}^n)
\end{tikzcd}$$

where $g(m, v, i) = \big(a(m, i), \lambda_{\mathbb{R}^n} \cdot b(m, i)(v)\big)$ and analogously for h. □

The following two theorems tell that the semantics defined above generalises the semantics of classical and probabilistic hybrid automata, which is proved by unfolding Definition 4.3.5. Recall that every hybrid automaton induces a labelled transition system (Z, L, t) (Definition 4.3.5). Then,

Theorem 4.4.3. *Consider a hybrid automaton and the corresponding* P*-representation* $(M, \langle a, b\rangle) \in \mathsf{RepHyb}(\mathrm{P})$. *Consider also two states* $(m_1, v_1), (m_2, v_2) \in Z$. *The following equivalences hold.*

$$(m_1, v_1) \xrightarrow{r} (m_1, v_2) \equiv \big(\pi_2 \cdot [\![\langle a, b\rangle]\!](m_1, v_1)\big)(r) = v_2$$
$$(m_1, v_1) \xrightarrow{*} (m_2, v_2) \equiv (m_2, v_2) \in \big(\pi_1 \cdot [\![\langle a, b\rangle]\!](m_1, v_1)\big) \qquad (\textit{if } v_1 \models \mathrm{grd}(m_1))$$

Let us now recall that every probabilistic hybrid automaton induces a probabilistic transition system (Z, L, t) (Definition 4.3.12). Then,

Theorem 4.4.4. *Consider a probabilistic hybrid automaton and the corresponding* PD*-representation* $(M, \langle a, b\rangle) \in \mathsf{RepHyb}(\mathrm{PD})$. *Consider also states* $(m_1, v_1) \in Z$, $(m_1, v_1) \in Z$, *and a distribution* $\mu \in \mathrm{D}Z$. *The following equivalences hold.*

$$(m_1, v_1) \xrightarrow{r} \delta_{(m_1,v_2)} \equiv \big(\pi_2 \cdot [\![\langle a, b\rangle]\!](m_1, v_1)\big)(r) = v_2$$
$$(m_1, v_1) \xrightarrow{*} \mu \equiv \mu \in \big(\pi_1 \cdot [\![\langle a, b\rangle]\!](m_1, v_1)\big) \qquad (\textit{if } v_1 \models \mathrm{grd}(m_1))$$

4.4.3 Φ-*bisimulation*

Our next task is to generalise the notion of Φ-bisimulation (described in Section 4.3) to F-representations by making it parametric on a transition type. In this quest, we will not consider inputs since all notions of Φ-bisimulation involved also do not, and moreover inputs can be accomodated later on straightforwardly, due to our coalgebraic approach.

Let us start by considering a functor $F : \mathsf{Set} \to \mathsf{Set}$ and a quotient map $q : X \twoheadrightarrow Q$. These data induce a functor between categories of coalgebras,

$$G_q : \mathsf{CoAlg}\,(F(-) \times \mathrm{H}X) \to \mathsf{CoAlg}\,(F(-) \times \mathrm{H}Q)$$

Also, an F-hybrid machine $(M \times \mathbb{R}^n, [\![c]\!])$ induces an $F(-) \times \mathrm{H}(M \times \mathbb{R}^n)$-coalgebra,

$$[\![c]\!]^\dagger : M \times \mathbb{R}^n \to F(M \times \mathbb{R}^n) \times \mathrm{H}(M \times \mathbb{R}^n)$$

defined by $(m, v) \mapsto \big(\pi_1 \cdot [\![c]\!](m, v), \langle \underline{m}, \pi_2 \cdot [\![c]\!](m, v)\rangle\big)$. Now,

Definition 4.4.5. Consider an F-representation (M, c) and an equivalence relation over the state space $\Phi \subseteq (M \times \mathbb{R}^n) \times (M \times \mathbb{R}^n)$. A relation $R \subseteq (M \times \mathbb{R}^n) \times (M \times \mathbb{R}^n)$ is a coalgebraic Φ-bisimulation if it is a bisimulation for the coalgebra $G_\Phi([\![(M, c)]\!]^\dagger)$.

Two states $s_1, s_2 \in M \times \mathbb{R}^n$ are coalgebraically Φ-bisimilar, in symbols $s_1 \sim^\Phi s_2$, if they are related by a coalgebraic Φ-bisimulation.

As discussed in Section 4.3, hybrid automata and probabilistic hybrid automata already carry a notion of Φ-bisimulation, detailed in Definition 4.3.8 and Definition 4.3.13. The following results relate both definitions with Definition 4.4.5.

Theorem 4.4.6. *Consider a hybrid automaton, the corresponding* P*-representation* $(M, c) \in \mathsf{RepHyb}(\mathrm{P})$*, and a* Φ*-bisimulation* $R \subseteq Z \times Z$*. The equivalence below holds.*

$$z_1 \sim^\Phi z_2 \;\equiv\; z_1 \equiv^\Phi z_2$$

Proof. Direct consequence of Lemma A.2.5 and Lemma A.2.6. □

Theorem 4.4.7. *Consider a probabilistic hybrid automaton, the corresponding* PD*-representation* $(M, c) \in \mathsf{RepHyb}(\mathrm{PD})$*, and a coalgebraic* Φ*-bisimulation* $R \subseteq Z \times Z$*. The equivalence below holds.*

$$z_1 \sim^\Phi z_2 \;\equiv\; z_1 \equiv^\Phi z_2$$

Proof. Direct consequence of Lemma A.2.7 and Lemma A.2.8. □

4.5 OPEN CHALLENGES

In the current chapter we built a basis for component-based software development in hybrid programming: we showed that its basic elements are representations (of hybrid black-box machines) and that they are naturally interpreted as hybrid components (*i. e.* hybrid programs with internal memory) in the sense of [BO03]. Using standard results of coalgebra, we developed the basic theory of representations and their models, including languages, notions of bisimulation, and observational behaviour.

Remark 4.5.1. In this work we make a conceptual distinction between discrete and continuous behaviour, emphasising that the former should be internal, hidden from the environment, and the latter external, making up the observable behaviour. Interestingly, a somewhat dual view appears in document [Jac00]. The author pursues an *object-oriented approach* for hybrid systems by seeing them as coalgebras equipped with a monoid action (to represent time) that acts over the state space, forcing continuous evolutions to be hidden from the environment. This brings forth physical processes that continuously evolve internally and that we can only interact with via external, discrete events at specific periods of time.

We showed that the work in Section 4.2 can be straightforwardly extended into a broader context, by allowing representations to have different types of internal transitions. We saw that this covers different variants of hybrid automata, including the classical and probabilistic cases.

Despite being the standard formalism for hybrid systems, hybrid automata frequently need to be modified so that different types of computational behaviour can be taken into account. But in the coalgebraic theory developed here, the variations can be studied in a uniform manner, allowing results to be stated at a generic level, independently of whatever intricacies a variant may have. We exemplified this with a generic semantics and a generic notion of Φ-bisimulation for hybrid automata

The work reported in this chapter raised several interesting questions for which we lack a definitive answer. Let us summarise them next.

Behaviour. In Section 4.4, we put two restrictions on F-representations: namely, (i) each internal mode must have exactly one guard and (ii) *as soon as* the corresponding guard is satisfied the mode must switch. The first case can be dropped by interpreting F-representations as coalgebras of the type,

$$(M \times \mathbb{R}^n) \to F(M \times \mathbb{R}^n \times \mathrm{H}(\mathbb{R}^n))$$

in following manner: we can assume the existence of a map $\mathrm{Tr} \times (\mathbb{R}^n)^{\mathbb{R}_{\geq 0}} \to \mathbb{R}^n \times \mathrm{H}(\mathbb{R}^n)$ that for every guard, assignment, and trajectory, it returns both the shortest evolution $\mathrm{H}(\mathbb{R}^n)$ whose last point satisfies the guard, and the assignment's application to this last point. Then, for an F-representation $M \to F(M \times \mathrm{Tr}) \times \mathrm{Ev}$ we obtain its model,

$$M \times \mathbb{R}^n \longrightarrow F(M \times \mathrm{Tr}) \times (\mathbb{R}^n)^{\mathbb{R}_{\geq 0}} \longrightarrow F(M \times \mathrm{Tr} \times (\mathbb{R}^n)^{\mathbb{R}_{\geq 0}}) \longrightarrow F(M \times \mathbb{R}^n \times \mathrm{H}(\mathbb{R}^n))$$

via the tensorial strength of F[1]. *Can Restriction (ii) also be dropped without significant consequences ?* A first analysis of this question yields a negative answer: in order to drop the restriction, one needs to have a function,

$$\mathrm{Tr} \times (\mathbb{R}^n)^{\mathbb{R}_{\geq 0}} \to \mathrm{P}(\mathbb{R}^n \times \mathrm{H}(\mathbb{R}^n))$$

1 Recall that all endofunctors on Set have a tensorial strength [Jac16, Exercises 2.5.4].

to collect all trajectories whose last point satisfies the guard given as input. Then, for an F-representation $M \to F(M \times \mathrm{Tr}) \times \mathrm{Ev}$ we can build its model as shown above,

$$M \times \mathbb{R}^n \longrightarrow F(M \times \mathrm{Tr} \times (\mathbb{R}^n)^{\mathbb{R}_{\geq 0}}) \longrightarrow F(M \times \mathrm{P}(\mathbb{R}^n \times \mathrm{H}\mathbb{R}^n)) \longrightarrow F\mathrm{P}(M \times \mathbb{R}^n \times \mathrm{H}(\mathbb{R}^n))$$

this time also using the tensorial strength of P. Note that without Restriction (ii), all F-representations have some degree of non-determinism and, worse, their interpretations do *not* admit a final coalgebra due to the presence of the powerset functor. This turns out to be a quite thorny problem.

A way of solving this issue, up to some extent, is to shift from the category Set to other categories where continuous behaviour can be better handled. For example the category of topological spaces or the category of Polish spaces. We will pursue this strategy in the following chapter.

Bisimulation. We resorted to the uniform framework of hybrid automata, developed in the previous sections, for providing a generic notion of Φ-bisimulation. *Can we also do this for other types of bisimulation ?* A very interesting case would be a quantitative variant [GP11].

Calculi for Representations. We derived a coalgebraic language for representations (Section 4.2), and, as discussed in [Sil10], this result can be extended to a more general setting by allowing the discrete type functor $F : \mathsf{Set} \to \mathsf{Set}$ to be, for example, polynomial, the powerset, or a weight functor.

Using [Sil10], we can additionally derive axiomatisations for these languages, but in most cases the axioms will not be rich enough: they do not reflect the fact that differential equations themselves are part of the language; in other words, they do not encompass a calculus for differential equations, and we will therefore obtain problems analogous to those of syntactic bisimulation (Section 4.2). *Is it possible to extend the work in [Sil10] to obtain a systematic method for deriving complete axiomatisations for F-representations ?*

Composition operators. The chapter's main goal was to provide a solid foundation for a component-based software development discipline in hybrid programming. Up to a large extent, it was devoted to developing the basic theory of the corresponding components, from languages, to notions of bisimulation and observable behaviour. Using the hybrid monad and the generic component calculus introduced in [BO03], we can also systematically develop different composition operators for these components, as well as wiring mechanisms and refinement techniques. In particular, we are very interested on knowing which kinds of composition operator are supported by component-based hybrid programming, similarly to our goal in Section 3.3.

We will show one simple example of one such operator, without getting into many details in this concluding note: let us consider two representations $\langle a, b, c\rangle : M \to M \times \mathrm{Tr} \times \mathrm{Ev}$ and $\langle d, e, f\rangle : N \to N \times \mathrm{Tr} \times \mathrm{Ev}$. We can *sequentially* compose their models,

$$[\![\langle a, b, c\rangle]\!] \ ; [\![\langle d, e, f\rangle]\!] : M \times N \times \mathbb{R}^n \to M \times N \times \mathbb{R}^n \times \mathrm{H}(\mathbb{R}^n)$$

by defining $(m, n, v) \mapsto \big(\ a \times d\,(m,n)\,, \lambda_{\mathbb{R}^n} \cdot (\mathsf{p}\ ;\ [\![e(n)]\!])\,(v)\,,\ \ \mathsf{p}\,(v)\ \big)$ where p denotes the composition of hybrid programs $[\![c(m)]\!]\ ;\ [\![b(m)]\!]\ ;\ [\![f(n)]\!]$. Intuitively, the first component $\langle a, b, c\rangle$ produces an evolution and changes its internal state space in the expected way; then, the second component takes action by continuing the previous evolution and at the end it changes its internal state space as well.

Example 4.5.2. Consider the two oscillators below, which produce *triangular waves* with the period of one milisecond.

$$\begin{tikzcd}
\dot{\mathtt{x}} = 1 \arrow[r, bend left, "1"] & \dot{\mathtt{x}} = -1 \arrow[l, bend left, "1"] & ; & \dot{\mathtt{x}} = 1 \arrow[r, bend left, "1"] & \dot{\mathtt{x}} = -1 \arrow[l, bend left, "1"]
\end{tikzcd}$$

Their composition is another oscillator but that produces triangular waves with the period of two miliseconds.

This sequential composition is essentially the component-based analogue of the sequential composition operator for hybrid programs that was discussed in Chapter 3.

5 WHEN SETS DO NOT SUFFICE

Overview. We tackle two problems that came up in Chapter 3 and Chapter 4: the lack of suitable formalised notions of (i) stability in hybrid programming and (ii) behaviour for certain types of representations (of hybrid black-box machines). In both cases, the strategy will consist on moving part of our previous results into the category Top of topological spaces and continuous maps.

In regard to (i), we show that this new setting yields a topological analogue of the hybrid monad which not only provides a notion of stability to hybrid programming, but also supports a *compositional* analysis of hybrid programs with respect to this notion, effectively bringing aspects of control theory into the programming practice.

For (ii), we show that the topological framework allows to significantly increase the class of representations for which a suitable notion of observational behaviour can be obtained. In the process of addressing this last problem, we introduce several new theoretical theoretical results on limits in categories of coalgebras.

Roadmap. The chapter starts with a review of the existing literature on limits in categories of coalgebras. In the same section, we introduce three new results on this topic – Theorem 5.1.14, 5.1.22, and 5.1.27 – which take an important role in our quest (Section 5.1). In order to tackle problem (ii), we introduce Vietoris coalgebras, discuss their ability to interpret representations, and examine the existence of limits in their categories, putting a special focus on final coalgebras for the obvious reasons (Section 5.2 and 5.3). After this, we concentrate on the problem of stability in hybrid programming (Section 5.4). As usual, we conclude by discussing some challenges that emerged from this chapter's research and that we think deserve further study (Section 5.5).

5.1 PRELIMINARIES: LIMITS IN CATEGORIES OF COALGEBRAS

Limits in categories of coalgebras are the building blocks of several useful constructions in Coalgebra, including notions of observational behaviour, given by final coalgebras (see Chapter 4), and of coequation, which are sometimes given by equalisers of coalgebras [Adá05, page 173].

As already mentioned, colimits often exist because the forgetful functor $\mathsf{CoAlg}\,(F) \to \mathsf{C}$ creates them (Chapter 4). The story about limits is far more complex. We start this section by reviewing some well-known results on the topic, and in particular by recalling some basic notions of diagram functors.

Definition 5.1.1. A diagram $\mathscr{D} : \mathsf{I} \to \mathsf{C}$ is codirected if I is a codirected partially ordered set, *i. e.* I is non-empty and for all $i, j \in \mathsf{I}$ there is some $k \in \mathsf{I}$ with $k \leq i$ and $k \leq j$. A cone for a codirected diagram is called a codirected cone, and a limit of a codirected diagram is called a codirected limit.

Inverse sequence (or ω^{op}) diagrams, which have the shape depicted below, are codirected.

$$\cdot \longleftarrow \cdot \longleftarrow \cdot \longleftarrow \dots$$

Inverse sequence diagrams have a central role in showing that a given functor admits a final coalgebra. This is attested by the following well-known theorem [Adá05, Corollary 3.18].

Theorem 5.1.2. *Let C be a category with a final object 1 and $F : \mathsf{C} \to \mathsf{C}$ be a functor. If the category C has a limit L for the diagram,*

$$1 \longleftarrow F1 \longleftarrow FF1 \longleftarrow \dots$$

and F preserves this limit, then the canonical isomorphism $L \to FL$ is a final F-coalgebra.

The following theorem is another standard result but for general limits.

Theorem 5.1.3. *Assume that $F : \mathsf{C} \to \mathsf{C}$ preserves limits of a certain type. The forgetful functor $\mathsf{CoAlg}\,(F) \to \mathsf{C}$ creates limits of the same type.*

An important consequence of the last theorem is that $\mathsf{CoAlg}\,(F)$ has all types of limit that C has and that the functor $F : \mathsf{C} \to \mathsf{C}$ preserves. Unfortunately, as we will see later on, this assumption is frequently too strong. The following results, which resort to the notion of a covarietor, are often more helpful.

Definition 5.1.4. A functor $F : \mathsf{C} \to \mathsf{C}$ is called a *covarietor* if the canonical forgetful functor $\mathsf{CoAlg}\,(F) \to \mathsf{C}$ is a left adjoint.

The defining condition of a covarietor is quite mild; in fact all finitary functors respect it [GS01; GS02]. If F is a covarietor and C has a final object, the category $\mathsf{CoAlg}\,(F)$ admits a final coalgebra [GS01; GS02]. The latter is given by the right adjoint of $\mathsf{CoAlg}\,(F) \to \mathsf{C}$, which preserves final objects. One can also take advantage of the theory of (co)monads regarding (co)completeness of Eilenberg-Moore (co)algebras [AHS09, Theorem 20.56] to derive the following theorem (see [Lin69], [Adá05, Remark 4.4]).

Theorem 5.1.5. *Let F be a covarietor over a complete category. If $\mathsf{CoAlg}\,(F)$ has equalisers then it is also complete.*

Related to this, J. Hughes proved the following theorem [Hug01, Theorem 2.4.2].

Theorem 5.1.6. *Let C be regularly wellpowered (*i. e. *RegMono-wellpowered), cocomplete, and possess equalisers. Moreover, assume that it has an (Epi, RegMono)-factorisation structure, and that the functor $F : \mathsf{C} \to \mathsf{C}$ preserves regular monomorphisms. Then $\mathsf{CoAlg}\,(F)$ has equalisers.*

Using Theorem 5.1.5, one can then easily deduce the following corollary.

Corollary 5.1.7. *If the conditions in the last theorem hold,* C *is complete, and* F *is a covarietor, then the category* $\mathsf{CoAlg}\,(F)$ *is complete.*

We refer the interested reader to other important results on limits in categories of coalgebras: in particular, the work of A. Kurz [Kur01, Theorem 1.6.3], which shows that $\mathsf{CoAlg}\,(F)$ is complete whenever it has a suitable factorisation structure, F is a covarietor, and C is complete; and [GS01], where the authors study the existence of equalisers and products in categories of coalgebras over Set.

Next, we provide an improvement to Hughes' theorem using the notion of factorisation structure for cones [AHS09, Section 15]. Consider a cone $\mathcal{C} = (f_i : X \to X_i)_{i \in I}$ and a morphism $f : Z \to X$. We will often denote the cone $(f_i \cdot f : Z \to X_i)_{i \in I}$ simply by $\mathcal{C} \cdot f$.

Definition 5.1.8. Given a small category I, a cone for I in a category C is a functor $\mathscr{D} : \mathsf{I} \to \mathsf{C}$ together with a cone for $\mathscr{D}$. Let E be a class of C-morphisms and $\mathcal{M}$ a class of cones for I. We say that C has an $(E, \mathcal{M})$-factorisation structure of cones for I if,

1. every morphism in E is closed under post-composition with isomorphisms and every cone in $\mathcal{M}$ is closed under pre-composition with isomorphisms (*i. e.* if $\mathcal{C} \in \mathcal{M}$ and $f \in \mathrm{Iso}(\mathsf{C})$ then $\mathcal{C} \cdot f \in \mathcal{M}$).

2. the category C has $(E, \mathcal{M})$-factorisations of cones for I, *i. e.* every cone for I can be written as a composition $\mathcal{C} \cdot e$, where $\mathcal{C} \in \mathcal{M}$ and $e \in E$.

3. the category also has the unique fill-in property, *i. e.* for every diagram,

$$\begin{tikzcd}
\bullet \arrow[r, "e"] \arrow[d, "f"'] & \bullet \arrow[d, "g"] \arrow[dl, dotted, "d"'] \\
\bullet \arrow[r, "m"'] & \bullet
\end{tikzcd}$$

   such that $e \in E$, $\mathcal{C} = (f_i)_{i \in I} \in \mathcal{M}$, $\mathcal{D} = (g_i)_{i \in I}$ are cones for I and $\mathcal{C} \cdot f = \mathcal{D} \cdot e$, there exists a unique morphism d such that $\mathcal{C} \cdot d = \mathcal{D}$.

Definition 5.1.9. Consider a class $\mathcal{M}$ of cones for I. Each functor $\mathscr{D} : \mathsf{I} \to \mathsf{C}$ induces the full subcategory of $\mathsf{Cone}(\mathscr{D})$ whose objects are cones in $\mathcal{M}$. We say that the category C is $\mathcal{M}$-wellpowered if for every functor $\mathscr{D} : \mathsf{I} \to \mathsf{C}$ the corresponding subcategory of $\mathsf{Cone}(\mathscr{D})$ is essentially small (*i. e.* it is equivalent to a small category).

The following result is the key ingredient for improving Hughes' theorem. It is in the spirit of [AHS09, Section 12], which shows that 'cocompleteness almost implies completeness'.

Lemma 5.1.10. *Let* C *be a cocomplete category and* I *be a small category. Let also* E *be a class of* C*-morphisms closed under post-composition with isomorphisms and* $\mathcal{M}$ *be a class of cones for* I *in* C*. If* C *is* $\mathcal{M}$*-wellpowered and every cone for* I *has* $(E, \mathcal{M})$*-factorisations, then* C *has limits of shape* I*.*

Proof. We will show that the diagonal functor,

$$\Delta : \mathsf{C} \to \mathsf{C}^{\mathsf{I}}$$

has a right adjoint, using Freyd's General Adjoint Functor Theorem [ML98, page 120]. The category C is cocomplete by assumption and the functor Δ clearly preserves colimits, so we just need to show that the Solution Set Condition holds. In this context, it unfolds into the following condition: for every functor $\mathscr{D} : \mathsf{I} \to \mathsf{C}$, there exists a set $\mathcal{S}$ of cones for $\mathscr{D}$ such that every cone $(f_i : X \to \mathscr{D}(i))_{i\in\mathsf{I}}$ for $\mathscr{D}$ factors through a cone in $\mathcal{S}$.

Since C is $\mathcal{M}$-wellpowered we have, by assumption, a set $\mathcal{S}$ of representants for $\mathscr{D}$ in $\mathcal{M}$. Moreover, every cone for I has a $(E, \mathcal{M})$-factorisation, which means that a cone $(f_i : X \to \mathscr{D}(i))_{i\in\mathsf{I}}$ can be factorised as depicted below,

$$\begin{tikzcd}
X \arrow[rr, "f_i"] \arrow[dr, "e"'] & & \mathscr{D}(i) \\
& A \arrow[ur, "g_i"'] &
\end{tikzcd}$$

with the cone $(g_i : A \to \mathscr{D}(i))_{i\in\mathsf{I}}$ in $\mathcal{S}$. □

In order to apply this lemma to categories of coalgebras, we will need the following two results, which are simple generalisations of Theorem 4.1.11 and Theorem 4.1.18.

Theorem 5.1.11. *Let C be a category with an $(E, \mathcal{M})$-factorisation structure of cones for I and consider a functor $F : \mathsf{C} \to \mathsf{C}$ that sends cones in $\mathcal{M}$ to cones in $\mathcal{M}$. Then the category $\mathsf{CoAlg}\,(F)$ has $(\mathrm{U}^{-1}E, \mathrm{U}^{-1}\mathcal{M})$-factorisations and the class $\mathrm{U}^{-1}E$ is closed under post-composition with isomorphisms.*

Theorem 5.1.12. *Let C be an $\mathcal{M}$-wellpowered category. The category $\mathsf{CoAlg}\,(F)$ is $\mathrm{U}^{-1}\mathcal{M}$-wellpowered as well.*

Finally,

Theorem 5.1.13. *Let $F : \mathsf{C} \to \mathsf{C}$ be a functor over a cocomplete category C and I be a small category. Assume that C has an $(E, \mathcal{M})$-factorisation structure of cones for I, is $\mathcal{M}$-wellpowered, and that F sends cones in $\mathcal{M}$ to cones in $\mathcal{M}$. The category $\mathsf{CoAlg}\,(F)$ has limits of shape I.*

Proof. Follows directly from the three previous results. □

The following lines present two ways of constructing $(E, \mathcal{M})$-factorisation structures of cones for I from classical (E, M)-factorisation structures. This will allow us to precisely relate the last result with Hughes' theorem.

Let us consider a category C with an $(E \subseteq \mathrm{Epi}(\mathsf{C}), M)$-factorisation structure such that C is M-wellpowered. Under mild assumptions, the factorisation structure can be extended to cones for I. Being more concrete,

1. assume that C has products. Then define,

$$\mathcal{M} = \left\{ \text{all cones } (f_i : X \to \mathscr{D}(i))_{i \in \mathsf{I}} \text{ for } \mathsf{I} \text{ where } \langle f_i \rangle_{i \in I} : X \to \prod_{i \in \mathsf{I}} \mathscr{D}(i) \text{ is in } M \right\}$$

   The category C has an $(E, \mathcal{M})$-factorisation structure [AHS09, Proposition 15.19]. Moreover, it is $\mathcal{M}$-wellpowered: take the set of M-representants for $\prod_{i \in \mathsf{I}} \mathscr{D}(i)$ and post-compose them with the cone $(\pi_i : \prod_{i \in \mathsf{I}} \mathscr{D}(i) \to \mathscr{D}(i))_{i \in \mathsf{I}}$.

2. Assume that $\mathsf{I} = \{1 \rightrightarrows 2\}$. The class of cones,

$$\mathcal{M} = \{\text{all cones } (f_i : X \to \mathscr{D}(i))_{i \in \mathsf{I}} \text{ for } \mathsf{I} \text{ with } f_1 \text{ in } M\},$$

   also provides an $(E, \mathcal{M})$-factorisation structure of cones for I.

Together with the previous theorems, the last construction yields the following result.

Theorem 5.1.14. *Let $F : \mathsf{C} \to \mathsf{C}$ be an endofunctor over a cocomplete category C. If C is regularly wellpowered, has an (Epi, RegMono)-factorisation structure and $F : \mathsf{C} \to \mathsf{C}$ preserves regular monomorphisms, then* $\mathsf{CoAlg}\,(F)$ *has equalisers.*

Proof. Let $\mathsf{I} = \{1 \rightrightarrows 2\}$ and use the previous construction to provide an $(E, \mathcal{M})$-factorisation structure of cones for I. In order to show that the category C is $\mathcal{M}$-wellpowered let us take an equaliser diagram $\mathscr{D} : \mathsf{I} \to \mathsf{C}$ and then the set of M-subobjects of $\mathscr{D}(1)$. A simple reasoning now proves that C is $\mathcal{M}$-wellpowered. Finally, to prove that F sends cones in $\mathcal{M}$ to cones in $\mathcal{M}$, one just needs to show that it preserves regular monomorphisms but this is part of the assumption. Now apply Theorem 5.1.13. □

This last theorem shows that Hughes' assumption on C having equalisers is not necessary. Note also that Theorem 5.1.13 (from which we derived Theorem 5.1.14) holds in a broader context than Hughes' result, since it allows to prove the existence not only of equalisers but of any type of limit. In the sequel, we will take advantage of this generalisation.

Next, we introduce another result on limits in categories of coalgebras. It requires the notion of a *topological functor* [AHS09, Definition 21.1], which is detailed below.

Definition 5.1.15. Let $F : \mathsf{A} \to \mathsf{B}$ be a functor. A cone $\mathcal{C} = (X \to Y_i)_{i \in I}$ in A is said to be *initial with respect to* F if for every cone $\mathcal{D} = (Z \to Y_i)_{i \in I}$ and every morphism $h : FZ \to FX$ such that $F\mathcal{D} = F\mathcal{C} \cdot h$, there exists a unique A-morphism $\bar{h} : Z \to X$ such that $\mathcal{D} = \mathcal{C} \cdot \bar{h}$ and $h = F\bar{h}$.

We simply say that the cone is initial whenever no ambiguities arise.

Examples 5.1.16. Let us consider some simple examples.

1. A cone $(f_i : X \to X_i)$ in Top is initial with respect to the forgetful functor $\mathsf{Top} \to \mathsf{Set}$ if and only if X is equipped with the so called initial (weak) topology. Explicitly, the topology generated by the subbasis,
$$f_i^{-1}(U) \qquad (i \in I, U \subseteq X_i \text{ open})$$
The subbasis is also a basis if the cone is codirected.

2. Let Meas be the category of measurable spaces and measurable maps. A cone $(f_i : X \to X_i)$ in Meas is initial with respect to the forgetful functor $\mathsf{Meas} \to \mathsf{Set}$ if and only if X is equipped with the σ-algebra generated by,
$$f_i^{-1}(A) \qquad (i \in I, A \subseteq X_i \text{ measurable})$$

3. Let Ord be the category of preordered sets and monotone maps. A cone $(f_i : X \to X_i)$ in this category is initial with respect to the forgetful functor $\mathsf{Ord} \to \mathsf{Set}$ if and only if the following condition holds for every $x_1, x_2 \in X$: if for all $i \in I$, $f_i(x_1) \leq f_i(x_2)$ then $x_1 \leq x_2$.

4. A monocone ([AHS09, Definition 10.5]) in the category $\mathsf{CompHaus}$ of compact Hausdorff spaces and continuous maps is initial in Top (*cf.* [Gou13, Theorem 4.4.27]). Interestingly, the converse also holds, as a initial cone in Top whose domain is a T_0 space is necessarily mono.

Theorem 5.1.17 ([AHS09, Proposition 13.15])**.** *Let $F : \mathsf{A} \to \mathsf{B}$ be a limit preserving faithful functor and $\mathscr{D} : \mathsf{I} \to \mathsf{A}$ a diagram. A cone $\mathcal{C}$ for $\mathscr{D}$ is a limit of $\mathscr{D}$ if and only if the cone $F\mathcal{C}$ is a limit of $F\mathscr{D}$ and $\mathcal{C}$ is initial with respect to F.*

Definition 5.1.18. A functor $U : \mathsf{A} \to \mathsf{B}$ is called *topological* if every cone $\mathcal{C} = (X \to UX_i)_{i \in I}$ in B has a U-initial lifting, *i. e.* an initial cone $\mathcal{D} = (A \to X_i)_{i \in I}$ with respect to $U : \mathsf{A} \to \mathsf{B}$ such that $\mathcal{C} = U\mathcal{D}$.

Example 5.1.19. The canonical forgetful functors of the categories Top, Meas, and Ord are topological. The forgetful functor of the category of pseudometric spaces and contractions is also topological [AHS09, Examples 21.2].

As discussed in [AHS09, Remark 21.4, Theorem 21.16], topological functors are extremely well-behaved. Among other things, they are both left and right adjoints, faithful, and lift (co)limits.

Next, we will show that it is frequently possible to lift topological functors to categories of coalgebras; *i. e.* for a topological functor $U : \mathsf{A} \to \mathsf{B}$ and two functors $\overline{F} : \mathsf{A} \to \mathsf{A}$ and $F : \mathsf{B} \to \mathsf{B}$ that make the diagram below commute,

$$\begin{tikzcd} \mathsf{A} \arrow[r, "\overline{F}"] \arrow[d, "U"'] & \mathsf{A} \arrow[d, "U"] \\ \mathsf{B} \arrow[r, "F"'] & \mathsf{B} \end{tikzcd}$$

we will show that under certain conditions there exists another topological functor,

$$\mathsf{CoAlg}\left(\overline{F}\right) \to \mathsf{CoAlg}\,(F)$$

In the following section, we will use this result to prove that all categories of polynomial coalgebras over Top are complete. Let us start with the following proposition.

Proposition 5.1.20. *Consider a topological functor $U : \mathsf{A} \to \mathsf{B}$, two functors $\overline{F} : \mathsf{A} \to \mathsf{A}$ and $F : \mathsf{B} \to \mathsf{B}$, and a natural transformation $\delta : U\overline{F} \to FU$. This induces a functor $\overline{U} : \mathsf{CoAlg}\left(\overline{F}\right) \to \mathsf{CoAlg}\,(F)$ defined by the equations,*

$$\overline{U}(X, c) = (UX, \delta_X \cdot Uc), \qquad \overline{U}f = Uf$$

that makes the diagram below commute.

$$\begin{tikzcd} \mathsf{CoAlg}\left(\overline{F}\right) \arrow[r] \arrow[d, "\overline{U}"'] & \mathsf{A} \arrow[d, "U"] \\ \mathsf{CoAlg}\,(F) \arrow[r] & \mathsf{B} \end{tikzcd}$$

If the functor $U : \mathsf{A} \to \mathsf{B}$ is faithful, the induced functor $\overline{U} : \mathsf{CoAlg}\left(\overline{F}\right) \to \mathsf{CoAlg}\,(F)$ is faithful as well.

Lemma 5.1.21. *Assume that the natural transformation $\delta : \overline{F}U \to UF$ is mono and that U is faithful. Let $(f_i : (X, c) \to (Y_i, d_i))_{i\in I}$ be a cone in $\mathsf{CoAlg}\left(\overline{F}\right)$, and $(f_i : X \to Y_i)_{i\in I}$ be initial with respect to $U : \mathsf{A} \to \mathsf{B}$. The cone $(f_i : (X, c) \to (Y_i, d_i))_{i\in I}$ is initial with respect to $\overline{U} : \mathsf{CoAlg}\left(\overline{F}\right) \to \mathsf{CoAlg}\,(F)$.*

Proof. In Appendix A. □

Theorem 5.1.22. *Assume that $\overline{F} : \mathsf{A} \to \mathsf{A}$ preserves initial cones and that the equation $U\overline{F} = FU$ holds. If $U : \mathsf{A} \to \mathsf{B}$ is topological, the induced functor $\overline{U} : \mathsf{CoAlg}\left(\overline{F}\right) \to \mathsf{CoAlg}\,(F)$ is topological as well.*

Proof. Take a cone $(f_i : (X, c) \to \overline{U}(Y_i, d_i))_{i\in I}$ in $\mathsf{CoAlg}\,(F)$. Since $U : \mathsf{A} \to \mathsf{B}$ is topological, the induced cone $(f_i : X \to UY_i)_{i\in I}$ admits a U-initial lifting,

$$(\overline{f}_i : A \to Y_i)_{i\in I}$$

The cone $\overline{F}(\overline{f}_i : A \to Y_i)_{i\in I}$ is initial (by assumption) and the following equations hold.

$$U(A \overset{\overline{f}_i}{\to} Y_i \overset{d_i}{\to} \overline{F}Y_i) = (X \overset{f_i}{\to} UY_i \overset{Ud_i}{\to} FUY_i)$$
$$U(\overline{F}A \overset{\overline{F}\,\overline{f}_i}{\to} \overline{F}Y_i) = (FX \overset{Ff_i}{\to} FUY_i)$$

Therefore, we have the factorisation below.

$$\begin{tikzcd} X \arrow[d, "c"'] \arrow[dr, "Ud_i \cdot f_i"] & \\ FX \arrow[r, "Ff_i"'] & FUY_i \end{tikzcd}$$

This provides an arrow $\overline{c} : A \to \overline{F}A$ such that $U\overline{c} = c$ and the diagram below commutes.

$$\begin{tikzcd} A \arrow[d, "\overline{c}"'] \arrow[dr, "d_i \cdot \overline{f}_i"] & \\ \overline{F}A \arrow[r, "\overline{F}\,\overline{f}_i"'] & \overline{F}Y_i \end{tikzcd}$$

Thus, we have a cone $(\overline{f}_i : (A, \overline{c}) \to (Y_i, d_i))_{i \in I}$ in $\mathsf{CoAlg}\,(\overline{F})$. To finish the proof recall that the cone $(\overline{f}_i : A \to Y_i)_{i \in I}$ is initial with respect to $U : \mathsf{A} \to \mathsf{B}$ and apply Lemma 5.1.21. □

Corollary 5.1.23. *Consider a topological functor $U : \mathsf{A} \to \mathsf{B}$ and two functors $\overline{F} : \mathsf{A} \to \mathsf{A}$ and $F : \mathsf{B} \to \mathsf{B}$. Assume that $\overline{F} : \mathsf{A} \to \mathsf{A}$ preserves initial cones and that the equation $U\overline{F} = FU$ holds. Under these conditions, the category $\mathsf{CoAlg}\,(\overline{F})$ is complete iff $\mathsf{CoAlg}\,(F)$ is complete.*

We already mentioned that in the next section we will use Theorem 5.1.22 – and in particular the last corollary – to prove that all categories of polynomial coalgebras over Top are complete. Despite our restriction to Top, it will become clear that it is possible to obtain analogous results for other well-known categories, such as the one of preordered sets or the one of measurable spaces (see Examples 5.1.22).

As a conclusion of this section, we will introduce a last result on limits in categories of coalgebras. More concretely, we will list a set of conditions under which a functor,

$$\mathsf{CoAlg}\,(F) \to \mathsf{CoAlg}\,(G)$$

is left adjoint. We will also see that often this functor is fully faithful, and when such is the case $\mathsf{CoAlg}\,(F)$ is as complete as $\mathsf{CoAlg}\,(G)$. We start with the definition below.

Definition 5.1.24. Consider a natural transformation $\sigma : F \to G$. It induces a faithful functor $\mathsf{CoAlg}\,(F) \to \mathsf{CoAlg}\,(G)$, defined by,

$$(X, c) \mapsto (X, \sigma_X \cdot c), \qquad\qquad f \mapsto f$$

Proposition 5.1.25. *If $\sigma : F \to G$ is a monomorphic natural transformation, the induced functor $I : \mathsf{CoAlg}\,(F) \to \mathsf{CoAlg}\,(G)$ is full.*

Proof. Take a homomorphism $f : I(X, c) \to I(Y, d)$. By assumption, the equation $Gf \cdot \sigma_X \cdot c = \sigma_Y \cdot d \cdot f$ holds. Then, by naturality and the fact that $\sigma_Y : FY \to GY$ is a monomorphism we obtain $Ff \cdot c = d \cdot f$. □

Assumption 5.1.26. In the rest of this section, let C be a cocomplete category with an (E, M)-factorisation structure with M included in the class of monomorphisms. Assume that C is M-wellpowered, that G preserves M-morphisms, and that $\sigma : F \to G$ is a natural transformation whose components σ_X are in M.

Theorem 5.1.27. *Under Assumption 5.1.26, the functor $I : \mathsf{CoAlg}\,(F) \to \mathsf{CoAlg}\,(G)$ is left adjoint.*

Proof. We will show that the assumptions of the General Adjoint Functor Theorem hold. Since C is cocomplete, the category $\mathsf{CoAlg}\,(F)$ is cocomplete as well. Moreover, $I : \mathsf{CoAlg}\,(F) \to \mathsf{CoAlg}\,(G)$ preserves colimits, because $UI : \mathsf{CoAlg}\,(F) \to \mathsf{C}$ preserves colimits and the forgetful functor $U : \mathsf{CoAlg}\,(G) \to \mathsf{C}$ reflects them. It remains to verify the Solution Set Condition. For this, take a coalgebra $d : Y \to GY$. Let $\mathcal{S}_0$ be a set of representatives of the collection of all C-objects Q admitting an M-morphism $Q \to Y$, and let $\mathcal{S}$ be the set of all F-coalgebras based on an object in $\mathcal{S}_0$. Let now (X, c) be an F-coalgebra and $f : (X, \sigma_X \cdot c) \to (Y, d)$ be a homomorphism of G-coalgebras. By hypothesis, $f : X \to Y$ factorises as $f = m \cdot e$,

$$X \xrightarrow{e} Q \xrightarrow{m} Y$$

with $e \in E$ and $m \in M$. Since $\sigma_Q : FQ \to GQ$ and $Gm : GQ \to GY$ are in M, there exists a diagonal $q : Q \to FQ$ so that the right hand square and the lower-left square in the diagram below commute.

$$\begin{tikzcd}
GX \arrow[r, "Ge"] & GQ \arrow[r, "Gm"] & GY \\
FX \arrow[u, "\sigma_X"] \arrow[r, "Fe"'] & FQ \arrow[u, "\sigma_Q"'] & \\
X \arrow[u, "c"] \arrow[r, "e"'] & Q \arrow[u, dotted, "q"'] \arrow[r, "m"'] & Y \arrow[uu, "d"']
\end{tikzcd}$$

commute. The upper-left square also commutes because σ is a natural transformation. It follows that $f : (X, \sigma_X \cdot c) \to (Y, d)$ factorises via the image of an object in $\mathcal{S}$. □

Corollary 5.1.28. *The category of coalgebras* $\mathsf{CoAlg}\,(F)$ *has limits of a certain type if* $\mathsf{CoAlg}\,(G)$ *does so.*

Proof. Since $\mathsf{CoAlg}\,(F) \to \mathsf{CoAlg}\,(G)$ is fully faithful, its image is a full subcategory X of $\mathsf{CoAlg}\,(G)$ that is equivalent to $\mathsf{CoAlg}\,(F)$. The functor is also left adjoint, which means that X is coreflective and therefore has all limits that $\mathsf{CoAlg}\,(G)$ has. The claim then follows from X and $\mathsf{CoAlg}\,(F)$ being equivalent. □

By assuming some additional properties in Theorem 5.1.27, we can obtain a direct and intuitive way of building the coreflection of a G-coalgebra (Y, d): intuitively, it boils down to taking the largest M-subcoalgebra of (Y, d) that factorises as an F-coalgebra. We will make this idea precise via the notion of M-taut natural transformation [Möb83; Man02].

Definition 5.1.29. A natural transformation $\sigma : F \to G$ is *M-taut* if each naturality square induced by a morphism in M is a pullback square. In other words, if for every morphism $m : X \to Y$ in M the diagram below is a pullback square.

$$\begin{tikzcd}
FX \arrow[r, "Fm"] \arrow[d, "\sigma_X"'] & FY \arrow[d, "\sigma_Y"] \\
GX \arrow[r, "Gm"'] & GY
\end{tikzcd}$$

Recall from Definition 4.1.12 that, for monomorphisms $m_1 : M_1 \to X$ and $m_2 : M_2 \to X$, m_1 is smaller than m_2 whenever there exists some $m : M_1 \to M_2$ with $m_2 \cdot m = m_1$. Then assuming that C has pullbacks, take a G-coalgebra (Y, d) and consider the following pullback square in C.

$$\begin{tikzcd}
S \arrow[r] \arrow[d, "i"'] & FY \arrow[d, "\sigma_Y"] \\
Y \arrow[r, "d"'] & GY
\end{tikzcd} \tag{13}$$

According to [AHS09, Proposition 14.15, second item of the proof], the morphism $i : S \to Y$ lives in M. Intuitively, S is the largest subobject of Y whose 'next states' are contained in FY.

Lemma 5.1.30. *Assume that the natural transformation $\sigma : F \to G$ is M-taut and let $m : (Q, q) \to (Y, d)$ be a homomorphism in $\mathsf{CoAlg}\,(G)$ where $m \in M$ and $m \leq i$. Then there exists an F-coalgebra structure $q' : Q \to FQ$ on Q with $\sigma_Q \cdot q' = q$.*

Proof. Let $\bar{m} : Q \to S$ be the arrow in C with $i \cdot \bar{m} = m$. Then, since in the diagram,

$$\begin{tikzcd}
& Y \arrow[rr, "d"] & & GY \\
S \arrow[ur, "i"] \arrow[rr] & & FY \arrow[ur, "\sigma_Y"] & \\
& & & GQ \arrow[uu, "Gm"'] \\
Q \arrow[uu, "\bar{m}"] \arrow[rr, dotted] \arrow[uuur, bend left, "m"] \arrow[urrr, bend right, "q"'] & & FQ \arrow[uu, "Fm"'] \arrow[ur, "\sigma_Q"] &
\end{tikzcd}$$

the right hand parallelogram is a pullback square and the outer diagram and the top parallelogram commute, we can obtain the desired arrow $Q \to FQ$. $\square$

Theorem 5.1.31. *In addition to Assumption 5.1.26, let C have pullbacks and σ be M-taut. The coreflection of a G-coalgebra (Y, d) is the supremum of all G-subcoalgebras $m : (Q, q) \rightarrowtail (Y, d)$ such that $m : Q \rightarrowtail Y$ is smaller than $i : S \to Y$ (as defined by the pullback square (13)).*

Proof. In Appendix A. $\square$

5.2 LIMITS IN CATEGORIES OF VIETORIS COALGEBRAS

5.2.1 *Representations meet Topology*

In Section 4.2, we saw that representations of hybrid black-box machines can be naturally interpreted as coalgebras typed as $M \times \mathbb{R}^n \to M \times \mathbb{R}^n \times \mathrm{H}(\mathbb{R}^n)$ if the following condition is assumed: *as soon as* the guard of a given mode is satisfied, the mode must switch. We showed that in order to drop this condition one would need to consider instead coalgebras of the type $M \times \mathbb{R}^n \to \mathrm{P}(M \times \mathbb{R}^n \times \mathrm{H}(\mathbb{R}^n))$ as the models of representations. Such coalgebras, however, do not admit a final coalgebra and so representations would lack a notion of observational behaviour which is essential for their analysis.

A possible way of fixing this, as detailed in the following section, is to adopt weaker versions of the assumption instead of completely dropping it. One particularly promising case is to force transitions to occur *not* as soon as the corresponding guard is satisfied, but within a compact, or more permissively, a closed interval of time. Under this strategy, the models of representations would naturally be seen as coalgebras over Top typed as,

$$M \times \mathbb{R}^n \to \mathrm{V}(M \times \mathbb{R}^n \times \mathrm{H_c}(\mathbb{R}^n)) \tag{14}$$

where $\mathrm{V} : \mathsf{Top} \to \mathsf{Top}$ is a Vietoris functor (detailed below) and $\mathrm{H_c}$ is a topological variant of the hybrid monad (to be introduced in the following section).

Of course for this approach to succeed, coalgebras of the type (14) must admit a final coalgebra. Moreover, if we also wish to consider F-representations (Section 4.4), then we will need to show that coalgebras typed as $M \times \mathbb{R}^n \to F\mathrm{V}(M \times \mathbb{R}^n \times \mathrm{H_c}(\mathbb{R}^n))$ have a final coalgebra as well.

Driven by this challenge, the main goal of the current section is to carefully examine the existence of limits in categories of coalgebras whose underlying functor is *Vietoris polynomial* – intuitively, the topological analogue to a Kripke polynomial functor . As compositions of constant, (co)product, identity, and powerset functors, Kripke polynomial ones have long been recognised as a particularly relevant class of functors [Rut00; BRS09; KKV04]. They are intuitive and the corresponding coalgebras subsume several types of state-based transition systems. Moreover, they are well-behaved with respect to the existence of limits in their categories of coalgebras if the powerset functor is subjected to certain cardinality restrictions. These reasons render their topological analogues a natural starting point for interpreting F-representations in Top.

Remark 5.2.1. Vietoris polynomial functors arise from the composition of different Vietoris functors [Vie22; Mic51; CT97] – the topological analogues to the powerset – with polynomial functors over Top. To keep the nomenclature simple, every coalgebra whose underlying functor is Vietoris polynomial will be called a *Vietoris coalgebra*.

We consider two instances of Vietoris functors: namely, the *lower Vietoris* and the *compact Vietoris*. Both have already been addressed in a coalgebraic setting: the former in [BKR07] and the latter in multiple documents [KKV04; BFV10; VV14; DDG16; BBH12]. Some of these

works study the existence of final coalgebras. For example, [KKV04] shows that compact Vietoris polynomial functors in the category Stone of Stone spaces and continuous maps admit a final coalgebra. Also, document [DDG16, Proposition B.4] presents a theorem that can be generalised to show that the compact Vietoris functor in the category $\mathsf{CompHaus}$ of compact Hausdorff spaces and continuous maps, admits a final coalgebra. In fact, this generalised result is also implicitly mentioned in [Eng89, page 245].
Since the current section is relatively big, and rather theoretical, we summarise our main findings next.

- Using the results in Section 5.1, we show that all categories of (sub)polynomial coalgebras over Top are complete,
- and that the same applies to all categories of compact Vietoris coalgebras over $\mathsf{CompHaus}$.
- Using [Zen70, Lemma B], we show that all categories of compact Vietoris coalgebras are complete in the category Haus of Hausdorff spaces and continuous maps.
- We take advantage of the limit-preserving properties of the inclusion functors $\mathsf{CompHaus} \to \mathsf{Top}$ and $\mathsf{Haus} \to \mathsf{Top}$ to show that every compact Vietoris polynomial functor over Top that can be restricted either to $\mathsf{CompHaus}$ or Haus admits a final coalgebra.
- We show that all categories of lower Vietoris coalgebras over the category $\mathsf{StablyComp}$ of stably compact spaces and spectral maps are complete.

5.2.2 *Polynomial coalgebras over* Top *and a brief review of Vietoris functors*

Definition 5.2.2. Let C be a category with (co)products. An endofunctor in C is called *polynomial* if it can be defined recursively from the grammar below,

$$F \ni F + F \mid F \times F \mid A \mid \mathrm{Id}$$

where A is a C-object.

Alternatively, one can define the class of polynomial functors as the smallest class of endofunctors in C that contains the identity, all constant functors, and is closed under products and sums. Given two functors $F, G : \mathsf{C} \to \mathsf{C}$, their product and their sum are defined, respectively, as the compositions,

$$\mathsf{C} \xrightarrow{\langle F,G \rangle} \mathsf{C} \times \mathsf{C} \xrightarrow{(\times)} \mathsf{C} \qquad\qquad \mathsf{C} \xrightarrow{\langle F,G \rangle} \mathsf{C} \times \mathsf{C} \xrightarrow{(+)} \mathsf{C}$$

Theorem 5.2.3. *Let $F : \mathsf{Top} \to \mathsf{Top}$ be a (sub)polynomial functor. Its category of coalgebras is complete.*

Proof. Clearly, the identity and constant functors preserve initial cones. The product also $(\times) : \mathsf{Top} \times \mathsf{Top} \to \mathsf{Top}$ preserves initial cones by [HST14, page 141], and it is straightforward to show that the sum $(+) : \mathsf{Top} \times \mathsf{Top} \to \mathsf{Top}$ preserves initial cones as well. Now apply Theorem 5.1.22 and Theorem 5.1.27. $\square$

In Set, the powerset functor is a standard construction for bringing non-deterministic behaviour into the scene. In the topological context, the situation is more complex, because a number of functors can be seen as analogues to the powerset. Most of them have their roots in the Hausdorff metric [Pom05; Hau14] and in Vietoris' "Bereiche zweiter Ordnung" [Vie22] whose powerset construction is detailed next.

Consider a compact Hausdorff space X. The classical Vietoris space $\mathrm{V}X$ [Vie22] is the set of all closed subsets of X, $\mathrm{V}X = \{K \subseteq X \mid K \text{ is closed}\}$, equipped with the hit-and-miss topology, generated by the sets,

$$U^{\Diamond} = \{A \in \mathrm{V}X \mid A \cap U \neq \varnothing\} \qquad (\text{“}A \text{ hits } U\text{”})\,,$$
$$U^{\Box} = \{A \in \mathrm{V}X \mid A \subseteq U\} \qquad (\text{“}A \text{ misses } X \setminus U\text{”}),$$

where $U \subseteq X$ is open. There exist several variants of this archetype [Mic51; CT97], but for now let us concentrate on the two cases detailed below, which are based on the category of topological spaces and continuous maps.

Examples 5.2.4. Let X be a topological space and $f : X \to Y$ be a continuous map.

1. Define $\mathrm{V}X = (\{K \subseteq X \mid K \text{ is compact }\}, \text{hit-and-miss topology on } X)$ and the continuous map $\mathrm{V}f : \mathrm{V}X \to \mathrm{V}Y$ by $\mathrm{V}f(A) = f[A]$. We call this variant *compact Vietoris.*

2. Define $\mathrm{V}X = (\{K \subseteq X \mid K \text{ is closed}\}, \text{hit topology on } X)$ and the continuous map $\mathrm{V}f : \mathrm{V}X \to \mathrm{V}Y$ by $\mathrm{V}f(A) = \overline{f[A]}$, where $\overline{f[A]}$ denotes the closure of $f[A]$. This variant is frequently called *lower Vietoris* [CLP91; Pet96; BKR07].

Remark 5.2.5. Adding the sets $U^{\Box}$ to the subbasis of Example 5.2.4 (2) does not define a functor: consider the set $\{1,2,3\}$ equipped with the topology generated by the sets $\{1,2\}$ and $\{2,3\}$. For the subspace embedding $i : \{1,2\} \hookrightarrow \{1,2,3\}$, $(\mathrm{V}i)^{-1}(\{1,2\}^{\Box}) = \{\varnothing, \{1\}\}$. However, every open set of $\mathrm{V}\{1,2\}$ that contains $\{1\}$ contains $\{1,2\}$.

Definition 5.2.6. Let $\mathrm{V} : \mathsf{Top} \to \mathsf{Top}$ be the lower Vietoris functor. We qualify an endofunctor in the category Top as *lower Vietoris polynomial* if it can be recursively defined from the grammar below. If we consider instead the compact Vietoris functor $\mathrm{V} : \mathsf{Top} \to \mathsf{Top}$, the endofunctor is called *compact Vietoris polynomial.*

$$F \ni F + F \mid F \times F \mid A \mid \mathrm{Id} \mid \mathrm{V}$$

5.2.3 *Vietoris Coalgebras*

Theorem 5.2.3, which shows that a category of (sub)polynomial coalgebras over Top is always complete, is a positive result in our study. When we add Vietoris functors to the mix, however, a whole new level of difficulty arises, which calls for an investigation of the preservation properties of these functors and in particular the preservation of codirected limits (Theorem 5.1.2). This is the goal of the current subsection.

We start with the following lemma, which will help us to introduce some results on the preservation of codirected cones by Vietoris functors.

Lemma 5.2.7. *Let X be a topological space and $\mathcal{B}$ be a basis for the topology of X.*

1. *The set $\{B^{\Diamond} \mid B \in \mathcal{B}\}$ is a subbasis for the lower Vietoris space* $\mathrm{V}X$*.*
2. *If $\mathcal{B}$ is closed under finite unions, the set $\{B^{\Diamond} \mid B \in \mathcal{B}\} \cup \{B^{\Box} \mid B \in \mathcal{B}\}$ is a subbasis for the compact Vietoris space* $\mathrm{V}X$*.*

Proof. Let $\mathcal{S}$ be a set of open subsets of X. First note that, for both the lower and the compact Vietoris space,

$$\left(\bigcup \mathcal{S}\right)^{\Diamond} = \bigcup \left\{S^{\Diamond} \mid S \in \mathcal{S}\right\}$$

This proves the first statement. To see that the second one is also true, observe that,

$$\left(\bigcup \mathcal{S}\right)^{\Box} = \bigcup \left\{\left(\bigcup \mathcal{F}\right)^{\Box} \mid \mathcal{F} \subseteq \mathcal{S} \text{ finite}\right\}$$

since we only consider compact subsets of X. □

Lemma 5.2.8. *Both the compact and the lower Vietoris functor* $\mathrm{V} : \mathsf{Top} \to \mathsf{Top}$ *preserve initial codirected cones.*

Proof. Let $(f_i : X \to X_i)_{i \in I}$ be an initial codirected cone in Top. The set,

$$\left\{f_i^{-1}(U) \mid i \in I, U \subseteq X_i \text{ open}\right\}$$

is a basis for the topology of X (Examples 5.1.16) and it is closed under finite unions. Therefore, by the previous lemma, the proof follows from the equations,

$$((f_i)^{-1}(U))^{\Box} = (\mathrm{V}f_i)^{-1}(U^{\Box}) \qquad\qquad ((f_i)^{-1}(U))^{\Diamond} = (\mathrm{V}f_i)^{-1}(U^{\Diamond}),$$

for all $i \in I$ and $U \subseteq X_i$ open, which are straightforward to show. □

Neither the compact nor the lower Vietoris functor preserve monocones in general. Take, for example, a compact Hausdorff space X with at least two elements. The set $A = \{(x, x) \mid x \in X\}$ is closed in $X \times X$ and it is different from $B = X \times X$. Then,

$$\mathrm{V}\pi_1(A) = \mathrm{V}\pi_1(B) = X = \mathrm{V}\pi_2(A) = \mathrm{V}\pi_2(B)$$

which shows that the cone $\mathrm{V}(\pi_i : X \times X \to X)_{i \in 2}$ is not mono. However,

Theorem 5.2.9. *The lower Vietoris functor preserves initial codirected monocones. The compact Vietoris functor preserves initial codirected monocones of Hausdorff spaces.*

Proof. For a topological space X the lower Vietoris space $\mathrm{V}X$ is T_0 and if X is Hausdorff the compact Vietoris space $\mathrm{V}X$ is Hausdorff as well [Mic51, Theorem 4.9.8]. Then note that a initial cone in Top whose domain is T_0 (or T_2) is necessarily mono: for a cone $f_i : X \to X_i$ with X a T_0-space and two elements $x \neq y \in X$ we can find $x \in f_i^{-1}(U)$ and $y \notin f_i^{-1}(U)$ for some $i \in I$ and open $U \subseteq X_i$. This means that $f_i(x) \in U$ and $f_i(y) \notin U$ and therefore $f_i(x) \neq f_i(y)$. Finally, apply Lemma 5.2.8. $\square$

As shown by the following theorem, the two last results provide a clear picture with respect to the existence of equalisers in categories of Vietoris coalgebras.

Theorem 5.2.10. *The category of coalgebras of a Vietoris polynomial functor always has equalisers.*

Proof. All polynomial functors preserve regular monomorphisms (Lemma A.2.9) and the lower Vietoris functor preserves them as well (Theorem 5.2.9). The compact Vietoris functor preserves initial morphisms (Lemma 5.2.8) and we can show that it preserves monomorphisms $f : X \to Y$ in the following manner: take two subsets $A, B \subseteq X$ such that $A \neq B$. We can assume the existence of an element $a \in A$ such that $a \notin B$. Then $f(a) \notin f[B]$ because every element $b \in B$ is different than a which entails that $f(a) \neq f(b)$. Finally, an application of Theorem 5.1.14 proves our claim. $\square$

But what we really need, is to prove the existence of final coalgebras rather than of equalisers. The traditional approach for this is to show that the Vietoris polynomial functors preserve limits of ω^{op}-chains, or equivalently, codirected limits [AR94, Corollary 1.7]. They turn out to not preserve such limits, as demonstrated by the following examples.

Examples 5.2.11. Neither the compact Vietoris functor nor the lower one preserve codirected limits. For example,

1. Consider the diagram functor $\mathscr{D} : \mathbb{N} \to \mathsf{Set}$ that sends $n \leq m$ to the inclusion map $\{0, \dots, n\} \hookrightarrow \{0, \dots, m\}$. The set $\mathbb{N}$ is a colimit of this directed diagram and, since contravariant hom-functors send colimits into limits, the composition $\hom(-, \mathbb{N}) \cdot \mathscr{D}^{\mathsf{op}} : \mathbb{N}^{\mathsf{op}} \to \mathsf{Set}$ defines a codirected diagram with limit $\hom(\mathbb{N}, \mathbb{N})$ whose projections $\pi_i : \hom(\mathbb{N}, \mathbb{N}) \to \mathsf{Set}(\{0, \dots, i\}, \mathbb{N})$ restrict the domain of a given function. To equip all sets with the indiscrete topology provides a codirected limit in Top. The compact Vietoris functor, however, does not send this limit to a monocone, because the projections $(\mathrm{V}\pi_i)_{i \in \mathbb{N}}$ cannot distinguish between the set $\hom(\mathbb{N}, \mathbb{N})$ and,

$$\{f : \mathbb{N} \to \mathbb{N} \mid \{n \in \mathbb{N} \mid f(n) \neq 0\} \text{ is finite}\}$$

This proves that the compact Vietoris functor does not preserve ω^{op}-limits.

2. The next example is based on the ‘empty inverse limit’ [Wat72]. Let I be the set of all finite subsets of $\mathbb{R}$ equipped with the usual order; note that it is directed. Then, let $\mathscr{D} : \mathsf{I}^{\mathsf{op}} \to \mathsf{Top}$ be the diagram functor that sends $\mathscr{D}(n)$ to the discrete space of all injective functions $n \to \mathbb{N}$ and $n \subseteq m$ to the map $\mathscr{D}(m) \to \mathscr{D}(n)$ that restricts the domain of a given function. The limit of this diagram is empty, because if it had an element it would define an injective function $\mathbb{R} \to \mathbb{N}$. Note also that each map $\mathscr{D}(n \subseteq m)$ is surjective. These two facts entail that the lower Vietoris functor does not send the limit of $\mathscr{D}$ to the limit of $\mathrm{V}\mathscr{D}$: the former is empty and the latter has at least two elements, namely $(\varnothing)_{n\in \mathsf{I}}$ and $(\mathscr{D}(n))_{n\in\mathsf{I}}$.

   Using the indiscrete topology *in lieu* of the discrete one, shows that the lower Vietoris functor does not preserve codirected limits of diagrams of compact spaces and closed maps.

3. The previous example holds even when one considers spaces that are T_0, compact, and locally compact: consider the natural numbers $\mathbb{N}$ equipped with the topology,

$$\{\uparrow n \mid n \in \mathbb{N}\} \cup \{\varnothing\}$$

   where $\uparrow n = \{k \in \mathbb{N} \mid n \leq k\}$. The space $\mathbb{N}$ is T_0 and every non-empty collection of open subsets of $\mathbb{N}$ has a largest element with respect to inclusion $\subseteq$. This implies that, for every finite set n, every subset of $\mathbb{N}^n$ is compact. To see why, recall that the topology of $\mathbb{N}^n$ is the initial one for the projections $(\pi_i : \mathbb{N}^n \to \mathbb{N})_{i\leq n}$. Then consider a subset C of $\mathbb{N}^n$ and assume that it is covered by the subbasic sets,

$$C \subseteq \bigcup_{\lambda\in\Lambda} \pi_{a_\lambda}^{-1}(\uparrow b_\lambda)$$

   For every natural number $i \leq n$, the set $\left\{\pi_{a_\lambda}^{-1}(\uparrow b_\lambda) \mid \lambda \in \Lambda \text{ such that } a_\lambda = i\right\}$ has a largest element which we denote by m_i. This proves that,

$$C \subseteq \bigcup_{i\leq n} m_i$$

   which, according to Alexander’s Subbase Theorem [Gou13, Theorem 4.4.29][1], entails that the set C is compact.

   Define I as in the previous example. Then, let $\mathscr{D} : \mathsf{I}^{\mathsf{op}} \to \mathsf{Top}$ be the diagram functor that sends a natural number n to the subspace of $\mathbb{N}^n$ whose functions are injective, and $n \leq m$ to the map $\mathscr{D}(m) \to \mathscr{D}(n)$ that restricts the domain of a given function. The limit of $\mathscr{D}$ still is the empty set and the limit of $\mathrm{V}\mathscr{D}$ still contains the elements $(\varnothing)_{n\in\mathsf{I}}$ and $(\mathscr{D}(n))_{n\in\mathsf{I}}$.

These examples show that it is highly problematic to consider all topological spaces, because the lower and the compact Vietoris functors do not preserve codirected limits in Top. We will, therefore, restrict our attention to different subcategories, where more positive results appear.

Let us start by establishing some conventions on notation, nomenclature, and also recall some well-known definitions: consider a subset $S \subseteq X$ of a partially ordered space X. We denote

1 Note that it requires the axiom of choice.

the upper-closure of S by $\uparrow S$ and similarly for the down-closure. We will also consider partially ordered spaces X equipped with a topology. In this context, a subset $S \subseteq X$ is called upper-open if it is simultaneously an upper subset and open, and similarly for upper-closed subsets. Finally, recall that a subset $S \subseteq X$ of a topological space X is called *saturated* if S is a finite intersection of open subsets of X, equivalently if $\uparrow S = S$ with respect to the specialisation order of X [Law11].

Definition 5.2.12. A topological space is called *stably compact* whenever it is T_0, locally compact, well-filtered, and every finite intersection of compact saturated subsets is compact [Law11; Gou13]. A continuous map between stably compact spaces is called *spectral* whenever the inverse image of compact saturated subsets is compact. Stably compact spaces and spectral maps form a category which we denote by $\mathsf{StablyComp}$.

The reader will find in documents [Gie+03; Jun04; Law11; Gou13] detailed accounts of stably compact spaces.

Theorem 5.2.13. *The category* $\mathsf{StablyComp}$ *is complete and regularly wellpowered. The inclusion functor* $\mathsf{StablyComp} \to \mathsf{Top}$ *preserves limits and finite coproducts.*

Proof. Finite coproducts of stably compact spaces are stably compact as well [Gou13, Proposition 9.2.1]. The other claims hold because the inclusion functor $\mathsf{StablyComp} \to \mathsf{Top}$ is monadic [Sim82] and monadic functors create limits and detect wellpoweredness [AHS09, Proposition 20.12]. Note that [Sim82] uses the designation *well-compacted* instead of stably compact. $\square$

Further properties of the category $\mathsf{StablyComp}$ can be easily derived if ones takes an order-theoretic perspective.

Definition 5.2.14. A *partially ordered compact space* is a triple $(X, \leq, \tau)$ that consists of a set X, a partial order $\leq$ on X, and a compact topology τ on X, so that the set,

$$\{(x, y) \in X \times X \mid x \leq y\}$$

is closed with respect to the product topology.

Remark 5.2.15. Every partially ordered compact space $(X, \leq, \tau)$ is Hausdorff. This follows from the antisymmetry property of the relation $\leq$ and the continuity of the map $\langle \pi_2, \pi_1 \rangle : X \times X \to X \times X$: both entail that the diagonal $\{(x, x) \mid x \in X\}$ is closed in $X \times X$.

The category $\mathsf{StablyComp}$ is isomorphic to the category $\mathsf{PosComp}$ of partially ordered compact spaces and monotone continuous maps [Law11, Remark 2.12]. The isomorphism $\mathsf{PosComp} \to \mathsf{StablyComp}$ sends a partially ordered compact space $(X, \leq, \tau)$ to the stably compact space with the same underlying set and the topology defined by the upper-open sets of $(X, \leq, \tau)$. Its inverse functor uses the specialisation order of a topological space, defined by $x \leq y$ iff $x \in \overline{\{y\}}$. It maps a stably compact space (X, τ) into the space $(X, \tau', \leq)$ whose relation $\leq$ is the specialisation order and τ' is the *patch topology* of (X, τ), *i. e.* the topology generated by the complements of compact saturated subsets and also the opens in (X, τ).

The canonical forgetful functor $\mathsf{PosComp} \to \mathsf{CompHaus}$ has a left adjoint which equips a compact Hausdorff space with the discrete order. Using the isomorphism between $\mathsf{StablyComp}$ and $\mathsf{PosComp}$, the adjunction,

$$\begin{tikzcd} \mathsf{PosComp} \arrow[r, bend left, "\text{forgetful}"] \arrow[r, phantom, "\top"] & \mathsf{CompHaus} \arrow[l, bend left, "\text{discrete}"] \end{tikzcd}$$

then reads in the language of stably compact spaces as,

$$\begin{tikzcd} \mathsf{StablyComp} \arrow[r, bend left, "\text{patch}"] \arrow[r, phantom, "\top"] & \mathsf{CompHaus} \arrow[l, bend left, "\text{inclusion}"] \end{tikzcd}$$

In the sequel we will jump freely between both perspectives.

Theorem 5.2.16. *The category* $\mathsf{PosComp}$ *is cocomplete and has an (Epi,RegMono)-factorisation structure.*

Proof. The first claim follows directly from [Tho09, Corollary 2]. The second one follows from [HN16, Proposition 4.7, Corollary 4.8]. □

Let us turn our attention back to Vietoris functors. The lower Vietoris functor on Top can be restricted to a functor on $\mathsf{StablyComp}$ [Law11, Theorem 5.7]. Its counterpart on $\mathsf{PosComp}$ is described in the following manner.

Theorem 5.2.17. *The isomorphism* $\mathsf{StablyComp} \simeq \mathsf{PosComp}$ *and the lower Vietoris functor on* $\mathsf{StablyComp}$ *induce the functor,*

$$\mathsf{PosComp} \to \mathsf{PosComp}$$

that sends a partially ordered compact space X to the space of all lower-closed subsets of X, with inclusion $\subseteq$ as the order, and the topology generated by the sets,

$$\begin{aligned} &\{A \subseteq X \mid A \textit{ lower-closed and } A \cap U \neq \varnothing\} \quad (U \subseteq X \textit{ upper-open}), \\ &\{A \subseteq X \mid A \textit{ lower-closed and } A \cap K = \varnothing\} \quad (K \subseteq X \textit{ upper-closed}) \end{aligned} \tag{15}$$

For a map $f : X \to Y$ in $\mathsf{PosComp}$, *the functor returns the map that sends a lower-closed subset $A \subseteq X$ to the down-closure ${\downarrow} f[A]$.*

Proof. In Appendix A. □

The lower Vietoris functor preserves codirected initial monocones (Theorem 5.2.9), which entails that for every every codirected diagram functor $\mathscr{D} : \mathsf{I} \to \mathsf{StablyComp}$ with limit cone $(\pi_i : L_{\mathscr{D}} \to \mathscr{D}(i))_{i \in \mathsf{I}}$, the canonical comparison map,

$$h : \mathrm{V} L_{\mathscr{D}} \to L_{\mathrm{V}\mathscr{D}}, \qquad\qquad K \mapsto (\overline{\pi_i[K]})_{i \in I}$$

is a subspace embedding. To show that $\mathrm{V} : \mathsf{StablyComp} \to \mathsf{StablyComp}$ preserves these limits, we are only left with the task of proving that h is also surjective since the functor $\mathsf{StablyComp} \to \mathsf{Top}$ reflects isomorphisms [AHS09, Proposition 20.12]. To do so, we will use the fact that $\mathsf{StablyComp}$ inherits a nice characterisation of codirected limits from the category $\mathsf{CompHaus}$. This characterisation was first hinted in [Bou66, Proposition 8], but, to the best of our knowledge, is rarely used in the literature. Actually, we were not able to find a proof except for [Hof99] which is written in German. So we sketch one below.

Theorem 5.2.18. *Let $\mathscr{D} : \mathsf{I} \to \mathsf{CompHaus}$ be a codirected diagram and $(p_i : L \to \mathscr{D}(i))_{i \in \mathsf{I}}$ be a cone for $\mathscr{D}$. The following conditions are equivalent:*

1. *The cone $(p_i : L \to \mathscr{D}(i))_{i \in \mathsf{I}}$ is a limit of $\mathscr{D}$.*
2. *The cone $(p_i : L \to \mathscr{D}(i))_{i \in \mathsf{I}}$ is mono and for every $i \in \mathsf{I}$ the inequality below holds.*

$$\bigcap_{j \to i} \operatorname{img} \mathscr{D}(j \to i) \subseteq \operatorname{img} p_i$$

Proof. Assume that the cone $(p_i : L \to \mathscr{D}(i))_{i \in \mathsf{I}}$ satisfies condition (2) and consider a cone $(f_i : X \to \mathscr{D}(i))_{i \in \mathsf{I}}$ for $\mathscr{D}$. Take an element $x \in X$ and for every $i \in \mathsf{I}$ write $A_i = p_i^{-1}(f_i(x))$. Since $\mathscr{D}(i)$ is a T_1-space, the subset $A_i \subseteq X$ is closed. Moreover, it is non-empty because,

$$\operatorname{img} f_i \subseteq \bigcap_{j \to i} \operatorname{img} \mathscr{D}(j \to i) \subseteq \operatorname{img} p_i$$

Since the family $(A_i)_{i \in I}$ is codirected and L is compact, there exists some element $z \in \bigcap_{i \in I} A_i$. So write $f(x) = z$, and in this way we obtain a map $f : X \to L$ such that $p_i \cdot f = f_i$ for all $i \in I$. Since $(p_i : L \to \mathscr{D}(i))_{i \in \mathsf{I}}$ is a monocone it must be a limit of $\mathscr{D}$.

Conversely, if $(p_i : L \to \mathscr{D}(i))_{i \in \mathsf{I}}$ is a limit of $\mathscr{D}$ then it must also be a monocone. Consider two elements $i_0 \in \mathsf{I}$ and $x \in \bigcap_{j \to i_0} \operatorname{img} \mathscr{D}(j \to i_0)$. We may assume that i_0 is final in I [AR94, page 18]. Then for each $i \in \mathsf{I}$ define,

$$A_i = \left\{ (x_i)_{i \in \mathsf{I}} \in \prod_{i \in \mathsf{I}} \mathscr{D}(i) \mid x_{i_0} = x \text{ and for all } i \to j \in \mathsf{I}, x_j = \mathscr{D}(i \to j)(x_i) \right\}$$

The set A_i is non-empty, and, moreover, closed in $\prod_{i \in \mathsf{I}} \mathscr{D}(i)$ because it is an equaliser of continuous maps between Hausdorff spaces. It is routine to show that for every map $i \to j \in \mathsf{I}$ the inequality $A_i \subseteq A_j$ holds. Hence, there exists some element $z \in \bigcap_{i \in \mathsf{I}} A_i$, which, by construction, lives in L and respects the equation $p_{i_0}(z) = x$. □

For every cone $(p_i : X \to \mathscr{D}(i))_{i \in \mathsf{I}}$ the inequation $\operatorname{img} p_i \subseteq \bigcap_{j \to i} \operatorname{img} \mathscr{D}(j \to i)$ always holds. Therefore, according to the previous theorem, if the cone $(p_i : X \to \mathscr{D}(i))_{i \in \mathsf{I}}$ is a limit for $\mathscr{D}$ the following equation holds for every $i \in \mathsf{I}$,

$$\operatorname{img} p_i = \bigcap_{j \to i} \operatorname{img} \mathscr{D}(j \to i)$$

Proposition 5.2.19. *Let $\mathcal{A}$ be a codirected set of closed subsets of a partially ordered compact space X. Then, ${\downarrow}\bigcap_{A\in\mathcal{A}} A = \bigcap_{A\in\mathcal{A}} {\downarrow}A$.*

Proof. In Appendix A. □

Proposition 5.2.20. *Let $\mathscr{D} : \mathsf{I} \to \mathsf{PosComp}$ be a codirected diagram functor, $(\pi_i : L_{\mathscr{D}} \to \mathscr{D}(i))_{i\in\mathsf{I}}$ be a limit for $\mathscr{D}$, and $(L_{\mathrm{V}\mathscr{D}} \to \mathrm{V}\mathscr{D}(i))_{i\in\mathsf{I}}$ be a limit for $\mathrm{V}\mathscr{D} : \mathsf{I} \to \mathsf{PosComp}$. Then the function $h : \mathrm{V}L_{\mathscr{D}} \to L_{\mathrm{V}\mathscr{D}}$ defined by $K \mapsto ({\downarrow}\pi_i[K])_{i\in\mathsf{I}}$ is surjective.*

Proof. Consider an element $(K_i)_{i\in\mathsf{I}} \in L_{\mathrm{V}\mathscr{D}}$. For every $i \in \mathsf{I}$ the subset $K_i \subseteq \mathscr{D}(i)$ is closed, and therefore $K_i \in \mathsf{PosComp}$. Thus, consider the codirected diagram functor $\mathscr{K} : \mathsf{I} \to \mathsf{PosComp}$ defined by $\mathscr{K}(i) = K_i$ and $\mathscr{K}(i \to j)$ as the map induced by the restriction of $\mathscr{D}(i \to j)$ to $\mathscr{K}(i)$. Note that the equation,

$$ {\downarrow}\mathscr{K}(i \to j)[\mathscr{K}(i)] = \mathscr{K}(j) \tag{16}$$

holds for every $i \to j \in \mathsf{I}$.

Let $(p_i : L_{\mathscr{K}} \to \mathscr{K}(i))_{i\in\mathsf{I}}$ be a limit for $\mathscr{K}$. By construction, and since a compact subset of a Hausdorff space is closed, the subset $L_{\mathscr{K}} \subseteq L_{\mathscr{D}}$ is lower-closed and therefore $L_{\mathscr{K}} \in \mathrm{V}L_{\mathscr{D}}$. We claim that $h[L_{\mathscr{K}}] = (K_i)_{i\in\mathsf{I}}$. Let $i_0 \in \mathsf{I}$. Since the diagram of forgetful functors,

$$\begin{tikzcd}
\mathsf{PosComp} \arrow[rr] \arrow[dr] & & \mathsf{CompHaus} \arrow[dl] \\
& \mathsf{Set} &
\end{tikzcd}$$

commutes and $\mathsf{PosComp} \to \mathsf{CompHaus}$ preserves limits, Theorem 5.2.18 tells that the equation

$$p_{i_0}[L_{\mathscr{K}}] = \bigcap_{j\to i_0} \mathscr{K}(j \to i_0)[\mathscr{K}(j)]$$

holds. Recall that a continuous map whose domain is compact and codomain is Hausdorff is necessarily closed, and finally apply Proposition 5.2.19 and Equation (16) to obtain,

$$\begin{aligned}
{\downarrow}p_{i_0}[L_{\mathscr{K}}] &= {\downarrow}\bigcap_{j\to i_0} \mathscr{K}(j \to i_0)[\mathscr{K}(j)] \\
&= \bigcap_{j\to i_0} {\downarrow}\mathscr{K}(j \to i_0)[\mathscr{K}(j)] \\
&= \mathscr{K}(i_0) \\
&= K_{i_0}
\end{aligned}$$

□

Corollary 5.2.21. *All lower Vietoris polynomial functors in the category $\mathsf{StablyComp}$ preserve codirected limits.*

Proof. It follows from Theorem 5.2.9, Proposition 5.2.20, and the monadicity of the inclusion functor $\mathsf{StablyComp} \to \mathsf{Top}$ that the lower Vietoris functor preserves codirected limits. For the remaining cases, recall that that the inclusion functor $\mathsf{StablyComp} \to \mathsf{Top}$ preserves and reflects limits, and proceed with the proof as in Lemma A.2.9. □

Theorem 5.2.22. *The category of coalgebras for a lower Vietoris polynomial functor in* $\mathsf{StablyComp}$ *is always complete.*

Proof. Theorems 5.2.13 and 5.2.16 clearly guarantee the assumptions of Theorem 5.1.6. This entails that the category of coalgebras for a lower Vietoris polynomial functor in $\mathsf{StablyComp}$ has equalisers. The assertion then follows from Corollary 5.2.21 and [Bar93, Theorem 2.1]. □

In what concerns final coalgebras, there is still room to improve the theorem above: the inclusion functor $\mathsf{StablyComp} \to \mathsf{Top}$ preserves limits and therefore,

Theorem 5.2.23. *Every lower Vietoris polynomial functor in* Top *that can be restricted to* $\mathsf{StablyComp}$ *admits a final coalgebra.*

Proof. Consider a lower Vietoris polynomial functor $F : \mathsf{Top} \to \mathsf{Top}$ and its restriction $\overline{F} : \mathsf{StablyComp} \to \mathsf{StablyComp}$ (assuming it exists) to the category $\mathsf{StablyComp}$. Denote the diagram,

$$1 \longleftarrow F1 \longleftarrow FF1 \longleftarrow \cdots$$

by $\mathscr{D} : \mathbb{N} \to \mathsf{Top}$ and observe that it can be written as a composition,

$$\mathbb{N} \xrightarrow{\mathscr{K}} \mathsf{StablyComp} \xrightarrow{I} \mathsf{Top}$$

Our goal is to show that the isomorphism $F(\lim \mathscr{D}) \simeq \lim F\mathscr{D}$ holds. So we reason,

$$\begin{aligned}
F(\lim \mathscr{D}) &\simeq F(\lim I\mathscr{K}) \\
&\simeq FI(\lim \mathscr{K}) && (I \text{ preserves limits}) \\
&\simeq I\overline{F}(\lim \mathscr{K}) \\
&\simeq I(\lim \overline{F}\mathscr{K}) && (\overline{F} \text{ preserves codirected limits}) \\
&\simeq (\lim I\overline{F}\mathscr{K}) \\
&\simeq (\lim FI\mathscr{K}) \\
&\simeq (\lim F\mathscr{D})
\end{aligned}$$

□

The lower and the compact Vietoris functor in Top are seemingly unrelated. However, when restricted respectively to the categories $\mathsf{StablyComp}$ and $\mathsf{CompHaus}$, they become very close to each other: the lower Vietoris functor in $\mathsf{PosComp}$ induces the compact Vietoris functor in $\mathsf{CompHaus}$ via the composition,

$$\mathsf{CompHaus} \xrightarrow{\text{discrete}} \mathsf{PosComp} \xrightarrow{\mathrm{V}} \mathsf{PosComp} \xrightarrow{\text{forgetful}} \mathsf{CompHaus}$$

Since the functor $\mathsf{PosComp} \xrightarrow{\text{forgetful}} \mathsf{CompHaus}$ is a right adjoint, it must preserve all limits. The inclusion functor $\mathsf{CompHaus} \xrightarrow{\text{discrete}} \mathsf{PosComp}$ also does so, because the monadic functor $\mathsf{StablyComp} \to \mathsf{Top}$ reflects limits and the inclusion $\mathsf{CompHaus} \to \mathsf{Top}$ preserves them [AHS09, Examples 13.31]. An important consequence of this is that studying the preservation of limits by the lower Vietoris functor in the category $\mathsf{StablyComp} \simeq \mathsf{PosComp}$ encompasses studying the preservation of limits by the compact Vietoris functor in $\mathsf{CompHaus}$. In particular, the following results come for free.

Corollary 5.2.24. *All compact Vietoris polynomial functors in the category* $\mathsf{CompHaus}$ *preserve codirected limits.*

Proof. The lower Vietoris functor in $\mathsf{StablyComp}$ preserves codirected limits, so it follows that the compact Vietoris functor in $\mathsf{CompHaus}$ preserves these limits as well. For the remaining cases, recall that the inclusion $\mathsf{CompHaus} \to \mathsf{Top}$ preserves limits and since it is full it also reflects them. □

By taking advantage of the fact that a compact subspace of an Hausdorff space is also compact Hausdorff, [Zen70, Lemma B] proves that,

Theorem 5.2.25. *The compact Vietoris functor in* Haus *preserves codirected limits.*

Proof. Even though a proof of this claim is already given in [Zen70, Lemma B], we will sketch an alternative, more categorical version using the current chapter's results.

We want to show that the compact Vietoris functor $\mathrm{V} : \mathsf{Haus} \to \mathsf{Haus}$ preserves codirected limits. So start with a codirected diagram $\mathscr{D} : \mathsf{I} \to \mathsf{Haus}$ and its limit $(\pi_i : L_{\mathscr{D}} \to \mathscr{D}(i))_{i\in\mathsf{I}}$. We will show that for every cone $(A_i : 1 \to \mathrm{V}\mathscr{D}(i))_{i\in\mathsf{I}}$ the following diagram factorises.

$$\begin{tikzcd}
1 \arrow[r, dotted] \arrow[dr, "A_i"'] & \mathrm{V}L_{\mathscr{D}} \arrow[d, "\mathrm{V}\pi_i"] \\
& \mathrm{V}\mathscr{D}(i)
\end{tikzcd}$$

We will see later on that this is the only thing that we need for proving our claim.

First, observe that for every $i \in \mathsf{I}$ the space $A_i \subseteq \mathscr{D}(i)$ is compact Hausdorff. Second, by the definition of the compact Vietoris functor $\mathrm{V} : \mathsf{Haus} \to \mathsf{Haus}$, we can show that for every map $i \to j \in \mathsf{I}$ the following equation holds.

$$\mathscr{D}(i \to j)[A_i] = A_j$$

This allows to define the codirected diagram $\mathscr{K} : \mathsf{I} \to \mathsf{CompHaus}$ that sends each $i \in \mathsf{I}$ to A_i and each $i \to j$ to the restriction of $\mathscr{D}(i \to j)$ to the domain A_i. Now let us take the limit

$(p_i : L_{\mathscr{K}} \to \mathscr{K}(i))_{i \in \mathsf{I}}$ in CompHaus. The space $L_{\mathscr{K}}$ is compact by definition and it is a subspace of $L_{\mathscr{D}}$. Hence, $L_{\mathscr{K}} \in \mathrm{V}L_{\mathscr{D}}$. The last step in showing the existence of the factorisation above is to prove that for every $i \in \mathsf{I}$ the following equation holds.

$$\mathrm{V}\pi_i(L_{\mathscr{K}}) = \pi_i[L_{\mathscr{K}}] = A_i$$

According to Theorem 5.2.18, the following equation holds

$$\pi_i[L_{\mathscr{K}}] = p_i[L_{\mathscr{K}}] = \bigcap_{j \to i} \underbrace{\operatorname{img} \mathscr{K}(j \to i)}_{A_i} = A_i$$

which proves our claim about the factorisation. This essentially proves that every cone $(f_i : X \to \mathrm{V}\mathscr{D}(i))_{i \in \mathsf{I}}$ in Haus can be factorised as shown by the diagram below.

$$\begin{tikzcd} X \arrow[r, dotted] \arrow[dr, "f_i"'] & \mathrm{V}L_{\mathscr{D}} \arrow[d, "\mathrm{V}\pi_i"] \\ & \mathrm{V}\mathscr{D}(i) \end{tikzcd}$$

In order to prove that the compact Vietoris functor $\mathrm{V} : \mathsf{Haus} \to \mathsf{Haus}$ preserves codirected limits we just need to show that the factorisation $X \to \mathrm{V}L_{\mathscr{D}}$ is continuous and that is the only one that makes the diagram above commute. But this follows directly from the fact that the compact Vietoris functor $\mathrm{V} : \mathsf{Haus} \to \mathsf{Haus}$ preserves initial codirected monocones of Hausdorff spaces (Theorem 5.2.9). □

We can then develop the following results.

Theorem 5.2.26. *Every compact Vietoris polynomial functor in the category* Haus *preserves codirected limits.*

Proof. The previous theorem tells that the compact Vietoris functor in Haus preserves codirected limits. For the remaining cases, note that Haus is a reflective full subcategory of Top [AHS09, Examples 16.2] which means that the inclusion Haus $\to$ Top preserves and reflects limits. Then proceed as in Lemma A.2.9. □

Corollary 5.2.27. *The category of coalgebras for a compact Vietoris polynomial functor in* Haus *is always complete.*

Proof. Since it is a full reflective subcategory of Top, the category Haus is both complete and cocomplete. It is also regularly wellpowered, a property that inherits directly from Top. Recall that regular monomorphisms in Haus are precisely the closed embeddings [AHS09, Examples 7.58] and observe that the compact Vietoris functor preserves these maps. Moreover, the category Haus has an (Epi,RegMono)-factorisation structure, in which continuous functions $f : X \to Y$ are factorised as,

$$X \xrightarrow{f} \overline{f[X]} \hookrightarrow Y$$

These data ensure that we can apply Theorem 5.1.6 to prove that the category of coalgebras for a compact Vietoris polynomial functor has equalisers. The assertion then follows from Theorem 5.2.26 and [Bar93, Theorem 2.1]. □

Theorem 5.2.28. *Every compact Vietoris polynomial functor in the category* Top *that can be restricted to* Haus *admits a final coalgebra.*

Proof. Since Haus is a reflective subcategory of Top the inclusion $\mathsf{Haus} \to \mathsf{Top}$ preserves limits. Then proceed analogously to Theorem 5.2.23. □

We can then extend the results above to subfunctors of Vietoris polynomial ones.

Corollary 5.2.29. *Let* $F : \mathsf{Top} \to \mathsf{Top}$ *be a lower Vietoris polynomial functor that can be restricted to* $\mathsf{StablyComp}$*. Every subfunctor of* F *admits a final coalgebra.*

Proof. Follows directly from Theorem 5.2.23 and Theorem 5.1.28. □

Corollary 5.2.30. *Let* $F : \mathsf{Top} \to \mathsf{Top}$ *be a compact Vietoris polynomial functor that can be restricted to* Haus*. Every subfunctor of* F *admits a final coalgebra.*

Proof. Follows directly from Theorem 5.2.28 and Theorem 5.1.28. □

This last corollary applies to several well-known variants of the compact Vietoris functor. Some interesting cases include the variant that discards the empty set, the variant that takes infinite sets out of comission, and the variant that only considers compact and connected subsets [Dud72].

5.2.4 *Some notes about related work*

To conclude this section, a few remarks on the relation of our work with those reported in [KKV04; BKR07] are in order. Recall that document [KKV04] considers compact Vietoris polynomial functors in the category Stone of Stone spaces and continuous maps. Recall also that [BKR07] considers coalgebras for the lower Vietoris functor in the category $\mathsf{Spectral}$ of spectral spaces and spectral maps.

The categories Stone and $\mathsf{Spectral}$ are closely related with $\mathsf{CompHaus}$ and $\mathsf{StablyComp}$, respectively. Using this section's results, we will capitalise on this relation to derive the two following results: (i) every compact Vietoris polynomial functor in Stone admits a final coalgebra (as already proved in [KKV04]), (ii) and every lower Vietoris polynomial functor in $\mathsf{Spectral}$ admits a final coalgebra as well. We start with the following remark.

Remark 5.2.31. A Stone space X is a compact Hausdorff space with a basis of clopen sets. This is equivalent to saying that X is compact Hausdorff and that the cone of continuous maps $X \to 2$ to the discrete space 2 is initial with respect to the forgetful functor $\mathsf{Top} \to \mathsf{Set}$.

Lemma 5.2.32. *Let* $(X \to X_i)_{i \in I}$ *be an initial cone in* $\mathsf{CompHaus}$ *where* X_i *is a Stone space for every* $i \in I$*. Then* X *is a Stone space as well.*

Proof. Each space X_i induces a initial cone of continuous maps $X_i \to 2$. Moreover, initial cones are closed under composition [AHS09, Proposition 10.45]. This means that one can build, via composition, a initial cone of continuous maps $X \to 2$. □

Corollary 5.2.33. *The inclusion functor* $\mathsf{Stone} \to \mathsf{CompHaus}$ *creates limits. So the category* Stone *is complete and the inclusion preserves and reflects limits.*

Theorem 5.2.34. *Every compact Vietoris polynomial functor in the category* Stone *preserves codirected limits.*

Proof. Every compact Vietoris polynomial functor $F : \mathsf{Stone} \to \mathsf{Stone}$ [Gou13, Proposition 4.11.7] has an analogous functor $F : \mathsf{CompHaus} \to \mathsf{CompHaus}$ that makes the diagram below commute. The claim then follows directly from the inclusion $\mathsf{Stone} \to \mathsf{CompHaus}$ preserving and reflecting limits, and from Corollary 5.2.24.

$$\begin{tikzcd}
\mathsf{Stone} \arrow[r, "F"] \arrow[d] & \mathsf{Stone} \arrow[d] \\
\mathsf{CompHaus} \arrow[r, "F"'] & \mathsf{CompHaus}
\end{tikzcd}$$

□

Corollary 5.2.35. *Every compact Vietoris polynomial functor in the category* Stone *admits a final coalgebra.*

Proof. Follows directly from Theorem 5.2.34 and Theorem 5.1.2. □

Let us now focus on spectral spaces and the lower Vietoris functor.

Remark 5.2.36. A spectral space X is a stably compact space with a basis of compact open subsets. This is equivalent to saying that X is stably compact and that the cone of spectral maps $X \to 2$ to the Sierpiński space is initial.

Lemma 5.2.37. *Let* $(X \to X_i)_{i \in I}$ *be an initial cone in* $\mathsf{StablyComp}$ *where* X_i *is a spectral space for every* $i \in I$*. Then* X *is a spectral space as well.*

Proof. Each space X_i induces a initial cone of spectral maps $X_i \to 2$. Moreover, initial cones are closed under composition [AHS09, Proposition 10.45]. This means that one can build, via composition, a initial cone of spectral maps $X \to 2$. □

Corollary 5.2.38. *The inclusion* $\mathsf{Spectral} \to \mathsf{StablyComp}$ *creates limits. Hence, the category* $\mathsf{Spectral}$ *is complete, and the inclusion preserves and reflects limits.*

Theorem 5.2.39. *Every lower Vietoris polynomial functor* $F : \mathsf{Spectral} \to \mathsf{Spectral}$ *preserves codirected limits.*

Proof. Every lower Vietoris polynomial functor $F : \mathsf{Spectral} \to \mathsf{Spectral}$ [Gou13, page 433] has an analogous functor $F : \mathsf{StablyComp} \to \mathsf{StablyComp}$ that makes the diagram below commute. The claim then follows directly from the inclusion $\mathsf{Spectral} \to \mathsf{StablyComp}$ preserving and reflecting limits, and from Corollary 5.2.21.

$$\begin{tikzcd}
\mathsf{Spectral} \arrow[r, "F"] \arrow[d] & \mathsf{Spectral} \arrow[d] \\
\mathsf{StablyComp} \arrow[r, "F"'] & \mathsf{StablyComp}
\end{tikzcd}$$

□

Corollary 5.2.40. *Every lower Vietoris polynomial functor in the category* $\mathsf{Spectral}$ *admits a final coalgebra.*

Proof. Follows directly from Theorem 5.2.39 and Theorem 5.1.2. □

5.3 NOTIONS OF BEHAVIOUR FOR HYBRID MACHINES

5.3.1 *The topological hybrid monad*

Using the previous section's results on final coalgebras, we can now tackle the problem of representations lacking a canonical notion of behaviour when the condition 'as-soon-as' is dropped (see Section 4.5 and Section 5.2). Our first step in this quest is to build a topological variant of the hybrid monad in order to interpret representations as coalgebras over Top. We start with the following definition [Bor94b, Section 7.1].

Definition 5.3.1. Every topological space X induces the exponential functor,

$$(-)^X : \mathsf{Top} \to \mathsf{Top}$$

that maps a space Y to the space of continuous maps Y^X whose topology is generated by the sets,

$$\{f \in Y^X \mid f[K] \subseteq U\} \qquad (K \text{ compact in } X, U \text{ open in } Y)$$

The exponential $(-)^X$ is right adjoint of the functor $(- \times X) : \mathsf{Top} \to \mathsf{Top}$ if X is locally compact, *i. e.* if every neighbourhood of a point $x \in X$ also contains a compact neighbourhood of this point (see [Bor94b, Proposition 7.1.5]).

We will also need the following fact: the space of non-negative reals $\mathbb{R}_{\geq 0}$ is Hausdorff and locally compact. It has, therefore, a one-point compactification [Kel55, page 150], namely the set of extended non-negative reals $[0, \infty]$ equipped with the topology that is generated by the subsets $\{(i, \infty] \mid i \in \mathbb{R}_{\geq 0}\}$ and $\{(i, j) \cap \mathbb{R}_{\geq 0} \mid i, j \in \mathbb{R}\}$.

Let us start by building the topological analogue of the functor $\mathrm{H}_* : \mathsf{Set} \to \mathsf{Set}$.

Definition 5.3.2. Define $\mathrm{H_c} : \mathsf{Top} \to \mathsf{Top}$ as the subfunctor of,

$$(\,-\,)^{\mathbb{R}_{\geq 0}} \times [0, \infty] : \mathsf{Top} \to \mathsf{Top}$$

that maps a space X to the subspace $\big\{(f, d) \in X^{\mathbb{R}_{\geq 0}} \times [0, \infty] \mid f \cdot \min(\,-\,, d) = f\big\}$ where $\min : \mathbb{R}_{\geq 0} \times [0, \infty] \to \mathbb{R}_{\geq 0}$ is the minimum function.

The equality $f \cdot \min(\,-\,, d) = f$ says that for every real number $r \in [d, \infty)$ the equation $f(r) = f(d)$ holds.

Remark 5.3.3. The reader might be wondering why the functor $\mathrm{H_c}$ was not written as a coproduct of function spaces, like the hybrid monad in Set. The reason for this boils down to two important observations: first, for every space X, $\mathrm{H_c}X$ is in bijective correspondence with the coproduct of continuous maps,

$$\coprod_{i \in \mathbb{R}_{\geq 0}} X^{[0,i]} + X^{\mathbb{R}_{\geq 0}}$$

So the space $\mathrm{H_c}X$ and the expected coproduct presentation only differ in their topology. Second, every connected space X induces an isomorphism $\mathsf{Top}\left(X, \coprod_{i \in I} A_i\right) \simeq \coprod_{i \in I} \mathsf{Top}\left(X, A_i\right)$ [Bou66, page 109, Propositions 4 and 5], which means that if $\mathrm{H_c}$ had been presented as a coproduct, a hybrid program,

$$\mathbb{R}^n \to \mathrm{H_c}(\mathbb{R}^n) \simeq \coprod_{i \in \mathbb{R}_{\geq 0}} (\mathbb{R}^n)^{[0,i]} + X^{\mathbb{R}_{\geq 0}}$$

would not be able to output evolutions of different durations along different inputs (due to the space $\mathbb{R}^n$ being connected).

Now let us equip $\mathrm{H_c} : \mathsf{Top} \to \mathsf{Top}$ with the unit and multiplication maps of the hybrid monad (Chapter 3). Once shown that the inherited maps are continuous, the proof that $(\mathrm{H_c}, \eta, \mu)$ is a monad follows directly.

Proposition 5.3.4. *Every component of the unit $\eta : \mathrm{Id} \to \mathrm{H_c}$ is continuous and the same applies to the components of the multiplication $\mu : \mathrm{H_c}\mathrm{H_c} \to \mathrm{H_c}$.*

Proof. In Appendix A. □

Corollary 5.3.5. *The triple $(\mathrm{H_c}, \eta, \mu)$ is a monad.*

Proof. Let $\mathrm{U} : \mathsf{Top} \to \mathsf{Set}$ be the forgetful functor that sends topological spaces to their carrier and observe that there exists a monomorphic natural transformation $\mathrm{U}\mathrm{H_c} \hookrightarrow \mathrm{H_*}\mathrm{U}$. The proof then follows directly from the fact that $\mathrm{H_c}$ and $\mathrm{H_*}$ share their unit and multiplication. □

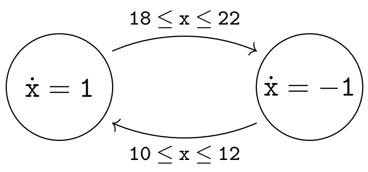
Theorem 5.3.6. *The functor $\mathrm{H_c} : \mathsf{Top} \to \mathsf{Top}$ can be restricted to the category of Hausdorff spaces.*

Proof. Consider a Hausdorff space X. The space $X^{\mathbb{R}_{\geq 0}}$ is finer than the product $\prod_{\mathbb{R}_{\geq 0}} X$ which entails that $X^{\mathbb{R}_{\geq 0}}$ is Hausdorff. The proof then follows directly from the fact that Hausdorff spaces are closed under products and subspaces. □

In the following subsection, we will work with the topological hybrid monad as if it did not contain evolutions with infinite duration. Such evolutions will come into play later in the chapter.

5.3.2 *Bounded representations*

Recall the notion of representation (Section 4.2): a coalgebra $M \to M \times \mathrm{Tr} \times \mathrm{Ev}$ on the category Set where Tr denotes the usual set of discrete assignments and Ev is the set of event-triggered differential equations. One example of a representation is the automaton below which encodes the behaviour of a (simplistic) thermostat.



Previously, we showed how to interpret representations as coalgebras over Set by adopting the 'as-soon-as' condition. We also showed that dropping the latter entails the loss of a notion of observational behaviour for representations, which includes the thermostat above.

One interesting thing about this thermostat, however, (which is common to a large number of representations) is that its guards are interpreted as compact subsets in $\mathbb{R}^n$, which hints at the possiblity of interpreting the thermostat in the category Top via the compact Vietoris functor $\mathrm{V} : \mathsf{Top} \to \mathsf{Top}$. For this, we will need the following two results.

Proposition 5.3.7. *There exists a natural transformation* $\beta : (-)^{\mathbb{R}_{\geq 0}} \times \mathbb{R}_{\geq 0} \to \mathrm{H_c}$ *defined at each space* X *by* $\beta_X(f, d) = f \cdot \min(d, -)$.

Proof. The continuity of the components follows directly from both the exponentials' universal property and the continuity of the minimum function. The naturality property is easy to prove. □

Proposition 5.3.8. *Let* $\mathrm{V} : \mathsf{Top} \to \mathsf{Top}$ *be the compact Vietoris functor. It has a tensorial strength* $\tau : \mathrm{V} \times \mathrm{Id} \to \mathrm{V}(- \times -)$ *defined at spaces* X *and* Y *by,*

$$\tau_{X,Y}(K, x) = K \times \{x\}$$

Proof. In Appendix A. □

Let us denote the thermostat's automaton by $\langle a, b, c\rangle : M \to M \times \mathrm{Tr} \times \mathrm{Ev}$. Using Proposition 5.3.8, it is straightforward to show that it induces a continuous map $f : \coprod_{m\in M} \mathbb{R}^n \to \mathrm{V}(\mathbb{R}_{\geq 0})$ defined by,

$$(m, v) \mapsto \begin{cases} \phi^{-1}_{(m,v)}([\![\psi]\!]) & \text{if } \phi^{-1}_{(m,v)}([\![\psi]\!]) \neq \emptyset \\ \{0\} & \text{otherwise} \end{cases}$$

where $\phi_{(m,v)}$ is the solution of $c(m)$ with v as the initial condition, and ψ is the guard in $c(m)$. Intuitively, f gives the instants of time at which the evolution $\phi_{(m,v)}$ satisfies the guard ψ if at least one such instant exists. Otherwise, it just returns $\{0\}$. Its continuity can be shown using the map $\coprod_{m\in M} \mathbb{R} \to \mathbb{R} \times \mathrm{V}\mathbb{R}$ that sends a mode and valuation to the same valuation and the interpretation of the corresponding guard, the map $\tau : \mathrm{V} \times \mathrm{Id} \to \mathrm{V}(- \times -)$, and the truncated subtraction $\ominus : \mathbb{R} \times \mathbb{R} \to \mathbb{R}_{\geq 0}$.

Remark 5.3.9. The fact that f returns $\{0\}$ if the evolution $\phi_{(m,v)}$ never intersects ψ may be somewhat odd, but it is necessary for f to be continuous since the empty set is an isolated point in $\mathrm{V}(\mathbb{R}_{\geq 0})$ [Kec12, Exercise 4.20]. Later on, we will show how to exclude this behaviour of f through a subcoalgebra construction.

Abbreviate the composition of continuous maps,

$$\mathrm{V}\beta_{\mathbb{R}^n} \cdot \tau_{\mathbb{R}_{\geq 0}, (\mathbb{R}^n)^{\mathbb{R}_{\geq 0}}} : \mathrm{V}(\mathbb{R}_{\geq 0}) \times (\mathbb{R}^n)^{\mathbb{R}_{\geq 0}} \to \mathrm{V}\mathrm{H_c}(\mathbb{R}^n)$$

by r. The expression $r(f(m, v), \phi_{(m,v)})$ is the set of restrictions of $\phi_{(m,v)}$ whose last point satisfy ψ. Then, let us interpret the thermostat's automaton as a $\mathrm{V}(- \times \mathrm{H_c}(\mathbb{R}^n))$-coalgebra of type $M \times \mathbb{R}^n \to \mathrm{V}(M \times \mathbb{R}^n \times \mathrm{H_c}(\mathbb{R}^n))$. Equip M with the discrete topology and define the mapping,

$$(m, v) \mapsto \{\ (a(m),\ [\![b(m)]\!] \cdot \lambda_{\mathbb{R}^n}(e),\ e) \mid e \in r(f(m, v), \phi_{(m,v)})\ \}$$

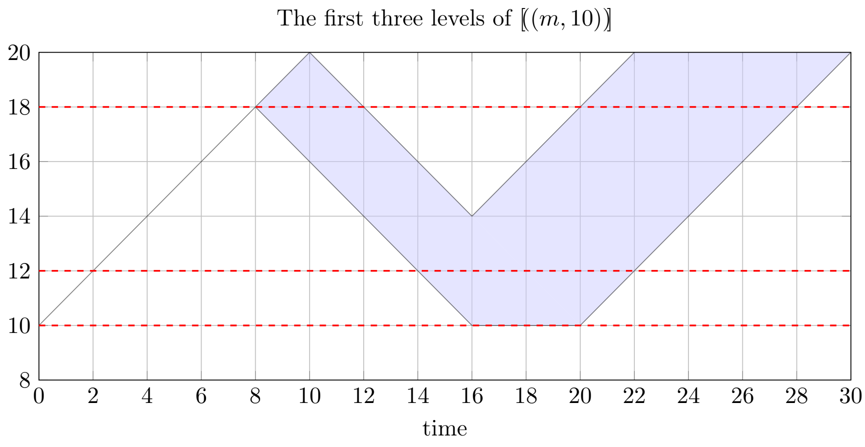
This mapping is continuous because it can be written as a composition of continuous functions via the tensorial strength of the Vietoris functor, the natural transformation $\lambda : \mathrm{H_c} \to \mathrm{Id}$ that sends an evolution to its last point, and the application map $(\mathbb{R}^n)^{\mathbb{R}^n} \times \mathbb{R}^n \to \mathbb{R}^n$. Moreover, the functor,

$$\mathrm{V}(- \times \mathrm{H_c}(\mathbb{R}^n)) : \mathsf{Top} \to \mathsf{Top}$$

can be restricted to the category of Hausdorff spaces (Theorem 5.3.6) which means that it admits a final coalgebra, and thus we obtain a notion of observable behaviour for the thermostat. Intuitively, the elements of the final $\mathrm{V}(- \times \mathrm{H_c}(\mathbb{R}^n))$-coalgebra can be seen as compactly branching trees – *i. e.* trees where the set of descendants of each node is compact – and whose edges are labelled by elements of $\mathrm{H_c}(\mathbb{R}^n)$.

An illustrative representation of an element in the final $\mathrm{V}(- \times \mathrm{H_c}(\mathbb{R}^n))$-coalgebra is the superposition of the evolutions in each level of the tree. For example,

Example 5.3.10. Consider the thermostat's automaton and denote its left mode by m. The first three levels of the tree $[\![(m,10)]\!]$ are depicted in the plot below.



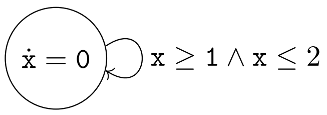
The dashed horizontal lines mark the region in which the guards are satisfied, and the shaded regions mark the values that are possible to observe at each instant of time.

Remark 5.3.11. Let m denote the left mode of the thermostat's automaton and n the right mode. The model $M \times \mathbb{R} \to \mathrm{V}(M \times \mathbb{R} \times \mathrm{H_c}(\mathbb{R}))$ of the thermostat has a subcoalgebra of the type,

$$\coprod_{i \in M} A_i \to \mathrm{V}(\coprod_{i \in M} A_i \times \mathrm{H_c}(\mathbb{R}))$$

with $A_m = (-\infty, 22]$ and $A_n = [10, \infty)$. In this case, the behaviour of f that was discussed in Remark 5.3.9 never occurs, because the evolutions that start either in A_m or A_n always intersect the corresponding guards.

We provided a notion of behaviour to the thermostat, but unfortunately we cannot apply the same strategy to all representations whose guards are interpreted as compact subsets of $\mathbb{R}^n$: consider, for example, the following automaton.



Starting with valuation one, a jump can occur at any instant of time because the evolution induced by $\dot{\mathtt{x}} = 0$ is stationary. This means that the function $f : \coprod_{m \in M} \mathbb{R}^n \to \mathrm{V}(\mathbb{R}_{\geq 0})$ considered before is not well-defined for this automaton, as $\phi^{-1}_{(m,1)}([1,2]) = \mathbb{R}_{\geq 0}$ is clearly not compact.

Up to this moment, we still do not know of any general characterisation of representations for ensuring that they can equipped with a notion of observational behaviour in the category Top. So checking that their function $f : \coprod_{m \in M} \mathbb{R}^n \to \mathrm{V}(\mathbb{R}_{\geq 0})$ is well-defined and continuous, as we did for the thermostat, is still our main approach for a equipping a representation with one such notion. We can, however, provide the following theorem.

Theorem 5.3.12. *Assume that* Ev *is the set of event-triggered differential equations of the type* $(\dot{x}_1 = r, \ldots, \dot{x}_n = t \,\&\, a \leq x_1 \leq b)$ *where* $r \neq 0$*, and* a *and* b *are real numbers. Then, representations,*

$$M \to M \times \mathrm{Tr} \times \mathrm{Ev}$$

can always be interpreted as $\mathrm{V}(- \times \mathrm{H_c}(\mathbb{R}^n))$*-coalgebras.*

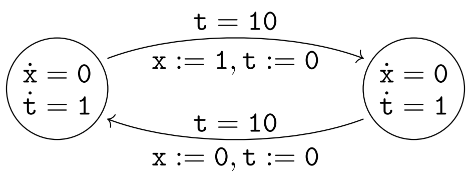
Proof. According to the previous remarks, we just need to show that the corresponding function $f : \coprod_{m \in M} \mathbb{R}^n \to \mathrm{V}(\mathbb{R}_{\geq 0})$ is well-defined and continuous. So take a mode $m \in M$ and consider its expression $(\dot{x}_1 = r, \ldots, \dot{x}_n = t \,\&\, a \leq x_1 \leq b)$. For this mode, the function f can be written as,

$$(m, (v_1, \ldots, v_n)) \mapsto \left[\max\left(0, \frac{a - v_1}{r}\right), \max\left(0, \frac{b - v_1}{r}\right)\right]$$

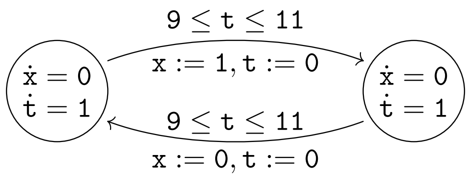
Since closed intervals of the type $[r_1, r_2]$ are compact in $\mathbb{R}_{\geq 0}$, the function is well-defined. Since the subtraction, division, and maximum operations are continuous, the function must be continuous as well. □

We qualify as *bounded* all representations of the type described in the theorem above. Let us analyse a simple example of one such representation.

Example 5.3.13. In Examples 4.2.21, we considered a semaphore that switches between a green light and a red one every ten seconds. This behaviour was represented by the automaton below.

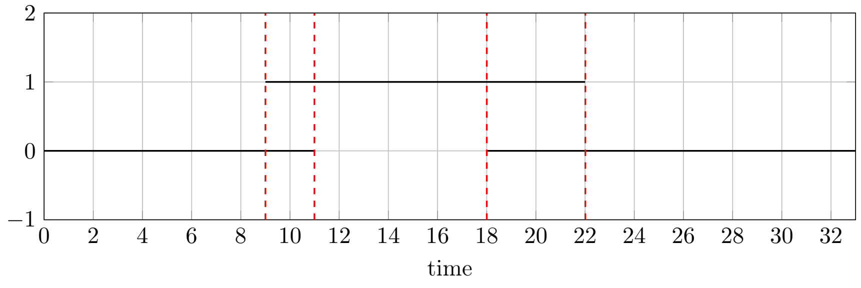


Now, a reasonable expectation is that the semaphore does not switch the lights when precisely ten seconds have passed, but *e. g.* between nine and eleven seconds. This yields the automaton below.



This representation is clearly bounded, and, therefore, by Theorem 5.3.12 it has a notion of observational behaviour. Denoting its left mode by m, we can analyse the behaviour *e. g.* of the state $(m, 0)$: the first three levels of the tree $[\![(m, 0)]\!]$ are depicted below.

The first three levels of $[\![(m,0)]\!]$



Note that between nine and eleven seconds we cannot know for sure which light is on, so we assume that both are on at the same time, and similarly for the interval of time $[18, 22]$. Note also that the margin of error increases by two seconds at each iteration.

Remark 5.3.14. It is straightfoward to extend the interpretation of representations presented in this section so that it encompasses non-deterministic assignments. For example, let Tr be the set of assignments of the type $x := t \times [r_1, r_2]$ with $r_1 \leq r_2$. The expression $t \times [r_1, r_2]$ denotes the multiplication of t with all values between r_1 and r_2. Then, consider a representation $\langle a, b, c\rangle : M \to M \times \mathrm{Tr} \times \mathrm{Ev}$, assume that it has a continuous function $f : \coprod_{m\in M} \mathbb{R}^n \to \mathrm{V}(\mathbb{R}_{\geq 0})$, and define $M \times \mathbb{R}^n \to \mathrm{V}(M \times \mathbb{R}^n \times \mathrm{H_c}(\mathbb{R}^n))$ by,

$$(m, v) \mapsto \bigcup \left\{ \, (a(m), u, e) \mid e \in r(f(m, v), \phi_{(m,v)}), u \in [\![b(m)]\!] \cdot \lambda_{\mathbb{R}^n}(e) \, \right\}$$

This mapping is continuous because it can be written as a composition of continuous functions via the tensorial strength of the Vietoris functor, the natural transformation $\bigcup : \mathrm{VV} \to \mathrm{V}$ that takes unions [Mic51, Theorem 2.5.2 and 5.7.2], the natural transformation $\lambda : \mathrm{H_c} \to \mathrm{Id}$ that sends an evolution to its last point, and the application map $(\mathrm{V}(\mathbb{R}^n))^{\mathbb{R}^n} \times \mathbb{R}^n \to \mathrm{V}(\mathbb{R}^n)$.

5.4 COMPACT STABILITY

We will now address stability – which, as already mentioned, is a central aspect of control theory [Lya92; Goe+04; Sho+07; GST09]. Roughly put, a system – in our context, a hybrid program – is called stable if small changes in its state or input do *not* make it behave very differently. Consider, for example, a bouncing ball in the edge of a precipice: it is an unstable system because slight changes in its location might produce drastic changes in its behaviour.

Even if not a common topic in computer science, the notion of stability is crucial for hybrid systems because of the deep interaction of computational devices with physical processes. In program verification, for example, it is important that the hybrid program at hand is stable, for otherwise it might behave as expected with a certain initial configuration, but behave very badly with a configuration close to the intended one: think again of the bouncing ball near a precipice.

There are different notions of stability: traditional examples include the widely famous Lyapunov stability [Lya92], extensively studied in the last decades, asymptotic Lyapunov stability [Goe+04], and bounded-input, bounded-output stability [Sho+07]. We will restrict ourselves to the classical Lyapunov stability. We will show that a slightly weaker and *more general version* of this notion can be directly encoded in the topological hybrid monad. And among other things, we will use this feature to generate different programming languages that support a *compositional* analysis of hybrid programs with respect to stability: *i. e.* by showing that the (simpler) constituents of a (complex) hybrid program are stable one proves that the latter is stable as well.

We start by introducing Lyapunov stability.

Definition 5.4.1. Let $\mathrm{U} : \mathsf{Top} \to \mathsf{Set}$ be the functor that sends topological spaces to their underlying set. Consider a metrisable space M and a function $f : \mathrm{U}M \to \mathrm{UH_c}M$. An element $x \in M$ is an *equilibrium point* if the evolution $f(x)$ is constant. An equilibrium point $x \in M$ is *Lyapunov stable* if for every $\epsilon > 0$ there exists a $\delta > 0$ such that,

$$d(x, y) < \delta \text{ entails } d(f(x, t), f(y, t)) < \epsilon$$

for every $y \in M$ and $t \in \mathbb{R}_{\geq 0}$. The function f is called Lyapunov stable if every equilibrium point is Lyapunov stable.

Example 5.4.2. The bouncing ball b that was described in Examples 3.2.14 has a unique equilibrium point: $(0, 0) \in \mathbb{R}^2$, *viz.* the ball is on the ground and has no velocity. The point is Lyapunov stable because for every real number $\epsilon > 0$, to *drop* the ball at height $\epsilon/2 (= \delta)$ will never make it reach the distance of ϵ units from the ground neither will it reach the velocity of ϵ units.

Let us now examine the notion of stability that is encoded in the topological hybrid monad and compare it against Lyapunov stability.

Definition 5.4.3. A function $f : \mathrm{U}X \to \mathrm{UH_c}X$ is qualified as *compact stable* if it is continuous with respect to the topological spaces X and $\mathrm{H_c}X$, in other words if it is a hybrid program $f : X \to \mathrm{H_c}X$ for the monad $\mathrm{H_c}$.

The definition is quite compact, and so at first sight it might not disclose much. However, it uses the notion of continuity and this brings an intuitive description into the picture: the map f is compact stable if close points in X yield close evolutions in $\mathrm{H_c}X$. Formally, the definition can be unravelled in the following manner: let X and Y be topological spaces where $\mathcal{S}$ is a subbasis of Y. Then recall that a function $f : X \to Y$ is called continuous if for every $x \in X$ and subbasic neighbourhood $S \in \mathcal{S}$ of $f(x)$ there exists an open neighbourhood U of x such that $f[U] \subseteq S$. Now, denote the sets,

$$\left\{ g \in Y^{\mathbb{R}_{\geq 0}} \mid g[K] \subseteq V \right\}$$

by $[K, V]$. The topology of $\mathrm{H_c}X$ has as subbasis the sets of type $([K, V] \times W) \cap \mathrm{H_c}X$ where K is a compact subset of $\mathbb{R}_{\geq 0}$, V is an open subset of X, and W is an open subset of $[0, \infty]$ (see Definition 5.3.1). Finally, the function $f : \mathrm{U}X \to \mathrm{UH_c}X$ is compact stable if for every element

$x \in X$ and every subbasic neighbourhood $([K,V] \times W) \cap \mathrm{H_c}X$ of $f(x)$ there exists an open neighbourhood U of x such that,

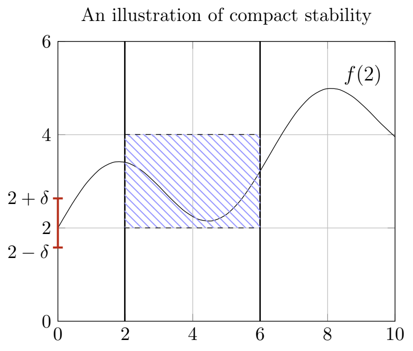
$$f[U] \subseteq ([K,V] \times W) \cap \mathrm{H_c}X$$

In words, the evolutions of points close to x satisfy two conditions: (i) their durations are contained in W, and (ii) in the interval of time K they are contained in the region V.

Example 5.4.4. Let X be the Euclidean space $\mathbb{R}$ and $f(2)$ be the evolution depicted in the plot below. If the function $f : \mathrm{U}X \to \mathrm{UH_c}X$ is compact stable we can choose, for example, the subbasic neighbourhood,

$$S = \big([[2,6],(2,4)] \times (10-0.1, 10+0.1)\big) \cap \mathrm{H_c}X$$

of $f(2)$ and there must exist a positive real number $\delta > 0$ such that, for every $x \in X$, $\mid x - 2 \mid < \delta$ entails $f(x) \in S$.



Intuitively, all evolutions of points close to 2 must have duration close to 10, and in the interval of time $[2,6]$ they must be contained in $(2,4)$.

The following theorem relates compact stability to the Lyapunov one in a precise way.

Theorem 5.4.5. *Consider a hybrid program $f : X \to \mathrm{H_c}X$ where X is a metrisable space. For every equilibrium point $x \in X$, real number $\epsilon > 0$, and compact subset K of $\mathbb{R}_{\geq 0}$, there exists a real number $\delta > 0$ such that,*

$$d(x,y) < \delta \textit{ entails } d(f(x,t), f(y,t)) < \epsilon$$

for every $y \in X$ and $t \in K$.

Proof. Take an equilibrium point $x \in X$, a value $\epsilon > 0$, and a compact subset K of $\mathbb{R}_{\geq 0}$. Then let B be the open ball centred in $f(x,0)$ and with radius ϵ. Since x is an equilibrium point, the condition $f[\{x\} \times K] \subseteq B$ holds. Therefore, one can build the subbasic neighbourhood,

$$S = ([K,B] \times \mathbb{R}_{\geq 0}) \cap \mathrm{H_c}X$$

of $f(x)$. The assumption on continuity provides an open ball U centred in x such that $f[U] \subseteq S$, and the radius δ of U satisfies the required condition. □

Intuitively, the theorem above tells that compact stability 'almost' entails Lyapunov stability; the difference is that the former considers compact durations and the latter considers the whole duration $\mathbb{R}_{\geq 0}$. In the case of all evolutions with finite duration compact stability entails Lyapunov stability.

Example 5.4.6. Let us consider the atomic program $[\![(\dot{\mathtt{x}} = \mathtt{1} \,\&\, \infty)]\!] : \mathbb{R} \to \mathrm{H_c}\mathbb{R}$. It is compact stable by definition, but it is *not* Lyapunov stable: it has an equilibrium point, *viz.* 0, and whatever positive point v close to it one chooses, the trajectory $[\![(\dot{x} = 1 \,\&\, \infty)]\!](v)$ will eventually become bigger than a given $\epsilon > 0$.

Every atomic program $[\![(\dot{\mathtt{x}} = \mathtt{1} \,\&\, \mathtt{d})]\!] : \mathbb{R} \to \mathrm{H_c}\mathbb{R}$ with $\mathtt{d}$ finite is Lyapunov stable (a consequence of the previous theorem).

When compared with Lyapunov stability, compact stability applies to a broader context: it does not need equilibrium points and it does not require the spaces involved to be metrisable. It also possesses an interesting feature which was mentioned before: it allows to reason about the stability of hybrid programs in a compositional manner. To see why, consider two continuous maps $f : X \to \mathrm{H_c}X$, $g : X \to \mathrm{H_c}X$ and observe that we can always form the composition $g \bullet f : X \to \mathrm{H_c}X$ which is another continuous map. This means that sequential composition of hybrid programs is closed under compact stability, and thus we obtain the following result: recall from Chapter 3 the notion of Kleisli representation. Then,

Corollary 5.4.7. *Let* $\mathrm{U} : \mathsf{V} \to \mathsf{Set}$ *be the forgetful functor of a quasi-variety* V. *The free extension of an interpretation map,*

$$A \to \mathrm{U}(\mathrm{End}_{\mathrm{H_c}}(X),\, \bullet\,, \eta_X, ([\![\sigma]\!])_{\sigma \in \Phi})$$

defines a hybrid programming language whose operations are closed under compact stability.

Example 5.4.8. Recall the time-triggered programming language,

$$\mathsf{p} = \mathsf{a} \in \mathsf{At}(X) \mid \mathsf{skip} \mid \mathsf{p}\,;\, \mathsf{p}$$

introduced in Chapter 3, and how it was generated: briefly, we took a finite set of real-valued variables $X = \{x_1, \ldots, x_n\}$ and denotde by $\mathsf{At}(X)$ the set induced by the grammar,

$$\varphi \ni (x_1 := t, \ldots, x_n := t) \mid (\dot{x}_1 = t, \ldots, \dot{x}_n = t \,\&\, d), \qquad t \ni r \mid r \cdot x \mid t + t$$

where d and r are real numbers, and $x \in X$. Using the notion of Kleisli representation, the programming language was (freely) generated by a map $\mathsf{At}(X) \to \mathrm{U}(\mathrm{End}_{\mathrm{H}}(\mathbb{R}^n),\, \bullet\,, \eta_{\mathbb{R}^n})$. The latter factorises through the monoidal inclusion [Per13, Section 3.1],

$$(\mathrm{End}_{\mathrm{H_c}}(\mathbb{R}^n),\, \bullet\,, \eta_{\mathbb{R}^n}) \hookrightarrow (\mathrm{End}_{\mathrm{H}}(\mathbb{R}^n),\, \bullet\,, \eta_{\mathbb{R}^n})$$

Thus, all atomic programs of this language are compact stable and all of its program constructs are closed under compact stability. In particular, the water tank system, the heating system, and the cruiser controller detailed in Examples 3.2.10 are all compact stable.

Recall the event-triggered programming language introduced in Chapter 3. Its atomic programs need not be compact stable: consider, for example, $(\dot{\mathtt{x}} = \mathtt{1} \mathbin{\&} \mathtt{x} = \mathtt{0} \vee \mathtt{x} = \mathtt{1})$. Given input zero, this program returns an evolution whose duration is zero, and for every positive input close to zero the program returns an evolution whose duration is close to one.

We can show, however, that all program operations of this language are also closed under compact stability: consider the interpretation map $[\![-]\!] : \mathsf{At}^{\mathsf{e}}(X) \to \mathrm{End}_{\mathrm{H}}(\mathbb{R}^n)$ with respect to the event-triggered programming language. Then let us take the pullback,

$$\begin{tikzcd} P \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \mathsf{At}^{\mathsf{e}}(X) \arrow[d, "{[\![-]\!]}"] \\ \mathrm{End}_{\mathrm{H_c}}(\mathbb{R}^n) \arrow[r, hook] & \mathrm{End}_{\mathrm{H}}(\mathbb{R}^n) \end{tikzcd}$$

It is essentially the subset $P \subseteq \mathsf{At}^{\mathsf{e}}(X)$ of compact stable atomic programs, and the function $P \to \mathrm{End}_{\mathrm{H_c}}(\mathbb{R}^n)$ is an interpretation map for them. The free monoidal extension of this interpretation map generates a subprogramming language of the one obtained from the free extension of $[\![-]\!] : \mathsf{At}^{\mathsf{e}}(X) \to \mathrm{End}_{\mathrm{H}}(\mathbb{R}^n)$. The former contains all atomic programs of the latter that are compact stable, and allows to compose them whilst preserving compact stability. For example, the heating program – which was written as a composition of compact stable programs (Examples 3.2.14) – is compact stable, and for every finite number n of bounces,

$$\underbrace{\mathsf{b}\ ;\ \ldots\ ;\mathsf{b}}_{n \text{ times}}$$

the bouncing ball is also compact stable (Examples 3.2.14).

The following theorem provides a complete characterisation of natural transformations $\mathrm{H_c} \times \mathrm{H_c} \to \mathrm{H_c}$. As mentioned before, these are the basis of hybrid program operations in the topological setting.

Theorem 5.4.9. *The binary natural transformations* $\mathrm{H_c} \times \mathrm{H_c} \to \mathrm{H_c}$ *are in bijective correspondence with the continuous maps* $m : [0, \infty] \times [0, \infty] \to \mathrm{H_c}(\mathbb{R}_{\geq 0} + \mathbb{R}_{\geq 0})$ *such that for every* $i, j \in [0, \infty]$, $m(i, j)$ *is a pair* (f, k) *with* $f[\mathbb{R}_{\geq 0}] \subseteq [0, i] + [0, j]$.

Proof. In Appendix A. □

Given a continuous map $m : [0, \infty] \times [0, \infty] \to \mathrm{H_c}(\mathbb{R}_{\geq 0} + \mathbb{R}_{\geq 0})$ as defined above, the associated natural transformation α^m is defined at each space X by,

$$\alpha^m_X((f, i), (g, j)) = ([f, g] \cdot (\pi_1 \cdot m(i, j)), \pi_2 \cdot m(i, j))$$

This means that natural transformations of the type $\mathrm{H_c} \times \mathrm{H_c} \to \mathrm{H_c}$ behave essentially like those typed as $\mathrm{H} \times \mathrm{H} \to \mathrm{H}$ (characterised in Chapter 3). There is, however, one remarkable difference, which arises from the topological context: continuity puts severe restrictions on the transformations $\mathrm{H_c} \times \mathrm{H_c} \to \mathrm{H_c}$, as demonstrated by the following calculation. The spaces $\mathbb{R}_{\geq 0}$ and $[0, \infty]$ are connected. Therefore,

$$\begin{aligned}
\hom\left([0,\infty]^2, \mathrm{H_c}(\mathbb{R}_{\geq 0} + \mathbb{R}_{\geq 0})\right) &\hookrightarrow \hom\left([0,\infty]^2, (\mathbb{R}_{\geq 0} + \mathbb{R}_{\geq 0})^{\mathbb{R}_{\geq 0}} \times [0,\infty]\right) \\
&\simeq \hom\left([0,\infty]^2, ((\mathbb{R}_{\geq 0})^{\mathbb{R}_{\geq 0}} + (\mathbb{R}_{\geq 0})^{\mathbb{R}_{\geq 0}}) \times [0,\infty]\right) \\
&\simeq \hom\left([0,\infty]^2, ((\mathbb{R}_{\geq 0})^{\mathbb{R}_{\geq 0}} \times [0,\infty]) + ((\mathbb{R}_{\geq 0})^{\mathbb{R}_{\geq 0}} \times [0,\infty])\right) \\
&\simeq \coprod_2 \hom\left([0,\infty]^2, (\mathbb{R}_{\geq 0})^{\mathbb{R}_{\geq 0}} \times [0,\infty]\right)
\end{aligned}$$

The calculation tells that for every pair of evolutions, (f, i), (g, j), a natural transformation $\mathrm{H_c} \times \mathrm{H_c} \to \mathrm{H_c}$ can only return an evolution (h, k) whose values come exclusively from (f, i) or exclusively from (g, j). In other words, (h, k) is either a redistribution of the values in (f, i) or a redistribution of the values in (g, j), which means that natural transformations $\mathrm{H_c} \times \mathrm{H_c} \to \mathrm{H_c}$ *cannot combine* evolutions.

This highlights one message of the thesis (Chapter 3): when developing hybrid program operations, one may want to look at natural transformations over algebraic structures. For example, in Chapter 3 we introduced a natural transformation $(+) : \mathrm{HU} \times \mathrm{HU} \to \mathrm{HU}$ over monoids, which multiplies evolutions pointwise; this allowed to sum the signals of two harmonic oscillators. We also introduced a natural transformation $(\|) : \mathrm{HU} \times \mathrm{HU} \to \mathrm{HU}$ over groups, which allowed for hybrid programs to compete over (or contribute to) shared resources. Both transformations can be defined in the topological setting by considering topological monoids and topological groups. Therefore, the programming language that allows to sum signals,

$$\mathsf{p} = \mathsf{a} \in \mathsf{At^e}(X) \mid \mathsf{skip} \mid \mathsf{p}\ ;\ \mathsf{p} \mid 1 \mid \mathsf{p} + \mathsf{p}$$

and the programming language with concurrency,

$$\mathsf{p} = \mathsf{a} \in \mathsf{At^e}(X) \mid \mathsf{skip} \mid \mathsf{p}\ ;\ \mathsf{p} \mid \mathsf{p} \parallel \mathsf{p}$$

(see Chapter 3) are closed under compact stability.
To conclude this section, we will examine the feedback operator $(-)^\omega$, introduced in Chapter 3, with respect to compact stability. We start with the following remark.

Remark 5.4.10. The feedback operator $(-)^\omega$ is not closed under compact stability. Take, for example, the map $f : \mathbb{R} \to \mathrm{H_c}\mathbb{R}$ defined by $f(r) = (t \mapsto r, \max(0, r))$. The evolution $f^\omega(0)$ has duration zero and the evolution $f^\omega(r)$ of every positive real number r close to zero has infinite duration.

This problem disappears if ones assumes that there exists a real number $r > 0$ such that for every $x \in X$, a hybrid program $f : X \to \mathrm{H_c}X$ outputs an evolution with duration greater than r, *i. e.* $\inf \pi_2 \cdot f[X] \neq 0$

Theorem 5.4.11. *Consider a hybrid program $f : X \to \mathrm{H_c}X$ with feedback. If the condition $\inf \pi_2 \cdot f[X] \neq 0$ holds the feedback is also a hybrid program $f^\omega : X \to \mathrm{H_c}X$.*

Proof. Using the assumptions above, we will show that the feedback of f can be written as a composition $X \to X^{\mathbb{R}_{\geq 0}} \rightarrowtail \mathrm{H_c}X$, where $X \to X^{\mathbb{R}_{\geq 0}}$ is a mediating map of a limit in Top and the injection sends a function f to (f, ∞). This is enough to prove our claim.

Let $\mathscr{D} : \mathbb{R}_{\geq 0} \to \mathsf{Top}$ be the diagram functor that sends a non-negative real number i to the space $[0, i)$ and an arrow $i \leq j$ to the corresponding inclusion map. The space $\mathbb{R}_{\geq 0}$ with the subspace inclusions $[0, i) \hookrightarrow \mathbb{R}_{\geq 0}$ is the colimit of $\mathscr{D}$. Now consider the contravariant exponential functor,

$$X^{(-)} : \mathsf{Top}^{\mathsf{op}} \to \mathsf{Top}$$

Lemma A.2.12 tells that the limit of $X^{\mathscr{D}^{\mathsf{op}}} : (\mathbb{R}_{\geq 0})^{\mathsf{op}} \to \mathsf{Top}$ is the space $X^{\mathbb{R}_{\geq 0}}$ equipped with the projections $X^{\mathbb{R}_{\geq 0}} \to X^{[0,i)}$ that restrict a function's domain to $[0, i)$. The next step is to build a suitable cone $\left(m_i : X \to X^{[0,i)}\right)_{i \in \mathbb{R}_{\geq 0}}$ for the diagram functor $X^{\mathscr{D}^{\mathsf{op}}}$.

Since the map f has a minimum time that is different from zero, there must exist a natural number $n \in \mathbb{N}$ such that, for every $x \in X$, the condition $\pi_2 \cdot f^n(x) \geq i$ holds. This naturally induces a map $m_i : X \to X^{[0,i)}$ that for every $x \in X$ outputs the evolution $f^n(x)$ restricted to the subspace $[0, i)$. Finally, we need to show that for every $i \leq j$ the diagram,

$$\begin{tikzcd}
 & X \arrow[dl, "m_i"'] \arrow[dr, "m_j"] & \\
X^{[0,i)} & & X^{[0,j)} \arrow[ll, "X^{\mathscr{D}^{\mathsf{op}}(i \leq j)}"]
\end{tikzcd}$$

commutes, but this follows directly from f being a unit-action (Chapter 3). When post-composed with the injection $X^{\mathbb{R}_{\geq 0}} \rightarrowtail \mathrm{H_c}X$, the map $\langle m_i \rangle_{i \in \mathbb{R}_{\geq 0}} : X \to X^{\mathbb{R}_{\geq 0}}$ is the feedback of f by Definition 3.4.12. □

Example 5.4.12. In Examples 5.4.8, we saw that all program operations of the time-triggered programming language there described are closed under compact stability. Now consider a program p of this language and assume that it can be put in feedback. Assume also that it can be written as a composition,

$$\mathsf{a_1} \mathbin{;} \ldots \mathbin{;} \mathsf{a_n}$$

such that at least one of the constituents $\mathsf{a_i}$ has the form $(\dot{x}_1 = t, \ldots, \dot{x}_n = t \;\&\; d)$ with $d \neq 0$. Under these assumptions, program p in feedback is compact stable. For example, the oscillator defined in Examples 3.4.13 is compact stable.

5.5 OPEN CHALLENGES

Even if most coalgebraic literature takes Set as the base, working category, state-based transition systems often demand a shift to richer categories, where more sophisticated mechanisms to handle their behaviour are available. Such was the case in [Pan09; Dob09], two research lines on the topic of stochastic systems that involved the category of measurable spaces, and in [KKV04; BFV10], where the category of Stone spaces and continuous maps plays a key role in setting an appropriate coalgebraic semantics for finitary modal logics.

In this chapter we adopted Top as the base category. One of the reasons was because the Set-based context proved to be insufficient for providing a notion of observable behaviour to representations when the condition 'as-soon-as' is dropped. Our shift to the topological setting allowed to remedy this problem up to some extent: we could provide a notion of behaviour to bounded representations, and, if certain conditions are satisfied, to other cases as well.

The topological setting also helped us to establish a notion of stability for hybrid programming: in particular, we introduced the *topological* hybrid monad, which yields the notion of compact stability – closely related and also more general than the widely acclaimed Lyapunov stability – and allows to generate different programming languages whose operators are closed under compact stability.

The chapter's work covered a broad spectrum of research, from theoretical developments in Section 5.1 and Section 5.2, to more practical ones in Section 5.3 and Section 5.4. In the process, several questions emerged for which we still lack a definitive answer. Let us summarise them next.

Final coalgebras for Vietoris functors. In Section 5.2, we examined the preservation of codirected limits by Vietoris functors in different topological contexts, showing cases in which they were preserved and cases in which they were not. Nonetheless, we are still not precisely sure what is the 'weakest' context in which they are preserved. For example, *is the Hausdorff condition necessary for the compact Vietoris functor to preserve codirected limits ? Also, can we relax the conditions on which we showed that the lower Vietoris functor preserves codirected limits ?*

Topological functors. In Section 5.1, we studied the existence of topological functors between categories of coalgebras (Theorem 5.1.22), which, among other things, allowed to prove that all categories of (sub)polynomial coalgebras over Top are complete. Since the study applies to *arbitrary* topological categories and not just Top, a natural research line is to inquiry *under which conditions one can prove that all categories of (sub)polynomials coalgebras over a topological category are complete.* Two prime examples that we are particularly interested in are the coalgebras over preordered sets, and the coalgebras over pseudometric spaces. These have significant relevance within the coalgebraic community (*e. g.* [Bal+14; BK11; BKV12]) and we believe that our study can contribute to the topic.

Beyond bounded representations. Theorem 5.3.12 tells that all bounded representations can be equipped with a notion of observable behaviour. *Is there a broader class of representations to which we can also provide this notion ?*

***F-representations in* Top.** We used the theoretical results of Section 5.2 to provide a notion of observable behaviour to deterministic representations, *i. e.* representations with $\mathrm{Id} : \mathsf{Top} \to \mathsf{Top}$

as the discrete transition type. However, these results apply to a much more general context, in particular to representations with different transition types $F : \mathsf{Top} \to \mathsf{Top}$. This raises the following interesting question: *for which discrete transition types can the corresponding representations be equipped with a canonical notion of observational behaviour ?*

Stability in hybrid programming. Finally, in Section 5.4 we showed that a topological analogue of the hybrid monad can be used to systematically generate different types of programming languages whose operators are closed under compact stability. This supports a compositional analysis of hybrid programs with respect to this notion, as exemplified with several systems in the section (*e. g.* oscillators, bouncing ball, heating systems. . .). A question that naturally arises from this work is *how do program operations behave when hybrid programs are stable* only *on certain inputs ? In particular, for which inputs is a composition of such hybrid programs stable?*

6 EPILOGUE

The writing of this note concludes the author's Ph.D project – but not his interest ! – on the topic of hybrid systems and their programming languages. The goal was to investigate these devices from a foundational point of view, by deconstructing the basic interactions between programs and physical processes, and building from, the ground up, the programming paradigm that naturally underlies them:

- we introduced different programming languages, and showed how to systematically extend them with classical program constructs. We listed all program operations (over algebraic structures) available, and showed how to build them in a simple manner, giving rise, for example, to programming languages that support superposition and different aspects of concurrency. We introduced a notion of stability to hybrid programming, showed how to reason about it compositionally, and used this feature to prove that many of the hybrid programs considered in the thesis are stable.

- We also established a component-based software development discipline in hybrid programming. Its basic constituents are hybrid programs with internal memory, for which we developed languages, notions of bisimulation, and observational behaviour. We generalised these results to provide a uniform framework of hybrid automata – the standard formalism in the project of hybrid systems. Along the process, we introduced new coalgebraic results, including proofs on the existence of canonical notions of behaviour for non-deterministic transition systems with an infinite state space.

It is our belief that these results provide a firm step towards the design of the next-generation programming languages.

One central pillar of our approach was the theory of monads: not only it was the basis for systematically generating hybrid programming languages, but it also allowed to smooth the integration of hybrid computations with other behavioural effects. The latter aspect was exemplified with partial and non-deterministic computations, which as we saw, can be combined with the hybrid case via suitable distributive laws.

The modular approach that monads provide resonates with another topic, that of *systematic combination of logics*, which the author has been pursuing in parallel to his Ph.D. The motivation for combining logics was elegantly summarised by J. Goguen and J. Meseguer in [GM87]: *"The right way to combine various programming paradigms is to discover their underlying logics, combine them, and then base a language upon the combined logic"*. The present note ends with some highlights of the author's research on this topic.

Hybrid(ised) logics. The hybridisation method [Mar+11] is a systematic process for extending a given, arbitrary logic with the typical features of hybrid logic [Bra11] – confusingly, the term 'hybrid' in the latter does *not* pertain hybrid systems, but the ability of this logic to pinpoint states of a Kripke frame. In document [Nev+16c], we further developed the hybridisation method so that it can also extend a calculus of the given, base logic into a calculus for the hybridised one. We showed that if the base calculus is complete the extended calculus will also be complete, which generalises standard completeness results of hybrid logic [Bra11].

Using the hybridisation method, we also established a methodology for the rigorous design of reconfigurable systems (those that switch between different modes of operation along their execution) from the very beginning of the development process, with natural language boilerplates, to the later stage of model verification. We also introduced and examined hierarchical hybrid logic in order to handle refinements processes in software design [Mad+18].

Renato Neves, Alexandre Madeira, Manuel A. Martins, and Luis S. Barbosa. "Proof theory for hybrid(ised) logics". In: Science of Computer Programming (2016).

Alexandre Madeira, Renato Neves, Luis S. Barbosa, and Manuel A. Martins. "A method for rigorous design of reconfigurable systems". In: Science of Computer Programming (2016).

Alexandre Madeira, Renato Neves, Manuel A. Martins, and Luis S. Barbosa. "Hierarchical hybrid logic". In: LSFA'17: Logical and Semantic Frameworks with Applications, 12th Workshop. Electronic Notes in Theoretical Computer Science, Elsevier (In press).

Asymmetric combination of logics. Asymmetric combination of logics, of which hybridisation, temporalisation [FG92], and probabilisation [Bal13] are particular cases, is a formal process for developing the characteristic features of a specific logic on top of another one, with the three cases mentioned above being prime examples. In [Nev+16a], we showed that such processes can frequently be regarded as endofunctors in a suitable category of logics and translations between them. The *op. cit.* allows to systematically extend the combination of logics to combinations of their translations and moreover opens several research avenues due to its functorial view: from (co)limit preservation by combinations to the study of their adjoints.

Renato Neves, Alexandre Madeira, Luis S. Barbosa, and Manuel A. Martins. "Asymmetric Combination of Logics is Functorial: A Survey". In: WADT'16: Recent Trends in Algebraic Development Techniques, 23rd International Workshop. Lecture Notes in Computer Science. Springer, 2016.

A PROOFS AND LEMMATA

A.1 PROOFS

Proof of Theorem 3.2.6.
We will start by showing that the equation $\mu \cdot \eta_{\mathrm{H}} = \mathrm{id}$ holds.

$$\begin{aligned}
\mu \cdot \eta_{\mathrm{H}} &= (+\!\!+) \cdot \langle \mathrm{H}\theta, \lambda_{\mathrm{H}} \rangle \cdot \eta_{\mathrm{H}} \\
&= (+\!\!+) \cdot \langle \mathrm{H}\theta \cdot \eta_{\mathrm{H}}, \lambda_{\mathrm{H}} \cdot \eta_{\mathrm{H}} \rangle \\
&= (+\!\!+) \cdot \langle \mathrm{H}\theta \cdot \eta_{\mathrm{H}}, \mathrm{id} \rangle && (\lambda \text{ is an inverse of } \eta) \\
&= (+\!\!+) \cdot \langle \eta \cdot \theta, \mathrm{id} \rangle && (\text{Naturality}) \\
&= \mathrm{id} && (\text{Lemma A.2.1})
\end{aligned}$$

Let us then show that $\mu \cdot \mathrm{H}\eta = \mathrm{id}$.

$$\begin{aligned}
\mu \cdot \mathrm{H}\eta &= (+\!\!+) \cdot \langle \mathrm{H}\theta, \lambda_{\mathrm{H}} \rangle \cdot \mathrm{H}\eta \\
&= (+\!\!+) \cdot \langle \mathrm{H}\theta \cdot \mathrm{H}\eta, \lambda_{\mathrm{H}} \cdot \mathrm{H}\eta \rangle \\
&= (+\!\!+) \cdot \langle \mathrm{H}\theta \cdot \mathrm{H}\eta, \eta \cdot \lambda \rangle && (\text{Naturality}) \\
&= (+\!\!+) \cdot \langle \mathrm{id}, \eta \cdot \lambda \rangle && (\theta \text{ is an inverse of } \eta) \\
&= \mathrm{id} && (\text{Lemma A.2.1})
\end{aligned}$$

Finally, we need to prove that the equation $\mu \cdot \mathrm{H}\mu = \mu \cdot \mu_{\mathrm{H}}$ also holds. This is a consequence of showing that the diagram below is commutative.

$$\begin{tikzcd}
\mathrm{HHH} \arrow[r, "\langle \mathrm{H}\theta_{\mathrm{H}}, \lambda_{\mathrm{HH}} \rangle"] \arrow[dd, "\mathrm{H}\mu"'] & \mathrm{HH} \times \mathrm{HH} \arrow[r, "(+\!\!+)"] \arrow[d, "\mathrm{H}\theta \times \langle \mathrm{H}\theta, \lambda_{\mathrm{H}} \rangle"'] & \mathrm{HH} \arrow[d, "\langle \mathrm{H}\theta, \lambda_{\mathrm{H}} \rangle"] \\
& \mathrm{H} \times \mathrm{H} \times \mathrm{H} \arrow[r, "(+\!\!+) \times \mathrm{id}"] \arrow[d, "\mathrm{id} \times (+\!\!+)"'] & \mathrm{H} \times \mathrm{H} \arrow[d, "(+\!\!+)"] \\
\mathrm{HH} \arrow[r, "\langle \mathrm{H}\theta, \lambda_{\mathrm{H}} \rangle"'] & \mathrm{H} \times \mathrm{H} \arrow[r, "(+\!\!+)"'] & \mathrm{H}
\end{tikzcd}$$

The right bottom square commutes by Lemma A.2.1, and the right top one by the following calculation:

$$\begin{aligned}
((+\!\!+) \times \mathrm{id}) \cdot (\mathrm{H}\theta \times \langle \mathrm{H}\theta, \lambda_{\mathrm{H}} \rangle) &= ((+\!\!+) \times \mathrm{id}) \cdot \langle \mathrm{H}\theta \times \mathrm{H}\theta, \lambda_{\mathrm{H}} \cdot \pi_2 \rangle && (\text{Product associativity}) \\
&= \langle (+\!\!+) \cdot (\mathrm{H}\theta \times \mathrm{H}\theta), \lambda_{\mathrm{H}} \cdot \pi_2 \rangle \\
&= \langle \mathrm{H}\theta \cdot (+\!\!+), \lambda_{\mathrm{H}} \cdot \pi_2 \rangle && (\text{Naturality}) \\
&= \langle \mathrm{H}\theta \cdot (+\!\!+), \lambda_{\mathrm{H}} \cdot (+\!\!+) \rangle \\
&= \langle \mathrm{H}\theta, \lambda_{\mathrm{H}} \rangle \cdot (+\!\!+)
\end{aligned}$$

Finally, the left square commutes because,

$$\begin{aligned}
\langle \mathrm{H}\theta, \lambda_{\mathrm{H}}\rangle \cdot \mathrm{H}\mu &= \langle \mathrm{H}\theta \cdot \mathrm{H}\mu, \lambda_{\mathrm{H}} \cdot \mathrm{H}\mu\rangle \\
&= \langle \mathrm{H}(\theta \cdot \mu), \mu \cdot \lambda_{\mathrm{HH}}\rangle && \text{(Naturality)} \\
&= \langle \mathrm{H}(\theta \cdot \mathrm{H}\theta), \mu \cdot \lambda_{\mathrm{HH}}\rangle && \text{(Lemma A.2.2)} \\
&= \langle \mathrm{H}\theta \cdot \mathrm{HH}\theta, \mu \cdot \lambda_{\mathrm{HH}}\rangle \\
&= \langle \mathrm{H}\theta \cdot \mathrm{H}\theta_{\mathrm{H}}, \mu \cdot \lambda_{\mathrm{HH}}\rangle && \text{(Naturality)} \\
&= (\mathrm{H}\theta \times \mu) \cdot \langle \mathrm{H}\theta_{\mathrm{H}}, \lambda_{\mathrm{HH}}\rangle
\end{aligned}$$

□

Proof of Theorem 3.4.9.

Let us start with the left triangle in the diagram below.

$$\begin{tikzcd}
\mathrm{H}_* \arrow[r, "\eta_{\mathrm{H}_*}"] \arrow[dr, "\mathrm{Id}"'] & \mathrm{H}_*\mathrm{H}_* \arrow[d, "\mu"] & \mathrm{H}_* \arrow[l, "\mathrm{H}_*\eta"'] \arrow[dl, "\mathrm{Id}"] \\
& \mathrm{H}_* &
\end{tikzcd}$$

It can be decomposed into the following diagram and therefore it is only necessary to show that the equation $\mu_1 \cdot \eta^{\mathrm{H}} = \mathrm{Id}$ holds. This is straightforward because the arrow μ_1 is essentially the multiplication map of the hybrid monad $\mathbb{H}$.

$$\begin{tikzcd}
\mathrm{H}_* \arrow[r, "\eta^{\mathrm{H}}"] \arrow[drr, bend right, "\mathrm{Id}"'] & \mathrm{HH}_* \arrow[r, tail] \arrow[dr, "\mu_1"'] & \mathrm{H}_*\mathrm{H}_* \arrow[d, "\mu"] \\
& & \mathrm{H}_*
\end{tikzcd}$$

Let us now focus on the right triangle of the previous unit diagram: it can be decomposed into the following diagram.

$$\begin{tikzcd}
\mathrm{I} \arrow[r, "\mathrm{I}\eta"] \arrow[d, tail] & \mathrm{IH}_* \arrow[d, tail] \arrow[dr, bend left, "\mu_2"] & \\
\mathrm{H}_* \arrow[r, "\mathrm{H}_*\eta"] & \mathrm{H}_*\mathrm{H}_* \arrow[r, "\mu"] & \mathrm{H}_* \\
\mathrm{H} \arrow[u, tail] \arrow[r, "\mathrm{H}\eta"'] & \mathrm{HH}_* \arrow[u, tail] \arrow[ur, bend right, "\mu_1"'] &
\end{tikzcd}$$

The transformation θ is an inverse of η and the equation $\mu_2 = i_2 \cdot \mathrm{I}\theta$ holds. It follows that $\mu_2 \cdot \mathrm{I}\eta = i_2$. Since, the triple $(\mathrm{H}, \eta^{\mathrm{H}}, \mu^{\mathrm{H}})$ is a monad and the arrow μ_1 behaves essentially as μ^{H}, the equation $\mu_1 \cdot \mathrm{H}\eta = i_1$ must also hold. It then follows by the universal property of coproducts that $\mu \cdot \mathrm{H}_*\eta = \mathrm{Id}$. Next, we need to prove that the following diagram commutes.

$$\begin{tikzcd}
\mathrm{H}_*\mathrm{H}_*\mathrm{H}_* \arrow[r, "\mathrm{H}_*\mu"] \arrow[d, "\mu"'] & \mathrm{H}_*\mathrm{H}_* \arrow[d, "\mu"] \\
\mathrm{H}_*\mathrm{H}_* \arrow[r, "\mu"'] & \mathrm{H}_*
\end{tikzcd}$$

It can be unravelled into the diagram,

$$\begin{tikzcd}
\mathrm{HH_*H_*} \arrow[r, tail] \arrow[d, "\mathrm{H}\mu"'] \arrow[dr, bend right, "\mu_{1\mathrm{H_*}}"'] & \mathrm{H_*H_*H_*} \arrow[d, dotted, "\mu_{\mathrm{H_*}}"] & \mathrm{IH_*H_*} \arrow[l, tail] \arrow[d, "\mathrm{I}\mu"] \arrow[dl, bend left, "\mu_{2\mathrm{H_*}}"] \\
 & \mathrm{H_*H_*} \arrow[d, "\mu"] & \\
\mathrm{HH_*} \arrow[r, "\mu_1"'] & \mathrm{H_*} & \mathrm{IH_*} \arrow[l, "\mu_2"]
\end{tikzcd}$$

which tells that the proof is finished once shown that the following equations hold.

$$\mu_2 \cdot \mathrm{I}\mu = \mu \cdot \mu_{2\mathrm{H_*}} \tag{17}$$

$$\mu_1 \cdot \mathrm{H}\mu = \mu \cdot \mu_{1\mathrm{H_*}} \tag{18}$$

So let us start with Equation (17), which corresponds to the diagram below.

$$\begin{tikzcd}
\mathrm{H_*H_*} \arrow[d, "\mu"'] & \mathrm{IH_*} \arrow[l, tail] \arrow[dl, "\mu_2"] & \mathrm{IH_*H_*} \arrow[l, "\mathrm{I}\theta_{\mathrm{H_*}}"'] \arrow[d, "\mathrm{I}\mu"] \\
\mathrm{H_*} & & \mathrm{IH_*} \arrow[ll, "\mu_2"]
\end{tikzcd}$$

It commutes because the equations $\theta \cdot \mu = \theta \cdot \mathrm{H_*}\theta = \theta \cdot \theta_{\mathrm{H_*}}$ hold (Lemma A.2.3 and naturality of θ).

Then, consider Equation (18). In order to prove that it holds, we will resort to the subfunctors F, G of $\mathrm{HH_*H_*}$ that are defined by,

$$FX = \big\{(f,d) \in \mathrm{HH_*H_*}X \mid f(d) \in \mathrm{HH_*}X\big\}$$
$$GX = \big\{(f,d) \in \mathrm{HH_*H_*}X \mid f(d) \in \mathrm{IH_*}X\big\}$$

Observe that $F + G \simeq \mathrm{HH_*H_*}$ and that the compositions,

$$F \xhookrightarrow{\iota_F} \mathrm{HH_*H_*} \xrightarrow{\mu_{1\mathrm{H_*}}} \mathrm{H_*H_*} \qquad\qquad G \xhookrightarrow{\iota_G} \mathrm{HH_*H_*} \xrightarrow{\mu_{1\mathrm{H_*}}} \mathrm{H_*H_*}$$

factorise through the injections $\mathrm{HH_*} \rightarrowtail \mathrm{H_*H_*}$ and $\mathrm{IH_*} \rightarrowtail \mathrm{H_*H_*}$, respectively. This leads to the diagram below and we need to show that it commutes.

$$\begin{tikzcd}
F \arrow[dr, hook, "\iota_F"'] \arrow[rrr, dotted, bend left, "\mu_{1\mathrm{H_*}}"] & & G \arrow[dl, hook', "\iota_G"] \arrow[rrr, dotted, bend left, "\mu_{1\mathrm{H_*}}"] & \mathrm{HH_*} \arrow[dr, tail] \arrow[ddr, bend right, "\mu_1"'] & & \mathrm{IH_*} \arrow[dl, tail] \arrow[ddl, bend left, "\mu_2"] \\
 & \mathrm{HH_*H_*} \arrow[rrr, "\mu_{1\mathrm{H_*}}"] \arrow[d, "\mathrm{H}\mu"'] & & & \mathrm{H_*H_*} \arrow[d, dotted, "\mu"] & \\
 & \mathrm{HH_*} \arrow[rrr, "\mu_1"'] & & & \mathrm{H_*} &
\end{tikzcd}$$

Since the triple $(\mathrm{H}, \eta^{\mathrm{H}}, \mu^{\mathrm{H}})$ is a monad and the arrow μ_1 is essentially equal to μ^{H}, it is straightforward to show that the equation $\mu_1 \cdot \mathrm{H}\mu \cdot \iota_F = \mu_1 \cdot \mu_{1\mathrm{H}_*}$ holds. Thus, it remains to show that the equation $\mu_1 \cdot \mathrm{H}\mu \cdot \iota_G = \mu_2 \cdot \mu_{1\mathrm{H}_*}$ holds as well. So we calculate,

$$\begin{aligned}
\mu_1 \cdot \mathrm{H}\mu \cdot \iota_G(f,d) &= (\theta \cdot \mu \cdot f, d) \mathbin{+\!\!+} \mu(f\ d) \\
&= (\theta \cdot \mu \cdot f, d) \mathbin{+\!\!+} (\mathrm{H}_*\theta\,(f\ d)) && ((f,d) \in GX) \\
&= (\theta \cdot \mathrm{H}_*\theta \cdot f, d) \mathbin{+\!\!+} (\mathrm{H}_*\theta\,(f\ d)) && (\text{Lemma A.2.3}) \\
&= (\theta \cdot \theta_{\mathrm{H}_*} \cdot f, d) \mathbin{+\!\!+} (\mathrm{H}_*\theta\,(f\ d)) && (\text{Naturality}) \\
&= \mathrm{H}_*\theta\,((\theta_{\mathrm{H}_*} \cdot f, d) \mathbin{+\!\!+} (f\ d)) && (\text{Naturality}) \\
&= \mu_2 \cdot \mu_{1\mathrm{H}_*}(f,d)
\end{aligned}$$

□

Proof of Theorem 3.4.16.
Observe that the following diagram commutes.

$$\begin{tikzcd}
(\mathbb{N}_0, \leq) \arrow[r, "f^{(\,-\,)}"] \arrow[drr, bend right, "\langle f^{(\,-\,)} \bullet f(x)\rangle_{x\in X}"'] & (\mathrm{End}_{\mathrm{H}_*}(X), \sqsubseteq) \arrow[r, "(\,-\,\bullet f)"] & (\mathrm{End}_{\mathrm{H}_*}(X), \sqsubseteq) \arrow[d, tail, two heads, "\langle \pi_x\rangle_{x\in X}"] \\
 & & (\mathrm{H}_*X, \sqsubseteq)
\end{tikzcd}$$

According to it, we just need to show that the following equation holds.

$$\langle \pi_x\rangle_{x\in X}\left(\left(\mathrm{colim}\ f^{(\,-\,)}\right) \bullet f\right) = \mathrm{colim}\ \langle f^{(\,-\,)} \bullet f(x)\rangle_{x\in X}$$

More concretely, that $f^\omega \bullet f(x) = \mathrm{colim}\ f^{(\,-\,)} \bullet f(x)$ for every element $x \in X$. So first assume that $f(x)$ is an evolution with infinite duration. For this case, the colimit of the chain,

$$(\mathbb{N}_0, \leq) \xrightarrow{f^{(\,-\,)} \bullet f(x)} (\mathrm{H}_*X, \sqsubseteq)$$

is simply $f(x)$ and clearly the equation $f^\omega \bullet f(x) = f(x)$ holds. Now assume that $f(x)$ is an evolution with finite duration and denote it by $(f(x, -\,), d)$. Then reason,

$$\begin{aligned}
f^\omega \bullet f(x) &= f(x, -\,) \mathbin{+\!\!+} f^\omega(f(x,d)) \\
&= f(x, -\,) \mathbin{+\!\!+} \left(\mathrm{colim}\ f^{(\,-\,)}\,(f(x,d))\right) \\
&= \mathrm{colim}\ \left(f(x, -\,) \mathbin{+\!\!+} f^{(\,-\,)}\,(f(x,d))\right) && (\text{Lemma A.2.4}) \\
&= \mathrm{colim}\ \left(f^{(\,-\,)} \bullet f(x)\right)
\end{aligned}$$

□

Proof of Theorem 3.5.2.
It is straightforward to prove that $\delta : \mathrm{HQ} \to \mathrm{QH}$ is a natural transformation, and that it makes the following diagram commute.

$$\begin{tikzcd}
 & \mathrm{H} \arrow[dl, "\mathrm{H}\eta^{\mathrm{Q}}"'] \arrow[dr, "\eta^{\mathrm{Q}}_{\mathrm{H}}"] & \\
\mathrm{HQ} \arrow[rr, "\delta"'] & & \mathrm{QH} \\
 & \mathrm{Q} \arrow[ul, "\eta^{\mathrm{H}}_{\mathrm{Q}}"] \arrow[ur, "\mathrm{Q}\eta^{\mathrm{H}}"'] &
\end{tikzcd}$$

So we will just that the natural transformation $\delta : \mathrm{HQ} \to \mathrm{QH}$ also makes the following diagram commute.

$$\begin{tikzcd}
\mathrm{HQQ} \arrow[r, "\delta_{\mathrm{Q}}"] \arrow[d, "\mathrm{H}\mu"'] & \mathrm{QHQ} \arrow[r, "\mathrm{Q}\delta"] & \mathrm{QQH} \arrow[d, "\mu_{\mathrm{Q}}"] \\
\mathrm{HQ} \arrow[rr, "\delta"'] & & \mathrm{QH} \\
\mathrm{HHQ} \arrow[u, "\mu_{\mathrm{Q}}"] \arrow[r, "\mathrm{H}\delta"'] & \mathrm{HQH} \arrow[r, "\delta_{\mathrm{H}}"'] & \mathrm{QHH} \arrow[u, "\mathrm{Q}\mu"']
\end{tikzcd}$$

Start with the upper square by considering an element $(f, d) \in \mathrm{HQQ}X$. A straightforward calculation provides the following equations.

$$\begin{aligned}
\delta_X \cdot \mathrm{H}\mu_X(f, d) &= \{ (g, d) \in \mathrm{H}X \mid g \in \cup \cdot f \} \\
\mu_{\mathrm{Q}X} \cdot \mathrm{Q}\delta_X \cdot \delta_{\mathrm{Q}X}(f, d) &= \bigcup \{ \delta_X\,(h, d) \mid (h, d) \in \mathrm{HQ}X \wedge h \in f \}
\end{aligned}$$

and so we just need to prove that both sets are the same. For this, start with an element $(g, d) \in \mathrm{H}X$, and reason,

$$\begin{aligned}
g \in \cup \cdot f &\equiv \forall t \in \mathbb{R}_{\geq 0}.\, g\,(t) \in \cup \cdot f \\
&\equiv \forall t \in \mathbb{R}_{\geq 0}.\, \exists A \in f\,(t)\,.\, g\,(t) \in A \\
&\equiv \exists (h, d) \in \mathrm{HQ}X.\, g \in h \wedge h \in f \\
&\equiv \exists (h, d) \in \mathrm{HQ}X.\, (g, d) \in \delta_X\,(h, d) \wedge h \in f
\end{aligned}$$

Both sets are therefore indeed the same. Let us now concentrate on the lower square. Consider an element $(f, d) \in \mathrm{HHQ}X$ and let the expression $\pi_2 \cdot f(d)$ be denoted by e. A straightforward calculation shows that the following equations hold.

$$\begin{aligned}
\delta_X \cdot \mu_{\mathrm{Q}X}\,(f, d) &= \left\{ (g, d + e) \in \mathrm{H}X \mid g \in (\theta_{\mathrm{Q}X} \cdot f, d) +\!\!+ (f(d)) \right\} \\
\mathrm{Q}\mu_X \cdot \delta_{\mathrm{H}X} \cdot \mathrm{H}\delta_X\,(f, d) &= \left\{ (\theta_X \cdot g, d) +\!\!+ (g(d)) \mid (g, d) \in \mathrm{HH}X \wedge g \in \delta_X \cdot f \right\}
\end{aligned}$$

We will show that both sets are also the same: start with an element $(h, d+e) \in \mathrm{H}X$, and reason as follows.

$$\begin{aligned}
& h \in (\theta_{\mathrm{Q}X} \cdot f, d) +\!\!+ (f(d)) \\
\equiv\ & \forall t \leq d\ .\ h\,(t) \in \theta_{\mathrm{Q}X} \cdot f\,(t)\ \wedge\ \forall t > d\ .\ h\,(t) \in (f(d))\,(t-d) \\
\equiv\ & \forall t \leq d\ .\ h\,(t) \in \mathrm{Q}\theta_X \cdot \delta_X \cdot f\,(t)\ \wedge\ \exists (g,e) \in \delta_X\,(f(d))\ .\forall t > d\ .\ h\,(t) = g\,(t-d) \\
\equiv\ & \exists (g,d) \in \mathrm{HH}X\ .\ \forall t \leq d\ .\ g\,(t) \in \delta_X\,(f(t))\ \wedge\ \theta_X \cdot g\,(t) = h\,(t)\ \wedge\ \forall t > d\ .\ h\,(t) = (g(d))\,(t-d) \\
\equiv\ & \exists (g,d) \in \mathrm{HH}X\ .\ (h, d+e) = (\theta_X \cdot g, d) +\!\!+ (g(d))\ \wedge\ g \in \delta_X \cdot f
\end{aligned}$$

□

Proof of Theorem 4.2.15.
We will show that the map $f : M \to \mathrm{Exp}$ that associates each mode with its expression makes the diagram below commute.

$$\begin{tikzcd}
M \arrow[r, "f"] \arrow[d, "\overline{\langle \mathrm{nxt}, \mathrm{out} \rangle}"'] & \mathrm{Exp} \arrow[d, "\delta"] \\
(M \times \mathrm{Cmd})^I \arrow[r, "(f \times \mathrm{id})^I"'] & (\mathrm{Exp} \times \mathrm{Cmd})^I
\end{tikzcd}$$

After this we simply need to recall Examples 4.1.4 (5) to finish the proof. First, in order to keep the notation simple, for a $(-\times \mathrm{Cmd})^I$-coalgebra $(S, \overline{\langle a, b\rangle})$ denote the function $a(-, i)$ by $(-)_i$. Then, note that for every input $i \in I$ the equation $\mathrm{out}(m_l, i) = \mathrm{out}(\epsilon_l, i)$ always holds. It remains to show that the equation $(m_l)_i = (\epsilon_l)_i$ also holds. So let us reason,

$$\begin{aligned}
(A_l^n)_i =\ & ((\mu m_l.\psi_l)\{A_1^0/m_1\}\ldots\{A_n^{n-1}/m_n\})_i \\
& \text{(Definition of } (\_)_i) \\
=\ & (\psi_l\{A_1^0/m_1\}\ldots\{A_{l-1}^{l-2}/m_{l-1}\}\{A_{l+1}^l/m_{l+1}\}\ldots\{A_n^{n-1}/m_n\}[A_l^n/m_l])_i \\
& \text{(Definition of } \psi_l \text{ and } A_l^n \text{ has no free variables)} \\
=\ & (\psi_l\{A_1^0/m_1\}\ldots\{A_{l-1}^{l-2}/m_{l-1}\}[A_l^n/m_l]\{A_{l+1}^l/m_{l+1}\}\ldots\{A_n^{n-1}/m_n\})_i \\
& (A_l^n \text{ has no free variables}) \\
=\ & (\psi_l\{A_1^0/m_1\}\ldots\{A_{l-1}^{l-2}/m_{l-1}\}\{A_l^n/m_l\}\{A_{l+1}^l/m_{l+1}\}\ldots\{A_n^{n-1}/m_n\})_i \\
& (*) \\
=\ & (\psi_l\{A_1^0/m_1\}\ldots\{A_{l-1}^{l-2}/m_{l-1}\}\{A_l^{l-1}/m_l\}\{A_{l+1}^l/m_{l+1}\}\ldots\{A_n^{n-1}/m_n\})_i \\
& \text{(Definition of } (\_)_i \text{ and definition of } \psi_l) \\
=\ & A_k^{k-1}\{A_{k+1}^k/x_{k+1}\}\ldots\{A_n^{n-1}/x_n\} \\
=\ & A_k^n
\end{aligned}$$

The step $(*)$ relies on the equality,

$$\varphi\{B\{C_1/y_1\}\ldots\{C_n/y_n\}/x\}\{C_1/y_1\}\ldots\{C_n/y_n\} = \varphi\{B/x\}\{C_1/y_1\}\ldots\{C_n/y_n\}$$

which always holds if φ does not bind the variables $y_1, \ldots, y_n$ and all C_i do not introduce free variables y_i. Note that A_k^{k-1} can only bind the variables $m_1 \leq m \leq m_k$.

□

Proof of Lemma 5.1.21.
Let $(f_i : (X, c) \to (Y_i, d_i))_{i\in I}$ be a cone in $\mathsf{CoAlg}\left(\overline{F}\right)$ and $(f_i : X \to Y_i)_{i\in I}$ be initial with respect to $U : \mathsf{A} \to \mathsf{B}$. Consider another cone $(g_i : (Z, e) \to (Y_i, d_i))_{i\in I}$ in $\mathsf{CoAlg}\left(\overline{F}\right)$ and assume that its $\overline{U}$-image is factorised as shown by the diagram below.

$$\begin{tikzcd}
\overline{U}(Z,e) \arrow[d, "h"'] \arrow[dr, "\overline{U}g_i"] & \\
\overline{U}(X,c) \arrow[r, "\overline{U}f_i"'] & \overline{U}(Y_i,d_i)
\end{tikzcd} \tag{19}$$

The canonical forgetful functor $\mathsf{CoAlg}\,(F) \to \mathsf{B}$ yields the following factorisation of the cone $(Ug_i : UZ \to UY_i)_{i\in I}$.

$$\begin{tikzcd}
UZ \arrow[d, "h"'] \arrow[dr, "Ug_i"] & \\
UX \arrow[r, "Uf_i"'] & UY_i
\end{tikzcd}$$

Since the cone $(f_i : X \to Y_i)_{i\in I}$ is initial with respect to $U : \mathsf{A} \to \mathsf{B}$, there exists a unique arrow $\overline{h} : Z \to X$ in A such that for all $i \in I$ the following equations hold.

$$g_i = f_i \cdot \overline{h} \qquad\qquad U\overline{h} = h$$

It remains to show that the arrow $\overline{h} : Z \to X$ is also a coalgebra homomorphism $\overline{h} : (Z, e) \to (X, c)$. Diagram (19) tells that the equation $Fh \cdot \delta_Z \cdot Ue = \delta_X \cdot Uc \cdot h$ holds, which gives,

$$\begin{aligned}
Fh \cdot \delta_Z \cdot Ue = \delta_X \cdot Uc \cdot h &\equiv FU\overline{h} \cdot \delta_Z \cdot Ue = \delta_X \cdot Uc \cdot U\overline{h} \\
&\equiv \delta_X \cdot U\overline{F}\,\overline{h} \cdot Ue = \delta_X \cdot Uc \cdot U\overline{h} && \text{(Naturality)} \\
&\Rightarrow U\overline{F}\,\overline{h} \cdot Ue = Uc \cdot U\overline{h} && (\delta \text{ is mono}) \\
&\equiv U(\overline{F}\,\overline{h} \cdot e) = U(c \cdot \overline{h}) \\
&\Rightarrow \overline{F}\,\overline{h} \cdot e = c \cdot \overline{h} && (U \text{ is faithful})
\end{aligned}$$

□

Proof of Theorem 5.1.31.
Recall the definition of coreflection (*e. g.* [AHS09, Definition 4.25]) and observe that the supremum of a set of G-subcoalgebras of (Y, d) always exists (Theorem 4.1.13). Since there exists

a fully faithful functor $\mathsf{CoAlg}\,(F) \to \mathsf{CoAlg}\,(G)$, we will interpret $\mathsf{CoAlg}\,(F)$ as a full subcategory of $\mathsf{CoAlg}\,(G)$.

Now, let us build the supremum $C(Y,d)$ of G-subcoalgebras of (Y,d) as described in the theorem's statement. According to Lemma 5.1.30, the coalgebra $C(Y,d)$ lives in $\mathsf{CoAlg}\,(F)$, and since it is a subcoalgebra of (Y,d), there exists a G-homomorphism $c : C(Y,d) \rightarrowtail (Y,d)$ in M.

Then, consider an F-coalgebra (Z,z) with a G-homomorphism $f : (Z,z) \to (Y,d)$. We need to show that it can be uniquely written as a composition,

$$\begin{tikzcd} (Z,z) \arrow[r] & C(Y,d) \arrow[r, tail, "c"] & (Y,d) \end{tikzcd}$$

with the first morphism living in $\mathsf{CoAlg}\,(F)$. The factorisation of $f : (Z,z) \to (Y,d)$ yields a subcoalgebra of (Y,d) that is smaller than $C(Y,d)$. By Lemma 5.1.30, this subcoalgebra also lives in $\mathsf{CoAlg}\,(F)$. The claim then follows directly from $c : C(Y,d) \rightarrowtail (Y,d)$ being mono.

$\square$

Proof of Theorem 5.2.17.
Let us consider a partially ordered compact space $(X,\leq,\tau)$ and its stably compact correspondent (X,σ). The underlying set of $\mathrm{V}(X,\sigma)$ is the set of all lower-closed subsets of $(X,\leq,\tau)$, because the opens of (X,σ) are precisely the upper-opens of $(X,\leq,\tau)$.

We will now show that the specialisation order of $\mathrm{V}(X,\sigma)$ is the inclusion $\subseteq$. Take two lower-closed subsets $A, B \subseteq X$ and assume that $A \subseteq B$. It follows that for every upper-open U of $(X,\leq,\tau)$ the condition $A \in U^{\Diamond} \Rightarrow B \in U^{\Diamond}$ holds, which proves that $A \in \overline{\{B\}}$. Conversely, assume that $A \in \overline{\{B\}}$. Since B is lower-closed, the condition $A \in (X\backslash B)^{\Diamond} \Rightarrow B \in (X\backslash B)^{\Diamond}$ holds and consequently $A \subseteq B$.

Let us now concentrate on maps in $\mathsf{PosComp}$. Document [Nac65, Proposition 4] shows that for every function $f : X \to Y$ in $\mathsf{PosComp}$ and every lower-closed subset $A \subseteq X$, the down-closure $\downarrow f[A]$ is closed in Y. Observe that every closed set of (Y,σ) that contains $f[A]$ must be down-closed and that $\downarrow f[A]$ is the smallest such set.

It remains to prove that the patch topology of $\mathrm{V}(X,\sigma)$ coincides with the topology defined by (15). First note that every set of the form,

$$\{A \subseteq X \mid A \text{ lower-closed and } A \cap U \neq \varnothing\} \quad (U \subseteq X \text{ upper-open}),$$

is open in $\mathrm{V}(X,\sigma)$ and therefore is also open in the patch topology. For $K \subseteq X$ upper-closed, the complement of the set,

$$\{A \subseteq X \mid A \text{ lower-closed and } A \cap K = \varnothing\}$$

is equal to $K^{\Diamond}$. Using Alexander's Subbase Theorem, it is straightforwad to verify that $K^{\Diamond}$ is compact in $\mathrm{V}(X,\sigma)$, because K is compact in X and the downclosure $\downarrow k$ of an element $k \in X$ is lower-closed in X. Since the specialisation order of $\mathrm{V}(X,\sigma)$ is subset inclusion, $K^{\Diamond}$ is saturated in $\mathrm{V}(X,\sigma)$ [Law11, page 127]. The topology generated by (15) is thus coarser than the patch topology of $\mathrm{V}(X,\sigma)$. Since the former is also Hausdorff, by [Jun04, Lemma 2.2 (i)], both topologies coincide [Gou13, Theorem 4.4.27].

$\square$

Proof of Proposition 5.2.19.
It is straightforward to show that the inequality $\downarrow\bigcap_{A\in\mathcal{A}} A \subseteq \bigcap_{A\in\mathcal{A}} \downarrow A$ holds because $\downarrow A = A$ for every $A \in \mathcal{A}$. In order to prove that the reverse inequality also holds, consider an element $z \in \bigcap_{A\in\mathcal{A}} \downarrow A$. For every $A \in \mathcal{A}$, the set $\uparrow z \cap A$ is non-empty, and closed because the singleton $\{z\}$ is compact [Nac65, Proposition 4]. Since $\mathcal{A}$ is codirected, the set $\{\uparrow z \cap A \mid A \in \mathcal{A}\}$ has the finite intersection property. Therefore, by compactness, it follows that $\uparrow z \cap \bigcap_{A\in\mathcal{A}} A \neq \varnothing$, which implies that $z \in \downarrow\bigcap_{A\in\mathcal{A}} A$.

$\square$

Proof of Proposition 5.3.4.
It follows from the exponential's universal property that every component $\eta_X : X \to \mathrm{H_c}X$ is continuous. So it remains to show that all components $\mu_X : \mathrm{H_c}\mathrm{H_c}X \to \mathrm{H_c}X$ are continuous as well. Via straightforward calculations, one proves that for every space X the function μ_X is defined by,

$$\mu_X(f,d) = \big(t \mapsto g\left(\min(t,d), t \ominus d\right), m(f,d)\big)$$

where $\ominus : \mathbb{R}_{\geq 0} \times [0,\infty] \to \mathbb{R}_{\geq 0}$ is the truncated subtraction, m is the map defined by,

$$m(f,d) = \begin{cases} (\pi_2 \cdot f(d)) + d & \text{if } d \neq \infty \\ \infty & \text{otherwise} \end{cases}$$

and $g : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \to X$ is the function obtained from f by via the continuous map $\mathrm{H_c}\pi_1$ and the property of exponential functors being product preserving. Therefore, the proof is finished once shown that the three maps min, $\ominus$, and m are continuous. This is achieved by an application of Lemma A.2.10.

$\square$

Proof of Proposition 5.3.8.
Let X and Y be topological spaces. For all $S \in \mathrm{V}X$ and $y \in Y$, since S is compact, the product $S \times \{y\}$ is also compact, which entails that $S \times \{y\} \in \mathrm{V}(X \times Y)$. Then, continuity of the map $\tau_{X,Y}$ is a direct consequence of the equalities below.

$$\tau^{-1}_{X,Y}\left[\Big(\bigcup_{i\in I} U_i \times V_i\Big)^{\Diamond}\right] = \bigcup_{i\in I} (U_i)^{\Diamond} \times V_i$$

$$\tau^{-1}_{X,Y}\left[\Big(\bigcup_{i\in I} U_i \times V_i\Big)^{\Box}\right] = \bigcup\left\{\Big(\bigcup_{i\in F} U_i\Big)^{\Box} \times \bigcap_{i\in F} V_i \mid F \subseteq I \text{ finite}\right\}$$

The proof that all naturality squares commute is straightforward because this mapping is essentially the same than the tensorial strength for the powerset functor.

□

Proof of Theorem 5.4.9.
First, observe that there exists a retraction,

$$\begin{tikzcd} \mathrm{H_c} \arrow[r, hook, bend left] & (-)^{\mathbb{R}_{\geq 0}} \times [0,\infty] \arrow[l, two heads, bend left, "\lceil - \rceil"] \end{tikzcd}$$

with the epimorphism defined at each space X by $\lceil (f,d) \rceil_X = (f \cdot \min(-,d), d)$. Then let F be the set of functions,

$$m : [0,\infty] \times [0,\infty] \to \mathrm{H_c}(\mathbb{R}_{\geq 0} + \mathbb{R}_{\geq 0})$$

such that for every $i,j \in [0,\infty]$, $m(i,j)$ is a pair (f,k) with $f[\mathbb{R}_{\geq 0}] \subseteq [0,i] + [0,j]$. Let also F_c be the subset of F whose maps are continuous. Lemma A.2.11 provides the diagram below.

$$\begin{tikzcd} {[\mathsf{Top},\mathsf{Top}](\mathrm{H_c} \times \mathrm{H_c}, \mathrm{H_c})} \arrow[r, hook] & {[\mathsf{Top},\mathsf{Set}](\mathrm{UH_c} \times \mathrm{UH_c}, \mathrm{UH_c})} \arrow[d, "\simeq"] \\ F_\mathrm{c} \arrow[r, hook] & F \end{tikzcd}$$

Our strategy is to show that the composition in the upper right corner factorises through the inclusion $F_\mathrm{c} \hookrightarrow F$, and that the composition in the bottom right corner factorises through the other inclusion $[\mathsf{Top},\mathsf{Top}](\mathrm{H_c} \times \mathrm{H_c}, \mathrm{H_c}) \hookrightarrow [\mathsf{Top},\mathsf{Set}](\mathrm{UH_c} \times \mathrm{UH_c}, \mathrm{UH_c})$.

Now, Lemma A.2.11 tells that the composition in the upper right corner sends a natural transformation $\alpha : \mathrm{H_c} \times \mathrm{H_c} \to \mathrm{H_c}$ to the map $m : [0,\infty] \times [0,\infty] \to \mathrm{H_c}(\mathbb{R}_{\geq 0} + \mathbb{R}_{\geq 0})$ defined by,

$$m(i,j) = \alpha_{\mathbb{R}_{\geq 0} + \mathbb{R}_{\geq 0}}(\lceil (i_1, i) \rceil, \lceil (i_2, j) \rceil)$$

with $i_1, i_2 : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0} + \mathbb{R}_{\geq 0}$ the usual injections into the non-negative reals. This makes easy to see that m is a composition of continuous maps and therefore it is continuous.
Let us now focus on the composition in the bottom right corner. Pick a continuous map $m : [0,\infty] \times [0,\infty] \to \mathrm{H_c}(\mathbb{R}_{\geq 0} + \mathbb{R}_{\geq 0})$. In Lemma A.2.11, we saw that the composition sends this map to the natural transformation $\alpha^m : \mathrm{H_c} \times \mathrm{H_c} \to \mathrm{H_c}$ that is defined at each space X by,

$$\alpha^m_X((f,i),(g,j)) = ([f,g] \cdot (\pi_1 \cdot m(i,j)), \pi_2 \cdot m(i,j))$$

Alternatively, α^m_X can be written as the composition of continuous maps below, which proves its continuity.

$$\begin{aligned} \mathrm{H_c}X \times \mathrm{H_c}X &\overset{(*)}{\rightarrowtail} X^{\mathbb{R}_{\geq 0}} \times X^{\mathbb{R}_{\geq 0}} \times [0,\infty] \times [0,\infty] \\ &\simeq X^{\mathbb{R}_{\geq 0} + \mathbb{R}_{\geq 0}} \times [0,\infty] \times [0,\infty] \\ &\xrightarrow{\mathrm{id} \times m} X^{\mathbb{R}_{\geq 0} + \mathbb{R}_{\geq 0}} \times \mathrm{H_c}(\mathbb{R}_{\geq 0} + \mathbb{R}_{\geq 0}) \\ &\hookrightarrow X^{\mathbb{R}_{\geq 0} + \mathbb{R}_{\geq 0}} \times (\mathbb{R}_{\geq 0} + \mathbb{R}_{\geq 0})^{\mathbb{R}_{\geq 0}} \times [0,\infty] \\ &\xrightarrow{(\,\cdot\,) \times \mathrm{id}} X^{\mathbb{R}_{\geq 0}} \times [0,\infty] \\ &\overset{\lceil - \rceil}{\twoheadrightarrow} \mathrm{H_c}X \end{aligned}$$

The map $(*)$ arises from the inclusion $\mathrm{H_c}X \hookrightarrow X^{\mathbb{R}_{\geq 0}} \times [0,\infty]$ and the swap operation $A \times B \rightarrowtail B \times A$. The isomorphism results from coproducts of locally compact spaces being locally compact as well.

$\square$

A.2 LEMMATA

Lemma A.2.1. *The following conditions hold for every set X.*

(i) Every evolution $(f,d) \in \mathrm{H}X$ has $(\underline{f(0)},0)$ as a left unit and $(\underline{f(d)},0)$ as a right unit.

(ii) Consider three evolutions $(f,d),(g,e),(h,i) \in \mathrm{H}X$. If $f(d) = g(0)$ and $g(e) = h(0)$ then $(f \mathbin{+\!\!+} g) \mathbin{+\!\!+} h = f \mathbin{+\!\!+} (g \mathbin{+\!\!+} h)$.

Proof. Consider two intervals $[0,d]$ and $[0,e]$. They induce the pushout depicted in the diagram below.

$$\begin{tikzcd}
1 \arrow[r, "0"] \arrow[d, "d"'] & {[0,e]} \arrow[d, "(+d)"] \arrow[ddr, bend left, "g"] & \\
{[0,d]} \arrow[r, hook] \arrow[drr, bend right, "f"'] & {[0,d+e]} \arrow[dr, dotted, "f \mathbin{+\!\!+} g"] & \\
 & & X
\end{tikzcd}$$

Using this property, it is easy to show that $(\underline{f(0)},0)$ is a left unit for (f,d). In fact, the pairs $(\underline{f(0)},0)$, (f,d) give rise to the commutative diagram,

$$\begin{tikzcd}
1 \arrow[r, "0"] \arrow[d, "0"'] & {[0,d]} \arrow[d, "(+0)"] \arrow[ddr, bend left, "f"] & \\
{[0,0]} \arrow[r, hook] \arrow[drr, bend right, "\underline{f(0)}"'] & {[0,0+d]} \arrow[dr, dotted, "\underline{f(0)} \mathbin{+\!\!+} f"] & \\
 & & X
\end{tikzcd}$$

and since $(+0) = \mathrm{id}$, the equation $\underline{f(0)} \mathbin{+\!\!+} f = f$ must hold. An analogous reasoning shows that $(\underline{f(d)},0)$ is a right unit for (f,d).

In order to show that (ii) holds, we will prove that both morphisms $(f \mathbin{+\!\!+} g) \mathbin{+\!\!+} h$ and $f \mathbin{+\!\!+} (g \mathbin{+\!\!+} h)$ mediate the diagram of pushouts below.

$$\begin{tikzcd}
 & & 1 \arrow[r, bend left, "0"] \arrow[ddd, bend right, "d+e"'] & {[0,i]} & \\
 & & 1 \arrow[r, "0"] \arrow[d, "e"] & {[0,i]} \arrow[d, "+e"] & \\
1 \arrow[rrr, bend left, "0"] \arrow[dr, bend right, "d"'] & 1 \arrow[r, "0"] \arrow[d, "d"] & {[0,e]} \arrow[r, hook] \arrow[d, "+d"] & {[0,e+i]} \arrow[d, "+d"] \arrow[ddr, bend left, "g \mathbin{+\!\!+} h"] & \\
 & {[0,d]} \arrow[r, hook] & {[0,d+e]} \arrow[r, hook] \arrow[drr, bend right, "f \mathbin{+\!\!+} g"'] & {[0,d+e+i]} \arrow[dr, dotted] & \\
 & & & & X
\end{tikzcd}$$

Start with $(f \mathbin{+\!\!+} g) \mathbin{+\!\!+} h$. We need to prove that the following system of equations holds.

$$\begin{cases} ((f \mathbin{+\!\!+} g) \mathbin{+\!\!+} h) \cdot \iota_{[0,d+e]} = f \mathbin{+\!\!+} g \\ ((f \mathbin{+\!\!+} g) \mathbin{+\!\!+} h) \cdot (+d) = g \mathbin{+\!\!+} h \end{cases}$$

The equation $((f \mathbin{+\!\!+} g) \mathbin{+\!\!+} h) \cdot \iota_{[0,d+e]} = f \mathbin{+\!\!+} g$ holds because $(f \mathbin{+\!\!+} g) \mathbin{+\!\!+} h$ arises from the pushout below.

$$\begin{tikzcd}
1 \arrow[r, "0"] \arrow[d, "d+e"'] & {[0,i]} \arrow[d, "(+d)\cdot(+e)"] \arrow[dr, bend left, "h"] & \\
{[0,d+e]} \arrow[r, hook] \arrow[rr, bend right, "f \mathbin{+\!\!+} g"'] & {[0,d+e+i]} \arrow[r, dotted, "(f \mathbin{+\!\!+} g) \mathbin{+\!\!+} h"] & X
\end{tikzcd}$$

In regard to equation $((f \mathbin{+\!\!+} g) \mathbin{+\!\!+} h) \cdot (+d) = g \mathbin{+\!\!+} h$, observe that it is equivalent to the following system of equations.

$$\begin{cases} ((f \mathbin{+\!\!+} g) \mathbin{+\!\!+} h) \cdot (+d) \cdot (+e) = h \\ ((f \mathbin{+\!\!+} g) \mathbin{+\!\!+} h) \cdot (+d) \cdot \iota_{[0,e]} = g \end{cases}$$

Clearly, the first equation holds. For the second one, we calculate,

$$\begin{aligned} ((f \mathbin{+\!\!+} g) \mathbin{+\!\!+} h) \cdot (+d) \cdot \iota_{[0,e]} &= ((f \mathbin{+\!\!+} g) \mathbin{+\!\!+} h) \cdot \iota_{[0,d+e]} \cdot (+d) \\ &= (f \mathbin{+\!\!+} g) \cdot (+d) \\ &= g \end{aligned}$$

The proof that $f \mathbin{+\!\!+} (g \mathbin{+\!\!+} h)$ also mediates the diagram follows by an analogous reasoning. $\square$

Lemma A.2.2. *The equation $\theta \cdot \mathrm{H}\theta = \theta \cdot \mu$ holds.*

Proof. One simply calculates,

$$\begin{aligned} \theta \cdot \mu &= \theta \cdot (+\!\!+) \cdot \langle \mathrm{H}\theta, \lambda_{\mathrm{H}} \rangle \\ &= \theta \cdot \pi_1 \cdot \langle \mathrm{H}\theta, \lambda_{\mathrm{H}} \rangle \\ &= \theta \cdot \mathrm{H}\theta \end{aligned}$$

□

Lemma A.2.3. *The following diagram commutes.*

$$\begin{tikzcd} \mathrm{H}_*\mathrm{H}_* \arrow[r, "\mu"] \arrow[d, "\mathrm{H}_*\theta"'] & \mathrm{H}_* \arrow[d, "\theta"] \\ \mathrm{H}_* \arrow[r, "\theta"'] & \mathrm{Id} \end{tikzcd}$$

Proof. For evolutions with finite duration the proof is essentially that of Lemma A.2.2. For evolutions with infinite duration one just needs to show that the left square in the diagram below commutes, which is clearly true.

$$\begin{tikzcd} \mathrm{I}\mathrm{H}_* \arrow[r, tail] \arrow[d, "\mathrm{I}\theta"'] \arrow[rr, bend left, "i_2 \cdot \mathrm{I}\theta"] & \mathrm{H}_*\mathrm{H}_* \arrow[r, "\mu"] \arrow[d, "\mathrm{H}_*\theta"'] & \mathrm{H}_* \arrow[d, "\theta"] \\ \mathrm{I} \arrow[r, tail] & \mathrm{H}_* \arrow[r, "\theta"'] & \mathrm{Id} \end{tikzcd}$$

□

Lemma A.2.4. *Every map $h \in \mathrm{H}_*X$ induces a functor,*

$$(h +\!\!+ -) : (\mathrm{H}_*X, \sqsubseteq) \to (\mathrm{H}_*X, \sqsubseteq)$$

that preserves colimits.

Proof. Take a diagram functor $\mathscr{D} : \mathsf{I} \to (\mathrm{H}_*X, \sqsubseteq)$. If the family $\pi_2 \cdot \mathscr{D}$ is bounded and has no greatest element then there is no colimit to preserve (see Remark 3.4.11). On the other hand, if $\pi_2 \cdot \mathscr{D}$ is unbounded, *i. e.* if Condition (7) holds, the family $\pi_2 \cdot (h +\!\!+ \mathscr{D})$ is unbounded as well, and thus $h +\!\!+ (\text{colim } \mathscr{D})$ is a cocone for $h +\!\!+ \mathscr{D}$. Since there can only be one such cocone, due to the unboundedness property, it must also be a colimit for $h +\!\!+ \mathscr{D}$. If $\pi_2 \cdot \mathscr{D}$ has a greatest element then $\mathscr{D}$ has a final object, *i. e.* Condition (8) holds, which by definition is a colimit for $\mathscr{D}$. Since $(h +\!\!+ -)$ is a functor, the diagram $h +\!\!+ \mathscr{D}$ has $h +\!\!+ (\text{colim } \mathscr{D})$ as a final object which again by definition is a colimit for $h +\!\!+ \mathscr{D}$. □

Lemma A.2.5. *Consider a hybrid automaton, the corresponding* P*-representation* $(M, c) \in \mathsf{RepHyb}(\mathrm{P})$*, and a* Φ*-bisimulation* $R \subseteq Z \times Z$*. The relation* $R \subseteq Z \times Z$ *is a coalgebraic* Φ*-bisimulation as well.*

Proof. Assume that $(m_1, v_1) R (m_2, v_2)$. We will show that the following cases hold.

- The condition $\big(\pi_2 \cdot [\![c]\!]^\dagger(m_1, v_1)\big) \;\Phi\; \big(\pi_2 \cdot [\![c]\!]^\dagger(m_2, v_2)\big)$ holds.

$$\begin{aligned}
& (m_1, v_1)\, R\, (m_2, v_2) \\
& \quad (\text{ Definition of } \Phi\text{-bisimulation and Theorem 4.4.3 }) \\
\Rightarrow\; & \big(\pi_2 \cdot [\![c]\!]^\dagger(m_1, v_1)\big)\, R\, \big(\pi_2 \cdot [\![c]\!]^\dagger(m_2, v_2)\big) \\
& \quad (\text{ Definition of } \Phi\text{-bisimulation }) \\
\Rightarrow\; & \big(\pi_2 \cdot [\![c]\!]^\dagger(m_1, v_1)\big)\, \Phi\, \big(\pi_2 \cdot [\![c]\!]^\dagger(m_2, v_2)\big)
\end{aligned}$$

- For every $x \in \big(\pi_1 \cdot [\![c]\!]^\dagger(m_1, v_1)\big)$ there exists $y \in \big(\pi_1 \cdot [\![c]\!]^\dagger(m_2, v_2)\big)$ such that $x\, R\, y$.

  Let v_1^l be $\lambda_{\mathbb{R}^n} \cdot \pi_2 \cdot [\![c]\!](m_1, v_1)$ – *i. e.* the last point of the evolution generated by (m_1, v_1) – and similarly for v_2^l. Then,

$$\begin{aligned}
& (m_1, v_1)\, R\, (m_2, v_2), \;\; x \in \big(\pi_1 \cdot [\![c]\!]^\dagger(m_1, v_1)\big) \\
& \quad (\text{ Definition of } \Phi\text{-bisimulation and Theorem 4.4.3 }) \\
\Rightarrow\; & (m_1, v_1^l)\, R\, (m_2, v_2^l) \\
& \quad (\text{ Definition of } \Phi\text{-bisimulation and Theorem 4.4.3 }) \\
\Rightarrow\; & (m_2, v_2^l) \xrightarrow{*} y, \;\; x\, R\, y \\
& \quad (\text{ Theorem 4.4.3 }) \\
\Rightarrow\; & y \in \big(\pi_1 \cdot [\![c]\!]^\dagger(m_2, v_2)\big), \;\; x\, R\, y
\end{aligned}$$

- For every $y \in \big(\pi_1 \cdot [\![c]\!]^\dagger(m_2, v_2)\big)$ there exists $x \in \big(\pi_1 \cdot [\![c]\!]^\dagger(m_1, v_1)\big)$ such that $x\, R\, y$.

  Analogous to the previous case.

□

To assume that $R \subseteq Z \times Z$ is a coalgebraic Φ-bisimulation does *not* entail that it is a Φ-bisimulation. This is explained by our adoption of the black-box perspective which, in contrast to the traditional semantics of hybrid automata, interprets the valuations between jumps as outputs and not as internal states. We can, however, construct a relation $\overline{R} \subseteq Z \times Z$ such that $R \subseteq \overline{R}$ and show that it is a Φ-bisimulation: define $\overline{R}$ as the smallest relation such that $R \subseteq \overline{R}$ and if $x\, R\, y$ then $\big(\pi_2 \cdot [\![c]\!]^\dagger(x)\big)\, \overline{R}\, \big(\pi_2 \cdot [\![c]\!]^\dagger(y)\big)$.

Lemma A.2.6. *Consider a hybrid automaton, the corresponding* P*-representation* $(M, c) \in \mathsf{RepHyb}(\mathrm{P})$*, and a coalgebraic* Φ*-bisimulation* $R \subseteq Z \times Z$*. The relation* $\overline{R} \subseteq Z \times Z$ *is a* Φ*-bisimulation.*

Proof. Assume that $(m_1, v_1)\,\overline{R}\,(m_2, v_2)$. We will show that the following cases hold.

- The condition $(m_1, v_1)\;\Phi\;(m_2, v_2)$ holds.

$$(m_1, v_1)\,\overline{R}\,(m_2, v_2)$$

(Definition of $\overline{R}$)

$$\Rightarrow\; \exists a, b \in Z, r \in \mathbb{R}_{\geq 0}.\; a\,R\,b \;\wedge\; \big(\pi_2 \cdot [\![c]\!]^{\dagger}(a)\big)(r) = (m_1, v_1) \;\wedge\; \big(\pi_2 \cdot [\![c]\!]^{\dagger}(b)\big)(r) = (m_2, v_2)$$

(Definition of coalgebraic Φ-bisimulation)

$$\Rightarrow\; (m_1, v_1)\;\Phi\;(m_2, v_2)$$

- If $(m_1, v_1) \xrightarrow{l} x$ then there exists a state y such that $(m_2, v_2) \xrightarrow{l} y$ and $x\,\overline{R}\,y$.

  In the case of an evolution step,

$$(m_1, v_1)\,\overline{R}\,(m_2, v_2),\;\; (m_1, v_1) \xrightarrow{r} \big(\pi_2 \cdot [\![c]\!]^{\dagger}(m_1, v_1)\big)(r)$$

(Definition of $\overline{R}$)

$$\Rightarrow\; \big(\pi_2 \cdot [\![c]\!]^{\dagger}(m_1, v_1)\big)\,\overline{R}\,\big(\pi_2 \cdot [\![c]\!]^{\dagger}(m_2, v_2)\big),\;\; (m_1, v_1) \xrightarrow{r} \big(\pi_2 \cdot [\![c]\!]^{\dagger}(m_1, v_1)\big)(r)$$

(Theorem 4.4.3)

$$\Rightarrow\; (m_2, v_2) \xrightarrow{r} \big(\pi_2 \cdot [\![c]\!]^{\dagger}(m_2, v_2)\big)(r),\;\; \big(\pi_2 \cdot [\![c]\!]^{\dagger}(m_1, v_1)\big)(r)\,\overline{R}\,\big(\pi_2 \cdot [\![c]\!]^{\dagger}(m_2, v_2)\big)(r)$$

  In the case of a discrete transition,

$$(m_1, v_1)\,\overline{R}\,(m_2, v_2),\;\; (m_1, v_1) \xrightarrow{*} x$$

(Definition of $\overline{R}$)

$$\Rightarrow\; \exists a, b \in Z, r \in \mathbb{R}_{\geq 0}.\; a\,R\,b \;\wedge\; \big(\pi_2 \cdot [\![c]\!]^{\dagger}(a)\big)(r) = (m_1, v_1) \;\wedge\; \big(\pi_2 \cdot [\![c]\!]^{\dagger}(b)\big)(r) = (m_2, v_2)$$

($\big(\pi_1 \cdot [\![c]\!]^{\dagger}(a)\big) = \big(\pi_1 \cdot [\![c]\!]^{\dagger}(m_1, v_1)\big)$ and Theorem 4.4.3)

$$\Rightarrow\; x \in \big(\pi_1 \cdot [\![c]\!]^{\dagger}(a)\big)$$

(Definition of coalgebraic Φ-bisimulation)

$$\Rightarrow\; y \in \big(\pi_1 \cdot [\![c]\!]^{\dagger}(b)\big),\;\; x\,R\,y$$

($\big(\pi_1 \cdot [\![c]\!]^{\dagger}(b)\big) = \big(\pi_1 \cdot [\![c]\!]^{\dagger}(m_2, v_2)\big)$, Theorem 4.4.3, and $R \subseteq \overline{R}$)

$$\Rightarrow\; (m_2, v_2) \xrightarrow{*} y,\;\; x\,\overline{R}\,y$$

- If $(m_2, v_2) \xrightarrow{l} y$ then there exists a state x such that $(m_1, v_1) \xrightarrow{l} x$ and $x\,\overline{R}\,y$. Analogous to the previous case.

□

Lemma A.2.7. *Consider a probabilistic hybrid automaton, the corresponding* PD*-representation* $(M, c) \in \mathsf{RepHyb}(\mathrm{PD})$*, and a* Φ*-bisimulation* $R \subseteq Z \times Z$*. The relation* $R \subseteq Z \times Z$ *is a coalgebraic* Φ*-bisimulation as well.*

Proof. Analogous to Lemma A.2.5. □

Lemma A.2.8. *Consider a probabilistic hybrid automaton, the corresponding* PD*-representation* $(M, c) \in \mathsf{RepHyb}(\mathrm{PD})$*, and a coalgebraic* Φ*-bisimulation* $R \subseteq Z \times Z$*. The relation* $\overline{R} \subseteq Z \times Z$ *is a* Φ*-bisimulation.*

Proof. Assume that $(m_1, v_1)\, \overline{R}\, (m_2, v_2)$. We will show that the following cases hold.

- The condition $(m_1, v_1)\ \Phi\ (m_2, v_2)$ holds.

$$
\begin{aligned}
& (m_1, v_1)\, \overline{R}\, (m_2, v_2) \\
& \quad (\text{ Definition of } \overline{R}\ ) \\
\Rightarrow\ & \exists a, b \in Z, r \in \mathbb{R}_{\geq 0}.\ a\, R\, b\ \wedge \big(\pi_2 \cdot [\![c]\!]^{\dagger}(a)\big)(r) = (m_1, v_1)\ \wedge\ \big(\pi_2 \cdot [\![c]\!]^{\dagger}(b)\big)(r) = (m_2, v_2) \\
& \quad (\text{ Definition of coalgebraic } \Phi\text{-bisimulation }) \\
\Rightarrow\ & (m_1, v_1)\ \Phi\ (m_2, v_2)
\end{aligned}
$$

- If $(m_1, v_1) \xrightarrow{l} \mu_1$ then there exists μ_2 such that $(m_2, v_2) \xrightarrow{l} \mu_2$ and $\mu_1 \asymp_{\overline{R}} \mu_2$.

  Let us first consider an evolution step.

  Denote $\big(\pi_2 \cdot [\![c]\!]^{\dagger}(m_1, v_1)\big)(r)$ by x, $\big(\pi_2 \cdot [\![c]\!]^{\dagger}(m_2, v_2)\big)(r)$ by y, and assume that $(m_1, v_1) \xrightarrow{r} \delta_x$. It is then straightforward to prove that $(m_2, v_2) \xrightarrow{r} \delta_y$, so we will just show that the condition $\delta_x \asymp_{\overline{R}} \delta_y$ holds. Consider the Dirac distribution $\delta_{(x,y)}$ of (x, y). We will show that the three conditions below hold.

  1. $\delta_{(x,y)}(a, b) > 0$ entails $a\, \overline{R}\, b$.

$$
\begin{aligned}
& (m_1, v_1)\, \overline{R}\, (m_2, v_2),\ \ \delta_{(x,y)}(a, b) > 0 \\
& \quad (\text{ Definition of } \overline{R}\ ) \\
\Rightarrow\ & \big(\pi_2 \cdot [\![c]\!]^{\dagger}(m_1, v_1)\big)\, \overline{R}\, \big(\pi_2 \cdot [\![c]\!]^{\dagger}(m_2, v_2)\big),\ \ \delta_{(x,y)}(a, b) > 0 \\
& \quad (\text{ Definition of } \delta_{(x,y)}\ ) \\
\Rightarrow\ & a\, \overline{R}\, b
\end{aligned}
$$

  2. $\delta_x(a) = \delta(\{a\} \times Z)$.

     Follows directly from the definition of $\delta_{(x,y)}$.

3. $\delta_y(b) = \delta(Z \times \{b\})$.

   Analogous to the previous case.

For the discrete case, assume that $(m_1, v_1)\, \overline{R}\, (m_2, v_2)$ and $(m_1, v_1) \xrightarrow{*} \mu_1$.

$$(m_1, v_1)\, \overline{R}\, (m_2, v_2), \quad (m_1, v_1) \xrightarrow{*} \mu_1$$

(Definition of $\overline{R}$)

$$\Rightarrow \exists a, b \in Z, r \in \mathbb{R}_{\geq 0}.\, a\, R\, b \wedge \big(\pi_2 \cdot [\![c]\!]^\dagger(a)\big)(r) = (m_1, v_1) \wedge \big(\pi_2 \cdot [\![c]\!]^\dagger(b)\big)(r) = (m_2, v_2)$$

($\big(\pi_1 \cdot [\![c]\!]^\dagger(a)\big) = \big(\pi_1 \cdot [\![c]\!]^\dagger(m_1, v_1)\big)$ and Theorem 4.4.4)

$$\Rightarrow \mu_1 \in \big(\pi_1 \cdot [\![c]\!]^\dagger(a)\big)$$

(Definition of coalgebraic Φ-bisimulation)

$$\Rightarrow \mu_2 \in \big(\pi_1 \cdot [\![c]\!]^\dagger(b)\big), \quad \mu_1 \asymp_R \mu_2$$

($\big(\pi_1 \cdot [\![c]\!]^\dagger(b)\big) = \big(\pi_1 \cdot [\![c]\!]^\dagger(m_2, v_2)\big)$, Theorem 4.4.4, and $R \subseteq \overline{R}$)

$$\Rightarrow (m_2, v_2) \xrightarrow{*} \mu_2, \quad \mu_1 \asymp_{\overline{R}} \mu_2$$

- If $(m_2, v_2) \xrightarrow{l} \mu_2$ then there exists μ_1 such that $(m_1, v_1) \xrightarrow{l} \mu_1$ and $\mu_1 \asymp_{\overline{R}} \mu_2$. Analogous to the previous case.

□

Lemma A.2.9. *All polynomial functors in* Top *preserve regular monomorphisms.*

Proof. Let us start by recalling a few categorical definitions that concern the notion of regular monomorphism.

A category C is said to be connected if it is non-empty and every two objects $A, B \in$ C can be connected by a finite zig-zag of morphisms as depicted below.

$$A \leftarrow \cdot \rightarrow \cdots \leftarrow \cdot \rightarrow B$$

A diagram $\mathscr{D} : \mathsf{I} \to \mathsf{C}$ is called a *connected diagram* if I is connected, and a limit of $\mathscr{D}$ is called a *connected limit* if $\mathscr{D} : \mathsf{I} \to \mathsf{C}$ is connected. Recall that a regular monomorphism is an equaliser of a pair of morphisms and that equalisers are connected limits.

Then, note that the identity functor Id : Top → Top preserves all limits and that the constant functor A : Top → Top trivially preserves the connected ones. The functor $(\times) : \mathsf{C} \times \mathsf{C} \to \mathsf{C}$ is right adjoint, so if two functors $F, G : \mathsf{C} \to \mathsf{C}$ preserve limits of a certain type their product preserves limits of this type as well. Finally, it is well-known that the functor $(+) : \mathsf{Set} \times \mathsf{Set} \to \mathsf{Set}$ preserves connected limits, so observe that $(+) : \mathsf{Top} \times \mathsf{Top} \to \mathsf{Top}$ preserves initial cones and apply Theorem 5.1.17. This entails that if two functors $F, G : \mathsf{C} \to \mathsf{C}$ preserve connected limits their sum preserves limits of this type as well. □

Lemma A.2.10. *Consider a continuous map $f : X \times \mathbb{R}_{\geq 0} \to Y$. If the conditions below are satisfied, it can be extended into a continuous map of the type $X \times [0, \infty] \to Y$.*

1. *For every element $x \in X$, the limit $\lim_{r\to\infty} f(x, r)$ is well-defined.*
2. *For every element $x \in X$ and neighbourhood V of $\lim_{r\to\infty} f(x, r)$, there exists a neighbourhood U of x and a number $k \in \mathbb{R}_{\geq 0}$ such that,*

$$f[U \times (k, \infty)] \subseteq V$$

Proof. Consider a continuous function $f : X \times \mathbb{R}_{\geq 0} \to Y$. We can extend it into a function $\overline{f} : X \times [0, \infty] \to Y$ by applying condition (1); more concretely, by defining,

$$\overline{f}(x, \infty) = \lim_{r\to\infty} f(x, r)$$

at every point $(x, \infty) \in X \times [0, \infty]$. It remains to show that the extended map is continuous at the point (x, ∞). So take a neighbourhood V of $\overline{f}(x, \infty)$. Condition (2) provides an open $U \times (k, \infty)$ such that,

$$\overline{f}[U \times (k, \infty)] = f[U \times (k, \infty)] \subseteq V$$

Since $\overline{f}(x, \infty) \in V$, the condition $\overline{f}[U \times (k, \infty]] \subseteq V$ holds, and clearly the property $(x, \infty) \in U \times (k, \infty]$ holds as well. The proof then follows from the definition of continuity in terms of neighbourhoods. $\square$

Lemma A.2.11. *Let* $\mathrm{U} : \mathsf{Top} \to \mathsf{Set}$ *be the forgetful functor that sends topological spaces to their carrier. The natural transformations* $\mathrm{UH_c} \times \mathrm{UH_c} \to \mathrm{UH_c}$ *are in bijective correspondence with the maps $m : [0, \infty] \times [0, \infty] \to \mathrm{UH_c}(\mathbb{R}_{\geq 0} + \mathbb{R}_{\geq 0})$ such that for every $i, j \in [0, \infty]$, $m(i, j)$ is in the image of* $\mathrm{H_c}(\iota_i + \iota_j)$.

Proof. Let $\downarrow(-)$ be the operation that sends an element $a \in [0, \infty]$ to the down-closure $\{r \in \mathbb{R}_{\geq 0} \mid r \leq a\}$, and observe that for every $i, j \in \mathbb{R}_{\geq 0}$ there exists an injection,

$$\mathrm{H_c}(\iota_i + \iota_j) : \mathrm{H_c}(\downarrow i + \downarrow j) \rightarrowtail \mathrm{H_c}(\mathbb{R}_{\geq 0} + \mathbb{R}_{\geq 0})$$

There also exists an isomorphism,

$$\mathrm{UH_c} \simeq \coprod_{i\in[0,\infty]} \hom(\downarrow i, -)$$

Moreover, in Top binary products distribute over arbitrary coproducts. It follows that,

$$\begin{aligned}\mathrm{UH_c} \times \mathrm{UH_c} &\simeq \Big(\coprod_{i\in[0,\infty]} \hom(\downarrow i, -)\Big) \times \Big(\coprod_{j\in[0\infty]} \hom(\downarrow j, -)\Big)\\ &\simeq \coprod_{i,j\in[0,\infty]} \hom(\downarrow i, -) \times \hom(\downarrow j, -)\\ &\simeq \coprod_{i,j\in[0,\infty]} \hom(\downarrow i + \downarrow j, -)\end{aligned}$$

where the last step follows from contravariant hom functors sending colimits to limits. Then we calculate,

$$\begin{aligned}[\mathsf{Top},\mathsf{Set}]\,(\mathrm{UH_c}\times\mathrm{UH_c},\mathrm{UH_c}) &\simeq [\mathsf{Top},\mathsf{Set}]\,\Big(\coprod_{i,j\in[0,\infty]}\hom(\downarrow i+\downarrow j,\,-\,),\mathrm{UH_c}\Big)\\ &\simeq \prod_{i,j\in[0,\infty]}[\mathsf{Top},\mathsf{Set}]\,(\hom(\downarrow i+\downarrow j,\,-\,),\mathrm{UH_c})\\ &\simeq \prod_{i,j\in[0,\infty]}\mathrm{UH_c}(\downarrow i+\downarrow j)\end{aligned}$$

where the last step is an application of the Yoneda lemma. Given an element $s\in\prod_{i,j\in[0,\infty]}\mathrm{UH_c}(\downarrow i+\downarrow j)$ with components $s_{ij}=(a_{ij},b_{ij})\in\mathrm{UH_c}(\downarrow i+\downarrow j)$, the respective natural transformation α^s is defined at each X by,

$$\alpha^s_X((f,i),(g,j))=([f_i,g_j]\cdot a_{ij},b_{ij})$$

where f_i and g_j are the restrictions of f and g to the domains $\downarrow i$ and $\downarrow j$, respectively. The injection $\mathrm{H_c}(\iota_i+\iota_j):\mathrm{H_c}(\downarrow i+\downarrow j)\rightarrowtail\mathrm{H_c}(\mathbb{R}_{\geq 0}+\mathbb{R}_{\geq 0})$ gives rise to the composition,

$$\begin{tikzcd}
{[\mathsf{Top},\mathsf{Set}](\mathrm{UH_c}\times\mathrm{UH_c},\mathrm{UH_c})} \arrow[d, "\simeq"'] & \hom([0,\infty]\times[0,\infty],\mathrm{UH_c}(\mathbb{R}_{\geq 0}+\mathbb{R}_{\geq 0})) \\
\prod_{i,j\in[0,\infty]}\mathrm{UH_c}(\downarrow i+\downarrow j) \arrow[r, tail] & \prod_{i,j\in[0,\infty]}\mathrm{UH_c}(\mathbb{R}_{\geq 0}+\mathbb{R}_{\geq 0}) \arrow[u, "\simeq"']
\end{tikzcd}$$

whose image (isomorphic to the domain) is the set of functions $m:[0,\infty]\times[0,\infty]\to\mathrm{UH_c}(\mathbb{R}_{\geq 0}+\mathbb{R}_{\geq 0})$ such that for every $i,j\in[0,\infty]$, $m(i,j)$ is in the image of $\mathrm{H_c}(\iota_i+\iota_j)$ *i. e.* for every elements $i,j\in[0,\infty]$, $m(i,j)$ is a pair (f,k) with $f[\mathbb{R}_{\geq 0}]\subseteq[0,i]+[0,j]$. □

Lemma A.2.12. *Let $\mathscr{D}:\mathbb{R}_{\geq 0}\to\mathsf{Top}$ be the diagram functor that sends a non-negative real number i to the subspace $[0,i)$ and the arrow $i\leq j$ to the respective inclusion map. The contravariant exponential functor $X^{(\,-\,)}:\mathsf{Top}^{\mathsf{op}}\to\mathsf{Top}$ sends the colimit of $\mathscr{D}$ to the limit of $X^{\mathscr{D}^{\mathsf{op}}}:(\mathbb{R}_{\geq 0})^{\mathsf{op}}\to\mathsf{Top}$.*

Proof. The hom functor $\hom:(\,-\,,X):\mathsf{Top}^{\mathsf{op}}\to\mathsf{Set}$ sends colimits to limits. So according to Theorem 5.1.17, we just need to show that the cone,

$$\Big(f_i:X^{\mathbb{R}_{\geq 0}}\to X^{[0,i)}\Big)_{i\in\mathbb{R}_{\geq 0}}$$

formed by the maps f_i that restrict a function's domain to $[0,i)$, respects the following condition: for every subbasic set $[K,U]$ of $X^{\mathbb{R}_{\geq 0}}$ there exists a non-negative real number i and an open subset V of $X^{[0,i)}$ such that $[K,U]=f_i^{-1}(V)$.

The subset $K\subseteq\mathbb{R}_{\geq 0}$ is compact, therefore there exists a non-negative real number i such that $K\subseteq[0,i)$. It is then routine to show that $[K,U]=f_i^{-1}([K,U])$. □

BIBLIOGRAPHY

[AH17] Sokolova A. and Woracek H. ‘Termination in Convex Sets of Distributions’. In: *CALCO: 7th Conference on Algebra and Coalgebra in Computer Science, June 12-16, 2017 - Ljubljana, Slovenia.* Ed. by Filippo Bonchi and Barbara König. Vol. 72. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, 22:1–22:16. ISBN: 978-3-95977-033-0.

[AAG03] Michael G. Abbott, Thorsten Altenkirch and Neil Ghani. ‘Categories of Containers’. In: *FoSSaCS: Foundations of Software Science and Computational Structures, 6th International Conference, Warsaw, Poland, April 7-11, 2003.* Ed. by Andrew D. Gordon. Vol. 2620. Lecture Notes in Computer Science. Springer, 2003, pp. 23–38.

[Adá05] Jiří Adámek. ‘Introduction to coalgebra’. In: *Theory and Applications of Categories* 14.8 (2005), pp. 157–199.

[AHS09] Jiří Adámek, Horst Herrlich and George E. Strecker. *Abstract and Concrete Categories - The Joy of Cats.* Dover Publications, 2009. ISBN: 978-0486469348.

[AR94] Jiří Adámek and Jiří Rosický. *Locally presentable and accessible categories.* Vol. 189. London Mathematical Society Lecture Note Series. Cambridge University Press, 1994. ISBN: 0-521-42261-2.

[Alu15] Rajeev Alur. *Principles of Cyber-Physical Systems.* MIT Press, 2015. ISBN: 978-0262029117.

[AD94] Rajeev Alur and David L. Dill. ‘A theory of timed automata’. In: *Theoretical Computer Science* 126.2 (1994), pp. 183 –235. ISSN: 0304-3975.

[AH97] Rajeev Alur and Thomas A. Henzinger. ‘Modularity for Timed and Hybrid Systems’. In: *CONCUR ’97: Concurrency Theory, 8th International Conference, Warsaw, Poland, July 1-4, 1997.* Ed. by Antoni W. Mazurkiewicz and Józef Winkowski. Vol. 1243. Lecture Notes in Computer Science. Springer, 1997, pp. 74–88.

[ATP04] Rajeev Alur, Salvatore La Torre and George J. Pappas. ‘Optimal paths in weighted timed automata’. In: *Theoretical Computer Science* 318.3 (2004), pp. 297 –322. ISSN: 0304-3975.

[Alu+93] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger and Pei-Hsin Ho. ‘Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems’. In: *Workshop on the Theory of Hybrid Systems, Denmark. 1992.* Ed. by Robert L. Grossman, Anil Nerode, Anders P. Ravn and Hans Rischel. Vol. 736. Lecture Notes in Computer Science. Springer, 1993, pp. 209–229.

[Alu+95] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis and Sergio Yovine. 'The Algorithmic Analysis of Hybrid Systems'. In: *Theoretical Computer Science* 138.1 (1995), pp. 3–34.

[AAS05] Aaron Ames, Alessandro Abate and Shankar Sastry. 'Sufficient conditions for the existence of Zeno behavior'. In: *CDC-ECC'05: Decision and Control and European Control Conference, 44th IEEE Conference, Seville, Spain, December, 2005*. IEEE, 2005, pp. 696–701.

[BK11] Adriana Balan and Alexander Kurz. 'Finitary functors: from Set to Preord and Poset'. In: *CALCO: 4th Conference on Algebra and Coalgebra in Computer Science, August 30 - September 2, Winchester, UK*. Ed. by Andrea Corradini, Bartek Klin and Cîrstea Corina. Vol. 6859. Springer, 2011, pp. 85–99.

[BKV12] Adriana Balan, Alexander Kurz and Jiri Velebil. 'On coalgebraic logic over posets'. In: *Short Contributions of CMCS'12: Coalgebraic Methods in Computer Science, 11th International Workshop, Tallinn, Estonia, 31st March - 1st April* (2012).

[Bal+14] Paolo Baldan, Filippo Bonchi, Henning Kerstan and Barbara König. 'Behavioral Metrics via Functor Lifting'. In: *FSTTCS'14: Foundations of Software Technology and Theoretical Computer Science, 34th International Conference, December 15-17, 2014, New Delhi, India*. Ed. by Venkatesh Raman and S. P. Suresh. Vol. 29. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014, pp. 403–415.

[Bal13] Pedro Baltazar. 'Probabilization of Logics: Completeness and Decidability'. English. In: *Logica Universalis* 7.4 (2013), pp. 403–440. ISSN: 1661-8297.

[Ban+15] Richard Banach, Michael J. Butler, Shengchao Qin, Nitika Verma and Huibiao Zhu. 'Core Hybrid Event-B I: Single Hybrid Event-B machines'. In: *Science of Computer Programming* 105 (2015), pp. 92–123.

[Bar03] Luís S. Barbosa. 'Towards a calculus of state-based software components'. In: *Journal of Universal Computer Science* 9 (2003), pp. 891–909.

[BO03] Luís S. Barbosa and José N. Oliveira. 'State-based components made generic'. In: *Electronic Notes in Theoretical Computer Science* 82.1 (2003), pp. 39–56.

[Bar93] Michael Barr. 'Terminal coalgebras in well-founded set theory'. In: *Theoretical Computer Science* 114.2 (1993), pp. 299–315. ISSN: 0304-3975.

[Bee+06] D. A. van Beek, Ka L. Man, Michel A. Reniers, J. E. Rooda and Ramon R. H. Schiffelers. 'Syntax and consistent equation semantics of hybrid Chi'. In: *Journal of Logical and Algebraic Methods in Programming* 68.1-2 (2006), pp. 129–210.

[BBH12] Guram Bezhanishvili, Nick Bezhanishvili and John Harding. 'Modal compact Hausdorff spaces'. In: *Journal of Logic and Computation* 25.1 (2012), pp. 1–35.

[BFV10] Nick Bezhanishvili, Gaëlle Fontaine and Yde Venema. 'Vietoris Bisimulations'. In: *Journal of Logic and Computation* 20.5 (2010), pp. 1017–1040.

[BBW06] Patrick Blackburn, Johan F. A. K. van Benthem and Frank Wolter. *Handbook of Modal Logic, Volume 3 (Studies in Logic and Practical Reasoning)*. New York, NY, USA: Elsevier Science Inc., 2006. ISBN: 978-0444516909.

[Blo13] Sylvan Charles Bloch. *Introduction to Classical and Quantum Harmonic Oscillators*. John Wiley & Sons, 2013. ISBN: 978-0471147442.

[BKR07] Marcello M. Bonsangue, Alexander Kurz and Ingrid M. Rewitzky. 'Coalgebraic representations of distributive lattices with operators'. In: *Topology and its Applications* 154.4 (2007), pp. 778–791. ISSN: 0166-8641.

[BRS09] Marcello M. Bonsangue, Jan J. M. M. Rutten and Alexandra Silva. 'An Algebra for Kripke Polynomial Coalgebras'. In: *LICS'09:Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, 11-14 August 2009, Los Angeles, CA, USA*. IEEE Computer Society, 2009, pp. 49–58.

[Bor94a] Francis Borceux. *Handbook of Categorical Algebra*. Vol. 1. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1994. ISBN: 978-0521061193.

[Bor94b] Francis Borceux. *Handbook of categorical algebra*. Vol. 2. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1994. ISBN: 978-0521441797.

[Bou66] Nicolas Bourbaki. *General topology, part I*. chapters 1–4. Hermann, Paris and Addison-Wesley, 1966. ISBN: 978-3-540-64241-1.

[Bou06] Patricia Bouyer. 'Weighted Timed Automata: Model-Checking and Games'. In: *Electronic Notes in Theoretical Computer Science* 158 (2006), pp. 3–17.

[Bra11] Torben Braüner. *Proof-Theory of Propositional Hybrid Logic*. Hybrid Logic and its Proof-Theory, 2011.

[BS02] Michael Brin and Garrett Stuck. *Introduction to dynamical systems*. Cambridge University Press, 2002. ISBN: 978-0511755316.

[Bro+12] David Broman, Edward A. Lee, Stavros Tripakis and Martin Törngren. '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*. Ed. by Cécile Hardebolle, Eugene Syriani, Jonathan Sprinkle and Tamás Mészáros. ACM, 2012, pp. 49–54.

[CHR91] Zhou Chaochen, C. A. R. Hoare and Anders P. Ravn. 'A Calculus of Durations'. In: *Information Processing Letters* 40.5 (1991), pp. 269–276.

[CJR96] Zhou Chaochen, Wang Ji and Anders P. Ravn. 'A formal description of hybrid systems'. English. In: *Hybrid Systems III*. Ed. by Rajeev Alur, Thomas A. Henzinger and Eduardo D. Sontag. Vol. 1066. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1996, pp. 511–530. ISBN: 978-3-540-61155-4.

[CRH93] Zhou Chaochen, Anders P. Ravn and Michael R. Hansen. ‘An extended duration calculus for hybrid real-time systems’. In: *Hybrid Systems*. Ed. by Robert L. Grossman, Anil Nerode, Anders P. Ravn and Hans Rischel. Vol. 736. Lecture Notes in Computer Science. Springer, 1993, pp. 36–59.

[Che13] Liang-Ting Chen. ‘On a purely categorical framework for coalgebraic modal logic’. PhD thesis. School of Computer Science: University of Birminghan, 2013.

[CLP91] R. Cignoli, S. Lafalce and A. Petrovich. ‘Remarks on Priestley duality for distributive lattices’. In: *Order* 8.3 (1991), pp. 299–315. ISSN: 0167-8094.

[CT97] Maria Manuel Clementino and Walter Tholen. ‘A characterization of the Vietoris topology’. In: *Proceedings of the 12th Summer Conference on General Topology and its Applications (North Bay, ON, 1997)*. Vol. 22. 1997, pp. 71–95.

[DDG16] Fredrik Dahlqvist, Vincent Danos and Ilias Garnier. ‘Giry and the Machine’. In: *Electronic Notes in Theoretical Computer Science* 325 (2016), pp. 85–110.

[DPS17] Fredrik Dahlqvist, Louis Parlant and Alexandra Silva. ‘Layer by layer-Combining Monads’. In: *CoRR* abs/1712.01113 (2017). URL: https://arxiv.org/abs/1712.01113.

[Dav97] Jennifer M. Davoren. ‘On Hybrid Systems and the Modal μ-calculus’. In: *Hybrid Systems V*. Vol. 1567. Lecture Notes in Computer Science. Springer, 1997, pp. 38–69.

[DN00] Jennifer M. Davoren and Anil Nerode. ‘Logics for Hybrid Systems’. In: *Proceedings of the IEEE*. Vol. 88. 2000, pp. 985–1010.

[Dob09] Ernst-Erich Doberkat. *Stochastic Coalgebraic Logic*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2009, pp. I–XV, 1–231. ISBN: 978-3-642-02994-3.

[Dud72] Roman Duda. ‘One result on inverse limits and hyperspaces’. In: *General Topology and its Relations to Modern Analysis and Algebra* (1972), pp. 99–102.

[Eng89] Ryszard Engelking. *General topology*. Vol. 6. Sigma Series in Pure Mathematics. Translated from the Polish by the author. Berlin: Heldermann Verlag, 1989, pp. viii + 529. ISBN: 3-88538-006-4.

[FG92] Marcelo Finger and Dov Gabbay. ‘Adding a temporal dimension to a logic system’. English. In: *Journal of Logic, Language and Information* 1.3 (1992), pp. 203–233. ISSN: 0925-8531.

[Fre12] Tim Freegarde. *Introduction to the Physics of Waves*. Cambridge University Press, 2012. ISBN: 978-0521197571.

[Fri14] Peter Fritzson. *Principles of object-oriented modeling and simulation with Modelica 3.3: a cyber-physical approach*. John Wiley & Sons, 2014. ISBN: 978-1118859124.

[FH13] William Fulton and Joe Harris. *Representation theory: a first course.* Vol. 129. Springer Science & Business Media, 2013.

[GK13] Nicola Gambino and Joachim Kock. ‘Polynomial functors and polynomial monads’. In: *Mathematical proceedings of the cambridge philosophical society.* Vol. 154. 1. Cambridge University Press. 2013, pp. 153–192.

[Gie+03] Gerhard Gierz, Karl Heinrich Hofmann, Klaus Keimel, Jimmie D. Lawson, Michael W. Mislove and Dana S. Scott. *Continuous lattices and domains.* Vol. 93. Encyclopedia of Mathematics and its Applications. Cambridge: Cambridge University Press, 2003, pp. xxxvi+591. ISBN: 978-0521803380.

[GP11] Antoine Girard and George J. Pappas. ‘Approximate Bisimulation: A Bridge Between Computer Science and Control Theory’. In: *European Journal of Control* 17.5–6 (2011), pp. 568 –578. ISSN: 0947-3580.

[GST09] Rafal Goebel, Ricardo G Sanfelice and Andrew R Teel. ‘Hybrid dynamical systems’. In: *IEEE Control Systems* 29.2 (2009), pp. 28–93.

[Goe+04] Rafal Goebel, Joao Hespanha, Andrew R Teel, Chaohong Cai and Ricardo Sanfelice. ‘Hybrid systems: generalized solutions and robust stability’. In: *NOLCOS04’: Nonlinear Control Systems 2004, 6th Symposium, Stuttgart, Germany, 1-3 September. Elsevier* 37.13 (2004), pp. 1–12.

[GM87] Joseph A. Goguen and José Meseguer. ‘Models and equality for logical programming’. In: *TAPSOFT ’87.* Ed. by Hartmut Ehrig, Robert Kowalski, Giorgio Levi and Ugo Montanari. Vol. 250. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1987, pp. 1–22. ISBN: 978-3-540-17611-4.

[GS13] Sergey Goncharov and Lutz Schröder. ‘A Relatively Complete Generic Hoare Logic for Order-Enriched Effects’. In: *LICS’13: 28th Annual ACM/IEEE Symposium on Logic in Computer Science, New Orleans, USA, June 25-28, 2013.* IEEE, 2013, pp. 273–282.

[Gou13] Jean Goubault-Larrecq. *Non-Hausdorff Topology and Domain Theory—Selected Topics in Point-Set Topology.* Vol. 22. New Mathematical Monographs. Cambridge University Press, Mar. 2013. ISBN: 978-1107034136.

[GS01] Peter H. Gumm and Tobias Schröder. ‘Products of coalgebras’. In: *algebra universalis* 46.1 (2001), pp. 163–185.

[GS02] Peter H. Gumm and Tobias Schröder. ‘Coalgebras Of Bounded Type’. In: *Mathematical Structures in Computer Science* 12.5 (2002), pp. 565–578.

[Gun+14] Volkan Gunes, Steffen Peter, Tony Givargis and Frank Vahid. ‘A survey on concepts, applications, and challenges in cyber-physical systems.’ In: *Transactions on Internet and Information Systems* 8.12 (2014), pp. 4242–4268.

[HK15] Helle H. Hansen and Clemens Kupke. ‘Weak Completeness of Coalgebraic Dynamic Logics’. In: *FICS’15: Fixed Points in Computer Science, 10th International Workshop, Berlin, Germany, September 12-12, 2015*. Vol. 191. Electronic Notes in Theoretical Computer Science. 2015, pp. 90–104.

[HKL14] Helle H. Hansen, Clemens Kupke and Raul A. Leal. ‘Strong Completeness for Iteration-Free Coalgebraic Dynamic Logics’. In: *IFIP TC’14: Theoretical Computer Science, 8th International Conference, Rome, Italy, September 1-3, 2014*. Vol. 8705. LNCS. Springer, 2014, pp. 281–295.

[HKR17] Helle H. Hansen, Clemens Kupke and Jan Rutten. ‘Stream Differential Equations: Specification Formats and Solution Methods’. In: *Logical Methods in Computer Science* 13.1 (2017).

[HC97] Michael R. Hansen and Zhou Chaochen. ‘Duration calculus: Logical foundations’. In: *Formal Aspects of Computing* 9.3 (1997), pp. 283–330. ISSN: 0934-5043.

[HKT00] David Harel, Dexter Kozen and Jerzy Tiuryn. *Dynamic logic*. MIT press, 2000. ISBN: 978-0262527668.

[HJS07] Ichiro Hasuo, Bart Jacobs and Ana Sokolova. ‘Generic Trace Semantics via Coinduction’. In: *Logical Methods in Computer Science* 3.4 (2007).

[Hau14] Felix Hausdorff. *Grundzüge der Mengenlehre*. German. Leipzig: Veit & Comp. VIII and 476 pages, 1914.

[Hen96] Thomas A. Henzinger. ‘The Theory of Hybrid Automata’. In: *LICS96’: Logic in Computer Science, 11th Annual Symposium, New Jersey, USA, July 27-30, 1996*. IEEE, 1996, pp. 278–292.

[Hof99] Dirk Hofmann. ‘Natürliche Dualitäten und das verallgemeinert Stone-Weierstraß Theorem’. PhD thesis. University of Bremen, 1999.

[HNN18] Dirk Hofmann, Renato Neves and Pedro Nora. ‘Limits in Categories of Vietoris Coalgebras’. In: *Mathematical Structures in Computer Science (In press)* (2018). URL: http://arxiv.org/abs/1612.03318.

[HN16] Dirk Hofmann and Pedro Nora. *Enriched Stone-type dualities*. Tech. rep. 2016. URL: http://arxiv.org/abs/1605.00081.

[HST14] Dirk Hofmann, Gavin J. Seal and Walter Tholen, eds. *Monoidal Topology. A Categorical Approach to Order, Metric, and Topology*. Cambridge University Press, July 2014, p. 518. ISBN: 978-1107063945.

[Höf09] Peter Höfner. ‘Algebraic calculi for hybrid systems’. PhD thesis. University of Augsburg, 2009.

[HM09] Peter Höfner and Bernhard Möller. ‘An algebra of hybrid systems’. In: *The Journal of Logic and Algebraic Programming* 78.2 (2009), pp. 74 –97. ISSN: 1567-8326.

[HM11] Peter Höfner and Bernhard Möller. ‘Fixing Zeno gaps’. In: *Theoretical Computer Science* 412.28 (2011). Festschrift in Honour of Jan Bergstra, pp. 3303 –3322.

[Hol90] Philip Holmes. ‘Poincaré, celestial mechanics, dynamical-systems theory and "chaos"’. In: *Physics Reports* 193.3 (1990), pp. 137 –163. ISSN: 0370-1573.

[Hug01] Jess Hughes. ‘A study of categories of algebras and coalgebras’. PhD thesis. Carnegie Mellon University, 2001.

[HPP06] Martin Hyland, Gordon D. Plotkin and John Power. ‘Combining effects: Sum and tensor’. In: *Theoretical Computer Science* 357.1-3 (2006), pp. 70–99.

[Jac00] Bart Jacobs. ‘Object-oriented hybrid systems of coalgebras plus monoid actions’. In: *Theoretical Computer Science* 239.1 (2000), pp. 41 –95. ISSN: 0304-3975.

[Jac10] Bart Jacobs. ‘Convexity, duality and effects’. In: *Theoretical Computer Science. IFIP Advances in Information and Communication Technology.* Ed. by C. S. Calude and V. Sassone. Vol. 323. Springer. 2010, pp. 1–19.

[Jac16] Bart Jacobs. *Introduction to Coalgebra: Towards Mathematics of States and Observation.* Vol. 59. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2016. ISBN: 9781316823187.

[Jak18] Julian Jakob. ‘Exploring Zeno-like Iteration for Hybrid Components’. MA thesis. Friedrich-Alexander Universität Erlangen-Nürnberg, 2018.

[Joh86] Peter T. Johnstone. *Stone spaces.* Vol. 3. Cambridge university press, 1986. ISBN: 978-0521337793.

[Jun04] Achim Jung. ‘Stably compact spaces and the probabilistic powerspace construction’. In: *Domain-theoretic Methods in Probabilistic Processes.* Ed. by J. Desharnais and P. Panangaden. Vol. 87. Electronic Notes in Theoretical Computer Science. 15pp. Elsevier, Nov. 2004, pp. 5–20.

[KH96] Anatole Katok and Boris Hasselblatt. *Introduction to the modern theory of dynamical systems.* Encyclopedia of mathematics and its applications. Cambridge: Cambridge university press, 1996. ISBN: 978-0521575577.

[Kec12] Alexander Kechris. *Classical descriptive set theory.* Vol. 156. Springer Science & Business Media, 2012.

[Kel55] John Kelley. *General Topology.* Reprinted by Springer-Verlag, Graduate Texts in Mathematics, 27, 1975. Van Nostrand, 1955. ISBN: 978-0387901251.

[Kha08] Uzma Khadim. ‘Process Algebras for Hybrid Systems: Comparison and Development’. PhD thesis. Eindhoven University of Technology, 2008.

[KK12] K. D. Kim and P. R. Kumar. ‘Cyber-Physical Systems: A Perspective at the Centennial’. In: *Proceedings of the IEEE* 100.Special Centennial Issue (2012), pp. 1287–1308. ISSN: 0018-9219.

[Kle07] Harold Klee. *Simulation of dynamic systems with MATLAB and Simulink*. CRC Press, 2007.

[Kou+13] Yanni Kouskoulas, David W. Renshaw, André Platzer and Peter Kazanzides. 'Certifying the Safe Design of a Virtual Fixture Control Algorithm for a Surgical Robot'. In: *HSCC'13: Hybrid Systems: Computation and Control, Philadelphia, USA, April 8-13, 2013*. Ed. by Calin Belta and Franjo Ivancic. ACM, 2013, pp. 263–272.

[Koz97] Dexter Kozen. 'Kleene Algebra with Tests'. In: *ACM Transactions on Programming Languages and Systems* 19.3 (1997), pp. 427–443.

[KKV04] Clemens Kupke, Alexander Kurz and Yde Venema. 'Stone Coalgebras'. In: *Theoretical Computer Science* 327.1-2 (Oct. 2004), pp. 109–134. ISSN: 0304-3975.

[Kur01] Alexander Kurz. 'Logics for coalgebras and applications to computer science'. PhD thesis. Ludwig-Maximilians-Universität München, 2001.

[Law11] Jimmie Lawson. 'Stably compact spaces'. In: *Mathematical Structures in Computer Science* 21.1 (2011), pp. 125–169. ISSN: 0960-1295.

[Lee06] Edward A. Lee. 'Cyber-Physical Systems - Are Computing Foundations Adequate?' In: *Position Paper for NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap*. 2006.

[Lee08] Edward A Lee. 'Cyber physical systems: Design challenges'. In: *ISORC '08: Object Oriented Real-Time Distributed Computing, 11th IEEE Symposium, Orlando, USA*. IEEE. 2008, pp. 363–369.

[LS16] Edward A. Lee and Sanjit A. Seshia. *Introduction to embedded systems: A cyber-physical systems approach*. MIT Press, 2016. ISBN: 978-0262533812.

[Lin66] Fred E. J. Linton. 'Some aspects of equational categories'. In: *Proceedings of the Conference on Categorical Algebra*. Springer. 1966, pp. 84–94.

[Lin69] Fred E. J. Linton. 'Coequalizers in categories of algebras'. In: *Seminar on Triples and Categorical Homology Theory*. Republished in: Reprints in Theory and Applications of Categories, No. 18 (2008) pp. 61-72. Springer, 1969, pp. 75–90.

[Liu+99] Jie Liu, Xiaojun Liu, Tak-Kuen J. Koo, B. Sinopoli, S. Sastry and E. A. Lee. 'A hierarchical hybrid system model and its simulation'. In: *Decision and Control, 38th IEEE Conference, Phoenix, USA, December 7–10, 1999*. Vol. 4. IEEE, 1999, pp. 3508–3513.

[LG02] Christoph Lüth and Neil Ghani. 'Composing monads using coproducts'. In: *ICFP'02: Functional Programming, 7th ACM SIGPLAN International Conference, Pittsburgh, USA, October 04 - 06, 2002*. Ed. by M. Wand and S. L. Peyton Jones. ACM, 2002, pp. 133–144.

[Lya92] Aleksandr Mikhailovich Lyapunov. 'The general problem of the stability of motion'. In: *International Journal of Control* 55.3 (1992), pp. 531–534.

[ML98] Saunders Mac Lane. *Categories for the working mathematician.* Vol. 5. springer, 1998. ISBN: 978-1475747218.

[Mad+18] Alexandre Madeira, Renato Neves, Manuel A. Martins and Luís S. Barbosa. 'Hierarchical hybrid logic'. In: *LSFA'17: Logical and Semantic Frameworks with Applications, 12th Workshop.* Vol. In press. Electronic Notes in Theoretical Computer Science, 2018.

[Mal10] Oded Maler. 'Amir Pnueli and the dawn of hybrid systems'. In: *HSCC'10: Hybrid Systems: Computation and Control, 13th ACM International Conference, Stockholm, Sweden, April 12-15, 2010.* Ed. by Karl Henrik Johansson and Wang Yi. ACM, 2010, pp. 293–295.

[MMP91] Oded Maler, Zohar Manna and Amir Pnueli. 'From Timed to Hybrid Systems'. In: *Real-Time: Theory in Practice, The Netherlands, June 3-7, 1991.* Ed. by J. W. de Bakker, Cornelis Huizing, Willem P. de Roever and Grzegorz Rozenberg. Vol. 600. Lecture Notes in Computer Science. Springer, 1991, pp. 447–484.

[Man02] Ernest G. Manes. 'Taut monads and $T0$-spaces'. In: *Theoretical Computer Science* 275.1-2 (Mar. 2002), pp. 79–109. ISSN: 0304-3975.

[MM07] Ernie Manes and Philip Mulry. 'Monad compositions I: general constructions and recursive distributive laws'. In: *Theory and Applications of Categories* 18.7 (2007), pp. 172–208.

[Mar+11] Manuel A. Martins, Alexandre Madeira, Răzvan Diaconescu and Luís Soares Barbosa. 'Hybridization of Institutions'. In: *CALCO: Algebra and Coalgebra in Computer Science, Winchester, UK, August 30 - September 2, 2011).* Ed. by A. Corradini, B. Klin and C. Cîrstea. Vol. 6859. Lecture Notes in Computer Science. Springer, 2011, pp. 283–297.

[MB06] Sun Meng and Luís S. Barbosa. 'Components as coalgebras: The refinement dimension'. In: *Theoretical Computer Science* 351.2 (2006), pp. 276 –294. ISSN: 0304-3975.

[Mic51] Ernest Michael. 'Topologies on spaces of subsets'. In: *Transactions of the American Mathematical Society* 71.1 (1951), pp. 152–182.

[Möb83] Axel Möbus. 'Alexandrov compactification of relational algebras'. In: *Archiv der Mathematik* 40.6 (1983), pp. 526–537. ISSN: 0003-889X.

[Mog89] Eugenio Moggi. 'Computational Lambda-Calculus and Monads'. In: *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), Pacific Grove, California, USA, June 5-8, 1989.* IEEE Computer Society, 1989, pp. 14–23.

[Mog91] Eugenio Moggi. 'Notions of computation and monads'. In: *Information and computation* 93.1 (1991), pp. 55–92.

[Nac65] Leopoldo Nachbin. *Topology and order.* Translated from the Portuguese by Lulu Bechtolsheim. Van Nostrand Mathematical Studies, No. 4. D. Van Nostrand Co., 1965, pp. vi + 122.

[NB17] Renato Neves and Luís S. Barbosa. 'Languages and models for hybrid automata: A coalgebraic perspective'. In: *Theoretical Computer Science* (2017). ISSN: 0304-3975.

[NB16] Renato Neves and Luís Soares Barbosa. 'Hybrid Automata as Coalgebras'. In: *Theoretical Aspects of Computing - ICTAC 2016 - 13th International Colloquium, Taipei, Taiwan, October 24-31, 2016.* Ed. by Augusto Sampaio and Farn Wang. Vol. 9965. Lecture Notes in Computer Science. 2016, pp. 385–402.

[Nev+16a] Renato Neves, Alexandre Madeira, Luís S. Barbosa and Manuel A. Martins. 'Asymmetric Combination of Logics is Functorial: A Survey'. In: *Recent Trends in Algebraic Development Techniques - 23rd IFIP WG 1.3 International Workshop, WADT 2016, Gregynog, UK, September 21-24, 2016, Revised Selected Papers.* Ed. by Phillip James and Markus Roggenbach. Vol. 10644. Lecture Notes in Computer Science. Springer, 2016, pp. 39–55.

[Nev+16b] Renato Neves, Luís S. Barbosa, Dirk Hofmann and Manuel A. Martins. 'Continuity as a computational effect'. In: *Journal of Logical and Algebraic Methods in Programming* 85.5 (2016), pp. 1057–1085. ISSN: 2352-2208.

[Nev+16c] Renato Neves, Alexandre Madeira, Manuel A. Martins and Luís S. Barbosa. 'Proof theory for hybrid(ised) logics'. In: *Science of Computer Programming* 126 (2016), pp. 73–93.

[Pan09] Prakash Panangaden. *Labelled Markov Processes.* Imperial College Press, 2009. ISBN: 978-1848162877.

[Per13] Lawrence Perko. *Differential equations and dynamical systems.* Vol. 7. Springer Science & Business Media, 2013. ISBN: 978-1461265269.

[Pet96] Alejandro Petrovich. 'Distributive lattices with an operator'. In: *Studia Logica* 56.1-2 (1996). Special issue on Priestley duality, pp. 205–224. ISSN: 0039-3215.

[Pla08] André Platzer. 'Differential Dynamic Logic for Hybrid Systems.' In: *Journal of Automated Reasoning* 41.2 (2008), pp. 143–189. ISSN: 0168-7433.

[Pla10] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics.* Heidelberg: Springer, 2010. ISBN: 978-3-642-14508-7.

[Pla12] André Platzer. 'The Complete Proof Theory of Hybrid Systems'. In: *LICS'12: Proceedings of the 27th Annual IEEE/ACM Symposium on Logic in Computer Science, New Orleans, Louisiana, June 25 - 28, 2012.* IEEE, 2012, pp. 541–550. ISBN: 978-1-4673-2263-8.

[PC09] André Platzer and Edmund M. Clarke. 'Formal Verification of Curved Flight Collision Avoidance Maneuvers: A Case Study'. In: *FM' 09: Proceedings of the 16th International Symposium on Formal Methods, Eindhoven, Netherlands, November 2-6, 200*. Ed. by Ana Cavalcanti and Dennis Dams. Vol. 5850. LNCS. Springer, 2009, pp. 547–562.

[PQ08] André Platzer and Jan-David Quesel. 'KeYmaera: A Hybrid Theorem Prover for Hybrid Systems (System Description)'. In: *IJCAR'08: Proceedings of the 4th International Joint Conference on Automated Reasoning, Sydney, Australia, August 12 – 15, 2008*. Ed. by Alessandro Armando, Peter Baumgartner and Gilles Dowek. Vol. 5195. Lecture Notes in Computer Science. Springer, 2008, pp. 171–178.

[PP01a] Gordon Plotkin and John Power. 'Adequacy for algebraic effects'. In: *FoSSaCS'01: Foundations of Software Science and Computation Structures, 4th International Conference, Genova, Italy, April 2–6, 2001*. Vol. 2030. Springer, 2001, pp. 1–24.

[PP01b] Gordon Plotkin and John Power. 'Semantics for algebraic operations'. In: *Electronic Notes in Theoretical Computer Science* 45 (2001), pp. 332–345.

[PP03] Gordon Plotkin and John Power. 'Algebraic operations and generic effects'. In: *Applied Categorical Structures* 11.1 (2003), pp. 69–94.

[Pom05] Dimitrie Pompeiu. 'Sur la continuité des fonctions de variables complexes'. In: *Annales de la Faculté des Sciences de l'Université de Toulouse pour les Sciences Mathématiques et les Sciences Physiques. 2ième Série* 7.3 (1905), pp. 265–315.

[Raj+10] Ragunathan Raj Rajkumar, Insup Lee, Lui Sha and John Stankovic. 'Cyber-physical systems: the next computing revolution'. In: *DAC'10: Design Automation Conference, 47th ACM/IEEE Conference, Anaheim, USA, June 13-18, 2010*. IEEE, 2010, pp. 731–736.

[RL01] Mauno Rönkkö and Xuandong Li. 'Linear Hybrid Action Systems'. In: *Nordic Journal of Computing* 8.1 (Mar. 2001), pp. 159–177. ISSN: 1236-6064.

[RS03] William C. Rounds and Hosung Song. 'The Phi-Calculus: A Language for Distributed Control of Reconfigurable Embedded Systems'. In: *Hybrid Systems: Computation and Control, 6th International Workshop, HSCC 2003 Prague, Czech Republic, April 3-5, 2003, Proceedings*. Ed. by Oded Maler and Amir Pnueli. Vol. 2623. Lecture Notes in Computer Science. Springer, 2003, pp. 435–449.

[Rut00] Jan Rutten. 'Universal coalgebra: a theory of systems'. In: *Theoretical Computer Science* 249.1 (2000). Modern Algebra, pp. 3 –80. ISSN: 0304-3975.

[Rut06] Jan Rutten. 'Algebraic specification and coalgebraic synthesis of mealy automata'. In: *Electronic notes in theoretical computer science* 160 (2006), pp. 305–319.

[Sea13] Gavin J. Seal. 'Tensors, monads and actions'. In: *Theory and Applications of Categories* 28.15 (2013), pp. 403–433.

[Sho+07] Robert Shorten, Fabian Wirth, Oliver Mason, Kai Wulff and Christopher King. 'Stability criteria for switched and hybrid systems'. In: *Society for Industrial and Applied Mathematics (review)* 49.4 (2007), pp. 545–592.

[Sif+15] Joseph Sifakis, Saddek Bensalem, Simon Bliudze and Marius Bozga. 'A Theory Agenda for Component-Based Design'. In: *Software, Services, and Systems - Essays Dedicated to Martin Wirsing on the Occasion of His Retirement from the Chair of Programming and Software Engineering*. Ed. by Rocco De Nicola and Rolf Hennicker. Vol. 8950. Lecture Notes in Computer Science. Springer, 2015, pp. 409–439.

[Sil10] Alexandra Silva. 'Kleene Coalgebra'. PhD thesis. Radboud Universiteit Nijmegen, 2010.

[Sim82] Harold Simmons. 'A couple of triples'. In: *Topology and its Applications* 13.2 (1982), pp. 201–223. ISSN: 0166-8641.

[Sok05] Ana Sokolova. 'Coalgebraic analysis of probabilistic systems'. PhD thesis. Technische Universiteit Eindhoven, 2005.

[Spr00a] Jeremy Sproston. 'Decidable Model Checking of Probabilistic Hybrid Automata'. In: *FTRTFT'00: International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, Pune, India, September 20–22, 2000*. Ed. by Mathai Joseph. Vol. 1926. Lecture Notes in Computer Science. Springer, 2000, pp. 31–45.

[Spr00b] Jeremy Sproston. 'Model Checking of Probabilistic Timed and Hybrid systems'. PhD thesis. School of Computer Science, University of Birmingham, 2000.

[Sta11] Sam Staton. 'Relating coalgebraic notions of bisimulation'. In: *Logical Methods in Computer Science*. Vol. 7. 1. 2011.

[Ste11] Benjamin Steinberg. *Representation theory of finite groups: an introductory approach*. Springer, 2011. ISBN: 978-1461407768.

[SH11] Kohei Suenaga and Ichiro Hasuo. 'Programming with infinitesimals: A while-language for hybrid system modeling'. In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2011, pp. 392–403.

[Szy98] Clemens Szyperski. *Component Software, Beyond Object-Oriented Programming*. Addison-Wesley, 1998. ISBN: 978-0321753021.

[Tav87] Lucio Tavernini. 'Differential automata and their discrete simulators'. In: *Nonlinear Analysis: Theory, Methods & Applications* 11.6 (1987), pp. 665 –683. ISSN: 0362-546X.

[Tho09] Walter Tholen. 'Ordered topological structures'. In: *Topology and its Applications* 156.12 (2009), pp. 2148–2157. ISSN: 0166-8641.

[VV14] Yde Venema and Jacob Vosmaer. 'Modal Logic and the Vietoris Functor'. In: *Leo Esakia on Duality in Modal and Intuitionistic Logics*. Ed. by Guram Bezhanishvili. Dordrecht: Springer Netherlands, 2014, pp. 119–153.

[Vie22] Leopold Vietoris. 'Bereiche zweiter Ordnung'. In: *Monatshefte für Mathematik und Physik* 32.1 (1922), pp. 258–280. ISSN: 0026-9255.

[Wad95] Philip Wadler. 'Monads for Functional Programming'. In: *Advanced Functional Programming, First International Spring School on Advanced Functional Programming Techniques, Båstad, Sweden, May 24-30, 1995, Tutorial Text*. Ed. by Johan Jeuring and Erik Meijer. Vol. 925. Lecture Notes in Computer Science. Springer, 1995, pp. 24–52.

[Wat72] William C. Waterhouse. 'An empty inverse limit'. In: *Proceedings of the American Mathematical Society* 36.2 (1972), p. 618. ISSN: 0002-9939.

[Wit66] Hans Witsenhausen. 'A class of hybrid-state continuous-time dynamic systems'. In: *IEEE Transactions on Automatic Control* 11.2 (1966), pp. 161–167.

[Wor05] James Worrell. 'On the final sequence of a finitary set functor'. In: *Theoretical Computer Science* 338.1-3 (2005), pp. 184–199.

[Zen70] Phillip Zenor. 'On the completeness of the space of compact subsets'. In: *Proceedings of the American Mathematical Society* 26.1 (1970), pp. 190–192. ISSN: 0002-9939, 1088-6826. URL: http://www.ams.org/proc/1970-026-01/S0002-9939-1970-0261538-4/.

[Zha+01] Jun Zhang, Karl Henrik Johansson, John Lygeros and Shankar Sastry. 'Zeno hybrid systems'. In: *International Journal of Robust and Nonlinear Control* 11.5 (2001), pp. 435–451.

INDEX