Coalgebraic Modelling of Timed Processes

Marco Kick

Doctor of Philosophy

Laboratory for Foundations of Computer Science

School of Informatics

University of Edinburgh

2003

Abstract

This thesis presents an abstract mathematical account of *timed processes* and their *operational semantics*, where time is modelled by a special kind of *monoids*, so-called *time domains*, and (the operational behaviour of) timed processes is represented by special *labelled transition systems*, so-called *timed transition systems* (TTSs), together with *time bisimulation* as an appropriate notion of equivalence of such processes.

The importance of monoid-related notions for describing timed phenomena is then illustrated by showing that TTSs are the same as the *(partial) actions* of the monoid of time; moreover, *total* monoid actions are also shown to arise naturally in this approach in the form of *delay operators*. The two kinds of monoid actions are suitably combined in a new notion of *biaction* which captures the interplay of two very important features of timed processes: letting time pass and delaying.

The TTSs are then characterised as *coalgebras* of a novel *evolution comonad*, which is inspired by well-known categorical descriptions of total monoid actions; in doing so, a coalgebraic description of time bisimulation is also provided. Additionally, biactions are characterised as *bialgebras* of a *distributive law* of a monad (for total monoid actions) over a comonad (the evolution comonad for partial monoid actions).

Building on these results, it is possible to obtain an abstract categorical treatment of *operational rules* for timed processes. The approach taken here is based on the framework by Turi and Plotkin [TP97], using distributive laws and bialgebras (similar to the treatment of biactions), and which, subsequently, is extended to accommodate *behaviour comonads*, as required for the coalgebraic description of TTSs.

These abstract rules then form the basis for the development of several new *syntactic rule formats* for timed processes which describe classes of particularly 'well-behaved' languages for specifying timed processes.

Acknowledgements

It is hard to express my enormous gratitude towards my two supervisors, Gordon Plotkin and Daniele Turi: the combination of Gordon's vast experience and (sometimes almost frightening) quickness in recognising the right mathematical structures (often while—or despite?—discussing some half-baked ideas of mine with him), and Daniele's enthusiam, immediacy and patience, together with their shared appreciation of beautiful and elegant mathematical solutions (not to speak of Gordon's generosity in financial matters!), provided me with an almost ideal working environment; I hope that I have not failed completely to meet their expectations.

I would also like to thank John Power for many valuable suggestions and discussions, in particular for his help with the material on heterogeneous timed processes in Chapter 7, effectively taking on the job of third supervisor during the last stages of writing up; I would also like to acknowledge helpful discussions with Alexander Kurz and Alex Simpson; further thanks go to Terry Stroup for his help with the application for the PhD place.

I am also grateful to Margaret Davis, Dyane Goodchild, Monika Lekuse, and Bill Orrok for efficiently handling administrative tasks, and to EPSRC for financial support under grant GR/M56333.

Furthermore, I would like to thank the following people for their friendship and/or making Edinburgh a great place to stay: Tom Chotia, Markus Frick, Carsten Führmann, Martin and Berit Grohe, Jan Jürjens, Martin and Becky Lange, Bruce McAdam, Conor McBride, and those I forgot to mention.

Last, but certainly not least, I would like to thank my parents, my brother, and my girlfriend Andrea for their continuing support, who helped me through the sometimes difficult period of more than three years in a foreign country, where people think that having two water taps is an act of genius—which it definitely is not!

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Preliminary versions of (parts of) chapters 3–6 have appeared in the papers [Kic02a, Kic02b].

(*Marco Kick*)

Table of Contents

Chapter 1

Introduction

We aim at presenting a mathematical theory of *timed processes*, and their *operational behaviour*. In applications, such timed processes are usually used to model *distributed real-time systems*, e.g., in traffic control, or embedded systems.

Our approach to providing such a theory is based on the use of *categorical* methods, some of which have been present in the field of programming language semantics for quite some time—to some extent since at least the 1970's, i.e., almost since the inception of the whole field.

In this introduction, we want to briefly survey the different aspects mentioned above, in order to give a brief outline of what is to come in later chapters. We will start with a brief motivation as to why time plays an important role in the domain of distributed systems, in particular such ones that deal with safety-critical applications.

Following this, we will give some hints at the wide field of programming language semantics, focusing on operational and denotational semantics, and their mutual relationship, as established in so-called adequacy results.

At that point, we want to spell out some of the apparent challenges involved in our goal, followed by a brief outline of our approach how to overcome these challenges, and the contributions of this thesis.

We conclude the chapter with a brief summary of all remaining chapters of the thesis.

1.1 Distributed Real-Time Systems

Computer systems have become pervasively used in daily life, in particular to carry out complex tasks with precision and speed often beyond human abilities, e.g., systems for traffic control must govern switching of lights in a whole city, or coordinate starting and landing of air planes; another example would be an operating system that, by providing a layer of abstraction above the actual hardware, forms the basis for other programs to be run on top of it. What such systems have in common is that they are usually not started once, calculate some results and then terminate, but that they should (at least ideally) keep on running *continuously*, all the while *interacting* with their environment.

Because of this, systems like the above have been called *reactive systems* [Pnu85]: rather then carrying out monolithic tasks, such programs usually *react* appropriately to stimuli from outside, e.g., if a pedestrian presses a button on a traffic light, the traffic control system should eventually change the lights in such a way such that the pedestrian can cross the street; in the operating system example, if a program requests more memory or demands access to some particular piece of hardware, the operating system has to react to such a request.

Reactive systems are usually also *distributed* systems, in general consisting of several components in different locations which communicate via different means. As an example, again consider the traffic control system: it could have little control units, spread all over a city at all the traffic lights, which exchange signals with a central coordination facility. Analogously, an operating system consists of several different parts interacting with each other: a kernel, a scheduler, hardware drivers and so forth, in addition to the programs, also known as 'processes,' which are running on top of it.

Applications like such traffic control systems are furthermore referred to as *real-time* since (software or hardware) components involved are required to satisfy (often tight) *timing constraints* whose violation could have dire consequences: we already said that the pedestrian should be allowed *eventually* to cross, for example, inside some reasonable interval of time like two minutes. The same applies to operating systems: allocation of memory, or access to hardware must be dealt with swiftly as not to affect the execution of the actual programs.

Usually, these systems are also *safety-critical*: they need to work *correctly* and *robustly* because failure to do so could have catastrophic effects. Just imagine the traffic control allowing cars from all sides of a crossing to drive at the same time due to a system failure! Or consider the domain of *embedded systems* as, e.g., used in cars for controlling the engine and other parts: breakdown or malfunctioning of such systems can have drastic consequences, e.g., imagine the brakes failing in a car. In addition to the potentially catastrophic outcome of system failure, since such systems are usually at least partially realised in hardware, it could also be extremely costly to correct mistakes once they have occurred, thereby also posing a serious *economic* threat.

In general, reactive systems like the above have been successfully modelled using *processes* and *process calculi*, e.g., CCS [Mil80, Mil89], CSP [BHR84, Hoa85], and ACP [BK84, BW90], to only name the most well-known ones: these are small formal languages representing abstractions of concrete systems, focusing on specific aspects of the systems, in particular the synchronisation and communication taking place in distributed systems. Such calculi usually have a *formal* semantics, i.e., the behaviour of programs is specified in precise, mathematical terms. Because of this, process calculi are very well-suited for applications of *formal verification techniques*, i.e., determining (as far as possible) whether a program will behave correctly *before* it is actually run, e.g., by means of *model checking* [CE82]. As a result of this, they have also been used to model safety-critical reactive systems. There are also other approaches to specifying and modelling concurrent systems, in particular automata-theoretic ones, see, e.g., [Var91].

Furthermore, in order to cope with real-time requirements, programming or specification languages for reactive systems also have to include features to specify *timing behaviour* of programs, in addition to describing functionality as in traditional programming languages. Consequently, over the past decade and a half, various attempts have been made to extend processes and process calculi, as well as automata-theoretic approaches to specifying concurrent systems, with some notion of time in order to model (abstractions of) real-time systems—see, e.g., [RR88, MT90, Wan90, Wan91, BB91, Sch95, SDJ+91, HNSY92, NS94, AD94, WPD94, HR95, DB96, LW00b, LW00a, BM01]. Many of the resulting models and calculi feel quite ad-hoc: some of the underlying assumptions and design decisions seem at least questionable, in particular as far as the mathematical foundation for such *timed processes* and their behaviour is concerned. One design decision that is essentially uniformly present in the literature, and one that we will also adopt, is to *separate* timing behaviour from computation by means of using two distinct kinds of transitions: *action* transitions modelling *instantaneous* computations, and *time* transitions, modelling the passage of time without any external interactions[1].

1.2 Semantics of Programming Languages

1.2.1 Operational Semantics

Operational semantics focuses on the description of programs in terms of their (step-by-step) *execution* on an (abstract) machine, the main concern being the *behaviour* of programs. This can be regarded as describing an interpreter for the language, hence the operational semantics provides a basis for implementations by a *machine-oriented* view of the language. A program's meaning is defined by *operational rules* which determine sequences of *transitions*, i.e., atomic steps the program can perform. In this way, one obtains a *(labelled) transition system* ((L)TS) [Plo81, WN95] on the set of programs.

The most common way to present such rules is Plotkin's *structural operational semantics* (SOS) [Plo81] where transitions are defined by induction on the structure of programs: starting from the basic constructs of the language, i.e., the simplest possible programs, the transitions of compound programs are defined from the transitions of its components, together with the intended meaning of program constructors. In this way, one ends up with some form of transition system for the programs which is called the *intended operational model* of the language.

Taking the operational point of view, two programs should be considered equal when they exhibit the same behaviour. Hence, we want a notion of *operational* or *be-*

[1]The mentioned exception to this is the operational semantics for Timed CSP reported in [Sch95] which also uses two kinds of transitions, yet action transitions also have a *duration* attached to them, i.e., are, in some sense, not necessarily instantaneous; this will be discussed at a later point, cf. Chapter 7.

havioural equivalence of programs. For an operational semantics to be 'well-behaved,' its operational equivalence should be a *congruence* with respect to the constructs of the language: if two programs P_1 and P_2 are behaviourally equivalent, it should be possible to exchange P_1 and P_2 inside a bigger program P without affecting the operational behaviour of P; this property is also known as *substitutivity*.

In order to obtain definitions of operational semantics for which the associated behavioural equivalences are indeed congruences, substantial amount of work has, in particular, been done in order to identify syntactical restrictions on SOS rules, so-called *rule formats*, which ensure such a crucial property. Drawbacks of such formats are that, firstly, finding a suitable format is not really explaining the deeper reason for the congruence result to hold as a format is intrinsically syntactic. Secondly, they cannot easily be extended to cope with extra information needed to incorporate additional features.

Rule formats have (mostly) been developed and explored for languages without variable binding, so-called *first-order languages*, which have been extensively used in the study of non-determinism, concurrency, and distributed systems. In particular, the formal (operational) semantics of process calculi, as mentioned before, is usually specified by SOS rules. The behavioural equivalence, then often called *process equivalence*, is usually some variant of *(strong) bisimulation* [Par81, Mil89]; see [Gla01] for an overview of behavioural equivalences for processes.

A very well-studied and expressive format for such languages is *GSOS* [BIM95], which is known to verify the property that strong bisimulation is a congruence if the rules are of a specific shape; obtaining such result by hand can be quite difficult, even for simple languages. Other formats include the *tyft/tyxt* format [GV92], and its extension to allow *negative premises*, the *ntyft/ntyxt* format [FvG96]; see also the handbook article [AFV01] for more details on SOS and rule formats.

1.2.2 Denotational Semantics

The machine-oriented view of operational semantics is traditionally contrasted with denotational semantics [Sco70, SS71, Ten91]. Denotational semantics corresponds to a more *human-oriented* point of view (in the sense that a machine can hardly deal with the involved methods), maybe less elementary but more appropriate for abstract reasoning (as humans are capable of). It is based on a more mathematical view of programs, describing the semantics of a language by an *interpretation (function)*, mapping a program to its 'meaning,' also known as the *denotation*, in some abstract *semantic domain*. Always, such interpretations are *compositional*: the denotation of a compound program is given as a function of the denotations of its components, and how those are assembled.

The semantic domain of an interpretation is typically a final (i.e., greatest) solution to a *domain equation* of the form $X \cong BX$ for a suitable endofunctor B on a category $\mathcal{C}$, usually some kind of *complete partial orders* (cpo's) to account for undefinedness, recursion, and approximation. In this way, a domain for interpreting for all the constructs of the language is provided; the greatest solution is taken to obtain denotations for 'infinite,' i.e., recursive, and hence potentially non-terminating, computations. This aspect of denotational semantics has been thoroughly studied in field of *domain theory* [Sco82, Plo83, Plo85, GS90, AJ94, FJM$^+$96].

In the classical setting of general programming languages, the denotation of a program is usually given as a function mapping inputs to outputs: see [Win93] for a textbook on the subject. Since one is commonly working with cpo's, such functions have to satisfy the requirement of being *continuous* in the sense of preserving certain least upper bounds in the partial order. Intuitively, these upper bounds correspond to the limits of approximations, and consequently, the denotation of such a limit should be obtained as the limit of the denotations of its approximations; this is also related to issues about *computability* (see [Plo83]).

Unfortunately, denotational semantics for non-deterministic, and/or concurrent, languages is not nearly as well developed as for deterministic (functional or imperative) languages. A very common approach, also for operational semantics, is to model concurrency by non-determinism, e.g., consider languages like CCS which satisfy the so-called *expansion law* [Mil89]. Based on this, one then models non-deterministic computations by functions which, for any given input, do not necessarily return a single result, but a *set* of results, each element of the set corresponding to a different, non-deterministic execution of the program.

This approach was advocated by Plotkin [Plo76, HP79, Plo82a], for finite or countable non-determinism, i.e., allowing at most finitely many, or also countably many different executions of the same program. It is based on so-called *power domains*, which are essentially suitable adaptations of the power-set functor on **Set** to partially ordered sets; this is possible by observing that the power-set functor on **Set** is actually the *free semi lattice* monad, which can then be transferred to other categories, cf. [HP79]. Later, powerdomains have also been applied to account for *probabilistic* non-determinism [Plo82b, JP89, Jon90], i.e., non-determinism arising from probabilistic computations, e.g., to model unreliable systems which fail with a certain probability.

1.2.3 Adequacy Results

For a sufficiently complete description of a language, both operational and denotational semantics are required as the two complement each other. Therefore, it has to be ensured that operational and denotational semantics agree. Taking operational semantics as the more primitive notion to measure the 'quality' of a denotational semantics against, this means that, if two programs have the same denotations, they should also be operationally equivalent, usually defined by saying that the two programs cannot be distinguished in any program context, see [Mil77]. A denotational semantics with this property is called *adequate*—see, e.g., [Win93]. There is also a notion of *computational adequacy* [Pit94]: a denotational semantics is adequate in this sense, again with respect to an operational one, if operational evaluation terminates only if the denotation of a program is defined, i.e., again, the denotational semantics must coincide with the operational semantics.

Conversely, one would also like to have that, if two programs are equivalent operationally, they also have the same denotation. If this is the case, the denotational semantics is called *fully abstract* [Mil77] with respect to the operational equivalence: one semantics *completely determines* the other one. Adequacy can usually be ensured by appropriately defining the denotational semantics, while full abstraction fails in a lot of cases because the semantic domain for a given denotational semantics contains 'too many' denotations, what was called 'over-generous' in [Mil77], i.e., there are elements in the semantic domain which cannot be denoted by any program.

Full abstraction has mainly been studied in connection with pure functional languages, in particular the full-abstraction problem for the language PCF [Plo77], a simply typed λ-calculus [Bar84] based on Scott's LCF [Sco93, Mil72], was a long-standing open problem since the 1970s, when Plotkin [Plo77] showed that the standard model, using cpo's and continuous functions, was *not* fully-abstract, because the model contained a *parallel-or* construct which could not be specified in what is essentially a sequential programming language. Finally, in 1994, independently in two papers by Abramsky et al. and Hyland and Ong (see [AJM00, HO00] for extended journal versions), a non-syntactic fully-abstract model for PCF was constructed (already as early as [Mil77], Milner introduced a *term* model and showed that this was, up to isomorphism, unique).

1.3 Aspects of Categorical Methods in Semantics

The intuitive duality between operational and denotational semantics becomes even more apparent when considering categorical formulations of the two styles of programming language semantics.

1.3.1 Initial Algebra Semantics

In the case of a first-order language, the constructs of the language are usually described by a *signature* Σ in the sense of universal algebra, see [BS81]. To each such signature, we can associate a so-called *polynomial* functor [RB85, Tur96] also denoted by Σ. Assuming that a category $\mathcal{C}$ has enough structure to interpret Σ, a Σ-*algebra*, cf. [Mac97], is then a *denotational model* for the language, the semantic domain being one particular such model. Such models are usually called Σ-interpretations [BJ89, EFT96]. It turns out that the set of programs is the *initial* Σ-algebra, and so, there exists a *unique* homomorphism of Σ-algebras mapping programs to the semantic domain. This unique map is called *initial algebra semantics* [GTW78] and, being a *homomorphism* of Σ-algebras, is automatically compositional.

1.3.2 Final Coalgebra Semantics

Final coalgebra semantics uses a *canonical* semantic domain starting from a *behaviour*. Therefore, it provides a link between operational and denotational semantics. As explained previously, operational semantics is essentially based on *labelled transition systems* [Plo81]. Such transition systems can be viewed as *B-coalgebras* [Acz88] for an appropriate *behaviour functor* B on the category of sets and functions (of course, this can be generalised to an arbitrary category $\mathcal{C}$). In this way, one obtains a category of B-coalgebras, defined dually to the category of algebras, corresponding to a class of transition systems, 'transition systems of *type B*,' which can be regarded as *operational models*. For example, the coalgebras for the functor[2]

$$B_A X \stackrel{\mathsf{df}}{=} \mathcal{P}_{\mathsf{fi}}(X)^A \tag{1.1}$$

correspond to the class of all *image-finite* transition systems with labels from the set A, where $\mathcal{P}_{\mathsf{fi}}(X)$ denotes the set of all *finite* subsets of X: given a state $x \in X$ and a label $a \in A$, there are only finitely many successor states in X.

The intended operational model of the language, defined by the SOS, is an example of a B-coalgebra on the set of programs. The final B-coalgebra—if it exists—is then a *canonical* operational model, viz., the collection of *abstract* or *global behaviours*: by finality, there is a unique morphism of B-coalgebras from the intended operational model to the final B-coalgebra, mapping a program to its 'abstract' behaviour. This map is called *final coalgebra semantics* [RT94, TR98]. There is also a coalgebraic notion of bisimulation, due to Aczel and Mendler [AM89], and the final coalgebra satisfies the property that its internal equality corresponds to this notion of bisimulation (this is also known as 'internal full abstraction' [Abr91] or 'strong extensionality' [TP97]). In the example of image-finite transition systems, i.e., the B_A-coalgebras, the final coalgebra exists, see, e.g., [Bar93, RT94, Tur96, JPT$^+$01], and is the set of all image-finite *synchronisation trees* [Mil80, WN95] quotiented by strong bisimulation, which is coalgebraic B_A-bisimulation, so bisimilar elements are indeed equal.

As the semantic domain for initial algebra semantics is typically a final solution to a domain equation $X \cong BX$, it is the carrier of the final B-coalgebra. Furthermore,

[2] The notation B_A is chosen to indicate the *action* behaviour of a process, as opposed to the timing behaviour; the subscript is not connected to the set A of labels.

if the intended operational model can also be expressed as a B-coalgebra on the programs, both initial algebra semantics and final coalgebra semantics define maps from the set of programs to the semantic domain, a compositional one stemming from initial algebra semantics, the other stemming from final coalgebra semantics, respecting coalgebraic B-bisimulation. If these two maps coincide, one has an adequate denotational semantics with respect to the operational one [RT94, Tur96]: the compositional denotational semantics stemming from initial algebra semantics inherits the property from final coalgebra semantics that observationally equivalent (here: B-bisimilar) programs have the same denotation.

1.4 Challenges

In order to arrive at a satisfactory mathematical account of timed processes and their operational semantics, several challenges have to be overcome. Firstly, it is not at all easy to determine what 'timed process' should actually mean: which models do we want to consider when trying to work out a mathematical basis for them? There are a plethora of models in the literature, some based on transition systems, some automata-theoretic. As for calculi, the situation is even worse, there are many languages with slightly different assumptions: which of those should be taken into account when trying to give a foundational account?

A second challenge is the required mathematics: what are the principal notions needed to describe timing behaviour? Timed processes have a very rich structure, e.g., how should time itself be modelled? How should an account of *communication* taking place over time be given since reactive systems usually have components distributed over a network that are communicating with each other? What is the right mathematics for describing the timed operational behaviour of such communicating timed processes?

Finally, once the questions about timed processes and the required mathematics have been settled, how can the two things be combined to give a satisfactory mathematical account of the (structural) operational semantics which is generally based on *rules of inference*? What framework is there to be able to really bring to light the right mathematical notions without becoming too static in the sense that the focus on interactive and reactive systems is lost (consider, e.g., methods for denotational semantics which are mathematical yet, necessarily, un-operational)?

1.5 Our Approach

To address the above challenges we plan to proceed as follows. As far as models of timed processes are concerned, we will present one of the most frequently used models, viz., *timed transition systems* as, e.g., occurring in [MT90, Wan90, NS94, HR95], explaining the main concepts underlying this model. As for calculi based on this model, we will try to exhibit some of the more common design decisions and present the calculi according to a taxonomy based on these design decisions.

As far as the required mathematics goes, we take the view that the domain of time should be modelled as a (partially ordered) monoid. An important role will be played by the *actions* of this monoid of time: such monoid actions describe a way of "transforming" or "changing" elements of a set in a way that conforms with the monoid operations. When the monoid represents time an action of that monoid therefore means a transformation by, or over, time.

It will turn out that we will need both *total* and also *partial* actions of monoids to give good accounts of various features of timed processes. The typical example of a total action is given by *delaying*, modelled using the addition of the monoid: delaying a process simply "adds" to its delay potential in the sense that it can wait or run longer; it also means that the process starts doing its actual work later.

The main example of a partial action is, somewhat dually, the way a process *consumes* time, utilising a *partial subtraction* function on the monoid of time: the longer a process operates the shorter the amount of time it has left to consume during the calculations to be carried out. Partiality comes into play since some processes may only have a certain *limited* amount of time to consume for their calculations, e.g., due to some *timing constraints* placing deadlines upon them, so these processes must not exceed their time limits and hence sometimes may not be able to consume any more time. Hence, partial actions play a central role in describing the way a process *evolves* through time.

To combine consuming time and delaying, we need structures endowed with both partial and total monoid actions of the time domain and with the two actions suitably related in order to represent the intuitive connection between waiting and delaying: only as much time can be consumed as was previously added by delaying, but in general not more. Such structures will be introduced in the form of *biactions*, and they will essentially describe a primitive algebra of timed processes.

In order to be able to give an account of the operational semantics of timed processes incorporating the outlined mathematical notions, we will use the *categorical* approach of [TP97], which is based on the notions of distributive law [Bec69, PW99] and the *bialgebras* [TP97] for such a law; in particular, we will describe the models of timed processes in a *coalgebraic* fashion.

To be able to do this, we will give categorical descriptions of both total and partial actions of a monoid. For total actions, there are several well-known categorical characterisations in the literature, e.g., they can be considered as presheaves over a category with one object representing the monoid, leading to both algebraic and coalgebraic descriptions of such total actions. For partial actions, the situation is a bit more complex since we want the actions to be partial but their morphisms to be *total* maps respecting the actions (up to being defined). We will give a categorical description of such partial actions as coalgebras for a new *evolution comonad* on the category of sets and total functions: essentially, this comonad enables us to "curry" partial actions.

In order to model timed processes based on timed transition systems as structures which can both delay and consume time, we have previously mentioned the biactions of a time domain. We will also provide a categorical characterisation of these biactions by defining a specific distributive law, which distributes total over partial monoid actions, whose bialgebras turn out to be biactions.

Since partial actions are defined as coalgebras for a comonad, we need to slightly extend the framework of [TP97] to give *abstract operational rules* for timed processes. Building on that, we will present concrete rule formats obtained by instantiating the abstract format to capture existing calculi from the literature.

1.6 Contributions

We provide the mathematical basis for an operational understanding of timed processes. To be able to use categorical methods makes it necessary to identify deeper structure due to the abstract nature of category theory (which is sometimes cited as its disadvantage). In our case, the abstract framework forces us to go to the heart of how timed processes behave and therefore, we hope to present a clear and thorough picture, unravelling the mathematics lying at the core of the operational semantics of timed processes as a first contribution.

As far as the categorical framework from [TP97] is concerned we hope to document its utility and flexibility by accommodating timed processes in pretty much the same way, after the already mentioned technical extension to behaviour comonads, as was done in [TP97, Tur97] for standard process calculi. In particular, substantiating the claim of [TP97] that abstract categorical rules can lead to concrete rule formats, we present such formats for timed processes, derived from our abstract rules, which ensure "nice" properties for calculi that fit them and we show that some existing calculi do indeed fit these formats. To our knowledge, no general rule formats for timed processes have been proposed before, so this is our second contribution.

A third contribution, related to the first one, is a partial unification of the field of timed processes: due to our abstract categorical approach, similar structures throughout the field are identified and highlighted by the coalgebraic treatment. Many calculi are seen to fit the same basic format for operational semantics, distinctions arising from different design decisions. Hence, the design decisions can also be classified by how 'naturally' they fit the categorical framework, i.e., whether they introduce any obstacles for a conceptual treatment. We believe that this form of naturality gives some justification as to which design decisions should be considered more canonical than others, which should also further a unified view of the field.

1.7 Layout of this Thesis

We will now briefly present the layout of the thesis, giving short summaries of all chapters. We present the background to our work in Chapter 2, including some general mathematical preliminaries mostly to do with notation, some information on process calculi and their timed extensions, as well as a brief summary of the necessary notions from category theory.

Chapter 3 is then going to present the mathematical model we will use for timed processes: time domains to model time, and timed transition systems and bisimulations, and their reformulations in terms of partial monoid actions of a time domain, to model the behaviour of processes themselves; this is complemented by the introduction of delay operators, which are simply total monoid actions, and biactions, in order to model the interplay between idling and delaying.

The following Chapter 4 then presents categorical characterisations of the notions introduced in the preceding chapter. In order to be able to describe partial monoid actions, which are not as well studied in the literature as their total counterparts, we introduce the new comonad of *evolutions,* whose coalgebras are precisely the partial monoid actions introduced in Chapter 3. Furthermore, we show that over *discrete* time, corresponding to taking the natural numbers as the model for time, the partial actions can be described in a much less complicated way as the coalgebras for a simple functor. Finally, we give a categorical formulation of the previously introduced biactions as bialgebras for a distributive law.

This then puts us in a position to describe abstract operational rules for timed processes in Chapter 5 by means of adapting the bialgebraic approach presented by Turi and Plotkin in [TP97], which we are first going to explain in some detail. In the case of discrete time, we can immediately apply that framework, due to the previously established simpler characterisation of TTSs in that particular case. To deal with the general case, we need to extend the bialgebraic framework to deal with constrained operational models which are given by the coalgebras of a comonad. We present two different types of abstract rules, one in the most general fashion (which will also be shown to be as general as possible), and another one which is less general yet will turn out to still be powerful enough to deal with languages for timed processes.

Having given abstract operational rules for timed processes, we can then turn our attention to *syntactic* rule formats for that case. Again, for discrete time, things are a lot simpler due to the more primitive categorical characterisation; we present a format, which we call *deterministic single-label GSOS*, which is closely related to the well-established GSOS format. In the general case, reflecting the two kinds of abstract rules, we will present two kinds of syntactic characterisations. The first one, based on the less general abstract rules, uses *schematic* rules involving variables ranging over time; this 'format' is constrained to the extent that we even hesitate to call it a format: we really only specify a bunch of *operator formats*, which might be regarded as 'blueprints' for concrete operators, and that these operator formats induce abstract categorical rules matching the ones given in Chapter 5. Despite the obvious shortcomings in terms of generality, this restricted 'format' is still expressive enough to include most of the operators found in the literature. We conclude the chapter with a complete characterisation of the more general abstract rules from Chapter 5: for this, we use *meta rules* which are convenient abbreviations for infinite sets of infinitary rules. Even so, it will turn out that, in order to completely capture all possible rules, we still need infinite sets of such meta rules, thereby showcasing the generality of the abstract rules, yet also more than hinting at shortcomings in the abstract model.

Up to this point of the thesis, the focus has exclusively been on modelling the timing behaviour without any attention paid to the actual *computational* behaviour of timed processes. To this end, Chapter 7 presents ways to combine letting time pass with performing computations in the way that this is done in the standard calculi for timed processes. For discrete time this is not very hard: one simply uses the *product* of the appropriate behaviours for both action and time transitions. We wish to use the same idea for arbitrary time domains, yet we encounter the problem that the time transitions are defined by coalgebras for the evolution comonad; hence we need to form the product *in the category of comonads*, which is vastly different from simply taking the point-wise product as in the case of functors. As it turns out, the required *product comonad* is defined as a composite comonad induced by a distributive law and so, we can also use that law in a two-level approach, exploiting the equivalence between distributive laws and *liftings* of functors/comonads. Based on the thus obtained behaviours for combining the two kinds of transitions, we can then describe abstract rules for that case, as well as present work towards syntactic rule formats.

We conclude the thesis with Chapter 8, containing a summary of the thesis, some discussion of the results, including comparison with related work, and directions for future research.

Chapter 2

Background

2.1 General Preliminaries

We mark the ends of proofs, definitions, and examples (and related things) with $\square$, $\blacksquare$, and $\Diamond$, respectively. For dealing with partial mathematical expressions, we introduce the following notations:

Definition 2.1
For a partial expression e, $e\downarrow$ ($e\uparrow$) denotes that e is defined (resp. undefined). Define *Kleene implication* $\precsim$ of two partial expressions e and e' as

$$e \sqsubseteq\!\!\!\!\!\sim e' \overset{\mathrm{df}}{\Leftrightarrow} (e\downarrow \Rightarrow\ (e'\downarrow \wedge\, e = e'))$$

(where $=$ is the standard equality predicate). *Kleene equality* $\simeq$ is defined as

$$e \simeq e' \overset{\mathrm{df}}{\Leftrightarrow} (e \sqsubseteq\!\!\!\!\!\sim e' \wedge\ e' \sqsubseteq\!\!\!\!\!\sim e)$$

Given a partial order $\leq$, define *Kleene inequality* $\lesssim$ as

$$e \lesssim e' \overset{\mathrm{df}}{\Leftrightarrow} (e\downarrow \Rightarrow\ (e'\downarrow\ \wedge e \leq e'))$$

extending $\leq$ to partial expressions. $\blacksquare$

More concretely, we have that $e \simeq e' \ \Leftrightarrow\ (e\downarrow \Leftrightarrow\ e'\downarrow) \wedge\ (e\downarrow \Rightarrow\ e = e')$. Note that $\lesssim$ is an 'ordered version' of $\sqsubseteq\!\!\!\!\!\sim$ and the two notions coincide if $\leq$ is the discrete order, i.e., $\leq$ is the identity relation. In any case, $e \simeq e'$ is always equivalent to $e \lesssim e' \wedge e' \lesssim e$.

When defining functions, we write $f : X \to Y$ to mean that f, X, and Y are, respectively, *name*, *domain*, and *codomain* of the function. Such a function is *well-defined* if $x \in X$ implies $f(x) \in Y$. Sometimes, λ*-notation* is used (inspired by the λ*-calculus* [Bar84]):

$$f = \lambda x.e(x)$$

to state that the value $f(x)$ is equal to $e(x)$, where e is any mathematical expression (which might, or might not, contain occurrences of x).

When defining partial functions, we use the notation $f : X \rightharpoonup Y$ with the same interpretation as above, except that $f(x)$ need not always be defined; λ-notation is used in the following way

$$f = \lambda x. \begin{cases} e(x) & \text{if } e(x)\downarrow \\ \text{undef} & \text{if } e(x)\uparrow \end{cases}$$

expressing that the (potentially partially defined) expression e describes the values of f, i.e., f is the partial function such that $f(x) \simeq e(x)$. Well-definedness is suitably adapted for partial functions: $f(x)\downarrow$ implies $f(x) \in Y$ (much like the distinction between *total* and *partial correctness* of programs, see [AO97]).

We denote the set of all natural (rational, real) numbers by $\mathbb{N}$ (resp. $\mathbb{Q}$, $\mathbb{R}$), and we denote the *cardinality* of a set X by $|X|$.

2.2 Transition Systems, Bisimulation, and Operational Semantics

The definitions of an *operational semantics* for many programming languages, in particular in the area of distributed and concurrent processes, is usually based on *labelled transition systems* [Plo81]: (the meaning of) a program is described by a series of *atomic* steps, so-called *transitions*, intuitively corresponding to individual instructions being executed when running the program on an *(abstract) machine*.

Definition 2.2
A *labelled transition system* (LTS) is a tuple $\langle S, L, \to \rangle$ where

- S is a set of *states*,

- L is a set of *labels*, and
- $\rightarrow \subseteq S \times L \times S$ is the *transition relation*

Instead of writing $\langle s, l, s' \rangle \in \rightarrow$, one commonly uses the *transition* notation $s \xrightarrow{l} s'$, where s and s' are called the *source* and the *target*, respectively, of the transition. ■

Describing (in particular distributed) systems by such LTSs, an important question is when two LTSs should be considered equal. There are many different such notions of equivalence amongst which *(strong) bisimulation* [Mil80, Par81] is a standard one because it intuitively captures the idea that two equivalent systems should be able to perform 'the same steps' in an 'interactive' manner, i.e., during the respective executions; states in LTSs represent different stages of a program's execution, so the following definition formalises the described intuition:

Definition 2.3

Relative to two LTSs $\langle S_i, L, \rightsquigarrow_i \rangle$, $i \in \{1, 2\}$, with the same labels L, an *(action) bisimulation* is a relation $R \subseteq S_1 \times S_2$ such that $\langle s_1, s_2 \rangle \in R$ implies for all $l \in L$

$$(\forall s_1' \in S_1).\ s_1 \xrightarrow{l} s_1' \ \Rightarrow\ (\exists s_2' \in S_2).\ s_2 \xrightarrow{l} s_2' \ \wedge\ \langle s_1', s_2' \rangle \in R$$
$$(\forall s_2' \in S_2).\ s_2 \xrightarrow{l} s_2' \ \Rightarrow\ (\exists s_1' \in S_1).\ s_1 \xrightarrow{l} s_1' \ \wedge\ \langle s_1', s_2' \rangle \in R$$

Then the relation $s_1 \sim_a s_2$ holds if there is an action bisimulation containing the pair $\langle s_1, s_2 \rangle$. ■

We should point out that Definition 2.3 is slightly more general than the one usually used, e.g., in [Mil89], because it allows for two different sets of states: bisimulation is usually defined relative to only *one* LTS. Furthermore, the notation $\sim_a$ for strong bisimulation is non-standard; we need to introduce the extra subscript to distinguish later on bisimulations with respect to two disjoint sets of labels.

Coming back to the operational semantics of programming languages, one uses *operational rules*

$$\frac{\text{Hypotheses}}{\text{Conclusion}}\ \text{Side conditions} \tag{2.1}$$

where both the hypothesis and the conclusion are, respectively, sets of transitions and single transitions, while the side conditions usually involve labels of transitions; note that quite frequently, also *negative premises* are used, i.e., expressions of the form

$$s \not\rightarrow \qquad \text{or} \qquad s \overset{l}{\not\rightarrow}$$

which, respectively, denote the absence of *any* transitions of with source $s \in S$, or of transitions with label $l \in L$ from s; however, one has to be careful whether a set of operational rules with negative premises actually makes sense—see [vG96].

Structural operational semantics (SOS) [Plo81] is the most common way of specifying the operational behaviour of programming languages: 'structural' refers to the fact that, based on the behaviour of *components* described in the hypotheses, the conclusion then *inductively* defines the semantics of compound terms; for example, consider the following (standard) rule for *parallel composition*:

$$\frac{x \overset{\alpha}{\rightarrow} x'}{x|y \overset{\alpha}{\rightarrow} x'|y}$$

Note how the transitions of $x|y$ are defined inductively with respect to those of x.

2.3 Timed Process Algebras

Our work aims at a theory of well-behaved operational rules for timed processes; hence, we have to consider languages for such timed systems. We do so in two steps: first, we present a taxonomy of design decisions for such languages, and second, we describe some examples of languages to be used later on and classify them according to the taxonomy.

2.3.1 A Taxonomy of Design Decisions

This taxonomy presents a (by no means exhaustive) overview of common design principles for timed process calculi; we shall keep the classification rather abstract since the different decisions will be revisited, and explained in more detail, in the subsequent section on specific languages.

The most common design decision is to *separate* the *computational* and *time-passing* (without external interaction, sometimes also referred to as *idling*) aspects of real-time behaviour. More concretely, two kinds of transitions are used: *action* transitions (like in standard process algebras [Hoa85, Mil89]) modelling computations, or interactions with the environment, and *time* transitions with their labels denoting *durations* of idling periods. As a consequence of this separation, action transitions are usually assumed to be *instantaneous*, i.e., they take *no* time[1].

After adopting the separation in some form, the next decision is the choice of *labels* for time transitions; more abstractly, this corresponds to choosing a suitable representation of (a model of) time. Obvious choices are concrete sets of numbers, like $\mathbb{N}$ (the set of natural numbers), $\mathbb{Q}_{\geq 0}$ (the set of non-negative rational numbers), or $\mathbb{R}_{\geq 0}$ (the set of non-negative real numbers). If such a set is chosen, this is called *quantitative* time because the resulting model contains explicit *absolute* timing information about durations, delays, and so on[2].

A different possibility is to use *symbolic* durations, obtaining a *qualitative* notion of time: a time transition labelled with such a symbolic duration denotes the passage of *some unspecified* amount of time; consequently, the resulting models then can only express *relative* timing information, e.g., one computation occurs after another because a time transition was performed in between them. A convenient intuitive interpretation is to think of these symbolic durations as *clocks*, as used in connection with synchronous hardware. This interpretation is, e.g., advocated in [AM94, NS94, CLM97]: thus, time transitions can be thought of as *ticks* of such clocks signalling the start of a new *clock cycle*.

Subsequently, according to which view of time is chosen, there are more decisions to be taken. When using quantitative time, the most common choice is between *discrete* and *continuous* time, i.e., using either $\mathbb{N}$ or $\mathbb{R}_{\geq 0}$, respectively, as the set of labels

[1] In some languages, in particular the operational semantics of *Timed* CSP presented in [Sch95], while still keeping separate time transitions, action transitions are also adorned with durations; such transitions can intuitively be interpreted as non-instantaneous computations. However, since these ‘prolonged’ transitions can simply be regarded as abbreviations, viz., as a (now instantaneous) action followed, or preceded, by a time transition of the appropriate duration (see Section 7.2.2.3), we do not know of any other calculus where this design decision used.

[2] Note that many calculi only use *sets* of numbers, while others *axiomatise* sets with appropriate structure, usually viewing time as a partial order or a monoid.

for time transitions. For qualitative time, the choice is between either using one single clock or multiple of them, corresponding to having one single label for time transitions or more. These two possibilities are known as *global* and *local* qualitative time, respectively, the terminology referring to the clock-intuition: global qualitative time represents having a single (and hence global) clock, while local qualitative time allows for multiple clocks in a more distributed system.

During the course of this thesis, in a sense to be made precise, it will become clear that discrete quantitative time and global qualitative time are essentially the same.

There are also design decisions on the level of models and languages, once a representation of time is chosen. A standard operator, in some form or other, in process calculi is the *action prefix* as used, e.g., in [Hoa85, Mil89], written as $\alpha.p$ for an action α and a process p; the intuitive semantics of such a prefixed process is that it has the capability of performing the action α and then behaves as p does. However, should the process $\alpha.p$ be able to let time pass or not? After all, the syntax only contains information about action behaviour. If not, i.e., if $\alpha.p$ cannot idle at all, one speaks of *insistent prefixes*; alternatively, the prefix can let an *arbitrary*[3] amount of time pass, leading to so-called *relaxed prefixes*.

More abstractly, this last choice is concerned with *time stop mechanisms*, i.e., prohibiting the passage of time by means of not allowing any time transitions. It is certainly debatable whether such time stops are realistic; however, they can be used effectively for detecting mistakes in system specifications: it is very desirable that systems always allow progress of time—how could a specification with time stops ever be implemented? The previously described insistent prefixes allow control over such time stops on a very fine level of granularity: unless specified otherwise, all action prefixes will cause time to stop.

When using relaxed prefixes, a different means of stopping time is required. This is usually given in the form of the *maximal progress assumption* [HdR89], sometimes also known as *urgency* [NS91], or the *synchrony hypothesis* [BG92a]: this assumption states that communications have to be performed as soon as they are possible, without any delays; we shall give a more concrete description when describing individual

[3] We are not aware of any languages where a choice different from these two extremes is taken.

languages. The most important point about this assumption is that it provides another means of prohibiting the passage of time, complementing the use of relaxed prefixes.

2.3.2 Some Languages

There are three main non-timed process calculi: CCS [Mil89], CSP [Hoa85], and ACP [BK84]. Based on these three main strands, there are then a number of timed extensions, some of which we are now going to describe in some more detail.

As for timed extensions of CCS, an important specimen is *Temporal* CCS [MT90] (abbreviated here as TeCCS). Some of its SOS rules are presented in Figure 2.1 (the operators we omitted, renaming and restriction, behave in the evident way; we also slightly adapted the rules of weak choice, replacing the *maximum delay* predicate of [MT90] with negative premises); note that we follow [MT90] in using slightly differing notations for the two kinds of transitions. Time is modelled by natural numbers, i.e., discrete quantitative time (actually, only non-zero times are allowed), insistent prefixes are deployed, and the maximal progress assumption is not adopted. The action rules on the right-hand side of Figure 2.1 are completely standard; we also adopt, from CCS, that standard notational convention of denoting a general action $\neq \tau$, a so-called *observable*, or *controllable* [Wan90], action, by a and its *co-action* by $\bar{a}$, while an arbitrary action, which could as well be the *silent action* τ of CCS, is denoted by α (i.e., Roman vs. Greek letters).

In addition to the usual action prefix $\alpha.p$, there is the *delay* prefix $\delta.p$: intuitively, $\delta.p$ can idle forever in its initial state, while still being able to perform any action transition that p can perform. Hence, $\delta.\alpha.p$ is a relaxed prefix; however, since delay prefixes are *dynamic* with respect to action transitions, any consecutive action prefixes will again be insistent. Furthermore, the *time prefix* $(t).p$, where $t \in \mathbb{N} \setminus \{0\}$, delays all actions of p by t units of time, and during this enforced initial waiting period, *no* action transitions can be performed (the operator has no action rules).

Moreover, on top of the standard non-deterministic choice $+$, there is a second, so-called *weak* choice $\oplus$ present. The action rules of the two choices are identical, the difference lies solely in their time transitions: where $+$ demands *synchronous* progress through time (both components must be able to participate), $\oplus$ allows to discard one

$$
\begin{array}{cc}
\dfrac{}{\alpha.p \xrightarrow{\alpha} p} & \dfrac{}{\delta.p \overset{t}{\leadsto} \delta.p} \\[2ex]
\dfrac{p \xrightarrow{\alpha} p'}{\delta.p \xrightarrow{\alpha} p'} & \dfrac{}{(s+t).p \overset{s}{\leadsto} (t).p} \\[2ex]
\dfrac{p \xrightarrow{\alpha} p'}{p+q \xrightarrow{\alpha} p'} & \dfrac{}{(t).p \overset{t}{\leadsto} p} \\[2ex]
\dfrac{q \xrightarrow{\alpha} q'}{p+q \xrightarrow{\alpha} q'} & \dfrac{p \overset{s}{\leadsto} p'}{(t).p \overset{s+t}{\leadsto} p'} \\[2ex]
\dfrac{p \xrightarrow{\alpha} p'}{p \oplus q \xrightarrow{\alpha} p'} & \dfrac{p \overset{t}{\leadsto} p',\ q \overset{t}{\leadsto} q'}{p+q \overset{t}{\leadsto} p'+q'} \\[2ex]
\dfrac{q \xrightarrow{\alpha} q'}{p \oplus q \xrightarrow{\alpha} q'} & \dfrac{p \overset{t}{\leadsto} p',\ q \overset{t}{\leadsto} q'}{p \oplus q \overset{t}{\leadsto} p' \oplus q'} \\[2ex]
\dfrac{p \xrightarrow{\alpha} p'}{p|q \xrightarrow{\alpha} p'|q} & \dfrac{p \overset{t}{\leadsto} p',\ q \overset{t}{\not\leadsto}}{p \oplus q \overset{t}{\leadsto} p'} \\[2ex]
\dfrac{q \xrightarrow{\alpha} q'}{p|q \xrightarrow{\alpha} p|q'} & \dfrac{q \overset{t}{\leadsto} q',\ p \overset{t}{\not\leadsto}}{p \oplus q \overset{t}{\leadsto} q'} \\[2ex]
\dfrac{p \xrightarrow{a} p',\ q \xrightarrow{\bar{a}} q'}{p|q \xrightarrow{\tau} p'|q'} & \dfrac{p \overset{t}{\leadsto} p',\ q \overset{t}{\leadsto} q'}{p|q \overset{t}{\leadsto} p'|q'}
\end{array}
$$

Figure 2.1: The SOS rules of TeCCS.

of the components if it cannot let time pass (intuitively, if it cannot wait long enough); in case both processes can let time pass, the two operators behave in the same way.

Finally, parallel composition is again standard with respect to action transitions, allowing asynchronous behaviour as well as handshake synchronisation resulting in τ-actions; however, it enforces synchronous behaviour with respect to time transitions, exactly like $+$. In the remainder of this thesis, TeCCS will be our principle example of a timed process calculus.

Another, also CCS-based, language is *Timed CCS* [Wan90], which will be referred to as TiCCS in order to distinguish it from TeCCS, is quite different, as can be seen when inspecting (the part of) the time rules[4] depicted in Figure 2.2.

$$\frac{}{\mathrm{nil} \overset{t}{\leadsto} \mathrm{nil}}$$

$$\frac{}{a.p \overset{t}{\leadsto} a.p}$$

$$\frac{p \overset{t}{\leadsto} p',\ q \overset{t}{\leadsto} q'}{p|q \overset{t}{\leadsto} p'|q'}\ \mathrm{sort}_t(p) \cap \overline{\mathrm{sort}_t(q)} = \emptyset$$

Figure 2.2: Some time rules of TiCCS.

Firstly, a more abstract view of time is used, allowing both discrete and continuous quantitative time. Secondly, the maximal progress assumption is adopted, in combination with relaxed prefixes (as far as possible). The assumption states that communications cannot be delayed; since, in CCS-derivatives, communication is modelled by $\overset{\tau}{\rightarrow}$-transitions, this means that the following axiom must hold:

(Maximal Progress) $$(\forall p).\ (p \overset{\tau}{\rightarrow}\ \Rightarrow\ (\forall t).\ p \overset{t}{\not\leadsto})$$

where p ranges over all processes and t over all non-zero[5] times. This has two consequences, corresponding to the two possibilities how $\overset{\tau}{\rightarrow}$-transitions can occur: the

[4]The action rules are completely standard. Note that we changed the original notation for time transitions in [Wan90] in order to use the same as for TeCCS, as well as using nil, instead of 0, to denote the inactive, or deadlocked, process from CCS.

[5]Non-zero because $\overset{0}{\leadsto}$-transitions are not always permitted, and (if present) treated in a somewhat special way—see Section 3.2.1.

action prefix $\tau.p$ has no time transitions, as opposed to the other relaxed prefixes (consequently, only for prefixes of the form $a.p$ is it possible to derive time transitions from the above rules); secondly, and more importantly, the time rule of parallel composition has a side condition. This side condition uses the *timed sort* of p within t units of time, written as $\mathrm{sort}_t(p)$. Omitting its formal definition (by structural induction), $\mathrm{sort}_t(p)$ intuitively contains all the *observable* (i.e., $\neq \tau$) actions that p can perform within the next t units of time; hence, the side condition simply expresses that p and q cannot synchronise within the next t time units. Moreover, since the premises of that rule imply that neither p nor q on their own can perform a τ-transitions, all of this together states that $p|q$ cannot perform a $\xrightarrow{\tau}$-transition within the next t units of time: hence, it is possible to allow a $\overset{t}{\rightsquigarrow}$-transition without violating (Maximal Progress).

The language TPL [HR95] is very similar to TiCCS: it is also based on CCS and uses relaxed prefixes in combination with maximal progress. However, time is modelled by a global qualitative notion, using special $\overset{\sigma}{\rightsquigarrow}$-transitions (again slightly changing the original notation). Since, in this approach, just the *next* clock cycle needs to be considered, the rule for parallel composition becomes the following rule

$$\frac{p \overset{\sigma}{\rightsquigarrow} p',\ q \overset{\sigma}{\rightsquigarrow} q'}{p|q \overset{\sigma}{\rightsquigarrow} p'|q'}\ p|q \overset{\tau}{\not\rightarrow}$$

which makes the intuition behind the previous side condition very explicit: p and q, when running in parallel, must not be able to perform a $\xrightarrow{\tau}$-transition, no matter whether it stems from a communication or is simply an internal action of one of the two components. The most interesting feature of TPL, as far as the language is concerned, is its *time-out* operator, as illustrated by the following rule.

$$\frac{p \overset{\tau}{\not\rightarrow}}{\lfloor p \rfloor(q) \overset{\sigma}{\rightsquigarrow} q}$$

Here, control switches from p to q by means of a tick of the global clock (represented by) σ, in case p cannot perform a $\xrightarrow{\tau}$-transition: according to the suitable adaptation of (Maximal Progress) for qualitative time, this communication would have to be performed immediately and, consequently, could not be *preempted* by the time-out.

The time-out operator of TPL was actually adapted from the language ATP [NS94] where the semantics of time-out was defined without reference to $\xrightarrow{\tau}$-transitions of p:

$$\frac{}{\lfloor p \rfloor (q) \overset{\sigma}{\rightsquigarrow} q} \tag{2.2}$$

Since ATP is based on ACP with its different synchronisation mechanism, in particular without using τ's to denote the outcome of (handshake) communications, ATP does not adopt the maximal progress assumption (hence also the different semantics of the time-out) and uses insistent prefixes. Therefore, it is closer in spirit to TeCCS than to TiCCS or TPL; however, in contrast to TeCCS, ATP, as presented in [NS94] is based on global qualitative time, modelling the tick of the global clock by $\overset{\chi}{\rightsquigarrow}$-transitions, much like in TPL.

The paper [NSY93] presents a generalisation of ATP called ATP_D, where D is a *time domain*, i.e., a special kind of monoid which models time. Thus, a step towards using quantitative time is taken, without forcing a choice between discrete or continuous time. Again, the maximal progress assumption is not adopted, while both insistent and relaxed prefixes are used (incorporated by actions with different decorations); consequently, also two nil-prefixes are used: one as in TeCCS (which cannot let time pass), the other one as in TiCCS (which can wait arbitrarily long).

The last language we would like to describe in some detail is PMC [AM94] because it is one of two languages we are aware of (the other being CSA [CLM97]) which deal with local qualitative time. The language extends CCS, and the basic setting additionally consists of a finite set C of (abstract) *clocks*, which denote the different local 'time lines,' but also can be interpreted as actual clocks in hardware systems; maximal progress is not adopted, and insistent prefixes are used. In addition to that, suitably generalised version of the time-out operators of ATP and TPL are used. Since more than one clock can tick, the time rules for this operator now look as follows:

$$\frac{}{\lfloor p \rfloor \sigma(q) \overset{\sigma}{\rightsquigarrow} q} \qquad\qquad \frac{p \overset{\sigma'}{\rightsquigarrow} p'}{\lfloor p \rfloor \sigma(q) \overset{\sigma'}{\rightsquigarrow} p'}\ \sigma \neq \sigma'$$

i.e., clocks different from the time-out clock σ do not cause switching control. Additionally, PMC introduces the *clock ignore* operator $p \uparrow \sigma$ with operational semantics[6]

$$\frac{}{p \uparrow \sigma \stackrel{\sigma}{\rightsquigarrow} p \uparrow \sigma} \qquad\qquad \frac{p \stackrel{\sigma'}{\rightsquigarrow} p'}{p \uparrow \sigma \stackrel{\sigma'}{\rightsquigarrow} p' \uparrow \sigma}\ \sigma \neq \sigma'$$

Intuitively, the ignore operator adds a σ-loop to every state reachable from p, *overriding* any previous $\stackrel{\sigma}{\rightsquigarrow}$-transitions. This is interpreted as p no longer being in the *scope* of σ: ignoring clocks *localises* them, although actually by means of disconnecting[7] processes from clocks; in this pristine setting, every process is assumed to be connected to every clock. Disregarding the differences due to the different base languages, PMC could be considered a *multidimensional*[8] version of ATP.

Other calculi worth mentioning are:

- The operational semantics for TiCSP [SDJ+91, DS95] presented in [Sch95], which uses continuous quantitative time in combination with relaxed prefixes; also the maximal progress assumption is adopted; due to the chosen base algebra CSP, also a denotational semantics (using *timed failures*) is presented. The semantics also adopts the separation approach, however in a somewhat 'exotic' fashion: in addition to time transitions, the calculus uses action transitions labelled with *pairs* $\langle \alpha, t \rangle$ consisting of an action α and a time t; as already briefly remarked before, this corresponds to dropping the instantaneous requirement for computations.

- Other timed extensions of ACP as described, e.g., in [BB91, BM01]: the former uses continuous quantitative time, albeit in an *absolute* way: processes are paired up with points in time, and action prefixes are decorated with *time stamps* denoting the exact instant when they have to be performed; this is very different from the *relative* approaches encountered so far where time transitions denote durations. The later version presents both discrete and continuous quantitative

[6] Its action transitions are simply the ones of p, however, the ignore is static, i.e., it is propagated to the successor states.

[7] This might seem rather unintuitive; however, replacing the ignore by its dual, the intuitively more appealing *(clock) attach* operator in [Kic99] (carried out for the calculus CSA which is very much influenced by PMC) made the whole theory much more complicated, so, at least for technical reasons, the ignore operators seems the more reasonable choice.

[8] This will be made more precise by later results.

time, in both absolute and relative settings, of ACP; the absolute version is similar to the one presented in [BB91], while the relative variant is similar to TeCCS (insistent prefixes, no maximal progress).

Note that we only treat calculi *without binding*, so-called *first-order* languages, so we cannot deal with timed variants of the π-calculus [MPW92], e.g., as presented in [RS02], or timed process calculi with binding like [Wan91, Che93]. Since our approach to operational semantics is based on categorical methods, in particular on the framework developed in [TP97], treating such calculi would necessitate combining the treatment of (first-order) timed processes developed in the following with previous work on categorical versions of both (abstract) syntax and operational semantics for languages with binding—see [FPT99, FT01, GP02].

2.4 Category Theoretic Preliminaries

Apart from the basic notions like *categories*, *functors*, and *natural transformations*, as well as standard *algebras* and *coalgebras* for functors, we assume familiarity with the following categorical notions (unless stated otherwise, see [Mac97] for definitions). We denote the category of sets and (total) functions by **Set**.

(Co)limits. In addition to the standard definitions, we also assume knowledge about *weak* (co)limits, and the specific cases of products, (weak) pullbacks, and coproducts, initial and terminal objects, as well as *preservation* of such (weak) (co)limits. The notion of *filtered colimit* might be less familiar, so here is a quick reminder. A category is *filtered* if any finite diagram in it has at least one cocone over it; a *filtered* colimit is the colimit of a functor $F : \mathcal{C} \to \mathcal{D}$ where $\mathcal{C}$ is a filtered category. These two notions can then be generalised, from finite diagrams to diagrams of size strictly less then a cardinal κ, leading to κ-*filtered* categories and colimits; the original definitions are obtained by setting $\kappa = \aleph_0 = |\mathbb{N}|$.

Adjunctions. Given two functors F and G in opposite directions, we denote the fact that F is the left adjoint of G (as usual) by $F \dashv G$. Furthermore, from [Tur96], we adopt the notation $(\cdot)^\flat$ and $(\cdot)^\sharp$ for the natural bijection of hom-sets ([Mac97] uses the notation φ and φ^{-1}, respectively). Concretely, $F \dashv G$ holds if and only if we have two assignments

$$(f : FX \to Y) \overset{(\cdot)^\flat}{\longmapsto} (f^\flat : X \to GY) \quad \text{and} \quad (g : X \to GY) \overset{(\cdot)^\sharp}{\longmapsto} (g^\sharp : FX \to Y)$$

which are mutually inverse and natural in X and Y (in the sense that the bijection is compatible with pre- and post-composition of arrows).

(Co)monads. We assume some familiarity with the definitions of (co)monads and the corresponding Eilenberg-Moore constructions, also that every adjunction $F \dashv G$ gives rise to a both monad GF and a comonad FG. Even so, as a reference point for later on, we now briefly state some of the basic definitions.

A comonad $D = \langle D, \varepsilon, \delta \rangle$ is an endofunctor $D : \mathcal{C} \to \mathcal{C}$, together with natural transformations $\varepsilon : D \Rightarrow \mathrm{Id}$ and $\delta : D \Rightarrow D^2$, its *counit* and *comultiplication*, respectively, such that the following diagrams commute:

$$\begin{tikzcd} & D \arrow[dl, Rightarrow, "\mathrm{Id}"'] \arrow[d, Rightarrow, "\delta"'] \arrow[dr, Rightarrow, "\mathrm{Id}"] & \\ D & D^2 \arrow[l, Rightarrow, "\varepsilon_D"] \arrow[r, Rightarrow, "D\varepsilon"'] & D \end{tikzcd} \qquad \begin{tikzcd} D \arrow[r, Rightarrow, "\delta"] \arrow[d, Rightarrow, "\delta"'] & D^2 \arrow[d, Rightarrow, "\delta_D"] \\ D^2 \arrow[r, Rightarrow, "D\delta"'] & D^3 \end{tikzcd} \tag{2.3}$$

Given two comonads $D = \langle D, \varepsilon, \delta \rangle$ and $D' = \langle D', \varepsilon', \delta' \rangle$ on the same category $\mathcal{C}$, a *comonad morphism* from D to D' is a natural transformation $\varphi : D \Rightarrow D'$ such that following diagrams commute:

$$\begin{tikzcd} D \arrow[rr, Rightarrow, "\varphi"] \arrow[dr, Rightarrow, "\varepsilon"'] & & D' \arrow[dl, Rightarrow, "\varepsilon'"] \\ & \mathrm{Id} & \end{tikzcd} \qquad \begin{tikzcd} D \arrow[r, Rightarrow, "\varphi"] \arrow[d, Rightarrow, "\delta"'] & D' \arrow[d, Rightarrow, "\delta'"] \\ DD \arrow[r, Rightarrow, "\varphi\varphi"] & D'D' \end{tikzcd} \tag{2.4}$$

where the map $\varphi\varphi$ is given by one of the equal (because of naturality!) composites $DD \overset{D\varphi}{\Longrightarrow} DD' \overset{\varphi_{D'}}{\Longrightarrow} D'D'$ or $DD \overset{\varphi_D}{\Longrightarrow} D'D \overset{D'\varphi}{\Longrightarrow} D'D'$. With such comonad morphisms as arrows, the comonads on $\mathcal{C}$ form a category, denoted by $\mathbf{Cmd}(\mathcal{C})$.

Given a comonad D on $\mathcal{C}$, an *Eilenberg-Moore coalgebra* for D on an object X of

$\mathcal{C}$ is a morphism $X \to DX$ in $\mathcal{C}$ such that the following diagrams commute:

$$\begin{tikzcd} & X \arrow[r, "k"] \arrow[dl, "\mathrm{id}_X"'] \arrow[d, "k"'] & DX \arrow[d, "Dk"] \\ X & DX \arrow[l, "\varepsilon_X"] \arrow[r, "\delta_X"'] & D^2X \end{tikzcd} \tag{2.5}$$

Homomorphisms of Eilenberg-Moore coalgebras for D are the same as used for coalgebras of functors, viz., morphisms $f : X_1 \to X_2$ between the carriers of two coalgebras k_1 and k_2 satisfying

$$\begin{tikzcd} X_1 \arrow[r, "f"] \arrow[d, "k_1"'] & X_2 \arrow[d, "k_2"] \\ DX_1 \arrow[r, "Df"'] & DX_2 \end{tikzcd} \tag{2.6}$$

There is the evident category of Eilenberg-Moore coalgebras of D which, following [JPT$^+$01], we shall denote by D**–Coalg**, in order to distinguish it from the category D**–coalg** of 'mere' coalgebras for D (regarded simply as an *endofunctor*); analogously, given a monad T, its category of Eilenberg-Moore algebras is denoted by T**–Alg**, as opposed to T**–alg** for the unconstrained algebras. We also use the distinctions algebra/Algebra and coalgebra/Coalgebra in plain text: e.g., a D-Coalgebra $k : X \to DX$ in fact denotes an Eilenberg-Moore coalgebra for the comonad D.

Transition systems and bisimulations, coalgebraically. As seen in Section 2.2, an LTS is a tuple $\langle S, L, \to \rangle$ such that $\to \subseteq S \times L \times S$. Aczel [Acz88] showed that LTSs are also the same as coalgebras for $\mathcal{P}(L \times \_)$, where $\mathcal{P}(\_)$ denotes the *power set* functor on **Set**. This has since been generalised: given an endofunctor B on some category $\mathcal{C}$, the B-coalgebras intuitively correspond to *transition systems of type* B; in particular, for B_A as defined in (1.1), the B_A-coalgebras are the same as *image finite* LTSs with labels in A.

There is an accompanying coalgebraic notion of bisimulation [AM89], which was slightly generalised in [TP97]. Let B be an endofunctor on a category $\mathcal{C}$, and let $k_i : X_i \to BX_i$ be two B-coalgebras. Then a *coalgebraic B-bisimulation* is a B-coalgebra

$r : R \to BR$ such that there is a *span* of B-coalgebras

$$\begin{tikzcd}
 & X_1 \arrow[rr, "k_1"] & & BX_1 \\
R \arrow[ur, "p"] \arrow[rr, "r"] \arrow[dr, "q"'] & & BR \arrow[ur, "Bp"'] \arrow[dr, "Bq"] & \\
 & X_1 \arrow[rr, "k_2"'] & & BX_1
\end{tikzcd}$$

Note that R and r are, in general, by no means uniquely determined; instantiating this for B_A as in (1.1) precisely yields strong bisimulation (see, e.g., [TP97]). This notion can be adapted to Coalgebras, demanding that k_1, k_2, *and also* r are Coalgebras.

(Co)free (co)monads. Given an endofunctor H on some category $\mathcal{C}$, there is the obvious *forgetful functor* $U : H\textbf{–coalg} \to \mathcal{C}$, sending an H-coalgebra $X \to HX$ to its *carrier* X. If U has a right adjoint $R : \mathcal{C} \to H\textbf{–coalg}$, the induced comonad UR is called the *cofree comonad on* H, denoted by H^∞; dually, if the analogous forgetful functor $U : H\textbf{–alg} \to \mathcal{C}$ has a left adjoint L, the induced monad UL is called the *free monad on* H, denoted by $H^\star$.

The most important property of cofree comonads is that $H^\infty\textbf{–Coalg} \cong H\textbf{–coalg}$; consequently, any H-coalgebra $k : X \to HX$ corresponds to a unique H^∞-Coalgebra $k^\infty : X \to H^\infty X$, its so-called *coinductive extension*; dually, $H^\star\textbf{–Alg} \cong H\textbf{–alg}$, and for an H-algebra $h : HX \to X$, there is its unique *inductive extension* $h^\star : H^\star X \to X$ to an $H^\star$-Algebra.

Provided H^∞ exists and $\mathcal{C}$ has products, there is a more concrete description of the cofree comonad: $H^\infty X$ is the *final coalgebra* of the functor $X \times H$. Consequently, by *Lambek's Lemma* [SP82], we obtain $H^\infty X \cong X \times HH^\infty X$, yielding the two *projections* $X \xleftarrow{\mathrm{fst}_X} H^\infty X \xrightarrow{\mathrm{snd}_X} HH^\infty X$. The comonad structure on H^∞ is derived by extending the map $X \mapsto H^\infty X$ first to a functor and then to a comonad H^∞ in $\mathbf{Cmd}(\mathcal{C})$. Explicitly (for proofs, see, e.g., [Tur96, §7]), given $f : X \to Y$, define $H^\infty f$ as the map making the following diagram commute:

(2.7)
$$\begin{tikzcd}
X \arrow[d, "f"'] & H^\infty X \arrow[l, "\mathrm{fst}_X"'] \arrow[r, "\mathrm{snd}_X"] \arrow[d, dashed, "H^\infty f"'] & HH^\infty X \arrow[d, "HH^\infty f"] \\
Y & H^\infty Y \arrow[l, "\mathrm{fst}_Y"] \arrow[r, "\mathrm{snd}_Y"'] & HH^\infty Y
\end{tikzcd}$$

This is well-defined by finality of $H^\infty Y$, which also implies functoriality. The counit ε_X is given by $\mathrm{fst}_X : H^\infty X \to X$, and the components of the comultiplication δ are obtained by using the universal property of $(H^\infty)^2 X$:

$$\begin{tikzcd} & H^\infty X \arrow[r, "\mathrm{snd}_X"] \arrow[dl, "\mathrm{id}_{H^\infty X}"'] \arrow[d, dashed, "\delta_X"] & HH^\infty X \arrow[d, "H\delta_X"] \\ H^\infty X & (H^\infty)^2 X \arrow[l, "\mathrm{fst}_{H^\infty X}"] \arrow[r, "\mathrm{snd}_{H^\infty X}"'] & H(H^\infty)^2 X \end{tikzcd} \tag{2.8}$$

Dually, if $\mathcal{C}$ has coproducts and $H^\star$ exists, $H^\star X$ is the initial $(X+H)$-algebra. Such free monads arise, for instance, in *universal algebra* [BS81]: given a *signature* Σ, there is a *polynomial* functor [RB85] (of the same name) such that the Σ-algebras (in the categorical sense) are precisely the Σ-*interpretations* from universal algebra. Writing $T \stackrel{\mathrm{df}}{=} \Sigma^\star$, TX is simply the set of all Σ-*terms* with variables in X; in particular, if X is the empty set $\emptyset$ (the initial object in **Set**), $T\emptyset$ is the set of all *closed*, i.e., variable-free, terms, which is thus established as the initial Σ-algebra (since $\emptyset + \Sigma \cong \Sigma$); these considerations form the basis of *initial algebra semantics* [GTW78].

Accessibility. Given a *regular cardinal* κ (i.e., a cardinal such that the cardinality of any union of less than κ many sets of cardinality less than κ is still less than κ, e.g., $\aleph_0$), recall from [AR94, MP89] the notion of a κ-*accessible* category $\mathcal{C}$: essentially, this means that $\mathcal{C}$ has all κ-filtered colimits, and that these special colimits are, in a precise sense, 'enough' to construct *all* other objects of $\mathcal{C}$; $\mathcal{C}$ is *locally presentable* if and only if it is cocomplete and accessible, see [JPT$^+$01]; for instance, **Set** is locally presentable.

Let $\mathcal{C}$ be an accessible category, F be an endofunctor on $\mathcal{C}$, and let κ be a regular cardinal. Then F has *rank* κ, or is κ-*accessible*, if it preserves λ-filtered colimits for any $\lambda < \kappa$; if a functor F preserves $\aleph_0$-filtered colimits, i.e., all finite filtered colimits, F is usually called *finitary* [Lin66, JPT$^+$98, Rob02].

If κ, λ are two cardinals such that $\lambda > \kappa$, then, by definition, any λ-filtered category $\mathcal{C}$ is automatically also κ-filtered. Hence, if a functor $F : \mathcal{C} \to \mathcal{C}$ preserves κ-filtered colimits (i.e., has rank κ), F in particular preserves λ-filtered colimits (thus also has rank λ). Moreover, endofunctors of a certain rank form a category: the composition of two functors with rank κ also has rank κ.

On the structure of categories of coalgebras. The following results are either taken directly from [JPT+01] or are (easy!) consequences of these results:

- Any accessible endofunctor on a locally presentable category generates a cofree comonad; in particular, the cofree comonad B_A^∞ on B_A exists.
- If the category $\mathcal{C}$ has pullbacks and the endofunctor $H : \mathcal{C} \to \mathcal{C}$ preserves weak pullbacks, the forgetful functor $U : H\text{–}\mathbf{coalg} \to \mathcal{C}$ also preserves weak pullbacks.
- If B preserves weak pullbacks and generates a cofree comonad, B^∞ also preserves weak pullbacks.

Distributive laws. Given endofunctors F and G on a category $\mathcal{C}$, a *distributive law of F over G* is a natural transformation $\ell : FG \Rightarrow GF$; note that in [TP97], the letter λ is used to denote distributive laws, however, since we are frequently using λ-notation for describing functions, we use this slightly non-standard notation. This notion can be adapted appropriately for a suitable combination of functor(s) and (co)monad(s), simply demanding that ℓ satisfies the evident diagrams relating it to the respective (co)monad structure(s); for a systematic approach to distributive laws, see [PW99, LPW00].

For instance, a distributive law of a monad $\langle T, \eta, \mu \rangle$ over a comonad $\langle D, \varepsilon, \delta \rangle$ is a natural transformation $\ell : TD \Rightarrow DT$ satisfying the following diagrams:

(2.9)
$$
\begin{tikzcd}
 & D \arrow[dl, Rightarrow, "\eta_D"'] \arrow[dr, Rightarrow, "D\eta"] & \\
TD \arrow[rr, Rightarrow, "\ell"] \arrow[dr, Rightarrow, "T\varepsilon"'] & & DT \arrow[dl, Rightarrow, "\varepsilon_T"] \\
 & T &
\end{tikzcd}
\qquad
\begin{tikzcd}
T^2D \arrow[r, Rightarrow, "T\ell"] \arrow[d, Rightarrow, "\mu_D"'] & TDT \arrow[r, Rightarrow, "\ell_T"] & DT^2 \arrow[d, Rightarrow, "D\mu"] \\
TD \arrow[rr, Rightarrow, "\ell"] \arrow[d, Rightarrow, "T\delta"'] & & DT \arrow[d, Rightarrow, "\delta_T"] \\
TD^2 \arrow[r, Rightarrow, "\ell_D"'] & DTD \arrow[r, Rightarrow, "D\ell"'] & D^2T
\end{tikzcd}
$$

Following [TP97, Bar02], *ℓ-bialgebras* for a distributive law $\ell : FG \Rightarrow GF$ are structures $FX \stackrel{h}{\to} X \stackrel{k}{\to} GX$ such that the following diagram commutes:

(2.10)
$$
\begin{tikzcd}
FX \arrow[r, "h"] \arrow[d, "Fk"'] & X \arrow[r, "k"] & GX \\
FGX \arrow[rr, "\ell_X"'] & & GFX \arrow[u, "Gh"']
\end{tikzcd}
$$

Homomorphisms of such ℓ-bialgebras are the evident ones, allowing to form the category ℓ**–Bialg** of ℓ-bialgebras; again, there are the obvious variations for (co)monads and their respective (Co)Algebras.

Liftings. Given two an endofunctors B and H on a category $\mathcal{C}$, a *lifting* of H to B**–coalg** is an endofunctor $\tilde{H}$ on B**–coalg** such that the following diagram commutes:

$$\begin{tikzcd} B\text{–}\mathbf{coalg} \arrow[r, "\tilde{H}"] \arrow[d, "U"'] & B\text{–}\mathbf{coalg} \arrow[d, "U"] \\ \mathcal{C} \arrow[r, "H"'] & \mathcal{C} \end{tikzcd}$$

where U denotes the forgetful functor. More concretely, the square demands that $\tilde{H}$ transforms a B-coalgebra with carrier X into a B-coalgebra with carrier HX; adapting this to liftings of a comonad to B**–coalg** yields a comonad $\tilde{D}$ on B**–coalg** such that both ε and δ are also homomorphisms of B-coalgebras, i.e., the comonad structure of D also lifts to B**–coalg**, cf. [TP97, Rem. 5.1]; both concepts can be dualised in order to obtain liftings to algebras/Algebras.

Distributive laws induce liftings. For a distributive law $\ell : FG \Rightarrow GF$, one obtains

- a lifting $\tilde{G}$ of G to F**–alg**: given $FX \stackrel{h}{\to} X$, we get $FGX \stackrel{\ell_X}{\to} GFX \stackrel{Gh}{\to} GX$
- a lifting $\tilde{F}$ of F to G**–coalg**: given $X \stackrel{k}{\to} GX$, we get $FX \stackrel{Fk}{\to} FGX \stackrel{\ell_X}{\to} GFX$

Again, there are suitable variations: liftings of functors to (Co)Algebras, or liftings of (co)monads to (co)algebras, requiring that either the distributive law respects the additional structure, or that the operations of the (co)monad lift, respectively.

In general, the existence of a lifting does not guarantee the existence of a distributive law. However, when lifting to categories of (Co)Algebras, the converse also holds [BW85, Jac94]; in particular, there is an *equivalence* between distributive laws $TD \Rightarrow DT$ of a monad T over a comonad D, liftings of the monad to D**–Coalg**, and liftings of the comonad to T**–Alg**—see [TP97, Thm. 7.1].

Distributive laws induce composite (co)monads. Given two comonads D and D' on a category $\mathcal{C}$, the composite functor DD' does, in general, not carry a comonad structure. However, if we have a distributive law of comonads $\ell : DD' \Rightarrow D'D$, there is a canonical comonad structure on DD' with counit $\varepsilon\varepsilon'$ and comultiplication defined as

$$DD' \xRightarrow{\delta\delta'} DDD'D' \xRightarrow{D\ell_{D'}} DD'DD'$$

Simultaneously, the distributive law ℓ also induces a lifting $\tilde{D}$ of D to D'**–Coalg**, as sketched above. In this situation, we have DD'**–Coalg** $\cong \tilde{D}$**–Coalg**; the dual result holds for monads—see [Jac94].

Chapter 3

A Mathematical Model of Timed Processes

This chapter presents the semantic basis underlying the languages for timed processes presented in the previous chapter: timed processes are described by *timed transition systems* (TTSs), i.e., restricted LTSs where transitions are labelled by elements of a *time domain* [JSV93], a special kind of monoid representing time. The restrictions imposed on the transition relations of TTSs express (widely accepted) properties of time passing related to the monoid structure of time. We also present *time bisimulation*, which is strong bisimulation between TTSs, as an appropriate notion of equivalence between TTSs. After exhibiting elementary descriptions of these notions, we show that *partial actions* of the monoid of time are the same as TTSs, which also results in a new characterisation of time bisimulation.

Note that this chapter exclusively focuses on time transitions; action transitions are not considered for the moment. However, one should keep in mind that there is the 'equation'

$$\text{processes} \;=\; \text{LTS} \;+\; \text{TTS}$$

where the LTS describes the action transitions modelling (instantaneous) computations while the TTS, as introduced in the following, represents the passage of time without external interactions by time transitions.

Further stressing the importance of the monoid structure on time, *total* monoid actions also play an important role in the form of *delay operators*, used to model *postponing* computations of timed processes. Moreover, the two concepts of time transitions and delay operators are combined in *biactions*: a set of processes endowed with a partial monoid action describing the idling capabilities of processes, together with a total monoid action representing a delay operator, and with the two actions suitably related. As such, these biactions could be considered as a very primitive algebra of timed processes, incorporating two of the most important concepts associated with such processes. Note that this, again, only takes into account the timing behaviour of such processes; action transitions are not (yet) considered.

Parts of this chapter have, in condensed and less general form, previously appeared in [Kic02a].

3.1 Notions of Time as Monoids

In this section, we are going to present a suitable formalisation of time itself in the form of *time domains*: special *monoids* which can be partially ordered in a way compatible with monoid composition. The notion presented here, taken from [JSV93], is general enough to encompass all important examples used later on. It is also more general than other similar notions found in the literature on timed processes, as demonstrated by a brief discussion concluding this section.

Definition 3.1 ([JSV93])
A (*commutative*) *time domain* is a (resp. commutative) monoid $\mathcal{T} = \langle \mathcal{T}, +, 0\rangle$ which is *anti-symmetric*

(AS) $$(\forall t, u \in \mathcal{T}).\ t + u = 0 \Rightarrow t = u = 0$$

and *left-cancellative*

(LC) $$(\forall s, t, u \in \mathcal{T}).\ s + t = s + u \Rightarrow t = u$$

A *homomorphism of time domains* is simply a standard monoid homomorphism. ■

It is immediate to see that the (small) time domains form a category, which is a full subcategory of the category of (small) monoids. Homomorphisms may be thought of as ‘time transformations,’ i.e., switching from one ‘time line’ (in the widest sense) to another one. Note that we use the additive notation with $+$ and 0, yet time domains are not necessarily assumed to be commutative: the notation merely yields a better match with the naive understanding of time and its properties. It is easily seen that the two axioms (LC) and (AS) are independent, i.e., one does not imply the other (e.g., any non-trivial group satisfies (LC) but not (AS), and in the other direction, consider any non-trivial finite-join semilattice where (AS) but not (LC) holds).

Despite adopting the notion from [JSV93], we are still going to give a detailed presentation of the motivation underlying, and also of the technicalities involved in, time domains since the literature on the subject is rather sparse, as regards intuition. After all, time is at the very heart of our considerations, and so its model should be presented with all due care.

There are at least two interpretations of the elements of a time domain $\mathcal{T}$, corresponding to different aspects of time. The first one is that $t \in \mathcal{T}$ represents an *absolute* point in time, e.g., 12:13 on July 31st 2002. Alternatively, from the point of view of *durations*, $t \in \mathcal{T}$ describes the amount of time it takes for an event to occur, *relative* to the current position in time. These two interpretations are closely related: being at a point $t \in \mathcal{T}$ in time, the new position in time after an event with duration $u \in \mathcal{T}$ should be $t + u \in \mathcal{T}$ obtained by *combining* t with u according to a binary *composition* function $+$ on $\mathcal{T}$. Additionally, for $t, u \in \mathcal{T}$, the composition $t + u$ intuitively corresponds to the duration of two *consecutive* events with the respective durations t and u.

These considerations already motivate the presence of a composition function, or *addition*, $+$ on $\mathcal{T}$. Furthermore, recall the common assumption in the previously presented calculi to separate instantaneous computations from pure time passing. In order to be able to also associate a duration to such instantaneous calculations, there should be a distinguished ‘zero’ duration in $\mathcal{T}$, written as 0, expressing precisely the *absence* of duration that characterises an action as being instantaneous.

Furthermore, the addition $+$ should at least be *associative* in order to provide a sensible way of repeatedly combining durations. This is in accordance with intuition: when three (or more) events occur in sequence, the same total duration should be obtained regardless of how the intermediate durations were added up. Moreover, the zero duration 0 should act as the *neutral element* for $+$, capturing the intuition about representing no duration at all: the duration of an event is not affected by performing an instantaneous action immediately before or after it.

These considerations justify a monoid structure on $\mathcal{T}$. However, that alone would allow too many instances not easily seen to represent time; additional axioms have to be imposed to exclude such unrealistic situations. One particularly important property is that time can only *progress*: it is not possible to ‘undo’ actual temporal progress, marked by a non-zero duration. Axiom (AS) expresses this by stating that, if adding two durations $t, u \in \mathcal{T}$ yields 0, this is only possible in the trivial case when both t and u are actually equal to 0. Consequently, there cannot be ‘negative’ durations: moving ‘backward’ in time is prohibited. Note that this in particular excludes all groups as models for time. Furthermore, note that (AS) could have also been stated as an equivalence since $t = u = 0$ anyway implies $t + u = 0$.

This rather indirect way of expressing the mentioned progress property arises from an attempt to use as few primitives as possible. Commonly, in particular with the absolute interpretation of time, a time domain $\mathcal{T}$ is also assumed to come with an order structure, and progress is then formulated in order-theoretic terms, e.g., 0 being the least element, and for each time $t \in \mathcal{T}$, there exists a time $u \in \mathcal{T}$ which is strictly greater. Yet, as we shall see shortly, assuming only the monoid structure as primitive already allows to *derive* a very natural order on $\mathcal{T}$. Moreover, in all our examples, this derived order coincides with the corresponding standard order, and the order-theoretic version of progress indeed follows from the axioms imposed on the monoid structure.

Another important property of naive time is that measuring durations is unaffected by the *context* in which the measurement takes place. This is, in a basic fashion, expressed in (LC); the connection should become clearer when considering the following logically equivalent formulation of the axiom:

$$(\forall s, t, u \in \mathcal{T}).\ t \neq u \Rightarrow s + t \neq s + u$$

This states that inequalities of durations are *stable under translations*: two different durations $t, u \in \mathcal{T}$ are still different after adding an *initial offset* $s \in \mathcal{T}$: the fact that t and u are different is not altered by first waiting for the same initial period of s units of time. Note that this expresses the fact that for each $s \in \mathcal{T}$, the *translation by* s, i.e., the function $(\lambda t.s+t)$ on $\mathcal{T}$, is injective; thus, relative to s, durations are *invertible* (this will be shown formally later on).

A general property of a time domain $\mathcal{T}$ is whether it is *commutative* or not, i.e., whether it satisfies the familiar axiom $t+u=u+t$ for all $t,u\in\mathcal{T}$. Intuitively, commutativity means that first waiting for t units of time and then for u units yields the same amount of time when waiting for those durations in the other order, first u units and then t units. This is very a very natural property as long as one regards time as a 'straight line' where there are no 'side routes.' When this is not the case (and, as we shall see, this makes perfect sense), it might happen that waiting for t and u units, respectively, leads to different such 'side roads' which never join again (like different branches of a tree) and so, commutativity will not hold. Let us now consider several examples of time domains (both commutative and non-commutative ones).

Example 3.2

1. The singleton set $1=\{0\}$ is a commutative time domain, the *trivial domain*; this embeds the *un-timed* case where *everything* happens instantaneously.

2. The natural numbers $\mathbb{N}$, with $+$ and 0 as monoid composition and neutral element, respectively, form the commutative time domain of *discrete time*.

3. The non-negative real numbers $\mathbb{R}_{\geq 0}$, again with $+$ and 0, form the (also commutative) time domain of *continuous*, or *dense*, *time*; in the following, we will frequently write $\mathbb{R}$ instead of the more precise $\mathbb{R}_{\geq 0}$ when no confusion can arise. As it will turn out, timed processes over continuous time can have very complicated behaviour, the theory works better for discrete time.

4. If $\mathcal{T}_1$ and $\mathcal{T}_2$ are (commutative) time domains, their *product* $\mathcal{T}_1\times\mathcal{T}_2$ (regarded as a monoid), with component-wise operations, is also a (resp. commutative) time domain because (AS) and (LC) are inherited from the $\mathcal{T}_i$. This generalises to arbitrary products, so, in particular, the *powers* $\mathbb{N}^n$ and $\mathbb{R}^n$ of n-tuples of natural and non-negative real numbers, respectively, with the component-wise operations, are commutative time domains. As an application, consider processes consisting of several components with independent local 'time lines,' cf. the

(real-valued) clocks of *timed automata* [AD94]; note that $\mathbb{N}^n$ is the *free commutative monoid* on n generators.

5. For a finite set $C = \{c_1, \ldots, c_n\}$, recall the *free monoid* $C^\star$ on C: $C^\star$ is the set of finite words over the alphabet C; monoid composition is given by concatenation; its neutral element is the empty word ε. Then $C^\star$ is the (for $|C| > 1$ non-commutative) time domain of *local qualitative time*: the $c_i \in C$ correspond to (abstract) *clocks* as, e.g., in PMC [AM94], and 'local' refers to the fact that different clocks are *independent*. This is reflected by the freeness of $C^\star$: there is no relation between the clocks (not even commutativity), a word $w \in C^\star$ simply describes a sequence of ticks of clocks. Note that both the trivial domain and discrete time are particular instances of this case, the former for $C = \emptyset$, the latter for $|C| = 1$ (cf. the unary representation of natural numbers). $\Diamond$

There is the long-standing general question concerning *embeddability* of monoids into groups: a monoid M can be *embedded* into a group G if there exists an injective monoid homomorphism $M \hookrightarrow G$, making M (isomorphic to) a sub-monoid of G; in this situation, G is also known as an *extension group* of M. Embeddability is important because many calculations become much easier when being able to carry them out in a group. For time domains, the following can be said with respect to embeddability.

Remark 3.3
Because of (AS), a non-trivial time domain can itself never be a group. However, by [vW66, §13], any commutative monoid M can be embedded into a group if and only if it satisfies the (left) *cancellation rule*

$$(\forall m, m', m'').\ m + m' = m + m'' \Rightarrow m' = m''$$

This is exactly (LC), and so any commutative time domain can be embedded into a group: $\mathbb{N}$ into the integers[1] $\mathbb{Z}$, and $\mathbb{R}_{\geq 0}$ into the set of all real numbers; analogous results hold for products, in particular, the free commutative monoid $\mathbb{N}^n$ can be embedded into the *free commutative group* on n generators, $\mathbb{Z}^n$.

[1] Taking equivalence classes of pairs of natural numbers representing the same 'distance'; cancellation is needed to obtain a well-defined extension of $+$ to these equivalence classes.

Moreover, $C^\star$ can also be embedded, viz., into the *free group* $F(C)$ on C: with C^{-1} denoting the set of *formal inverses* of C and $\oplus$ the disjoint union of the two sets, $F(C)$ is given by the quotient of $(C \oplus C^{-1})^\star$ generated by the equalities $cc^{-1} = c^{-1}c = \varepsilon$. Hence, it is possible in $F(C)$ to *cancel out* adjacent pairs of inverses, e.g., $ccdd^{-1}c^{-1}$ is actually equal to $c\varepsilon = c$ in $F(C)$. $\Diamond$

For general non-commutative monoids, embeddability is a notoriously hard problem: see [Mal37, Mal39, Mal40] for some classic results and counterexamples. For arbitrary non-commutative time domains, *right cancellation* (the symmetric version of (LC)) does *not necessarily* hold; this is shown in the following counterexample, due to G. Plotkin. Consequently, because both left and right cancellation are necessary conditions for embeddability (they actually hold in any group, so certainly also in the extension group), an arbitrary time domain cannot be embedded into a group.

Example 3.4
Take the free monoid $S^\star$ on the three-element set $S \stackrel{\text{df}}{=} \{a, b, c\}$ and consider its quotient M induced by the equivalence relation generated by the equation

$$ac = bc \tag{3.1}$$

The quotient M carries a monoid structure inherited from the free monoid, and (AS) clearly holds; also, by construction, M does *not* satisfy right cancellation: $ac = bc$, yet $a \neq b$. However, M satisfies (LC); for this, it is convenient to regard M as obtained from $S^\star$ by the *rewrite system* (see [BN98, DP01]) consisting of the following *rewrite rule*, obtained by directing (3.1):

$$uacv \to ubcv$$

where $u, v \in S^\star$, i.e., any substring ac can be replaced by bc.

This rewrite system is *Church-Rosser*: for $u, v, v' \in S^\star$ such that $u \to v$, $u \to v'$, and $v \neq v'$, there is $w \in S^\star$ such that $v \to w$ and $v' \to w$. This follows because there must be (at least) two disjoint occurrences of ac in u: otherwise, $v \neq v'$ could not hold. Hence, there is still (at least) another occurrence of ac left in both v and v', respectively, which can then rewritten to obtain the same $w \in S^\star$.

Strong normalisation also holds: any word that can be rewritten must be of the form $w_1acw_2\dots w_{n-1}acw_n$, where none of the $w_i \in S^\star$ contains any occurrences of ac, and its *normal form* then is $w_1bcw_2\dots w_{n-1}bcw_n$. Write $\vdash v = w$ if $v, w \in S^\star$ have the same normal form; note that $\vdash v = w$ is equivalent to $v = w$ in M.

Suppose $\vdash uv = uw$, i.e., $uv = uw$ in M; without loss of generality, u, v, w are in normal form. Moreover, if both uv and uw are also in normal form, they are identical and so $v = w$; thus assume for instance that uv is not in normal form. As u and v are in normal form, this implies that $u = u_1a$ and $v = cv_1$ with normal form $u_1bcv_1 = u_1bv$. If uw is in normal form, we get a contradiction since $uw = u_1aw$ and u_1bv cannot be equal; so we have that $w = cw_1$ and the normal form is u_1bw. Since $\vdash uv = uw$, we have that $u_1bv = u_1bw$, and because v and w are in normal form, we get $v = w$, i.e., (LC) holds in M. $\Diamond$

Another example where left- but not right-cancellation holds are the set of ordinals with their (non-commutative) addition.

3.1.1 The Commutative Case

Throughout this subsection, we assume that all time domains are commutative. There will be a special subsection following this one collecting all results that also hold for non-commutative ones, together with counterexamples for properties that fail to hold.

According to their definition, time domains are merely special monoids. Hence, there is not yet a way of accounting for such typical time-related statements as *earlier* or *later* points in time, or some events having *longer* or *shorter* durations than others. To express these properties, there must be a way of *comparing* elements of a time domain. This is, as usual, formalised by postulating that each time domain should carry a *(partial) order* $\leq$. Furthermore, the order should be compatible with the addition in the following sense. It should always hold that $t \leq t + s$: after all, time $t + s$ was intuitively reached from time t by waiting for the, according to (AS), 'non-negative' duration s. Moreover, this should be the *only* way one can reach a later point: whenever $t \leq u$ there should be a (right) 'witness' $s \in \mathcal{T}$ such that $t + s = u$, i.e., u can only be reached from t by waiting for s units of time in between. This leads to the following definition[2]:

Definition 3.5 ([JSV93])
Let $\mathcal{T}$ be a monoid. The *precedence relation* $\leq$ of $\mathcal{T}$ is defined, for all $t, u \in \mathcal{T}$, as

$$t \leq u \overset{\text{df}}{\Leftrightarrow} (\exists s \in \mathcal{T}).\, t + s = u \tag{3.2}$$

As usual, $t < u$ denotes $t \leq u \wedge t \neq u$, and if $I \subseteq \mathcal{T}$, $I \leq t$ denotes $(\forall u \in I).\, u \leq t$, i.e., that t is an *upper bound* of I. Furthermore, write $t \| u$ if neither $t \leq u$ nor $u \leq t$ holds, and call t and u *incomparable* in this case. ■

Note that incomparability is symmetric, i.e., $t \| u \;\Leftrightarrow\; u \| t$. The notation is derived from partial-order semantics for concurrency (see [NPW81, Pra86]): there, incomparable states can be executed in parallel, so we chose to use the symbol $\|$ usually associated with parallelism.

Proposition 3.6
Let $\mathcal{T}$ be a monoid. Then its precedence relation $\leq$ is reflexive and transitive, i.e., it is a pre-order.

Proof: Let $t \in \mathcal{T}$. Since $t + 0 = t$, it follows that $t \leq t$. Let $t, u, v \in \mathcal{T}$ such that $t \leq u$ and $u \leq v$. By definition, there must exist $s_1, s_2 \in \mathcal{T}$ such that $t + s_1 = u$ and $u + s_2 = v$. Substituting the first equation in the second, we obtain

$$v = u + s_2 = (t + s_1) + s_2 = t + (s_1 + s_2)$$

by associativity; hence, $t \leq v$ follows. □

Since $\leq$ is always a pre-order and defined in terms of $+$, it will also be referred to as the *induced pre-order*. For commutative time domains, one can show more (the first two items are already contained in [JSV93]):

Proposition 3.7
Let $\mathcal{T}$ be a commutative time domain.

[2] The terminology 'precedence relation' stems from the field of *tense logics*, see [vB89] for an overview.

1. The precedence relation $\leq$ on $\mathcal{T}$ is a partial order.

2. The neutral element $0 \in \mathcal{T}$ is the least element with respect to $\leq$. Furthermore, there are no maximal elements with respect to $\leq$, unless $\mathcal{T}$ is trivial.

3. The following equivalence holds for all $s, t, u \in \mathcal{T}$:

   $$t \leq u \;\Leftrightarrow\; s+t \leq s+u \tag{3.3}$$

   In particular, $+$ is monotone with respect to $\leq$.

Proof:

1. It remains to show that $\leq$ is anti-symmetric. Therefore, let $t, u \in \mathcal{T}$ such that $t \leq u$ and $u \leq t$. Hence, by definition, there exist $s_1, s_2 \in \mathcal{T}$ such that $t + s_1 = u$ and $u + s_2 = t$, and calculate: $t + 0 = t = u + s_2 = (t + s_1) + s_2 = t + (s_1 + s_2)$. Applying (LC) yields $0 = s_1 + s_2$, hence $s_1 = s_2 = 0$ by (AS).

2. Trivially $0 + t = t$, and so $0 \leq t$ for each $t \in \mathcal{T}$. Assume now that there is $m \in \mathcal{T}$ such that $t \leq m$ for all $t \in \mathcal{T}$; in particular $m + m \leq m$. Thus, by definition, there must be $s \in \mathcal{T}$ such that $m + (m + s) = (m + m) + s = m = m + 0$. Applying (LC) yields $m + s = 0$, so (AS) implies $m = 0$ and $\mathcal{T}$ is trivial.

3. If $t \leq u$, we have $t + v = u$ for some $v \in \mathcal{T}$; so $(s + t) + v = s + (t + v) = s + u$, i.e., $s + t \leq s + u$. In the other direction, assume $s + t \leq s + u$; then there exists $v \in \mathcal{T}$ such that $s + (t + v) = (s + t) + v = s + u$. By (LC), $t + v = u$, or $t \leq u$. The claim about monotonicity follows since the equivalence states that both functions $s + \_$ and $\_ + s$ are monotone, and being monotone in each argument separately is the same as being jointly monotone in both arguments since the order on the product is given by the component-wise order, see, e.g., [Plo83]. $\square$

Because the induced pre-order is actually a partial order for time domains, we call it the *induced order*. An immediate consequence of the preceding lemma is that any non-trivial time domain is automatically infinite: otherwise, there would be a maximal element. Moreover, homomorphisms $f : \mathcal{T} \to \mathcal{T}'$ of time domains $\mathcal{T}$ and $\mathcal{T}'$ are automatically monotone with respect to their respective induced orders $\leq$ and $\leq'$: for $t \leq u$, which is equivalent to $u = t + s$, we obtain

$$f(t) \leq' f(t) + f(s) = f(t+s) = f(u)$$

Note that one direction, viz., right-to-left, of the equivalence (3.3) is an order-theoretic version of (LC), and hence could be called *(left) order-cancellation*.

Remark 3.8

From the previous proposition, it follows that a commutative time domain is a monoid object [Mac97] in the category **POSets** of partially ordered sets and monotone functions. $\Diamond$

Example 3.9

Let us consider the previously given examples of time domains and describe their precedence relations concretely.

1. For the trivial domain, the order is also trivial.

2. For discrete or continuous time, the precedence relation is precisely the standard ‘less-than-or-equal relation; in both cases, the order is actually *linear*: any two elements of the respective domain are comparable with respect to $\leq$.

3. The precedence relations of $\mathbb{N}^n$ and $\mathbb{R}^n$ are the component-wise extensions of the standard order $\leq$ on $\mathbb{N}$ and $\mathbb{R}$, respectively; these are true partial orders, e.g., $\langle 1,0\rangle || \langle 0,1\rangle$. For a general product $\mathcal{T} \times \mathcal{T}'$, one also obtains the component-wise combination of the respective orders on $\mathcal{T}$ and $\mathcal{T}'$; if both $\mathcal{T}$ and $\mathcal{T}'$ are non-trivial, this is again only a partial order. $\Diamond$

Axiom (LC) has an important consequence:

Lemma 3.10 ([JSV93])

Let $\mathcal{T}$ be a commutative time domain with precedence relation $\leq$, and let $t, u \in \mathcal{T}$. Then if $t \leq u$ there is a unique $s \in \mathcal{T}$ such that $t + s = u$.

Proof: Assume $t \leq u$; hence, by definition, there exists $s \in \mathcal{T}$ such that $t + s = u$. Now suppose that there is $s' \in \mathcal{T}$ such that also $t + s' = u$, i.e., $t + s = u = t + s'$. Applying (LC) immediately yields $s = s'$, as desired. $\Box$

This enables us to make the following definition:

Definition 3.11

Let $\mathcal{T}$ be a commutative time domain with precedence relation $\leq$, and let $t, u \in \mathcal{T}$ be such that $t \leq u$. Then the (by Lemma 3.10) uniquely determined $s \in \mathcal{T}$ such that $t + s = u$ is called the *(right) relative inverse of* t *with respect to* u, written as $u - t$. These relative inverses induce a partial function $\mathcal{T} \times \mathcal{T} \rightharpoonup \mathcal{T}$

$$\langle u, t \rangle \mapsto \begin{cases} u - t & \text{if } t \leq u \\ \text{undef} & \text{otherwise} \end{cases} \tag{3.4}$$

which is called the *partial subtraction function*. ■

By Lemma 3.10, if $t \leq u$, we obtain $(u - t) \in \mathcal{T}$ as the unique solution to the equation

$$t + x = u \tag{3.5}$$

taken in the time domain $\mathcal{T}$. The notation $u - t$ is chosen because in analogy to the derived notion of *(right) subtraction* in the extension group G of $\mathcal{T}$, $u - t \overset{\text{df}}{=} (-t) + u$. By definition, $u - t$ satisfies the following calculations in G:

$$t + (u - t) = t + ((-t) + u) = (t + (-t)) + u = 0 + u = u$$

So the derived relative inverses in G are the (necessarily unique) solutions for (3.5) when taken in G. Consequently, the relative inverses in $\mathcal{T}$ are simply obtained by restricting the derived notion of subtraction in G to such pairs $\langle t, u \rangle$ in $\mathcal{T} \times \mathcal{T} \subset G \times G$ for which $t \leq u$ holds.

It would also be possible to define *left* relative inverses $u + (-t)$ which correspondingly are the unique solutions for y in the equation $y + t = u$. Naturally, for commutative groups, the two notions of relative inverse coincide.

Example 3.12

All our examples can be embedded into groups, so the relative inverses are just computed by restricting the derived subtraction from the respective extension groups:

1. For discrete and continuous time, the relative inverse of t with respect to u is $u-t$ for the usual subtraction, which is well-defined since $t \leq u$.

2. For $\mathbb{N}^n$ or $\mathbb{R}^n$, the relative inverses are obtained by component-wise subtraction: recall that $\vec{t} \leq \vec{u}$ iff each component of $\vec{t}$ is less than or equal to the corresponding component of $\vec{u}$, and so subtraction is well-defined for all components; this, again, can be generalised to arbitrary products. $\Diamond$

Lemma 3.13
Let $\mathcal{T}$ be a commutative time domain.

1. For all $t,u \in \mathcal{T}$, $t-t=0$ and $(t+u)-t=u$.

2. Let $s,t,u \in \mathcal{T}$. Then

   (a) $s \leq u \Rightarrow ((s-t)\downarrow \Rightarrow (u-t)\downarrow)$

   (b) $((s-u)\downarrow \wedge s-u \leq t) \Leftrightarrow u \leq s \leq u+t$

3. For $t \in \mathcal{T}$, $(\cdot)-t : \mathcal{T} \rightharpoonup \mathcal{T}$ is monotone with respect to $\lesssim$:

$$(\forall s,u \in \mathcal{T}).\ s \lesssim u \Rightarrow s-t \lesssim u-t$$

4. Let $s,t,u \in \mathcal{T}$. Then

   (a) $(u-s)+t \sqsubseteq_{\sim} (u+t)-s$

   (b) $t-(s-u) \sqsubseteq_{\sim} (u+t)-s$

Proof: All claims are trivially verified when calculation in an extension group G of $\mathcal{T}$, which exists by Remark 3.3. However, we can also establish them directly.

1. Since $t+0=t$, and $t-t$ is the unique element of $\mathcal{T}$ such that $t+(t-t)=t$, $0=t-t$ follows. Let now be $t,u \in \mathcal{T}$. The value of $(t+u)-t$ is the unique $s \in \mathcal{T}$ such that $t+s=t+u$; since trivially $t+u=t+u$, it follows that $(t+u)-t=u$, as claimed.

2. Let $s,t,u \in \mathcal{T}$.

(a) This is just the transitivity of $\leq$.

(b) Assume $(s-u)\downarrow$ and $s-u \leq t$. Then $u \leq s$; moreover, $s = u+(s-u) \leq u+t$. In the other direction, assume $u \leq s \leq u+t$. Then $(s-u)\downarrow$ and by (3.3), $s-u \leq (u+t)-u = t$.

3. Assume $s \lesssim u$; this is equivalent to $s \leq u$. Furthermore, assume $(s-t)\downarrow$; hence, by the previous point, $(u-t)\downarrow$. We then have

$$t+(s-t) = s \leq u = t+(u-t)$$

Applying Proposition 3.7 to cancel the t on both sides yields the claim.

4. Let $s,t,u \in \mathcal{T}$.

(a) Assume $((u-s)+t)\downarrow$; this is equivalent to $s \leq u$. Then $(u-s)\downarrow$. Moreover, since $u \leq u+t$, certainly also $((u+t)-s)\downarrow$, so the equation makes sense. Furthermore, $s \leq u$ is equivalent to $u = s+(u-s)$, and so we get

$$s+((u-s)+t) = (s+(u-s))+t = u+t$$

Since relative inverses are unique, we obtain $(u-s)+t = (u+t)-s$, as claimed.

(b) Assume $(t-(s-u))\downarrow$; this is equivalent to $u \leq s \leq u+t$ and hence, the equation makes sense. Now calculate

$$\begin{aligned} s+((u+t)-s) &= u+t = u+((s-u)+(t-(s-u))) \\ &= (u+(s-u))+(t-(s-u)) = s+(t-(s-u)) \end{aligned}$$

and so, by (LC), we obtain $t-(s-u) = (u+t)-s$, as claimed. □

Definition 3.14

Let $\mathcal{T}$ be a commutative time domain with precedence relation $\leq$. Define *truncated subtraction* [Law73], also called *monus* in [Jac96, Jac00], $\dot{-} : \mathcal{T} \times \mathcal{T} \to \mathcal{T}$ by

$$u \mathbin{\dot{-}} t = \begin{cases} u-t & \text{if } t \leq u \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

Moreover, say that a time domain $\mathcal{T}$ is *linear* if its precedence relation $\leq$ is a linear order, i.e., for all $t,u \in \mathcal{T}$ either $t \leq u$ or $u \leq t$ holds. ■

Note that, if $\mathcal{T}$ is not linear and $t \| u$, then $t \dot{-} u = u \dot{-} t = 0$. Also note that $(\cdot) \dot{-} t$ is monotone with respect to the precendence relation $\leq$: this follows from the analogous property of the partial subtraction function $-$ from (3.4) proved in Lemma 3.13.

Both $\mathbb{N}$ and $\mathbb{R}_{\geq 0}$ are examples of linear time domains, whereas products and powers like $\mathbb{N}^n$ and $\mathbb{R}^n$ in general are not. Since 0 is the least element in every time domain $\mathcal{T}$ and that there are no maximal elements, a linear time domain corresponds to the familiar view of time as a *time line*: starting at 0, it extends in a straight line towards 'infinity' (cf. Proposition 3.7 where it was established that non-trivial time domains cannot have maximal elements).

Previously, it was shown that every time domain $\mathcal{T}$ is a partially ordered monoid, so it is a category with objects $t \in \mathcal{T}$ and morphisms $t \to u$ if and only iff $u \leq t$ holds; in particular, 0 is an object of $\mathcal{T}$. Note that regarding (pre-)orders as categories in this way, as in [Law73], is *dual* to the standard approach presented in [Mac97]; the reason for doing so will become clear in the next proposition.

We have seen that the functions $t + \_ : \mathcal{T} \to \mathcal{T}$ are all monotone and hence functors on $\mathcal{T}$. In the linear case, the right adjoint of each of these functors exists:

Proposition 3.15

For every commutative time domain $\mathcal{T}$, the following are equivalent:

(i) for all $t \in \mathcal{T}$, truncated subtraction $(\cdot) \dot{-} t$ is the right adjoint of $t + (\cdot)$, i.e.,

$$t + s \geq u \;\Leftrightarrow\; s \geq u \dot{-} t \tag{3.7}$$

(ii) $\mathcal{T}$ is linear

Proof:

(i) $\Rightarrow$ (ii): Assume that (3.7) holds. Since we have to show that $\leq$ is linear, assume $t \not\leq u$. By definition of $\dot{-}$, this implies that $u \dot{-} t = 0$. Now choose $s = 0$; trivially, $0 = s \geq u \dot{-} t = 0$. Applying (3.7) then yields $t = t + 0 = t + s \geq u$. Consequently, $t \not\leq u$ implies $t \geq u$, i.e., $\mathcal{T}$ is linear.

(ii) $\Rightarrow$ (i): Let $t, u, s \in \mathcal{T}$. Two cases arise since $\mathcal{T}$ is linear.

- $t \leq u$: Lemma 3.13 then states

$$((u - t)\downarrow \;\wedge\; u - t \leq s) \;\Leftrightarrow\; t \leq u \leq t + s$$

However, by $t \leq u$, $(u - t)\downarrow$ and $u \dot{-} t = u - t$, so this is equivalent to (3.7).

- $u \leq t$: Then trivially $u \leq t + s$, $u \dot{-} t = 0$, and also $s \geq 0$, so (3.7) holds. □

Additionally, since we assume $\mathcal{T}$ to be commutative, $(\cdot) + t = t + (\cdot)$; so also $(\cdot) + t$ is monotone, i.e., a functor on $\mathcal{T}$, making $+$ a bifunctor on $\mathcal{T}$. Moreover, the monoid laws then make $\langle \mathcal{T}, +, 0 \rangle$ into a (symmetric) monoidal category. An obvious question is whether this monoidal structure on $\mathcal{T}$ is closed in the sense of [EK66, Law73], leading to *closed* time domains. From Proposition 3.15, we obtain:

Corollary 3.16
Every linear commutative time domain is closed. □

There is an interesting phenomenon happening for $\mathcal{T} = \mathbb{N} \times \mathbb{N}$: it is a commutative, non-linearly ordered time domain. Moreover, it is also closed: this follows because the product of closed categories is again closed; concretely, the closed structure on $\mathcal{T}$ is given component-wise, i.e.,

$$\langle m_1, n_1 \rangle \dot{-} \langle m_2, n_2 \rangle \stackrel{\mathsf{df}}{=} \langle m_1 \dot{-} m_2, n_1 \dot{-} n_2 \rangle$$

It is easy to see that this definition validates (3.7). However, this is not the truncated partial subtraction on $\mathbb{N} \times \mathbb{N}$ from (3.6): according to the component-wise definition, $\langle 1, 0 \rangle \dot{-} \langle 0, 1 \rangle = \langle 1, 1 \rangle$, while the result for the order-induced variant is $\langle 0, 0 \rangle$ because $\langle 1, 0 \rangle || \langle 0, 1 \rangle$; consequently, (3.7) does *not* hold for the order-induced variant.

3.1.2 The Non-Commutative Case

After presenting the properties of commutative time domains, we can do the same for the general notion. The aim of this section is to illustrate how far properties carry over from the commutative case; we are not going to reprove those results here that hold in all generality, we simply note that the previous proofs did not use commutativity.

Proposition 3.17
Let $\mathcal{T}$ be an arbitrary time domain. Then:

1. Its induced pre-order $\leq$, as defined in (3.2), is a partial order with 0 as the least element, and there are no maximal elements with respect to $\leq$ unless $\mathcal{T}$ is trivial.

2. The equivalence (3.3) still holds as it is written:

$$t \leq u \Leftrightarrow s+t \leq s+u$$

In particular, the function $s+\_ : \mathcal{T} \to \mathcal{T}$ is monotone. $\square$

Example 3.18
For local qualitative time, the precedence relation is exactly the *prefix relation* on finite words: let $w, v \in C^\star$ be two words; then $w \leq v$ holds if and only if v can be written as ww' for some word $w' \in C^\star$. For $|C| > 1$, this only yields a partial order, while for $|C| = 1$, this is simply the standard order on the naturals. $\Diamond$

Again, homomorphisms are always monotone. Note that, in general, the function $\_+s : \mathcal{T} \to \mathcal{T}$ is not monotone with respect to $\leq$: if $\mathcal{T} = C^\star$ for the three-element set $C = \{a, b, c\}$, 'post-fixing' does not preserve the prefix relation, e.g., $a \leq ab$ yet $ac \not\leq abc$, hence $\_+c$ is not monotone.

Lemma 3.19
Any time domain has (right) relative inverses: if $t \leq u$ there exists the uniquely determined $u-t \in \mathcal{T}$ such that $t+(u-t) = u$. $\square$

Note that, in contrast to the commutative case, the relative inverses $u-t$, for $t \leq u$, in general only satisfy $t+(u-t) = t$, i.e., they are only inverses when added *on the right*; analogously, if $\mathcal{T}$ can be embedded into a group (which is not always the case for non-commutative $\mathcal{T}$, see Example 3.4), the two notions of relative inverses in the group are different (if $\mathcal{T}$ and, consequently, also its extension group are non-commutative): for instance, consider the free group on the two-element set $\{c, d\}$ where $c^{-1}d \neq dc^{-1}$.

Example 3.20
For local qualitative time, $w \leq v$ can only hold if w is a prefix of v, i.e., $v = ww'$, and then the relative inverse $v-w$ is given by removing the prefix w from v, i.e., $v-w = ww'-w = w'$. $\Diamond$

Lemma 3.13 carries over completely:

Lemma 3.21
Let $\mathcal{T}$ be an arbitrary time domain. Then:

1. For all $t, u \in \mathcal{T}$, $t - t = 0$ and $(t + u) - t = u$.
2. Let $s, t, u \in \mathcal{T}$. Then
   (a) $s \leq u \Rightarrow ((s-t)\downarrow \Rightarrow (u-t)\downarrow)$
   (b) $((s-u)\downarrow \wedge s - u \leq t) \Leftrightarrow u \leq s \leq u + t$
3. For $t \in \mathcal{T}$, $(\cdot) - t : \mathcal{T} \rightharpoonup \mathcal{T}$ is monotone with respect to $\lesssim$:
$$(\forall s, u \in \mathcal{T}).\ s \lesssim u \Rightarrow s - t \lesssim u - t$$
4. Let $s, t, u \in \mathcal{T}$. Then
   (a) $(u - s) + t \sqsubseteq_{\sim} (u + t) - s$
   (b) $t - (s - u) \sqsubseteq_{\sim} (u + t) - s$ □

The definitions of truncated subtraction $\dot{-}$ and linearity generalise can be to arbitrary time domains, also the proof of Proposition 3.15 does not use commutativity:

Proposition 3.22
For an arbitrary time domain $\mathcal{T}$, the following are equivalent:

(i) for all $t \in \mathcal{T}$, truncated subtraction $(\cdot) \dot{-} t$ is the right adjoint of $t + (\cdot)$

(ii) $\mathcal{T}$ is linear □

As an example, consider the time domain of ordinals with their non-commutative addition: it is linear, hence truncated subtraction is the right adjoint to $+$.

However, non-commutative time domains are, in general, not closed: in fact, they are not even always monoidal categories as $\_ + s$ may not be monotone (e.g., for $C^{\star}$). The requirement '$\_ + s$ is monotone' is equivalent to

$$(\forall s, t \in \mathcal{T}).\ (\exists s' \in T).\ s + s' = t + s \tag{3.8}$$

which is satisfied if $\mathcal{T}$ is commutative, but does not hold for $C^\star$. Consequently, closure for non-commutative time domains only makes sense *after* imposing suitable conditions like (3.8) which guarantee that $+$ is a bifunctor.

For $|C| > 2$, for example $C = \{a,b\}$, $C^\star$ is a non-commutative and non-linear time domain. So Proposition 3.22 does not hold: the *specific* functor $(\cdot) \dot{-} t$ is not the right adjoint of $t + (\cdot)$. Actually, there cannot be *any* right adjoint for $t + (\cdot)$ in $C^\star$: its existence would imply that $\{a,b\}$ has an upper bound in $C^\star$, which is impossible.

3.1.3 Comparison with Other Models

After presenting this formalisation of time by the time domains from [JSV93], we want to briefly compare it with other notions from the literature. Frequently, languages for timed processes are only presented with respect to one concrete model of time, e.g., $\mathbb{N} \setminus \{0\}$ in [MT90], $\mathbb{R}_{\geq 0} \setminus \{0\}$ in [Sch95], and $\mathbb{R}_{\geq 0}$ in [Wan90]; on top of that, this model is usually simply regarded as a *set*, making no reference to the semigroup/monoid and/or order structure, despite quite freely making use of both when defining the operational semantics of processes and proving propositions related to it. Aiming for a *formal* and *conceptual* account of timed processes, we chose to take an *axiomatic* approach in an attempt to make our work as 'generic' as possible.

Apart from the concrete approaches, there are also a number of other axiomatic accounts of time. These all turn out to be less general than the one from [JSV93]. Jeffrey, in [Jef91], postulates that a time domain is a left-cancellative monoid whose precedence relation $\leq$ is a linear order with greatest lower bounds $\inf T$ of all non-empty subsets $T \subseteq \mathcal{T}$. Therefore, it rules out both free and product monoids since those are only partially ordered. Finally, Nicollin and Sifakis in [NS91] (as well as in the later paper [NSY93]), introduce time domains as commutative monoids satisfying

$$t + t' = t \Rightarrow t' = 0 \tag{3.9}$$

which are, additionally, linearly ordered by the precedence relation $\leq$ as introduced before. Starting from these assumptions, they are able to show that 0 is the least element with respect to $\leq$, and that the relative inverses $u - t$ exists for $t \leq u$ (essentially following the same arguments as above). It is obvious that their definition is strictly less general than the one adopted here: their condition (3.9) is a consequence of our axioms (AS) and (LC); consequently, each of their time domains is one according to our definition, while their definition again excludes the partially ordered or non-commutative examples of product and free monoids given above.

All in all, we believe the presented variant of time domains strikes a reasonable balance between modelling intuitive properties of time and still being general enough to include many interesting examples, making it an adequate choice for providing the basis of a mathematical treatment of timed processes.

3.2 Transition Systems for Timed Processes

Having formalised time by time domains, this section now provides a precise model of timed processes based on a special kind of transition systems, together with a natural and appropriate notion of equivalence. After a concrete definition of these transition systems, a more mathematical characterisation of them as *partial monoid actions* is presented.

3.2.1 Timed Transition Systems and Time Bisimulation

We are now going to describe *timed transition systems* (TTSs)[3] as a model of the way processes evolve over time; the notion we present is synthesised from various accounts in the literature, viz., [Wan90, NS91, JSV93]. To distinguish these TTSs from standard labelled transition systems as in Def. 2.2, which are also used for describing the action transitions of timed processes, a slightly different notation, inspired by some of the calculi for timed processes, will be used. In the following definition, recall that the operations of the time domain $\mathcal{T}$ are written additively even although $\mathcal{T}$ need not be commutative.

Definition 3.23

A *timed transition system* (TTS) is a labelled transition system $\langle P, \mathcal{T}, \leadsto \rangle$ where P is a set of *processes*, $\mathcal{T} = \langle \mathcal{T}, +, 0 \rangle$ is a time domain, and the *time transition relation*

[3]Not to be confused with the *typed* transition systems of [Kah96]!

$\rightsquigarrow \subseteq P \times \mathcal{T} \times P$ satisfies the following axioms, writing $\langle p, t, p' \rangle \in \rightsquigarrow$ as $p \overset{t}{\rightsquigarrow} p'$, and using $p, p', \ldots$ and $s, t, u, \ldots$ to range over P and $\mathcal{T}$, respectively:

(Determinacy) $\quad p \overset{t}{\rightsquigarrow} p' \wedge p \overset{t}{\rightsquigarrow} p'' \Rightarrow p' = p''$

(ZeroDelay) $\quad p \overset{0}{\rightsquigarrow} p$

(Continuity) $\quad p \overset{t+u}{\rightsquigarrow} p' \Leftrightarrow (\exists p'').\ p \overset{t}{\rightsquigarrow} p'' \overset{u}{\rightsquigarrow} p'$

If $\mathcal{T}$ and $\rightsquigarrow$ are clear from the context, we sometimes identify a TTS simply with its set P of states; given a TTS $\langle P, \mathcal{T}, \rightsquigarrow \rangle$, we sometimes call it a *TTS over* $\mathcal{T}$ when we want to particularly emphasise which time domain $\mathcal{T}$ is used as the set of labels. ■

Let us now try to intuitively justify the restrictions imposed by the axioms; again, although the notion of TTS is synthesised from the literature, we still want to give some motivation to make the underlying ideas more palatable, partly also prompted by the lack of explanatory material in the literature. Note that the name 'continuity' might seem a little odd but it is standard in the literature.

All[4] models and languages from the literature, e.g., in [Jef91, JSV93, MT90, NS91, Sch95, Wan90, BM01], assume that time passes *deterministically*, as is expressed in (Determinacy): given a state p, waiting for the same amount of time invariably results in reaching the same state $p' = p''$. Hence, time transitions in a TTS model *time passing without non-deterministic side-effects*. The underlying intuition is that the only way non-determinism can be introduced is by 'real' computations, i.e., by action transitions, which actually 'do something.' Consequently, all choices arising from non-deterministic computations should also be resolved exclusively by action transitions; time transitions simply model 'waiting for a prescribed period of time.'

Note that this could indeed affect the potential for action behaviour, but in a deterministic fashion. In particular, by only excluding *non-deterministic* side effects, we have *not* ruled out *time-out operators* as are sometimes present in languages for timed processes, e.g., [AM94, HR95, NS94]. As a mundane example, consider going to a cinema with several screens where different films are shown at different times. When waiting for too long, the choice of films being shown changes; more abstractly, this corresponds to time-outs: the possibility to see a certain film at a certain time is gone once

[4] With the exception of the early paper [Gro90] which has since been superseded.

that time has passed. Yet that does not mean that time passes non-deterministically: two people waiting together will not suddenly have different choices of films; idling may change the potential behaviour, provided that it always does so in the same way, producing the same results.

Furthermore, note that (Determinacy) is also different from the *persistency* axiom advocated in [Wan90, Wan91, Sch95]:

$$\text{(Persistency)} \qquad p \stackrel{\alpha}{\rightarrow} \;\wedge\; p \stackrel{t}{\rightsquigarrow} p' \;\Rightarrow\; p' \stackrel{\alpha}{\rightarrow}$$

In contrast to (Determinacy), (Persistency) indeed expresses that observable[5] action transitions, i.e., action transitions with labels $\neq \tau$, cannot be disabled by the passage of time. Assuming (Maximal Progress), it prohibits time-outs which are not triggered by performing an internal event, i.e., a $\stackrel{\tau}{\rightarrow}$-transition: compare the difference between the respective time-out operators in [NS91] and in [Sch95] (the languages in [Wan90, Wan91] do not contain such an operator). These initial internal actions strike us as unnecessary ([NS91] defines a time-out without internal actions), and even unrealistic (what is their intuitive interpretation?), they seem more like a 'hack,' designed to validate (Persistency) despite the presence of time-outs. Consequently, we chose not to adopt this property.

More conceptually, since TTSs provide the *model* for time passing (and nothing else!), their definition should *not* make any reference to action transitions: the two kinds of transitions represent (abstractions of) *orthogonal* properties of real-life events. Hence, they should not be mixed on the level of models, otherwise the separation would be rather superfluous. On the level of *languages* for timed processes, this is a completely different issue. There, it makes perfect sense to have the two kinds of transitions mutually interdependent, cf. calculi like TiCCS [Wan90] which adopt (any form of) the maximal progress assumption [HdR89]: languages define processes with both action and time transitions, it is only the two separate models which, in our opinion, should be kept strictly apart.

However, using a model already containing two kinds of transitions (see, e.g., Chapter 7), we would no longer object to postulating axioms which relate the two

[5]Since the mentioned languages also adopt (Maximal Progress), if $\alpha = \tau$ then only $t = 0$ is possible, and then (ZeroDelay) anyway implies $p' = p$; consequently, (Persistency) holds anyway for internal transitions.

kinds of transitions because the axioms then indeed stay 'withing the model.' It is only when we consider the time transitions *on their own*, as now, that the model should be 'closed,' i.e., contain no 'external references' to other structures not formally part of it. This is in the same vein as introducing time domains as special monoids, not just sets: we want to use addition, the precedence relation, etc., in the operational semantics, so we have to make it part of the model when attempting to be precise and formal.

As for (ZeroDelay), most languages and models do *not* consider $\overset{0}{\leadsto}$-transitions, exceptions being [Jef91, JSV93]. Yet, such transitions can safely be added according to the axiom, there is nothing intrinsic to the languages that would prohibit doing so. Regarding its intuitive content, (ZeroDelay) is very clear, in particular when recalling the intuition that 0 represents the absence of time passing: when a process is in a state p and then no time passes, the state p should not be affected. In other words: no instantaneous state transitions should be possible. Note that this has no effect on the assumption of instantaneous *action* transitions, they model *computations* which rightly should be allowed to change the state of a process; their being instantaneous is simply an abstraction of the real world. However, since time transitions represent *idling* phases, i.e., doing nothing, instantaneous state changes, when *no* idling occurs, in addition to the absence of action transitions, would seem counterintuitive. In some sense, this axiom again expresses the deterministic nature of time passing: there are no spontaneous transitions in the absence of idling (and computations). Furthermore note that one of the motivations to include $\overset{0}{\leadsto}$-transitions in our model is to also have a measure for the duration of (instantaneous) action transitions, as already explained during the justification of the monoid structure on a time domain.

Finally, (Continuity) is again widely accepted, e.g., [MT90, NS91, Sch95, Wan90]. There also exist variants in the literature which consider only one of the implications, viz., the direction left-to-right in [JSV93], and the right-to-left direction in [Jef91]. Yet not only can the use of both implications be intuitively justified, it also yields a mathematically more pleasing model. As regards the latter point, we believe that using the equivalence, as stated above, fittingly incorporates a form of 'homomorphism' property with respect to the time domain for the labels: while (ZeroDelay) lifts the neutral element $0 \in \mathcal{T}$ in an adequate way to the level of time transitions, (Continuity) expresses that the time transitions comply with the monoid composition $+$. Hence, the structure of the time domain is reflected in the structure of TTSs built on top of it.

Of course, the previous ‘argument’ is mostly formalistic, or even ‘aesthetic’; nevertheless, it has its virtues: one of our goals is a *conceptually* sound model for timed processes, and with such an objective, criteria of a more aesthetic nature should not be too easily dismissed. Besides, since our interpretation of ‘conceptual’ is ‘within an abstract categorical framework,’ there are also technical considerations to support this choice of axiomatisation, as will become clear in Chapter 4: this formulation makes TTSs much more amenable[6] to a categorical treatment.

Apart from these ‘meta’-considerations, there is also a more ‘hands-on’ intuition for (Continuity): the idea behind the axiom, according to [Wan90], is that ‘if an agent [= timed process] proceeds from one instant to the other, it must reach all intermediate instants in between.’ To illustrate this, let us consider the two implications separately. The right-to-left direction, which incorporates a form of *transitivity* (or *additivity*), should always be valid since it precisely captures the intuition of the monoid composition: it describes the duration of consecutive events. If p first idles for t units of time, becoming p'' in doing so, and then p'' idles for u units of time, i.e., two events with respective durations t and u are performed consecutively, $t+u$ precisely represents the total duration of the two events put together and p should in this situation be able to perform a $\overset{t+u}{\rightsquigarrow}$-transition: starting in state p, nothing but idling for $t+u$ units of time is done, simply the intermediate state p'' (or the state reached after an ‘intermediate instant’ in the terminology of [Wan90]) is made explicit.

The implication from left to right, which could be called *interpolation* (or *density*), states that any time transition with a composite label of the form $t+u$ can be *decomposed* according to the monoid addition. For its justification, recall that in the induced order structure on a time domain $\mathcal{T}$, the only way for $t \leq s$ to hold is that $s = t+u$ for some (necessarily unique) $u \in \mathcal{T}$. Therefore, $t+u$ is really just an arbitrary duration greater than or equal to t. With this in mind, during an idling period of $t+u$ units of time, one should in particular have to wait for the shorter amount of t units of time

[6] Note additionally that, in our opinion, evidence from the literature suggests that categorical techniques favour ‘beautiful’ solutions over ad-hoc ones.

first: time should not have ‘holes’ allowing to ‘pass by’ intermediate times[7].

Consequently, if p can perform a $\overset{t+u}{\leadsto}$-transition to p', there should exist a state p'' such that there is a $\overset{t}{\leadsto}$-transition from p to p'', allowing to idle for (the shorter amount of) t units of time. Furthermore, by (Determinacy), p'' and p' are unique. So at this point of the argument, we know that p can perform a $\overset{t}{\leadsto}$-transition to p'', in addition to the assumed $\overset{t+u}{\leadsto}$-transition to p'. Suppose now, in contradiction to (Continuity), that it is impossible to perform a $\overset{u}{\leadsto}$-transition from p'' to p'. This would result in quite a paradoxical situation: starting in state p, one can idle for t units of time and then continue idling for u units of time in order to achieve the total idling time of duration $t+u$, yet when starting in state p'', having been reached from p by idling for t units of time, it would not be possible to wait for u units of time. From this point of view, it seems completely natural that there should be a $\overset{u}{\leadsto}$-transition from p'' to p', allowing to ‘close the gap’.

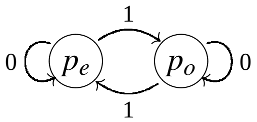
Let us now present some examples of TTSs.

Example 3.24

1. As a simple example, consider the case of $\mathcal{T} = \mathbb{N}$ and let $P = \{p_e, p_o\}$ with the transition relation ‘generated’ by the following $\overset{1}{\leadsto}$-transitions:

$$p_e \overset{1}{\leadsto} p_o \qquad p_o \overset{1}{\leadsto} p_e$$

After adding $\overset{0}{\leadsto}$-transitions according to (ZeroDelay), we can picture the transition system as follows:



Note that this is not yet a TTS since the LTS does not satisfy (Continuity): for example, the following sequence of transitions

$$p_e \overset{1}{\leadsto} p_o \overset{1}{\leadsto} p_e$$

[7] Note that this is quite different from non-continuous *observation* of processes: it is perfectly plausible that one is only interested in the state of processes at specific temporal ‘inspection points’. Yet this should not be confused with the fact that the process has to ‘exist’, i.e., has to be in some state, at times prior to an observation point. After all, how should any kind of inspection be performed if the process had ‘died’ or ‘disappeared’ earlier on? Moreover, where to should it disappear?

is derivable but the transition system does not contain a transition labelled with $1+1=2$ from p_e to itself. To rectify this, we have to add, for each $n \in \mathbb{N}$, the following transitions:

$$\begin{array}{cc} p_e \overset{2n}{\leadsto} p_e & p_o \overset{2n}{\leadsto} p_o \\ p_e \overset{2n+1}{\leadsto} p_o & p_o \overset{2n+1}{\leadsto} p_e \end{array}$$

With these transitions added, it is now easy to show that the resulting transition system is indeed a TTS (the names p_e and p_o were chosen with the intuition that, with p_e considered as the start state, p_e is the state reached after an *e*ven number of time units, while p_o is the state corresponding to an *o*dd number of time units having passed).

2. To give a more interesting example of a TTS, consider the set of TeCCS agents, i.e., all closed process expressions derivable from the grammar of the language (cf. [MT90]). After adding transitions according to (ZeroDelay), one obtains a TTS $\langle \text{TeCCS}, \mathbb{N}, \leadsto \rangle$ where $\leadsto$ is the time transition relation on agents defined by the time rules of the language. ◊

Next, we are going to present an adapted version of strong bisimulation from Definition 2.3 as a natural equivalence for TTSs.

Definition 3.25

Given two TTSs $\langle P_i, \mathcal{T}, \leadsto_i \rangle$, $i \in \{1,2\}$, a relation $R \subseteq P_1 \times P_2$ is a *(strong) time bisimulation (over $\mathcal{T}$)* if $\langle p_1, p_2 \rangle \in R$ implies for all $t \in \mathcal{T}$ that

$$\begin{array}{l} p_1 \overset{t}{\leadsto}_1 p_1' \Rightarrow (\exists p_2').\ p_2 \overset{t}{\leadsto}_2 p_2' \wedge \langle p_1', p_2' \rangle \in R \\ p_2 \overset{t}{\leadsto}_2 p_2' \Rightarrow (\exists p_1').\ p_1 \overset{t}{\leadsto}_1 p_1' \wedge \langle p_1', p_2' \rangle \in R \end{array}$$

We write $p_1 \sim_t p_2$ if there exists a strong time bisimulation containing $\langle p_1, p_2 \rangle$, i.e.,

$$\sim_t \overset{\text{df}}{=} \bigcup \{ R \subseteq P_1 \times P_2 \mid R \text{ is a time bisimulation} \} \tag{3.10}$$

If $P_1 = P_2 = P$, we say that R is a *time bisimulation on P*. ■

This notion of equivalence is very frequently used (usually combined with bisimulation for actions, see later on), the only exceptions we are aware of are TiCSP [Sch95], since in the CSP-tradition, (timed) *failures* [Hoa85] are the standard notion of equivalence, and TPL [HR95], where an approach based on *testing* [DH83] is proposed.

Example 3.26

1. As a simple example of a time bisimulation, again consider the previous example of a TTS with the two states p_e and p_o. Intuitively, the two states are equivalent since their potential for time transitions is the same: with a $\overset{2n}{\rightsquigarrow}$-transition, i.e., a time transition of even duration (recall that $\mathcal{T} = \mathbb{N}$ in this example), both stay in the same state, and for a $\overset{2n+1}{\rightsquigarrow}$-transition, i.e., a time transition of odd duration, one changes from p_e to p_o, and vice versa. This is formalised by the fact that the relation
$$R = \{\langle p_e, p_e\rangle, \langle p_e, p_o\rangle, \langle p_o, p_e\rangle, \langle p_o, p_o\rangle\}$$
is a time bisimulation.

2. The relation $\sim$ on TeCCS agents as defined in [MT90] is a time bisimulation on the TTS obtained on the set of TeCCS agents from the previous example. Note that this relation $\sim$ is different from the relation $\sim_t$ introduced in Definition 3.25 since the former also takes into account action transitions. We are going to discuss these two different notions shortly. $\Diamond$

If $P_1 = P_2 = P$, the relation $\sim_t \subseteq P \times P$ introduced in Definition 3.25 enjoys analogous properties to standard bisimulation, see e.g., [Mil89, Gla01]:

Proposition 3.27

Let $\langle P, \mathcal{T}, \rightsquigarrow\rangle$ be a TTS. Then the relation $\sim_t \subseteq P \times P$ is the largest time bisimulation on P and additionally an equivalence relation. $\Box$

Given a TTS $\langle P, \mathcal{T}, \rightsquigarrow\rangle$, we have seen that the largest bisimulation $\sim_t$ on P is an equivalence relation. As usual, we can therefore consider the set $P/_{\sim_t}$ of *processes modulo time bisimulation*, the set of equivalence classes of processes modulo the equivalence relation $\sim_t$. For $p \in P$, write its equivalence class with respect to $\sim_t$ as
$$[p]_{\sim_t} \overset{\text{df}}{=} \{q \in P \mid p \sim_t q\} \ ,$$

often dropping the subscript $\sim_t$, simply writing $[p]$. As we will now show, the TTS on P can be extended to the equivalence classes $P/_{\sim_t}$:

Proposition 3.28

Let $\langle P, \mathcal{T}, \rightsquigarrow\rangle$ be a TTS with largest bisimulation $\sim_t \subseteq P \times P$. For $p, p' \in P$ and $s \in \mathcal{T}$, define

$$[p] \overset{s}{\rightsquigarrow} [p'] \overset{\text{df}}{\Leftrightarrow} (\exists p'' \in P).\ p \overset{s}{\rightsquigarrow} p'' \ \wedge\ p'' \sim_t p'\ , \tag{3.11}$$

In this way, one obtains a TTS over $\mathcal{T}$ on $P/_{\sim_t}$.

Proof: To begin with, we have to show that the above notion of transition relation on $P/_{\sim_t}$ is well-defined. Let therefore be $p, p' \in P$ and $s \in \mathcal{T}$ such that $[p] \overset{s}{\rightsquigarrow} [p']$, and assume that there exist $q, q' \in P$ such that $p \sim_t q$ and $p' \sim_t q'$. We have to show that also $[q] \overset{s}{\rightsquigarrow} [q']$ holds.

By definition, $[p] \overset{s}{\rightsquigarrow} [p']$ means that there exists some $p'' \in P$ such that $p'' \sim_t p'$ and $p \overset{s}{\rightsquigarrow} p''$. From this last point, by the assumption $p \sim_t q$, we can deduce that there exists $q'' \in P$ such that $q \overset{s}{\rightsquigarrow} q''$ and $p'' \sim_t q''$. Since we know that $p'' \sim_t p' \sim_t q'$, and by the transitivity of $\sim_t$, we therefore obtain $q'' \sim_t q'$. Hence we have shown that $q \overset{s}{\rightsquigarrow} q''$ for some $q'' \in P$ such that $q'' \sim_t q'$, in other words $[q] \overset{s}{\rightsquigarrow} [q']$, as desired. Hence, $\rightsquigarrow$ is well-defined on $P/_{\sim_t}$. We now have to check the axioms for TTSs.

For (Determinacy), let $p, p_1, p_2 \in P$, $t \in \mathcal{T}$, and assume $[p] \overset{s}{\rightsquigarrow} [p_1]$ and $[p] \overset{s}{\rightsquigarrow} [p_2]$. Hence, by definition,

$$p \overset{s}{\rightsquigarrow} p_1' \ \wedge\ p_1' \sim_t p_1$$
$$p \overset{s}{\rightsquigarrow} p_2' \ \wedge\ p_2' \sim_t p_2$$

Since $\langle P, \mathcal{T}, \rightsquigarrow\rangle$ is a TTS and in particular satisfies (Determinacy), it must hold that $p_1' = p_2'$ and so $p_1 \sim_t p_1' = p_2' \sim_t p_2$. Therefore, due to the transitivity of $\sim_t$, $p_1 \sim_t p_2$ and hence $[p_1] = [p_2]$.

As for (ZeroDelay), let $p \in P$. By assumption, we have that $p \overset{0}{\rightsquigarrow} p$, and by the reflexivity of $\sim_t$, we get $p \sim_t p$, sufficing to establish $[p] \overset{0}{\rightsquigarrow} [p]$.

Finally, let $p, p' \in P$, $t, u \in \mathcal{T}$ and assume $[p] \overset{s+u}{\rightsquigarrow} [p']$. By definition, this means that there exists $p'' \in P$ such that

$$p \overset{s+u}{\rightsquigarrow} p'' \ \wedge\ p'' \sim_t p'$$

By (Continuity), this implies that there exists $r \in P$ such that $p \overset{s}{\rightsquigarrow} r \overset{u}{\rightsquigarrow} p''$. Obtaining $r \sim_t r$ from the reflexivity of $\sim_t$, this means that $[p] \overset{s}{\rightsquigarrow} [r]$. Furthermore, by definition, we obtain $[r] \overset{u}{\rightsquigarrow} [p']$. Conversely, assume that $[p] \overset{s}{\rightsquigarrow} [r] \overset{u}{\rightsquigarrow} [q]$ for $p, q, r \in P$ and $s, u \in \mathcal{T}$. The definition applied twice yields

$$p \overset{s}{\rightsquigarrow} p' \sim_t r$$
$$r \overset{u}{\rightsquigarrow} r' \sim_t q$$

Since $p' \sim_t r$ and $r \overset{u}{\rightsquigarrow} r'$, there exists some $p'' \in P$ such that $p' \overset{u}{\rightsquigarrow} p''$ and $p'' \sim_t r' \sim_t q$. Axiom (Continuity) and the transitivity of $\sim_t$ now yield $p \overset{s+u}{\rightsquigarrow} p'' \sim_t q$, which means nothing but $[p] \overset{s+u}{\rightsquigarrow} [q]$, concluding the proof. □

Example 3.29

1. In the example with the two states p_e and p_o, since they are bisimilar, the quotient modulo $\sim_t$ has just one state which has a $\overset{t}{\rightsquigarrow}$-loop for each $t \in \mathbb{N}$.

2. Applying the last two propositions, the quotient $\text{TeCCS}_{/\sim_t}$ of TeCCS agents modulo the largest time bisimulation is also a TTS $\langle \text{TeCCS}_{/\sim_t}, \mathbb{N}, \rightsquigarrow \rangle$. ◊

Remark 3.30

When considering actual calculi like TeCCS which do not just deal with the timing behaviour of processes but also take into account their action transitions, it is necessary to combine time bisimulation with standard bisimulation for action transitions, as is done, for example, in [MT90] for TeCCS (where the resulting notion of equivalence is further shown to be a congruence). Note that this implies that processes bisimilar in the combined sense are equivalent with respect to both standard action bisimulation and time bisimulation. The crucial point for this combined notion of bisimulation is that processes deemed to be equivalent not only have to have the same (action and time) transitions but the respective successor processes must again (co-inductively) satisfy that very same property.

For precisely this reason, it is not enough to simply have two separate bisimulations for action transitions and for time transitions, i.e., considering processes equivalent if they are bisimilar with respect to action transitions and if they are time bisimilar. The coinductive properties satisfied in this case are not strong enough: after one step, say a time transition, one only obtains processes which are guaranteed to be again time bisimilar but no knowledge about the capabilities for action transitions can be derived. The analogous problem occurs after an action transition: no guarantees about the timing behaviour can be given.

The following example will show two transition systems with both action and time transitions which are bisimilar for the separate bisimulations but not in the combined sense, thus establishing that the combined notion is strictly finer. Assuming $\mathcal{T} = \mathbb{N}$ (again only for simplicity), consider the processes p and q as given by the two transition systems in Figure 3.1 whose time transitions are easily seen to form TTSs.

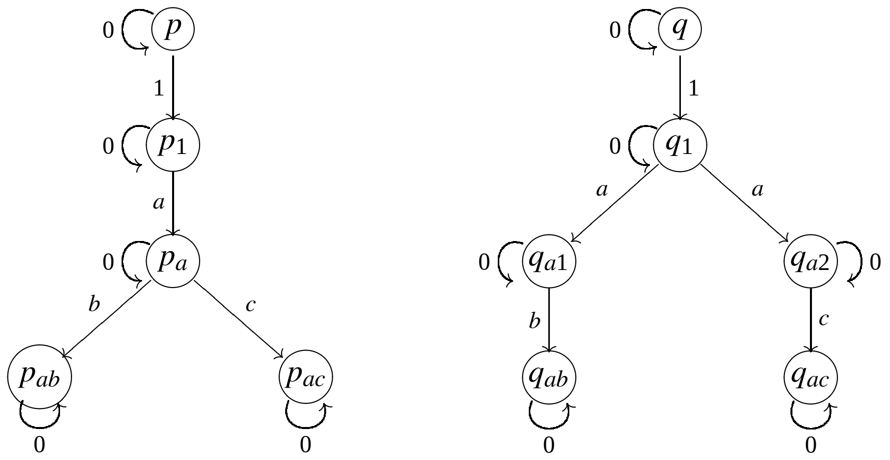


Figure 3.1: Two example transition systems

The states p and q are trivially action bisimilar since neither has any action transitions. Furthermore, the relation $\{\langle p, q\rangle, \langle p_1, q_1\rangle\}$ is a time bisimulation: both p and q have 0-loops and the $\overset{1}{\rightsquigarrow}$-transition of p to p_1 is matched by the transition from q to q_1, and vice versa, and the resulting pair is contained in the relation, both of which again only have 0-loops as time transitions. However, it is well-known that p_1 and q_1 are not action bisimilar, hence they cannot be bisimilar in the combined sense. As a consequence, p and q cannot be bisimilar in the combined sense either: any combined bisimulation containing the tuple $\langle p, q\rangle$ would also have to contain the tuple $\langle p_1, q_1\rangle$. Hence, this shows that the combined notion is strictly finer. $\Diamond$

3.2.2 TTSs as Partial Monoid Actions

We can now introduce the notion of (right) *partial monoid action* which plays a very prominent role in our mathematical account of timed processes: it will turn out that such actions are the same as TTSs. Note that we again write the monoid operations additively without assuming commutativity. Partial monoid actions are generalisations of the notion of partial *group* action [Exe98, KL02] (usually presented as left actions), and the ones we are going to introduce are a slight variation on the ones from [MS02] (again, modulo switching from left to right actions).

Definition 3.31
Let X be a set and $M = \langle M, +, 0\rangle$ be a monoid. Then a *partial (right) monoid action of M on X* is a *partial* function $\alpha : X \times M \rightharpoonup X$ satisfying the following two axioms for all $x \in X$ and $m, n \in M$

$$\alpha(x,0) \simeq x \tag{3.12}$$

$$\alpha(\alpha(x,m),n) \simeq \alpha(x,m+n) \tag{3.13}$$

Usually, we write $x * m$ instead of $\alpha(x,m)$, and simply say that X is a *partial (right) M-set*, when the action α is clear from the context. The set X is also known as the *carrier* of the partial action α. ■

Note that, in (3.12), one could replace Kleene equality $\simeq$ by the standard equality $=$ since the expression x is always defined, i.e., both sides of the equation up to $\simeq$ always have to be defined anyway. In contrast, (3.13) is in general only up to Kleene equality, allowing for 'really partial' partial actions. One important consequence of (3.13) is that

$$(x * (m+n))\downarrow \Rightarrow\ (x * m)\downarrow \tag{3.14}$$

by the definition of Kleene equality. When M is only a semigroup, one can drop (3.12) and get an appropriate notion of partial *semigroup* action. Demanding that M is a group, and that α is a total function, one obtains the well-known notion of a *group action*, see, e.g., [Lan93]. The difference to the partial monoid actions of [MS02] is that we use a stronger axiomatisation: in place of our axiom (3.13), they use the following, weaker one, when translated to our notation of additive right actions:

$$(x*m)\downarrow \Rightarrow \ (x*m)*n \simeq x*(m+n) \tag{3.15}$$

This, in turn, is stronger than the corresponding axiom for partial group actions found in [KL02], which arise by restricting total (group) actions to subsets of their carrier[8]:

$$(x*m)*n \sqsubseteq_{\sim} x*(m+n) \tag{3.16}$$

We are now going to present some examples illustrating our notion of partial monoid actions.

Example 3.32

1. The monoid addition $+$ (from the right) is a partial action of M on M itself. This is an immediate consequence from the monoid axioms. Moreover, $+$ even satisfies (3.13) up to standard equality, not just Kleene equality: it is a *total* monoid action, and will be discussed later on from a different point of view.

2. Assume $\mathcal{T} = \langle \mathcal{T}, +, 0\rangle$ is a time domain. Then the partial subtraction function from (3.4) defines a partial action of $\mathcal{T}$ on itself.

   Proof: As was shown in Proposition 3.17, 0 is the least element with respect to $\leq$. Hence, $0 \leq u$ always holds and therefore, $u - 0$ is always defined; it also immediately follows that $u - 0 = u$, establishing (3.12).

   Let now be $s, t, u \in \mathcal{T}$. We have to show

   $$(s-t)-u \simeq s-(t+u)$$

   Since Kleene equality $\simeq$ is defined in terms of two Kleene implications $\sqsubseteq_{\sim}$, we establish these two separately. Assume therefore $((s-t)-u)\downarrow$. Consequently, we obtain $(s-t)\downarrow$, by definition meaning that we have $s = t + v$ for some unique $v \in \mathcal{T}$, and so $s - t = v$. Furthermore, substituting this in $((s-t)-u)\downarrow$, we get $(v-u)\downarrow$, which again means that $v = u + w$ for a unique $w \in \mathcal{T}$, and $v - u =$

[8] We would like to thank M. Lawson for pointing out this intuition to us via email.

$(s-t)-u=w$. Putting these results together, we obtain $s=t+v=t+(u+w)=(t+u)+w$, in other words, $(t+u)\leq s$ and $s-(t+u)=w=(s-t)-u$, as we had to show.

Conversely, suppose $s-(t+u)\downarrow$. This, by definition, means $t+u\leq s$ and $s=(t+u)+w$ for the unique $w\in\mathcal{T}$ such that $w=s-(t+u)$. Using associativity, we get $s=(t+u)+w=t+(u+w)$ and so $t\leq s$. Consequently, $(s-t)\downarrow$ and $s-t=u+w$. Analogously, one obtains $u\leq u+w=s-t$ and $(s-t)-u=(u+w)-u=w$, showing $s-(t+u)=w=(s-t)-u$; in particular $((s-t)-u)\downarrow$, establishing the second Kleene implication. $\square$

3. Using the same argument as before, it can be shown that

$$\mathbb{N}\times\mathbb{N}^2\rightharpoonup\mathbb{N},\ \langle m,\langle n_1,n_2\rangle\rangle\mapsto m-(n_1+n_2)$$

is a partial $\mathbb{N}^2$-action on $\mathbb{N}$. This can be generalised to any commutative time domain $\mathcal{T}$ to obtain a $\mathcal{T}^n$-action on $\mathcal{T}$. $\Diamond$

Note that the truncated subtraction $\dot{-}$ from Definition 3.14 is merely a suitably 'totalised' version of the partial subtraction (filling in 0 in undefined places). We can now prove the following result explaining our interest in partial monoid actions:

Theorem 3.33
Each TTS $\langle P,\mathcal{T},\rightsquigarrow\rangle$ is equivalent to a partial $\mathcal{T}$-action on P, and vice versa, the correspondence being given by

$$p\overset{t}{\rightsquigarrow}p'\Leftrightarrow p'=p*t \tag{3.17}$$

Proof: Let $\langle P,\mathcal{T},\rightsquigarrow\rangle$ be a TTS. Because of (Determinacy), given $p\in P$ and $t\in\mathcal{T}$, there is at most one process $p'\in P$ such that $p\overset{t}{\rightsquigarrow}p'$. This means that we can indeed regard the time transition relation $\rightsquigarrow$ as a partial function $*:P\times\mathcal{T}\rightharpoonup P$, and its value is given as in (3.17). Furthermore, (ZeroDelay) states that always $p*0=p$, for each process $p\in P$, i.e., it precisely corresponds to axiom (3.12) for partial actions. Finally, (Continuity) states that $p*(t+u)$ must be defined if and only if both $p*t$ is defined and $(p*t)*u$ are defined, and moreover, they have to be equal, precisely as expressed in (3.13) for partial actions.

In the converse direction, (3.17) shows how to define a transition relation $\rightsquigarrow$ on the carrier P of a partial action $* : P \times \mathcal{T} \rightharpoonup P$. From the type of $*$, it follows that, for each $p \in P$ and $t \in \mathcal{T}$, there is at most one possible $p' \in P$ such that $p' = p * t$, and so $\rightsquigarrow$ satisfies (Determinacy). Finally, (ZeroDelay) and (Continuity) are just consequences of (3.12) and (3.13), respectively. □

Had we taken the definition of partial monoid actions as in [MS02], we would not have been able to prove this equivalence, or rather, we would have needed to alter the definition of TTSs analogously: in place of (Continuity), we would have had to use 'weak' continuity, corresponding to (3.15):

$$(\forall p''). \; p \overset{t}{\rightsquigarrow} p'' \;\Rightarrow\; (p \overset{t+u}{\rightsquigarrow} p' \Leftrightarrow p \overset{t}{\rightsquigarrow} p'' \overset{u}{\rightsquigarrow} p')$$

which is quite unlike all axioms for timed processes. We therefore believe that our definition of partial monoid actions, although non-standard, is more appropriate with respect to our objective of a mathematical and conceptual study of timed processes.

Example 3.34

Since we have now obtained an alternative characterisation of TTSs as partial monoid actions, we can regard the two standard examples of partial monoid actions, viz., the monoid composition and the partial subtraction of a time domain, as TTSs and study their properties with respect to time bisimulation.

1. For an arbitrary monoid $\langle M, +, 0\rangle$ (hence including all time domains), consider the partial M-action on M itself given by the addition $+$. Using (3.17), we can translate it into a TTS with M as the set of processes, yielding

   $$m \overset{m''}{\rightsquigarrow} m' \;\Leftrightarrow\; m' = m + m'' \tag{3.18}$$

   after replacing the Kleene equality $\simeq$ from (3.17) with the standard equality $=$ since the expression $m + m''$ on the right-hand side is always defined. With this TTS on M at hand, an evident question to ask is when two elements $m, m' \in M$ are time bisimilar. For this, consider the following relation

   $$R_{\langle m,m'\rangle} \overset{\text{df}}{=} \{\langle m+n, m'+n\rangle \mid n \in M\}$$

   We claim that this is always a time bisimulation.

Proof: Let $\langle a, b\rangle \in R_{\langle m,m'\rangle}$. By the definition of $R_{\langle m,m'\rangle}$ this means that there exists $n \in M$ such that $a = m+n$ and $b = m'+n$. Now assume $a \overset{m''}{\rightsquigarrow} a'$ which is equivalent to $a' = a+m'' = (m+n)+m'' = m+(n+m'')$, using the associativity of $+$. Now, choosing $b' \overset{\text{df}}{=} b+m'' = (m'+n)+m'' = m'+(n+m'')$, we obtain from (3.17), applied to this case, that $b \overset{m''}{\rightsquigarrow} b'$, and by the definition of $R_{\langle m,m'\rangle}$, we also obtain $\langle a', b'\rangle \in R_{\langle m,m'\rangle}$ and analogously in the symmetric case, establishing $R_{\langle m,m'\rangle}$ as a time bisimulation. □

Hence, for each pair $\langle m, m'\rangle$, there exists at least one time bisimulation containing it, viz., $R_{\langle m,m'\rangle}$, meaning that $\sim = M \times M$, the complete cartesian product, the largest possible equivalence relation. Put differently, $M_{/\sim} = \{\star\} = 1$ in this special case: the quotient *collapses*, all states are time bisimilar.

2. Let $\langle \mathcal{T}, +, 0\rangle$ be a time domain. We have already explained how the relative inverses $u - t$ for $t \le u$ in the induced order induce a partial monoid action, the partial subtraction action $\mathcal{T} \times \mathcal{T} \rightharpoonup \mathcal{T}$. Using (3.17), this then defines a TTS on $\mathcal{T}$ as follows, for $t, u, u' \in \mathcal{T}$:

$$u \overset{t}{\rightsquigarrow} u' \overset{\text{df}}{\Leftrightarrow} u' \simeq u * t \simeq u - t \tag{3.19}$$

We can again ask what time bisimulation on this TTS on $\mathcal{T}$ means and, where for $+$ everything was equivalent, here we obtain the other extreme: no two distinct elements of $\mathcal{T}$ are equivalent, for all $t, u \in \mathcal{T}$, $t \neq u$ implies $t \not\sim u$.

Proof: It trivially holds that $0 \sim 0$. Moreover, for $t > 0$, $t \not\sim 0$: $t \overset{t}{\rightsquigarrow} 0$ yet $0 \overset{t}{\not\rightsquigarrow}$. Therefore, $[0]_\sim = \{0\}$, i.e., 0 is only bisimilar to itself. Let now $t, u \in \mathcal{T}$ such that $t \neq u$. Then $t \overset{t}{\rightsquigarrow} 0$ but since $t \neq u$, it cannot hold that $u \overset{t}{\rightsquigarrow} u' \simeq u - t$ and $0 \sim u'$: this would imply $u' = 0$, and so we would obtain $u - t = 0$, and consequently $t = u$. Hence $t \not\sim u$. □

Using the contra-position $t \sim u \Rightarrow t = u$ of the claim, this last result can also be rephrased as $\sim = \mathrm{Id}_{\mathcal{T}}$, i.e., $\sim$ is the smallest possible equivalence relation. ◊

Considering the TTSs induced by the two standard monoid actions on the monoid (or time domain) itself again underlines their intuitively dual nature: whereas the one, the composition $+$, induces a TTS in which all states are equivalent, the partial subtraction $-$ induces a TTS where no distinct points in time are time bisimilar.

Usually, when introducing 'objects with (algebraic) structure,' one defines homomorphisms of such objects simply as functions respecting, or preserving, the extra structure. This is exactly the case for partial monoid actions:

Definition 3.35

Let M be a monoid and let α_i be partial M-actions on X_i, $i \in \{1,2\}$, the α_i written as $*_i$, respectively. A *homomorphism from* $*_1$ *to* $*_2$ (or, less precisely, from X_1 to X_2) is a (total) function $f : X_1 \to X_2$ such that for $x \in X_1$ and $m \in M$

$$f(x *_1 m) \simeq (fx) *_2 m \tag{3.20}$$

Alternatively, such maps are also called *equivariant* [MS02]. With this notion of homomorphism, partial M-actions form a category M**–pAct**. ■

When a partial action is actually a total function $X \times M \to X$, we call it a *total monoid action*; the prototypical example is given by monoid addition $+$. Homomorphisms of total monoid actions are defined using the same equation (3.20) as in the partial case: since all involved functions are total, $\simeq$ and $=$ coincide. In the remainder of this thesis, whenever M is a monoid, denote by M**–Act** the full subcategory of M**–pAct** consisting of total right M-actions and their homomorphisms.

Remark 3.36

Using the correspondence from Theorem 3.33 and the definition (3.20) of homomorphism of partial action in Definition 3.35, we could obtain a corresponding notion of morphism of TTSs as follows. Given two TTSs $\langle P_i, \mathcal{T}, \rightsquigarrow_i \rangle$, a morphism between them is a (total!) function $f : P_1 \to P_2$ such that, after translating the TTSs into partial $\mathcal{T}$-actions $P_1 \times \mathcal{T} \rightharpoonup P_i$, f is a homomorphism of the corresponding partial $\mathcal{T}$-actions. In concrete terms, using the correspondence (3.17), this means that $f : P_1 \to P_2$ is a morphism of TTSs if and only if the following holds, for all $p \in P_1$ and $t \in \mathcal{T}$:

$$(\forall q' \in P_2).\ ((\exists p' \in P_1).\ p \overset{t}{\rightsquigarrow}_1 p' \ \wedge\ fp' = q') \ \Rightarrow\ (fp \overset{t}{\rightsquigarrow}_2 q')$$
$$(\forall q' \in P_2).\ (fp \overset{t}{\rightsquigarrow}_2 q') \ \Rightarrow\ ((\exists p' \in P_1).\ p \overset{t}{\rightsquigarrow}_1 p' \ \wedge\ fp' = q')$$

Note that, somewhat sloppily, this could be written as

$$p \overset{t}{\rightsquigarrow}_1 p' \;\Leftrightarrow\; fp \overset{t}{\rightsquigarrow}_2 fp' \tag{3.21}$$

The reason for our not introducing this notion of homomorphism of TTSs is because it is slightly odd, when considered on its own. Traditionally, a homomorphism of transition systems has to satisfy only the left-to-right direction of (3.21): if there is a transition possible in the domain of the homomorphism, then there should be a corresponding transition in the codomain (cf. the more general case of directed graphs). Regardless of that, using (3.21) as it is, we could extend Theorem 3.33 to an equivalence of categories. ◊

Note that despite using partial functions describing the actions, we homomorphisms are *total* functions. Apart from the common mathematical practise to do so, there is also an intuitive reason for this: in (3.20), which is the natural property a homomorphism from $*_1$ to $*_2$ should satisfy, the only 'sources of partiality' should be the partial actions themselves, not the homomorphism f, which is merely a 'transformation'. Also, having the total actions as a *full* subcategory of M**–pAct** is further mathematical evidence for the right choice. On top of that, as was the case for TTS, it also turns out that this particular formalisation, using total functions as homomorphisms rather than partial ones, yields a mathematically more pleasing framework, in particular the treatment of (bisimulation) relations becomes a good deal more natural.

Using the description (3.17) of TTSs as partial actions and the properties of their homomorphisms, we can give the following re-formulation of what it means to be a time bisimulation. This will play an important part when linking up the concrete definition of time bisimulation with the coalgebraic description of bisimulation introduced in [AM89].

Proposition 3.37

Let $\mathcal{T}$ be a time domain and P_i, $i \in \{1,2\}$ be partial $\mathcal{T}$-actions, i.e., TTSs with labels in $\mathcal{T}$, written as $*_i$. A relation $R \subseteq P_1 \times P_2$ is a time bisimulation if and only if $\langle p_1, p_2 \rangle \in R$ implies for all $t \in \mathcal{T}$:

$$(p_1 *_1 t)\downarrow \Leftrightarrow (p_2 *_2 t)\downarrow \tag{3.22}$$

$$(p_1 *_1 t)\downarrow \Rightarrow \langle p_1 *_1 t, p_2 *_2 t\rangle \in R \tag{3.23}$$

Furthermore, each time bisimulation R can be endowed with a canonical partial action $*$ which, for $\langle p_1, p_2\rangle \in R$ and $t \in \mathcal{T}$, is defined as follows:

$$\langle p_1, p_2\rangle * t \overset{\text{df}}{\simeq} \langle p_1 *_1 t, p_2 *_2 t\rangle \tag{3.24}$$

This partial action $*$ satisfies the property that the *projections* $\pi_i : R \to X_i$ are always homomorphisms from $*$ to $*_i$.

Proof: In the one direction, let $R \subseteq P_1 \times P_2$ be a relation satisfying (3.22) and (3.23), let $\langle p_1, p_2\rangle \in R$, and let $t \in \mathcal{T}$. By (3.22), we know that $(p *_1 t)\downarrow \Leftrightarrow (p *_2 t)\downarrow$. Using the correspondence (3.17) between TTSs and partial $\mathcal{T}$-actions, this means the $p_1 \overset{t}{\rightsquigarrow}_1 \Leftrightarrow p_2 \overset{t}{\rightsquigarrow}_2$, and furthermore, $p_1 \overset{t}{\rightsquigarrow}_1 (p_1 *_1 t) \Leftrightarrow p_2 \overset{t}{\rightsquigarrow}_2 (p_2 *_2 t)$. Assume therefore $(p_1 *_1 t)\downarrow$. Then, by (3.22), also $(p_2 *_2 t)\downarrow$, and moreover, by (3.23), this implies $\langle p_1 *_1 t, p_2 *_2 t\rangle \in R$. Again using (3.17), this corresponds to the fact that, if $p_1 \overset{t}{\rightsquigarrow} p_1 * t$, then also p_2 can perform a $\overset{t}{\rightsquigarrow}_2$-transition, and the resulting states are again related by R, i.e., precisely as stated in the definition of a time bisimulation.

For the converse, assume that $R \subseteq P_1 \times P_2$ is a time bisimulation. Since R is a time bisimulation, we know that $\langle p_1, p_2\rangle \in R$ implies in particular, for all $t \in \mathcal{T}$, that $p_1 \overset{t}{\rightsquigarrow}_1 \Leftrightarrow p_2 \overset{t}{\rightsquigarrow}_2$. Under the correspondence (3.17), this is precisely equivalent to (3.22). Furthermore, assume that $p_1 \overset{t}{\rightsquigarrow}_1 p_1'$. Then we know that $p_1' = p_1 *_1 t$, and the analogous property holds for $p_2 \overset{t}{\rightsquigarrow}_2 p_2'$. Since R is a time bisimulation, we know that then $\langle p_1', p_2'\rangle \in R$, in other words $\langle p_1 *_1 t, p_2 *_2 t\rangle \in R$, as stated in (3.23).

Finally, if R is a time bisimulation, define the action $*$ of $t \in \mathcal{T}$ on $\langle p_1, p_2\rangle$ as in (3.24). This is well-defined: by (3.22), either both $p_i *_i t$ are defined or both are undefined. Since R also satisfies (3.23), we moreover know that $(\langle p_1, p_2\rangle * t) \in R$, so we obtain a partial function $* : R \times \mathcal{T} \rightharpoonup R$. It is then routine to verify that $*$ is indeed a partial $\mathcal{T}$-action. $\square$

Thus, if R is a time bisimulation, Prop. 3.37 states that R is itself a partial monoid action because it is *closed* with respect to the partial monoid actions $*_1$ and $*_2$. Again, this is a mathematically very pleasing fact: the 'right' relations between 'sets with structure' (the partial monoid actions) should be the ones which are closed with respect to the extra structure, and which themselves can be regarded as carrying the extra structure, i.e., which are also objects in the corresponding category. Furthermore, note how the rewritten definition automatically incorporates (Determinacy): $p_1 \overset{t}{\leadsto}_1 p_1'$ can only hold for the unique $p_1' = p_1 *_1 t$, hence the existential quantification is no longer required.

3.3 Delay Operators and Total Monoid Actions

Up to now, we have been concerned with a *model* for timed processes, and we have presented TTSs as an adequate formalisation of the properties of such processes. We now turn our attention to a slightly more language-oriented view by considering *delay operators*. What we want to model is an operation on a TTS $\langle P, \mathcal{T}, \leadsto \rangle$ which takes a time $t \in \mathcal{T}$ and a process $p \in P$, and the result should again be a state in P, which we will write as $t \cdot p$. The intuition is that $t \cdot p$ should denote the process which, after an *initial waiting period* of t units of time, behaves like p, very similar to the standard action prefix $\alpha.p$ to be found in all important process specification languages.

The concept should be reasonably clear, yet the question remains which, if any, axioms one should impose on such a delay operator. As a very important point, in particular to distinguish delaying from letting time pass, a delay operator should be modelled by a *total* function of the type $\mathcal{T} \times P \to P$: we can think of no naturally arising case when a process, on the level of TTSs, should not be delayable. Note that this is fundamentally different from (Maximal Progress) expressing that certain (usually internal) *computations* of a process cannot be delayed.

Apart from totality, there are other properties delay operators intuitively should satisfy, related to the monoid structure on $\mathcal{T}$[9]. For the first, recall that $0 \in \mathcal{T}$ denotes the absence of duration, and so delaying by 0 units of time should mean adding *no* initial idling phase: in other words, we want that $0 \cdot p = p$.

Moreover, delaying twice, first by t, then by u, should be the same as delaying by $u + t$. The twofold delay first enforces t units of time of initial delay, on top of which u units of, *again initial*, delay are then added, so to speak *from the outside in*:

[9]This should not come as a surprise to the attentive reader.

we should get a process with an initial waiting period of u units, followed by another idling period of t units of time, i.e., delaying operates like a *stack* where always the top element is manipulated. This is depicted in Figure 3.2, where the box named p describes (the idling capabilities of) a process p, to which subsequently an initial delay of t units, then of u units of time are added. However, this precisely corresponds to

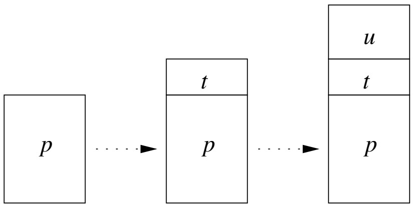


Figure 3.2: The 'delay-as-stacking' intuition.

adding a waiting period of $u+t$ units of time, recalling the interpretation of the monoid addition. This leads to the following definition:

Definition 3.38

Given a TTS $\langle P, \mathcal{T}, \rightsquigarrow \rangle$, a *delay operator* on P is a function of type $\mathcal{T} \times P \to P$, usually written as $t \cdot p$, which is a (left) *total action* of the monoid $\mathcal{T}$ on the set P of processes, i.e., it satisfies the following equations, for all $t, u \in \mathcal{T}$:

$$0 \cdot p = p \tag{3.25}$$

$$u \cdot (t \cdot p) = (u+t) \cdot p \tag{3.26}$$

If there is such a delay operator on P, we also speak of a *TTS with delay*. ■

Note that the left actions used for delaying have to satisfy a different additivity axiom, compared to the right partial actions equivalent to TTSs. This is done in order to capture the intuition that the delays are stacked, rather than queued, as was the case for time transitions. Furthermore, although a delay operator is defined with respect to a TTS, its axioms make no reference to the time transitions, i.e., *any* total monoid action is a delay operator.

Example 3.39

We will now consider some of the previously mentioned examples of TTSs and consider delay operators on them.

1. Let $\langle M, +, 0 \rangle$ be a monoid. We have seen that the addition $+$ induces a TTS on M. First of all, the monoid addition $+$ on the left(!), i.e., $m \cdot n = m + n$, is a delay operator on M. Furthermore, since in the TTS, any two states are time bisimilar, the TTS obtained by quotienting with respect to bisimulation has only one state, i.e., $M_{/\sim} = 1$. This implies that the unique (constant) function of the type $M \times 1 \cong M \to 1$ (because 1 is terminal in **Set**) is a delay operator.

2. In contrast, the partial subtraction $-$ induces a TTS on $\mathcal{T}$ such that no two distinct elements of $\mathcal{T}$ are time bisimilar. Consequently, a delay operator on $\mathcal{T}_{/\sim}$ is still nothing but a total monoid action of $\mathcal{T}$ on itself, nothing is gained by quotienting. This makes addition $+$ an example also in this case.

3. As a further specialisation, consider the time domain $\mathbb{N}^2$, and recall that we can obtain a partial $\mathbb{N}^2$-action on $\mathbb{N}$, using the partial subtraction (cf. Section 3.2.2). In this situation, we obtain a delay operator on $\mathbb{N}$, i.e., a total $\mathbb{N}^2$-action, on $\mathbb{N}$ by defining
$$\langle \langle n, m \rangle, k \rangle \;\mapsto\; n + m + k$$
as is easily checked.

4. Given a time domain $\mathcal{T}$, $t \in \mathcal{T}$, and $\vec{u} \stackrel{\mathsf{df}}{=} \langle u_1, \ldots, u_n \rangle \in \mathcal{T}^n$, defining
$$t \cdot \vec{u} \stackrel{\mathsf{df}}{=} \langle t + u_1, \ldots, t + u_n \rangle$$
yields a delay operator on $\mathcal{T}^n$. $\Diamond$

Another ‘example’ of a delay operator should be the *time prefixing* $(t).p$ of TeCCS: its intended meaning is exactly that of delaying processes by an amount t of time. Yet, unfortunately, it is *not* a delay operator on the set of TeCCS terms, for two reasons. First of all, (3.25) does not hold because $(0).p$ is not even a term of the language. Moreover, (3.26) does not hold either: on the level of *syntax*, the two processes $(s).(t).p$ and $(s + t).p$, corresponding to both sides of the equation, are *not* equal. However, as

$(s).(t).p \sim_t (s+t).p$, after augmenting TeCCS with $(0).p$ time prefixes, time prefixing is a delay operator on the set TeCCS$/\sim_t$ of TeCCS terms quotiented by the largest time bisimulation; since this would also identify a lot more processes than simply the ones needed to validate (3.26), we will now show how to obtain a delay operator on TeCCS which is almost equal to time prefixing and requires less identifications.

To do so, recall the semantics of time prefixing from [MT90], cf. Figure 2.1. Stressing once more that the syntax of TeCCS does *not* allow $(0).p$ as a process, we can slightly sharpen the SOS rules for time prefixing compared to [MT90], one difference being the previously implicit additional side condition in the leftmost of the following three rules:

$$\text{(3.27)} \qquad \frac{}{(s+t).p \overset{s}{\rightsquigarrow} (t).p}\,(t>0) \qquad \frac{}{(t).p \overset{t}{\rightsquigarrow} p} \qquad \frac{p \overset{s}{\rightsquigarrow} p'}{(t).p \overset{t+s}{\rightsquigarrow} p'}$$

Furthermore, while the original version of the rightmost rule had an $\overset{s+t}{\rightsquigarrow}$-transition in the conclusion, we have changed this to a $\overset{t+s}{\rightsquigarrow}$-transition. According to our interpretation of addition, $t+s$ corresponds to first waiting for t units of time followed by waiting for s units of time, and so the $\overset{t+s}{\rightsquigarrow}$-transition more accurately reflects the intuitive behaviour of time prefixing: in order for $(t).p$ to 'reach' the $\overset{s}{\rightsquigarrow}$-transition from p to p', it first has to 'consume' the t units of added *initial* delay expressed in the time prefix, cf. Figure 3.2.

The problem that $(0).p$ is not even a term of TeCCS can be overcome very easily: simply define a delay operator on TeCCS which maps $\langle 0, p\rangle$ to p, for any term p. As for (3.26), the we take the set TeCCS and simply identify, via *quotienting*, terms of the form $(s).(t).p$ with $(s+t).p$. Note that, in this way, a term of the form $(s).(t).(u).p$ is identified with the term $(s+t+u).p$: nested time prefixes as the topmost operators are 'flattened' by adding from the outside in.

The quotient TeCCS$^+$ obtained in this way then still carries a TTS, which is very easy to see: the construction is similar to the one in Proposition 3.28. Moreover, the map on equivalence classes

$$\langle t, [p]\rangle \;\mapsto\; \begin{cases} [p] & \text{if } t = 0 \\ [(t).p] & \text{if } t > 0 \end{cases}$$

is well-defined and a delay operator on TeCCS$^+$.

If we wanted to extend *the language itself* to allow $(0).p$ as a process and obtain delay operator simply via $\langle t, [p] \rangle \mapsto [(t).p]$ for *all* $t \in \mathcal{T}$, leaving the rules (3.27) as they are would *not* work because it would then be possible to derive the transitions $(0).p \overset{0}{\rightsquigarrow} (0).p$, enforced by (ZeroDelay), yet also $(0).p \overset{0}{\rightsquigarrow} p$, by the second rule for time prefixing, and consequently invalidate (Determinacy): we would also have to identify $(0).p$ and p, leading to a more complex quotient of the TeCCS terms.

Avoiding the above quotient construction, we can alternatively rearrange the grammar for the syntax of the language: distinguishing whether the topmost operator already is time prefixing or not, we only allow time prefixing when this is not the case. Concretely, (part of) the grammar for TeCCS, with only nil (instead of the 0-process, to avoid confusion with the monoid structure), time prefixing, and $+$ looks as follows, d being the start non-terminal, and t still ranging over *non-zero* times in $\mathcal{T}^+ = \mathcal{T} \setminus \{0\}$:

$$
\begin{aligned}
d &::= u \mid (t).u \\
u &::= \mathrm{nil} \mid d + d
\end{aligned}
$$

or equivalently, substituting the right-hand side of u in d:

$$p ::= \mathrm{nil} \mid p + p \mid (t).\mathrm{nil} \mid (t).(p + p)$$

It should be clear that, identifying multiple time prefixes with the 'added up' one, the same terms as in the original grammar are derivable, the only difference being that, in this fashion, one can never have nested time prefixings, thus precisely removing what destroys validity of (3.26). Furthermore, one still obtains a TTS, using the same rules as before. Using this approach, we get a delay operator as follows:

$$
\langle t, p \rangle \;\mapsto\; \begin{cases} p & \text{if } t = 0 \\ \left(\begin{cases} (t).p & \text{if } p \neq (u).q \\ (u+t).q & \text{if } p = (u).q \end{cases} \right. & \text{if } t > 0 \Bigg) \end{cases}
$$

This is well-defined, since each derivable term either has a time prefix as its topmost operator or not, and also (3.26) holds. Moreover, it should be reasonably clear that the two approaches, using a quotient of the original language, or rearranging the grammar, yield essentially the same result: both prohibit nesting of time prefixing[10], and both simply return p as the value of $0 \cdot p$.

Note that this use of total monoid actions to model delaying, as opposed to the partial actions equivalent to TTSs, further stresses the obvious duality between addition and subtraction. On the one hand, $+$ describes delaying, adding more idling potential, in some sense corresponding to a form of 'production'. On the other hand, the (partial) subtraction $-$ describes the effect of time transitions, 'consuming' the idling capabilities of timed processes. Once again, exploiting the monoid structure on the time domain $\mathcal{T}$ allows to derive a way of mathematically expressing properties of an important construct, time-prefixing, which occurs pervasively in the literature on timed processes.

3.4 Biactions

In the last two sections, two distinct notions of monoid actions of a time domain $\mathcal{T}$ have been introduced: firstly, the partial actions of $\mathcal{T}$ on a set P of processes were shown to be the same as TTSs on P; secondly, relative to such a TTS, total monoid actions on the set P of states were used to introduce the concept of a delay operator, yet no restrictions were imposed as to the interplay between the two kinds of actions.

Conceptually, a TTS with delay describes a structure consisting of a set (of states) carrying both a partial and a total $\mathcal{T}$-action (the time transition relation and the delay operator, respectively). This suggests to combine the two notions of monoid actions into one, which we shall call *$\mathcal{T}$-biaction*. These $\mathcal{T}$-biactions provide a minimal account of timed processes which can be delayed and perform time transitions, as defined by the two actions. In order to ensure that the actions really correspond to delaying and idling, we have to impose an axiom on their interplay, which simply expresses the duality between delaying and time transitions, adding and removing idling potential.

[10]The first approach only removes such nestings on the *outside* of a term, while the second removes them *anywhere* in a term. In order to obtain a complete equivalence, one would have to close the quotient under substitutions, i.e., say that one identifies $(s).(t).p$ with $(s+t).p$ *in all contexts* of the language, thus identifying, e.g., the terms $\mathrm{nil} + (s).(t).p$ and $\mathrm{nil} + (s+t).p$.

Definition 3.40

Let $\mathcal{T}$ be a time domain. A $\mathcal{T}$-*biaction* consists of a set P which is the carrier of both a (right) partial action, written as $p * t$, and a (left) total action, written as $t \cdot p$, such that, for all $t, u \in \mathcal{T}$ and $p \in P$,

$$(t \cdot p) * t \simeq p \tag{3.28}$$

As usual, we confuse biaction and carrier, simply saying that P is the biaction. The *homomorphisms* of $\mathcal{T}$-biactions are the evident ones: maps which are equivariant with respect to both actions. Thus, we obtain a category $\mathcal{T}$**–BiAct**. ■

Observe that (3.28) could also be written as a traditional equality since p on the right-hand side is always defined. Note how (3.28) finally formalises the duality between delaying and idling mentioned on several occasions: an initial delay of t units of time is 'consumed' by idling for t units of time. Also note the asymmetry of this property: nothing is stated regarding first letting time pass and then adding delay; this is hardly a surprise: after all, idling might change the state from p to some q and there need not be any connection between the transitions of p and q.

Axiom (3.28) has two important consequences, showing that the single equation indeed captures how delaying and time passing should interact intuitively:

Proposition 3.41

Let $\mathcal{T}$ be a time domain, and let P be a $\mathcal{T}$-biaction. Then, for all $t, u \in \mathcal{T}$ and $p \in P$:

1. $u \leq t \;\Rightarrow\; (t \cdot p) * u \;\simeq\; (t - u) \cdot p$
2. $t \leq u \;\Rightarrow\; (t \cdot p) * u \;\simeq\; p * (u - t)$

Proof:

1. Let $t, u \in \mathcal{T}$ and $p \in P$, and assume $u \leq t$. This is equivalent to $(t - u)\downarrow$ and also implies that $t = u + (t - u)$, by definition of the relative inverses. Thus, we calculate

   $$(t - u) \cdot p \simeq (u \cdot ((t - u) \cdot p)) * u \simeq ((u + (t - u)) \cdot p) * u \simeq (t \cdot p) * u$$

   using the fact that $\cdot$ is a total monoid action, and (3.28).

2. Let $t, u \in \mathcal{T}$ be such that $t \leq u$, or equivalently, $(u - t)\downarrow$. This implies, by the definition of the relative inverses, that $u = t + (u - t)$ and so:

$$p * (u - t) \simeq ((t \cdot p) * t) * (u - t) \simeq (t \cdot p) * (t + (u - t)) \simeq (t \cdot p) * u$$

using the fact that $*$ is a partial monoid action, and (3.28). $\square$

Note that, if $\mathcal{T}$ is linear, the two mentioned cases in Proposition 3.41 are the only ones, i.e., (3.28) can then equivalently be expressed as

$$(t \cdot p) * u \simeq \begin{cases} (t - u) \cdot p & \text{if } u \leq t \\ p * (u - t) & \text{if } t \leq u \end{cases} \tag{3.29}$$

Observe that, if $t = u$, both results are equal to p, hence the overlapping case distinction causes no trouble. Using linearity, we can simplify (3.29) even further, obtaining exactly the equation that was used in [Kic02a] to *define* biactions:

$$(t \cdot p) * u \simeq (t \dot{-} u) \cdot (p * (u \dot{-} t)) \tag{3.30}$$

To see that (3.30) and (3.29) are equivalent, we simply note that always at least one of the two expressions $t \dot{-} u$ and $u \dot{-} t$ is equal to 0; if both are, $t = u$ must hold with result p, precisely as stated in (3.28).

Furthermore, note that (3.28) does not specify what happens to $(t \cdot p) * u$ in case $t \| u$ holds. Since both $t \dot{-} u$ and $u \dot{-} t$ are then equal to 0, (3.30) might suggest to use p as the value in that case, yet this violates (Continuity): let $t, u \in \mathcal{T}$ such that $t \| u$; additionally, assume $t \| u + t$ (this is necessarily true for $C^\star$); then we would obtain

$$p = (t \cdot p) * (u + t) = ((t \cdot p) * u) * t = p * t$$

where the last expression need not even be defined. Essentially the same argument prohibits the value being $t \cdot p$, i.e., the state cannot simply be left unchanged.

The only reasonable solution seems to leave the value unconstrained, since nothing is prescribed by (3.28), and none of the obvious choices result in good definitions. Intuitively, we can also justify it: the expression $t \cdot p$ denotes the process p with and initial idling period attached to it, cf. Figure 3.2; the effect of the partial action of

u should be to let time pass, yet from the way $t \cdot p$ is constructed, this means that (at least part of) the initial delay of p has to pass. However, this would imply that the idling duration has to be related to t. Since $t \| u$, the process $t \cdot p$ simply cannot perform a $\stackrel{u}{\rightsquigarrow}$-transition. This problem will become important for giving a categorical characterisation of biactions. We will now give examples of biactions.

Example 3.42

1. Any time domain $\mathcal{T}$ is itself a $\mathcal{T}$-biaction with respect to partial subtraction (on the right) and addition (on the left).

   Proof: Instantiating (3.28) in this concrete case, we have to show, for all $t, u \in \mathcal{T}$

   $$(t+u) - t \simeq t$$

   yet that was already proved in Lemma 3.21. □

2. Recall the total and partial $\mathbb{N}^2$-actions on $\mathbb{N}$ obtained by suitable adapting the addition and the partial subtraction of $\mathbb{N}^2$. It is trivially verified that these two actions also satisfy (3.28). Thus, we get an $\mathbb{N}^2$-biaction on $\mathbb{N}$.

3. Any of the two ways of how to obtain a delay operator for (variants of) TeCCS, as described in Section 3.3, induces a biaction, the time transitions defining the partial action, and the delay operator as the total action. This follows very easily, because both approaches essentially state that $t \cdot p = (t).p$, modulo the necessary technicalities. Furthermore, the semantics of time prefixing stays pretty much the same, and so we get

   $$t \cdot p = (t).p \stackrel{t}{\rightsquigarrow} p$$

   yet that is precisely what (3.28) looks like in this case. ◊

Remark 3.43

Translating from the partial action notation to the TTS notation, and simply treating the delay operator $t \cdot p$ as a *syntactic construct* $(t)p$ (the similarity with the TeCCS notation for time prefixing is deliberate!), (3.28) states that

$$(t)p \stackrel{t}{\rightsquigarrow} p$$

i.e., the middle rule of (3.27). Similarly, the two consequences of (3.28) stated in Proposition 3.41, exploiting the fact that $u \leq t$ is equivalent to $t = s + u$ for $u = t - s$ and the symmetric result, viz., $t \leq u$ is equivalent to $u = t + s$ for $s = u - t$, translate to

$$(s+u)p \overset{s}{\rightsquigarrow} (u)p \qquad \frac{p \overset{s}{\rightsquigarrow} p'}{(t)p \overset{t+s}{\rightsquigarrow} p'}$$

also expressing $(p * s)\downarrow$ as $(\exists p' \in P).\ p \overset{s}{\rightsquigarrow} p'$ for the second 'rule' and so, we obtain that the operator $(\_)_{=}$ has precisely the same 'operational semantics' as the TeCCS delay operator. In this way, we can treat a biaction like a (mini-)calculus of timed processes with a single operator and fixed operational semantics, and all states in P as constants, their semantics given by the TTS.

Although it is possible to derive $(0)p = p$ from (Determinacy), (ZeroDelay), and $(0)p \overset{0}{\rightsquigarrow} p$, we do, however, not believe that the two views are equivalent: the biaction-view also demands that $\cdot$ satisfies the other axiom (3.26) of a total monoid action which, as far as we can see, is not a consequence of (3.28) and so not necessarily built into the calculus-view. To obtain an equivalence, one would have to consider a *quotient* of the syntax, quite similar to what was done in the previous section to make time prefixing into a delay operator, only 'the other way round,' starting from an operator with a certain semantics and trying to obtain a biaction. $\Diamond$

Chapter 4

Timed Processes Categorically

In this chapter, we present categorical formulations of all the important concepts related to timed processes that were introduced concretely in the previous chapter. At first, we present several categorical characterisations of total monoid actions, thereby obtaining an abstract description of delay operators, one of the ‘ingredients’ of biactions.

Although we merely recall two of the well-known characterisations from the literature (some more material is contained in Appendix A), they still serve the purpose of preparing the ground for a categorical description of partial monoid actions which, to our knowledge, is new: to this end, we introduce a comonad of *evolutions* and show that its Coalgebras are precisely the partial monoid actions from Section 3.2.2.

We then establish some properties of this *evolution comonad*, in particular that the associated notion of coalgebraic bisimulation [AM89] is precisely time bisimulation from Section 3.2.1, yielding a pleasing match between the concrete and the categorical formulations.

Furthermore, we show how to obtain a simpler description of this comonad in the case of discrete time (where it actually is cofreely generated), substantiating the claim that discrete quantitative and global qualitative time are essentially the same.

Combining the coalgebraic characterisation of partial actions with an algebraic description of total monoid actions by a distributive law (see Section 2.4), we obtain that the *bialgebras* of this distributive law are the biactions introduced in Section 3.4.

4.1 Categorical Descriptions of Total Monoid Actions

In this section, we review several well-known categorical formulations of total monoid actions. Our reasons for doing so are twofold. First of all, total actions are in themselves (part of) the mathematical structure underlying delay operators and biactions: if we wish to treat these important concepts within a categorical framework, we have to have a suitable way of expressing them in categorical terms. Furthermore, the characterisations for the well-established total case guide our search for a corresponding characterisation of partial actions, which, to our knowledge, did not previously exist. Note that we only present two characterisations here, some more material can be found in Appendix A.

4.1.1 Total Monoid Actions as Algebras

The following characterisation of monoid actions as (Eilenberg-Moore) algebras for a monad is well known: for groups, the same construction is contained in [Mac97].

Proposition 4.1
Let $M = \langle M, +, 0\rangle$ be a monoid. Then the mapping $X \mapsto TX \stackrel{\mathsf{df}}{=} M \times X$ extends to a monad $\langle T, \eta, \mu\rangle$, its unit η and multiplication μ defined as follows:

$$\begin{aligned} \eta_X : X \to TX, &\quad x \mapsto \langle 0, x\rangle \\ \mu_X : T(TX) \to TX, &\quad \langle n, \langle m, x\rangle\rangle \mapsto \langle n+m, x\rangle \end{aligned}$$

Furthermore, T**–Alg** $\cong M$**–Act**. □

More abstractly, the monad T from Proposition 4.1 is the monad associated to the adjunction **Set** $\leftrightarrows$ M**–Act** induced by the forgetful functor M**–Act** $\to$ **Set** (there are additional remarks on the forgetful functor and its adjoint(s) in Appendix A).

Naively adapting this result for partial actions, the same construction on the category **pSet** of sets and *partial* functions indeed still induces a monad, with the same operations as before; moreover, its Algebras[1] are partial functions of type $M \times X \rightharpoonup X$. Thus, it seems as if we found the desired characterisation:

[1]Note that this yields left actions; for right actions, we would have to use the symmteric variant of the multiplication μ.

Proposition 4.2
The mapping $X \mapsto M \times X$ extends to a monad on **pSet** whose Algebras are partial M-actions. $\square$

However, in Section 3.2.2, we argued that equivariant maps should be *total* functions and not introduce additional partiality. Yet, the morphisms of Algebras on **pSet** are necessarily *partial functions*: hence, $(M \times \_)$**–Alg** is *not* equivalent to the category M**–pAct**. Although one could argue that the definition of homomorphisms as total functions is simply wrong, since that is the only point which makes the characterisation as Algebras fail, there are good reasons why this characterisation *should* not be the correct one, and why our definition should stand as it is. For one, as already mentioned, homomorphisms should not introduce additional partiality, in accordance with common mathematical practise: also the papers [Exe98, KL02, MS02] use total functions as homomorphisms. Furthermore, the category M**–Act** of total M-actions would no longer be a full subcategory.

As a technical point, using Algebras on **pSet** requires that both syntax and semantics of timed processes 'live' in **pSet**, and that category does not have very 'nice' structure[2], when compared to the standard category **Set** of sets and total functions. Very importantly, limits are very different, e.g., the cartesian product of two sets X and Y in **Set** is simply $X \times Y$, while in **pSet**, it is $X + Y + X \times Y$. As relations, and in particular bisimulations, are regarded as subsets of the cartesian product in the currently known approaches, this would result in strange phenomena.

That very same point also applies to modelling syntax in the initial algebra style of [GTW78] where binary products model (the arguments of) binary function symbols; hence, either of the arguments could be potentially undefined, resulting in a curious mix of syntax and semantics: the *formal expression* (e.g., 0^{-1}) should always exist (assuming it is well-formed), while only its *interpretation* might very well be undefined (e.g., the multiplicative inverse of 0 in $\mathbb{R}$).

Finally, describing processes categorically is based on representing LTSs as *coalgebras*, rather than algebras/Algebras. Therefore, even if the above characterisation

[2]There is no lack of structure on **pSet**—in fact, it is very rich—but it has the 'wrong' concrete properties.

as Algebras were to work, the question would still remain how to combine it with the relatively well-established framework of modelling (the operational semantics of) processes and bisimulations as coalgebras. Consequently, we believe that it would be preferable to find a different, coalgebraic characterisation of partial monoid actions.

4.1.2 Total Monoid Actions as Coalgebras

Previously, we have seen that the algebras for the monad $T = M \times (\cdot)$ are precisely the total monoid actions; we shall now see how to obtain a coalgebraic characterisation of total actions. As is well known, the category **Set** is cartesian closed: exponentials are given by function spaces; concretely, any function $f : M \times X \to X$, via *currying* corresponds to a unique function $g : X \to X^M$: where f takes its two arguments at once, g takes them one after the other. Defining $DX \stackrel{\mathsf{df}}{=} X^M$ yields an endofunctor D on **Set** with $T \dashv D$ which, using the operations of T and the adjunction, is even a comonad—see [MM92, §V.8, Theorem 1].

Furthermore, by [MM92, §V.8, Theorem 2], D**–Coalg** $\cong$ T**–Alg**; hence, since we have T**–Alg** $\cong$ M**–Act**, we obtain D**–Coalg** $\cong$ M**–Act**, yielding a Coalgebraic characterisation of M-actions:

Proposition 4.3
For a monoid M, the endofunctor D on **Set** mapping X to X^M is actually a comonad $\langle D, \varepsilon, \delta \rangle$, when defining

$$\varepsilon_X : DX \to X,\ g \mapsto g(0)$$
$$\delta_X : DX \to D^2X,\ g \mapsto \lambda m.g(\_ + m)$$

Moreover, D**–Coalg** $\cong$ M**–Act**. □

Note a subtlety involved in the definition of δ: the action of m on g is given by adding to the argument of *g on the right*; the symmetric variant would not work in a non-commutative setting: the square in (2.3) would not commute. The important point about this coalgebraic characterisation of M**–Act** is that, by using the cartesian closed structure on **Set**, it provides the necessary freedom to also be able to deal with partial monoid actions.

4.2 Partial Monoid Actions, Categorically

In this section, building on the preceding Coalgebraic description of total monoid actions, we present our novel, also Coalgebraic, characterisation of partial actions of a time domain[3] $\mathcal{T} = \langle \mathcal{T}, +, 0 \rangle$, and hence, by Theorem 3.33, also of TTSs over $\mathcal{T}$. It is based on our notion of *evolution*: specific partial functions $\mathcal{T} \rightharpoonup X$ satisfying properties which mimic the axioms (3.12) and (3.13) of partial actions as far as possible, obtained by 'currying' partial actions $X \times \mathcal{T} \rightharpoonup X$, yielding maps of type $X \to EX$ where EX is a suitable set of partial functions of type $\mathcal{T} \rightharpoonup X$. The following subsections are, in that order, devoted to introducing the comonad of evolutions, establishing the correspondence between its Coalgebras and partial actions, and showing that Coalgebraic bisimulation (the adapted notion from [AM89], see Chapter 2) is equivalent to time bisimulation, providing a nice match between concrete and categorical formulations.

4.2.1 The Evolution Comonad

Definition 4.4

A *$\mathcal{T}$-evolution (on X)* is a partial function $e : \mathcal{T} \rightharpoonup X$ with the following two properties:

$$e(0)\downarrow \tag{4.1}$$

$$(\forall t, u \in \mathcal{T}).\ e(t+u)\downarrow \Rightarrow e(t)\downarrow \tag{4.2}$$

The *domain* of an evolution is defined as usual, viz.,

$$\mathrm{dom}(e) \stackrel{\mathsf{df}}{=} \{t \in \mathcal{T} \mid e(t)\downarrow\}$$

The set all $\mathcal{T}$-evolutions on X is denoted by $E_{\mathcal{T}}X$, or, if $\mathcal{T}$ is clear from the context, simply by EX. ■

Example 4.5

Consider $\mathcal{T}$-evolutions for the trivial time domain $\mathcal{T} = 1$. For any set X, because of (4.1), an evolution $e \in E_1X$ must be defined at the only element $0 \in 1$, so it is simply a *total* function $1 \to X$. Therefore $E_1X = X^1 \cong X$, i.e., E_1 is the identity functor. ◊

[3] Although the following construction also works for an arbitrary monoid M, since we aim at a coalgebraic model of TTSs, we restrict our attention to time domains.

We would like to draw the reader's attention to the (so far only syntactic) similarity between (4.2) and (3.14). Furthermore, recalling the definition of the induced order $\leq$ on $\mathcal{T}$ from Definition 3.5, $t+u$ simply denotes an arbitrary element $s \in \mathcal{T}$ such that $t \leq s$; hence, (4.2) is equivalent to

$$s \in \mathrm{dom}(e) \wedge t \leq s \Rightarrow t \in \mathrm{dom}(e)$$

making $\mathrm{dom}(e)$ a *downward-closed*[4] subset of $\mathcal{T}$.

An *(order) ideal* [Plo83] in $\mathcal{T}$ is a downward-closed subset of $\mathcal{T}$ which additionally contains the least element 0, so $\{0\}$ and $\mathcal{T}$ are always examples of ideals (cf. the definitions for rings [Lan93] and lattices [Joh82]). In analogy to the terminology for rings, we call the set of ideals of $\mathcal{T}$ the *spectrum*[5] of $\mathcal{T}$, written as $\mathrm{spec}(\mathcal{T})$; an ideal $I \in \mathrm{spec}(\mathcal{T})$ is *proper* if $I \neq \mathcal{T}$. By (4.1), $\mathrm{dom}(e)$ must be an ideal, therefore $\mathrm{spec}(\mathcal{T})$ contains *all possible domains* of $\mathcal{T}$-evolutions. The *principal* ideal on, or the *ideal generated by* $t \in \mathcal{T}$ is the set $\{u \in \mathcal{T} \mid u \leq t\}$ of all elements of $\mathcal{T}$ below t.

Dually, upward-closed (or upper closed) subsets of $\mathcal{T}$ are called *filters*, i.e., a filter $F \subseteq \mathcal{T}$ satisfies $t \in F \wedge\ t \leq u\ \Rightarrow\ u \in F$ for all $t, u \in \mathcal{T}$; the *principal filter* on $t \in \mathcal{T}$ is the set $\{u \in \mathcal{T} \mid t \leq u\}$. Complements of ideals are always filters; hence, given an evolution e in EX, the set $\mathcal{T} \setminus \mathrm{dom}(e)$ is a filter. This can also be obtained directly, via the contraposition of (4.2):

$$(\forall t, u \in \mathcal{T}).\ e(t)\uparrow \Rightarrow\ e(t+u)\uparrow \tag{4.3}$$

or, again observing that $t+u$ simply denotes an element of $s \in \mathcal{T}$ such that $t \leq s$, $e(t)\uparrow \wedge t \leq s\ \Rightarrow\ e(s)\uparrow$.

The concrete time domains $\mathbb{N}$ and $\mathbb{R}$ are both linear and *complete* in the sense that every non-empty, bounded subset T (i.e., $T \neq \emptyset$ and there exists $t \in \mathcal{T}$ such that $T \leq t$) has a *least* upper bound, $\sup T$. In such a situation, there is a more specific characterisation of ideals by *intervals* in the partial order $\leq$, which are defined as usual, i.e., $[0,t] \stackrel{\mathrm{df}}{=} \{u \in \mathcal{T} \mid 0 \leq u \leq t\}$, and $[0,t) \stackrel{\mathrm{df}}{=} \{u \in \mathcal{T} \mid 0 \leq u < t\}$. Note that $[0,t) \neq \emptyset$ if and only if $t > 0$.

[4] Also known as *lower closed* [Vic89] or *left closed* [Plo83].

[5] The spectrum of a ring only consists of *prime* ideals; since primeness does not make sense in the order-theoretic framework, we drop this condition.

Proposition 4.6
Let $\mathcal{T}$ be a linear and complete time domain with respect to the induced order $\leq$. Then $\mathrm{spec}(T)$ only contains $\mathcal{T}$, and intervals of the form $[0,t]$ or $[0,u)$, for $t \in \mathcal{T}$ and $u > 0$.

Proof: Since we already know that $\mathcal{T}$ and all intervals of the given form are always contained in $\mathrm{spec}(\mathcal{T})$, we merely have to prove the 'only' part, i.e., that, under the described assumptions on $\mathcal{T}$, there are no other ones. Let therefore $I \subset \mathcal{T}$ be a proper ideal, and choose $u \in \mathcal{T}$ such that $u \notin I$. By (4.2) and linearity, it must hold that $I \leq u$ (otherwise, since I is downward-closed, $u \in I$ would follow), and by (4.1), I is non-empty. Hence, by the assumption of completeness, we know that $t \overset{\mathsf{df}}{=} \sup I$ exists. We get two cases:

1. Assume $t \in I$. Then, since $t = \sup I$, we definitely know that $I \leq t$, and so $I \subseteq [0,t]$. Yet, since I is downward closed and $t \in I$, also $[0,t] \subseteq I$, hence $I = [0,t]$.

2. Assume $t \notin I$. Note that, since $0 \in I$ by (4.1), $t > 0$ holds. Then again, since $t = \sup I$, $I < t$ and consequently, $I \subseteq [0,t)$. Suppose that there exists $u \in [0,t)$, i.e., $0 \leq u < t$, such that $u \notin I$. As above, this would imply $I \leq u < t$, contradicting the fact that $t = \sup I$ is the least upper bound of I. Therefore, $[0,t) \subseteq I$, and so $I = [0,t)$.

Hence any proper I must be an interval, as claimed. □

For $\mathcal{T} = \mathbb{N}$, $u > 0$ is equivalent to $(\exists t \in \mathcal{T}).\ u = t+1$, and $[0,t+1) = [0,t]$. Therefore, writing $\mathbb{N}_\infty \overset{\mathsf{df}}{=} \mathbb{N} \cup \{\infty\}$, we obtain the following a consequence:

Corollary 4.7 $\mathrm{spec}(\mathbb{N}) \cong \mathbb{N}_\infty$

Proof: As remarked above, all non-empty right-open intervals in $\mathbb{N}$ can be expressed as closed intervals, i.e., the spectrum of $\mathbb{N}$ contains only the intervals $[0,t]$, for all $t \in \mathbb{N}$, and $\mathbb{N}$ itself. One obtains the desired isomorphism by identifying $[0,t]$ with t, and $\mathbb{N}$ with ∞, (their respective 'upper bounds,' ∞ denoting unboundedness). □

In particular, when considering $\mathcal{T}$-evolutions for a linear and complete time domain $\mathcal{T}$, the domains of such an evolution can only be $\mathcal{T}$, i.e., the evolution is total, or an interval, since its domain has to be an ideal. Note that in the commutative case, these complete, linear time domains are equivalent to the ones proposed in [Jef91], where it is postulated that 0 is the least element with respect to the linear precedence relation $\leq$, and that there exist greatest lower bounds $\inf T$ for non-empty subsets $T \subseteq \mathcal{T}$: under the additional conditions, sup and inf are interdefinable, viz., $\sup T$ is the inf of the set of all upper bounds of T, and $\inf T$ is the sup of all lower bounds of T.

For Proposition 4.6 to hold, the linear order on $\mathcal{T}$ is essential: if $\mathcal{T}$ is complete but not linear, e.g., for $\mathcal{T} = C^\star$, it holds, for $e \in EX$ and $t \in \mathcal{T}$ such that $e(t)\downarrow$, that $[0,t] \subseteq \mathrm{dom}(e)$ but in general, the inclusion is proper. For example, let $C = \{a,b\}$ and let e be any $C^\star$-evolution with domain $\{\varepsilon, a, b\}$; this is clearly well-defined. Then there is no $w \in C^\star$ such that $\mathrm{dom}(e) = [\varepsilon, w]$ (nor $[\varepsilon, w)$, for that matter), for any such w must satisfy $a \leq w$ and $b \leq w$ which, since $\leq$is the prefix-order, is not possible. In this particular example, $\mathrm{dom}(e)$ is the *union* of the intervals defined by its (finitely many) *maximal* elements a, b, $\mathrm{dom}(e) = [\varepsilon, a] \cup [\varepsilon, b]$, yet in general, maximal elements might not even exist: consider the domain $a^\star = \{a^n \mid n \in \mathbb{N}\} \subset C^\star$ which can only be described by the infinite union of intervals $\bigcup_{n=0}^{\infty} [\varepsilon, a^n]$.

Intuitively, an evolution $e \in EX$ represents (the time transitions) of some (anonymous) timed process with states in X: $e(t)$, provided that it is defined, denotes the state in X the process has *evolved to* after $t \in \mathcal{T}$ units of time, hence the name evolution; in other words, an evolution e contains a *complete* description of all time transitions of a process. Because of this, we simply regard the evolution *itself* as the process:

$$e \overset{t}{\rightsquigarrow} x \overset{\mathrm{df}}{\Leftrightarrow} e(t) \simeq x \tag{4.4}$$

Note that only *one* step is defined: we know the transitions of ‘the process’ e, yet we do not know anything about the transitions of the successor states $e(t)$ if $e(t)\downarrow$.

As usual, we abbreviate $(\exists x \in X).\ e \overset{t}{\rightsquigarrow} x$ by $e \overset{t}{\rightsquigarrow}$, but by definition, $e \overset{t}{\rightsquigarrow}$ implies $e \overset{t}{\rightsquigarrow} e(t)$: for each $t \in \mathcal{T}$, there is *at most one* state e can reach by a $\overset{t}{\rightsquigarrow}$-transition, the ‘transition relation’ induced by e satisfies (Determinacy). With these notations, (4.1) and (4.2) can be rewritten as follows:

$$e \overset{0}{\rightsquigarrow} \qquad\qquad (\forall t, u \in \mathcal{T}).\ e \overset{t+u}{\rightsquigarrow} \Rightarrow e \overset{t}{\rightsquigarrow}$$

Hence, (4.1) simply expresses e can at least idle for 0 units of time. Since 0 denotes the absence of duration, this means that the process e must be in some initial state: (4.1) describes a basic version of (ZeroDelay) but there are no constraints expressed as to the choice of this initial state. Analogously, (4.2) describes a simplified variant of (Continuity); the states $e(t)$ and $e(t+u)$ exist but, unlike to what is postulated by (Continuity), no relation between the two is imposed. However, it should be obvious that evolutions incorporate (rudimentary versions of) all the axioms of TTSs.

We intend to model timed processes as coalgebras $X \to EX$, associating an evolution to each state $x \in X$; for this, we first have to make E into a functor:

Lemma 4.8
Given a function $f : X \to Y$, defining $Ef : EX \to EY$ by $e \mapsto f \circ e$ makes E an endofunctor on **Set**. Moreover, $\mathrm{dom}(e) = \mathrm{dom}(Ef(e))$.

Proof: Since post-composition by a total function does not add any undefinedness, the claim about the domains follows trivially. Also, the property of e being an evolution immediately carries over to $Ef(e) = f \circ e$. □

Given an arbitrary E-coalgebra $k : X \to EX$, we would like to define

$$x \overset{t}{\rightsquigarrow} x' \overset{\text{df}}{\Leftrightarrow} k(x)(t) \simeq x'$$

but this does not always yield a TTS on X: e.g., (4.1) implies $x \overset{0}{\rightsquigarrow} x'$ but there is no guarantee that $x = x'$ holds. Consequently, we need to impose additional restrictions on how evolutions are assigned to states, allowing only special coalgebras; as already stated, E is not only a functor:

Proposition 4.9
E is a comonad, with counit ε and comultiplication δ given by

$$\varepsilon_X : EX \to X,\ e \mapsto e(0)$$

$$\delta_X : EX \to E^2X,\ e \mapsto \lambda t. \begin{cases} e+t \overset{\text{df}}{=} \left(\lambda u. \begin{cases} e(t+u) & \text{if } e(t+u)\downarrow \\ \text{undef} & \text{if } e(t+u)\uparrow \end{cases}\right) & \text{if } e(t)\downarrow \\ \text{undef} & \text{if } e(t)\uparrow \end{cases}$$

Proof: It is obvious that ε_X is a total map: evolutions are always defined at 0; checking naturality is routine. To show that δ is well-defined, we note firstly that $\delta_X(e)$ maps

$t \in \mathcal{T}$ to the function $e+t = \lambda u.e(t+u)$ if and only if $e(t)$ is defined, hence the partial map $t \mapsto e+t$ is an evolution. Secondly, we only obtain $e+t$ as a value in the case that $e(t)\downarrow$, therefore $e+t(0)\downarrow$; furthermore, as e is an evolution, the property (4.2) also holds for $e+t$ showing that $e+t$ is itself an evolution and thus we actually obtain a map of the desired type $EX \to E(EX)$. Naturality is again easy to prove.

To show that E is a comonad, we have to check that the diagrams in (2.3) commute. Let us first check the two triangles. For the left hand one, $e \in EX$ is sent to $\delta_X(e)$ and applying ε_{EX} to that we obtain, as $e(0)$ is defined, $\lambda u.e(0+u) = \lambda u.e(u)$ and so this triangle commutes. For the right hand one, $\delta_X(e)$ is mapped by applying $E\varepsilon_X$ (ie post-composition with ε_X where $\delta_X(e)$ is defined) to the following function

$$\lambda t.\begin{cases}\varepsilon_X(\lambda u.e(t+u)) & \text{if } e(t)\downarrow \\ \text{undef} & \text{otherwise}\end{cases} = \lambda t.\begin{cases} e(t) & \text{if } e(t)\downarrow \\ \text{undef} & \text{if } e(t)\uparrow\end{cases}$$

using the fact that 0 is the neutral element for $+$. The result is equal to e and hence also the second triangle commutes.

As for the square in (2.3), first applying to δ_X and then $E\delta_X$ results in the following calculations

$$\begin{aligned}
e \stackrel{\delta_X}{\longmapsto} \lambda t.&\begin{cases}\delta_X(e)(t) & \text{if } e(t)\downarrow \\ \text{undef} & \text{otherwise}\end{cases} \stackrel{E\delta_X}{\longmapsto} \lambda t.\begin{cases}\delta_X(e+t) & \text{if } e(t)\downarrow \\ \text{undef} & \text{otherwise}\end{cases} \\
= \lambda t.&\begin{cases}\lambda u.\begin{cases}\lambda s.e+t(u+s) & \text{if } e+t(u)\downarrow \\ \text{undef} & \text{otherwise}\end{cases} & \text{if } e(t)\downarrow \\ \text{undef} & \text{otherwise}\end{cases} \\
= \lambda t.&\begin{cases}\lambda u.\begin{cases}\lambda s.e(t+(u+s)) & \text{if } e(t+u)\downarrow \\ \text{undef} & \text{otherwise}\end{cases} & \text{if } e(t)\downarrow \\ \text{undef} & \text{otherwise}\end{cases}
\end{aligned}$$

The other path, first applying δ_X and then δ_{EX} yields

$$e \xmapsto{\delta_X} \overbrace{\lambda t.\begin{cases}\lambda u.e(t+u) & \text{if } e(t)\downarrow \\ \text{undef} & \text{otherwise}\end{cases}}^{h} \xmapsto{\delta_{EX}} \lambda t.\begin{cases}\lambda u.h(t+u) & \text{if } h(t)\downarrow \\ \text{undef} & \text{otherwise}\end{cases}$$

$$= \lambda t.\begin{cases}\lambda u.\begin{cases}\lambda s.e((t+u)+s) & \text{if } e(t+u)\downarrow \\ \text{undef} & \text{otherwise}\end{cases} & \text{if } e(t)\downarrow \\ \text{undef} & \text{otherwise}\end{cases}$$

and because of the associativity of $+$ the two results are equal and the square commutes, concluding the proof that (E,ε,δ) is a comonad on **Set**. □

Note that, since our partial actions are *right* actions, the definition of δ is symmetric to the one used in the Comonadic description of total action in Proposition 4.3.

Definition 4.10
We call $\langle E,\varepsilon,\delta\rangle$ the *evolution comonad* on **Set**. ■

Considering the intuitive meanings of the comonad operations, ε could be called the *naming function*, based on the following considerations. Given an evolution $e \in EX$, $\varepsilon_X(e) = e(0)$ is the state that the 'process' e has reached after *no* time has passed, since precisely that was the intuition of 0. In other words, the process e after no units of time is in state $e(0)$, hence we call $e(0)$ the *name* of e. Furthermore, note that δ transforms an evolution e on X into an evolution $\delta(e)$ on EX by acting as a *parameterised shift* or *lookahead*, as is illustrated in Figure 4.1 for an evolution $e \in E\mathcal{T}$: $\delta(e)(t_0)$ is equal

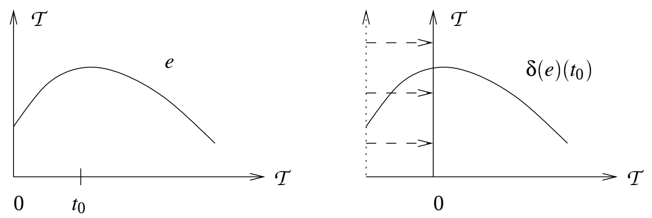


Figure 4.1: The shift-intuition of the comultiplication δ.

to $e+t_0$, i.e., the evolution e after $t_0 \in \mathcal{T}$ units of time have passed; the (rather complicated) case distinction only takes care of undefinedness that might arise. With this interpretation, the comonad law $\varepsilon_E \circ \delta = \mathrm{id}_E$ states $(\delta(e))(0) = e+0 = e$, i.e., shifting by 0 is the same as not shifting at all, or the name of the (un-shifted) shifted evolution $e+0$ is the evolution e itself.

Remark 4.11

Consider the two natural transformations of type $\mathrm{Id} \Rightarrow E$ defined as follows:

$$\gamma_X^{(1)} : X \to EX,\; x \mapsto \lambda t.\begin{cases} x & \text{if } t = 0 \\ \text{undef} & \text{if } t > 0 \end{cases}$$
$$\gamma_X^{(2)} : X \to EX,\; x \mapsto \lambda t.\, x$$

Both ‘transform’ elements of X, i.e., states, into evolutions on X, i.e., timed processes with states in X. The first, $\gamma_X^{(1)}$, takes a state and produces an evolution which is undefined except at 0, where the value is x. This corresponds to interpreting a state as a ‘dead’ process which only has a $\overset{0}{\leadsto}$-transition, to itself; cf. (ZeroDelay) and (4.4); this is the ‘minimal’ (with respect to the size of the domain) evolution a state can be associated to. The second transformation, $\gamma_X^{(2)}$, corresponds to the ‘maximal’ way of including states among evolutions: a single state x with a $\overset{t}{\leadsto}$-loop for all $t \in \mathcal{T}$.

Both $\gamma^{(i)}$ for $i \in \{1,2\}$ satisfy the following two equations

$$\varepsilon \circ \gamma^{(i)} = \mathrm{id} \qquad \delta \circ \gamma^{(i)} = \gamma_E^{(i)} \circ \gamma^{(i)} \tag{4.5}$$

yielding two distinct ways of making E a *computational comonad* [BG92b]. $\Diamond$

We will now prove several technical results about E which will be needed later.

Proposition 4.12

The final E-Coalgebra exists.

Proof: By dualising [Mac97, §VI.2, Theorem 1], it follows that the forgetful functor $U : E$–**Coalg** $\to$ **Set** has a right adjoint $R :$ **Set** $\to E$–**Coalg** mapping a set X to the *cofree* Coalgebra on X, $\delta_X : EX \to E(EX)$. Since R preserves all limits, its image of the final object 1 in **Set**, viz., $\delta_1 : E1 \to E(E1)$, is the final E-Coalgebra. $\square$

Since 1 is a singleton set, an evolution $e \in E1$ is completely determined by its domain $\mathrm{dom}(e) \in \mathrm{spec}(\mathcal{T})$, i.e., $E1 \cong \mathrm{spec}(\mathcal{T})$. Later, we explicitly describe the structure map δ_1; for now, we just give an example for a particular case.

Example 4.13
Consider $\mathcal{T} = \mathbb{N}$; we have already seen that $\mathrm{spec}(\mathbb{N}) = \mathbb{N}_\infty$, i.e., the carrier of the final $E_{\mathbb{N}}$-Coalgebra is $\mathbb{N}_\infty$. Moreover, the structure map δ_1 is nothing but the partial subtraction (3.4) extended with the clause $\infty - n = \infty$. $\Diamond$

Proposition 4.14
The evolution comonad E preserves pullbacks, i.e., if

$$\begin{tikzcd} P \arrow[r, "q"] \arrow[d, "p"'] & Y \arrow[d, "g"] \\ X \arrow[r, "f"] & Z \end{tikzcd} \tag{4.6}$$

is a pullback square in **Set**, then

$$\begin{tikzcd} EP \arrow[r, "Eq"] \arrow[d, "Ep"'] & EY \arrow[d, "Eg"] \\ EX \arrow[r, "Ef"] & EZ \end{tikzcd} \tag{4.7}$$

is also a pullback square in **Set**.

Proof: Assume that we have a pullback diagram as given in (4.6). It is well-known, see, e.g., [Mac97], that the pullback P can concretely be described as

$$P = \{\langle x, y\rangle \in X \times Y \mid fx = gy\} \subseteq X \times Y \tag{4.8}$$

while p and q are just the restrictions of the projections π_1 and π_2 from the cartesian product, i.e., they make the following diagram commute

$$\begin{tikzcd} & P \arrow[dl, "p"'] \arrow[dr, "q"] \arrow[d, phantom, "\cap|"] & \\ X & X \times Y \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] & Y \end{tikzcd} \tag{4.9}$$

Applying these concrete data to the 'lifted' pullback square (4.7), we obtain

$$\begin{aligned} EP &= \{e : \mathcal{T} \rightharpoonup P \mid e(0)\downarrow \wedge e(t+u)\downarrow \Rightarrow e(t)\downarrow\} \\ &= \{e : \mathcal{T} \rightharpoonup X \times Y \mid e(0)\downarrow \wedge e(t+u)\downarrow \Rightarrow e(t)\downarrow \wedge f(\pi_1(e(t))) = g(\pi_2(e(t)))\} \end{aligned}$$

The maps Ep and Eq are, by the definition of E on maps, simply post-composition with the restricted projections, and since E is a functor, the square (4.7) commutes.

In contrast to that, using (4.8) and the definition of E on maps,, the actual pullback Q of Ef and Eg is given as

$$\begin{aligned} Q &= \{\langle e, e'\rangle \in EX \times EY \mid Ef(e) = Eg(e')\} \\ &= \{\langle e, e'\rangle \in EX \times EY \mid f \circ e = g \circ e'\} \end{aligned}$$

We show $EP \cong Q$, defining mutually inverse maps in both directions.

To define a map from Q to EP, note that, if $\langle e, e'\rangle \in Q$ we must have $f \circ e = g \circ e'$, and since both f and g are total functions, this implies $\mathrm{dom}(e) = \mathrm{dom}(e')$. Hence, the following map is well-defined:

$$e \times e' \;:\; \mathcal{T} \rightharpoonup X \times Y,\ t \;\mapsto\; \begin{cases} \langle e(t), e'(t)\rangle & \text{if } e(t)\downarrow \wedge e'(t)\downarrow \\ \text{undef} & \text{if } e(t)\uparrow \vee e'(t)\uparrow \end{cases}$$

By definition of Q, we have $f(e(t)) = g(e'(t))$ and for $t \in \mathrm{dom}(e) = \mathrm{dom}(e')$ and so, by definition of P, we obtain that $(e \times e')(t) = \langle e(t), e'(t)\rangle \in P$. Since (4.1) and (4.2) carry over from e and e', we obtain $(e \times e') \in EP$.

In the other direction, let $e \in EP$, i.e., for each $t \in \mathrm{dom}(e)$, $e(t) \in P$, and so there exist $x \in X$ and $y \in Y$ such that $fx = gy$ and $e(t) = \langle x, y\rangle$; hence, we get two maps

$$e_i \stackrel{\mathsf{df}}{=} (\pi_i \circ e) = E\pi_i(e) : \mathcal{T} \rightharpoonup X,\ t \;\mapsto\; \begin{cases} \pi_i(e(t)) & \text{if } e(t)\downarrow \\ \text{undef} & \text{if } e(t)\uparrow \end{cases}$$

Clearly, $e_1 \in EX$ and $e_2 \in EY$ and, moreover, we can calculate

$$\begin{aligned} &(f \circ e_1)(t) = f(e_1(t)) = f(\pi_1(e(t))) \stackrel{(4.9)}{=} f(Ep(e)(t)) = ((Ef \circ Ep)(e))(t) \\ &\stackrel{(4.7)}{=} ((Eg \circ Eq)(e))(t) = g(Ep(e)(t)) \stackrel{(4.9)}{=} g(\pi_2(e(t))) = g(e_2(t)) = (g \circ e_2)(t) \end{aligned}$$

and so $\langle e, e'\rangle \in Q$, i.e., we obtain the function of type $EP \to Q$.

The remaining proof obligation is to show that the two maps are inverse to each other, i.e., the two possible composite maps are the respective identities. For one of the composites, $EP \to Q \to EP$, let $e \in EP$. This is first mapped to

$$\langle E\pi_1(e), E\pi_2(e)\rangle = \langle Ep(e), Eq(e)\rangle$$

which then gets mapped to $Ep(e) \times Eq(e)$ whose values are, for $t \in \mathcal{T}$:

$$\langle Ep(e)(t), Eq(e)(t)\rangle \simeq \langle \pi_1(e(t)), \pi_2(e(t))\rangle \simeq e(t)$$

and so this composite is the identity on EP. The other composite is equally easily seen to be the identity on Q, so we have established $EP \cong Q$, concluding the proof. $\square$

By [JPT$^+$01, Corollary 3.2], it then follows that E–**Coalg** is a topos [MM92].

Weak pullbacks (like all weak limits, cf. [Mac97]) are obtained by dropping the uniqueness requirement of pullbacks. If a functor F transforms a weak pullback diagram into a weak pullback diagram, analogously to Propositon 4.14, one says that *F preserves weak pullbacks*[6]. Preservation of pullbacks implies preservation of weak pullbacks [Rut00], hence:

Corollary 4.15 E preserves weak pullbacks.

Recall that, by Proposition 3.17, if $\mathcal{T}$ is non-trivial, it must be infinite; moreover, if $\mathcal{T}$ is trivial, i.e., $\mathcal{T} = 1$, the first regular cardinal greater than $|\mathcal{T}|$ is $\aleph_0$.

Proposition 4.16
Let $\mathcal{T}$ be a time domain. Then $E = E_{\mathcal{T}}$ has a rank, in particular E has rank κ where κ is an infinite regular cardinal with $\kappa > |\mathcal{T}|$.

Proof: If $\mathcal{T}$ is trivial, we have already seen in Example 4.5 that $E = E_1$ is the identity functor, which trivially preserves *all* colimits, hence E is in particular finitary. Let now $\mathcal{T}$ be a non-trivial time domain, and let κ be a regular cardinal such that $\kappa > |\mathcal{T}|$. We have to show that E preserves κ-filtered colimits. Let therefore I be a κ-filtered

[6] Note that there is the slightly different notion of a functor F *weakly preserving* pullbacks [JPT$^+$01, Gum01]: this expresses that F transforms pullback squares into weak pullback squares; however, in categories where all pullbacks exist (e.g., in **Set**), the two properties are equivalent—see [Gum01].

(index) category, i.e., every diagram in I of size less than κ has a cocone over it in I; we will write the objects of I as $\{I_\lambda \mid \lambda \leq |I|\}$. Let now $F : I \to \mathbf{Set}$ be a functor, i.e., a diagram F of 'shape' I in **Set**. Since **Set** is cocomplete [Mac97], we know that its colimit $\mathrm{Colim}\, F$, together with its colimiting cocone $\varphi : F \Rightarrow \Delta \mathrm{Colim}\, F$, exists; we will write the object FI_λ simply as F_λ, and the component $F_\lambda \to \mathrm{Colim}\, F$ of φ at the object F_λ as φ_λ. Our concrete task is then to show that $E\,\mathrm{Colim}\, F \cong \mathrm{Colim}\, EF$.

As E is a functor, it transforms commuting diagrams into commuting diagrams; therefore, applying it to the cocone $\varphi : F \Rightarrow \Delta \mathrm{Colim}\, F$ yields another cocone, viz., $E\varphi : EF \Rightarrow \Delta E\,\mathrm{Colim}\, F$, for the functor $EF : I \to \mathbf{Set}$. Again, this time for the functor EF, we get the colimiting cocone $\tau : EF \Rightarrow \Delta \mathrm{Colim}\, EF$, i.e., the universal cocone over EF. Thus, there exists a unique mediating map $i : \mathrm{Colim}\, EF \to E\,\mathrm{Colim}\, F$ such that

$$(E\varphi)_\lambda = i \circ \tau_\lambda \tag{4.10}$$

for each λ such that I_λ is an object in I. To obtain the desired isomorphisms, we have to define a map $j : E\,\mathrm{Colim}\, F \to \mathrm{Colim}\, EF$ such that both $i \circ j$ and $j \circ i$ are the identities on $E\,\mathrm{Colim}\, F$ and $\mathrm{Colim}\, EF$, respectively.

To define j, let $e \in E\,\mathrm{Colim}\, F$, i.e., $e : \mathcal{T} \rightharpoonup \mathrm{Colim}\, F$ such that $e(0)\downarrow$ and $e(t+u)\downarrow$ implies $e(t)\downarrow$. For each $t \in \mathrm{dom}(e)$, we get a value $v_t \stackrel{\mathrm{df}}{=} e(t) \in \mathrm{Colim}\, F$, and by the universal property of the colimit, for each such v_t, there exists some k_t and $f_t \in F_{k_t}$ in the diagram F, for some object I_{k_t} in I, such that $\varphi_{k_t}(f_t) = v_t$. Now $|\mathrm{dom}(e)| \leq |\mathcal{T}| < \kappa$, so we get less than κ many objects I_λ in I in this way and by κ-filteredness, there exists a cocone over them in I, i.e., a cocone which has all these I_{k_t} as its base and some object I_G as its vertex. Therefore, applying F to this cocone, we obtain a set $G = FI_G$, which is therefore contained among the F_λ, and a cocone σ such that the base of σ is given by the F_{k_t}, i.e., we have maps, for all $t \in \mathrm{dom}(e)$

$$\sigma_t : F_{k_t} \to G$$

Now, since $\varphi : F \Rightarrow \mathrm{Colim}\, F$ is in particular a (super-)cocone of σ, there must be a component $\varphi_G : G \to \mathrm{Colim}\, F$, and it must holds, for all $t \in \mathrm{dom}(e)$, that

$$\varphi_{k_t} = \varphi_G \circ \sigma_t \tag{4.11}$$

hence, $v_t = \varphi_{k_t}(f_t) = \varphi_G(\sigma_t(f_t))$, and, in particular, $\sigma_t(f_t) \in G$. Therefore, we obtain a $\mathcal{T}$-evolution over G by defining

$$e' : \mathcal{T} \rightharpoonup G,\ t \ \mapsto \begin{cases} \sigma_t(f_t) & \text{if } t \in \mathrm{dom}(e) \\ \text{undef} & \text{if } t \notin \mathrm{dom}(e) \end{cases}$$

which has the same domain as e. Moreover, by (4.11), we obtain for all $t \in \mathrm{dom}(e)$,

$$\varphi_G(e'(t)) = \varphi_G(\sigma_t(f_t)) = \varphi_{k_t}(f_t) = v_t = e(t)$$

In other words, $E\varphi_G(e') = \varphi_G \circ e' = e$, and $E\varphi_G : EG \to E\,\mathrm{Colim}\,F$ is part of the cocone $E\varphi : EF \to E\,\mathrm{Colim}\,F$, which therefore uniquely factors through the colimiting cocone $\tau : EF \Rightarrow \Delta\mathrm{Colim}\,EF$, i.e., we have $E\varphi_G = i \circ \tau_G$ for the mediating map $i : \mathrm{Colim}\,EF \to E\,\mathrm{Colim}\,F$.

We can now finally define the map $j : E\,\mathrm{Colim}\,F \to \mathrm{Colim}\,EF$ as follows, bearing in mind that $e \in E\,\mathrm{Colim}\,F$, and $e' \in EG$:

$$j(e) = \tau_G(e') \in \mathrm{Colim}\,EF \tag{4.12}$$

Note that this is well-defined: the value of j does not depend on the choice of G since τ is a cocone.

By (4.12), we know that $j(e) = \tau_G(e')$, and so, applying i to both sides, we obtain

$$i(j(e)) = i(\tau_G(e')) = E\varphi_G(e') = e$$

in other words, $i \circ j = \mathrm{id}_{E\,\mathrm{Colim}\,F}$.

For the other composite, $j \circ i$, let $x \in \mathrm{Colim}\,EF$. By a similar argument as above, there must exists a set G and an $e \in EG$ such that $x = \tau_G(e)$, otherwise the uniqueness of the mediating map i could not hold. We then calculate:

$$j(i(x)) = j(i(\tau_G(e))) \overset{(4.10)}{=} j(E\varphi_G(e)) \overset{(4.12)}{=} \tau_G(e) = x$$

Hence, also $j \circ i = \mathrm{id}_{\mathrm{Colim}\,EF}$ holds, and we have shown $E\,\mathrm{Colim}\,F \cong \mathrm{Colim}\,EF$. $\square$

4.2.2 Partial Actions as Coalgebras

Standard (image-finite) LTSs are coalgebras for B_A; in contrast, TTSs need the more powerful Coalgebras, as already hinted at: the Coalgebra laws constrain 'mere' E-coalgebras such that they become equivalent to TTSs. The formulation of the next proposition is a bit clumsy: we have not defined morphisms of TTSs explicitly, and we have not introduced a corresponding category, only the objects, so to speak—see Remark 3.36. Therefore, we can only compare the categories of partial $\mathcal{T}$-actions and the E-Coalgebras:

Theorem 4.17
The $E_{\mathcal{T}}$-Coalgebras are the same as the partial $\mathcal{T}$-actions, i.e., the TTSs over $\mathcal{T}$, and the passage from an E-Coalgebra $k : P \to EP$ to a partial action $P \times \mathcal{T} \rightharpoonup P$, to a TTS $\langle P, \mathcal{T}, \rightsquigarrow \rangle$, and vice versa, is given by

$$p \overset{t}{\rightsquigarrow} p' \iff p * t \simeq p' \iff k(p)(t) \simeq p' \tag{4.13}$$

Moreover, there is an isomorphism of categories $\mathcal{T}$**-pAct** $\cong E$**–Coalg**.

Proof: Since one half of the correspondence, viz., between TTSs and partial actions, was already shown in Theorem 3.33, we are here only going to establish the equivalence between E-Coalgebras and partial monoid actions. Assume therefore, that $k : X \to EX$ is an E-Coalgebra, i.e., it verifies the diagrams in (2.5). We have to show that k defines a partial $\mathcal{T}$-action on X, and vice versa, as given in (4.13). Let us now show (3.12) for the 'action' induced by k, i.e., we have to show that

$$x * 0 \simeq k(x)(0) \simeq x$$

Since $\varepsilon_X(e) = e(0)$ and $k(x) \in EX$, this is equivalent to

$$x \simeq \varepsilon_X(k(x)) \simeq (\varepsilon_X \circ k)(x)$$

Now notice that the triangle in (2.5) precisely states $\varepsilon_X \circ k = \mathrm{id}_X$, and so (3.12) holds.

We now have to show that also (3.13) holds for the defined notion, i.e., we have to show that

$$x * (t + u) \simeq k(x)(t + u) \simeq k(k(x)(t))(u) \simeq (x * t) * u$$

It is obvious that we must use the square in (2.5) to prove this property. The square states that $\delta_X \circ k = Ek \circ k$. Let us calculate the concrete descriptions of the two functions as follows:

$$\begin{aligned}
\delta_X \circ k : X \to E^2X,\ x \ \mapsto t.&\begin{cases} \lambda u.k(x)(t+u) & \text{if } k(x)(t)\downarrow \\ \text{undef} & \text{if } k(x)(t)\uparrow \end{cases} \\
= \lambda t.&\begin{cases} \lambda u.\begin{cases} k(x)(t+u) & \text{if } k(x)(t+u)\downarrow \\ \text{undef} & \text{if } k(x)(t+u)\uparrow \end{cases} & \text{if } k(x)(t)\downarrow \\ \text{undef} & \text{if } k(x)(t)\uparrow \end{cases}
\end{aligned}$$

and

$$\begin{aligned}
Ek \circ k : X \to E^2X,\ x \ \mapsto \ \lambda t.&\begin{cases} \lambda u.k(k(x)(t))(u) & \text{if } k(x)(t)\downarrow \\ \text{undef} & \text{if } k(x)(t)\uparrow \end{cases} \\
= \lambda t.&\begin{cases} \lambda u.\begin{cases} k(k(x)(t))(u) & \text{if } k(k(x)(t))(u)\downarrow \\ \text{undef} & \text{if } k(k(x)(t))(u)\uparrow \end{cases} & \text{if } k(x)(t)\downarrow \\ \text{undef} & \text{if } k(x)(t)\uparrow \end{cases}
\end{aligned}$$

Let us break down the desired Kleene equality into two Kleene implications. For $\precsim$, assume $(x*(t+u))\downarrow$. Since $x*(t+u) \simeq k(x)(t+u)$, and $k(x) \in EX$, this implies, by (4.2), that $k(x)(t)\downarrow$. Moreover, if $k(x)(t)\downarrow$, we have $(\delta_X \circ k)(t)(u) \simeq k(x)(t+u)$. Using the above equality of the functions $\delta_X \circ k = Ek \circ k$, we obtain that $k(k(x)(t))(u)\downarrow$ and moreover, is equal to $k(x)(t+u)$.

For the converse Kleene implication, assume $((x*t)*u)\downarrow$, i.e., $k(k(x)(t))(u)\downarrow$, and in particular, $k(x)(t)\downarrow$. Since $((Ek \circ k)(t))(u) \simeq k(k(x)(t))(u)$, $k(x)(t)\downarrow$ and, by assumption, $Ek \circ k$ being equal to $\delta_X \circ k$, we know that

$$k(k(x)(t))(u) \simeq ((Ek \circ k)(t))(u) \simeq ((\delta_X \circ k)(t))(u) \simeq k(x)(t+u)$$

Converting this into the action notation, as described in (4.13), we precisely obtain axiom (3.13), concluding the proof that k induces a partial $\mathcal{T}$-action.

As for the other direction, define the map

$$x \;\mapsto\; e_x \stackrel{\mathsf{df}}{=} \lambda t.\begin{cases} x*t & \text{if } (x*t)\downarrow \\ \text{undef} & \text{if } (x*t)\uparrow \end{cases}$$

We have to show that e_x is an evolution in EX. Since $x*0 \simeq x$, we have that $e_x(0) = x$, in particular $e_x(0)\downarrow$, establishing (4.1). Assume now that $e_x(t+u)\downarrow$. By its definition, $e_x(t+u) \simeq x*(t+u)$. Since this last expression, by (3.13), is equal to $(x*t)*u$, and so in particular, by (3.14), $(x*t)\downarrow$, but that exactly means $e_x(t)\downarrow$, i.e., we have shown (4.2).

Next, we have to show that the map $k : x \mapsto e_x$ is not only an E-coalgebra, but an E-Coalgebra, i.e., that it satisfies the diagrams in (2.5). For the triangle, we have to show $e_x(0) = x$ but that is an immediate consequence of (3.12). For the square, we have to show that $e_x(t+u) \simeq e_{e_x(t)}(u)$, but this means nothing but $x*(t+u) \simeq (x*t)*u$, which is exactly (3.13). Finally, it is easy to see that the two possible compositions of the two constructions, taking a partial $\mathcal{T}$-action to another partial $\mathcal{T}$-action, and an E-Coalgebra to another E-Coalgebra, are mutually inverse to each other and therefore, we have established the desired correspondence.

In order to show the categories isomorphic we also need to consider the respective morphisms of $\mathcal{T}$**-pAct** and E**–Coalg**. In $\mathcal{T}$**-pAct**, the morphisms are homomorphisms of partial $\mathcal{T}$-actions, i.e., maps $f : X \to Y$ for two partial $\mathcal{T}$-actions with respective carriers X and Y, such that equation (3.20) holds, the morphisms in E**–Coalg** are homomorphisms of E-Coalgebras as defined in (2.6). The square in (2.6) commuting states that two functions $X \to EY$, viz $k_2 \circ f$ and $Ef \circ k_1$, have to be equal whose values we compute as follows.

$$Ef \circ k_1 : X \to EY,\; x \mapsto \lambda t.\begin{cases} f(k_1(x)(t)) & \text{if } k_1(x)(t)\downarrow \\ \text{undef} & \text{if } k_1(x)(t)\uparrow \end{cases}$$

$$k_2 \circ f : X \to EY,\; x \mapsto \lambda t.\begin{cases} k_2(fx)(t) & \text{if } k_2(fx)(t)\downarrow \\ \text{undef} & \text{if } k_2(fx)(t)\uparrow \end{cases}$$

These two functions being equal means that

$$f(k_1(x)(t)) \simeq k_2(fx)(t)$$

or in the action notation, writing $x *_i t$ for $k_i(x)(t)$,

$$f(x *_1 t) \simeq (fx) *_2 t,$$

i.e., precisely the property of a homomorphism of partial $\mathcal{T}$-actions as stated in (3.20) and hence, the categories $\mathcal{T}$**-pAct** and E**–Coalg** also have the same morphisms, concluding the proof. □

Note that, when proving the $\precsim$-direction of (3.13) for an E-Coalgebra k, it is absolutely vital that evolutions satisfy (4.2): one has to be able to deduce $(x * t)\downarrow$ when knowing that $(x * (t+u))\downarrow$. The reason for this is that the property of E-Coalgebras expressed in the square in (2.5), is almost, but *not quite*, equivalent to (3.13), as seen from the concrete descriptions of the two functions $\delta_X \circ k$ and $Ek \circ k$ given in the proof: their being equal can be translated into

$$(\forall x \in X).\ (\forall t \in \mathcal{T}).\ k(x)(t)\downarrow \Rightarrow (\forall u \in \mathcal{T}).\ k(x)(t+u) \simeq k(k(x)(t))(u) \tag{4.14}$$

Hence, only under the assumption $k(x)(t)\downarrow$ we get the Kleene equality from (3.13), and (4.2) allows us to drop even that. Therefore, removing (4.2) from the definition of evolutions will result in destroying the correspondence between E-Coalgebras and partial monoid actions.

This prompts the question what kinds of structures we get when we do weaken the definition of evolution. There are several ways to do so. Dropping (4.2) from the definition of an evolution, one still obtains a comonad E' yet E'-Coalgebras are not the same as partial $\mathcal{T}$-actions, as already sketched above: in an E'-Coalgebra, it may well be the case that $k(x)(t+u)\downarrow$ despite $k(x)(t)\uparrow$, which is in conflict with the consequence (3.14) of axiom (3.13) for partial actions. Interestingly, we precisely obtain the right-equivalent of the weaker notion of partial monoid actions from [MS02], satisfying (3.15), which is the outcome of translating (4.14) from the coalgebraic formulation to the partial action notation.

It is not clear to us how to obtain a categorical description of partial group actions satisfying the axiom (3.16) from [KL02] since only a Kleene implication is used: our approach invariably results in Kleene equalities (= equality of partial functions). The use of implications, however, seems to point to an *(order-)enriched framework* with *lax*

Coalgebra homomorphisms and (order-)enriched (categorical) bisimulations [Fio96], dually to the use of lax algebra morphisms as, e.g., in [KL97].

Alternatively, one could also regard E' merely as a *co-pointed* functor [LPW00], i.e., forget about the comultiplication δ, and consider its corresponding coalgebras which only have to satisfy the triangle in (2.5). The resulting partial 'actions' then have to satisfy (3.12), yet no connection between the monoid addition and the action is imposed.

Dropping (4.1), ε cannot be defined, i.e., one no longer has a comonad, only a functor E'' with an 'associative' comultiplication, as expressed by the square of (2.3); one can define Coalgebras for such a 'semi-comonad' which only have to satisfy the square in (2.5). Concretely, E''-Coalgebras are partial semigroup actions: the correspondence from Theorem 4.17 works after leaving out axiom (3.12); from the point of view of TTSs, this means dropping axiom (ZeroDelay) while keeping the other two. Finally, removing both axioms for evolutions results simply in a functor, and its coalgebras correspond to *deterministic* transition systems; this is equivalent to dropping both (ZeroDelay) and (Continuity) from the definition of TTSs.

The final E-Coalgebra, i.e., the final partial $\mathcal{T}$-action, plays an important rôle in the general semantic framework as the collection of abstract behaviours, the 'archetypical' partial action: for any other action, there is a unique equivariant map into the final action. We already know that its carrier is $\mathrm{spec}(\mathcal{T})$ but we still need to describe its Coalgebra map δ_1 which, by Theorem 4.17, is a partial $\mathcal{T}$-action on $\mathrm{spec}(\mathcal{T})$.

Definition 4.18

For a time domain $\mathcal{T}$, define

$$I - t \stackrel{\mathsf{df}}{=} \{s \in \mathcal{T} \mid t + s \in I\}$$

for $I \in \mathrm{spec}(\mathcal{T})$ and $t \in T$. ■

Lemma 4.19

Let $\mathcal{T}$ be a time domain, $I \in \mathrm{spec}(\mathcal{T})$. Then the partial function defined as

$$\langle I, t\rangle \mapsto \begin{cases} I - t & \text{if } t \in I \\ \text{undef} & \text{if } t \notin I \end{cases} \tag{4.15}$$

is a partial $\mathcal{T}$-action on $\mathrm{spec}(\mathcal{T})$.

Proof: The map clearly is well-defined: $t \in I$ implies $0 \in I - t$, so it suffices to prove that $I - t$ is downward closed, which follows since $(\cdot) - t$, by Proposition 3.17, is monotone. Note that $s \in I - t$ if and only if there exists $u \in I$ such that $s = u - t$. Therefore, we can rewrite $I - t$ as follows:

$$I - t = \{s \in \mathcal{T} \mid (\exists u \in I).\ s = u - t\} = \{s - t \mid s \in I\}$$

The claim then follows from the uniqueness of the relative inverses, i.e., injectivity of $(\cdot) - t$, and since $-$ is a partial $\mathcal{T}$-action on $\mathcal{T}$ itself. □

Proposition 4.20

The partial $\mathcal{T}$-action δ_1 on the final E-Coalgebra $\mathrm{spec}(\mathcal{T})$ is given by (4.15).

Proof: An evolution $e \in E1$ is a partial function $\mathcal{T} \rightharpoonup 1$

$$e = \lambda t.\ \begin{cases} \star & \text{if } e(t)\downarrow \\ \text{undef} & \text{if } e(t)\uparrow \end{cases}$$

By the definition of δ, and writing I_e for $\mathrm{dom}(e)$, we obtain

$$\begin{aligned} \delta_1(e) &= \lambda t.\ \begin{cases} \left(\lambda u.\ \begin{cases} \star & \text{if } e(t+u)\downarrow \\ \text{undef} & \text{if } e(t+u)\uparrow \end{cases}\right) & \text{if } e(t)\downarrow \\ \text{undef} & \text{if } e(t)\uparrow \end{cases} \\ &= \lambda t.\ \begin{cases} \left(\lambda u.\ \begin{cases} \star & \text{if } t+u \in I_e \\ \text{undef} & \text{if } t+u \notin I_i \end{cases}\right) & \text{if } t \in I_e \\ \text{undef} & \text{if } t \in I_e \end{cases} \end{aligned}$$

Thus, one has $\mathrm{dom}(\delta_1(e)) = \mathrm{dom}(e) = I_e$, and for $t \in I_e$, we get

$$\mathrm{dom}(\delta_1(e)(t)) = \{u \in \mathcal{T} \mid t + u \in I_e\} = \{u \in \mathcal{T} \mid (\exists s \in I_e).\ t + u = s\}$$

Using the action notation for δ_1, and identifying e with I_e, this becomes

$$I_e * t = \{u \in \mathcal{T} \mid (\exists s \in I_e).\ s = t + u\} = I_e - t$$

and so we are done. □

For $\mathcal{T} = \mathbb{N}$, we have $\mathrm{spec}(\mathbb{N}) \cong \mathbb{N}_\infty$. The structure map δ_1 then works as follows: for $I = \mathbb{N}$, $\mathbb{N} - u = \mathbb{N}$, and so $\infty - t = \infty$; for $I = [0,t]$, $[0,t] - u = [0, t-u]$, corresponding to $t - u$. All in all, the partial $\mathbb{N}$-action on the final $E_{\mathbb{N}}$-Coalgebra is simply given by the (extended) partial subtraction on $\mathbb{N}_\infty$, as already claimed in Example 4.13. In this way, the concrete description of the final E-Coalgebra once again highlights the importance of ideals and the partial subtraction.

4.2.3 Coalgebraic E-Bisimulation

We will now show that Coalgebraic bisimulation for the evolution comonad (see Chapter 2) is (in a restricted sense) equivalent to the previously introduced notion of time bisimulation (see Definition 3.25). For this, recall from [Tur96] that a span

$$X \xleftarrow{r_1} R \xrightarrow{r_2} Y$$

is *jointly monic* if, given two arrows $g, h : S \to R$, $r_i \circ g = r_i \circ h$ implies $g = h$. Note that in **Set**, this is implies that $R \subseteq X \times Y$.

Proposition 4.21
Let $\mathcal{T}$ be a time domain, and write $E = E_{\mathcal{T}}$. Then:

(i) Any time bisimulation for TTSs over $\mathcal{T}$ is a Coalgebraic E-bisimulation.

(ii) Any jointly monic Coalgebraic E-bisimulation is a time bisimulation for TTSs over $\mathcal{T}$.

Proof:

(i) It was shown in Prop. 3.37 that any time bisimulation can be regarded as a partial $\mathcal{T}$-action such that the projections are homomorphisms of partial actions. Furthermore, by the characterisation of partial $\mathcal{T}$-actions and their homomorphisms as E-Coalgebras and homomorphisms of E-Coalgebras, resp., in Thm. 4.17, this means that each time bisimulation induces a Coalgebraic bisimulation.

(ii) Consider a jointly monic Coalgebraic E-Coalgebras bisimulation, so in particular the following diagram commutes

$$
\begin{tikzcd}
 & R \arrow[dl, "\pi_1"'] \arrow[d, "r"'] \arrow[dr, "\pi_2"] & \\
X \arrow[d, "k_1"'] & ER \arrow[dl, "E\pi_1"] \arrow[dr, "E\pi_2"'] & Y \arrow[d, "k_2"] \\
EX & & EY
\end{tikzcd}
\tag{4.16}
$$

Since the span is jointly monic, we have $R \subseteq X \times Y$ and π_1 and π_2 are the two projections. Let $\langle x, y \rangle \in R$. Then $r\langle x, y \rangle : \mathcal{T} \rightharpoonup R$ must satisfy

$$
\begin{aligned}
(E\pi_1 \circ r)\langle x, y \rangle = \pi_1 \circ r\langle x, y \rangle \overset{(4.16)}{=} (k_1 \circ \pi_1)\langle x, y \rangle = k_1(x) \\
(E\pi_2 \circ r)\langle x, y \rangle = \pi_2 \circ r\langle x, y \rangle \overset{(4.16)}{=} (k_2 \circ \pi_2)\langle x, y \rangle = k_2(y)
\end{aligned}
$$

Furthermore since $r\langle x, y \rangle$ has codomain $R \subseteq X \times Y$, we know that

$$
r\langle x, y \rangle(t)\downarrow \Leftrightarrow (\exists \langle x', y' \rangle \in X \times Y).\ r\langle x, y \rangle(t) = \langle x', y' \rangle \wedge \langle x', y' \rangle \in R
$$

Therefore, again using equation (4.16), we must have that

$$
r\langle x, y \rangle(t)\downarrow \Leftrightarrow\ k_1(x)(t)\downarrow \wedge k_2(y)(t)\downarrow
$$

and, putting together the above equations and equivalences, we obtain

$$
r\langle x, y \rangle(t) = \langle x', y' \rangle\ \Leftrightarrow\ k_1(x)(t) = x' \wedge k_2(y)(t) = y' \wedge \langle x', y' \rangle \in R
$$

When writing $x \underset{k_1}{\overset{t}{\rightsquigarrow}} x'$ for $k_1(x)(t) = x'$ and analogously for y and k_2, we get that $\langle x, y \rangle \in R$ implies for all $t \in \mathcal{T}$

$$
\begin{aligned}
x \underset{k_1}{\overset{t}{\rightsquigarrow}} x' \Rightarrow (\exists y' \in Y).\ y \underset{k_2}{\overset{t}{\rightsquigarrow}} y' \wedge \langle x', y' \rangle \in R \\
y \underset{k_2}{\overset{t}{\rightsquigarrow}} y' \Rightarrow (\exists x' \in X).\ x \underset{k_1}{\overset{t}{\rightsquigarrow}} x' \wedge \langle x', y' \rangle \in R
\end{aligned}
$$

which is precisely the definition of time bisimulation from Definition 3.25. □

Note that, since E preserves (weak) pullbacks, it holds that

$$
x\ \sim_t\ y\ \Leftrightarrow\ \nu(x) = \nu(y)
$$

where ν is the unique map to the final E-Coalgebra (see [Tur96, Ch. 12]).

4.3 Partial Actions of Discrete Time

Having introduced the evolution comonad E for a time domain $\langle \mathcal{T}, +, 0\rangle$ and proved that its Coalgebras are the same as partial $\mathcal{T}$-actions, we will now show, for the case $\mathcal{T} = \mathbb{N} = 1^\star$ that $E_\mathbb{N}$ is *cofreely generated* from a functor $B_\mathbb{N}$; in particular, $E_\mathbb{N}$**–Coalg** $\cong B_\mathbb{N}$**–coalg**, and hence TTSs over $\mathbb{N}$ can be described by the simpler coalgebras of the endofunctor $B_\mathbb{N}$. This generalises[7] results from [Jac95a, Jac95b].

Definition 4.22
The endofunctor $B_\mathbb{N}$ on **Set** is given by

$$X \mapsto (1+X) \tag{4.17}$$

For a fixed set X, define the endofunctor $B_\mathbb{N}^X$ by

$$B_\mathbb{N}^X \stackrel{\mathsf{df}}{=} X \times B_\mathbb{N}, \quad Y \mapsto B_\mathbb{N}^X(Y) = X \times (1+Y) \tag{4.18}$$

Moreover, define $X^+ \stackrel{\mathsf{df}}{=} X^\star \setminus \{\varepsilon\}$ to be the set of all finite, non-empty words over X, and let X^ω be the set of all infinite words over X. ■

The main result of the section is the following:

Theorem 4.23 $E_\mathbb{N}$ is cofreely generated from the functor $B_\mathbb{N}$.

To prove it, we first have to show the cofree comonad on $B_\mathbb{N}$ exists:

Proposition 4.24
For any set X, the final $B_\mathbb{N}^X$-coalgebra exists: its carrier is $X^+ + X^\omega$, and the structure map is given by the following chain of isomorphisms

$$\begin{aligned} X^+ + X^\omega &\cong X \times (X^* + X^\omega) \cong X \times ((1+X^+) + X^\omega) \\ &\cong X \times (1 + (X^+ + X^\omega)) = B_\mathbb{N}^X(X^+ + X^\omega) \end{aligned} \tag{4.19}$$

Proof: The constant functor X and $B_\mathbb{N}$ are ω-bicontinuous [Bar93], and so is $B_\mathbb{N}^X$ because ω-bicontinuous functors are closed under finite limits. For such ω-bicontinuous

[7] Note that an unpublished version of [Jac95b] contains a more general result than ours.

functors F, the final F-coalgebras exists and can be computed by applying the (dual of the) *Basic Lemma* from [SP82], viz., by computing the limit of the ω^{op}-chain

$$1 \xleftarrow{!} F1 \xleftarrow{F!} F^2 1 \xleftarrow{F^2!} \cdots \xleftarrow{F^n!} F^{n+1}1 \longleftarrow \cdots$$

where 1 is the final object and $! : F1 \to 1$ is the unique morphism given by finality; the concrete result then just follows by applying this in the case of $F = B^X_{\mathbb{N}}$. □

Since $X^+ + X^\omega \cong X \times (1 + (X^+ + X^\omega))$, we obtain the two projections

$$X \xleftarrow{\mathrm{hd}_X} X^+ + X^\omega \xrightarrow{\mathrm{tl}_X} 1 + (X^+ + X^\omega)$$

where hd stands for *head*, and tl for *tail*: a $z \in X^+ + X^\omega$ is a word over X which is either finite and non-empty, or (countably) infinite. The structure map then divides z into the pair consisting of its *head*, i.e., the first letter (which always exists), and its *tail* of z which may be the empty word ε, or another finite and non-empty word, or the infinite remainder of an infinite word, i.e., the summands of $1 + X^+ + X^\omega$.

Example 4.25

For $X = 1$, we obtain $B^X_{\mathbb{N}}(Y) \cong B_{\mathbb{N}}(Y)$. Therefore, instantiating Proposition 4.24 yields that the final $B_{\mathbb{N}}$-coalgebra exists: its carrier is $1^+ + 1^\omega \cong 1^\star + 1 \cong \mathbb{N} + 1 \cong \mathbb{N}_\infty$, and the structure map is given by the (partial) predecessor function

$$\mathrm{pred}_\infty : \mathbb{N}_\infty \to 1 + \mathbb{N}_\infty, \quad n \mapsto \begin{cases} \star & \text{if } n = 0 \\ m & \text{if } n = m + 1 \\ \infty & \text{if } n = \infty \end{cases}$$

obtained by instantiating the tail-function for this particular case. ◊

Definition 4.26

For a set X, write $D_{\mathbb{N}}X$ for the carrier $X^+ + X^\omega$ of the final $B^X_{\mathbb{N}}$-coalgebra.

We know from Section 2.4 that $D_{\mathbb{N}}$ is the cofree comonad on $B_{\mathbb{N}}$; we write e and d for counit and comultiplication, respectively, of $D_{\mathbb{N}}$. In order to prove Theorem 4.23, we have to show that $E_{\mathbb{N}} = D_{\mathbb{N}}$. To do so, we first show that the two comonads agree on objects by establishing an isomorphism $E_{\mathbb{N}}X \cong D_{\mathbb{N}}X$. Then, we prove that, up to this isomorphism, the comonads also act equally on morphisms. Finally, we show that the comonad structures agree, again up to the isomorphism obtained in the first lemma.

Lemma 4.27 $E_{\mathbb{N}}X \;\cong\; X^+ + X^\omega = D_{\mathbb{N}}X$

Proof: By Corollary 4.7, $\mathrm{spec}(\mathbb{N}) \cong \mathbb{N}_\infty$; therefore, each $e \in E_{\mathbb{N}}X$ is either a function of type $[0,n] \to X$, or of type $\mathbb{N} \to X$, as $\mathrm{dom}(e) \in \mathrm{spec}(\mathbb{N})$. Define a map into $X^+ + X^\omega$ sending the first kind of function to the finite and non-empty word $f(0)f(1)\cdots f(n)$, and the second kind to the infinite word $f(0)f(1)\cdots f(n)f(n+1)\cdots$; in the opposite direction, a word $w = x_0 \cdots x_n$ defines the evolution $e \in E_{\mathbb{N}}X$ such that $e(i) = x_i$ for $i \in [0,n]$, or for $w = x_0x_1\cdots x_nx_{n+1}\cdots$, define $e \in E_{\mathbb{N}}X$ by $e(i) = x_i$, $i \in \mathbb{N}$. These two constructions are easily seen to be mutually inverse. □

Lemma 4.28 Given $f : X \to Y$, $D_{\mathbb{N}}f = E_{\mathbb{N}}f$, up to the isomorphism from Lemma 4.27.

Proof: Given $f : X \to Y$, we will show that $E_{\mathbb{N}}f$ makes the defining diagram (2.7) of $D_{\mathbb{N}}f$ commute, and since $D_{\mathbb{N}}f$ was unique with that property, the desired equality follows. Let therefore $w \in D_{\mathbb{N}}X = X^+ + X^\omega$. For the sake of brevity, we assume $w = x_0 \cdots x_n \in X^+$ for some $n \in \mathbb{N}$, and leave the analogous case $w \in X^\omega$ to the reader. Under the isomorphism of Lemma 4.27, w corresponds to $e : [0,n] \to X$ such that $i \mapsto w_i$, for $0 \le i \le n$. Applying $E_{\mathbb{N}}f$ to e results in the function $f \circ e = [0,n] \to Y$, $i \mapsto f(x_i)$. So if we translate this back to $D_{\mathbb{N}}Y$, we obtain the word $v \stackrel{\mathrm{df}}{=} (fx_0)\cdots(fx_n) \in Y^+$. Let us now check that this assignment makes (2.7) commute.

The left-hand square commutes: first taking the head of w, i.e., x_0, and then applying f, resulting in fx_0, is the same as mapping w to v by f, and then taking v's head, also leading to fx_0. As for the right-hand square, taking the tail of w results in either the empty word ε or $x_1 \cdots x_n$, according to whether $n = 0$ or $n \ge 1$. If $n = 0$, then $v = D_{\mathbb{N}}f(w) = fx_0$ and so $\mathrm{tl}_Y(v) = \varepsilon$, just as for w, and if $n \ge 1$, we obtain $\mathrm{tl}_Y(v) = (fx_1)\cdots(fx_n)$. This shows that also the second square commutes as applying $D_{\mathbb{N}}f$ to the two possible cases for $\mathrm{tl}_X(w)$ is equal to first applying $D_{\mathbb{N}}f$, mapping w to v, and then taking the tail of v. □

Lemma 4.29 The comonad structures for $E_{\mathbb{N}}$ and $D_{\mathbb{N}}$ are the same.

Proof: As for the counits, $e_X = \mathrm{hd}_X : D_{\mathbb{N}}X \to X$ was the counit of $D_{\mathbb{N}}$. Concretely, it is given by taking the first letter of any element of $D_{\mathbb{N}}X$. Under the isomorphism from Lemma 4.27, the first letter of a word in $D_{\mathbb{N}}X$ is simply the value at 0 of the corresponding evolution in $E_{\mathbb{N}}X$, exactly as ε_X was defined for $E_{\mathbb{N}}$.

For the comultiplications d and δ, use the fact that δ for $E_{\mathbb{N}}$ was defined as the unique function making the diagram (2.8) commute. For this, we first characterise $\delta_X : E_{\mathbb{N}}X \to E^2_{\mathbb{N}}X$ in terms of $D_{\mathbb{N}}X$: given $w \in D_{\mathbb{N}}X$, translate it into $e : \mathbb{N} \rightharpoonup X$; this time, consider an infinite word $w \stackrel{\mathsf{df}}{=} x_0x_1\cdots x_nx_{n+1}\cdots$; then e is actually total. Applying δ_X to it results in following partial (actually total) function

$$\lambda t.\begin{cases}\lambda u.e(t+u) & \text{if } e(t)\downarrow \\ \text{undef} & \text{if } e(t)\uparrow\end{cases} \overset{e\ \text{total}}{=} \lambda t.(\lambda u.e(t+u))$$

Translating this back to $D_{\mathbb{N}}X$, the function $\lambda u.e(t+u)$ results in the infinite word $w_t \stackrel{\mathsf{df}}{=} w_tw_{t+1}w_{t+2}\cdots$, and so the whole function $\lambda t.(\lambda u.e(t+u))$ corresponds to the infinite word $w_0w_1\cdots$ over $D_{\mathbb{N}}X$, i.e., the word of words w_i, where w_i is w with the first i letters taken away. All that is now left to check is that this candidate of a function $D_{\mathbb{N}}X \to D^2_{\mathbb{N}}X$ makes (2.8) commute. The triangle says applying the candidate function to w and taking the head should leave w unchanged; in our case, the head of $w_0w_1\cdots$ is $w_0 = w$, i.e., the triangle commutes. As for the square, the right-down path takes w to w_1 and finally to $(w_i)_{i\geq 1}$. The down-right path sends w first to $(w_i)_{i\in\mathbb{N}}$ and then to $(w_i)_{i\geq 1}$, hence the square also commutes. As d_X was the unique function making (2.8) commute, $\delta_X = d_X$, i.e., also the comultiplications coincide. □

This concludes the proof of Theorem 4.23. As a consequence, we obtain:

Corollary 4.30 $E_{\mathbb{N}}$**–Coalg** $\cong B_{\mathbb{N}}$**–coalg**.

Hence, TTSs over $\mathbb{N}$ are the same as $B_{\mathbb{N}}$-coalgebras. The correspondence identifies $\overset{1}{\leadsto}$-transitions in a TTS over $\mathbb{N}$ with the (deterministic) 'next-step' relation defined by a $B_{\mathbb{N}}$-coalgebra, and the $\overset{1}{\leadsto}$-transitions determine *all* $\overset{n}{\leadsto}$-transitions because each $n \in \mathbb{N}$ can be written as $1+\cdots+1$, and the TTS must satisfy (Continuity). In this sense, the last result shows that discrete quantitative time is the same as global qualitative time.

Remark 4.31

Corollary 4.30 also implies that, if $\langle R, r\rangle$ is a $B_{\mathbb{N}}$-bisimulation between $\langle X_1, k_1\rangle$ and $\langle X_2, k_2\rangle$, then the corresponding span $k_1 \leftarrow r \rightarrow k_2$ of $B_{\mathbb{N}}$-coalgebras uniquely corresponds to a span of $E_{\mathbb{N}}$-Coalgebras, viz., via the isomorphism, we obtain the span of $E_{\mathbb{N}}$-Coalgebras ${k_1}^{\infty} \leftarrow r^{\infty} \rightarrow {k_2}^{\infty}$ of the respective coinductive extensions, in other words: an $E_{\mathbb{N}}$-bisimulation $\langle R, r^{\infty}\rangle$ between $\langle X_1, {k_1}^{\infty}\rangle$ and $\langle X_2, {k_2}^{\infty}\rangle$. The converse also holds, using the isomorphism from the corollary in the other direction. So we also obtain a one-to-one correspondence between $E_{\mathbb{N}}$- and $B_{\mathbb{N}}$-bisimulations. $\Diamond$

Remark 4.32

It is possible to show a more general result then Theorem 4.23. Let C be a finite set. Then

(4.20) $$E_C \stackrel{\mathsf{df}}{=} E_{C^\star} \text{ is cofreely generated from } B_C \stackrel{\mathsf{df}}{=} (1+X)^C$$

Theorem 4.23 is obtained by instantiating (4.20) for $C = 1$ since $\mathbb{N} = 1^\star$. Again, one can look at the two categories of coalgebras and obtain the consequence that

(4.21) $$E_C\text{–}\mathbf{Coalg} \cong B_C\text{–}\mathbf{coalg}$$

This means that, in the case of local qualitative time, we can use coalgebras for a functor, rather than a comonad, to describe TTSs over $C^\star$, i.e., partial $C^\star$-actions. This nicely matches the way calculi like PMC are described in the literature: they only use transitions of the form $p \stackrel{c}{\rightsquigarrow} p'$ for $c \in C$, rather than $p \stackrel{w}{\rightsquigarrow} p'$ for $w \in C^\star$. Note that $B_C = (B_{\mathbb{N}})^C$, the C-fold power of $B_{\mathbb{N}}$, and in this sense local qualitative time is a C-dimensional version of global qualitative time, or discrete quantitative time. $\Diamond$

4.4 Distributing Total over Partial Monoid Actions

Now that we have categorical characterisations of both partial and total monoid actions, we are finally in a position to combine these in order to present a characterisation of the biactions we introduced in Section 3.4. Recall that an *antichain* in a partially ordered set $\langle P, \leq\rangle$ is a set $A \subseteq P$ such that, for all $p, q \in A$, $p \neq q$ implies $p||q$; obviously, if $\leq$ is linear, all antichains are of cardinality ≤ 1.

Definition 4.33

A time domain $\mathcal{T}$ is *antichain monotone* if

(4.22) $$t||u \wedge\ u \leq u' \ \Rightarrow\ t||u'$$

for all $t, u, u' \in \mathcal{T}$. ■

Note that (4.22) could equivalently be formulated as ‘if, for $t,u \in \mathcal{T}$ such that $t \neq u$, $\{t,u\}$ is an antichain and $u \leq u'$, then also $\{t,u'\}$ is an antichain such that $t \neq u'$,’ hence the name of the property. Alternatively, one could describe it by postulating that ‘the principal filters on incomparable elements are disjoint,’ or ‘the relation $t \| \_$ is monotone with respect to $\leq$.’ Yet another equivalent characterisation would be to say that, whenever $t \| u$ holds, also $t \# u$ holds, meaning that t and u have no common upper bound. A final equivalent formulation is to say that ‘principal ideals are linearly ordered’; this is particularly nice because, regarding the principal ideal of a $t \in \mathcal{T}$ as the ‘history’ of t, then antichain monotonicity postulates a *unique history* (the linear order on the principal ideal allows only one ‘path’ to reach t). We will now first present some examples of antichain monotone time domains.

Example 4.34

1. Obviously, any linear time domain is antichain monotone since there are no two-element antichains, hence $\mathbb{N}$ and $\mathbb{R}$ are antichain monotone.

2. The free monoid $C^\star$ is also antichain monotone. Without loss of generality, we can assume that $|C| > 1$ since $1^\star = \mathbb{N}$ is linear anyway. Suppose now $w_1, w_2 \in C^\star$ such that $w_1 \neq w_2$ and $\{w_1, w_2\}$ is an antichain. This implies that there is a (potentially empty) shared prefix $w_s \in C^\star$ followed by different letters $c_1 \neq c_2$ in w_1 and w_2 and some (possibly empty) remainders v_1 and v_2, respectively, i.e., $w_i = w_s c_i v_i$ for $i \in \{1,2\}$. Let now be $w \in C^\star$ such that $w_2 \leq w$. This, by definition of $\leq$, means that w_2 is a prefix of w, in other words, there must be $v \in C^\star$ such that $w = w_2 v = (w_s c_2 v_2) v$ and it follows that w is incomparable with w_1, i.e., $\{w_1, w\}$ is also an antichain. $\Diamond$

Note that the time domain $\mathbb{N} \times \mathbb{N}$ is *not* antichain monotone: despite $\langle 1,0\rangle \| \langle 0,1\rangle$, we have $\langle 0,1\rangle \leq \langle 2,1\rangle$ and also $\langle 1,0\rangle \leq \langle 2,1\rangle$, in other words, $\langle 1,0\rangle \not\| \langle 2,1\rangle$. Consequently, the property is not preserved by products, as $\mathbb{N}$, being linear, is antichain monotone.

Theorem 4.35

Let $\mathcal{T}$ be an antichain monotone time domain. The map

$$(4.23)\quad \ell_X : \mathcal{T} \times EX \to E(\mathcal{T} \times X),\ \langle t, e\rangle \mapsto \lambda u. \begin{cases} \langle t-u, e(0)\rangle & \text{if } u < t \\ \langle 0, e(0)\rangle & \text{if } u = t \\ \langle 0, e(u-t)\rangle & \text{if } t < u \wedge e(u-t)\downarrow \\ \text{undef} & \text{otherwise} \end{cases}$$

induces a distributive law of the monad $\langle \mathcal{T} \times (\cdot), \eta, \mu\rangle$ for total $\mathcal{T}$-actions (as described in Proposition 4.1) over the evolution comonad $\langle E, \varepsilon, \delta\rangle$.

Note that we use the strict case distinction to facilitate the calculations, it would of course be possible to combine the first and second case, or only use non-strict inequalities since the cases yield the same results when overlapping. The strict case distinctions also mean that we get two disjoint cases for being undefined:

$$(4.24)\qquad \ell_X\langle t, e\rangle(u)\uparrow \Leftrightarrow\ (t < u \wedge\ e(u-t)\uparrow) \vee (t \| u)$$

We will split the proof into several lemmas.

Lemma 4.36

The map ℓ_X from (4.23) is well-defined and natural.

Proof: As for well-definedness, given $t \in \mathcal{T}$ and $e \in EX$, we have to show that $\ell_X\langle t, e\rangle$ is an evolution on $\mathcal{T} \times X$, i.e., we have to establish (4.1) and (4.2). As for the first axiom, if $t = 0$, $\ell_x\langle t, e\rangle(0) = \langle 0, e(0)\rangle$, in particular it is defined; if $t > 0$, we obtain the value $\langle t, e(0)\rangle$, and so we are done. In order to show (4.2), we are going to show its equivalent contra-position (4.3). Therefore, let $u \in \mathcal{T}$ such that $\ell_X\langle t, e\rangle(u)\uparrow$, and assume $u \leq u'$. According to (4.24), two possible cases arise:

1. Assume $t < u \wedge\ e(u-t)\uparrow$. Then, because $\leq$ is transitive, also $t < u'$, i.e., $(u'-t)\downarrow$, and since $(\cdot) - t$ is monotone, we get $u - t \leq u' - t$; furthermore, because e satisfies (4.3), it follows that $e(u'-t)\uparrow$, implying $\ell_X\langle t, e\rangle(u')\uparrow$.

2. Assume $t\|u$. By antichain monotonicity of $\mathcal{T}$, this implies $t\|u'$, and so also $\ell_X\langle t, e\rangle(u')\uparrow$.

The proof of naturality (which is trivial) is left to the reader. $\square$

Lemma 4.37 ℓ_X respects η.

Proof: For $e \in EX$, we calculate:

$$\begin{aligned}
\ell_X(\eta_{EX}(0)) &= \lambda u.\begin{cases} \langle 0-u, e(0)\rangle & \text{if } u<0 \\ \langle 0, e(0)\rangle & \text{if } u=0 \\ \langle 0, e(u-0)\rangle & \text{if } 0<u \wedge e(u-0)\downarrow \\ \text{undef} & \text{otherwise} \end{cases} \\
&= \lambda u.\begin{cases} \langle 0, e(u)\rangle & \text{if } e(u)\downarrow \\ \text{undef} & \text{if } e(u)\uparrow \end{cases} = \lambda u.\begin{cases} \eta_X(e(u)) & \text{if } e(u)\downarrow \\ \text{undef} & \text{if } e(u)\uparrow \end{cases} = E\eta_X(e)
\end{aligned}$$

and so the η-diagram from (2.9) commutes. $\square$

Lemma 4.38 ℓ_X respects ε.

Proof: Let $t \in \mathcal{T}$ and $e \in EX$. Then calculate:

$$\begin{aligned}
\varepsilon_{\mathcal{T}\times X}(\ell_X\langle t, e\rangle) &= \begin{cases} \langle t-0, e(0)\rangle & \text{if } t>0 \\ \langle 0, e(0)\rangle & \text{if } t=0 \end{cases} \\
&= \langle t, e(0)\rangle = \langle t, \varepsilon_X(e)\rangle = (\mathcal{T}\times\varepsilon_X)\langle t, e\rangle
\end{aligned}$$

Hence, also the ε-diagram from (2.9) commutes. $\square$

Lemma 4.39 ℓ_X respects μ.

Proof: Let $t, u \in \mathcal{T}$ and $e \in EX$, and set $h_t \stackrel{\mathsf{df}}{=} \ell_X t, e \in E(\mathcal{T}\times X)$. We then have to chase $\langle u, \langle t, e\rangle\rangle$ around the two possible paths of the μ-diagram in (2.9). The first path, given by $\ell_X \circ \mu_{EX}$, yields, omitting the intermediate results:

$$\mathsf{LHS} \stackrel{\mathsf{df}}{=} \lambda s.\begin{cases} \langle (t+u)-s, e(0)\rangle & \text{if } s<t+u \\ \langle 0, e(0)\rangle & \text{if } s=t+u \\ \langle 0, e(s-(t-u))\rangle & \text{if } t+u<s \wedge e(s-(t+u))\downarrow \\ \text{undef} & \text{otherwise} \end{cases}$$

For the other path, $E\mu_X \circ \ell_{\mathcal{T}\times X} \circ \mathcal{T} \times \ell_X$, omitting the intermediate steps, we obtain the following result:

$$\mathsf{RHS} \stackrel{\mathsf{df}}{=} \lambda s. \begin{cases} \langle (u-s)+t, e(0)\rangle & \text{if } s < u \\ \langle t, e(0)\rangle & \text{if } s = u \\ \langle t-(s-u), e(0)\rangle & \text{if } u < s \wedge\ s-u < t \\ \langle 0, e(0)\rangle & \text{if } u < s \wedge\ s-u = t \\ \langle 0, e((s-u)-t)\rangle & \text{if } u < s \wedge t < s-u-u \wedge\ e((s-u)-t)\downarrow \\ \text{undef} & \text{otherwise} \end{cases}$$

We now have to show that $\mathsf{LHS}(s) \simeq \mathsf{RHS}(s)$. To do so, we split the Kleene equality into two implications, and prove them both separately. In several places, we will use Proposition 3.17 and Lemma 3.21, the fact that $-$ is a partial $\mathcal{T}$-action, as well as the assumption of antichain monotonicity, always without reference. Let $s \in \mathcal{T}$. We will first show $\mathsf{LHS}(s) \precsim \mathsf{RHS}(s)$. Assume therefore that $\mathsf{LHS}(s)\downarrow$. This results in the following case distinction.

1. $s < u+t$. This prohibits $s||u$, and so we get the following three sub-cases:

   (a) $s < u$. This implies $\mathsf{RHS}(s)\downarrow$. Moreover, we get

   $$\mathsf{LHS}(s) = \langle (u+t)-s, e(0)\rangle = \langle (u-s)+t, e(0)\rangle = \mathsf{RHS}(s)$$

   (b) $s = u$. Again, $\mathsf{RHS}(s)\downarrow$, and

   $$\mathsf{LHS}(s) = \langle (u+t)-s, e(0)\rangle = \langle (u+t)-u, e(0)\rangle = \langle t, e(0)\rangle = \mathsf{RHS}(s)$$

   (c) $u < s$. Since $s < u+t$ implies $s-u < t$, $\mathsf{RHS}(s)\downarrow$. Then:

   $$\mathsf{LHS}(s) = \langle (u+t)-s, e(0)\rangle = \langle t-(s-u), e(0)\rangle = \mathsf{RHS}(s)$$

2. $s = u+t$. Again, $s||u$ is impossible, leading to three sub-cases:

   (a) $s < u$. Then $\mathsf{RHS}(s)\downarrow$ and

   $$\mathsf{LHS}(s) = \langle 0, e(0)\rangle = \langle (u+t)-s, e(0)\rangle = \langle (u-s)+t, e(0)\rangle = \mathsf{RHS}(s)$$

(b) $s = u$. Hence $t = 0$ and $\mathsf{RHS}(s)\downarrow$, and

$$\mathsf{LHS}(s) = \langle 0, e(0)\rangle = \langle t, e(0)\rangle = \mathsf{RHS}(s)$$

(c) $u < s$. Since $s = u + t$ implies $s - u = t$, we get $\mathsf{RHS}(s)\downarrow$ and

$$\mathsf{LHS}(s) = \langle 0, e(0)\rangle = \mathsf{RHS}(s)$$

3. $u + t < s \wedge\ e(s - (u + t))\downarrow$. As before, $s||u$ is impossible, and of the three resulting cases, both $s < u$ and $s = u$ are as well impossible. Hence, it must holds that $u < s$. From $u + t < s$, we obtain $t < s - u$ and $(s - u) - t = s - (u + t)$, hence $\mathsf{RHS}(s)\downarrow$. Moreover,

$$\mathsf{LHS}(s) = \langle 0, e(s - (u + t))\rangle = \langle 0, e((s - u) - t)\rangle = \mathsf{RHS}(s)$$

Hence, $\mathsf{LHS}(s) \sqsubseteq\!\!\!\!\sim \mathsf{RHS}(s)$. We leave the converse to the reader, and merely remark that a similar case distinction has to be used, this time into five different cases, corresponding to the five different cases when $\mathsf{RHS}(s)\downarrow$ can hold; the proof requires essentially the same equalities as for the other direction. Hence we have shown $\mathsf{LHS}(s) \simeq \mathsf{RHS}(s)$, i.e., that the μ-diagram in (2.9) commutes. □

Lemma 4.40 ℓ_X respects δ.

Proof: Let $t \in \mathcal{T}$ and $e \in EX$. We again have to show that chasing $\langle t, e\rangle$ around the two different paths of the δ-diagram in (2.9) yields the same result. The first path, $\delta_{\mathcal{T}\times X} \circ \ell_X$, produces the following result, omitting intermediate steps:

$$\mathsf{LHS} \stackrel{\mathsf{df}}{=} \lambda u. \begin{cases} \left(\lambda s. \begin{cases} \langle t - (p + s), e(0)\rangle & \text{if } p + s < t \\ \langle 0, e(0)\rangle & \text{if } p + s = t \\ \langle 0, e((u + s) - t)\rangle & \text{if } t < u + s \wedge\ e((u + s) - t)\downarrow \\ \text{undef} & \text{otherwise} \end{cases}\right) \\ \text{undef} \end{cases}$$

where $\mathsf{LHS}(u)\downarrow \Leftrightarrow\ (u < t) \vee (u = t) \vee (t < u \wedge e(u - t)\downarrow)$.

The other path, $E\ell_X \circ \ell_{EX} \circ (\mathcal{T} \times \delta_X)$, yields the following result, again dropping the intermediate calculations:

$$\mathsf{RHS} \stackrel{\mathsf{df}}{=} \lambda u. \begin{cases} \left(\lambda s. \begin{cases} \langle (t-u)-s, e(0)\rangle & \text{if } s < t-u \\ \langle 0, e(0)\rangle & \text{if } s = t-u \\ \langle 0, e(s-(t-u))\rangle & \text{if} t-u < s \wedge\ e(s-(t-u))\downarrow \\ \text{undef} & \text{otherwise} \end{cases}\right) \\ \left(\lambda s. \begin{cases} \langle 0, e(s)\rangle & \text{if} e(s)\downarrow \\ \text{undef} & \text{otherwise} \end{cases}\right) \\ \left(\lambda s. \begin{cases} \langle 0, e((u-t)+s)\rangle & \text{if} e((u-t)+s)\downarrow \\ \text{undef} & \text{otherwise} \end{cases}\right) \\ \text{undef} \end{cases}$$

where the three cases for $\mathsf{RHS}(u)$ being defined are, from the top down $u < t$, $u = t$, and $t < u \wedge\ e(u-t)\downarrow$. It should be entirely obvious that $\mathsf{LHS}(u)\downarrow \Leftrightarrow \mathsf{RHS}(u)\downarrow$. It remains to show that $\mathsf{LHS}(u)(s) \simeq \mathsf{RHS}(u)(s)$, we which will again separate into the two Kleene implications.

To show $\mathsf{LHS}(u)(s) \precsim \mathsf{RHS}(u)(s)$, assume $\mathsf{LHS}(u)(s)\downarrow$. This leads to three cases, arising from $\mathsf{LHS}(u)\downarrow$. Using antichain monotonicity, Proposition 3.17, Lemma 3.21, and the fact the $-$ is a partial $\mathcal{T}$-action, all cases and subcases can be dealt with very similarly to the proof of Lemma 4.39. The Kleene implicatiion $\mathsf{RHS}(u)(s) \precsim \mathsf{LHS}(u)(s)$ can be shown in an analogous fashion, concluding the proof of the lemma. □

The last five lemmas correspond to the proof of Theorem 4.35. The $\mathcal{T}$ biactions are obtained as bialgebras by distributing total over partial $\mathcal{T}$-actions given by the distributive law ℓ from the last theorem:

Theorem 4.41

Let $\mathcal{T}$ be an antichain monotone time domain. Then ℓ-bialgebras are $\mathcal{T}$-biactions. Moreover, the morphisms of ℓ-bialgebras are homomorphisms of $\mathcal{T}$-biactions.

Proof: Consider an ℓ-bialgebra on a set X. By definition, this implies that we get a $(\mathcal{T} \times \_)$-Algebra $h : \mathcal{T} \times X \to X$ and an E-Coalgebra $k : X \to EX$ which, moreover, satisfy (2.10). By Proposition 4.1, h corresponds to a total $\mathcal{T}$-action on X, and by Theorem 4.17, k corresponds to a partial $\mathcal{T}$-action on X. We thus only have to show that (2.10) implies the axiom (3.28) of biactions. The diagram (2.10) states that two functions are equal: $k \circ h$, and $Eh \circ \ell_X \circ (\mathcal{T} \times k)$. Let us therefore calculate the values of these two functions, first for $k \circ h$.

$$\langle t, x\rangle \overset{h}{\mapsto} h(t,x) \overset{k}{\mapsto} k(h(t,x)) \simeq \lambda u.\ k(h(t,x))(u)$$

Using the notations for partial and total actions, this result becomes:

$$(k \circ h)\langle t, x\rangle(u) \simeq (t \cdot x) * u$$

The other path around the diagram results in the following computations:

$$\begin{aligned}
\langle t, x\rangle \overset{\mathcal{T}\times k}{\longmapsto} \langle t, k(x)\rangle \overset{\ell_X}{\mapsto} \lambda u.\ &\begin{cases} \langle t-u, k(x)(0)\rangle & \text{if } u < t \\ \langle 0, k(x)(0)\rangle & \text{if } u = t \\ \langle 0, k(x)(u-t)\rangle & \text{if } t < u \wedge\ k(x)(u-t)\downarrow \\ \text{undef} & \text{otherwise} \end{cases} \\
= \lambda u.\ &\begin{cases} \langle t-u, x\rangle & \text{if } u < t \\ \langle 0, x\rangle & \text{if } u = t \\ \langle 0, x * (u-t)\rangle & \text{if } t < u \wedge\ k(x)(u-t)\downarrow \\ \text{undef} & \text{otherwise} \end{cases} \\
\overset{Eh}{\mapsto} \lambda u.\ &\begin{cases} (t-u) \cdot x & \text{if } u < t \\ x & \text{if } u = t \\ x * (u-t) & \text{if } t < u \wedge\ x * (u-t)\downarrow \\ \text{undef} & \text{otherwise} \end{cases}
\end{aligned}$$

During these calculations, we have used that, because k is a Coalgebra, $k(x)(0) = x$, and because h is an Algebra, $h(0,x) = x$.

The equality of the two functions means that, when evaluating them both at $u = t$, we must get the same result, i.e.,

$$(t \cdot x) * t \simeq (k \circ h)\langle t, x\rangle(t) \simeq (Eh \circ \ell_X \circ (\mathcal{T} \times k))\langle t, x\rangle(t) = x$$

In other words, k and h precisely satisfy (3.28) for biactions. The claim about the morphisms follows trivially from the fact that the morphisms in ℓ**–Bialg** are simply such maps of the carrier which are both an Algebra and a Coalgebra homomorphism, in other words: such maps which are both a homomorphism of total and of partial $\mathcal{T}$-actions. $\square$

For the converse, viz., describing $\mathcal{T}$-biactions as ℓ-bialgebras, we only obtain:

Theorem 4.42
Let $\mathcal{T}$ be an antichain monotonic time domain. Then a $\mathcal{T}$-biaction on X defines an ℓ-bialgebra with the same values of $(t \cdot p) * u$ for $t \leq u$ and $u \leq t$.

Proof: If X carries a $\mathcal{T}$-biaction, it in particular carries a total and a partial $\mathcal{T}$-action. By Proposition 4.1 and Theorem 4.17, these are equivalent to a $(\mathcal{T} \times \_)$-Algebra h and an E-Coalgebra k, respectively. Furthermore, applying Proposition 3.41, the value of $(t \cdot p) * u$, for comparable t and u, is completely determined by either the total or the partial action, i.e., by h or k. Furthermore, inspecting the definition of ℓ and the meaning of (2.10) as explained in the previous proof, one obtains precisely the same value for the ℓ-bialgebra. $\square$

For linear time domains, $t||u$ never holds, hence:

Corollary 4.43 For $\mathcal{T}$ a linear time domain, $\mathcal{T}$**–BiAct** $\cong$ ℓ**–Bialg**. $\square$

We do not obtain an exact correspondence between bialgebras and biactions because it is possible that a biaction still can let $(t \cdot p) * u$ be defined for $t||u$, while any ℓ-bialgebra *necessarily* is undefined, cf. the definition of ℓ and the remarks following Proposition 3.41. As a simple example consider the (completely total!) biaction of $\mathcal{T} \stackrel{\text{df}}{=} \{a,b\}^\star$ on the singleton set $X = \{1\}$. Then $(a \cdot 1) * b = 1$ although $a||b$. Consequently, this biaction *cannot* be described as a ℓ-bialgebra.

Combining the last two theorems, we obtain that, for antichain monotone time domains $\mathcal{T}$, there is a *retraction* ℓ**–Bialg** $\lhd$ $\mathcal{T}$**–BiAct** between the categories of ℓ-bialgebras and $\mathcal{T}$-biactions: we get functors in both directions such that one composite is the identity on ℓ**–Bialg**.

Note that, in order for the bialgebraic characterisation to work, we need $\mathcal{T}$ to be antichain monotone, otherwise ℓ is not even well-defined. This raises the question, since the single axiom (3.28) for biactions makes sense for an arbitrary time domain, why the additional axiom is necessary. At present, our best guess is that this must be imposed by trying to use bialgebras.

As for an alternative characterisation of biactions, recall that total $\mathcal{T}$-actions can alternatively be described as Coalgebras for the comonad $D = (\_)^{\mathcal{T}}$ (see Proposition 4.3 in Section 4.1.2); moreover, the distributive law ℓ induces a lifting $\tilde{E}$ of E to M–**Act** $\cong D$–**Coalg** such that $\tilde{E}$–**Coalg** $\cong \ell$–**Bialg** (by [TP97, Rem. 7.1]). Such liftings of a comonad to a category of Coalgebras is, by the dual of results from [Bec69, BW85], equivalent to a distributive law of comonads $ED \Rightarrow DE$, which then allows to form the composite comonad ED such that ED–**Coalg** $\cong \tilde{E}$–**Coalg**, dualising results from [Jac94]. As a consequence, we get

$$ED\text{–}\mathbf{Coalg} \cong \tilde{E}\text{–}\mathbf{Coalg} \cong \ell\text{–}\mathbf{Bialg}$$

yielding a purely comonadic characterisation of the ℓ-bialgebras; we do currently not have concrete descriptions of the involved data, but perhaps subtly changing little bits of them might result in a more precise characterisation of biactions.

Chapter 5

Abstract Rules for Timed Processes

In this chapter, we present an abstract categorical framework for defining the operational semantics of timed processes. This is obtained by generalising the abstract, categorical approach of [TP97] from behaviour functors to *behaviour comonads*, in order to accommodate our categorical description of timed processes as Coalgebras for the evolution comonad, as presented in the previous chapter.

The chapter is structured as follows. First, we recall the *bialgebraic* approach to operational semantics introduced in [TP97]. There, *abstract operational rules* were given by a special kind of natural transformation, parameterised by *functorial* notions of *signature* and *behaviour*; the naturality of such rules (in the categorical sense) essentially ensures the 'good' properties of the corresponding concrete rules derived from the abstract ones. It was also shown that some of the most well-known concrete formats, in particular *GSOS* [BIM95], can be derived from abstract rules.

In Section 4.3, we have seen that the evolution comonad $E_{\mathbb{N}}$ over the naturals is cofreely generated by the behaviour functor $B_{\mathbb{N}}$; more specifically, this implies that the $E_{\mathbb{N}}$-Coalgebras are the same as the standard $B_{\mathbb{N}}$-coalgebras. Hence, we can directly apply the abstract framework, as it is, to the case of timed processes over discrete time (and, more generally, over arbitrary free monoids, see Remark 4.32).

In the general case, i.e., for an arbitrary time domain, the evolution comonad is not cofreely generated (consider, e.g., $\mathcal{T} = \mathbb{R}$), and our operational models, the TTSs, are necessarily Coalgebras for a *comonad*, rather than just for an endofunctor. Although much of the work of [TP97] is stated in terms of distributive laws, it cannot fully deal with this more general case, only certain aspects of the general theory carry over, viz., the treatment of bisimulation (as the definition does not change for comonads) and final models.

Most importantly, specifying the operational semantics by (operational rules corresponding to) a simpler kind of natural transformation, which then *induces* a distributive law and subsequently forms the basis for syntactic rule formats, is only possible for cofree comonads and coalgebras for a functor. After some motivation how the abstract framework could be generalised to include our more general case, we present a general theory of 'well-behaved' rules for behaviour comonads, which we call *comonadic SOS (CSOS)*. Such abstract CSOS rules are more expressive than the ones presented in [TP97].

These CSOS rules are very general: in fact, when instantiated for the evolution comonad, they are more general than actually needed to model languages for timed processes (as they exist in the literature). Therefore, in the last section, we present slightly less powerful operational rules for timed processes, which we call *abstract temporal rules*. As will be shown in the next chapter, such rules are still sufficiently powerful to include all important instances of timed process calculi, while being at least easier to approximate (if not to capture completely) by syntactic means.

5.1 Bialgebraic Semantics

This section provides a summary of the necessary background from [TP97]. Essentially, the paper presents a theory of 'well-behaved' (structural) operational semantics, using categorical methods. Here, 'well-behaved' means that the semantics satisfies such important properties as '(an appropriate notion of) bisimulation is a congruence,' or 'there is an adequate denotational semantics with respect to the operational semantics.' Concrete instantiations were shown to include well-known *syntactic rule formats* whose good properties are, in this fashion, conceptually *explained*, rather than merely established.

Note that this last remark is by no means aimed at invalidating or depreciating previously obtained results on rule formats, e.g., [dS85, Fok94, BIM95, FvG96]: we wholeheartedly acknowledge that they constitute an important scientific contribution. However, they are unsatisfactory with respect to issues like modularity: they cannot be extended easily, and it is also hard (if not impossible) to transfer results once the operational model is different from standard LTSs, e.g., when using more specific kinds of transition systems like probabilistic LTSs [vGSS95], or TTSs as presented in Section 3.2.1. Consequently, one ends up proving similar results at the (meta) level of formats, while one wanted to avoid precisely that, on the lower level of languages, by using rule formats in the first place.

Coming back to [TP97], the motivating, and arguably the most important, example of such syntactic rule formats is *GSOS* [BIM95]. Consider the following kind of operational rules, over a given (first-order) signature Σ, a finite(!) set A of labels, and a countable set X of 'meta variables' (representing processes):

$$\frac{\{x_i \xrightarrow{a} y_{ij}^a\}_{1\leq j\leq m_i^a}^{1\leq i\leq n, a\in A_i} \qquad \{x_i \stackrel{b}{\not\rightarrow}\}_{b\in B_i}^{1\leq i\leq n}}{\sigma\langle x_1,\ldots,x_n\rangle \xrightarrow{c} \theta} \tag{5.1}$$

where $\sigma\in\Sigma$ is an n-ary function symbol, all the x_i and y_{ij}^a are variables in X, $a,b,c\in A$, $A_i,B_i\subseteq A$, and θ is a term over Σ and X. Then say that such a rule (5.1) is in *GSOS format*, or simply is a *GSOS rule*, if all the x_i and the y_{ij}^a are distinct, and, moreover, these are also the only variables occurring in the result term θ. Say that a *set* of GSOS rules is *image finite* if, for each operator symbol $\sigma\in\Sigma$ and each action $c\in A$, there are only finitely many rules (5.1) with matching σ and c in the conclusion; so in particular, any *finite* set of GSOS rules is image finite.

Next, recall from (1.1) the endofunctor B_A on **Set** given by $B_AX = \mathcal{P}_{\mathsf{fi}}(X)^A$, i.e., B_AX contains all functions from labels A to finite subsets of X. We have already seen that the B_A-coalgebras are precisely (image finite) LTSs. The following was shown in [TP97, Theorem 1.1], where Σ is the functor associated to the signature of the same name, and $T=\Sigma^\star$ is the free (term) monad on Σ, cf. Section 2.4:

Theorem 5.1

There is a correspondence between natural transformations of type

$$\rho : \Sigma(\mathrm{Id}\times B_A) \Rightarrow B_AT \tag{5.2}$$

and image finite sets of GSOS rules (5.1) (over a fixed denumerable set X of variables). Moreover, this correspondence is one-to-one up to equivalence of sets of rules. $\square$

Note the fact that certain natural transformations using the behaviour functor for image finite LTSs correspond to image finite sets of GSOS rules (as we shall see this is not coincidental). To illustrate this correspondence, we can translate (5.2) into concrete rules as follows. Recalling the definition of the functor Σ, for each set X of variables and each n-ary operator σ in the signature Σ, ρ 'contains' a map of the following type, which is natural in X:

$$[\![\sigma]\!] : (X \times \mathcal{P}_{\mathsf{fi}}(X)^A)^n \to \mathcal{P}_{\mathsf{fi}}(TX)^A$$

Its arguments intuitively correspond to the premises of a rule like (5.1): they are n pairs of variables $x_i \in X$ and 'behaviours' $\beta_i \in \mathcal{P}_{\mathsf{fi}}(X)^A$, interpreted as the *names* and *transitions* of the argument processes, respectively, the latter being described by mapping labels to targets of transitions with that label, i.e., we encode $x_i \xrightarrow{a} x_i'$ by saying that $x_i' \in \beta_i(a)$. Its result, corresponding to the conclusion of GSOS rules (5.1), is a behaviour which, for a given input, encodes the transitions of the composite process $\sigma(x_1, \ldots, x_n)$ *under the assumption* that the x_i behave as specified by the β_i.

Note that the (transitions specified by the) β_i only have 'simple' targets in the sense that the targets can only be 'variables' contained in X, while the conclusion can have arbitrary terms in TX as targets; this is the counterpart of the fact that a rule like (5.1) has the same restrictions on what kinds of transitions are allowed in the premises and in the conclusion, respectively. Furthermore, at this point also the reason for the restriction to image finite sets of GSOS rules becomes clear: otherwise, $[\![\sigma]\!]$ as above might not even be well-defined.

The fact that the rules have to be natural precisely accounts for the GSOS conditions on occurrences of variables and their being distinct, and vice versa. To see this, consider what it means for ρ to be natural, viz., given a function $f : X \to Y$ the following diagram has to commute:

$$\begin{tikzcd} \Sigma(X \times B_A X) \arrow[r, "\rho_X"] \arrow[d, "\Sigma(f \times B_A f)"'] & B_A TX \arrow[d, "B_A Tf"] \\ \Sigma(Y \times B_A Y) \arrow[r, "\rho_Y"'] & B_A TY \end{tikzcd} \tag{5.3}$$

Intuitively, this expresses that first applying ρ for X and then *renaming* the variables in the resulting terms according to f is equal to first renaming in the argument processes, followed by applying ρ for Y: application of the rules ρ is *invariant under variable renamings*.

Thinking about this from a different perspective, if rules of the form (5.1) are supposed to make (5.3) commute, it simply must not be possible to derive a transition under any assumption about variables being equal or distinct: otherwise, after deploying an appropriate renaming, the rule would not longer be applicable, and so the square would not commute. Similarly, if the target θ of a rule (5.1) were to contain a variable, say x, which would *not* occur anywhere among the arguments or the variables in the premises, it would be easy to construct a renaming f (even on X itself) such that the above square (5.3) would no longer commute, viz., by leaving the premises unchanged by f and renaming x to some $x' \neq x$ (possible because X was assumed to be infinite!).

Note that it is vital to include the Id-component, representing the names of arguments, in the type of the natural transformation. Consider, e.g., the following standard rule for parallel composition:

$$\frac{x \xrightarrow{a} x'}{x|y \xrightarrow{a} x'|y}$$

When using abstract operational rules of the type (5.2), it becomes very easy to model the rule by the following map, which is indeed natural in X:

$$[\![|]\!]\langle\langle x, \beta_x\rangle, \langle y, \beta_y\rangle\rangle = \lambda a.\ \{x'|y \mid x' \in \beta_x(a)\}$$

If, on the other hand, one uses only rules given by a natural transformation $\Sigma B_A \Rightarrow B_A T$, as in [Tur96], one has to resort to some ‘tricks’ to model the rules for such operators where only a subset of the argument processes perform transitions while others remain untouched (see [Tur96] for the details).

Natural transformations ρ of type (5.2) also make sense for behaviours other than B_A, and on different categories, provided they have ‘enough’ structure to interpret the rules ρ, so [TP97] introduced *abstract operational rules* as a natural transformation

$$\Sigma(\mathrm{Id} \times B) \Rightarrow BT \tag{5.4}$$

for arbitrary functorial notions of *signature* Σ (which is simply a polynomial functor in the usual first-order case) with freely generated monad T, and of *behaviour* B. The functor Σ describes the syntax of the language, while B gives the 'type' of computation under consideration, respectively: the Σ-algebras should be interpretations of language constructs and the B-coalgebras should correspond to the considered notion of 'transition systems' (in the widest possible sense).

Abstract rules (5.4) induce a lifting $\tilde{T}$ of the monad T to the category B**–coalg** of B-coalgebras (see Section 2.4), viz., via *structural recursion* (with 'accumulators,' cf. [TP97, Theorem 5.1]). Concretely, given abstract rules $\rho : \Sigma(\mathrm{Id} \times B) \Rightarrow BT$ and a B-coalgebra $k : X \to BX$, we obtain $\tilde{T}(k)$ as follows, using the universal property of TX as being the initial $(X + \Sigma)$-algebra:

(5.5)
$$\begin{tikzcd}
X \arrow[r, "\eta_X"] \arrow[d, "k"'] & TX \arrow[d, dashed, "\tilde{T}(k)"] & & \Sigma TX \arrow[ll, "\gamma_X"'] \arrow[d, "\Sigma\langle \mathrm{id}_{TX}, \tilde{T}(k)\rangle"] \\
BX \arrow[r, "B\eta_X"'] & BTX & BT^2X \arrow[l, "B\mu_X"] & \Sigma(TX \times BTX) \arrow[l, "\rho_{TX}"]
\end{tikzcd}$$

writing η, μ_X, and γ_X for the unit and multiplication of T, and the free Σ-algebra structure on TX, respectively. The naturality of ρ is essential for the proof to go through.

Instantiating this construction with the trivial initial B-coalgebra $0 \to B0$, one obtains a B-coalgebra structure on $T0$, the set of closed terms. This precisely is the *intended operational model*, i.e., the transition system of type B induced on the programs in $T0$ by the operational rules ρ.

Assuming that $D = B^\infty$ exists, we get that D**–Coalg** $\cong$ B**–coalg**, and so the monad $\tilde{T}$ is also a lifting of T to the D-Coalgebras. Such liftings are, by [TP97, Theorem 7.1], in one-to-one correspondence with *distributive laws* $TD \Rightarrow DT$ of the monad T over the comonad D, i.e., distributing free syntax over cofree behaviour. Hence:

Theorem 5.2

Abstract operational rules of type (5.4) induce a distributive law $TD \Rightarrow DT$ of the free monad T on Σ over the cofree comonad D on B. $\square$

As it will turn out, this particular property will explain the good behaviour of the GSOS format. Additionally, it was shown in [LPW00] that already abstract rules (5.4) themselves arise canonically: natural transformations (5.4) are in one-to-one correspondence with distributive laws of T over the cofree *co-pointed* endofunctor $\mathrm{Id} \times B$ on B. Spelled out, this means that abstract rules as in (5.4) are equivalent to a natural transformation of type $T(\mathrm{Id} \times B) \Rightarrow (\mathrm{Id} \times B)T$ satisfying the evident diagrams which relate it to both the monad structure of T and the structure of the co-pointed endofunctor $\varepsilon : \mathrm{Id} \times B \Rightarrow \mathrm{Id}$, which is simply the first projection π_1. Theorem 5.2 then states that such a distributive law can be extended to one of T over the cofree comonad $D = B^\infty$ on B.

In general, a distributive law $\ell : TD \Rightarrow DT$ of the free monad T over the cofree comonad D, which need not necessarily be induced by abstract rules (5.4), can be interpreted as describing the most *general* kind of abstract rules, and lie at the core of the approach of [TP97]. We have already introduced the notion of *bialgebras* for such a distributive law ℓ in Section 2.4. Intuitively, a bialgebra corresponds to a combination of a denotational model (a T-Algebra) with an operational model (a D-Coalgebra) in such a way that (2.10) holds. This law expresses that only such combined models are allowed which behave ‘nicely’ with respect to (the rules corresponding to) ℓ.

In case that ℓ was in fact obtained from a natural transformation ρ of type (5.4), the ℓ-bialgebras can alternatively described in the following way. Recall that, since T is the free monad on Σ, we obtain an isomorphism T**–Alg** $\cong$ Σ**–alg**: thus, given a Σ-algebra $h : \Sigma X \to X$, we obtain its corresponding inductive extension $h^\star : TX \to X$. Using this, ℓ-bialgebras are equivalent to pairs $\Sigma X \xrightarrow{h} X \xrightarrow{k} BX$ such that the following simplified version of (2.10) holds:

$$\begin{tikzcd} \Sigma X \arrow[rr, "\Sigma\langle \mathrm{id}_X, k\rangle"] \arrow[d, "h"'] & & \Sigma(X \times BX) \arrow[d, "\rho_X"] \\ X \arrow[r, "k"'] & BX & BTX \arrow[l, "Bh^\star"] \end{tikzcd} \tag{5.6}$$

As already explained, the Σ-algebra structure h gives a Σ-interpretation, i.e., a denotational model, and the B-coalgebra k defines a transition system of type B, i.e., an operational model. The diagram (5.6) then expresses that ℓ-bialgebras are all those combinations of denotational and operational models which, intuitively, ‘satisfy’ the rules ρ. Hence, such bialgebras were called ρ-*models* in [TP97]. If ρ, in turn, was induced by a finite set of (concrete) GSOS rules (5.1), the ρ-models, or equivalently, the ℓ-bialgebras for the induced distributive law ℓ, are exactly the *GSOS models* of [Sim95].

Going back to the general case, using the properties of the category ℓ**–Bialg** of bialgebras of a distributive law $\ell : TD \Rightarrow DT$ allows us to derive the following *adequacy* meta results:

Theorem 5.3

1. The forgetful functor $U^{\ell} : \ell$**–Bialg** $\to D$**–Coalg**, which forgets the Algebra-part of a bialgebra, has a left adjoint, in particular, there is an initial ℓ-bialgebra, induced by the trivial initial D-Coalgebra $0 \to D0$.

2. The forgetful functor $U_{\ell} : \ell$**–Bialg** $\to T$**–Alg**, which forgets the Coalgebra-part of a bialgebra, has a right adjoint, in particular, there is a final ℓ-bialgebra, induced by the trivial final T-Algebra $T1 \to 1$.

Therefore, there is a (super-)unique homomorphism of ℓ-bialgebras from the initial to the final such bialgebra. □

The theorem can in particular be applied to the case where ℓ is induced by abstract rules ρ of type (5.4), and hence ℓ**–Bialg** is the category of ρ-models. The initial ρ-model is the intended operational model (giving the B-coalgebra structure) on the set of programs (the (initial) Σ-algebra structure). Dually, the final ρ-model can be regarded as the *canonical* denotational model [TP97] for ρ: its coalgebra-part is given by the final B-coalgebra (which is also the final D-coalgebra), i.e., the collection of abstract behaviours, and the Σ-algebra structure on it describes a compositional denotational semantics for the language.

This leads to *universal semantics*: the unique (both by initiality and finality) homomorphism from the initial to the final ρ-model. This universal semantics is then both initial algebra and final coalgebra semantics—see [TP97, Corollary 7.3]. Consequently, it is a compositional interpretation (inherited from initial algebra semantics) which also preserves behavioural distinctions (from final coalgebra semantics).

Combining B-bisimulation with the dual notion of Σ-congruence, one can define a notion of *ℓ-bicongruence* and a corresponding category of such bicongruences between any pair of ℓ-bialgebras—see [TP97] for more details. One can then ask whether there is a *final* (intuitively: a largest) such bicongruence on a given ℓ-bialgebra, and the following result is obtained:

Corollary 5.4

If B preserves weak pullbacks, then every ℓ-bialgebra has a final bicongruence. □

In particular, B_A preserves weak pullbacks (for a proof see, e.g., [Tur96]). Therefore, the above corollary specialises to the (well-known) fact that strong bisimulation, which is precisely B_A-bisimulation, is a congruence for GSOS languages.

Finally, rather than just explaining the good properties of already existing rule formats like GSOS (or, by 'dualising' (5.4), certain classes of *tree rules* [Fok94, FvG96]), the bialgebraic approach can also be used to derive *new* rule formats. For one, it is possible to interpret (5.4) in categories other than **Set** to treat, e.g., languages with *variable binding* (e.g., the π-calculus [MPW92]) and their operational semantics: see [FPT99] for a treatment of abstract syntax with binding, including initial algebra semantics for this case (see also [GP02] for an alternative approach to syntax and initial algebra semantics for languages with binding), and [FT01] for a categorical account of operational semantics for such languages, extending [TP97]; however, at present, no concrete rule format has been developed. With the same ideas in mind, however, aiming for a general treatment of recursion (currently, only *guarded* recursion can be dealt with—see [Tur97]), one might want to consider operational semantics in some categories of domains, i.e., certain classes of cpo's (cf. Section 1.2.2); preliminary results have been reported in [Plo01].

Furthermore, one can model *different kinds* of transition systems by appropriately instantiating B, e.g., one might consider (discrete) probabilistic transition systems [LS91a, vGSS95], for the appropriate choice of behaviour. This can then be used to obtain a syntactic format for languages for probabilistic processes, similar to GSOS, as was recently achieved in [Bar02] for the discrete case, by analysing the constraints expressed in (5.4) for this case (see also [dV98, dVR99] for some results in a continuous setting).

Note that a different, yet essentially equivalent, approach to a categorical treatment of (operational) semantics was developed by Corradini and others in [CGRH98, CHM99, CHM02], where so-called *structured* transition systems are modelled as coalgebras in a category of Algebras for a monad: the monad specifies some algebraic structure, and the coalgebras define transition systems as usual. Hence, in this approach, the states of transition systems are endowed with algebraic structure.

5.2 Discrete Time

We can now briefly show how to define abstract rules for timed processes over discrete time, i.e., in the case that $\mathcal{T} = \mathbb{N}$. As was shown in Theorem 4.23, the evolution comonad $E_{\mathbb{N}}$ for discrete time is cofreely generated from the behaviour functor $B_{\mathbb{N}}X = 1 + X$. Consequently, as stated in Corollary 4.30, the Coalgebras for $E_{\mathbb{N}}$, i.e., TTSs over $\mathbb{N}$, are the same as the $B_{\mathbb{N}}$-coalgebras: a $B_{\mathbb{N}}$-coalgebra $X \to 1 + X$ is simply a partial function from X to itself which describes the 'next' step; in an $E_{\mathbb{N}}$-Coalgebra, this precisely amounts to the $\overset{1}{\rightsquigarrow}$-transitions.

The upshot of all this is that, in order to define the operational semantics of timed processes over discrete time, we can simply use the previously presented framework from [TP97], instantiated with $B_{\mathbb{N}}$ as the behaviour functor. Thus, we get abstract rules of type

$$\Sigma(\mathrm{Id} \times B_{\mathbb{N}}) \Rightarrow B_{\mathbb{N}}T \tag{5.7}$$

Such abstract rules then induce a distributive law of T over $B_{\mathbb{N}}^{\infty}$, and since $B_{\mathbb{N}}^{\infty} = E_{\mathbb{N}}$, this is a distributive law $TE_{\mathbb{N}} \Rightarrow E_{\mathbb{N}}T$. Applying the adequacy results from [TP97], we in particular obtain that $B_{\mathbb{N}}$-bisimulation is a congruence for languages whose operational rules fit the type (5.7). Yet, keeping in mind that $B_{\mathbb{N}}$-bisimulation is (modulo the isomorphism of the coalgebras) the same as $E_{\mathbb{N}}$-bisimulation, as already mentioned in Remark 4.31, we obtain that $E_{\mathbb{N}}$-bisimulation, i.e., time bisimulation over the naturals, is a congruence for the language.

Concretely, for a set X of 'meta variables' as before, this means that the operational semantics of an n-ary operator $\sigma \in \Sigma$ must be given by a function

$$[\![\sigma]\!] : (X \times (1 + X))^n \to 1 + TX \tag{5.8}$$

where, as before, an n-tuple $\langle\langle x_1, \beta_1\rangle, \ldots, \langle x_n, \beta_n\rangle\rangle$ in the domain of $[\![\sigma]\!]$ describes the names and behaviours of the argument processes. In this particularly simple case, the latter boils down to either no step or the unique successor process. The codomain of $[\![\sigma]\!]$ describes the potential successor process of the composite term $\sigma\langle x_1, \ldots, x_n\rangle$.

Note that this already imposes (implicit) conditions on concrete rules which correspond to (5.8): in order for $[\![\sigma]\!]$ to be *well-defined*, there can only be at most one successor process of $\sigma\langle x_1,\dots,x_n\rangle$ for each argument tuple, i.e., the rules have to be *deterministic*. This constitutes what could be called a *behaviour-inherent*, or *behaviour-dependent* condition. Similarly, we have already seen that all sets of GSOS rules corresponding to abstract rules (5.2) must be image finite, as was shown in Theorem 5.1. This again is an example of such behaviour-inherent conditions. Another example is given by probabilistic LTSs, where the probabilities of outgoing transitions of each state must add up to 1, and consequently, this has to be ensured by the rules and a corresponding concrete rule format; we refer the reader to [Bar02] for the details.

As an application of (5.7), the time rules of ATP fit the corresponding abstract format. Note that this refers to ATP as presented in [NS94], rather than the more general ATP_D of [NSY93]: the former uses one-step (qualitative) $\overset{\chi}{\leadsto}$-transitions to denote the progress of time, whereas the latter is parameterised over a time domain D, using $\overset{d}{\leadsto}$-transitions, which can essentially be dealt with along the same lines as TeCCS over an arbitrary time domain, which will be dealt with later on.

We can also define a one-step version of TeCCS, using $\leadsto$-transitions (intuitively corresponding to the $\overset{1}{\leadsto}$-transitions of the original calculus) which fits the constraints expressed in (5.7) and then prove, exploiting the isomorphism from Corollary 4.30, that we actually define the same TTS on the set $T0$ of closed TeCCS terms.

We will, at this point, not go into the details of these two constructions since we are going to present a completely *syntactic* characterisation of (5.7) anyway in the next chapter, based on the correspondence between GSOS and abstract rules (5.4) explained in the previous section. This format will then be shown to contain both ATP and the one-step variant of TeCCS, as well as the proof that the latter actually induces the same TTS on terms as the original semantics from [MT90].

Remark 5.5

In Remark 4.32, we have seen that, for $\mathcal{T} = C^\star$, the evolution comonad is cofreely generated from the endofunctor $B_C = B_{\mathbb{N}}^C = (1+X)^C$. Hence, the same as above applies: the operational semantics for timed processes over arbitrary free monoids can be dealt with in the usual fashion of [TP97]. In this way, it should be possible to also treat the rules of the calculus PMC [AM94]. $\Diamond$

5.3 Abstract SOS Rules for Behaviour Comonads

We have seen that, for timed processes, the 'right' notion of transition systems, viz., TTSs, actually corresponds to Coalgebras for the evolution comonad E, rather than for a functor. Moreover, in general, E is not cofreely generated, as was the case for TTSs over the naturals: in particular for $\mathcal{T} = \mathbb{R}$, we do not have any hope of finding a simpler coalgebraic characterisation of the corresponding TTSs. Because of this, we cannot directly apply the theory of [TP97] to obtain well-behaved operational rules for timed processes, except for the case of discrete time or other free monoids, as was shown in the previous section.

Therefore, if we want to be able to deal with timed processes in a similar way, we have to generalise the bialgebraic approach to deal with behaviour *comonads*. Intuitively, such behaviour comonads describe *complete*, or *global* computations, rather than just the 'next step,' as is being described by behaviour functors like B_A. This can be illustrated as follows. A 'behaviour' in $B_A X$ describes the labels of transitions, and the successor states of such transitions, of a process. In contrast, consider the cofree comonad D on B_A, which exists by [Bar93], and whose value DX at a set X is, by [Tur96, §13], the set of rooted, image finite trees quotiented by strong bisimulation whose nodes are labelled in X and whose branches are labelled in A. So the elements of DX essentially describe finite or infinite series of computational steps, each of which is, co-inductively, given by a behaviour in $B_A X$ (this is actually the characteristic property of cofree comonads).

The core of the bialgebraic approach was given by using abstract rules like (5.4), or similar ones, in order to obtain a distributive law $TD \Rightarrow DT$ of (free) syntax described by the monad T over (cofree) behaviour given by the comonad D, generated by a signature Σ and a behaviour functor B, respectively. With that as a starting point, using the bialgebras of this law, the whole theory could be developed, in particular the abstract adequacy results from Theorem 5.3 and Corollary 5.4 could be obtained.

In the following, let $D = \langle D, \varepsilon, \delta \rangle$ be an arbitrary comonad on some category $\mathcal{C}$. On the level of distributive laws, it is very easy to generalise to not necessarily cofreely generated comonads: we can simply use the same kind of distributive law $TD \Rightarrow DT$ as before. Unfortunately, it would be by far too complicated to specify operational rules in this way: one would have to define the behaviour of arbitrary terms, rather than, as in the inductive approach of SOS rules, simple terms with one constructor applied to variables. Consequently, we aim for abstract rules of similar type as (5.4) which *still* induce a distributive law, yet for an arbitrary comonad D.

The fact that we want to obtain abstract rules for a general comonad, and consequently a distributive law, should make it obvious that we need to impose additional *restrictions* on abstract rules: we need to relate them to the operations of D. Otherwise, the rules might just as well induce a natural transformation of the correct type which respects the monad operations of T, e.g., if the law was induced by induction as before, yet, in general, it will *not* respect the comonad structure of D. In the original case of [TP97], these conditions were vacuously true because the comonad was cofreely generated, and so it was sufficient to use the equivalent but unconstrained B-coalgebras.

Note that this introduces a second kind of restriction imposed on concrete rules: apart from the previously mentioned behaviour-dependent ones, we now also have *proof-theoretic* ones, arising from the extra restrictions needed to relate the rules to the comonad structure of D. Intuitively, the latter type of conditions is to do with *derivability* from the rules and, in a way, with lifting the conditions imposed on the D-Coalgebras by the comonad operations to the level of *derivations* from the rules: as we shall see for the case of timed processes, the conditions needed are very similar to the axioms (ZeroDelay) and (Continuity), while (Determinacy) will turn out to be a behaviour-inherent one. This corresponds to how these axioms are reflected on the level of E-Coalgebras: while (Determinacy) is guaranteed by the type $X \to EX$, the other two essentially correspond to the conditions imposed on E-Coalgebras by the comonad operations, cf. the proof of Theorem 4.17.

In order to find an appropriate type for abstract rules, let us begin with (5.4), except that we replace the behaviour functor B with the behaviour comonad D. This results in abstract rules of the type

$$\Sigma(\mathrm{Id} \times D) \Rightarrow DT$$

Recall that, in the domain of the natural transformation, the Id-component was used to have *names* available for the component processes, not only their behaviour. However, since D is a comonad, it comes with a counit $\varepsilon : D \Rightarrow \mathrm{Id}$, which we can use to obtain a natural transformation $D \Rightarrow \mathrm{Id} \times D$: simply apply $\langle \varepsilon, \mathrm{Id}_D \rangle$ to D. In other words: using a comonad D, we *already have access* in a, moreover, *canonical* way, to names of 'argument processes' by using the counit of D (cf. calling the counit of the evolution comonad the *naming* function!). This suggests to simplify the type of abstract rules to

$$\Sigma D \Rightarrow DT$$

and in fact, we will now show that using such rules, upon imposing certain conditions, indeed allow to derive a distributive law $TD \Rightarrow DT$ of the (still freely generated) monad T over the (general) comonad D. Note that, since D is only a retract of $\mathrm{Id} \times D$, using the simplified type of natural transformations (with domain ΣD) might seem like a loss of generality; however, as will be shown later on, there is in fact no such loss of generality.

5.3.1 Comonadic SOS

This section presents a general account of abstract rules for a behaviour comonad $D = \langle D, \varepsilon, \delta \rangle$. In the following, whenever Σ is a functor for which the free monad $\Sigma^\star$ exists, we will write it as $T = \langle T, \eta, \mu \rangle$.

Definition 5.6

Let Σ, F be endofunctors on the same category $\mathcal{C}$, and assume that Σ freely generates the monad T with free Σ-algebra structure $\gamma : \Sigma T \Rightarrow T$. Given a natural transformation $\rho : \Sigma F \Rightarrow FT$, define the natural transformation $\ell : TF \Rightarrow FT$ as the unique map making the following diagram commute (obtained by the freeness of T):

$$\begin{tikzcd}
F \arrow[r, Rightarrow, "\eta_F"] \arrow[dr, Rightarrow, "F\eta"'] & TF \arrow[d, Rightarrow, dashed, "\ell"] & & \Sigma T F \arrow[ll, Rightarrow, "\gamma_F"'] \arrow[d, Rightarrow, "\Sigma\ell"] \\
& FT & FT^2 \arrow[l, Rightarrow, "F\mu"] & \Sigma F T \arrow[l, Rightarrow, "\rho_T"]
\end{tikzcd} \tag{5.9}$$

In this situation, we call ℓ the *distributive law* induced by ρ, sometimes writing it as ℓ_ρ to stress that ℓ is defined with respect to ρ. ■

The next proposition shows that the terminology 'distributive law' for ℓ, as in (5.9), is justified:

Proposition 5.7

Let Σ, F, T and ℓ be as in Definition 5.6. Then $\ell : TF \Rightarrow FT$ is a distributive law of the monad T over the endofunctor F.

Proof: We have to show that the following two diagrams commute

$$
\begin{tikzcd}
 & F \arrow[dl, Rightarrow, "\eta F"'] \arrow[dr, Rightarrow, "F\eta"] & \\
TF \arrow[rr, Rightarrow, "\ell"'] & & FT
\end{tikzcd}
\qquad
\begin{tikzcd}
T^2F \arrow[r, Rightarrow, "T\ell"] \arrow[d, Rightarrow, "\mu_F"'] & TFT \arrow[r, Rightarrow, "\ell_T"] & FT^2 \arrow[d, Rightarrow, "F\mu"] \\
TF \arrow[rr, Rightarrow, "\ell"'] & & FT
\end{tikzcd}
\tag{5.10}
$$

i.e., 'one half' of (2.9).

The triangle in (5.10) commutes by definition of ℓ. Therefore, it suffices to show that also the square commutes, for which we will again use the freeness of T: we will show that both $\ell \circ \mu_F$ and $F\mu \circ \ell_T \circ T\ell$ fit as the unique map τ in the commuting diagram

$$
\begin{tikzcd}
TF \arrow[r, Rightarrow, "\eta_{TF}"] \arrow[dr, Rightarrow, "\ell"'] & T^2F \arrow[d, Rightarrow, dashed, "\tau"] & & \Sigma T^2F \arrow[ll, Rightarrow, "\gamma_{TF}"'] \arrow[d, Rightarrow, "\Sigma\tau"] \\
 & FT & FT^2 \arrow[l, Rightarrow, "F\mu"] & \Sigma FT \arrow[l, Rightarrow, "\rho_T"]
\end{tikzcd}
$$

For $\ell \circ \mu_F$, we can fill in the diagram as follows:

$$
\begin{tikzcd}
TF \arrow[r, Rightarrow, "\eta_{TF}"] \arrow[d, Rightarrow, "\mathrm{id}_{TF}"'] & T^2F \arrow[d, Rightarrow, "\mu_F"] & & \Sigma T^2F \arrow[ll, Rightarrow, "\gamma_{TF}"'] \arrow[d, Rightarrow, "\Sigma\mu_F"] \\
TF \arrow[r, Rightarrow, "\mathrm{id}_{TF}"] \arrow[dr, Rightarrow, "\ell"'] & TF \arrow[d, Rightarrow, "\ell"] & & \Sigma TF \arrow[ll, Rightarrow, "\gamma_F"'] \arrow[d, Rightarrow, "\Sigma\ell"] \\
 & FT & FT^2 \arrow[l, Rightarrow, "F\mu"] & \Sigma FT \arrow[l, Rightarrow, "\rho_T"]
\end{tikzcd}
$$

In the top row, the left square commutes because η and μ are unit and multiplication, respectively, of the monad T, the right square commutes because μ is defined as the inductive extension of id_T (see, e.g., [Tur96]), dually to the way that δ, for a cofree comonad, is defined as the coinductive extension of the identity—see (2.8). In the bottom row, the triangle trivially commutes, and the square on the right commutes by definition of ℓ. The diagram for $F\mu \circ \ell_T \circ T\ell$ looks as follows:

$$
\begin{tikzcd}
TF \arrow[r, Rightarrow, "\eta_{TF}"] \arrow[d, Rightarrow, "\ell"'] & T^2F \arrow[d, Rightarrow, "T\ell"] & & \Sigma T^2F \arrow[ll, Rightarrow, "\gamma_{TF}"'] \arrow[d, Rightarrow, "\Sigma T\ell"] \\
FT \arrow[r, Rightarrow, "\eta_{FT}"] \arrow[d, Rightarrow, "\mathrm{id}_{FT}"'] & TFT \arrow[d, Rightarrow, "\ell_T"] & & \Sigma TFT \arrow[ll, Rightarrow, "\gamma_{FT}"'] \arrow[d, Rightarrow, "\Sigma\ell_T"] \\
FT \arrow[r, Rightarrow, "F\eta_T"'] \arrow[d, Rightarrow, "\mathrm{id}_{FT}"'] & FT^2 \arrow[d, Rightarrow, "F\mu"] & FT^3 \arrow[l, Rightarrow, "F\mu_T"'] \arrow[d, Rightarrow, "FT\mu"] & \Sigma FT^2 \arrow[l, Rightarrow, "\rho_{T^2}"'] \arrow[d, Rightarrow, "\Sigma F\mu"] \\
FT \arrow[r, Rightarrow, "\mathrm{id}_{FT}"'] & FT & FT^2 \arrow[l, Rightarrow, "F\mu"] & \Sigma FT \arrow[l, Rightarrow, "\rho_T"]
\end{tikzcd}
$$

In the top row, the left square commutes because η is natural, the right square because of the naturality of γ. The middle row commutes because of the definition of ℓ (precomposed with T). Finally, in the bottom row, both the left and the middle square commute because of the monad laws, and the right square commutes because ρ is natural. Thus we have shown that $\ell \circ \mu_F = F\mu \circ \ell_T \circ T\ell$, making ℓ into a distributive law of the monad T over the endofunctor F. $\square$

Using $\ell = \ell_\rho$, it is now possible to formulate conditions under which abstract rules as in (5.12) are 'well-behaved' with respect to D:

Definition 5.8

Let Σ be a functor with freely generated monad T, let $D = \langle D, \varepsilon, \delta \rangle$ be a comonad. Let $\rho : \Sigma D \Rightarrow DT$ be a natural transformation with induced distributive law $\ell : TD \Rightarrow DT$ of T over D (viewed as an endofunctor). Then say that ρ *respects the structure of the comonad D* if the following two diagrams, referred to as the ε- and δ-diagram, respectively, commute:

(5.11)
$$
\begin{tikzcd}
\Sigma D \arrow[r, Rightarrow, "\rho"] \arrow[d, Rightarrow, "\Sigma\varepsilon"'] & DT \arrow[d, Rightarrow, "\varepsilon_T"] \\
\Sigma \arrow[r, Rightarrow, "\xi"'] & T
\end{tikzcd}
\qquad
\begin{tikzcd}
\Sigma D \arrow[rr, Rightarrow, "\rho"] \arrow[d, Rightarrow, "\Sigma\delta"'] & & DT \arrow[d, Rightarrow, "\delta_T"] \\
\Sigma D^2 \arrow[r, Rightarrow, "\rho_D"'] & DTD \arrow[r, Rightarrow, "D\ell"'] & D^2T
\end{tikzcd}
$$

where $\xi : \Sigma \Rightarrow T$ is defined as $\xi \stackrel{\mathrm{df}}{=} \gamma \circ \Sigma\eta$. *Abstract comonadic SOS (CSOS) rules for D* are then given by a natural transformation

(5.12)
$$\rho : \Sigma D \Rightarrow DT$$

which respects the structure of D in the sense of (5.11). ■

Our aim is now to show that abstract CSOS rules indeed induce a distributive law $TD \Rightarrow DT$ of the monad over the comonad. We first provide two technical lemmas concerning ξ which will be needed later on.

Lemma 5.9

We have

$$\gamma = \mu \circ \xi_T : \Sigma T \Rightarrow T \tag{5.13}$$

Proof: Consider the following diagram:

$$\begin{tikzcd}
\Sigma T \arrow[rr, Rightarrow, "\gamma"] \arrow[d, Rightarrow, "\Sigma\eta_T"'] \arrow[dr, Rightarrow, "\mathrm{id}_{\Sigma T}"] & & T \arrow[dd, Rightarrow, "\mathrm{id}_T"] \\
\Sigma T^2 \arrow[r, Rightarrow, "\Sigma\mu"'] \arrow[d, Rightarrow, "\gamma_T"'] & \Sigma T \arrow[ur, Rightarrow, "\gamma"] \arrow[dr, Rightarrow, "\gamma"] & \\
T^2 \arrow[rr, Rightarrow, "\mu"'] & & T
\end{tikzcd}$$

The diagram commutes because of the monad law $\mu \circ \eta_T = \mathrm{id}_T$, the definition of ξ (the composite map on the left-hand side, $\gamma_T \circ \Sigma\eta_T$, is equal to ξ_T), and the definition of μ as an inductive extension. $\square$

Lemma 5.10

The following diagram commutes:

$$\begin{tikzcd}
\Sigma \arrow[r, Rightarrow, "\Sigma\eta"] \arrow[dr, Rightarrow, "\xi"'] & \Sigma T \arrow[r, Rightarrow, "\xi_T"] & T^2 \arrow[dl, Rightarrow, "\mu"] \\
& T &
\end{tikzcd} \tag{5.14}$$

Proof: To show the commutativity of (5.14), we fill in the diagram as follows:

$$\begin{tikzcd}
\Sigma \arrow[r, Rightarrow, "\Sigma\eta"] \arrow[dd, Rightarrow, "\Sigma\eta"'] \arrow[ddrrr, Rightarrow, "\xi"'] & \Sigma T \arrow[r, Rightarrow, "\Sigma\eta_T"] \arrow[dr, Rightarrow, "\mathrm{id}_{\Sigma T}"] & \Sigma T^2 \arrow[r, Rightarrow, "\gamma_T"] \arrow[d, Rightarrow, "\Sigma\mu"] & T^2 \arrow[dd, Rightarrow, "\mu"] \\
& & \Sigma T \arrow[dr, Rightarrow, "\gamma"] & \\
\Sigma T \arrow[rrr, Rightarrow, "\gamma"'] & & & T
\end{tikzcd}$$

This diagram commutes because of the monad laws, the definition of ξ, and the definition of μ as an inductive extension of γ along id_T. $\square$

Abstract rules of type $\rho : \Sigma D \Rightarrow DT$, regardless whether they do or do not respect the structure of the comonad D, induce a distributive law $\ell = \ell_\rho$ of the monad T over (the endofunctor) D, as was shown in Proposition 5.7. If ρ are actually CSOS rules, i.e., additionally respect the structure of D, then ℓ also respects the structure of D, and so we obtain:

Theorem 5.11
Let $\rho : \Sigma D \Rightarrow DT$ be a natural transformation with induced distributive law $\ell = \ell_\rho$. If, additionally, ρ respects the structure of the comonad $D = \langle D, \varepsilon, \delta \rangle$, then $\ell : TD \Rightarrow DT$ is a distributive law of the monad T over the comonad D.

Proof: From Proposition 5.7, we already know that ℓ is a distributive law of the monad T over the *functor* D, so it remains to show that the following two diagrams commute, relating ℓ to the operations ε and δ of the comonad D, i.e., the 'other half' of (2.9):

(5.15)
$$\begin{tikzcd} TD \arrow[rr, Rightarrow, "\ell"] \arrow[dr, Rightarrow, "T\varepsilon"'] & & DT \arrow[dl, Rightarrow, "\varepsilon_T"] \\ & T & \end{tikzcd} \qquad \begin{tikzcd} TD \arrow[rr, Rightarrow, "\ell"] \arrow[d, Rightarrow, "T\delta"'] & & DT \arrow[d, Rightarrow, "\delta_T"] \\ TD^2 \arrow[r, Rightarrow, "\ell_D"'] & DTD \arrow[r, Rightarrow, "D\ell"'] & D^2T \end{tikzcd}$$

For the triangle, we show that both $T\varepsilon$ and $\varepsilon_T \circ \ell$ fit as the unique inductive extension τ of γ along $\varepsilon_T \circ D\eta$; diagrammatically this means

(5.16)
$$\begin{tikzcd} D \arrow[r, Rightarrow, "\eta_D"] \arrow[d, Rightarrow, "D\eta"'] & TD \arrow[d, Rightarrow, dashed, "\tau"] & \Sigma TD \arrow[l, Rightarrow, "\gamma_D"'] \arrow[d, Rightarrow, "\Sigma\tau"] \\ DT \arrow[r, Rightarrow, "\varepsilon_T"'] & T & \Sigma T \arrow[l, Rightarrow, "\gamma"] \end{tikzcd}$$

Once we manage to show that, we know that they must be equal. First, we fill in the diagram for $\varepsilon_T \circ \ell$.

$$\begin{tikzcd} D \arrow[r, Rightarrow, "\eta_D"] \arrow[dr, Rightarrow, "D\eta"] \arrow[dd, Rightarrow, "D\eta"'] & TD \arrow[d, Rightarrow, "\ell"] & & \Sigma TD \arrow[ll, Rightarrow, "\gamma_D"'] \arrow[d, Rightarrow, "\Sigma\ell"] \\ & DT \arrow[d, Rightarrow, "\varepsilon_T"] & DT^2 \arrow[l, Rightarrow, "D\mu"] \arrow[d, Rightarrow, "\varepsilon_{T^2}"] & \Sigma DT \arrow[l, Rightarrow, "\rho_T"] \arrow[d, Rightarrow, "\Sigma\varepsilon_T"] \\ DT \arrow[r, Rightarrow, "\varepsilon_T"'] & T & T^2 \arrow[l, Rightarrow, "\mu"] & \Sigma T \arrow[l, Rightarrow, "\xi_T"] \arrow[ll, Rightarrow, bend left, "\gamma"] \end{tikzcd}$$

In this diagram, the upper half commutes by definition of ℓ. In the lower half, the right-hand square commutes because of the ε-diagram in (5.11), the left-hand square commutes trivially, and the middle square commutes because of naturality of ε. Note that $\gamma = \mu \circ \xi_T$ by Lemma 5.9, hence the map at the bottom from ΣT to T is equal to γ. Now we consider the same diagram for the other map, $T\varepsilon$:

$$
\begin{tikzcd}
D \arrow[rr, Rightarrow, "\eta_D"] \arrow[dr, Rightarrow, "\varepsilon"] \arrow[dd, Rightarrow, "D\eta"'] & & TD \arrow[dd, Rightarrow, "T\varepsilon"] & \Sigma TD \arrow[l, Rightarrow, "\gamma_D"'] \arrow[dd, Rightarrow, "\Sigma T\varepsilon"] \\
& \mathrm{Id} \arrow[dr, Rightarrow, "\eta"] & & \\
DT \arrow[rr, Rightarrow, "\varepsilon_T"'] & & T & \Sigma T \arrow[l, Rightarrow, "\gamma"]
\end{tikzcd}
$$

In this diagram, the right half commutes because of naturality of γ. The two squares in the left half commute because of naturality of η and ε, respectively. Hence, the triangle in (5.15) commutes.

We now have to show that ℓ also respects the comultiplication δ, i.e., that the rectangle in (5.15) commutes. We again show that both maps, $\delta \circ \ell_T$ and $D_\ell \circ \ell_D \circ T\delta$, make the same universal diagram commute, hence they must be equal. For the first map, the diagram looks as follows:

$$
\begin{tikzcd}
D \arrow[r, Rightarrow, "\eta_D"] \arrow[dr, Rightarrow, "D\eta"'] \arrow[dd, Rightarrow, "D\eta"'] & TD \arrow[d, Rightarrow, "\ell"] & & & \Sigma TD \arrow[llll, Rightarrow, "\gamma_D"'] \arrow[d, Rightarrow, "\Sigma\ell"] \\
& DT \arrow[d, Rightarrow, "\delta_T"] & DT^2 \arrow[l, Rightarrow, "D\mu"] \arrow[d, Rightarrow, "\delta_{T^2}"] & & \Sigma DT \arrow[ll, Rightarrow, "\rho_T"] \arrow[d, Rightarrow, "\Sigma\delta_T"] \\
DT \arrow[r, Rightarrow, "\delta_T"'] & D^2T & D^2T^2 \arrow[l, Rightarrow, "D^2\mu"] & DTDT \arrow[l, Rightarrow, "D\ell_T"] & \Sigma D^2T \arrow[l, Rightarrow, "\rho_{DT}"]
\end{tikzcd}
$$

In this diagram, the upper half commutes by definition of ℓ; the square below the triangle commutes trivially; the rightmost square in the lower half commutes because of the δ-diagram in (5.11), and the square to the left of it commutes because δ is natural.

For the other map, we fill diagram in as follows:

$$
\begin{tikzcd}
D \arrow[r, equal] \arrow[ddd, Rightarrow, "D\eta"'] & D \arrow[r, Rightarrow, "\eta_D"] \arrow[d, Rightarrow, "\delta"] & TD \arrow[d, Rightarrow, "T\delta"] & & & \Sigma TD \arrow[llll, Rightarrow, "\gamma_D"'] \arrow[d, Rightarrow, "\Sigma T\delta"] \\
& D^2 \arrow[r, Rightarrow, "\eta_{D^2}"] \arrow[dr, Rightarrow, "D\eta_D"] \arrow[ddr, Rightarrow, "D^2\eta"'] & TD^2 \arrow[d, Rightarrow, "\ell_D"] & & & \Sigma TD^2 \arrow[llll, Rightarrow, "\gamma_{D^2}"'] \arrow[d, Rightarrow, "\Sigma\ell_D"] \\
& & DTD \arrow[d, Rightarrow, "D\ell"] & & DT^2D \arrow[ll, Rightarrow, "D\mu_D"] \arrow[d, Rightarrow, "DT\ell"] & \Sigma DTD \arrow[l, Rightarrow, "\rho_{TD}"'] \arrow[d, Rightarrow, "\Sigma D\ell"] \\
DT \arrow[r, equal] & DT \arrow[r, Rightarrow, "\delta_T"'] & D^2T & D^2T^2 \arrow[l, Rightarrow, "D^2\mu"] & DTDT \arrow[l, Rightarrow, "D\ell_T"] & \Sigma D^2T \arrow[l, Rightarrow, "\rho_{DT}"]
\end{tikzcd}
$$

Here, the square given by the bent arrow on the left composed with δ_T, and $D^2\eta \circ \delta$ commutes because of naturality of δ; the first row commutes because of naturality of η and γ; the second row commutes because of the definition of ℓ; the triangle with sides $D^2\eta$, $D\ell$, and $D\eta_D$ commutes because ℓ is a distributive law respecting the operations of the monad T; for the same reason, the square in the middle of the bottom row commutes; finally, the right square in the bottom row commutes because ρ is natural. Hence we have shown that ℓ is a distributive law of T over D. □

Theorem 5.11 states that abstract CSOS rules induce a distributive law of free syntax over the (arbitrary) behaviour comonad D. Using the bialgebras of the induced distributive law ℓ, given CSOS rules ρ, the same results follow as in [TP97], in particular:

Corollary 5.12

If D preserves weak pullbacks, then any ℓ-bialgebra has a final bicongruence. □

Further specialising this to the evolution comonad E, which, by Proposition 4.14, preserves pullbacks and so in particular weak pullbacks (Corollary 4.15), we obtain:

Corollary 5.13

If the operational rules of a language for timed processes induce CSOS rules for E, time bisimulation is a congruence for the language. □

In the following, the generality of abstract CSOS rules is made even more precise: such rules are in fact already *equivalent* to distributive laws of a free monad over a comonad, providing the converse of Theorem 5.11. First an auxiliary result, which is very similar to results in [LPW00, Wat02]:

Proposition 5.14
Let Σ, F be endofunctors, and let T be the free monad on Σ. Then natural transformations of type $\rho : \Sigma F \Rightarrow FT$ are in one-to-one correspondence with distributive laws $\ell : TF \Rightarrow FT$ of the monad T over the endofunctor F.

Proof: We first describe the correspondence. Given ρ, define $\ell : TF \Rightarrow FT$ as the unique map making (5.9) commute, and Proposition 5.7 then shows that ℓ respects the operations of the monad T. In the converse direction, given $\ell : TF \Rightarrow FT$ respecting the monad structure, we define $\rho_\ell = \ell \circ \xi_F : \Sigma F \Rightarrow FT$.

We now have to show that these two assignments $\rho \mapsto \ell$ and $\ell \mapsto \rho_\ell$ are mutually inverse. Hence we have to show that $\ell \circ \xi_F = \rho$ and $\ell_{\ell \circ \xi_F} = \ell$. For the first equation, consider the following diagram:

(5.17)
$$
\begin{tikzcd}
 & \Sigma F \arrow[rr, equal] \arrow[d, Rightarrow, "\xi_F"'] & & \Sigma F \arrow[d, Rightarrow, "\Sigma\eta_F"] \\
F \arrow[r, Rightarrow, "\eta_F"] \arrow[dr, Rightarrow, "F\eta"'] & TF \arrow[r, Rightarrow, shift right, "T\eta_F"'] \arrow[d, Rightarrow, "\ell"] & T^2F \arrow[l, Rightarrow, shift right, "\mu_F"'] & \Sigma TF \arrow[l, Rightarrow, "\xi_{TF}"] \arrow[d, Rightarrow, "\Sigma\ell"] \\
 & FT & FT^2 \arrow[l, Rightarrow, "F\mu"] & \Sigma FT \arrow[l, Rightarrow, "\rho_T"]
\end{tikzcd}
$$

In (5.17), the lower half commutes by definition of ℓ (using Lemma 5.9), and the composite on the right-hand edge, $\Sigma\ell \circ \sigma\eta_F$ is, by definition of ℓ, equal to $\Sigma F\eta$; the upper half commutes by applying Lemma 5.10, and by naturality of ξ. Furthermore, we obtain $\rho_T \circ \Sigma F\eta = FT\eta \circ \rho$, by naturality of ρ, and thus, using the monad laws,

$$F\mu \circ \rho_T \circ \Sigma\ell \circ \sigma\eta_F = F\mu \circ \rho_T \circ \Sigma F\eta = F\mu \circ FT\eta \circ \rho = \rho$$

which shows $\ell \circ \xi_F = \rho$. So we have shown that the mapping $\rho \mapsto \ell = \ell_\rho \mapsto \rho_\ell$ is the identity, establishing one half of the correspondence.

In the other direction, starting with a distributive law $\ell : TF \Rightarrow FT$, we have to show that we also obtain the identity mapping via $\ell \mapsto \rho_\ell \mapsto \ell_{\rho_\ell}$. To this end, since the distributive law ℓ_{ρ_ℓ} induced by the rules ρ_ℓ, which, in turn, are induced by ℓ, is defined using a universal property, we show that ℓ makes the defining diagram of ℓ_{ρ_ℓ} commute. The defining diagram is given as follows:

$$\begin{tikzcd} F \arrow[r, Rightarrow, "\eta_F"] \arrow[dr, Rightarrow, "F\eta"'] & TF \arrow[d, Rightarrow, "\ell_{\rho_\ell}"] & & \Sigma TF \arrow[ll, Rightarrow, "\gamma_F"'] \arrow[d, Rightarrow, "\Sigma\ell_{\rho_\ell}"] \\ & FT & FT^2 \arrow[l, Rightarrow, "F\mu"] & \Sigma FT \arrow[l, Rightarrow, "(\rho_\ell)_T"] \end{tikzcd}$$

We will now show that ℓ fits in place of ℓ_{ρ_ℓ} in the above diagram:

$$\begin{tikzcd} F \arrow[r, Rightarrow, "\eta_F"] \arrow[dr, Rightarrow, "F\eta"'] & TF \arrow[d, Rightarrow, "\ell"] & & T^2F \arrow[ll, Rightarrow, "\mu_F"'] \arrow[d, Rightarrow, "T\ell"] & \Sigma TF \arrow[l, Rightarrow, "\xi_{TF}"'] \arrow[d, Rightarrow, "\Sigma\ell"] \\ & FT & FT^2 \arrow[l, Rightarrow, "F\mu"] & TFT \arrow[l, Rightarrow, "\ell_T"] & \Sigma FT \arrow[l, Rightarrow, "\xi_{FT}"] \end{tikzcd}$$

This diagram commutes because ℓ, by assumption, respects η (for the triangle on the left), respects μ (for the middle square), and because ξ is natural (for the square on the right). Hence, bearing in mind that $\rho_\ell = \ell \circ \xi_F$ and $\mu \circ \xi_T = \gamma$ by Lemma 5.9, ℓ makes the same diagram commute as ℓ_{ρ_ℓ}, and so $\ell = \ell_{\rho_\ell}$, as desired, proving that the mapping $\ell \mapsto \rho_\ell \mapsto \ell_{\rho_\ell}$ is also the identity, concluding the proof. □

Substituting D for F, and adding the requirement that ρ respects the structure of D, the one-to-one correspondence from Proposition 5.14 extends to distributing free monads over comonads:

Theorem 5.15

Let Σ be a functor with freely generated monad T, and let D be a comonad. Then there is a one-to-one correspondence between abstract CSOS rules $\rho : \Sigma D \Rightarrow DT$ and distributive laws $\ell : TD \Rightarrow DT$ of the freely generated monad T over the comonad D.

Proof: We have already seen in Proposition 5.14 that abstract rules ρ are in one-to-one correspondence with distributive laws of the monad T over the endofunctor D. If we show that it is equivalent for ℓ to respect the comonad operations and to satisfy (5.11), we are done. This boils down to proving that the correspondence given in Proposition 5.14 preserves the comonad operations, under the given assumptions.

We have already shown in Theorem 5.11 that for abstract rules ρ, together with the two conditions (5.11), the induced distributive law $\ell : TD \Rightarrow DT$ respects the operations of comonad D; this is also the same law as used in the one-to-one correspondence. So the only thing left to prove is that the abstract rules obtained from a distributive law of the monad over the comonad respect the structure of the comonad.

So let $\ell : TD \Rightarrow DT$ be a distributive law of the monad T over the comonad D. We want to show that for the induced rules $\rho = \rho_\ell : \Sigma D \Rightarrow DT$, defined as

$$\rho = \Sigma D \overset{\xi_D}{\Longrightarrow} TD \overset{\ell}{\Longrightarrow} DT \quad ,$$

the two diagrams (5.11) commute. For the ε-diagram, consider the following diagram:

$$\begin{tikzcd}
\Sigma D \arrow[r, Rightarrow, "\xi_D"'] \arrow[rr, Rightarrow, bend left, "\rho"] \arrow[d, Rightarrow, "\Sigma\varepsilon"'] & TD \arrow[r, Rightarrow, "\ell"'] \arrow[d, Rightarrow, "T\varepsilon"] & DT \arrow[d, Rightarrow, "\varepsilon_T"] \\
\Sigma \arrow[r, Rightarrow, "\xi"'] & T \arrow[r, equal] & T
\end{tikzcd}$$

The above diagram commutes: the left square commutes because of naturality of ξ, the right square commutes because ℓ respects ε. For the δ-diagram, we consider a very similar calculation:

$$\begin{tikzcd}
\Sigma D \arrow[rrr, Rightarrow, "\rho"] \arrow[d, equal] & & & DT \arrow[d, equal] \\
\Sigma D \arrow[r, Rightarrow, "\xi_D"'] \arrow[d, Rightarrow, "\Sigma\delta"'] & TD \arrow[rr, Rightarrow, "\ell"'] \arrow[d, Rightarrow, "T\delta"] & & DT \arrow[d, Rightarrow, "\delta_T"] \\
\Sigma D^2 \arrow[r, Rightarrow, "\xi_{D^2}"] \arrow[d, equal] & TD^2 \arrow[r, Rightarrow, "\ell_D"] & DTD \arrow[r, Rightarrow, "D\ell"'] \arrow[d, equal] & D^2T \\
\Sigma D^2 \arrow[rr, Rightarrow, "(\ell\circ\xi_D)_D=\rho_D"'] & & DTD &
\end{tikzcd}$$

This diagram commutes as well: the left square in the middle row again by naturality of ξ; the rectangle to its right because ℓ respects δ; the topmost square commutes by definition of ρ; finally, $\rho_D = (\ell \circ \xi_D)_D = \ell_D \circ \xi_{D^2}$ because D and (pre-)composition with D are functors, therefore the square in the bottom row commutes as well. □

Intuitively, Theorem 5.15 states that, in the case of T being freely generated, (5.11) provides precisely the necessary and sufficient conditions on abstract rules of type $\Sigma D \Rightarrow DT$ such that the bialgebraic approach of [TP97] can be applied; in other words:

there is no way to use strictly more expressive abstract rules and still obtain a distributive law $TD \Rightarrow DT$ (for T being freely generated).

Note that this implies that the main result of [TP97], viz., that abstract operational rules (5.4) induce a distributive law of free syntax over cofree behaviour, is also included in Theorem 5.11. Assume that $D = B^\infty$ is cofreely generated from a behaviour functor B, e.g., consider timed processes over discrete time, in which case the evolution comonad is cofreely generated from $B_\mathbb{N}$. Then, any natural transformation ρ as in (5.4) gives rise to CSOS rules for D, i.e., a natural transformation of type $\Sigma D \Rightarrow DT$ which respects the structure of D: as was shown in [TP97], the rules ρ induce a distributive law $TD \Rightarrow DT$ of the monad over the comonad D; yet, in Theorem 5.15, we have shown that such distributive laws are in one-to-one correspondence with CSOS rules for D, so in particular there are CSOS rules which correspond to the abstract GSOS rules ρ. We are currently not aware of a more direct proof of this fact which does not use the correspondence from Theorem 5.15, although intuitively, the claim is obvious.

5.3.2 Abstract Temporal Rules

Consider the special case of timed processes, with the evolution comonad as the appropriate behaviour comonad. We could now use CSOS rules for E to describe the operational semantics of timed processes. Yet, as inspection of the literature on timed process calculi shows, this amount of generality is not even needed: operational rules corresponding to (5.12) could use arbitrary terms as targets of rule conclusions, while the rules for TeCCS, and all other calculi we are aware of, only use targets which contain at most one constructor, i.e., targets of rule conclusions are either a single variable or a composite term of the form $\sigma\langle x_1, \ldots, x_n\rangle$ where σ is an n-ary function symbol in Σ, and the x_i are variables.

To reflect this, we are going to show in this section that we can use a less general kind of natural transformation, together with correspondingly simpler conditions relating it to the comonad structure of a general comonad D, which will still induce a distributive law $TD \Rightarrow DT$; hence, we will still be able to apply the bialgebraic approach of [TP97]. Note that the present development is motivated by, and tailored towards, languages for timed processes, so when reading D for the comonad, one should really think of E, the evolution comonad.

One advantage of this simpler approach is that the conditions imposed on the operational rules, in order to ensure that we obtain a distributive law respecting the operations of D, will become easier. This will certainly become clear in the next chapter when we will derive syntactic characterisations of abstract CSOS rules. However, intuitively, this should already become clear here: in the light of Theorem 5.15, which states that CSOS rules are essentially the *same* as a distributive law, using rules which merely imply the existence of such a law, but not the converse, must necessarily be more restrictive. In order to force such consequently less expressive rules to respect the structure of D, it should be obvious that weaker conditions will suffice.

In the following, let $\mathcal{C}$ be an appropriate category with all the required structures (products, coproducts, distributivity etc.).

Definition 5.16

Let Σ be an endofunctor on $\mathcal{C}$ which freely generates a monad T, and let D be a comonad on $\mathcal{C}$. Moreover, let

$$\rho : \Sigma E \Rightarrow E(\mathrm{Id} + \Sigma) \tag{5.18}$$

be a natural transformation. We then call ρ *abstract temporal rules*, or say that ρ *respects the structure* of D, if ρ makes the following two diagrams commute:

$$\begin{tikzcd} \Sigma D \arrow[r, Rightarrow, "\rho"] \arrow[d, Rightarrow, "\Sigma\varepsilon"'] & D(\mathrm{Id}+\Sigma) \arrow[d, Rightarrow, "\varepsilon_{\mathrm{Id}+\Sigma}"] \\ \Sigma \arrow[r, Rightarrow, "\mathrm{inr}"'] & \mathrm{Id}+\Sigma \end{tikzcd} \qquad \begin{tikzcd} \Sigma D \arrow[rr, Rightarrow, "\rho"] \arrow[d, Rightarrow, "\Sigma\delta"'] & & D(\mathrm{Id}+\Sigma) \arrow[d, Rightarrow, "\delta_{\mathrm{Id}+\Sigma}"] \\ \Sigma D^2 \arrow[r, Rightarrow, "\rho_D"'] & D(D+\Sigma D) \arrow[r, Rightarrow, "D(D\mathrm{inl}+\rho)"'] & D^2(\mathrm{Id}+\Sigma) \end{tikzcd} \tag{5.19}$$

The two maps inl and inr in the diagrams denote the left and right injections, respectively, into the coproduct. We will again refer to the two diagrams in (5.19) as the ε- and the δ-diagram, respectively, since, due to the type of ρ, no confusion can arise with the corresponding diagrams for CSOS rules. ■

Given abstract temporal rules ρ, we can again show that they induce a distributive law:

Proposition 5.17

Let Σ, F be endofunctors on **Set** such that Σ freely generates the monad T, and let $\rho : \Sigma F \Rightarrow F(\mathrm{Id} + \Sigma)$ be a natural transformation. Then ρ induces a distributive law $\ell : TF \Rightarrow FT$ of the monad T over the endofunctor F.

Proof: Let ρ be a natural transformation of the given type. Then define ℓ as the unique map making the following diagram commute:

$$\begin{tikzcd}
F \arrow[r, Rightarrow, "\eta_F"] \arrow[dr, Rightarrow, "F\eta"'] & TF \arrow[d, Rightarrow, dashed, "\ell"] & & \Sigma TF \arrow[ll, Rightarrow, "\gamma_F"'] \arrow[d, Rightarrow, "\Sigma\ell"] \\
& FT & F(T+\Sigma T) \arrow[l, Rightarrow, "{F[\mathrm{id}_T,\gamma]}"] & \Sigma FT \arrow[l, Rightarrow, "\rho_T"]
\end{tikzcd} \tag{5.20}$$

Using this definition, calculations very similar to Proposition 5.7 show that ℓ indeed respects the monad structure. $\square$

Like in the situation for CSOS rules, we now know that from the natural transformation alone, we get a distributive law ℓ that respects the monad structure of T. Further, if ρ respects D, ℓ also respects the comonad structure, and so we obtain our desired distributive law:

Theorem 5.18

Let Σ be a functor freely generating a monad T. Furthermore, suppose ρ is a natural transformation of type (5.18) respecting the structure of D. Then ρ induces a distributive law $\ell : TD \Rightarrow DT$ of the monad T over the comonad D.

Proof: We have already seen, in Proposition 5.17, that we get a distributive law ℓ of the monad T over the endofunctor D. Under the additional assumptions, similarly to Theorem 5.11, we can further deduce that ℓ also respects the comonad operations. In fact, in order to prove that ℓ respects the counit ε of D, we can use exactly the same diagram as before, viz., again showing that both $\varepsilon_T \circ \ell$ and $T\varepsilon$ fit as the unique map τ making (5.16) commute: in fact, the proof for $T\varepsilon$ stays exactly the same; for the other map, we merely apply the modified ε-diagram instead of the previous ε-diagram. The same holds for showing that ℓ respects the comultiplication δ: the same diagram can be used, the ‘new’ δ-diagram is applied in place of the previous one, but the calculations essentially stay the same. $\square$

So we gain another, weaker congruence result for $D = E$:

Corollary 5.19
Time bisimulation is a congruence for any language whose operational rules induce abstract temporal rules. □

In the next chapter, we are going to show that in particular the time rules of TeCCS induce abstract temporal rules. Therefore, we obtain:

Corollary 5.20
Time bisimulation is a congruence for TeCCS. □

Note that this is a weaker congruence result than what was shown in [MT90]: there, the bisimulation obtained by *combining* action and time bisimulation was a congruence; here, we only obtain that each of the two on its own is a congruence: for the time-part, this follows from the preceding corollary, and for the action rules, since they are GSOS, from [TP97]. We will, in Chapter 7, show how to combine action and time transitions in such a way that the categorical framework automatically establishes the stronger congruence result.

Remark 5.21
As already hinted at in [Kic02a], the two kinds of abstract rules introduced so far, viz., abstract CSOS rules (5.12) and abstract temporal rules (5.18), with their corresponding conditions, only form the extreme points of suitable types of natural transformations to model the operational semantics of a language: abstract temporal rules only allow either a variable or one single constructor (applied to variables) as targets of conclusions, while CSOS allows arbitrary terms with any nesting of constructors, just like GSOS—cf. the use of rules like (5.1). In between, there is a whole hierarchy of increasingly expressive types of natural transformations.

This higher expressivity is traded off against the fact that the conditions to be satisfied by the rules (in order to respect the comonad structure) also become harder. For example, in the next chapter, we shall see that the δ-diagram in (5.19) directly corresponds to a derivation-based version of axiom (Continuity), i.e., being able to *derive* a $\overset{t+u}{\rightsquigarrow}$-transition, applying the rules *once*, must be equivalent to being able to derive consecutive $\overset{t}{\leadsto}$- and $\overset{u}{\leadsto}$-transitions, again by a *single* application of the rules, with the same resulting process; for CSOS rules, this has to be generalised to allow derivations of the $\overset{u}{\leadsto}$-transition by an *arbitrary* number of rule applications, and so, the 'coherence conditions' imposed on the rules become very difficult indeed to check. $\Diamond$

Chapter 6

Rule Formats for Timed Processes

In this chapter, based on the previously developed framework of abstract rules for behaviour comonads and hence also for timed processes, we are going to present *syntactic* formats for operational rules for timed processes. All of these are derived systematically from the corresponding abstract rules. Note that all of these formats still deal with the specific case that the time rules are independent from the action rules, i.e., that there are two disjoint sets of rules for the two kinds of transitions.

Since we presented three different kinds of abstract rules, there are also three rule formats. The first, which we call the *dsl-format* (abbreviating deterministic single-label), deals with the case of discrete time, where the evolution comonad is cofreely generated from a simple behaviour functor. Hence, similar techniques as in the treatment of GSOS in [TP97] can be applied; as a consequence, the format will be based on specialised GSOS rules. The dsl-format is general enough to include all the rules we managed to find in the literature on process algebras with discrete time.

The two other formats both deal with the case of an arbitrary time domain, and hence the case where the evolution comonad is no longer cofreely generated. The first, and simpler, format is based on *schematic* rules which, by using time variables, allows to *uniformly* derive time transitions of process expressions over a signature. There are several *shapes* of schematic rules, and only specific combinations of those shapes are allowed to form *admissible operators*; effectively, the schematic 'format' only specifies a bunch of timing behaviours, and as such is not quite on a par with 'proper' rule formats like GSOS or tyft/tyxt. It should therefore not come as a surprise that admissible operators are *sound*, in the sense of inducing abstract temporal rules and thus allowing to infer the corresponding congruence results, yet are not *complete*: there are (in fact very simple) concrete rules which cannot be described by admissible operators while still inducing abstract temporal rules. Despite all their shortcomings, admissible operators are still expressive enough to capture all 'well-behaved' operators we found in the literature.

Finally, we present a complete characterisation of CSOS rules for timed processes. This format is based on special *meta rules* which are somewhat in between syntax and semantics, providing a finitary notation for behaviours of an inherently infinitary flavour. Despite this first level of infinity, viz., using certain 'abbreviations' for infinite sets of time rules, the format still needs to use infinitely many such meta rules, otherwise no complete characterisation could be obtained. Rather than providing a real basis for specification languages for timed processes, this format, regarded realistically, serves as a clear demonstration of the intricacies involved in 'pinning down' syntactically the behaviour of timed processes over an arbitrary time domain. Consequently, it goes to show that there is still so much we do not really understand, in particular how to cut down, let alone remove completely, the various 'levels' of infinity involved.

6.1 A Format for Discrete Time

Let us now consider the special case $\mathcal{T} = \mathbb{N}$. As shown in Theorem 4.23, $E_{\mathbb{N}}$ is cofreely generated by the functor $B_{\mathbb{N}} = 1 + \mathrm{Id}$, and consequently, $E_{\mathbb{N}}$**–Coalg** $\cong B_{\mathbb{N}}$**–coalg**, i.e., TTSs over the naturals are the same as $B_{\mathbb{N}}$-coalgebras. Any of the latter is given by a function $k : X \to 1 + X$ which induces a transition system (without labels or, equivalently, with a single label) by defining

$$x \rightsquigarrow x' \stackrel{\mathrm{df}}{\Leftrightarrow} k(x) = \mathrm{inr}(x') \tag{6.1}$$

with inr again denoting the (right) injection into the coproduct $1 + X$; note that we will usually suppress mentioning either injection, writing $\star$ for the single element of the 1-component.

Using (6.1), we obtain the TTS over $\mathbb{N}$ on X by the following two rules, for all $n \in \mathbb{N}$:

$$\frac{}{x \overset{0}{\rightsquigarrow} x} \qquad\qquad \frac{x \overset{n}{\rightsquigarrow} x'' \quad x'' \rightsquigarrow x'}{x \overset{n+1}{\rightsquigarrow} x'}$$

Hence, the $\rightsquigarrow$-transitions, as defined by the $B_{\mathbb{N}}$-coalgebra, precisely correspond to the $\overset{1}{\rightsquigarrow}$-transitions in the TTS, as was already remarked before.

These considerations mean that we can directly apply the framework of [TP97] to describe abstract operational rules for timed processes over discrete time, i.e., use a natural transformation like (5.7):

$$\Sigma(\mathrm{Id} \times B_{\mathbb{N}}) \Rightarrow B_{\mathbb{N}} T$$

Since, in our case, Σ is simply a polynomial functor associated with a first-order signature, this means that for each n-ary operator $\sigma \in \Sigma$, we must describe a function as in (5.8), i.e.,

$$[\![\sigma]\!] : (X \times (1+X))^n \to 1 + TX$$

which must be natural in X, i.e., invariant under variable renamings. The question is now how to translate maps like $[\![\sigma]\!]$ into concrete rules.

For this, let $\mathcal{V}$ be a fixed countable set of 'meta' variables with a fixed enumeration (without repetitions), i.e., we have

$$\mathcal{V} = \{v_k \mid k \in \mathbb{N} \wedge k \geq 1\}$$

In the following, we will sometimes refer to such a set $\mathcal{V}$ as an *enumerated* set. We will use the enumeration to make sure that the rules induce a *deterministic* transition relation which, evidently, is the behaviour-inherent condition associated with $B_{\mathbb{N}}$.

Definition 6.1

Let $\mathcal{V}$ be an enumerated set of variables and Σ a (first-order) signature. Then say that a *deterministic single-label prerule*, in short *dsl-prerule* or even only *prerule*, over Σ is an operational rule of the form

$$\frac{\{v_i \rightsquigarrow v_{n+i}\}_{i \in I} \quad \{v_j \not\rightsquigarrow\}_{j \in J}}{\sigma\langle v_1, \ldots, v_n\rangle \rightsquigarrow \theta} \tag{6.2}$$

where: σ is an n-ary operator symbol in Σ; $v_k \in \mathcal{V}$; $I, J \subseteq \{1, \ldots, n\}$; θ is a term over Σ and $\mathcal{V}$ such that all variables occurring in θ are contained in the set

$$\{v_k \mid 1 \leq k \leq n\} \cup \{v_{n+i} \mid i \in I\}$$

(this union is actually disjoint since the enumeration on $\mathcal{V}$ was assumed to be without repetitions). ■

Note the following subtlety in the definition of a dsl-prerule: it is *not*—as falsely claimed in [Kic02a]—equivalent to say that θ is allowed to contain *all* variables in the set $\{v_k \mid 1 \leq k \leq n + |I|\}$, at least not with the current definition. As an example, consider the following example for $n \geq 3$:

$$\frac{v_1 \rightsquigarrow v_{n+1} \quad v_3 \rightsquigarrow v_{n+3}}{\sigma\langle v_1, \ldots, v_n\rangle \rightsquigarrow v_{n+2}}$$

According to the alternative 'definition,' since $I = \{1, 3\}$, the above rule would be a dsl-prerule since $\theta = v_{n+2}$ only contains variables in the set $\{v_1, \ldots, v_{n+2}\}$ although it allows θ to contain variables not occurring anywhere else in the rules (showing that such a rule need not necessarily adhere to the GSOS conditions). However, according to our (correct) definition, it is not a dsl-prerule since v_{n+2} does not occur among either the argument variables nor the targets of positive premises.

Note that this allows for 'holes' in the sequence of variables occurring in θ: using the same premises as in the above example rule, θ could contain both v_{n+1} and v_{n+3} yet not v_{n+2}. It would be possible to adapt the alternative definition to really be equivalent to the one above, yet that would lead into too much unnecessary bookkeeping concerning indices; hence we chose the above definition as it is. Furthermore, note that according to the definition of a dsl-prerule, each prerule can have *at most* one positive and one negative premise for any of the argument variables: I and J are sets, hence only at most once contain an index. We can now prove the following proposition:

Proposition 6.2
Every dsl-prerule is a GSOS rule.

Proof: By our use of the enumeration $\mathcal{V}$, it automatically follows that all the argument variables $v_1, \ldots, v_n$ of a dsl-prerule must be distinct, the same applies to the variables

v_{n+i} for $i \in I$. Moreover, by restricting θ in such a way that it can only contain either argument variables or targets of positive premises, also the second GSOS condition is satisfied: any variable occurring in θ must occur somewhere else in the rule. □

Definition 6.3

Given a dsl-prerule (6.2) for an n-ary operator symbol $\sigma \in \Sigma$, its *type* is the tuple $\langle I,J \rangle \in \mathcal{P}(\{1,\ldots,n\})^2$. Such a prerule is *consistent* if $I \cap J = \emptyset$, and it is *complete* if $I \cup J = \{1,\ldots,n\}$. A *dsl-rule* is a consistent and complete dsl-prerule. ■

If a prerule is consistent, no argument variable v_k, $1 \leq k \leq n$, can appear in both a positive and a negative premise which, intuitively, makes perfect sense: a prerule containing both of the premises $v_i \rightsquigarrow v_{n+i}$ and $v_i \not\rightsquigarrow$ for some $i \in I \subseteq \{1,\ldots,n\}$ would never be applicable because always one of the premises must be false. In the context of GSOS rules, such rules are also known as 'junk rules' because they do not contain any information; hence no dsl-rule can be a junk rule: it must always allow to derive a step, when instantiated with suitable arguments.

In contrast to that, a dsl-prerule is complete if each of the argument variables v_k, $1 \leq k \leq n$, occurs as the source of at least one premise, be it positive or negative (we have already remarked that there cannot be more than one positive or negative premise for the same variable in a dsl-prerule). Intuitively, this means that a complete prerule does not allow for the behaviour of any argument process to be unspecified: all arguments have to be 'inspected,' so to speak, in order to determine whether the rule is applicable or not.

Combining consistency and completeness, the characteristic feature of a dsl-rule, on top of the GSOS conditions on variable occurrences which are satisfied because a dsl-rule is in particular a prerule, cf. Proposition 6.2, is that each argument variable occurs as the source of *exactly one* premise, either a positive or a negative one, as illustrated in the following example of a dsl-rule (with no particular intuitive meaning):

$$\frac{v_1 \rightsquigarrow v_3 \quad v_2 \not\rightsquigarrow}{\sigma\langle v_1, v_2 \rangle \rightsquigarrow \sigma\langle v_1, v_3 \rangle}$$

Note that the following is *not* a dsl-rule since it is only consistent but not complete:

$$\frac{}{\sigma(v_1) \rightsquigarrow \sigma(v_1)} \tag{6.3}$$

However, the same semantics for σ can be specified by using the following pair of dsl-rules:

$$\frac{v_1 \rightsquigarrow v_2}{\sigma(v_1) \rightsquigarrow \sigma(v_1)} \qquad \frac{v_1 \not\rightsquigarrow}{\sigma(v_1) \rightsquigarrow \sigma(v_1)}$$

This indicates the basis of an algorithm which transforms a given set of consistent prerules into a set of dsl-rules by this kind of *completion* process: if a rule is not complete, introduce the case distinction in the premises as above. For the extreme case, consider the following n-ary variant of (6.3):

$$\frac{}{\sigma\langle v_1, \ldots, v_n\rangle \rightsquigarrow \sigma\langle v_1, \ldots v_n\rangle}$$

This behaviour needs just a single incomplete rule, yet to describe it using only complete rules one actually needs 2^n many rules, one rule for each possible value of an n-ary bit vector describing which of the arguments can or cannot perform a step.

Even though this might suggest to drop the completeness requirement on dsl-rules, keeping it makes the theory a lot simpler. When using incomplete rules, redundancies can arise, e.g., consider the following two rules, both of which are consistent, the second is even a dsl-rule:

$$\frac{}{\sigma(v_1) \rightsquigarrow \sigma(v_1)} \qquad \frac{v_1 \rightsquigarrow v_2}{\sigma(v_1) \rightsquigarrow \sigma(v_1)}$$

Clearly, the second rule is *subsumed* by the first in the sense that whenever the second is applicable, also the first is applicable and allows to derive the same step, while the converse is not true; effectively, it would be possible to drop the second rule without changing the possible derivations. Since our goal is to obtain a correspondence between natural transformations (5.7) and certain sets of concrete rules similar to (6.2), such redundancies create problems for obtaining a one-to-one correspondence between abstract and concrete rules: allowing incomplete rules, we would only be able to prove that, starting from a *minimal* set of rules in the sense that no rule is subsumed by another, one would get the very same set of rules back after translating concrete into abstract rules and back. The problematic step is to derive concrete rules from abstract ones: in order to obtain a deterministic algorithm, this necessitates to derive either minimal or maximal (i.e., with the maximal amount of redundancy) sets of rules; hence, when one starts with an intermediate amount of redundancy, the translation back into concrete rules will only be equivalent up to proving the same steps, as was the case for GSOS in [TP97].

So far, using arbitrary sets of dsl-rules, we cannot yet guarantee to really describe a deterministic transition system on the terms of the language under consideration: consider, e.g., the following two dsl-rules

$$\frac{v_1 \rightsquigarrow v_3 \quad v_2 \rightsquigarrow v_4}{\sigma\langle v_1, v_2\rangle \rightsquigarrow v_1} \qquad \frac{v_1 \rightsquigarrow v_3 \quad v_2 \rightsquigarrow v_4}{\sigma\langle v_1, v_2\rangle \rightsquigarrow v_2}$$

Using these two rules, the process $\sigma(v_1, v_2)$ could have two distinct $\rightsquigarrow$-successors, viz., v_1 or v_2, and thus the rules are not deterministic. The problem with the two rules is that both are applicable *at the same time*. This leads to the following definition:

Definition 6.4

Two dsl-prerules for the same n-ary operator symbol $\sigma \in \Sigma$ with respective types $\langle I_k, J_k\rangle$, $1 \leq k \leq 2$, are *mutually exclusive* if

$$(I_1 \cap J_2) \cup (I_2 \cap J_1) \neq \emptyset \tag{6.4}$$

A set of dsl-rules is in *dsl-format* if any two distinct rules for the same operator are mutually exclusive. ■

Let us analyse the intuitive meaning of (6.4). If satisfied, we know that $I_1 \cap J_2 \neq \emptyset$ or $I_2 \cap J_1 \neq \emptyset$; without loss of generality, assume the first case, i.e., $I_1 \cap J_2 \neq \emptyset$. Therefore, there exists $i \in \{1, \ldots, n\}$ such that $i \in I_1$ and $i \in J_2$. By the definition of what it means to be prerule of type $\langle I_k, J_k\rangle$, this states that there is a positive premise in the first rule of the form $v_i \rightsquigarrow v_{n+i}$, and a negative premise of the form $v_i \not\rightsquigarrow$ in the second one. For any instantiation p_i of v_i with any concrete process, we have that *either* $p_i \rightsquigarrow$ *or* $p_i \not\rightsquigarrow$, in other words: *exactly one* of the two rules is applicable. Hence, mutually exclusive rules are never applicable at the same time. Thus, given a set of dsl-prerules which are pairwise mutually exclusive, always at most one rule is applicable in any given situation. Note that for dsl-rules, (6.4) is equivalent to simply $I_1 \neq I_2$, since in this specific case, $J_k = \{1, \ldots, n\} \setminus I_k$.

So what does it mean for a set of dsl-rules to be in dsl-format? First of all, each rule must be consistent, thereby excluding junk rules which do not allow to derive any steps anyway. Next, all rules must be complete, i.e., there must be a (either positive or negative, but not both because of consistency) premise for each argument variable, thereby prohibiting the possibility that rules subsume each other: one easily shows that a complete prerule for some operator σ subsumes another complete prerule for σ if and only if they are identical. Hence, using complete rules makes redundancy impossible. Finally, mutual exclusion enforces a deterministic transition relation, as just discussed. Note that the behaviour-inherent condition associated with $B_{\mathbb{N}}$, viz., determinacy, can only be enforced at the level of sets of rules, not on a per-rule basis, similar to the restriction to image finite sets of GSOS rules in [TP97]. These three properties enable us to prove the following proposition:

Proposition 6.5
If Σ is an arbitrary signature and $\sigma \in \Sigma$, then any rules over Σ in dsl-format can have only finitely many rules for σ. Consequently, if Σ is finite (as is usually the case), all sets of rules in dsl-format are automatically finite.

Proof: Let $\sigma \in \Sigma$ be an n-ary operator symbol. By the definition of prerules, any dsl-rule for σ must always use the argument variables $v_1, \ldots, v_n \in \mathcal{V}$. Moreover, there are only 2^n many different possibilities of those arguments being able or unable to perform a step. Hence, since any two rules for σ must be mutually exclusive, there can only be at most 2^n many rules without violating mutual exclusion. $\square$

Now we can prove the following important result, relative to a fixed enumerated set $\mathcal{V}$ of variables:

Theorem 6.6
There is a one-to-one correspondence between operational rules in dsl-format of $\mathcal{V}$ and natural transformations of type $\Sigma(\mathrm{Id} \times B_{\mathbb{N}}) \Rightarrow B_{\mathbb{N}}T$

We will split the proof of Theorem 6.6 into several results, but first we need to introduce some definitions.

Definition 6.7
Given a set X, and an n-tuple $\langle \vec{x}, \vec{\beta} \rangle \stackrel{\mathrm{df}}{=} \langle \langle x_1, \beta_1 \rangle, \ldots, \langle x_n, \beta_n \rangle \rangle \in (X \times B_{\mathbb{N}}X)^n$, we say that its *type* is $\langle I, J \rangle$ for $I, J \subseteq \{1, \ldots, n\}$ if $I = \{1 \leq i \leq n \mid \beta_i \neq \star\}$ and $J = \{1, \ldots, n\} \setminus I$;

for a function $f : X \to Y$, we denote the tuple $(f \times B_{\mathbb{N}} f)^n \langle \vec{x}, \vec{\beta} \rangle \in (Y \times B_{\mathbb{N}} Y)^n$ simply by $\langle \overrightarrow{fx}, \overrightarrow{f\beta} \rangle$. Given a (first-order) signature Σ with term monad T, a set X, and a term $\theta \in TX$, we denote the usual *substitution* of $x_1, \ldots, x_n \in X$ by $y_1, \ldots, y_n \in X$ by $\theta[y_1/x_1, \ldots, y_n/x_n]$. ■

We now prove that to each set R of rules in dsl-format, we can associate a natural transformation $[\![R]\!]$ of type (5.7), and from such a natural transformation ρ, we can derive a set $\langle\!\langle \rho \rangle\!\rangle$ of rules in dsl-format such that the two constructions are additionally mutually inverse.

Lemma 6.8
Let Σ be a signature with term monad T, and let R be a set of operational rules in dsl-format. Then this induces a natural transformation $[\![R]\!] : \Sigma(\mathrm{Id} \times B_{\mathbb{N}}) \Rightarrow B_{\mathbb{N}} T$.

Proof: For any set X, and any n-ary operator symbol $\sigma \in \Sigma$, we have to define a map

$$[\![\sigma]\!]_X : (X \times B_{\mathbb{N}} X)^n \to B_{\mathbb{N}} TX = 1 + TX$$

So let $\langle \vec{x}, \vec{\beta} \rangle \in (X \times B_{\mathbb{N}} X)^n$ with type $\langle I, J \rangle$. If R does not contain any rule of this very type, define $[\![\sigma]\!]_X \langle \vec{x}, \vec{\beta} \rangle = \star$.

Assume then that there is a rule in R of type $\langle I, J \rangle$. Since R is in dsl-format, this rule is necessarily *unique*: otherwise, mutual exclusion would be violated. This unique *matching rule* is then necessarily of the form

$$r = \frac{\{v_i \rightsquigarrow v_{n+i}\}_{i \in I} \quad \{v_j \not\rightsquigarrow\}_{j \in J}}{\sigma\langle v_1, \ldots, v_n \rangle \rightsquigarrow \theta}$$

where θ only contains variables from the set $S \stackrel{\mathsf{df}}{=} \{v_1, \ldots, v_n\} \cup \{v_{n+i} \mid i \in I\}$. Since the rule r and the tuple $\langle \vec{x}, \vec{\beta} \rangle$ have the same type, we have that, for all $i \in I$, $\beta_i \neq \star$. Hence, $[\![\sigma]\!]_X$ is well-defined (all v_j in the substitution are distinct!):

$$[\![\sigma]\!]_X \langle \vec{x}, \vec{\beta} \rangle = \theta[x_1/v_1, \ldots, x_n/v_n, (\forall i \in I).\ \beta_i/v_{n+i}] \in TX$$

We now have to prove that this defines the components of a natural transformation $[\![\sigma]\!] : (\mathrm{Id} \times B_{\mathbb{N}})^n \Rightarrow B_{\mathbb{N}} T$. For this, let X, Y be sets and $f : X \to Y$ be a function; we have to show that the following square commutes:

$$\begin{tikzcd}
(X \times B_{\mathbb{N}}X)^n \arrow[r, "[\![\sigma]\!]_X"] \arrow[d, "(f\times B_{\mathbb{N}}f)^n"'] & 1+TX \arrow[d, "1+Tf"] \\
(Y \times B_{\mathbb{N}}Y)^n \arrow[r, "[\![\sigma]\!]_Y"] & 1+TY
\end{tikzcd}$$

Let therefore $\langle \vec{x}, \vec{\beta} \rangle \in (X \times B_{\mathbb{N}}X)^n$ be a tuple of type $\langle I, J \rangle$. The important point to note is that $\langle \overrightarrow{fx}, \overrightarrow{f\beta} \rangle$ also has type $\langle I, J \rangle$, simply by the definition of $1+f$: if $\beta_k = \star$, then $(1+f)(\beta_k) = \star$, and if $\beta_k = x$, for some $x \in X$, $(1+f)(\beta_k) = fx$. An analogous property holds for $1+Tf$: if $[\![\sigma]\!]_X \langle \vec{x}, \vec{\beta} \rangle = \star$, then also $1+Tf$ applied to $\star$ is also $\star$.

Now assume that $[\![\sigma]\!]_X \langle \vec{x}, \vec{\beta} \rangle = \star$, by definition of $[\![\sigma]\!]$ meaning that R contains no matching rule of type $\langle I, J \rangle$. This implies that also $[\![\sigma]\!]_Y \langle \overrightarrow{fx}, \overrightarrow{f\beta} \rangle = \star$, since the type is not changed; hence the diagram commutes in this case.

Now suppose there is a matching rule in R, i.e., a rule r in R of the form

$$\frac{\{v_i \rightsquigarrow v_{n+i}\}_{i \in I} \quad \{v_j \not\rightsquigarrow\}_{j \in J}}{\sigma\langle v_1, \ldots, v_n \rangle \rightsquigarrow \theta}$$

for some term $\theta \in TX$ subject to the conditions on dsl-(pre-)rules. Then we know that

$$[\![\sigma]\!]_X \langle \vec{x}, \vec{\beta} \rangle = \theta[x_1/v_1, \ldots, x_n/v_n, (\forall i \in I).\ \beta_i/v_{n+i}]$$

and that substitution really captures *all* variables occurring in θ. Analogously, since the type remains the same, the same rule r matches with $\langle \overrightarrow{fx}, \overrightarrow{f\beta} \rangle$, and thus, we obtain

$$[\![\sigma]\!]_Y \langle \overrightarrow{fx}, \overrightarrow{f\beta} \rangle = \theta[fx_1/v_1, \ldots, fx_n/v_n, (\forall i \in I).\ f\beta_i/v_{n+i}]$$

Applying Tf to the term $\theta[x_1/v_1, \ldots, x_n/v_n, (\forall i \in I).\ \beta_i/v_{n+i}]$ results in precisely this very same result in TY, which is easily seen. Note that it is essential that θ only contains variables in S: otherwise, the naturality square might fail to commute. $\square$

Hence we know that, starting from a set of operational rules in dsl-format, we can derive a natural transformation. The converse holds as well:

Lemma 6.9

Given a first-order signature Σ, a natural transformation $\rho : \Sigma(\mathrm{Id} \times B_{\mathbb{N}}) \Rightarrow B_{\mathbb{N}}T$ induces a set $\langle\!\langle \rho \rangle\!\rangle$ of operational rules over $\mathcal{V}$ in dsl-format.

Proof: We derive $\langle\!\langle\rho\rangle\!\rangle$ from the component

$$\rho_{\mathcal{V}} : \Sigma(\mathcal{V} \times B_{\mathbb{N}}\mathcal{V}) \to B_{\mathbb{N}}T\mathcal{V}$$

Let therefore $\sigma \in \Sigma$ be an n-ary operator symbol; the procedure for obtaining $\langle\!\langle\rho\rangle\!\rangle$ works as follows:

1. Initialise $\langle\!\langle\rho\rangle\!\rangle = \emptyset$
2. Let $I \subseteq \{1, \dots, n\}$ and take $J = \{1, \dots, n\} \setminus I$.
3. 'Build' a tuple in $(\mathcal{V} \times B_{\mathbb{N}}\mathcal{V})^n$ as follows: $\langle\vec{v}, \vec{\beta}\rangle \stackrel{\mathsf{df}}{=} \langle\langle v_1, \beta_1\rangle, \dots, \langle v_n, \beta_n\rangle\rangle$ by setting $\beta_k \stackrel{\mathsf{df}}{=} v_{n+k}$ if $k \in I$; otherwise, set $\beta_k = \star$.
4. Compute $\rho_{\mathcal{V}}\langle\vec{v}, \vec{\beta}\rangle = z$.
5. (a) If $z = \star$, go back to second step with a different I; go through all 2^n possibilities; if there is no other choice left for I, terminate.

   (b) If $z = \theta \in T\mathcal{V}$, add the following rule to $\langle\!\langle\rho\rangle\!\rangle$:
   $$\frac{\{v_i \rightsquigarrow v_{n+i}\}_{i \in I} \quad \{v_j \not\rightsquigarrow\}_{j \in J}}{\sigma\langle v_1, \dots, v_n\rangle \rightsquigarrow \theta}$$
   and then also go back the the second step for different choice of I; if there is no other choice left for I, terminate.

Since this algorithm performs exactly 2^n tests of the function $\rho_{\mathcal{V}}$, termination is clear. We now have to show that all rules in $\langle\!\langle\rho\rangle\!\rangle$ are dsl-rules and that $\langle\!\langle\rho\rangle\!\rangle$ as a whole is in dsl-format.

For the first proof obligation, it is obvious, by construction, that each rule added to $\langle\!\langle\rho\rangle\!\rangle$ by the procedure is consistent and complete. The only thing left to prove is that the variables occurring in a resulting term $\theta \in T\mathcal{V}$ are contained in the set $S \stackrel{\mathsf{df}}{=} \{v_1, \dots, v_n\} \cup \{v_{n+i} \mid i \in I\}$.

Assume for a contradiction that there is a variable $v \in \mathcal{V} \setminus S$ which occurs in θ; since $\mathcal{V}$ is countably infinite, we can furthermore find another $v' \in \mathcal{V} \setminus S$ such that $v \neq v'$. Define the function $f : \mathcal{V} \to \mathcal{V}$ by setting

$$v_k \mapsto \begin{cases} v_k & \text{if } v_k \neq v \\ v' & \text{if } v_k = v \end{cases}$$

Intuitively, f simply maps v to v' and leaves any other variable unchanged. We know that ρ is natural, and $(f \times B_{\mathbb{N}} f)^n \langle \vec{v}, \vec{\beta} \rangle = \langle \vec{v}, \vec{\beta} \rangle$, by definition of f. Furthermore, $\rho_{\mathcal{V}}(\sigma\langle \vec{v}, \vec{\beta} \rangle) = \theta \in \mathcal{V}$ contains an occurrence of $v \notin S$. Yet, $B_{\mathbb{N}} T f(\theta)$, which by naturality should be equal to θ, does no longer contain any occurrence of v since f replaces any such occurrence by v', hence contradicting the naturality of ρ. Therefore, θ can only contain variables in S, showing that $\langle\!\langle \rho \rangle\!\rangle$ contains only dsl-rules.

As for mutual exclusion, we merely remark that the procedure for constructing $\langle\!\langle \rho \rangle\!\rangle$ adds *at most one* rule for any given type, so mutual exclusion is trivially guaranteed. Consequently, $\langle\!\langle \rho \rangle\!\rangle$ is indeed in dsl-format. □

Note that $\langle\!\langle \rho \rangle\!\rangle$ effectively only depends on the component $\rho_{\mathcal{V}}$. Thus, we now have two constructions, $[\![\cdot]\!]$ and $\langle\!\langle \cdot \rangle\!\rangle$, mapping rules to natural transformations and vice versa. We now have to show that the two are mutually inverse, with respect to a fixed enumerated set $\mathcal{V}$ of variables.

Lemma 6.10
If R is a set of operational rules in dsl-format, $R = \langle\!\langle [\![R]\!] \rangle\!\rangle$.

Proof: Assume we have a rule

$$\frac{\{v_i \rightsquigarrow v_{n+i}\}_{i \in I} \quad \{v_j \not\rightsquigarrow\}_{j \in J}}{\sigma\langle v_1, \ldots, v_n \rangle \rightsquigarrow \theta}$$

in R. By definition of $[\![R]\!]$, this implies in particular that $[\![R]\!]_{\mathcal{V}}(\sigma\langle \vec{v}, \vec{\beta} \rangle) = \theta$, where β_k is given by v_{n+k}, if $k \in I$, and equal to $\star$ otherwise. But then $\langle\!\langle [\![R]\!] \rangle\!\rangle$ precisely contains the rule we started from, showing that $R \subseteq \langle\!\langle [\![R]\!] \rangle\!\rangle$.

Assume now we are given a rule in $\langle\!\langle [\![R]\!] \rangle\!\rangle$ of the form

$$\frac{\{v_i \rightsquigarrow v_{n+i}\}_{i \in I} \quad \{v_j \not\rightsquigarrow\}_{j \in J}}{\sigma\langle v_1, \ldots, v_n \rangle \rightsquigarrow \theta}$$

By definition, this means that $[\![R]\!]_{\mathcal{V}}(\sigma\langle \vec{v}, \vec{\beta} \rangle) = \theta$ for appropriate choice of $\vec{\beta}$. This, in turn, can only be the case if there is a rule in R of matching type, and by the use of the enumeration in dsl-rules, this means that it must be equal to the rule we started from, showing also $\langle\!\langle [\![R]\!] \rangle\!\rangle \subseteq R$. □

Also the other composite is the identity:

Lemma 6.11

For a natural transformation $\rho : \Sigma(\mathrm{Id} \times B_{\mathbb{N}}) \Rightarrow B_{\mathbb{N}}T$, we have $\rho = [\![\langle\!\langle \rho \rangle\!\rangle]\!]$.

Proof: Let X be an arbitrary set; we will show that the two components ρ_X and $[\![\langle\!\langle \rho \rangle\!\rangle]\!]_X$ are equal. Assume that $\sigma \in \Sigma$ is an n-ary operator symbol, and let $\langle \vec{x}, \vec{\beta} \rangle \in (X \times B_{\mathbb{N}}X)^n$ a tuple with type $\langle I, J \rangle$. If $[\![\langle\!\langle \rho \rangle\!\rangle]\!]_X(\sigma\langle \vec{x}, \vec{\beta} \rangle) = \star$, then $\langle\!\langle \rho \rangle\!\rangle$ does not contain a matching rule of the same type. This, in turn, implies that $\rho_{\mathcal{V}}$, applied to a suitable tuple of the same type, also yields $\star$. Using the naturality of ρ to construct an appropriate renaming $\mathcal{V} \to X$, we obtain $\rho_X(\sigma\langle \vec{x}, \vec{\beta} \rangle) = \star$, and the argument also goes through in the converse direction: if $\rho_X(\sigma\langle \vec{x}, \vec{\beta} \rangle) = \star$, then also $[\![\langle\!\langle \rho \rangle\!\rangle]\!]_X(\sigma\langle \vec{x}, \vec{\beta} \rangle) = \star$.

Assume now that $[\![\langle\!\langle \rho \rangle\!\rangle]\!]_X(\sigma\langle \vec{x}, \vec{\beta} \rangle) = \overline{\theta} \in TX$. By definition of $[\![\cdot]\!]$, this means that $\langle\!\langle \rho \rangle\!\rangle$ must contain a dsl-rule with matching type of the form

$$\frac{\{v_i \rightsquigarrow v_{n+i}\}_{i \in I} \quad \{v_j \not\rightsquigarrow\}_{j \in J}}{\sigma\langle v_1, \ldots, v_n \rangle \rightsquigarrow \theta}$$

such that $\overline{\theta} = \theta[x_1/v_1, \ldots, x_n/v_n, (\forall i \in I).\ \beta_i/v_{n+i}]$. In turn, by definition of $\langle\!\langle \cdot \rangle\!\rangle$, this means that $\rho_{\mathcal{V}}(\sigma\langle \vec{v}, \vec{\gamma} \rangle) = \theta$, where γ_k is defined as v_{n+k} if $k \in I$, and as $\star$ otherwise.

Consider then the function $f : \mathcal{V} \to X$ defined by

$$v_k \mapsto \begin{cases} x_k & \text{if } 1 \leq k \leq n \\ \beta_i & \text{if } k = n+i \text{ for some } i \in I \\ v & \text{otherwise} \end{cases}$$

where $v \in \mathcal{V} \setminus S$, for $S \stackrel{\mathsf{df}}{=} \{v_1, \ldots, v_n\} \cup \{v_{n+i} \mid i \in I\}$, is arbitrarily chosen. Then the following two properties hold:

1. $\langle \overrightarrow{fv}, \overrightarrow{f\gamma} \rangle = \langle \vec{x}, \vec{\beta} \rangle$
2. $Tf(\theta) = \overline{\theta}$

The first property follows immediately from the definition of f, while the second one is obtained from the characterisation of $\overline{\theta}$ as the result of applying a substitution to θ. Then, using the naturality of ρ, we get that

$$\rho_X(\sigma\langle \vec{x}, \vec{\beta} \rangle) = \rho_X(\sigma\langle \overrightarrow{fv}, \overrightarrow{f\gamma} \rangle) = Tf(\rho_{\mathcal{V}}(\sigma\langle \vec{v}, \vec{\gamma} \rangle)) = Tf(\theta) = \overline{\theta} = [\![\langle\!\langle \rho \rangle\!\rangle]\!]_X\langle \vec{x}, \vec{\beta} \rangle$$

concluding the proof. $\square$

This last lemma concludes the proof of Theorem 6.6, showing that there is a precise one-to-one correspondence between operational rules in dsl-format and natural transformations of type (5.7), and consequently giving an elementary syntactic format for well-behaved rules for timed processes over discrete time: any language whose time rules fit the dsl-format satisfies the property that time bisimulation is a congruence.

Example 6.12
An example of rules in the above format is given by the time rules of the language ATP [NS94], for example, we can write down the rules for non-deterministic choice, written $\oplus$ in ATP, and the time-out operator as dsl-rules:

$$\frac{v_1 \rightsquigarrow v_3,\ v_2 \rightsquigarrow v_4}{v_1 \oplus v_2 \rightsquigarrow v_3 \oplus v_4}$$

$$\frac{v_1 \rightsquigarrow v_3,\ v_2 \rightsquigarrow v_4}{\lfloor v_1 \rfloor (v_2) \rightsquigarrow v_2} \qquad \frac{v_1 \rightsquigarrow v_3,\ v_2 \not\rightsquigarrow}{\lfloor v_1 \rfloor (v_2) \rightsquigarrow v_2}$$

$$\frac{v_1 \not\rightsquigarrow,\ v_2 \rightsquigarrow v_4}{\lfloor v_1 \rfloor (v_2) \rightsquigarrow v_2} \qquad \frac{v_1 \not\rightsquigarrow,\ v_2 \not\rightsquigarrow}{\lfloor v_1 \rfloor (v_2) \rightsquigarrow v_2}$$

The type of the rule for $\oplus$ is $\langle\{1,2\},\emptyset\rangle$; for time-out, note that the single rule (2.2) of the original calculus was turned into a set of *four* dsl-rules in order to satisfy the completeness requirement; this is precisely as indicated by the completion procedure of a set of consistent but not complete rules sketched before; also note that the third rule, of type $\langle\{2\},\{1\}\rangle$, is an example of a rule where v_4, but not v_3, is used in the premises. $\Diamond$

6.1.1 Single-step TeCCS

We can now finally define a new single-step version of the time rules of TeCCS fitting the dsl-format. In the next section, we will present a format which contains the general case of TeCCS over an arbitrary time domain. Using Theorem 6.6 and the isomorphism of $E_{\mathbb{N}}$-Coalgebras and $B_{\mathbb{N}}$-coalgebras, we will then prove categorically that the one-step rules induce the same TTS on the set of TeCCS terms as the original rules.

The syntax of the simplified version stays the same, only the operational semantics changes from using $\overset{t}{\rightsquigarrow}$-transitions (for $t \in \mathcal{T}$) to $\rightsquigarrow$-transitions, capturing their intuitive meaning as being equal to $\overset{1}{\rightsquigarrow}$-transitions (in particular when considering the rules for time prefixing), as shown in Figure 6.1.

$$\frac{}{\delta.p \rightsquigarrow \delta.p}$$

$$\frac{}{(t+1).p \rightsquigarrow (t).p} \qquad \frac{}{(1).p \rightsquigarrow p}$$

$$\frac{p \rightsquigarrow p' \quad q \rightsquigarrow q'}{p+q \rightsquigarrow p'+q'}$$

$$\frac{p \rightsquigarrow p' \quad q \rightsquigarrow q'}{p \oplus q \rightsquigarrow p' \oplus q'}$$

$$\frac{p \rightsquigarrow p' \quad q \not\rightsquigarrow}{p \oplus q \rightsquigarrow p'} \qquad \frac{p \not\rightsquigarrow \quad q \rightsquigarrow q'}{p \oplus q \rightsquigarrow q'}$$

$$\frac{p \rightsquigarrow p' \quad q \rightsquigarrow q'}{p|q \rightsquigarrow p'|q'}$$

Figure 6.1: The operational rules of Single-step TeCCS.

As in the original paper [MT90], there are *no rules* for the Nil-process 0 and action prefixing $\alpha.\_$; in contrast, there are only two rules for time prefixing $(t).\_$ because the third rule from the original calculus

$$\frac{p \overset{s}{\rightsquigarrow} p'}{(t).p \overset{t+s}{\rightsquigarrow} p'}$$

(already written down in the slightly corrected way, cf. Section 3.3) does not make sense in the context of single-step transitions as defined by a $B_{\mathbb{N}}$-coalgebra.

The above set of single-step rules for TeCCS is not yet in dsl-format: they do not use variables from an enumerated set, and additionally, the rules for both the δ-prefix and for time prefixing are not complete rules. But as sketched above, since all rules are consistent and satisfy the conditions on variable occurrences, they can easily be turned into a set of dsl-rules in dsl-format; in particular, we know that time bisimulation over the naturals is a congruence for this simplified version of TeCCS.

Additionally, we know that the simplified rules induce a natural transformation of the type

$$\Sigma(\mathrm{Id} \times B_{\mathbb{N}}) \Rightarrow B_{\mathbb{N}}T$$

which induces a distributive law of T over the cofree comonad $B_{\mathbb{N}}^{\infty} = E_{\mathbb{N}}$ over $B_{\mathbb{N}}$. The rules also induce a lifting $\tilde{T}_B$ of T to the $B_{\mathbb{N}}$-coalgebras and so in particular, we obtain a $B_{\mathbb{N}}$-coalgebra on the set $T0$ of closed TeCCS terms, derived from the trivial initial $B_{\mathbb{N}}$-coalgebra $! : 0 \to B_{\mathbb{N}}0$. This is then the intended operational model described by the rules above:

$$
\begin{tikzcd}
0 \arrow[r, "\eta_0"] \arrow[d, "!"'] & T0 \arrow[d, dashed, "\tilde{T}_B(!)"'] & & \Sigma T0 \arrow[ll, "\gamma_{T0}"'] \arrow[d, "\Sigma\langle \mathrm{id}_{T0}, \tilde{T}_B(!)\rangle"] \\
B_{\mathbb{N}}0 \arrow[r, "B_{\mathbb{N}}\eta_0"] & B_{\mathbb{N}}T0 & B_{\mathbb{N}}T^2 0 \arrow[l, "B_{\mathbb{N}}\mu_0"'] & \Sigma(T0 \times B_{\mathbb{N}}T0) \arrow[l, "\rho_{T0}"']
\end{tikzcd}
\tag{6.5}
$$

writing η, μ, and γ for the operations of the syntax monad T freely generated from the signature Σ, and its free Σ-algebra structure, respectively.

Let us quickly describe the $B_{\mathbb{N}}$-coalgebra $\tilde{T}_B(!) : T0 \to B_{\mathbb{N}}T0 = 1 + T0$ in concrete terms; we will do this by case analysis on the structure of the argument term $\theta \in T0$ (for some of the operators):

1. $\theta = \mathrm{nil}$ or $\theta = \alpha.\theta'$: $\tilde{T}_B(!)(\theta) = \star$

2. $\theta = \delta.\theta'$: $\tilde{T}_B(!)(\theta) = \theta$

3. $\theta = (t).\theta'$: $\tilde{T}_B(!)(\theta) = \begin{cases} (t-1).\theta' & \text{if } t > 1 \\ \theta' & \text{if } t = 1 \end{cases}$

4. $\theta = \theta_1 + \theta_2$:

$$\tilde{T}_B(!)(\theta) = \begin{cases} \tilde{T}_B(!)(\theta_1) + \tilde{T}_B(!)(\theta_2) & \text{if } \tilde{T}_B(!)(\theta_1) \neq \star \wedge \tilde{T}_B(!)(\theta_2) \neq \star \\ \star & \text{if } \tilde{T}_B(!)(\theta_1) = \star \vee \tilde{T}_B(!)(\theta_2) = \star \end{cases}$$

   The analogous definition applies to parallel composition.

5. $\theta = \theta_1 \oplus \theta_2$:

$$\tilde{T}_B(!)(\theta) = \begin{cases} \tilde{T}_B(!)(\theta_1) \oplus \tilde{T}_B(!)(\theta_2) & \text{if } \tilde{T}_B(!)(\theta_1) \neq \star \wedge \tilde{T}_B(!)(\theta_2) \neq \star \\ \tilde{T}_B(!)(\theta_1) & \text{if } \tilde{T}_B(!)(\theta_1) \neq \star \wedge \tilde{T}_B(!)(\theta_2) = \star \\ \tilde{T}_B(!)(\theta_2) & \text{if } \tilde{T}_B(!)(\theta_1) = \star \wedge \tilde{T}_B(!)(\theta_2) \neq \star \\ \star & \text{if } \tilde{T}_B(!)(\theta_1) = \star \wedge \tilde{T}_B(!)(\theta_2) = \star \end{cases}$$

Since $B_{\mathbb{N}}$**–coalg** $\cong E_{\mathbb{N}}$**–Coalg**, $\tilde{T}_B(!)$ corresponds to a unique $E_{\mathbb{N}}$-Coalgebra on $T0$, i.e., a TTS on the set of closed TeCCS terms, just like the intended operational model for the original calculus. We will, in the next section, show how to obtain a categorical formulation of the latter, and we will also show that it is (modulo the isomorphism of the two categories of coalgebras) equal to the simpler model just defined.

Remark 6.13

We have seen (in Remark 4.32) that also for local qualitative time, i.e., for $\mathcal{T} = C^{\star}$, the evolution comonad is cofreely generated from the functor $B_C = (B_{\mathbb{N}})^C = (1+\mathrm{Id})^C$. Hence, the framework of [TP97] can again be applied directly: abstract rules are obtained by instantiating (5.4), as was already mentioned in the previous chapter, and a syntactic rule format can, at least in principle, be derived using similar methods as shown in this section. This was not done due to lack of time during the later stages of writing the thesis but at least some ideas shall be briefly presented here.

The rules in a format for local qualitative time would be of the form

$$\frac{x_1 \overset{c_1}{\rightsquigarrow} x_1' \cdots x_n \overset{c_n}{\rightsquigarrow} x_n'}{\sigma\langle x_1,\ldots,x_n\rangle \overset{c}{\rightsquigarrow} \theta}$$

where $c, c_1, \ldots, c_n \in C$. We do not expect any restrictions on the relation between these different clocks to be necessary, in particular on them being equal or not. One has again to guarantee that the rules are deterministic, which needs C-dimensional extensions of the above constraints of consistency, completeness, and mutual exclusivity.

It should then follow easily that the format includes the clock rules of PMC—after also including time-parameterised, or rather, *clock-parameterised* operators, cf. the rules for time-out and ignore presented in Section 2.3; such operators might give rise to some additional constraints which relate clocks occurring as parameters to the ones occurring as labels of transitions in premises and conclusions of rules. Note that one only obtains finite sets of finitary rules in the case that C is finite. Otherwise, the rules may have infinitely many premises, allowing to ‘test’ for potential successors in all C different dimensions, as is definitely the case when using complete rules. Moreover, sets adhering to the format can then also be infinite: over infinitary rules, mutual exclusion does not guarantee finiteness. $\Diamond$

6.2 The General Case

6.2.1 An Elementary Format

This section presents a simple way of specifying a well-behaved operational semantics of timed processes over an arbitrary time domain. In the previous section, we have seen how to deal with the case of the specific free monoid $\mathbb{N}$ and, in principle at least, with arbitrary free monoids $C^{\star}$. In the remainder of this section, we assume that the time domain $\mathcal{T}$ over which we will define the operational semantics, is antichain-monotone (if $t||u$ holds, then also $t'||u$ for any $t' \geq t$ in the induced order, cf. Definition 4.33), which does not say anything about commutativity or linearity. However, the principal example is of course $\mathbb{R}$ since for $\mathbb{N}$ (or other free monoids), one would certainly use the simpler dsl-format (or its multidimensional variant, cf. the preceding section); note that this excludes time domains like $\mathbb{N}^n$ and $\mathbb{R}^n$.

As for the rule format itself, we use *schematic* operational rules: the rules only contain time transitions labelled by *time variables*, rather than concrete time values. Then, in order to derive concrete time transitions, the time variables in such a rule have to be instantiated with actual values, subject to applicability of the rule. Based on such schematic rules, certain *rule shapes* for defining time transitions are introduced, and only certain combinations of these shapes are allowed as *admissible operators*: instead of a 'format,' the present approach really just yields a collection of 'operator blueprints.'

In order to describe timed processes, admissible operators have to include *time-parameterised* operators, i.e., operators which have time(s) as parameters, in addition to the usual parameters for processes. It should be fairly clear that such operators have to be treated in a special way: after all, using the operations of the time domain, time parameters and the labels of time transitions could potentially be combined in various ways, in particular using the monoid addition (as we shall do).

In deriving the schematic rules, the time rules of TeCCS served as the guiding example. In particular, time prefixing $(t).p$ of TeCCS, for a time $t \neq 0$ and a process p, is the prototype of a time-parameterised operator. For simplicity, the set of admissible operators therefore only allows at most one time parameter. Consequently (and not surprisingly), the admissible operators encompass the time rules of TeCCS, but also additional rules from [NS91]. *Soundness* of the admissible operators is established by showing that admissible operators indeed induce natural transformations as in Theorem 5.18. However, the failure of *completeness* is then demonstrated by presenting a simple example of abstract rules not expressible by an admissible operator.

This is followed by a proof that the intended operational model of TeCCS (over $\mathbb{N}$) is the same as the one induced by the simplified version of the previous section, which only uses $\rightsquigarrow$-transitions. Finally, a way to make the schematic shapes (a little bit) more permissive is discussed by considering them *relative* to a specific time domain, rather than describing operators regardless of the chosen time domain. In this way, at least a little more freedom is gained, and a class of intuitively natural operators can be additionally accommodated.

In the remainder of this section, let $\mathcal{V}$ be a countable set of variables with an enumeration (without repetitions) as in the previous section, and let $\mathcal{T}$ be an antichain-monotone time domain, again writing $\mathcal{T}^+ \stackrel{\text{df}}{=} \mathcal{T} \setminus \{0\}$. Note that *all* time variables in the following rule shapes only range over $\mathcal{T}^+$: it is therefore impossible to derive $\stackrel{0}{\rightsquigarrow}$-transitions. The reason for this restriction is that the ε-diagram in (5.19): it already determines the targets such transitions, as will become clear in the proof of Prop. 6.17. Furthermore, potential time parameters of time parameterised operators cannot have value 0 either, cf. that $(t).p$ is only a TeCCS process for $t \neq 0$ (see also Section 3.3).

Definition 6.14

For $n \in \mathbb{N}$ and $v_i \in \mathcal{V}$, write $\vec{v} \stackrel{\text{df}}{=} \langle v_1, \ldots, v_n \rangle$, and write $\vec{v}' \stackrel{\text{df}}{=} \langle v_{n+1}, \ldots, v_{2n} \rangle$ for the next n variables in the enumeration. Then, let: Σ be a signature and $\sigma \in \Sigma$ be a function symbol of arity $n \in \mathbb{N}$; s, t *time variables* ranging over $\mathcal{T}^+$; $I \subseteq \{1, \ldots, n\}$ and $1 \leq j \leq n$ such that $j \notin I$. Then allowed *rule shapes* are operational rules of the following kinds:

1. Standard operators defined by rules of the shapes

(A)
$$\frac{-}{\sigma\langle\vec{v}\rangle \stackrel{t}{\rightsquigarrow} \sigma\langle\vec{v}\rangle}$$

(B)
$$\frac{v_1 \stackrel{t}{\rightsquigarrow} v_{n+1} \quad \cdots \quad v_n \stackrel{t}{\rightsquigarrow} v_{2n}}{\sigma\langle\vec{v}\rangle \stackrel{t}{\rightsquigarrow} \sigma\langle\vec{v}'\rangle}$$

$$(\mathrm{C}_j) \qquad \frac{v_j \overset{t}{\rightsquigarrow} v_{n+j}, (\forall 1 \leq k \leq n).\ k \neq j \Rightarrow v_k \overset{t}{\not\rightsquigarrow}}{\sigma\langle\vec{v}\rangle \overset{t}{\rightsquigarrow} v_{n+j}}$$

2. Time-parameterised operators defined by rules of the shapes

$$(\mathrm{tA}_I) \qquad \frac{\{v_i \overset{t}{\rightsquigarrow} v_{n+i}\}_{i\in I},\ (\forall 1 \leq k \leq n).\ w_k = \begin{cases} v_{n+k} & \text{if } k \in I \\ v_k & \text{if } k \notin I \end{cases}}{\sigma\langle t+s, \vec{v}\rangle \overset{t}{\rightsquigarrow} \sigma\langle s, \vec{w}\rangle}$$

$$(\mathrm{tB}_{I,j}) \qquad \frac{\{v_i \overset{t}{\rightsquigarrow} v_{n+i}\}_{i\in I}}{\sigma\langle t, \vec{v}\rangle \overset{t}{\rightsquigarrow} v_j}$$

$$(\mathrm{tC}_{I,j}) \qquad \frac{\{v_i \overset{t}{\rightsquigarrow} v_{n+i}\}_{i\in I},\ v_j \overset{s}{\rightsquigarrow} v_{n+j}}{\sigma\langle t, \vec{v}\rangle \overset{t+s}{\rightsquigarrow} v_{n+j}} \qquad \blacksquare$$

Note that for a constant c, i.e., a function symbol c of arity 0, the two shapes (A) and (B) become equal, and there can be no rule of shape (C_j). Furthermore, note that the use of the enumeration, as already in the previous section, again guarantees that all possible rule shapes are special GSOS rules. Note in particular rule shape ($\mathrm{tC}_{I,j}$): we can use the variable v_{n+j} because, by assumption, $j \notin I$, and so v_{n+j} does not appear among the other premises, which only contain v_{n+i} for $i \in I$; hence, v_{n+j} occurs as the target of only one premise.

One quite striking restriction in these rule shapes is that the same $t \in \mathcal{T}$ is always used in both the premises and the conclusions of rules, i.e., in some sense, the rules define time transitions *uniformly* over $\mathcal{T}$, the sole exception being shape ($\mathrm{tC}_{I,j}$) where also an additional $\overset{s}{\rightsquigarrow}$-transition is tested in the premises. We will, in Section 6.2.1.2, show how to make the rule shapes a little bit more permissive in that respect, by allowing certain 'well-behaved' (partial) time transformations $f : \mathcal{T} \rightharpoonup \mathcal{T}^n$ which specify by $f(t)$, if defined, what transitions the n premises have to perform in order to be able to derive a $\overset{t}{\rightsquigarrow}$-transition in the conclusion of a rule.

Another important restriction is the fact that in all rule shapes, the target of a rule conclusion is either of the form $\sigma\langle\vec{v}\rangle$ for some operator symbol σ and variables $v_i \in \mathcal{V}$ (suitably adapted for time-parameterised operators), or simply a variable as, e.g., in shape (C_j). This general restriction arises because we only want to describe abstract temporal rules, i.e., a natural transformation of type $\Sigma E \Rightarrow E(\mathrm{Id} + \Sigma)$, which so to speak have this restriction 'built-in', cf. Remark 5.21.

What makes the rule shapes even more restrictive than that is the fact that it is always the case that each rule shape for some operator symbol σ has *precisely the same* operator symbol σ in the target of the conclusion, unless it is simply a variable: using the above rule shapes, we can only define operators with transitions $\sigma\langle\vec{v}\rangle \overset{t}{\leadsto} \sigma\langle\vec{v}'\rangle$ or $\sigma\langle\vec{v}\rangle \overset{t}{\leadsto} v'$. This is not induced by the type of the natural transformation but a restriction imposed by ourselves and (amongst others) definitely one of the restrictions that make completeness fail, as shall be illustrated later. To avoid this problem, one could consider a more restrictive kind of natural transformation than abstract temporal rules, with the 'same top-most operator' condition built-in; but even then, we doubt that that one could achieve completeness: one obvious restriction is that admissible operators only use finitary rules, while, as we shall see in the next section, evolutions allow to derive infinitary rules.

Definition 6.15

Let Σ be a signature and let $\sigma \in \Sigma$ be a function symbol with arity $n \in \mathbb{N}$. Then, in addition to the trivial case of no rules at all, the *admissible operators* are given as follows. For standard operators:

1. for arity $n = 1$

   (a) one rule of shape (A), or

   (b) one rule of shape (B), or

   (c) one rule of shape (C_j) for $j = 1$

2. for arity $n > 1$

   (a) one rule of shape (A), or

   (b) one rule of shape (B), or

3. Additionally for arity $n = 2$, one rule of shape (B), two rules of shape (C_j), one each for $j = 1$ and $j = 2$

For time-parameterised operators of arbitrary arity, the following operators are admissible:

1. one rule of shape (tA$_I$) for some $I \subseteq \{1,\ldots,n\}$, or
2. one rule for each of the shapes (tA$_I$), (tB$_{I,j}$), and (tC$_{I,j}$), all with matching $I \subseteq \{1,\ldots,n\}$ and $1 \leq j \leq n$ such that $j \notin I$ ■

Again for the case of a constant c, note that the only non-trivial (i.e., consisting of at least one rule shape) admissible operator is given by the case of one rule of shape (A), or equivalently, by shape (B), as was discussed in a previous remark. It should be noted that shape (C$_j$) can only be used for arities $n = 1$ or $n = 2$; this shall be explained later on.

Example 6.16
All the operators of TeCCS can be modelled using the above admissible operators, e.g.:

1. Nil process 0 and action prefix $\alpha.\_$: no rules;
2. Delay prefix $\delta.\_$: the unary case of one rule of shape (A);
3. Strong choice $+$ and parallel composition $|$: the binary case of one rule of shape (B)
4. Weak choice $\oplus$: the binary case of one rule of shape (B) and two rules of shape (C$_j$), one each for $j = 1$ and $j = 2$;
5. Time-prefixing $(t).\_$ for $t \in \mathcal{T}^+$: the unary case of one rule each of the shapes (tA$_I$), (tB$_{I,j}$), and (tC$_{I,j}$) for $I = \emptyset$ and $j = 1$; note the subtle difference, using a $\stackrel{t+s}{\rightsquigarrow}$- instead of a $\stackrel{s+t}{\rightsquigarrow}$- transition in the conclusion, between the original rules in [MT90] and the shape (tC$_{I,j}$); this has no effect when assuming $\mathcal{T}$ to be commutative (see also Section 3.3), but in our case it is a slight change. ♢

The main result of this section is showing that the admissible operators of Definition 6.15 provide a sound operational semantics for timed processes. For this result, note the restriction that the topmost operator in rules of admissible operators either vanishes or stays the same, as was remarked above. This restriction means that, in order to verify whether the rules respect the structure of E, and in particular the δ-diagram, we can do that for each operator separately: the δ-diagram essentially (as will become clear intuitively in the proof of the following proposition, and also formally in the next section on CSOS rules for timed processes) states that applying the rules ρ twice after applying the comultiplication δ is the same as first applying ρ followed by δ; since the topmost operator in the target of conclusions, if at all present, does not change for admissible operators, applying ρ is the same as applying the individual map $[\![\sigma]\!]$ describing the meaning of an operator symbol σ.

Proposition 6.17
Let Σ be a signature and $\sigma \in \Sigma$ an n-ary function symbol. If the time rules for σ can be described by any of the above admissible operators they induce a map, for each set X,

$$[\![\sigma]\!]_X : (EX)^n \to E(X + \Sigma X) \quad \text{or} \quad [\![\sigma]\!]_X : \mathcal{T}^+ \times (EX)^n \to E(X + \Sigma X) \ ,$$

depending on whether σ is a standard or a time-parameterised operator. Moreover, the corresponding map is natural in X and respects the structure of E.

Proof: We have to show several things here: to begin with, we need to translate the rule shapes for a given operator σ into a map $[\![\sigma]\!]_X$ of the appropriate type, for each set X; once that is achieved, we have to show naturality of $[\![\sigma]\!]_X$; finally, we have to show that the natural transformation $[\![\sigma]\!]$ respects the structure of E.

There are eight possible admissible operators, so we would have to describe all of them as maps $[\![\sigma]\!]_X$; since most of the calculations are quite routine and simply slightly different instances of the same problem, we will only present the two most complex cases, viz., a standard operator σ of arity $n = 2$ defined by the two shapes (B) and (C_j), and a time-parameterised operator σ' defined by all three rule shapes (tA_I), ($\mathrm{tB}_{I,j}$) and ($\mathrm{tC}_{I,j}$); the other cases are analogous, but simpler.

As for translating the rules for σ and σ' into maps $[\![\sigma]\!]_X : (EX)^n \to E(X + \Sigma X)$ and $[\![\sigma']\!]_X : \mathcal{T}^+ \times (EX)^n \to E(X + \Sigma X)$, respectively, one thing to note is that the rule shapes themselves do *not* define $\overset{0}{\rightsquigarrow}$-transitions for either operator: we only assumed the time variables to range over $\mathcal{T}^+$. So the first problem is how to define the value of $[\![\sigma]\!]_X\langle\vec{e}\rangle \in E(X+\Sigma X)$ and $[\![\sigma']\!]_X\langle t,\vec{e}\rangle$ at $0 \in \mathcal{T}$. For this, consider the ε-diagram in (5.19). For $\langle e_1,\ldots,e_n\rangle = \vec{e} \in (EX)^n$, it states

$$[\![\sigma]\!]_X\langle\vec{e}\rangle(0) = \varepsilon_{X+\Sigma X}([\![\sigma]\!]_X\langle\vec{e}\rangle) = \mathrm{inr}((\varepsilon)^n\langle\vec{e}\rangle) = \sigma\langle e_1(0),\ldots,e_n(0)\rangle \tag{6.6}$$

and analogously for σ'. Using (4.4) for translating the evolution-based view to a process- or transition-based view, (6.6) states that

$$[\![\sigma]\!]_X\langle\vec{e}\rangle \overset{0}{\leadsto} \sigma\langle e_1(0),\ldots,e_n(0)\rangle$$

When we use variables $\vec{v} \in \mathcal{V}$ as names of (processes whose behaviours are described by) the evolutions $\vec{e}$, thereby identifying an evolution e_i with the name v_i, as well as not distinguishing between syntax $\sigma\langle\vec{v}\rangle$ and application of rules manifested by a function $[\![\sigma]\!]\langle\vec{e}\rangle$, we obtain

$$\sigma\langle\vec{v}\rangle \overset{0}{\leadsto} \sigma\langle\vec{v}\rangle$$

In other words, the ε-diagram imposes a rule-based version of axiom (ZeroDelay) of TTSs on well-behaved operational rules. Using this constraint in the *definition* of $[\![\sigma]\!]$, we obtain two things: a sensible definition of $[\![\sigma]\!]\langle\vec{e}\rangle$ at 0, plus automatic satisfaction of the ε-diagram.

We can now present the translations of the two admissible operators into functions. For $[\![\sigma]\!]_X : (EX)^2 \to E(X+\Sigma X)$, we obtain:

$$\langle e_1,e_2\rangle \;\mapsto\; \lambda t.\begin{cases} \sigma\langle e_1(t),e_2(t)\rangle & \text{if } e_1(t)\downarrow \wedge e_2(t)\downarrow \\ e_1(t) & \text{if } e_1(t)\downarrow \wedge e_2(t)\uparrow \\ e_2(t) & \text{if } e_1(t)\uparrow \wedge e_2(t)\downarrow \\ \text{undef} & \text{if } e_1(t)\uparrow \wedge e_2(t)\uparrow \end{cases}$$

The first clause is prompted by the shape (B), while the other 2 defined clauses correspond to the two rules of shape (C_j) for $j=1$ or $j=2$. Note that $[\![\sigma]\!]\langle e_1,e_2\rangle(t)\downarrow$ if at least one $e_i(t)\downarrow$; consequently, $[\![\sigma]\!]\langle e_1,e_2\rangle(t)\uparrow$ if both $e_i(t)\uparrow$. It is easy to verify that the above map is well-defined, i.e., that, for each tuple $\langle e_1,e_2\rangle$ of arguments, $[\![\sigma]\!]_X\langle e_1,e_2\rangle$ is indeed an evolution on $X+\Sigma X$.

For $[\![\sigma']\!]_X$, we use the following abbreviation

$$A_I^{\vec{e}}(t) \;\equiv\; \bigwedge_{i\in I} e_i(t)\downarrow \;,$$

i.e., $A_I^{\vec{e}}(t)$ holds if and only if all e_i such that $i \in I$ are defined at $t \in \mathcal{T}$, corresponding to the premises $\{v_i \overset{t}{\rightsquigarrow} v_{n+i}\}_{i \in I}$ in the rule shapes. We then obtain the following definition of the function $[\![\sigma']\!]_X : \mathcal{T}^+ \times (EX)^n \to E(X + \Sigma X)$:

$$\langle t, \vec{e} \rangle \mapsto h_t^{\vec{e}} \overset{\mathsf{df}}{=} \lambda s. \begin{cases} \sigma' \langle t - s, e'_1(s), \ldots, e'_n(s) \rangle & \text{if } s < t \wedge A_I^{\vec{e}}(s) \\ e_j(0) & \text{if } s = t \wedge A_I^{\vec{e}}(t) \\ e_j(s - t) & \text{if } s > t \wedge A_I^{\vec{e}}(t) \wedge e_j(s-t)\downarrow \\ \text{undef} & \text{otherwise} \end{cases}$$

where $e'_k(s) = e_k(s)$ if $k \in I$, and $e'_k(s) = e_k(0)$ if $k \notin I$. The first clause of this definition corresponds to shape (tA$_I$), with the e'_k representing the v'_k. The second clause corresponds to shape (tB$_{I,j}$), and the third to shape (tC$_{I,j}$). It follows that $[\![\sigma']\!]\langle t, \vec{e}\rangle(s)\downarrow$ if s and t are comparable, i.e., $\neg(s \| t)$, $A_I^{\vec{e}}(\min\{s,t\})$ holds, and, in the case $s > t$, $e_j(s-t)\downarrow$, i.e., e_j must be able to 'continue' the time transition begun by using up the t units of time we started with; equivalently, this can be expressed as $e_j(s \dot{-} t)\downarrow$: if $s \leq t$, then $s \dot{-} t = 0$ and $e_j(0)\downarrow$ by (4.1); if $s > t$ then $s \dot{-} t = s - t$, as wanted.

Note that the restriction to antichain-monotone time domains is needed for establishing the well-definedness of $[\![\sigma']\!]$: for each argument tuple $\langle t, \vec{e}\rangle$, the resulting function $h_t^{\vec{e}}$ needs to be an evolution over $X + \Sigma X$, i.e., $h_t^{\vec{e}}$ needs to satisfy the two conditions (4.1) and (4.2) of evolutions. The former is no problem since the arguments are evolutions and $0 < t$ for all $t \in \mathcal{T}^+$. However, for the latter, assuming $h_t^{\vec{e}}(s+u)\downarrow$, we need to be able to deduce that $h_t^{\vec{e}}(s)\downarrow$. If $s + u < t$ or $s + u = t$, this is no problem since in both cases, we obtain $s < t$; furthermore, $A_I^{\vec{e}}(s+u) \Rightarrow A_I^{\vec{e}}(s)$, hence $h_t^{\vec{e}}(s)\downarrow$ follows. In the case $s + u > t$, it is, a priori, not clear whether $s \| t$ or not. Under the assumption of antichain-monotonicity, however, it follows that $s \not\| t$, hence either $s < t$, $s = t$ or $s > t$; moreover, in each case, using (4.2) for the e_i, and also Lemma 3.21 for $s > t$, we obtain $h_t^{\vec{e}}(s)\downarrow$.

Now that we know how to translate the rules into the functions $[\![\sigma]\!]_X$ and $[\![\sigma']\!]_X$, we have to show that they are natural. And indeed, naturality holds: all rule shapes are in fact GSOS rules, as remarked above, so similar arguments as in [TP97] and the previous section on the dsl-format apply, together with the fact that any renaming function is *total* and so does not affect the domain of definition of an evolution, hence the naturality square commutes.

So we know that both $[\![\sigma]\!]$ and $[\![\sigma']\!]$ are indeed natural transformations of the appropriate type. We now have to verify that also both satisfy the diagrams in (5.19). As for the ε-diagram, inspection of the two function definitions shows that they both satisfy (6.6) since, by (4.1), evolutions are always defined at 0; moreover, in the definition of $[\![\sigma']\!]$, t only ranges over $\mathcal{T}^+$, i.e., automatically $t > 0$ holds. Hence, the ε-diagram commutes for both.

So the only thing left to verify is the δ-diagram from (5.19) and, as remarked above, we can simply verify that by diagram chasing, replacing ρ with $[\![\sigma]\!]$ and $[\![\sigma']\!]$, respectively. So let us perform the necessary calculations for $[\![\sigma]\!]$. The two resulting functions, $\delta_{X+\Sigma X} \circ [\![\sigma]\!]_X$ and $E(E\mathrm{inl}_X + [\![\sigma]\!]_X) \circ [\![\sigma]\!]_{EX} \circ \Sigma\delta_X$, are shown in Figure 6.2 and 6.3. Careful case analysis shows that the functions are equal, hence the diagram commutes; for this, one uses axiom (4.2) of evolutions, viz., $e_i(t)\uparrow$ implies $e_i(t+u)\uparrow$.

$$\lambda t.\ \begin{cases} \lambda u.\ \begin{cases} \sigma\langle e_1(t+u), e_2(t+u)\rangle & \text{if } e_1(t+u)\downarrow \wedge e_2(t+u)\downarrow \\ e_1(t+u) & \text{if } e_1(t+u)\downarrow \wedge e_2(t+u)\uparrow \\ e_2(t+u) & \text{if } e_1(t+u)\uparrow \wedge e_2(t+u)\downarrow \\ \text{undef} & \text{otherwise} \end{cases} & \text{if } e_1(t)\downarrow \vee e_2(t)\downarrow \\ \text{undef} & \text{if } e_1(t)\uparrow \wedge e_2(t)\uparrow \end{cases}$$

Figure 6.2: The function $\delta_{X+\Sigma X} \circ [\![\sigma]\!]$

For $[\![\sigma']\!]$, we have to carry out the same verifications. The resulting functions are shown in Figure 6.4 and 6.5.

Careful inspection again yields that the two functions are equal; this time, some of the inequalities about the relative inverses have to be recalled from Section 3.1, in particular the ones from Lemma 3.21. Hence, both $[\![\sigma]\!]$ and $[\![\sigma']\!]$ respect the structure of E, and the claim follows. □

It is quite interesting that, for showing that $[\![\sigma']\!]$ is well-defined, the property of antichain-monotonicity once again plays a decisive role, much like it did in Section 4.4,

$$\lambda t.\ \begin{cases} \lambda u.\ \begin{cases} \sigma\langle e_1(t+u), e_2(t+u)\rangle & \text{if } e_1(t+u)\downarrow \wedge e_2(t+u)\downarrow \\ e_1(t+u) & \text{if } e_1(t+u)\downarrow \wedge e_2(t+u)\uparrow \\ e_2(t+u) & \text{if } e_1(t+u)\uparrow \wedge e_2(t+u)\downarrow \\ \text{undef} & \text{if } e_1(t+u)\uparrow \wedge e_2(t+u)\uparrow \end{cases} & \text{if } e_1(t)\downarrow \wedge e_2(t)\downarrow \\ \lambda u.\ \begin{cases} e_1(t+u) & \text{if } e_1(t+u)\downarrow \\ \text{undef} & \text{if } e_1(t+u)\uparrow \end{cases} & \text{if } e_1(t)\downarrow \wedge e_2(t)\uparrow \\ \lambda u.\ \begin{cases} e_2(t+u) & \text{if } e_2(t+u)\downarrow \\ \text{undef} & \text{if } e_2(t+u)\uparrow \end{cases} & \text{if } e_1(t)\uparrow \wedge e_2(t)\downarrow \\ \text{undef} & \text{if } e_1(t)\uparrow \wedge e_2(t)\uparrow \end{cases}$$

Figure 6.3: The function $E(E\mathrm{inl}_X + [\![\sigma]\!]_X) \circ [\![\sigma]\!]_{EX} \circ \Sigma\delta_X$

$$\lambda s.\ \begin{cases} \lambda u.\ \begin{cases} \sigma'\langle t-(s+u), e'_1(s+u), \ldots, e'_n(s+u)\rangle & \text{if } s+u < t \wedge A_I^{\vec{e}}(s+u) \\ e_j(0) & \text{if } s+u = t \wedge A_I^{\vec{e}}(s+u) \\ e_j((s+u)-t) & \text{if } s+u > t \wedge A_I^{\vec{e}}(t) \wedge e_j((s+u)-t)\downarrow \\ \text{undef} & \text{otherwise} \end{cases} \\ \text{undef} \end{cases}$$

The two cases are distinguished by $[\![\sigma']\!]\langle t, \vec{e}\rangle(s)$ being defined or not, i.e., the defined clause is chosen if $\neg(s\|t) \wedge A_I^{\vec{e}}(\min\{s,t\}) \wedge e_j(s \dot{-} t)\downarrow$, and the undefined clause otherwise.

Figure 6.4: The function $\delta_{X+\Sigma X} \circ [\![\sigma']\!]$

$$\lambda s.\begin{cases}\lambda u.\begin{cases}\sigma'\langle(t-s)-u, e'_1(s+u),\dots,e'_n(s+u)\rangle & \text{if } u<t-s\wedge A_I^{\vec{e}}(s+u)\\ e_j(0) & \text{if } u=t-s\wedge A_I^{\vec{e}}(s+u)\\ e_j(u-(t-s)) & \text{if } u>t-s\wedge A_I^{\vec{e}}(t)\wedge e_j(u-(t-s))\downarrow\\ \text{undef} & \text{otherwise}\end{cases}\\ \lambda u.\begin{cases}e_j(u) & \text{if } e_j(u)\downarrow\\ \text{undef} & \text{if } e_j(u)\uparrow\end{cases}\\ \lambda u.\begin{cases}e_j((s-t)+u) & \text{if } e_j((s-t)+u)\downarrow\\ \text{undef} & \text{if } e_j((s-t)+u)\uparrow\end{cases}\\ \text{undef}\end{cases}$$

The four outer cases are given as in the definition of $[\![\sigma']\!]$: from the top down, they are $s<t\wedge A_I^{\vec{e}}(s)$, $s=t\wedge A_I^{\vec{e}}(s)$, $s>t\wedge A_I^{\vec{e}}(t)\wedge e_j(s-t)\downarrow$, and finally, otherwise for the undefined clause.

Figure 6.5: The function $E(E\mathrm{inl}_X+[\![\sigma']\!]_X)\circ[\![\sigma']\!]_{EX}\circ\Sigma\delta_X$

when trying to define a distributive law whose bialgebras are exactly the biactions of a time domain. It seems that the property has some deeper connection with time domains and timed processes than is currently clear to us.

Remark 6.18

If the time-parameterised operator σ' is time prefixing from TeCCS, as described in Example 6.16, then the induced map $[\![\sigma']\!]$ is (almost) equal to the distributive law ℓ in (4.23) used in Section 4.4 for obtaining biactions (of an antichain-monotone time domain) as bialgebras: the only difference is the fact that $[\![\sigma']\!]$ can only have non-zero time parameters. Moreover, since

$$X+(\mathcal{T}^+\times X)\cong(\{0\}\times X)+(\mathcal{T}^+\times X)\cong(\{0\}+\mathcal{T}^+)\times X\cong\mathcal{T}\times X$$

we have that $[\![\sigma']\!]_X:\mathcal{T}^+\times EX\to E(\mathcal{T}\times X)$, while $\ell_X:\mathcal{T}\times EX\to E(\mathcal{T}\times X)$. Thus, we simply obtain

$$[\![\sigma']\!]_X=\ell_X|_{(\mathcal{T}^+\times EX)}$$

i.e., the semantics of σ' is obtained by appropriately restricting the distributive law ℓ. This match should not come as a surprise since biactions, as already remarked in Section 3.4, can alternatively be viewed as a TTS together with a syntactic (delay) operator which comes with a prescribed 'operational semantics,' i.e., natural axioms derived from axiom (3.28). $\diamondsuit$

Note that the restriction to binary operators for use with rule shape (C_j) is necessary. Using that shape with arities $n > 2$, and then using one rule for each $1 \leq j \leq n$, would not result in well-defined functions

$$(EX)^n \to E(X + \Sigma X)$$

This can be seen as follows. Consider the case $n = 3$, hence we have one rule of shape (B), and three rules of shape (C_j); this results in the following function:

$$\langle e_1, e_2, e_3 \rangle \;\mapsto\; h \stackrel{\mathsf{df}}{=} \lambda t. \begin{cases} \sigma\langle e_1(t), e_2(t), e_3(t)\rangle & \text{if } \bigwedge_{i=1}^{3} e_i(t)\downarrow \\ e_1(t) & \text{if } e_1(t)\downarrow \wedge e_2(t)\uparrow \wedge e_3(t)\uparrow \\ e_2(t) & \text{if } e_2(t)\downarrow \wedge e_1(t)\uparrow \wedge e_3(t)\uparrow \\ e_3(t) & \text{if } e_3(t)\downarrow \wedge e_1(t)\uparrow \wedge e_2(t)\uparrow \\ \text{undef} & \text{otherwise} \end{cases}$$

The problem with this 'definition' of a function $(EX)^n \to E(X + \Sigma X)$ is that it in fact is *not* a proper definition, h is *not* an evolution on $X + \Sigma X$ because it might not satisfy axiom (4.2) of evolutions: $[\![\sigma]\!]\langle e_1, e_2, e_3\rangle(t+u)\downarrow$ does, in general, not imply $[\![\sigma]\!]\langle e_1, e_2, e_3\rangle(t)\downarrow$. For $h(t)$ to be defined, either *all* arguments need to be defined, or *exactly one*. Yet, it is possible that precisely one, say e_1, is defined at $t+u$, while there is an additional one, say e_2, also defined at t (e_1, by (4.2), must be defined at t). Thus, $h(t+u)$ would be defined but $h(t)$ would be undefined, consequently violating (4.2). Note that, apart from well-definedness, all other calculations would actually go through for this 'definition,' in particular the δ-diagram commutes[1]. We do unfortunately not know how this can be salvaged in order to obtain a working definition for arities > 2.

[1] Since the δ-diagram is quite a restrictive condition which usually does the dirty work, i.e., excludes concrete operators, the fact that it commutes for a function like the above led us to the (now falsified) claim that such operators were also admissible for arities $n > 2$ in [Kic02b].

Furthermore, the condition $j \notin I$ is necessary for the rule shapes: otherwise, simply instantiating with $n = 1$, $I = \{1\}$, and $j = 1$, would result in a map $[\![\sigma]\!]$ of the appropriate type, yet which does not respect the structure of E: the δ-diagram does not commute in that case.

Note that the intuitive meaning of the δ-diagram is illustrated in this proof: applying the rules once to derive a $\overset{t+u}{\leadsto}$-transition (expressed in the right-down path in the diagram) must lead to the same process as applying the rules twice, first deriving a $\overset{t}{\leadsto}$-transition to an intermediate state, and then deriving a $\overset{u}{\leadsto}$-transition from there (the other path). In other words, the diagram represents a rule-, or derivation-based, version of time continuity.

Soundness now follows by combining the maps $[\![\sigma]\!]$ for all operators $\sigma \in \Sigma$:

Theorem 6.19

If the time rules of a language only use the admissible operators as described above, they induce a natural transformation $\Sigma E \Rightarrow E(\mathrm{Id} + \Sigma)$ respecting the structure of E. $\square$

In addition to the time rules of TeCCS, there are other operators in the literature that can be described using admissible operators. For example, consider the *time-out* operator $p \overset{t}{\rhd} q$ from [NS91], with time rules

$$\frac{p \overset{t'}{\leadsto} p',\ t' < t}{p \overset{t}{\rhd} q \overset{t'}{\leadsto} p' \overset{t-t'}{\rhd} q} \qquad \frac{p \overset{t}{\leadsto} p'}{p \overset{t}{\rhd} q \overset{t}{\leadsto} q} \qquad \frac{p \overset{t}{\leadsto} p',\ q \overset{t'}{\leadsto} q'}{p \overset{t}{\rhd} q \overset{t+t'}{\leadsto} q'}$$

Intuitively, $p \overset{t}{\rhd} q$ behaves like p *strictly* before time t; then at time t the control switches to q, simply discarding p: if p waits too long, viz., does not perform its intended task within t units of time, it gets *preempted* by the 'time-out handler' q. However, note that p really must wait until the point of preemption for the time-out to become effective: if p, for some reason, cannot idle at least for t units of time, q never gets activated.

The fact that $t' < t$ implies there is a $t'' \in \mathcal{T}^+$ such that $t + t'' = t$ (see Section 3.1), so the first rule can be rewritten as

$$\frac{p \overset{t'}{\leadsto} p'}{p \overset{t'+t''}{\rhd} q \overset{t'}{\leadsto} p' \overset{t''}{\rhd} q}$$

Then it fits the rule shapes as the binary case of shape (tA_I) with $I = \{1\}$; so do the other two rules, fitting shape ($\mathrm{tB}_{I,j}$) and shape ($\mathrm{tC}_{I,j}$), respectively, both with $I = \{1\}$

and $j = 2$. Since this is one of the allowed combinations for admissible operators in Definition 6.15, the time-out operator induces a map which respects the structure of E.

In contrast to that, the rules of the *start-delay* operator $\lfloor p \rfloor^t(q)$ from [NS91] do *not* fit any operator format:

$$\frac{p \overset{u}{\leadsto} p',\ u < t}{\lfloor p \rfloor^t q \overset{u}{\leadsto} \lfloor p' \rfloor^{t-u} q} \qquad \frac{(\forall s < t).\ p \overset{s}{\not\leadsto},\ u < t}{\lfloor p \rfloor^t q \overset{u}{\leadsto} \lfloor p \rfloor^{t-u} q} \qquad \frac{p \overset{t}{\leadsto} p'}{\lfloor p \rfloor^t q \overset{t}{\leadsto} q} \qquad \frac{p \overset{t}{\leadsto} p',\ q \overset{u}{\leadsto} q'}{\lfloor p \rfloor^t q \overset{t+u}{\leadsto} q}$$

Intuitively, $\lfloor p \rfloor^t q$ is very similar to $p \overset{t}{\triangleright} q$, if p can idle for at least t units of time the two processes even behave in exactly the same way (as expressed in the first, third, and fourth rule). Yet, if p cannot idle long enough, there is a subtle difference: where the time-out $p \overset{t}{\triangleright} q$ simply cannot idle either, the start-delay $\lfloor p \rfloor^t q$, as stated in the second rule, allows further progress, provided p cannot perform *any* time transition whatsoever. Consequently, p's potential for transitions is preserved for longer than it would have been present originally.

Intuitively, this is essentially the reason why the operator *violates* time continuity which is at the very heart of the TTS-based approach: for $\mathcal{T} = \mathbb{N}$ and any q, the rules allow the derivation $\lfloor (1).0 \rfloor^3 q \overset{1}{\leadsto} \lfloor 0 \rfloor^2 q \overset{1}{\leadsto} \lfloor 0 \rfloor^1 q$; yet $\lfloor (1).0 \rfloor^3 q \overset{2}{\not\leadsto}$, since $(1).0 \overset{2}{\not\leadsto}$ but $(1).0 \overset{1}{\leadsto} 0$, and so neither the first nor the second rule applies. Hence this particular operator is *not* compatible with our abstract model of timed processes, and its exclusion is actually *desirable* rather than problematic. On the level of abstract rules, the failure of time continuity is mirrored by the δ-diagram failing for the induced map $[\![\lfloor \_ \rfloor^t \_]\!]$ which consequently does *not* respect the structure of E, conceptually underpinning the decision not to include the operator.

Although the admissible operators exclude at least one undesirable operator, they do not provide a complete description of all possible well-behaved rules—actually, quite far from that. A simple counter-example is given by the following function

$$[\![\sigma(e)]\!] = \lambda t. \begin{cases} \sigma(e(0)) & \text{if } t = 0 \\ \sigma'(e(0)) & \text{if } t > 0 \end{cases} \tag{6.7}$$

or, spelled out as a schematic time rule, again with t ranging only over $\mathcal{T}^+$:

$$\frac{-}{\sigma(x) \overset{t}{\leadsto} \sigma'(x)}$$

It is trivial to check that $[\![\sigma]\!]$ is natural and respects the structure of E, yet, for $\sigma \neq \sigma'$, the rule does not fit any of the rule shapes since the top-most operator is changed in the conclusion of the rule (from σ to σ').

The reason for this restriction on admissible operators is the δ-diagram. Intuitively demanding that the rules satisfy continuity, changes in the topmost operator would cause problems, especially in the case of *cyclic dependencies* between operators. Assume, for instance, that there is a time transition from a term with σ as its topmost operator to a term with σ' on top, and vice versa; once allowing such instances, it seems almost impossible to *syntactically* guarantee continuity. Besides, all relevant operators from the literature fit the format anyway, so it seems general enough as it is.

Furthermore, after repeated attempts, it seems that more permissive rule shapes or admissible operators invariably invalidate well-definedness or the δ-diagram. A (to us) particularly convincing point in this direction is that we could not find an obvious n-ary version of the weak choice $\oplus$ from TeCCS, using shape (C_j) for any arity strictly greater than 2.

6.2.1.1 TeCCS revisited

We have already seen, in Example 6.16, that all operators of TeCCS can be modelled by admissible operators, weak choice $\oplus$ actually *being* one such admissible operator. Hence, we know that the rules of TeCCS induce abstract temporal rules $\rho : \Sigma E \Rightarrow E(\mathrm{Id} + \Sigma)$ respecting the structure of E. Consequently, ρ induces a distributive law $TE \Rightarrow ET$ of the term monad T over the evolution comonad E.

This distributive law is equivalent to a lifting $\tilde{T}$ of the monad T to the E-Coalgebras (cf. Section 2.4), which, instantiated with the trivial initial E-Coalgebra $! : 0 \dashrightarrow E0$ (given by initiality of the empty set 0), induces an E-Coalgebra $\tilde{T}_E(!) : T0 \to ET0$ on the set $T0$ of closed TeCCS terms by induction:

(6.8)
$$
\begin{tikzcd}
0 \arrow[r, "\eta_0"] \arrow[d, "!"'] & T0 \arrow[d, dashed, "\tilde{T}_E(!)"'] & & \Sigma T0 \arrow[ll, "\gamma_{T0}"'] \arrow[d, "\Sigma\tilde{T}_E(!)"] \\
E0 \arrow[r, "E\eta_0"'] & ET0 & E(T0 + \Sigma T0) \arrow[l, "E(\mathrm{id}_{T0}+\gamma_0)"] & \Sigma ET0 \arrow[l, "\rho_{T0}"]
\end{tikzcd}
$$

We can again concretely describe the resulting E-Coalgebra $\tilde{T}_E(!)$ on $T0$, by induction on the term structure of $\theta \in T0$:

1. $\theta = \mathrm{nil}$ or $\theta = \alpha.\theta'$: $\tilde{T}_E(!)(\theta) = \lambda t.\begin{cases} \theta & \text{if } t = 0 \\ \mathrm{undef} & \text{if } t > 0 \end{cases}$

2. $\theta = \delta.\theta'$: $\tilde{T}_B(!)(\theta) = \lambda t.\, \theta$

3. $\theta = (t).\theta'$: $\tilde{T}_B(!)(\theta) = \lambda s.\begin{cases} (t-s).\theta' & \text{if } s < t \\ \theta' & \text{if } s = t \\ \theta'' & \text{if } s > t \wedge \tilde{T}_E(!)(\theta')(s-t) = \theta'' \\ \mathrm{undef} & \text{otherwise} \end{cases}$

4. $\theta = \theta_1 + \theta_2$:

$$\tilde{T}_E(!)(\theta) = \lambda t.\begin{cases} \tilde{T}_E(!)(\theta_1)(t) + \tilde{T}_E(!)(\theta_2)(t) & \text{if } \tilde{T}_E(!)(\theta_1)(t)\downarrow \wedge \tilde{T}_E(!)(\theta_2)\downarrow \\ \mathrm{undef} & \text{if } \tilde{T}_E(!)(\theta_1)(t)\uparrow \vee \tilde{T}_E(!)(\theta_2)\uparrow \end{cases}$$

   The analogous definition applies to parallel composition.

5. $\theta = \theta_1 \oplus \theta_2$:

$$\tilde{T}_E(!)(\theta) = \begin{cases} \tilde{T}_E(!)(\theta_1)(t) \oplus \tilde{T}_E(!)(\theta_2)(t) & \text{if } \tilde{T}_E(!)(\theta_1)(t)\downarrow \wedge \tilde{T}_E(!)(\theta_2)(t)\downarrow \\ \tilde{T}_E(!)(\theta_1)(t) & \text{if } \tilde{T}_E(!)(\theta_1)(t)\downarrow \wedge \tilde{T}_E(!)(\theta_2)(t)\uparrow \\ \tilde{T}_E(!)(\theta_2)/t) & \text{if } \tilde{T}_E(!)(\theta_1)(t)\uparrow \wedge \tilde{T}_E(!)(\theta_2)(t)\downarrow \\ \mathrm{undef} & \text{if } \tilde{T}_E(!)(\theta_1)(t)\uparrow \wedge \tilde{T}_E(!)(\theta_2)(t)\uparrow \end{cases}$$

In the case $\mathcal{T} = \mathbb{N}$, $E_{\mathbb{N}}$**–Coalg** $\cong B_{\mathbb{N}}$**–coalg**, by Proposition 4.30, and so $\tilde{T}_E(!)$ corresponds to one unique $B_{\mathbb{N}}$-coalgebra $T0 \to B_{\mathbb{N}}T0 = 1 + T0$. We now prove that this $B_{\mathbb{N}}$-coalgebra is actually equal to $\tilde{T}_B(!)$, the ($B_{\mathbb{N}}$-coalgebra obtained as the) intended operational model of the single-step semantics of TeCCS from Section 6.1.1. For this, we are going to show that the image of $\tilde{T}_E(!)$ under the isomorphism of the categories of coalgebras is equal to $\tilde{T}_B(!)$; this suffices by the isomorphism property: applying the inverse map then necessarily maps $\tilde{T}_B(!)$ to $\tilde{T}_E(!)$. Let us therefore illustrate how this direction of the isomorphism works.

Since $E_{\mathbb{N}}$ is cofreely generated from $B_{\mathbb{N}}$, we know that $E_{\mathbb{N}}X \cong X \times B_{\mathbb{N}}E_{\mathbb{N}}X$, yielding the two projections

$$X \xleftarrow{\mathrm{fst}_X} E_{\mathbb{N}}X \xrightarrow{\mathrm{snd}_X} B_{\mathbb{N}}E_{\mathbb{N}}X$$

as presented in Section 2.4. Let $k : X \to E_{\mathbb{N}}X$ be an $E_{\mathbb{N}}$-Coalgebra. The isomorphism then sends k to the following composite

$$X \xrightarrow{k} E_{\mathbb{N}}X \xrightarrow{\mathrm{snd}_X} B_{\mathbb{N}}E_{\mathbb{N}}X \xrightarrow{B_{\mathbb{N}}\varepsilon_X} B_{\mathbb{N}}X \tag{6.9}$$

which we now have to decode in terms of $\overset{t}{\rightsquigarrow}$- and $\rightsquigarrow$-transitions.

By Proposition 4.24, the set $E_{\mathbb{N}}X$ is isomorphic to $X^+ + X^\omega$, and the $B_{\mathbb{N}}$-coalgebra snd_X is given by the *tail* function $\mathrm{tl}_X : X^+ + X^\omega \to 1 + X^+ + X^\omega$. Rephrased in terms of $\overset{t}{\rightsquigarrow}$-transitions, for $t \in \mathbb{N}$, the coalgebra $k : X \to E_{\mathbb{N}}X$ maps some $x \in X$ to a (finite and non-empty, or infinite) string over X, i.e., $k(x) = x_1 \dots x_n$ for $n \geq 1$, or $k(x) = x_1 x_2 \dots x_n \dots$; we obtain $k(x)(t) = x_t$ if the string $k(x)$ has at least length t, and $k(x)(t)\uparrow$ otherwise. However, all the tail function tl_X does is to remove the first element of the string $k(x)$, resulting in either the empty string ε, another non-empty but finite string, or an infinite stream. The counit ε_X, being equal to fst_X, simply returns the first element of a string in $E_{\mathbb{N}}X$. So the composite (6.9) returns, if possible, the targets of $\overset{1}{\rightsquigarrow}$-transitions, as defined in the TTS on X induced by k.

Therefore, in order to show that the two operational models correspond to each other under the isomorphism of coalgebras, we have to prove that

$$(\forall \theta, \theta' \in T0).\ (\theta \overset{1}{\rightsquigarrow} \theta') \Leftrightarrow (\theta \rightsquigarrow \theta')$$

or more precisely:

$$\tilde{T}_E(!)(\theta)(1)\uparrow \Leftrightarrow \tilde{T}_B(!)(\theta) = \star \tag{6.10}$$

$$\tilde{T}_E(!)(\theta)(1) = \theta' \Leftrightarrow \tilde{T}_B(!)(\theta) = \theta' \tag{6.11}$$

Proof: We will prove the two properties by induction on the structure of θ, although only for some of the operators.

1. $\theta = \mathrm{nil}$: By the definitions of $\tilde{T}_E(!)$ and $\tilde{T}_B(!)$, the two conditions follow because $\tilde{T}_E(!)(\mathrm{nil})(1)\uparrow$ and also $\tilde{T}_B(!)(\mathrm{nil}) = \star$, establishing (6.10), while (6.11) is vacuously satisfied.

2. $\theta = (t).\theta'$: By the definition of the syntax, $t > 0$, hence $t \geq 1$ and (6.10) never applies. Now assume that $t > 1$; then $\tilde{T}_E(!)(\theta)(1) = (t-1).\theta' = \tilde{T}_B(!)(\theta)$. In the case $t = 1$, we obtain $\tilde{T}_E(!)(\theta)(1) = \theta' = \tilde{T}_B(!)(\theta)$. Hence, (6.11) holds for time prefixing.

3. $\theta = \theta_1 \oplus \theta_2$: We only prove (6.11) in this case, and assume that $\tilde{T}_E(!)(\theta)(1) = \theta'$. By definition of $\tilde{T}_E(!)$, there are three distinct cases; we will only deal with the first one, the remaining two cases are analogous. So assume that both $\tilde{T}_E(!)(\theta_i)(1)\downarrow$, say $\tilde{T}_E(!)(\theta_i)(1) = \theta'_i$, and $\theta' = \theta'_1 \oplus \theta'_2$. We can thus apply the induction hypothesis for (6.11), and obtain that $\tilde{T}_B(!)(\theta_i) = \theta'_i \neq \star$. Consequently, by definition of $\tilde{T}_B(!)$, this means that $\tilde{T}_B(!)(\theta) = \tilde{T}_B(!)(\theta_1 \oplus \theta_2) = \theta'_1 \oplus \theta'_2 = \theta' = \tilde{T}_E(!)(\theta)(1)$, and we are done. The converse direction works in the analogous way.

The remaining cases for $+$ and $|$ are similar. □

Hence, we obtain:

Proposition 6.20
Up to the isomorphism $B_{\mathbb{N}}$**–coalg** $\cong E_{\mathbb{N}}$**–Coalg**, the intended operational model $\tilde{T}_B(!)$ is the same as the original model $\tilde{T}_E(!)$ induced by the time rules from [MT90], i.e., the operational semantics of single-step TeCCS is essentially the same as for the original language. □

6.2.1.2 Refining the Rule Shapes

Coming back to the rule shapes, even though arguably expressive enough, the admissible operators are by no means as general as they could be. As an illustrative example, consider the following ‘speed-halving’ operator[2]:

$$\frac{p \stackrel{t}{\rightsquigarrow} p'}{\sigma(p) \stackrel{t+t}{\rightsquigarrow} \sigma(p')} \tag{6.12}$$

[2]Suggested to us by an anonymous referee.

This operator can be described by the function

$$\llbracket\sigma\rrbracket : \Sigma EX \to E(X + \Sigma X)$$
$$e \mapsto \lambda t. \begin{cases} \sigma(e(t)) & \text{if } \exists u.\ u+u=t \wedge e(u)\downarrow \\ \text{undef} & \text{otherwise} \end{cases}$$

which, in general, is not well-defined: for $\mathcal{T} = \mathbb{N}$, $\llbracket\sigma\rrbracket$ potentially allows the derivation of a $\overset{2}{\rightsquigarrow}$-transition (in case $e(1)\downarrow$), yet never of a $\overset{1}{\rightsquigarrow}$-transition (there is no $u \in \mathbb{N}$ such that $u+u=1$) and therefore, axiom (4.2) of evolutions would not hold for $\llbracket\sigma\rrbracket(e)$.

However, when considered over the *specific* time domain $\mathbb{R}_{\geq 0}$, $\llbracket\sigma\rrbracket$ is natural, it fits the type (5.18), and it respects the structure of E. Thus, for the time domain $\mathbb{R}_{\geq 0}$ and the specific *time transformation* $t \mapsto t+t$ on it, (6.12) results in a well-behaved operator. This can be generalised as sketched in the following tentative development, allowing rule shapes parameterised by a time domain $\mathcal{T}$ and 'well-behaved' transformations, although we are going to turn the rule (6.12) 'upside down' in the following presentation.

Definition 6.21

Given two monoids $M = \langle M, +, 0\rangle$ and $M' = \langle M', +', 0'\rangle$, a *partial monoid homomorphism* from M to M' is a partial function $f : M \rightharpoonup M'$ which satisfies the following two Kleene equalities

$$f(0) \simeq 0' \tag{6.13}$$

$$f(m+m') \simeq (fm) +' (fm') \tag{6.14}$$

Note that in the first equation, we could also use $=$. ■

In what follows, it is not possible to use a weaker notion of partial homomorphism, e.g., obtained by replacing $\simeq$ with $\sqsubseteq$, because the following calculations are aimed at establishing *equalities* between partial functions, and as such need both halves of $\simeq$.

Given an enumerated set $\mathcal{V}$ of variables, a time domain $\mathcal{T}$, and a partial monoid homomorphism $f : \mathcal{T} \rightharpoonup \mathcal{T}^n$, we can allow the following rule shape, writing t_i instead of $\pi_i(f(t))$, provided that $f(t)\downarrow$:

$$\frac{v_1 \overset{t_1}{\rightsquigarrow} v_{n+1} \ \cdots \ v_n \overset{t_n}{\rightsquigarrow} v_{2n}}{\sigma\langle\vec{v}\rangle \overset{t}{\rightsquigarrow} \sigma\langle\vec{v}'\rangle}\ f(t)\downarrow \tag{6.15}$$

again writing $\vec{v}' = \langle v_{n+1}, \ldots, v_{2n}\rangle$. Compared to (6.12), we apply f in the *premises* rather than the conclusion of the rule shape. With this, we obtain:

Proposition 6.22
If the semantics of an operator symbol σ are defined by a single operational rule (6.15), for a time domain $\mathcal{T}$ and a partial monoid homomorphism $f : \mathcal{T} \rightharpoonup \mathcal{T}^n$, we obtain a map, for each set X,

$$[\![\sigma]\!]_X : (EX)^n \to E(X + \Sigma X)$$

which is natural in X and also respects the structure of E.

Proof: Using similar reasoning as above in the proof of Proposition 6.17, we can translate the rule (6.15) into the function

$$\begin{aligned}
&[\![\sigma]\!]_X : (EX)^n \to E(X + \Sigma X) \\
&\qquad\qquad \langle\vec{e}\rangle \mapsto \lambda t.\ \begin{cases} \sigma\langle e_1(t_1), \ldots, e_n(t_n)\rangle & \text{if } f(t)\downarrow \wedge \bigwedge_{i=1}^n e_i(t_i)\downarrow \\ \text{undef} & \text{if } f(t)\uparrow \vee \bigvee_{i=1}^n e_i(t_i)\uparrow \end{cases}
\end{aligned}$$

Since $f(0) = \vec{0} = \langle 0, \ldots, 0\rangle$, certainly $[\![\sigma]\!]\langle\vec{e}\rangle(0)\downarrow$, i.e., axiom (4.1) of evolutions holds for $[\![\sigma]\!]_X$. Assuming $[\![\sigma]\!]_X\langle\vec{e}\rangle(t+u)\downarrow$ means that $f(t+u)\downarrow$ and $e_i(t+u)\downarrow$, for all $1 \leq i \leq n$. As f is a partial homomorphism, $f(t+u) \simeq f(t) + f(u)$, and so in particular $f(t)\downarrow$; additionally, since the e_i are evolutions and consequently satisfy (4.2), also $e_i(t)\downarrow$ for all $1 \leq i \leq n$, i.e., $[\![\sigma]\!]\langle\vec{e}\rangle(t)\downarrow$, showing that $[\![\sigma]\!]_X$ is indeed well-defined.

For the ε-diagram, we have already stated that $f(0) = \langle 0\rangle$, and so we obtain

$$[\![\sigma]\!]\langle\vec{e}\rangle(0) = \sigma\langle e_1(0), \ldots, e_n(0)\rangle$$

which is exactly the meaning of the ε-diagram, cf. (6.6).

For the δ-diagram, the first path around the diagram, $\delta_{X+\Sigma X} \circ [\![\sigma]\!]$, results in the following function:

$$\lambda t.\ \begin{cases} \lambda u.\ \begin{cases} \sigma\langle e_1(t_1+u_1), \ldots, e_n(t_n+u_n)\rangle & \text{if } f(t+u)\downarrow \wedge \bigwedge_{i=1}^n e_i(t_i+u_i)\downarrow \\ \text{undef} & \text{if } f(t+u)\uparrow \vee \bigvee_{i=1}^n e_i(t_i+u_i)\uparrow \end{cases} \\ \text{undef} \end{cases}$$

where the defined clause has the condition $f(t)\downarrow \wedge \bigwedge_{i=1}^{n} e_i(t_i)\downarrow$, and the undefined one $f(t)\uparrow \vee \bigvee_{i=1}^{n} e_i(t_i)\uparrow$. Note that these computations use the homomorphism property of f, in particular additivity (6.14). The other path, $E[E\mathrm{inl}, [\![\sigma]\!]_X] \circ [\![\sigma]\!]_{EX} \circ (\delta_X)^n$, results in exactly the same function (as is routinely checked), concluding the proof that (6.15) constitutes an admissible operator. □

The question is now what we gain by using these rule shapes which are parametric over the considered time domain. First of all, using these partial homomorphisms, we obtain the following simplification of the rule shapes: the cases of no rule, one rule of shape (A), and one rule of shape (B) are all subsumed by (6.15), for appropriate choices of $f : \mathcal{T} \rightharpoonup \mathcal{T}^n$—*regardless* of the arity $n \in \mathbb{N}$:

1. Considering the function

$$\mathrm{undef}_0 : \mathcal{T} \rightharpoonup \mathcal{T}^n,\ t \ \mapsto \begin{cases} \vec{0} & \text{if } t = 0 \\ \mathrm{undef} & \text{if } t > 0 \end{cases}$$

   we obtain the same effect as having no rules.

2. Considering the function $\mathrm{const}_0 : \mathcal{T} \to \mathcal{T}^n,\ t \ \mapsto \ \vec{0}$, one obtains the same function $[\![\sigma]\!]$ as when using one rule of shape (A).

3. Finally, by considering the function $\langle \mathrm{id}_{\mathcal{T}}, \ldots, \mathrm{id}_{\mathcal{T}} \rangle : \mathcal{T} \to \mathcal{T}^n, t \ \mapsto \ \vec{t} = \langle t, \ldots, t \rangle$, the same function $[\![\sigma]\!]_X$ is induced as by one rule of shape (B).

Note the suspicious absence of the rule shape (C_j), and time-parameterised operators: there does not seem to be a reasonable way to account for those using such homomorphisms. Since shape (C_j), to our current knowledge, can only appear in connection with binary operators, this should not come as a surprise since homomorphisms are in some sense *uniform*, i.e., independent of the arity n.

Further examples of partial homomorphisms are all functions of the form $t \mapsto c * t$, for $c \in \mathbb{N}$ (where $c * t$ is an abbreviation for the c-fold sum $t + \cdots + t$). In the special case $\mathcal{T} = \mathbb{N}$, these are, moreover, the *only* such homomorphisms $f : \mathbb{N} \rightharpoonup \mathbb{N}$ which are different from undef_0; this can easily seen by considering the value $f(1)$: if it is undefined, by additivity (6.14), f cannot be defined at *any* $t \neq 0$, while if $f(1)\downarrow$ with value $f(1) = c$, it follows that $f = c * \mathrm{id}_{\mathbb{N}}$.

For $\mathcal{T} = \mathbb{R}_{\geq 0}$, the situation is a lot more complex: certainly all the (total!) functions $c * \mathrm{id}_{\mathbb{R}_{\geq 0}}$, now for $c \in \mathbb{R}_{\geq 0}$, are partial homomorphisms $\mathbb{R}_{\geq 0} \rightharpoonup \mathbb{R}_{\geq 0}$, however, it would seem surprising if the homomorphism property is so strong as to only allow total functions. One way to obtain a 'really partial' partial homomorphism is to find a non-empty, proper subset $S \subset \mathbb{R}_{\geq 0}$ such that both S and $\mathbb{R}_{\geq 0} \setminus S$ are closed under addition, and then simply consider the injection of such a subset into $\mathbb{R}_{\geq 0}$; as it happens, using the axiom of choice, one can construct such a set, showing indeed that the simple characterisation from above for the naturals does not carry over to $\mathbb{R}_{\geq 0}$.

Compared to the original attempt (6.12), as sketched in [Kic02b], using homomorphisms 'in the other direction' as shown in (6.15), gives us some extra power: previously, the only acceptable transformation on $\mathbb{N}$ was $\mathrm{id}_{\mathbb{N}}$ while now, at least all constant speed-up operators are possible, as witnessed by the characterisation of partial homomorphisms on $\mathbb{N}$. All in all, at least a little more flexibility is gained, and some very natural operators now fit within the framework of admissible operators, allowing such partial homomorphisms. Even so, a lot of open questions remain, in particular whether there is a way to include time-parameterised operators.

6.2.2 The Complete Characterisation

As an application of CSOS rules, this section presents a syntactic characterisation of CSOS rules for the evolution comonad E. The format is based on the notion of a *meta rule*, which will serve as a convenient notational shorthand for infinite sets of infinitary rules. These meta rules will contain evolutions in places where variables are placed in conventional rules, hence they are a somewhat mixed notation in between pure syntax and the categorical operational semantics for timed processes presented in terms of abstract rules in Chapter 5. Even so, the format will still consist of infinitely many such meta rules to completely capture natural transformations $\rho : \Sigma E \Rightarrow ET$ for a signature Σ with freely generated term monad T.

In the following, fix a non-trivial time domain $\mathcal{T}$—hence $\mathcal{T}$ has infinite cardinality, see Proposition 3.17; note in particular that antichain monotonicity is *not* assumed.

Furthermore, let $\mathcal{V}$ be a set of variables such that $|\mathcal{V}| = |\mathcal{T}|$, and fix *n-ary enumerations* (without repetitions) of $\mathcal{V}$ for each $n \in \mathbb{N}$; by this, we mean disjoint subsets $\mathcal{V}_i = \{v_i^t \mid t \in \mathcal{T}\} \subseteq \mathcal{V}$ for each $1 \leq i \leq n$ such that $|\mathcal{V}_i| = |\mathcal{T}|$ and that the maps $t \mapsto v_i^t$ are all injective. If $\mathcal{V}$ is a set with such n-ary enumerations, we call $\mathcal{V}$ an *n-enumerated set*. These n-ary enumerations will play a similar rôle as the enumerations before have played, viz., determine which variables will be used in certain places. In the following, we need several technical definitions:

Definition 6.23

Let X be a set, and $e \in EX$ be an evolution on X. Since e is a special partial function $\mathcal{T} \rightharpoonup X$, its *domain* and *codomain* are, respectively,

$$\begin{aligned} \mathrm{dom}(e) &\stackrel{\text{df}}{=} \{t \in \mathcal{T} \mid e(t)\downarrow\} \subseteq \mathcal{T} \\ \mathrm{cod}(e) &\stackrel{\text{df}}{=} e(\mathcal{T}) = \{x \in X \mid (\exists t \in \mathcal{T}).\, x = e(t)\} \subseteq X \end{aligned}$$

whereas for $\vec{e} = \langle e_1, \ldots, e_n \rangle \in (EX)^n$, define domain and *range* of $\vec{e}$ by

$$\begin{aligned} \mathrm{dom}(\vec{e}) &\stackrel{\text{df}}{=} \langle \mathrm{dom}(e_1), \ldots, \mathrm{dom}(e_n) \rangle \\ \mathrm{rng}(\vec{e}) &\stackrel{\text{df}}{=} \bigcup_{i=1}^{n} \mathrm{cod}(e_i) \end{aligned}$$

If $e \in ETX$, i.e., an evolution on terms, the *variables* $\mathrm{vars}(e)$ of e are all the variables occurring in the terms $\theta \in \mathrm{cod}(e) \subseteq TX$ (a potentially infinite set!):

$$\mathrm{vars}(e) = \bigcup_{\theta \in \mathrm{cod}(e)} \mathrm{vars}(\theta)$$

For tuples $\vec{e} \in (ETX)^n$, define

$$\mathrm{vars}(\vec{e}) \stackrel{\text{df}}{=} \bigcup_{i=1}^{n} \mathrm{vars}(e_i)$$

Finally, for $n \in \mathbb{N}$, an *n-ary domain* is a tuple $\vec{d} \in \mathrm{spec}(\mathcal{T})^n$. ■

These notions are all pretty much self-explanatory, and simply will be needed to be able to introduce more complicated notions based on them. For $e \in ETX$, it is equivalent to define

$$\mathrm{vars}(e) = \bigcup_{t \in \mathrm{dom}(e)} \mathrm{vars}(e(t))$$

since each $\theta \in \mathrm{cod}(e)$ is of the form $e(t)$ for some $t \in \mathrm{dom}(e) \subseteq \mathcal{T}$. Note that, for $\vec{e} \in (EX)^n$, it is possible to restrict each e_i to the range $\mathrm{rng}(\vec{e})$, i.e., the function $e_i \restriction^{\mathrm{rng}(\vec{e})}$ is well-defined. Next, we need to generalise the conditions on GSOS rules that certain variables have to be distinct:

Definition 6.24
Call a tuple $\vec{e} \in (EX)^n$ of evolutions *generic* if all the e_i have disjoint codomains, and each e_i is injective. Furthermore, for $n \in \mathbb{N}$ and an n-ary domain $\vec{d} \in \mathrm{spec}(\mathcal{T})^n$, the n-ary *canonical* tuple with domain $\vec{d}$ is given by the n evolutions ε_i, $1 \le i \le n$, defined by

$$(\forall t \in d_i).\ \varepsilon_i(t) = v_i^t \in \mathcal{V}_i \subseteq \mathcal{V}$$

In the following, $\vec{\varepsilon}$ will denote a canonical tuple of some n-ary domain $\vec{d} \in \mathrm{spec}(\mathcal{T})^n$. For $\vec{e} \in (EX)^n$ with $\vec{d} = \mathrm{dom}(\vec{e})$, the *corresponding canonical tuple* $\vec{\varepsilon}$ is a canonical n-tuple $\vec{\varepsilon} \in (E\mathcal{V})^n$ with domain $\vec{d}$. ■

Note that, relative to an n-enumerated set $\mathcal{V}$, and a specific n-ary domain $\vec{d}$, there is a *unique* canonical tuple with domain $\vec{d}$. Since the n-ary enumerations on $\mathcal{V}$ do not contain repetitions, canonical tuples are trivially generic. Moreover, given a tuple $\vec{e} \in (EX)^n$, its corresponding canonical tuple $\vec{\varepsilon}$ is also unique with that property: for any $t \in \mathrm{dom}(e_i)$, the value of $\varepsilon_i(t)$ must necessarily be $v_i^t \in \mathcal{V}_i$. The tuple $\vec{\varepsilon}$ is some kind of 'normal form,' which shall later be used to derive meta rules from a natural transformation.

Generic tuples are completely unassuming as far as the identities of successor processes are concerned. This, amongst other things, is illustrated by the lemma below: each element of the range of a generic tuple can be uniquely 'traced back' to an evolution in the tuple. Intuitively, generic tuples play the same rôle in the rule format as the condition on GSOS rules that all variables must be distinct, i.e., the rules must treat argument processes 'anonymously': although a rule can be instantiated with the same processes in different places, it cannot *demand* such identifications. Canonical tuples are thus simply generic tuples which, by utilising the n-ary enumeration on $\mathcal{V}$, completely determine which variable is to be used where while still remaining 'unassuming' in the above sense.

Lemma 6.25

1. If $\vec{e} \in (EX)^n$ is a generic tuple and $x \in \text{rng}(\vec{e})$ then there exist unique $1 \leq i \leq n$ and $t \in \mathcal{T}$ such that $x = e_i(t)$.

2. Each tuple $\vec{e} \in (EX)^n$, with corresponding canonical tuple $\vec{\varepsilon}$, induces a unique, and moreover surjective, map $\varphi_{\vec{e}} : \text{rng}(\vec{\varepsilon}) \to \text{rng}(\vec{e}) \subseteq X$ such that

$$E\varphi_{\vec{e}}(\varepsilon_i \mid^{\text{rng}(\vec{\varepsilon})}) = \varphi_{\vec{e}} \circ \varepsilon_i \mid^{\text{rng}(\vec{\varepsilon})} = e_i \tag{6.16}$$

Proof:

1. By assumption, the range of $\vec{e}$ is a disjoint union of the codomains of the e_i, hence each $x \in \text{rng}(\vec{e})$ must belong to the codomain of a unique e_i; since this e_i is additionally injective, there exists the desired unique $t \in \text{dom}(e_i) \subseteq \mathcal{T}$ such that $e_i(t) = x$.

2. For $v_i^t \in \text{rng}(\vec{\varepsilon})$, define the map $\varphi_{\vec{e}}$ by $\varphi_{\vec{e}}(v_i^t) \stackrel{\text{df}}{=} e_i(t)$. Since $\text{dom}(\vec{\varepsilon}) = \text{dom}(\vec{e})$, it follows that $\varphi_{\vec{e}}$ is surjective and also satisfies the desired property (6.16). Now suppose that $f : \text{rng}(\vec{\varepsilon}) \to \text{rng}(\vec{e})$ satisfies (6.16), and let $1 \leq i \leq n$ and $t \in \text{dom}(\varepsilon_i) = \text{dom}(e_i)$. By (6.16), we then know that $e_i(t) = f(\varepsilon_i(t)) = f(v_i^t)$, by definition of canonical tuples, and so f must be identical to $\varphi_{\vec{e}}$, proving the uniqueness claim. □

Later on, $\varphi_{\vec{e}}$ will sometimes be used as a (total) function of type $\mathcal{V} \to X$; this is achieved by arbitrarily assigning values in X to variables in $\mathcal{V} \setminus \text{rng}(\vec{\varepsilon})$; we can also use $\varphi_{\vec{e}}$ as a *partial* function $\mathcal{V} \rightharpoonup X$. Because the function $\varphi_{\vec{e}}$ is uniquely determined on $\text{rng}(\vec{\varepsilon})$, which is really the essential part of $\vec{\varepsilon}$, we call it *essentially unique*. Next, we introduce the main ingredient of the complete characterisation of CSOS rules for timed processes:

Definition 6.26

Let Σ be a signature, $\sigma \in \Sigma$ be an n-ary function symbol, $\vec{e} \in (EX)^n$, and $\vartheta \in ETX$. Then an expression of the form

$$\sigma\langle e_1, \ldots, e_n \rangle \Longrightarrow \vartheta \tag{6.17}$$

is a *meta rule for* σ. In the following, we write $\theta_t \stackrel{\text{df}}{=} \vartheta(t) \in TX$ for $t \in \mathrm{dom}(\vartheta)$. The domain of a meta rule such as (6.17) is defined to be $\mathrm{dom}(\vec{e})$; the meta rule is generic (canonical) if $\vec{e}$ is a generic (resp. canonical) tuple of evolutions. ■

Each meta rule (6.17) is an abbreviation of the (infinite) set of infinitary time rules, ranging over $t \in \mathrm{dom}(\vartheta)$, of the form

$$(6.18) \quad \frac{\{e_i(0) \stackrel{t_i}{\rightsquigarrow} e_i(t_i) \,|\, t_i \in \mathcal{T} \wedge e_i(t_i)\downarrow\}_{1\leq i\leq n} \quad \{e_i(0) \stackrel{t_i}{\not\rightsquigarrow} \,|\, t_i \in \mathcal{T} \wedge e_i(t_i)\uparrow\}_{1\leq i\leq n}}{\sigma\langle e_1(0),\ldots,e_n(0)\rangle \stackrel{t}{\rightsquigarrow} \theta_t},$$

occasionally abbreviated as $\sigma(\vec{e}) \stackrel{t}{\rightsquigarrow} \theta_t$, blurring the distinction between rule and rule conclusions. Note that each meta rule contains a *complete* (or *global*) description of the arguments' behaviour, not just local tests for the presence or absence of specific time transitions as, e.g., in the schematic format. This is in line with the interpretation of behaviour comonads as modelling *global behaviour*. The following development will be based entirely on meta rules to make it more concise; of course, all of it could also be carried out using standard time rules, via the correspondence (6.18).

It is worthwhile noting that, in a meta rule (6.17), ϑ has to be an evolution, and so in particular has to be defined at 0, there is always at least a rule (6.18) with the conclusion

$$\sigma\langle e_1(0),\ldots,e_n(0)\rangle \stackrel{0}{\rightsquigarrow} \theta_0$$

no matter whether is ϑ defined at any other points $t \in \mathcal{T}^+$.

A meta rule $\sigma\langle\vec{e}\rangle \Longrightarrow \vartheta$ is a somewhat mixed notation, halfway between syntax and semantics: it uses elements of the signature, $\sigma \in \Sigma$, together with evolutions, $\vec{e}$ and ϑ, which stem from the realm of the semantics of timed processes. In doing so, we hope to obtain a suggestive and (reasonably) concise notation for describing what is essentially a huge set of time rules like (6.18): a time rule potentially for each $t \in \mathcal{T}$ is described by only a single such meta rule.

However, it should be clear already at this point that we are going to need a lot of meta rules to describe CSOS rules for timed processes, in other words: an infinite set of (finite representations of) infinite sets of infinitary rules. We want to stress this in order to clarify the point that the rule format obtained in this section should be treated as an additional illustration of the complexities involved in the operational semantics of timed processes, rather than as a proper contender of formats like GSOS, tyft/tyxt, or even our dsl-format for discrete time, all of which are finite objects using only finitary operational rules.

Definition 6.27

A meta rule $\sigma\langle\vec{e}\rangle \Longrightarrow \vartheta$ is a *GSOS meta rule* if it is canonical and if $\mathrm{vars}(\vartheta) \subseteq \mathrm{rng}(\vec{e})$. A set of meta rules over a signature Σ is *complete* (*deterministic*) if, for each n-ary operator symbol $\sigma \in \Sigma$ and each n-ary domain $\vec{d}$, there is at least (resp. at most) one meta rule for σ with domain $\vec{d}$. A set of deterministic, complete GSOS meta rules is called *admissible*.

It follows immediately from the definitions that an admissible set of meta rules contains exactly one meta rule for each operator symbol σ and each appropriate domain. The terminology 'GSOS meta rule' is justified since in this case the induced time transitions (6.18) are indeed (infinitary) GSOS rules: all the e_i have disjoint ranges, i.e., all variables in the premises are distinct—in particular the $e_i(0)$—and since $\mathrm{vars}(\vartheta) \subseteq \mathrm{rng}(\vec{e})$, each variable occurring in some θ_t must occur somewhere in the premises.

Note that, strictly speaking, it is not necessary that GSOS meta rules are canonical: the induced rules would still be GSOS rules if we would use generic tuples such that $\mathrm{vars}(\vartheta) \subseteq \mathrm{rng}(\vec{e})$. However, the enumeration, together with the deterministic behaviour modelled by evolutions, shall allow us to derive an exact one-to-one correspondence, rather than up to provability as in [TP97]. The following results will show that admissible sets of meta rules are the correct characterisation of natural transformations $\Sigma E \Rightarrow ET$. First, we need some definitions to do with substitutions:

Definition 6.28

Let Σ be a signature and let $\theta \in TX$ be a term over Σ and some set X; let $f : X \rightharpoonup Y$ be partial function such that $\mathrm{vars}(\theta) \subseteq \mathrm{dom}(f)$. It is then possible to define the *simultaneous substitution* of $f(x_i) \in Y$ for $x_i \in \mathrm{vars}(\theta)$ in θ, and denote it by $\theta[f]$, or explicitly by $\theta[f(x_i)/x_i]$. Extending this notion to evolutions $\vartheta \in ETX$ on terms, subject to the condition $\mathrm{vars}(\vartheta) \subseteq \mathrm{dom}(f)$ for $f : X \rightharpoonup Y$, we write $\vartheta[f]$ or $\vartheta[f(x_i)/x_i]$ to denote the evolution in ETY whose value at each time $t \in \mathrm{dom}(\vartheta) \subseteq \mathcal{T}$ is the term $\theta_t[f] = \theta_t[f(x_i)/x_i]$, for all variables $x_i \in \mathrm{vars}(\theta_t)$. ■

Proposition 6.29
Each admissible set R of meta rules induces a natural transformation $[\![R]\!] : \Sigma E \Rightarrow ET$.

Proof: Based on the information in R, we need to define a map

$$[\![R]\!]_X : \Sigma EX \to ETX$$

for any set X. By the definition of admissible sets of meta rules, we will start with one particular such component and then use this as the 'canonical blueprint' for all other components, and obviously, the component to start with is $[\![R]\!]_{\mathcal{V}}$. As we will see, once $[\![R]\!]_{\mathcal{V}}$ is defined for canonical tuples only, the other values can then be inferred by applying suitable substitutions. Therefore, let $\sigma \in \Sigma$ be an n-ary operator symbol, and let $\vec{\varepsilon}$ be a canonical n-tuple of some n-ary domain. Since R is admissible, there must be meta rule contained in R which is of the form $\sigma\langle\vec{\varepsilon}\rangle \Longrightarrow \vartheta$, for some $\vartheta \in ET\mathcal{V}$. Then simply define $[\![R]\!]_{\mathcal{V}}(\sigma\langle\vec{\varepsilon}\rangle) = \vartheta$. This defines $[\![R]\!]_{\mathcal{V}}$ for all canonical argument tuples.

Let now X be an arbitrary set, and $\vec{e} \in (EX)^n$ be an arbitrary n-tuple of evolutions on X, and let $\vec{\varepsilon}$ be its corresponding canonical tuple. By the previous step, we know that we have already define $[\![R]\!]_{\mathcal{V}}(\sigma\langle\vec{\varepsilon}\rangle) = \vartheta$ for some $\vartheta \in ET\mathcal{V}$ with R containing the appropriate meta rule. By Lemma 6.25(2), there is a map $\varphi \stackrel{\text{df}}{=} \varphi_{\vec{e}} : \mathcal{V} \rightharpoonup X$ which satisfies the property (6.16), i.e., $E\varphi(\varepsilon_i) = e_i$; consequently, $(E\varphi)^n\langle\vec{\varepsilon}\rangle = \vec{e}$. Using this, define

$$[\![R]\!]_X(\sigma\langle\vec{e}\rangle) \stackrel{\text{df}}{=} \vartheta[\varphi] = ET\varphi(\vartheta) = \vartheta[e_i(t)/v_i^t]$$

Since R is an admissible set of meta rules, $\text{vars}(\vartheta) \subseteq \text{rng}(\vec{\varepsilon}) = \text{dom}(\varphi)$; consequently, the substitution $\vartheta[\varphi]$ makes sense and, by the definition of φ, is equal to the more concrete description $\vartheta[e_i(t)/v_i^t]$. So this is a good definition.

This assignment then completely defines $[\![R]\!]_X$, for any argument, also the so far missing values of $[\![R]\!]_{\mathcal{V}}$ are 'filled in.' We now have to show that this induces a natural transformation of the desired type. Let therefore X, Y be two arbitrary sets, and let $f : X \to Y$ be a function. We have to show that the following diagram commutes:

$$\begin{tikzcd}
\Sigma EX \arrow[r, "{[\![R]\!]_X}"] \arrow[d, "\Sigma Ef"'] & ETX \arrow[d, "ETf"] \\
\Sigma EY \arrow[r, "{[\![R]\!]_Y}"'] & ETY
\end{tikzcd}$$

Let therefore $\sigma \in \Sigma$ be an n-ary operator symbol, and $\vec{e} \in (EX)^n$ an n-tuple of evolutions on X, abbreviating $(Ef)^n\langle\vec{e}\rangle = \langle f \circ e_1, \ldots, f \circ e_n\rangle$ as $\overrightarrow{fe}$. By definition of the evolution comonad, we know that $\mathrm{dom}(\vec{e}) = \mathrm{dom}(\overrightarrow{fe})$. Let $\vec{\varepsilon} \in (E\mathcal{V})^n$ be their unique corresponding canonical tuple, and $\varphi_e : \mathcal{V} \rightharpoonup X$ be the essentially unique partial function such that (6.16) holds for $\vec{e}$, i.e., $\varphi_e(v_i^t) = e_i(t)$ for all $t \in \mathrm{dom}(e_i)$; we denote the analogously obtained (also essentially unique) function for $\overrightarrow{fe}$ by φ_f, which then satisfies $\varphi_f(v_i^t) = (f \circ e_i)(t) = f(e_i(t))$ for all $t \in \mathrm{dom}(e_i)$. We then have the following situation:

(6.19)
$$\begin{tikzcd}
\Sigma E\mathcal{V} \arrow[rrr, "[\![R]\!]_{\mathcal{V}}"] \arrow[dr, "\Sigma E\varphi_e"] \arrow[ddr, "\Sigma E\varphi_f"'] & & & ET\mathcal{V} \arrow[dl, "ET\varphi_e"'] \arrow[ddl, "ET\varphi_f"] \\
& \Sigma EX \arrow[r, "[\![R]\!]_X"] \arrow[d, "\Sigma Ef"] & ETX \arrow[d, "ETf"'] & \\
& \Sigma EY \arrow[r, "[\![R]\!]_Y"] & ETY &
\end{tikzcd}$$

where we know that in the left triangle, we get

$$\sigma\langle\vec{\varepsilon}\rangle \overset{\Sigma E\varphi_e}{\longmapsto} \sigma\langle\vec{e}\rangle$$
$$\sigma\langle\vec{\varepsilon}\rangle \overset{\Sigma E\varphi_f}{\longmapsto} \sigma\langle\overrightarrow{fe}\rangle = \Sigma Ef(\sigma\langle\vec{e}\rangle)$$

i.e., the triangle commutes for the specific element $\sigma\langle\vec{\varepsilon}\rangle \in \Sigma E\mathcal{V}$. Similarly, if it is the case that $[\![R]\!]_{\mathcal{V}}(\sigma\langle\vec{\varepsilon}\rangle) = \vartheta \in ET\mathcal{V}$, then, by definition of $[\![R]\!]_X$ and $[\![R]\!]_Y$, we obtain

$$[\![R]\!]_X(\sigma\langle\vec{e}\rangle) = \vartheta[\varphi_e]$$
$$[\![R]\!]_Y(\sigma\langle\overrightarrow{fe}\rangle) = \vartheta[\varphi_f]$$

It now remains to show that $\vartheta[\varphi_f] = ETf(\vartheta[\varphi_e])$. One thing that is certain is that the term structure of $\vartheta[\varphi_e]$ and $\vartheta[\varphi_f]$ is the same: by that, we mean the tree structure in terms of operator symbols, and the claim holds because both are obtained by applying two different substitutions to the very same evolution $\vartheta \in ET\mathcal{V}$. The only question is what happens to the variables which are at the leaves of the term tree of ϑ. For this, let $v_i^t \in \mathrm{vars}(\vartheta)$ which, by definition, means that there exists some $t \in \mathrm{dom}(\vartheta)$ such that $v_i^t \in \mathrm{vars}(\theta_t)$, for $\theta_t = \vartheta(t)$. We have that $\vartheta[\varphi_e](t) = \theta_t[\varphi_e]$, and so, by definition of φ_e, v_i^t gets renamed to $e_i(t)$ which, be ETf gets further renamed to

$f(e_i(t))$. Finally, $\vartheta[\varphi_f](t) = \theta_t[\varphi_f]$, and by definition, we obtain that v_i^t gets renamed to $(f \circ e_i)(t) = f(e_i(t))$, i.e., to the same element of Y as on the other possible path. Thus, the naturality square commutes, and we are done. $\square$

Also the converse of Proposition 6.29 holds:

Proposition 6.30
Let $\rho : \Sigma E \Rightarrow ET$ be a natural transformation. Then ρ induces an admissible set $\langle\!\langle\rho\rangle\!\rangle$ of meta rules.

Proof: Let $\sigma \in \Sigma$ be an n-ary operator symbol, and let $\vec{\varepsilon} \in (E\mathcal{V})^n$ be a canonical tuple with some n-ary domain. The set $\langle\!\langle\rho\rangle\!\rangle$ is then defined to contain the meta rule

$$\sigma\langle\vec{\varepsilon}\rangle \Longrightarrow \vartheta \stackrel{\mathsf{df}}{=} \rho_{\mathcal{V}}(\sigma\langle\vec{\varepsilon}\rangle)$$

This is then repeated for all possible n-ary domains, in this way making sure that the set $\langle\!\langle\rho\rangle\!\rangle$ contains precisely one meta rule per domain. The only thing left to prove is that any meta rule in $\langle\!\langle\rho\rangle\!\rangle$ adheres to the GSOS conditions, i.e., in the situation above, that $\mathrm{vars}(\vartheta) \subseteq \mathrm{rng}(\vec{\varepsilon})$.

Now suppose there exists a variable $v \in \mathcal{V}$ which is contained in $\mathrm{vars}(\vartheta)$ but not in $\mathrm{rng}(\vec{\varepsilon})$. By definition, this means that $v = v_i^t$ for some unique $1 \leq i \leq n$ and $t \in \mathcal{T}$; moreover, $\varepsilon_i(t)\uparrow$, otherwise v_i^t would be contained in $\mathrm{rng}(\vec{\varepsilon})$.

Since $\mathcal{T}$ has no maximal elements with respect to the induced order $\leq$—see Section 3.1— there exists $u \in \mathcal{T}$ such that $t \leq u$. Consequently, since ε_i is an evolution, also $\varepsilon_i(u)\uparrow$, and consequently, also $v_i^u \notin \mathrm{rng}(\vec{\varepsilon})$. Consider now the renaming function $f : \mathcal{V} \to \mathcal{V}$, defined as

$$v \mapsto \begin{cases} v_i^u & \text{if } v = v_i^t \\ v & \text{otherwise} \end{cases}$$

All that f does, is rename v_i^t to v_i^u and leave every other variable in $\mathcal{V}$ untouched, so in particular $Ef(\varepsilon_i) = f \circ \varepsilon_i = \varepsilon_i$ since $u \notin \mathrm{dom}(\varepsilon_i)$. By naturality of ρ, we know that the following square must commute:

$$\begin{tikzcd} \Sigma E\mathcal{V} \arrow[r, "\rho_{\mathcal{V}}"] \arrow[d, "\Sigma E f"'] & ET\mathcal{V} \arrow[d, "ETf"] \\ \Sigma E\mathcal{V} \arrow[r, "\rho_{\mathcal{V}}"'] & ET\mathcal{V} \end{tikzcd}$$

Let us see what happens for $\sigma\langle\vec{\varepsilon}\rangle \in \Sigma E\mathcal{V}$. First going down, applying $\Sigma E f$, still yields $\sigma\langle\vec{\varepsilon}\rangle$, by definition of f. Then, applying $\rho_{\mathcal{V}}$, results in ϑ which, in particular, contains the variable $v_i^t \notin \mathrm{rng}(\vec{\varepsilon})$. Chasing $\sigma\langle\vec{\varepsilon}\rangle$ around the other path of the diagram, first applying $\rho_{\mathcal{V}}$, yields ϑ, as before, and then, applying ETf, produces the result $\vartheta[f]$. However, $\vartheta[f]$ does *not*, by definition of f, contain any occurrence of v_i^t: all of those have been substituted by v_i^u. Hence, the square does *not* commute, yielding a contradiction to the assumption that ρ is natural. Therefore, we obtain that all meta rules in $\langle\!\langle\rho\rangle\!\rangle$ are indeed GSOS meta rules, concluding the proof of admissibility. $\Box$

Using canonical tuples to describe admissible sets of meta rules, we even obtain.

Theorem 6.31
The two constructions $R \mapsto [\![R]\!]$ and $\rho \mapsto \langle\!\langle\rho\rangle\!\rangle$ are mutually inverse. Hence, there is a one-to-one correspondence between admissible sets of meta rules and natural transformations of type $\Sigma E \Rightarrow ET$.

Proof: It should be obvious that one of the two composites, viz., $R \mapsto [\![R]\!] \mapsto \langle\!\langle[\![R]\!]\rangle\!\rangle$, is the identity: given a meta rule $\sigma\langle\vec{\varepsilon}\rangle \Longrightarrow \vartheta$ in R, $[\![R]\!]$ is defined in such a way that $[\![R]\!]_{\mathcal{V}}(\sigma\langle\vec{\varepsilon}\rangle) = \vartheta$, and consequently, $\langle\!\langle[\![R]\!]\rangle\!\rangle$ contains the meta rule $\sigma\langle\vec{\varepsilon}\rangle \Longrightarrow \vartheta$; the converse direction, starting with a meta rule from $\langle\!\langle[\![R]\!]\rangle\!\rangle$, also works since all of the used steps are logical equivalences.

Given a natural transformation $\rho : \Sigma E \Rightarrow ET$ such that $\rho_{\mathcal{V}}(\sigma\langle\vec{\varepsilon}\rangle) = \vartheta$, the set $\langle\!\langle\rho\rangle\!\rangle$ contains the meta rule $\sigma\langle\vec{\varepsilon}\rangle \Longrightarrow \vartheta$; consequently, $[\![\langle\!\langle\rho\rangle\!\rangle]\!]_{\mathcal{V}}(\sigma\langle\vec{\varepsilon}\rangle) = \vartheta$, i.e., $\rho_{\mathcal{V}}$ and $[\![\langle\!\langle\rho\rangle\!\rangle]\!]_{\mathcal{V}}$ coincide for arguments with canonical tuples $\vec{\varepsilon}$.

However, this is enough to conclude that the complete natural transformations are identical. This follows by essentially the same argument as when showing that $[\![R]\!]$ is natural: the 'unassuming' nature of canonical tuples (their genericity), together with naturality, imply that the values of *all* components for arbitrary arguments are determined by the values of the $\mathcal{V}$-component at canonical tuples, using renamings obtained from Lemma 6.25(2). $\Box$

Note that under the correspondence, $(\sigma\langle\vec{\varepsilon}\rangle \Longrightarrow \vartheta) \in R$ iff $[\![R]\!]_{\mathcal{V}}(\sigma(\vec{\varepsilon})) = \vartheta$, and $\rho_{\mathcal{V}}(\sigma(\vec{\varepsilon})) = \vartheta$ iff $(\sigma\langle\vec{\varepsilon}\rangle \Longrightarrow \vartheta) \in \langle\!\langle\rho\rangle\!\rangle$, for an admissible set R of meta rules and a natural transformation $\rho : \Sigma E \Rightarrow ET$, respectively, so the important part of ρ is really just the component at $\mathcal{V}$, which we could therefore call the *reference component*. This is, unsurprisingly, very reminiscent of what was going on in the dsl-format and its corresponding type of natural transformations.

The next goal for the format is a (meta) rule-based characterisation of the ε-diagram for which we introduce a notion of co-pointedness; the terminology stems from the fact that $\langle E, \varepsilon\rangle$ is a co-pointed endofunctor.

Definition 6.32

Call a meta rule $\sigma\langle\vec{e}\rangle \Longrightarrow \vartheta$ *co-pointed* if $\theta_0 = \sigma\langle e_1(0), \ldots, e_n(0)\rangle$, and call a set R of meta rules co-pointed if each meta rule in R is.

Analogous reasoning as for (6.6) immediately yields:

Theorem 6.33

Admissible co-pointed sets of meta rules are in one-to-one correspondence with natural transformations $\rho : \Sigma E \Rightarrow ET$ satisfying the ε-diagram. $\square$

Finally, in order to produce a similar characterisation of the δ-diagram, which uses the maps ρ_E and the induced distributive law $\ell = \ell_\rho$, which in turn uses ρ_T, one characterises the values of $\rho_{E\mathcal{V}}$ and $\rho_{T\mathcal{V}}$ in terms of $\rho_{\mathcal{V}}$ (which, as seen above, essentially describes the correspondence from Theorem 6.31) for specific arguments, viz., the ones that appear in the diagrams.

Lemma 6.34

Let: $\vec{\varepsilon}$ be a canonical n-ary tuple; $\vec{\vartheta} \in (ET\mathcal{V})^n$ with $\mathrm{dom}(\vec{\vartheta}) = \mathrm{dom}(\vec{\varepsilon})$; $\rho : \Sigma E \Rightarrow ET$ be a natural transformation; $\sigma \in \Sigma$ be an n-ary function symbol. Then:

1. $\rho_{E\mathcal{V}}(\sigma(\overrightarrow{\delta\varepsilon})) = \vartheta[\varepsilon_i + t/v_i^t]$ if and only if $\rho_{\mathcal{V}}(\sigma(\vec{\varepsilon})) = \vartheta$.
2. $\rho_{T\mathcal{V}}(\sigma(\vec{\vartheta})) = \vartheta[\vartheta_i(t)/v_i^t]$ if and only if $\rho_{\mathcal{V}}(\sigma(\vec{\varepsilon})) = \vartheta$.

Proof:

1. Since $\vec{\varepsilon}$ is canonical, and $\mathrm{dom}(\overrightarrow{\delta\varepsilon}) = \mathrm{dom}(\vec{\varepsilon})$, $\vec{\varepsilon}$ is the corresponding canonical tuple of $\overrightarrow{\delta\varepsilon}$. Thus, applying Lemma 6.25(2), we obtain the function

$$\varphi_E \stackrel{\mathsf{df}}{=} \varphi_{\overrightarrow{\delta\varepsilon}} : \mathcal{V} \to E\mathcal{V}$$

which satisfies the property that $E\varphi_E(\varepsilon_i) = \delta\varepsilon_i$, and which maps $v_i^t \in \mathrm{dom}(\varepsilon_i)$ to $\delta\varepsilon_i(t) = \varepsilon_i + t$. Then, since ρ was assumed to be natural, the following square must commute:

$$\begin{tikzcd} \Sigma E\mathcal{V} \arrow[r, "\rho_{\mathcal{V}}"] \arrow[d, "\Sigma E\varphi_E"'] & ET\mathcal{V} \arrow[d, "ET\varphi_E"] \\ \Sigma E(E\mathcal{V}) \arrow[r, "\rho_{E\mathcal{V}}"'] & ETE\mathcal{V} \end{tikzcd}$$

Say that $\rho_{\mathcal{V}}(\sigma\langle\vec{\varepsilon}\rangle) = \vartheta \in ET\mathcal{V}$. Then, chasing it around the two equivalent paths of the square yields that $\rho_{E\mathcal{V}}(\sigma\langle\overrightarrow{\delta\varepsilon}\rangle) = ET\varphi_E(\vartheta) = \vartheta[\varphi_E]$; spelled out in more detail, this is precisely the claim.

2. Since $\mathrm{dom}(\vec{\vartheta}) = \mathrm{dom}(\vec{\varepsilon})$, again $\vec{\varepsilon}$ is the corresponding canonical tuple for $\vec{\vartheta}$ and so, by Lemma 6.25(2), we obtain the map

$$\varphi_T \stackrel{\mathrm{df}}{=} \varphi_{\vec{\vartheta}} : \mathcal{V} \to T\mathcal{V}$$

which maps $v_i^t \in \mathrm{dom}(\varepsilon_i)$ to $\vartheta_i(t) \in TX$, hence the claim follows. □

Now that we have obtained characterisations of both $\rho_{E\mathcal{V}}$ and $\rho_{T\mathcal{V}}$ at specific arguments, we can now proceed to present a meta rule-based characterisation of the induced distributive law ℓ for a natural transformation $\rho : \Sigma E \Rightarrow ET$ in the form of *R-derivations*, R being a set of canonical meta rules. It is important that all meta rules in R are canonical since we want to use the renaming function obtained by applying Lemma 6.25(2). Note that, for the sake of simplicity, we omit any reference to unit η and multiplication μ of T in the following definition.

Definition 6.35

Given a set R of canonical meta rules, define the notion of *R-derivation* as follows. For $\zeta \in TE\mathcal{V}$ and $\vartheta \in ET\mathcal{V}$, say that ϑ is R-derived by ζ, writing $R \vdash \zeta \Longrightarrow \vartheta$, if there is a finite proof using only the two following rules:

1. If $\zeta = e$ for some $e \in E\mathcal{V}$ then $R \vdash e \Longrightarrow e$

2. If $\sigma \in \Sigma$ is an n-ary function symbol and $\zeta_1, \ldots, \zeta_n \in TE\mathcal{V}$ then

$$\frac{\{R \vdash \zeta_i \Longrightarrow \vartheta_i,\ \mathrm{dom}(\vartheta_i) = \mathrm{dom}(\varepsilon_i)\}_{1 \le i \le n} \quad (\sigma\langle\vec{\varepsilon}\rangle \Longrightarrow \vartheta) \in R}{R \vdash \sigma\langle\vec{\zeta}\rangle \Longrightarrow \vartheta[\varphi_{\vec{\vartheta}}]}$$ ■

In particular, if $(\sigma\langle\vec{\varepsilon}\rangle \Longrightarrow \vartheta) \in R$, then $R \vdash \sigma\langle\vec{\varepsilon}\rangle \Longrightarrow \vartheta$, so one could call the expression $R \vdash \sigma\langle\vec{\varepsilon}\rangle \Longrightarrow \vartheta$ an *axiom*. Note that it is not necessary that R is admissible for R-derivations to make sense. However, if it is, the proof system is deterministic: if $R \vdash \zeta \Longrightarrow \vartheta$, then there exists a unique derivation, and also ϑ is unique (in the general case, this need not be true as R is not assumed to be deterministic, i.e., there might be several meta rules in R applicable at the same time). Moreover, if R is admissible, it induces the natural transformation $[\![R]\!]$ which, in turn, induces the distributive law ℓ, cf. Definition 5.6:

Lemma 6.36
Let R be an admissible set of meta rules, and let $\rho = [\![R]\!]$ be the induced natural transformation with induced distributive law $\ell = \ell_\rho : TE \Rightarrow ET$. Then the following equivalence holds for $\zeta \in TE\mathcal{V}$ and $\vartheta \in ET\mathcal{V}$:

$$\ell_{\mathcal{V}}(\zeta) = \vartheta \iff R \vdash \zeta \Longrightarrow \vartheta$$

Proof: We proceed by induction on the structure of $\zeta \in TE\mathcal{V}$, based on the isomorphism $TE\mathcal{V} \cong E\mathcal{V} + \Sigma TE\mathcal{V}$; consequently, we get two cases.

1. Suppose $\zeta = e \in E\mathcal{V}$. For this case, the defining diagram (5.9) states that $\ell_{\mathcal{V}}(e)$, or, being completely precise, $\ell_{\mathcal{V}}(\eta e)$, is equal to $E\eta(e) = \eta \circ e$, which we identify with simply $e = \vartheta \in ET\mathcal{V}$. Hence this case is taken care of by the first rule of R-derivations, stating that $R \vdash e \Longrightarrow e$.

2. Suppose $\sigma \in \Sigma$ is an n-ary operator symbol, and $\zeta_i \in TE\mathcal{V}$, for $1 \leq i \leq n$, and $\zeta = \sigma\langle\vec{\zeta}\rangle$. By (5.9), we have that

$$\ell_{\mathcal{V}}(\sigma\langle\vec{\zeta}\rangle) = E\mu_{\mathcal{V}}(\rho_{T\mathcal{V}}(\sigma\langle\ell_{\mathcal{V}}(\zeta_1), \ldots, \ell_{\mathcal{V}}(\zeta_n)\rangle))$$

   Applying the induction hypothesis to $\ell_{\mathcal{V}}(\zeta_i)$, we obtain that, for all $1 \leq i \leq n$, $\ell_{\mathcal{V}}(\zeta_i) = \vartheta_i$ if and only if $R \vdash \zeta_i \Longrightarrow \vartheta_i$. In Lemma 6.34, we have seen that, for $\vec{\varepsilon} \in (E\mathcal{V})^n$ being the corresponding canonical tuple to $\vec{\vartheta}$, it holds that

$$\rho_{T\mathcal{V}}(\sigma\langle\vec{\vartheta}\rangle) = \vartheta[\vartheta_i(t)/v_i^t] = \vartheta[\varphi_{\vec{\vartheta}}]$$

   if and only if $\rho_{\mathcal{V}}(\sigma\langle\vec{\varepsilon}\rangle) = \vartheta$, where $\varphi_{\vec{\vartheta}} : \mathcal{V} \to TX$ is the map obtained from Lemma 6.25(2) such that $\langle\varphi_{\vec{\vartheta}} \circ \varepsilon_1, \ldots, \varphi_{\vec{\vartheta}} \circ \varepsilon_n\rangle = \vec{\vartheta}$. By the definition of $\rho =$

$[\![R]\!]$, cf. Proposition 6.29, $\rho_{\mathcal{V}}(\sigma\langle\vec{\varepsilon}\rangle) = \vartheta$ if and only if R contains the meta rule $\sigma\langle\vec{\varepsilon}\rangle \Longrightarrow \vartheta$, which is the case because R is admissible. Putting all these things together, omitting reference to μ, we obtain precisely that $R \vdash \sigma\langle\vec{\zeta}\rangle \Longrightarrow \vartheta[\varphi_{\vec{\vartheta}}]$; the converse direction also holds as all steps involved are equivalences. $\square$

Intuitively, R-derivations capture the notion of *provability* from a set of rules: meta rules only apply to 'simple' terms with exactly one function symbol; the *inductive extension* given by the R-derivations then determines the action of the rules on complex terms, iterating applications of the rules (subject to necessary substitutions). Moreover, for admissible R, R-derivations describe a natural transformation (viz. ℓ), so whenever $R \vdash \zeta \Longrightarrow \vartheta$, it holds that $\text{vars}(\vartheta) \subseteq \text{vars}(\zeta)$ (taking $\text{vars}(\zeta)$ to be $\text{vars}(\vec{e})$ if $\vec{e}$ are all the evolutions occurring as variables in $\zeta \in TEX$); otherwise, following from the same argument as showing that GSOS (meta) rules induce a natural transformation, ℓ could not be natural.

Now we have all the ingredients to obtain a (meta) rule-based description of the δ-diagram:

Definition 6.37
Let R be a set of canonical meta rules, Σ a signature, $\sigma \in \Sigma$ an n-ary function symbol, $\sigma\langle\vec{\varepsilon}\rangle \Longrightarrow \vartheta$ a meta rule in R, and $t, u \in \mathcal{T}$. Then R is called *continuous* if the following two statements are equivalent:

1. $\sigma\langle\vec{\varepsilon}\rangle \stackrel{t+u}{\leadsto} \theta_{t+u}$, and
2. $\sigma\langle\vec{\varepsilon}\rangle \stackrel{t}{\leadsto} \theta_t \wedge R \vdash \theta_t[\varphi_{\overrightarrow{\delta\varepsilon}}] \Longrightarrow \vartheta' \wedge \vartheta'(u) = \theta_{t+u}$ ■

The terminology 'continuous' is used since the equivalence is a generalised, rule-based version of time continuity: if one single application of the rules allows to derive a $\stackrel{t+u}{\leadsto}$-transition, it must be possible to first derive a $\stackrel{t}{\leadsto}$-transition in one step, followed by a derivation (of arbitrary finite length) of a $\stackrel{u}{\leadsto}$-transition: the latter holds because, by the notation used in (4.4), we can rewrite $\vartheta'(u) = \theta_{t+u}$ as $\vartheta' \stackrel{u}{\leadsto} \theta_{t+u}$.

This use of derivations also precisely marks the difference between the two δ-diagrams in (5.19) and (5.11): the former specifies that the $\stackrel{u}{\leadsto}$-transition must be derivable at once, whereas the latter, as just stated, allows several steps to derive the transition; this is due to the fact that the abstract rules in (5.18) only allow terms in rule conclusions with at most one function symbol, so at most one rule application is necessary/possible, whereas in (5.12), arbitrary terms are allowed. Note that continuous sets of meta rules need not be admissible, yet using Lemmas 6.34 and 6.36, we get:

Theorem 6.38

There is a one-to-one correspondence between admissible, continuous sets of meta rules and natural transformations $\Sigma E \Rightarrow ET$ satisfying the δ-diagram.

Proof: The only thing left to prove is that, if R is an admissible set of meta rules, continuity of R is equivalent to satisfaction of the δ-diagram for $\rho = [\![R]\!]$, which really is a natural transformation because R is admissible. So assume that R is continuous, and let $\sigma\langle\varepsilon\rangle \Longrightarrow \vartheta$ be a meta rule in R. By the definition of continuity, we have to show that $\sigma\langle\vec{\varepsilon}\rangle \overset{t+u}{\rightsquigarrow} \theta_{t+u}$ if and only if $\sigma\langle\vec{\varepsilon}\rangle \overset{t}{\rightsquigarrow} \theta_t$ and $R \vdash \theta_t[\varphi_{\overrightarrow{\delta\varepsilon}}] \Longrightarrow \vartheta'$ and $\vartheta' \overset{u}{\rightsquigarrow} \theta_{t+u}$. So assume $\sigma\langle\vec{\varepsilon}\rangle \overset{t+u}{\rightsquigarrow} \theta_{t+u}$ which, by definition, is equal to $\delta_{T\mathcal{V}}(\rho_{\mathcal{V}}(\sigma\langle\vec{\varepsilon}\rangle))(t)(u)$.

Because of continuity, we also know that $\sigma\langle\vec{\varepsilon}\rangle \overset{t}{\rightsquigarrow} \theta_t = \vartheta(t)$. Consequently, applying Lemma 6.34 and Lemma 6.25(2), we obtain that $\rho_{E\mathcal{V}}(\sigma\langle\overrightarrow{\delta\varepsilon}\rangle)(t) = \theta_t[\varphi_{\overrightarrow{\delta\varepsilon}}] \in TE\mathcal{V}$. As shown in Lemma 6.36, $\ell_{\mathcal{V}}(\theta_t[\varphi_{\overrightarrow{\delta\varepsilon}}]) = \vartheta' \in ETX$ if and only if $R \vdash \theta_t[\varphi_{\overrightarrow{\delta\varepsilon}}] \Longrightarrow \vartheta'$, and thus, $\vartheta'(u)$ is exactly the value of $E\ell_{\mathcal{V}}(\rho_{E\mathcal{V}}(\sigma\langle\overrightarrow{\delta\varepsilon}\rangle))(t)(u)$. Since, by continuity, $\vartheta'(u) = \theta_{t+u}$, this means that the δ-diagram commutes when chasing round $\sigma\langle\vec{\varepsilon}\rangle$. Since all involved maps are natural in $\mathcal{V}$, and ρ in particular is completely determined by the values of exactly such tuples, this implies that continuity implies the commutation of the δ-diagram in general.

In the converse direction, assuming the the δ-diagram commutes, one simply has to chase round terms $\sigma\langle\vec{\varepsilon}\rangle \in \Sigma E\mathcal{V}$ because they are all that continuity is concerned with. Since both Lemma 6.34 and Lemma 6.36 are equivalences, and the diagram asserts the equality of partial functions, i.e., a Kleene equality, which also is an equivalence, we obtain the defining equivalence of continuity. $\square$

Corollary 6.39

There is a one-to-one correspondence between abstract CSOS for timed processes and admissible, co-pointed, and continuous sets of meta rules. $\square$

As already shown, the schematic format from Section 6.2.1 induces CSOS rules for E, hence:

Corollary 6.40
The schematic format induces an admissible, co-pointed and continuous set of meta rules. □

There is also a concrete way to derive the set of meta rules corresponding to an operator defined by some admissible operator from the schematic format. Consider the case of TeCCS's strong choice $+$ operator with associated map

$$[\![+]\!] : (E\mathcal{V})^2 \to E(\mathcal{V} + \Sigma\mathcal{V}) \subseteq ET\mathcal{V}$$

which was already shown to respect the structure of E in Prop. 6.17. Let $(\varepsilon_1, \varepsilon_2)$ be an arbitrary canonical tuple. Then $[\![+]\!]$ induces the following meta rule:

$$\sigma\langle\varepsilon_1, \varepsilon_2\rangle \Longrightarrow \vartheta \stackrel{\mathsf{df}}{=} \lambda t. \begin{cases} \sigma(v_1^t, v_2^t) & \text{if } \varepsilon_1(t)\downarrow \wedge\, \varepsilon_2(t)\downarrow \\ \text{undef} & \text{if } \varepsilon_1(t)\uparrow \vee\, \varepsilon_2(t)\uparrow \end{cases}$$

In this manner, going through all possible canonical tuples, $[\![+]\!]$ and similarly, any admissible operator necessarily results in an admissible set of meta rules.

Remark 6.41
We would like to stress once more the fact that the characterisation of CSOS rules for timed processes obtained in Corollary 6.39 by suitable admissible sets of meta rules is *by no means* an effective description of a rule format for timed processes: after all, the characterisation uses infinite sets of meta rules, each of which already represents an infinite set of operational rules, as shown in (6.18). Actually, quite the contrary is true: we believe that this particular characterisation simply serves the purpose to illustrate the expressivity of abstract CSOS rules for timed processes.

Due to its complexity and ineffectiveness, the characterisation also raises the question whether the model we are using, viz., timed processes described by evolutions—or equivalently, by TTSs—is adequate in the sense that the model has the infinite aspect, which becomes apparent in the meta rule-based characterisation, built into its very foundation: evolutions (or TTSs) almost by design induce infinitely branching structures, and in order to completely capture such structures, infinitary rules (and infinite sets thereof) are required. To sum up, the previously presented 'rule format' should be taken with a (quite big) pinch of salt: although it is certainly a demonstration of the complexity of the situation, it is also very clearly a call to arms for finding a more appropriate and realistic model for timed processes. $\Diamond$

Remark 6.42

Bartels [Bar02], when deriving his rule format *PGSOS* (GSOS for probablistic trasition systems), uses a *decomposition* approach: instead of trying to find immediately a syntactic characterisation of abstract GSOS rules instantiated with the behaviour for probabilistic transition systems, he first develops, in a top-down way, a 'toolkit' of decomposition results allowing to equivalently express the original natural transformation in terms of increasingly simpler ones; next, having reached an appropriate level of simplicity, he provides a syntactic characterisation for the particular type of natural transformation; finally, reversing the decomposition results on the level of syntax, in a bottom-up way, he provides characterisations of the more complex natural transformations, in the end obtaining the complete rule format.

Despite his success, we would not expect to be able to benefit from applying a similar approach in our case: the final syntactic characterisation still has to capture the infinitary nature of the evolution comonad; hence, even if we could limit ourselves to finitary objects during the decomposition, at some point the transition to infinite objects must occur, and we expect to run into similar difficulties then as we did while obtaining the presented results in our more direct way. $\Diamond$

Chapter 7

Heterogeneous Processes

In this chapter, finally, we deal with the long-postponed problem of combining action and time transitions in one model. To do so, we will pass through all the stages we have previously passed through for time transitions alone, and adapt them appropriately in order to also incorporate action transitions. Up to this point, we represented timed processes by transition systems with only one type of transitions, viz., TTSs; therefore, we could call such timed processes *homogeneous*. The ones to be dealt with now are consequently *heterogeneous* because they contain two kinds of transitions[1].

The first task, since our whole approach is based on coalgebraically describing the 'right' kind of transition systems, is to formally introduce transition systems with the two kinds of transitions, together with an appropriate notion of bisimulation. This leads us to *heterogeneous transition systems* (HTSs) and *heterogeneous bisimulation*, obtained by *independently* combining an LTS and a TTSs on the same set of states.

In order to then find a suitable coalgebraic characterisation of such transition systems, it is a crucial point whether the considered time domain is a free monoid or equivalently, whether the corresponding evolution comonad E is cofreely generated, cf. Section 4.3. Since HTSs merely consist of the interference-free juxtaposition of an LTS and a TTS on the same set of states, the appropriate behaviour is obtained by using a suitable notion of *product* of the corresponding behaviours for the two types of

[1]Note that, generalising this approach, one could consider processes performing several different types of steps, according to different notions of 'computations,' as long as these do not interfere with, or depend on, each other.

transition systems.

In the special case of discrete time, $E_{\mathbb{N}}$ is cofreely generated from the functor $B_{\mathbb{N}}$ (see Theorem 4.23). Hence, TTSs over $\mathbb{N}$ are completely described by coalgebras for the functor $B_{\mathbb{N}}$, exactly like (image finite) LTSs are the coalgebras for the functor B_A from (1.1). So we simply use the (point-wise) product of the behaviour functors B_A and $B_{\mathbb{N}}$ to model HTSs over discrete time.

However, if E is not cofreely generated, the situation becomes quite a bit more complex: this is due to fact that TTSs, in general, can only be described as Coalgebras, i.e., the coalgebras have to satisfy additional constraints, unlike the coalgebras corresponding to LTSs. Since there is no way to pass from an E-Coalgebra to a simple coalgebra, as in the discrete case, we proceed in the opposite direction: the cofree comonad B_A^∞ on B_A exists (as was shown, e.g., in [Bar93]) and, in particular, it satisfies the property that the Coalgebras for B_A^∞ are the same as the standard coalgebras for B_A, i.e., as image finite LTSs.

Consequently, our solution for this case of HTSs is to again use the product, only this time that of comonads, viz., $B_A^\infty \times E$. One has to be a bit careful, though, since this product turns out to be vastly different from the simple point-wise product of functors; it is, a priori, not even clear whether it exists. Fortunately, by exploiting recent results by Hyland, Plotkin, and Power on the dual case of sums of monads in [HPP], we can show that the product indeed exists in the case under consideration, and also give a (reasonably) concrete description of it as a composite comonad.

Following these two different tracks for the definition of appropriate behaviours, we can then investigate corresponding abstract operational rules: for discrete time, the theory from [TP97] can be used, as was already the case for TTSs over discrete time, while for arbitrary time domains, we have to apply the framework for behaviour comonads developed in Chapter 5.

Building on this, we can, furthermore, present some results concerning syntactic rule formats for the heterogeneous case, again based on what was shown previously in [TP97] and Chapter 6. Unfortunately, at the moment, these results are far from complete, much must be left for further research, in particular, the case that action and time rules are no longer independent (as, e.g., in TiCCS [Wan90]). Despite this unfinished and unpolished state of the development presented here, we still manage to give a complete and conceptual explanation of the congruence result for TeCCS.

However, the success of the coalgebraic approach should not be measured purely by the fact that we are able to give abstract proofs which imply the well-established congruence results for concrete languages like TeCCS: proving these results directly is not very hard. What is more important is that the coalgebraic approach provides *'formats' for rule formats* (via abstract rules) and *semantic* explanations beyond specific syntax.

7.1 Heterogeneous Transition Systems

In order to talk about processes which do not only perform time but also action transitions, we have to deal with transition systems like the ones in Figure 3.1 which we now introduce formally:

Definition 7.1
Let A be a finite set of actions, let $\mathcal{T}$ be a time domain, and let P be a set. A *heterogeneous transition system* (HTS) on P is a tuple $\langle P, A, \mathcal{T}, \rightarrow, \rightsquigarrow\rangle$ such that

- $\langle P, A, \rightarrow\rangle$ is an image finite LTS
- $\langle P, \mathcal{T}, \rightsquigarrow\rangle$ is a TTS

and sometimes simply say that P is an HTS over $\langle A, \mathcal{T}\rangle$ when the two transition relations are clear from the context. The same notational conventions about transitions apply as before, e.g., instead of $\langle p, \alpha, p'\rangle \in\rightarrow$, we write $p \overset{\alpha}{\rightarrow} p'$. ∎

This definition simply formalises the intuition that 'proper' processes can perform both computations and let time pass, with the two kinds of transitions completely independent of each other, unlike, e.g., the model presented in [Wan90, Wan91]: this uses the axiom (Persistency), in addition to (Maximal Progress), both of which relate action and time transitions. This has very clear effects on the coalgebraic description of HTSs, and also on abstract and concrete rules for such processes. Even though HTSs are general enough to also include such models, complications eventually arise when treating such languages within the categorical framework due to the mismatch between model (no relating axioms) and languages (using axioms like (Maximal Progress) and (Persistency)). For such languages, one should really already impose the relevant conditions on the models, in order to be as precise as possible.

If we restrict attention to a finite the set of actions[2] A, the operational semantics of TeCCS, as presented in [MT90], defines an HTS on the set of all closed terms of the language over $\langle A, \mathbb{N} \rangle$.

There is an appropriate heterogeneous version of bisimulation for such HTSs:

Definition 7.2

Let $\langle P_i, A, \mathcal{T}, \rightarrow_i, \rightsquigarrow_i \rangle$, for $1 \leq i \leq 2$, be two HTSs over $\langle A, \mathcal{T} \rangle$. A *heterogeneous bisimulation* between P_1 and P_2 is a relation $R \subseteq P_1 \times P_2$ such that $\langle p_1, p_2 \rangle \in R$ implies, for all $\alpha \in A$ and $t \in \mathcal{T}$, that

$$\begin{aligned}
p_1 \xrightarrow{\alpha}_1 p_1' &\Rightarrow (\exists p_2').\ p_2 \xrightarrow{\alpha}_2 p_2' \wedge \langle p_1', p_2' \rangle \in R \\
p_2 \xrightarrow{\alpha}_2 p_2' &\Rightarrow (\exists p_1').\ p_1 \xrightarrow{\alpha}_1 p_1' \wedge \langle p_1', p_2' \rangle \in R \\
p_1 \overset{t}{\rightsquigarrow}_1 p_1' &\Rightarrow (\exists p_2').\ p_2 \overset{t}{\rightsquigarrow}_2 p_2' \wedge \langle p_1', p_2' \rangle \in R \\
p_2 \overset{t}{\rightsquigarrow}_2 p_2' &\Rightarrow (\exists p_1').\ p_1 \overset{t}{\rightsquigarrow}_1 p_1' \wedge \langle p_1', p_2' \rangle \in R
\end{aligned}$$

We write $p \sim q$ if there exists a heterogeneous bisimulation R such that $\langle p, q \rangle \in R$. ■

From the same standard principles as in Proposition 3.27, it follows that $\sim$ is the largest heterogeneous bisimulation between P_1 and P_2 and, in case $P_1 = P_2$, it is additionally an equivalence relation. Clearly, any heterogeneous bisimulation is both an action and a time bisimulation, establishing $\sim \subseteq (\sim_a \cap \sim_t)$. Conversely, Remark 3.30 shows that $(\sim_a \cap \sim_t) \not\subseteq \sim$: there are processes which are both action bisimilar and time bisimilar, yet not bisimilar in the heterogeneous sense, viz., the two transition systems in Figure 3.1. Therefore, we obtain $\sim \subsetneqq (\sim_a \cap \sim_t)$, i.e., the inclusion is strict. Also note that heterogeneous bisimulation is precisely the one used for TeCCS in [MT90].

[2] Any system specification in the language can use only finitely many labels anyway, so this restriction is not very strong.

7.2 Heterogeneous Behaviours

In this section, we present coalgebraic characterisations of HTSs in the same vein as the characterisation of image finite LTSs as B_A-coalgebras, or how TTSs were characterised as E-Coalgebras. We first present a general result which we then refine in the following two subsections, depending on whether E is cofreely generated or not.

Definition 7.3

Let $\mathcal{C}$ be a category, let H be an endofunctor on $\mathcal{C}$, and let D be a comonad on $\mathcal{C}$. Define the category $\langle H,D\rangle$**–Coalg** as the *pullback category* in the following diagram

$$\begin{tikzcd} \langle H,D\rangle\textbf{–Coalg} \arrow[r] \arrow[d] & D\textbf{–Coalg} \arrow[d, "U_D"] \\ H\textbf{–coalg} \arrow[r, "U_H"'] & \mathcal{C} \end{tikzcd} \tag{7.1}$$

where U_D and U_H denote the respective forgetful functors from Coalgebras and coalgebras, respectively, to their carriers. The description of the pullback category is essentially the same as for sets, viz., its collection of objects is the 'sub-collection' of the product category given by those pairs of objects which have the same value after applying the two maps U_D and U_H; this boils down to pairs consisting of both an H-coalgebra and a D-Coalgebra which have the same carrier:

$$\begin{tikzcd} & X \arrow[dl, "k_H"'] \arrow[dr, "k_D"] & \\ HX & & DX \end{tikzcd} \tag{7.2}$$

where k_D is a D-Coalgebra. The morphisms of $\langle H,D\rangle$**–Coalg** are maps in $\mathcal{C}$ between the carriers which are homomorphisms of both H-coalgebras and D-Coalgebras. ■

The notation $\langle H,D\rangle$**–Coalg**, suggesting a 'pairing' of behaviours, is chosen to point in the direction of products, as will indeed be the case, although in not necessarily the most straightforward fashion. Instantiating the definition for $\mathcal{C} = \textbf{Set}$, $H = B_A$, and $D = E$, we obtain the category $\langle B_A,E\rangle$**–Coalg** and the following characterisation:

Proposition 7.4

Let A be a finite set of labels, let and $\mathcal{T}$ be a time domain. Then an HTS over $\langle A,\mathcal{T}\rangle$ on X is the same as an object in $\langle B_A,E\rangle$**–Coalg**.

Proof: This follows immediately from the correspondence between image finite LTSs (TTSs over $\mathcal{T}$) and B_A-coalgebras (resp. E-Coalgebras), and the fact that, according to the definition of HTSs, no connection between the two different transition relations is assumed. $\square$

Note that $\langle H,D\rangle$**–Coalg** is not the same as the coalgebras for the (point-wise) product of H and D, the latter being regarded only as a *functor*, since then we would only get a D-coalgebra, rather than a D-Coalgebra, in the resulting structures; for the special case of B_A and E, this would result in not necessarily obtaining a TTS, only a deterministic LTS with labels in $\mathcal{T}$ for which axioms (ZeroDelay) and (Continuity) might not hold, cf. Section 4.2.2.

Next, we shall derive 'proper' coalgebraic characterisations of $\langle B_A,E\rangle$**–Coalg**, in the sense of finding a functor (comonad) such that its coalgebras (resp. Coalgebras) are equivalent to 'mixed' pairs of coalgebras in $\langle B_A,E\rangle$**–Coalg**, which we need in order to define abstract operational rules for (languages for) HTSs.

7.2.1 Discrete Time

For $\mathcal{T}=\mathbb{N}$, we have seen that the evolution comonad $E=E_{\mathbb{N}}$ is the cofree comonad on the functor $B_{\mathbb{N}}=1+\mathrm{Id}$ (see Theorem 4.23). Consequently, for any $E_{\mathbb{N}}$-Coalgebra $k_E : X\to E_{\mathbb{N}}X$, there is a unique $B_{\mathbb{N}}$-coalgebra $k_{\mathbb{N}} : X\to B_{\mathbb{N}}X = 1+X$ which induces the same TTS on X as k_E, i.e., $k_{\mathbb{N}}$ is the unique $B_{\mathbb{N}}$-coalgebra such that $k_E=(k_{\mathbb{N}})^{\infty}$. Hence, an object $\langle X,k_B,k_E\rangle$ of $\langle B_A,E\rangle$**–Coalg** can equivalently be described as

(7.3)
$$\begin{tikzcd} & X \arrow[dl, "k_B"'] \arrow[dr, "k_{\mathbb{N}}"] & \\ B_A X & & B_{\mathbb{N}} X \end{tikzcd}$$

i.e., a pair of coalgebras on the same carrier, rather than a 'mixed' coalgebra/Coalgebra pair as in the original definition of $\langle B_A,E\rangle$**–Coalg**. By the universal property of the product, the two maps k_B and $k_{\mathbb{N}}$ are equivalent to one single, moreover unique, map $k : X\to B_AX\times B_{\mathbb{N}}X$, viz., the *pairing* of k_B and $k_{\mathbb{N}}$; since the product of functors is simply taken point-wise, we have $B_AX\times B_{\mathbb{N}}X=(B_A\times B_{\mathbb{N}})X$, and so:

Proposition 7.5
HTSs over $\langle A, \mathbb{N} \rangle$ are in one-to-one correspondence with $(B_A \times B_{\mathbb{N}})$-coalgebras. $\square$

Note that this characterisation only works because there is no interference between the two transition systems on X, otherwise one could not simply use the product functor as the appropriate behaviour.

Remark 7.6
To coalgebraically model HTSs where action and time transitions have to satisfy some conditions relating them to each other, e.g., (Maximal Progress) or (Persistency), one might not want to use the product functor (also in the general case, the product of comonads is too 'generous,' see the next section) because it does not account for such dependencies. In such a case, one would rather use some kind of (monoidal?) tensor $B_A \otimes B_{\mathbb{N}}$ such that the coalgebras for $B_A \otimes B_{\mathbb{N}}$ then correspond to HTSs where the desired axioms relating the two kinds of transitions are satisfied, or a *subfunctor* of the product. A similar remark applies, as we shall see, to the case of an arbitrary time domain: there, the resulting behaviour will turn out to be exactly the product *comonad* of B_A^∞ and E, i.e., apart from the more complex technicalities, essentially the same solution as in the discrete case and consequently, the above still remains valid in the generalised setting (generalised to either a tensor comonad or a subcomonad). $\Diamond$

As a corollary, because morphisms in $(B_A \times B_{\mathbb{N}})$**–coalg** are exactly the morphisms in $\langle B_A, E \rangle$**–Coalg** (modulo the isomorphism $B_{\mathbb{N}}$**–coalg** $\cong E$**–Coalg**), we obtain:

Corollary 7.7
Coalgebraic bisimulation for $B_A \times B_{\mathbb{N}}$ is the same as heterogeneous bisimulation between HTSs over $\langle A, \mathbb{N} \rangle$. $\square$

Thus, $B_A \times B_{\mathbb{N}}$ is the appropriate behaviour functor to describe the operational semantics of heterogeneous timed processes over discrete time. Next, we are going to prove two easy results on the preservation of (weak) pullbacks. For this, given a pullback square

$$
\begin{tikzcd}
P \arrow[r, "q"] \arrow[d, "p"'] & Y \arrow[d, "g"] \\
X \arrow[r, "f"'] & Z
\end{tikzcd}
\tag{7.4}
$$

recall that we have the following concrete description of P:

$$P = \{\langle x,y\rangle \in X \times Y \mid fx = gy\} \subseteq X \times Y \tag{7.5}$$

Using this, we can show:

Proposition 7.8

1. $B_{\mathbb{N}}$ preserves pullbacks.

2. In **Set**, the pullback of a product is the product of the pullbacks, i.e., given two pullback squares for $i \in \{1,2\}$,

$$\begin{tikzcd} P_i \arrow[r, "q_i"] \arrow[d, "p_i"'] & Y_i \arrow[d, "g_i"] \\ X_i \arrow[r, "f_i"'] & Z_i \end{tikzcd}$$

and the pullback square

$$\begin{tikzcd} P \arrow[r, "q_1 \times q_2"] \arrow[d, "p_1 \times p_2"'] & Y_1 \times Y_2 \arrow[d, "g_1 \times g_2"] \\ X_1 \times X_2 \arrow[r, "f_1 \times f_2"'] & Z_1 \times Z_2 \end{tikzcd}$$

then $P \cong P_1 \times P_2$.

Proof: 1. Given a pullback square (7.4), we have to show that $B_{\mathbb{N}}P = 1 + P \cong Q$ where Q is the pullback of

$$\begin{tikzcd} Q \arrow[r, "s"] \arrow[d, "r"'] & 1+Y \arrow[d, "1+g"] \\ 1+X \arrow[r, "1+f"'] & 1+Z \end{tikzcd}$$

For this, calculate

$$Q = \{\langle \alpha,\beta\rangle \in (1+X) \times (1+Y) \mid (1+f)(\alpha) = (1+g)(\beta)\}$$

By definition, $(1+f)(\alpha) \in 1+Z$ is in the 1-component if and only if the same holds for $(1+g)(\beta) \in 1+Z$, so $Q \cong 1+P = B_{\mathbb{N}}P$.

2. Using the concrete description (7.5), we calculate

$$\begin{aligned}
P &= \{\langle \alpha, \beta\rangle \in (X_1 \times X_2) \times (Y_1 \times Y_2) \mid (f_1 \times f_2)(\alpha) = (g_1 \times g_2)(\beta)\} \\
&= \{\langle\langle x_1, x_2\rangle, \langle y_1, y_2\rangle\rangle \in (X_1 \times X_2) \times (Y_1 \times Y_2) \mid \langle f_1x_1, f_2x_2\rangle = \langle g_1y_1, g_2y_2\rangle\} \\
&\cong \{\langle x_1, x_2, y_1, y_2\rangle \in X_1 \times X_2 \times Y_1 \times Y_2 \mid (f_1x_1 = g_1y_1) \wedge (f_2x_2 = g_2y_2)\} \\
&\cong P_1 \times P_2
\end{aligned}$$

□

Moreover, by [Tur96, TR98], we know that in **Set**, *weak pullbacks embed pullbacks*: in the situation of (7.4), a set W, together with a pair of maps $W \to X$, $W \to Y$ such that the resulting square commutes, constitutes a weak pullback if and only if there is an injection $P \hookrightarrow W$. Using this, we obtain:

Corollary 7.9

1. $B_{\mathbb{N}}$ preserves weak pullbacks.
2. $B_A \times B_{\mathbb{N}}$ preserves weak pullbacks.

Proof:

1. Follows from the previous proposition and [Rut00].
2. By, e.g., [Tur96, TR98], we know that B_A preserves weak pullbacks; by the previous point, we also know it for $B_{\mathbb{N}}$. Consider now the commuting square

$$\begin{tikzcd}
B_AP \times B_{\mathbb{N}}P \arrow[r, "B_Aq \times B_{\mathbb{N}}q"] \arrow[d, "B_Ap \times B_{\mathbb{N}}p"'] & B_AY \times B_{\mathbb{N}}Y \arrow[d, "B_Ag \times B_{\mathbb{N}}g"] \\
B_AX \times B_{\mathbb{N}}X \arrow[r, "B_Af \times B_{\mathbb{N}}f"'] & B_AZ \times B_{\mathbb{N}}Z
\end{tikzcd}$$

obtained by applying $B_A \times B_{\mathbb{N}}$ to the pullback square (7.4). By the previous proposition, its ‘real’ pullback Q is isomorphic to $Q_A \times Q_{\mathbb{N}}$, where Q_A and $Q_{\mathbb{N}}$ are the ‘real’ pullbacks of the square (7.4) with B_A and $B_{\mathbb{N}}$, respectively, applied to it. Since both B_A and $B_{\mathbb{N}}$ preserve weak pullbacks, we know that there are injections $Q_A \hookrightarrow B_AP$ and $Q_{\mathbb{N}} \hookrightarrow B_{\mathbb{N}}P$ and so, we get an injection

$$Q \cong Q_A \times Q_{\mathbb{N}} \hookrightarrow B_AP \times B_{\mathbb{N}}P$$

establishing $B_AP \times B_{\mathbb{N}}P$ as a weak pullback. □

7.2.2 The General Case

Now consider a general time domain $\mathcal{T}$ such that E may not be cofreely generated, the main example being $\mathbb{R}$. Due to Theorem 4.17, we still obtain that the Coalgebras for E are the same as TTSs over $\mathcal{T}$, but no further simplification is possible. However, since B_A is accessible, there exists the cofree comonad B_A^∞ on it—see, e.g., [JPT+01]. As a consequence, we get B_A**–coalg** $\cong B_A^\infty$**–Coalg** and hence, any structure $\langle X, k_B, k_E\rangle$ in $\langle B_A, E\rangle$**–Coalg** can equivalently be regarded as a pair of Coalgebras

$$\begin{tikzcd} & X \arrow[dl, "k_B^\infty"'] \arrow[dr, "k_E"] & \\ B_A^\infty X & & EX \end{tikzcd} \tag{7.6}$$

where k_B^∞ is the coinductive extension of k_B. Hence, we now have a pair of Coalgebras describing HTSs over $\langle A, \mathcal{T}\rangle$:

Proposition 7.10

There is a one-to-one correspondence between HTSs over$\langle A, \mathcal{T}\rangle$ and pairs of Coalgebras as in (7.6). □

Again more abstractly, we can consider the case of a category $\mathcal{C}$ and two comonads D and D' on $\mathcal{C}$. The last characterisation of HTSs as pairs of Coalgebras means that it is an instance of a pullback category

$$\begin{tikzcd} \langle D, D'\rangle\textbf{–Coalg} \arrow[r] \arrow[d] & D\textbf{–Coalg} \arrow[d, "U_D"] \\ D'\textbf{–Coalg} \arrow[r, "U_{D'}"'] & \mathcal{C} \end{tikzcd} \tag{7.7}$$

again with U_D and $U_{D'}$ denoting the respective forgetful functors. Note that, in case $D' = H^\infty$, this is equivalent to the pullback category $\langle H, D\rangle$**–Coalg** in (7.1), since H^∞**–Coalg** $\cong H$**–coalg**; thus, $\langle B_A, E\rangle$**–Coalg** $\cong \langle B_A^\infty, E\rangle$**–Coalg**.

The question is now how to obtain a coalgebraic characterisation of $\langle B_A, E\rangle$**–Coalg** or, more generally of $\langle H, D\rangle$**–Coalg**. For this, we will use a canonical distributive law of the comonad D over the functor HD which, under the assumption that the cofree comonad $(HD)^\infty$ on HD exists, yields a comonad structure on the composite comonad $D(HD)^\infty$ with the property that $\langle H, D\rangle$**–Coalg** $\cong D(HD)^\infty$**–Coalg**.

On a seemingly different trajectory, we shall then show that $\langle D, D' \rangle$**–Coalg** is equal to the category $(D \times D')$**–Coalg**, provided the product $D \times D'$ exists. However, instantiated for the special case $D' = H^\infty$, the two categories $\langle D, D' \rangle$**–Coalg** and $\langle H, D \rangle$**–Coalg** are isomorphic, and we have just obtained a Coalgebraic characterisation of the latter. Consequently, under suitable assumptions, we obtain that $\langle H, D \rangle$**–Coalg** $\cong (H^\infty \times D)$**–Coalg** because comonads are determined by their Coalgebras (dually to the case of monads and their Algebras contained in [Mac97]). So, as in the discrete case, the product, though of this time of behaviour *comonads*, provides the appropriate behaviour for modelling HTSs over arbitrary time.

In addition to that, as explained in Section 2.4, the distributive law used in the construction of the comonad $D(HD)^\infty$ induces a lifting $\tilde{D}$ of the comonad D to the category HD**–coalg**. For general reasons—also see Section 2.4—its category of Coalgebras $\tilde{D}$**–Coalg** is isomorphic to $\langle H, D \rangle$**–Coalg** and so, we obtain a second, equivalent characterisation of $\langle H, D \rangle$**–Coalg** in terms of a 'two-level' approach. We will just briefly describe it in a third subsection since it gives a different way to obtain abstract rules for heterogeneous timed processes.

7.2.2.1 A Coalgebraic Characterisation of $\langle B_A, E \rangle$–Coalg

We now turn our attention to constructing a comonad whose category of Coalgebras is isomorphic to $\langle B_A, E \rangle$**–Coalg**. In the following, we will use an arbitrary category $\mathcal{C}$, an endofunctor H and a comonad $D = \langle D, \varepsilon, \delta \rangle$, both on $\mathcal{C}$, in place of **Set**, B_A, and E, respectively, in accordance with the abstract view of $\langle B_A, E \rangle$**–Coalg** as an instance of the pullback category $\langle H, D \rangle$**–Coalg** defined in (7.1).

Proposition 7.11
The natural transformation

$$\ell \;=\; D(HD) \overset{\varepsilon_{HD}}{\Longrightarrow} HD \overset{H\delta}{\Longrightarrow} (HD)D \tag{7.8}$$

is a distributive law of the comonad D over the endofunctor HD.

Proof: We have to show that ℓ respects the structure of D. For the counit ε, we compute

$$\begin{tikzcd}
D(HD) \arrow[r, Rightarrow, "\varepsilon_{HD}"] \arrow[d, Rightarrow, "\varepsilon_{HD}"'] & HD \arrow[r, Rightarrow, "H\delta"] \arrow[dl, equal] \arrow[dr, equal] & (HD)D \arrow[d, Rightarrow, "HD\varepsilon"] \\
HD \arrow[rr, equal] & & HD
\end{tikzcd}$$

and the diagram commutes trivially except for the upper-right triangle, which commutes because of the comonad laws (2.3). For the comultiplication δ, consider the following diagram:

$$\begin{tikzcd}
D(HD) \arrow[rr, Rightarrow, "\varepsilon_{HD}"] \arrow[d, Rightarrow, "\delta_{HD}"'] \arrow[dr, equal] \arrow[drr, Rightarrow, "DH\delta"] & & HD \arrow[rr, Rightarrow, "H\delta"] \arrow[dr, Rightarrow, "H\delta"] & & (HD)D \arrow[d, Rightarrow, "HD\delta"] \\
D^2(HD) \arrow[r, Rightarrow, "D\varepsilon_{HD}"'] & D(HD) \arrow[r, Rightarrow, "DH\delta"'] & D(HD)D \arrow[r, Rightarrow, "\varepsilon_{(HD)D}"'] & (HD)D \arrow[r, Rightarrow, "H\delta_D"'] & (HD^2)D \cong (HD)D^2
\end{tikzcd}$$

where the smaller diagrams commute trivially, by naturality of ε, or the comonad laws (2.3). $\square$

As a consequence of the distributive law, we obtain:

Corollary 7.12
Defining $\tilde{D}\langle X, h : X \to HDX\rangle \ = \ \langle DX, DX \xrightarrow{Dh} DHDX \xrightarrow{\ell_X} HDDX\rangle$ yields a lifting $\tilde{D}$ of the comonad D to HD**–coalg**. $\square$

Having obtained the comonad $\tilde{D}$ on HD**–coalg**, consider the category of its Coalgebras $\tilde{D}$**–Coalg** where a $\tilde{D}$-Coalgebra consists of

- an object X of $\mathcal{C}$ as its carrier
- an HD-coalgebra $h : X \to HDX$, making it an object in HD**–coalg**
- a D-Coalgebra $k : X \to DX$ with the additional property that the following diagram commutes:

(7.9)
$$\begin{tikzcd}
X \arrow[r, "k"] \arrow[ddd, "h"'] & DX \arrow[d, "Dh"] & \\
& DHDX \arrow[dd, "\ell_X"] \arrow[dr, bend left, "\varepsilon_{HDX}"] & \\
& & HDX \arrow[dl, bend left, "H\delta_X"] \\
HDX \arrow[r, "HDk"'] & HDDX &
\end{tikzcd}$$

Thus, $\tilde{D}$-Coalgebras are special pairs of coalgebras satisfying a *coherence condition* with respect to ℓ. Dualising a result from [HPP], we can then prove the following, obtaining a coalgebraic description of $\langle H,D\rangle$**–Coalg**:

Proposition 7.13
The functor $F : \tilde{D}\textbf{–Coalg} \to \langle H,D\rangle\textbf{–Coalg}$ defined by

$$\langle X, X \xrightarrow{h} HDX, X \xrightarrow{k} DX\rangle \ \mapsto \ \langle X, X \xrightarrow{h} HDX \xrightarrow{H\varepsilon_X} HX, X \xrightarrow{k} DX\rangle$$

is an isomorphism of categories $\tilde{D}\textbf{–Coalg} \cong \langle H,D\rangle\textbf{–Coalg}$: we claim that the functor $G : \langle H,D\rangle\textbf{–Coalg} \to \tilde{D}\textbf{–Coalg}$, defined as

$$\langle X, X \xrightarrow{h} HX, X \xrightarrow{k} DX\rangle \ \mapsto \ \langle X, X \xrightarrow{h} HX \xrightarrow{Hk} HDX, X \xrightarrow{k} DX\rangle$$

is its inverse.

Proof: It is clear that F is well-defined and a functor since ε is natural. Moreover, functoriality of G is also obvious, for well-definedness we have to check that this assignment verifies (7.9). For this, consider the following diagram:

$$\begin{tikzcd}
X \arrow[r, bend left, "k"] \arrow[dd, "h"'] & DX \arrow[l, bend left, "\varepsilon_X"] \arrow[d, "Dh"] \arrow[dd, bend left=60, "D(Hk\circ h)"] \\
 & DHX \arrow[dl, "\varepsilon_{HX}"'] \arrow[d, "DHk"] \\
HX \arrow[dd, "Hk"'] & DHDX \arrow[ddl, "\varepsilon_{HDX}"'] \arrow[d, "\varepsilon_{HDX}"] \arrow[dd, bend left=60, "\ell_X"] \\
 & HDX \arrow[d, "H\delta_X"] \\
HDX \arrow[r, bend right, "HDk"'] & HDDX \arrow[l, bend right, "HD\varepsilon_X"']
\end{tikzcd}$$

where the squares in the interior commute because ε is natural and is respected by ℓ. Based on it, we can compute as follows, using functoriality of D, naturality of ε, the fact that k is a Coalgebra, and that ℓ respects ε:

$$HDk \circ (Hk \circ h) = HDk \circ Hk \circ h \circ \mathrm{id}_X$$

$$
\begin{aligned}
&= HDk \circ Hk \circ h \circ \varepsilon_X \circ k \\
&= HDk \circ Hk \circ \varepsilon_{HX} \circ Dh \circ k \\
&= HDk \circ \varepsilon_{HDX} \circ DHk \circ Dh \circ k \\
&= HDk \circ HD\varepsilon_X \circ H\delta_X \circ \varepsilon_{HDX} \circ DHk \circ Dh \circ k \\
&= \mathrm{id}_{HDX} \circ H\delta_X \circ \varepsilon_{HDX} \circ DHk \circ Dh \circ k \\
&= \ell_X \circ DHk \circ Dh \circ k \\
&= \ell_X \circ D(Hk \circ h) \circ k
\end{aligned}
$$

i.e., precisely the equality demanded by (7.9). Consequently, G is well-defined. It remains to show that F and G are mutually inverse.

For this, take a $\tilde{D}$-Coalgebra $\langle X, X \xrightarrow{h} HDX, X \xrightarrow{k} DX\rangle$. Its image under F is

$$\langle X, X \xrightarrow{h} HDX \xrightarrow{H\varepsilon_X} HX, X \xrightarrow{k} DX\rangle$$

which, by G, gets sent to

$$\langle X, X \xrightarrow{h} HDX \xrightarrow{H\varepsilon_X} HX \xrightarrow{Hk} HDX, X \xrightarrow{k} DX\rangle$$

In order to show $G \circ F = \mathrm{Id} : \tilde{D}$**–Coalg** $\to \tilde{D}$**–Coalg**, we need to prove that the two HD-coalgebras are equal, i.e., we need to show $h = Hk \circ H\varepsilon_X \circ h$. This is taken care of in the following diagram:

$$
\begin{tikzcd}
X \arrow[rr, "h"] \arrow[d, "k"] \arrow[dd, bend right=60, "\mathrm{id}_X"'] & & HDX \arrow[r, "H\varepsilon_X"] \arrow[d, "HDk"] & HX \arrow[dd, "Hk"] \\
DX \arrow[r, "Dh"] \arrow[d, "\varepsilon_X"] & DHDX \arrow[dr, "\varepsilon_{HDX}"'] & HDDX \arrow[dr, "H\varepsilon_{DX}"] & \\
X \arrow[rr, "h"'] & & HDX \arrow[u, "H\delta_X"] \arrow[r, equal] & HDX
\end{tikzcd}
\tag{7.10}
$$

where the various diagrams commute either because of the comonad laws (2.3), or because ε is natural, or by the defining property (7.9) of $\tilde{D}$-Coalgebras.

In the other direction, let $\langle X, X \xrightarrow{h} HX, X \xrightarrow{k} DX\rangle$ be an object in $\langle H, D\rangle$**–Coalg**. By G, it gets mapped to

$$\langle X, X \xrightarrow{h} HX \xrightarrow{Hk} HDX, X \xrightarrow{k} DX\rangle$$

which then, applying F, results in

$$\langle X, X \xrightarrow{h} HX \xrightarrow{Hk} HDX \xrightarrow{H\varepsilon_X} HX, X \xrightarrow{k} DX\rangle$$

Since k satisfies the properties (2.5) of a Coalgebra for the comonad D and H is a functor, we have $H\varepsilon_X \circ Hk = \mathrm{id}_{HX}$ and so, $F \circ G = \mathrm{Id} : \langle H, D\rangle$**–Coalg** $\to \langle H, D\rangle$**–Coalg**. □

Thus, we have obtained a first Coalgebraic characterisation of $\langle H, D\rangle$**–Coalg**, albeit for a comonad on HD**–coalg**, not on $\mathcal{C}$. This characterisation forms the basis of the two-level approach described later. It also enables us to finally prove the desired theorem, exploiting the connection between liftings, distributive laws, and composite (co)monads sketched in Section 2.4:

Theorem 7.14
Assume that the cofree comonad $(HD)^\infty$ exists. Then there exists a canonically given comonad structure on the composite comonad $D(HD)^\infty$ such that

$$\langle H, D\rangle\text{–}\mathbf{Coalg} \cong D(HD)^\infty\text{–}\mathbf{Coalg}$$

Proof: We know that $(HD)^\infty$**–Coalg** $\cong HD$**–coalg**. Hence, the comonad $\tilde{D}$, as obtained in Corollary 7.12 from the distributive law ℓ in (7.8), is also a lifting of D to $(HD)^\infty$**–Coalg**. Consequently, as shown in Section 2.4, it induces a distributive law of *comonads* $D(HD)^\infty \Rightarrow (HD)^\infty D$. Finally, the distributive law yields a canonical comonad structure on the composite $D(HD)^\infty$ which, by Proposition 7.13, in addition to general results on composite comonads (see Section 2.4 and [Jac94]), satisfies $D(HD)^\infty$**–Coalg** $\cong \tilde{D}$**–Coalg** $\cong \langle H, D\rangle$**–Coalg**. □

Remark 7.15
Following [HPP], we could now proceed by giving an explicit description of counit and comultiplication of the comonad $D(HD)^\infty$; however, since we will never actually use these data, we omit them here. ◊

Because composition of accessible functors is still accessible, and both B_A and E are accessible (for the former, see [JPT⁺01], for latter, see Proposition 4.16), we obtain from [JPT⁺01, Prop.2.3] that $(B_A E)^\infty$ exists, thus appropriate instantiation yields:

Corollary 7.16 $\langle B_A, E\rangle$**–Coalg** $\cong E(B_A E)^\infty$**–Coalg** □

Hence, we have indeed obtained a comonad on **Set** whose Coalgebras are HTSs.

7.2.2.2 Product Comonads

In this section, based on dualising recent results by Hyland, Plotkin, and Power [HPP], and (the duals of) well-known general results about *Kan extensions* [Dub74, KL97, Mac97], we show that the pullback category $\langle D, D'\rangle$**–Coalg** from (7.7) is equivalent to the category of Coalgebras for the comonad $D \times D'$, provided that the product exists. Combining this with the results from the previous section for the case that $D' = H^\infty$, we actually obtain that $(H^\infty \times D)$**–Coalg** $\cong \langle H, D\rangle$**–Coalg**, if that product exists.

Definition 7.17
Let $\mathcal{A}, \mathcal{B}, \mathcal{C}$ be categories, and let $F : \mathcal{C} \to \mathcal{A}$ and $G : \mathcal{C} \to \mathcal{B}$ be functors. Then the *left Kan extension of F along G* is a functor $\mathrm{Lan}_G(F) : \mathcal{B} \to \mathcal{A}$ together with a natural transformation $\alpha : F \Rightarrow \mathrm{Lan}_G(F)G$ satisfying the universal property that, given a functor $H : \mathcal{B} \to \mathcal{A}$ and a natural transformation $\varphi : F \to HG$, φ uniquely factors through α, meaning that there is a unique natural transformation $\varphi^\sharp : \mathrm{Lan}_G(F) \Rightarrow H$ such that $\varphi = \varphi^\sharp_G \circ \alpha$. ■

Diagrammatically, the situation of the previous definition can be pictured as follows, abbreviating $\mathrm{Lan}_G(F)$ by L:

(7.11)
$$\begin{tikzcd}
\mathcal{C} \arrow[r, "G"] \arrow[dr, "F"'] \arrow[dr, phantom, bend left=10, "\overset{\alpha}{\Rightarrow}"] & \mathcal{B} \arrow[d, "L"] \arrow[d, bend left=60, "H"] \arrow[d, phantom, shift left=3ex, "\overset{\varphi^\sharp_G}{\Rightarrow}"] \\
& \mathcal{A}
\end{tikzcd}
\qquad \overset{\varphi}{\Longrightarrow}$$

Equivalently, the universal property of $\mathrm{Lan}_G(F)$ states that there is the following one-to-one correspondence

(7.12)
$$\frac{F \Rightarrow HG}{\mathrm{Lan}_G(F) \Rightarrow H}$$

Given $\varphi : F \Rightarrow HG$, its corresponding natural transformation is $\varphi^\sharp : \mathrm{Lan}_G(F) \Rightarrow H$ as depicted in (7.11) and obtained by the universal property; in the other direction, given

$\psi : \mathrm{Lan}_G(F) \Rightarrow H$, its corresponding $\psi^\flat : F \Rightarrow HG$ is obtained as

$$\psi^\flat \;=\; F \overset{\alpha}{\Longrightarrow} \mathrm{Lan}_G(F)G \overset{\psi_G}{\Longrightarrow} HG$$

Then $(\cdot)^\sharp$ and $(\cdot)^\flat$ are mutually inverse and natural.

Note that $\alpha = \mathrm{id}^\flat : F \Rightarrow \mathrm{Lan}_G(F)G$ for $\mathrm{id} : \mathrm{Lan}_G(F) \Rightarrow \mathrm{Lan}_G(F)$, and the two sets of data are equivalent—see [Dub74] for the dual result for right Kan extensions. More abstractly, the bijection (7.12) states that $\mathrm{Lan}_G(\_)$ is left adjoint to the 'pre-composition with G'-functor $(\cdot) \circ G$. We will use this quite heavily in the following proofs.

Kan extensions are a very powerful concept, aptly illustrated by the section titled 'All concepts are Kan extensions' in [Mac97]: for instance, it is possible to express the existence of a right adjoint by the existence and preservation of a canonical left Kan extension, similar results hold for colimits, and so on; we refer the reader to [Mac97].

We are only interested in very special left Kan extensions, as will be illustrated now. Given a category $\mathcal{C}$, there is a one-to-one correspondence between objects and arrows in $\mathcal{C}$, and functors $1 \to \mathcal{C}$ with their natural transformations, where 1 denotes the category with exactly one object and its identity arrow. Thus, regarding an object X of $\mathcal{C}$ as such a functor $1 \to \mathcal{C}$, it makes sense to consider the left Kan extension $L \overset{\mathsf{df}}{=} \mathrm{Lan}_X(X)$ of X along itself; if it exists, it is consequently an endofunctor $\mathcal{C} \to \mathcal{C}$.

When expanding the definitions, L comes with its associated natural transformation $\alpha : X \Rightarrow LX$, as shown in the following diagram:

$$\begin{tikzcd}
1 \arrow[r, "X"] \arrow[dr, "X"'] & \mathcal{C} \arrow[d, "L"] \arrow[dl, phantom, "\overset{\alpha}{\Longrightarrow}"] \\
 & \mathcal{C}
\end{tikzcd}$$

Since α is a natural transformation between functors with domain 1, it is simply an arrow in $\mathcal{C}$ of type $X \to LX$, i.e., an L-coalgebra. Accordingly, the universal property of L boils down to the fact that, given an endofunctor $H : \mathcal{C} \to \mathcal{C}$ and an H-coalgebra $\varphi : X \to HX$, there exists a unique natural transformation $\psi : L \Rightarrow H$ such that $\varphi = \psi_X \circ \alpha$, i.e., the following diagram commutes:

$$\begin{tikzcd} X \arrow[rr, "\varphi"] \arrow[dr, "\alpha"'] & & HX \\ & LX \arrow[ur, "\psi_X"'] & \end{tikzcd}$$

Assuming that L exists, we can show that it carries a canonical comonad structure:

Definition 7.18

Let $\mathcal{B}, \mathcal{C}$ be a categories, let $X : \mathcal{B} \to \mathcal{C}$ be a functor such that its left Kan extension $L \stackrel{\text{df}}{=} \mathrm{Lan}_X(X) : \mathcal{C} \to \mathcal{C}$ along itself exists, with natural transformation $\alpha : X \Rightarrow LX$. Define natural transformations $\varepsilon^L : L \Rightarrow \mathrm{Id}$ and $\delta^L : L \Rightarrow L^2$ as follows, using the bijection from (7.12):

$$\frac{X \stackrel{\mathrm{id}}{\Longrightarrow} X}{L \stackrel{\varepsilon^L}{\Longrightarrow} \mathrm{Id}} \tag{7.13}$$

i.e., $\varepsilon = \mathrm{id}_X^\sharp$, and

$$\frac{X \stackrel{\alpha}{\Longrightarrow} LX \stackrel{L\alpha}{\Longrightarrow} LLX}{L \stackrel{\delta^L}{\Longrightarrow} LL} \tag{7.14}$$

i.e, $\delta = (L\alpha \circ \alpha)^\sharp$. ■

It is worthwhile to expand the definitions of ε^L and δ^L in terms of commuting diagrams. Since $\varepsilon^L = \mathrm{id}^\sharp$, as stated in (7.13), we get:

$$\mathrm{id} \;=\; X \stackrel{\alpha}{\Longrightarrow} LX \stackrel{\varepsilon^L_X}{\Longrightarrow} X \tag{7.15}$$

Analogously, from (7.14), we obtain that the following diagram commutes:

$$\begin{tikzcd} X \arrow[r, Rightarrow, "\alpha"] \arrow[d, Rightarrow, "\alpha"'] & LX \arrow[d, Rightarrow, "\delta^L_X"] \\ LX \arrow[r, Rightarrow, "L\alpha"'] & LLX \end{tikzcd} \tag{7.16}$$

With this, we can prove:

Theorem 7.19

Under the same assumptions as in the preceding definition, $\langle L, \varepsilon^L, \delta^L\rangle$ is a comonad.

Proof: To ease notation, we drop the superscripts and simply write ε and δ throughout this proof. We have to show that the comonad laws in (2.3) are satisfied. First, let us show the two triangles. One states that $\varepsilon_L \circ \delta = \mathrm{id}$. Consider the following diagram:

$$\begin{tikzcd}
X \arrow[r, Rightarrow, "\alpha"] \arrow[dr, Rightarrow, "\mathrm{id}"'] & LX \arrow[r, Rightarrow, "L\alpha"] \arrow[d, Rightarrow, "\varepsilon_X"] & LLX \arrow[d, Rightarrow, "\varepsilon_{LX}"] \\
& X \arrow[r, Rightarrow, "\alpha"'] & LX
\end{tikzcd}$$

This diagram commutes: the triangle on the left because of (7.15), the square because ε is natural. Hence, we obtain two equal maps of type $X \Rightarrow LX$. The first, α, is equal to $\mathrm{id}^\flat$ for $id : LX \Rightarrow LX$. For the second, consider the following diagram:

$$\begin{tikzcd}
X \arrow[r, Rightarrow, "\alpha"] \arrow[dr, Rightarrow, "\alpha"'] & LX \arrow[r, Rightarrow, "L\alpha"] & LLX \arrow[r, Rightarrow, "\varepsilon_{LX}"] & LX \\
& LX \arrow[ur, Rightarrow, "\delta_X"] \arrow[urr, Rightarrow, "(\varepsilon_L \circ \delta)_X"'] & &
\end{tikzcd}$$

This commutes because of (7.16) and functoriality of $(\cdot) \circ X$ and so, the second map is equal to $(\varepsilon_L \circ \delta)^\flat$. Since (7.12) is a bijection, $\mathrm{id}^\flat = (\varepsilon_L \circ \delta)^\flat$ implies $\mathrm{id} = \varepsilon_L \circ \delta$ as desired.

For the other triangle in (2.3), stating that $\mathrm{id} = L\varepsilon \circ \delta$, consider the following diagram:

$$\begin{tikzcd}
X \arrow[r, Rightarrow, "\alpha"] \arrow[d, Rightarrow, "\alpha"'] & LX \\
LX \arrow[ur, Rightarrow, "\mathrm{id}"] \arrow[r, Rightarrow, "L\alpha"'] & LLX \arrow[u, Rightarrow, "L\varepsilon_X"']
\end{tikzcd}$$

The upper triangle commutes trivially, the lower one because of (7.15). So again, we get two equal maps $X \Rightarrow LX$, one of which is $\alpha = \mathrm{id}^\flat$. The second can again be rewritten, using (7.16) and functoriality of $(\cdot) \circ X$:

$$\begin{tikzcd}
X \arrow[r, Rightarrow, "\alpha"] \arrow[dr, Rightarrow, "\alpha"'] & LX \arrow[r, Rightarrow, "L\alpha"] & LLX \arrow[r, Rightarrow, "L\varepsilon_X"] & LX \\
& LX \arrow[ur, Rightarrow, "\delta_X"] \arrow[urr, Rightarrow, "(L\varepsilon \circ \delta)_X"'] & &
\end{tikzcd}$$

i.e., we obtain the map $(L\varepsilon \circ \delta)^\flat$. Consequently, we get $\mathrm{id} = L\varepsilon \circ \delta$, showing that also the second triangle commutes.

Finally, for the square in (2.3), consider the following diagram:

$$\begin{tikzcd}
X \arrow[r, Rightarrow, "\alpha"] \arrow[d, Rightarrow, "\alpha"'] & LX \arrow[r, Rightarrow, "L\alpha"] \arrow[d, Rightarrow, "L\alpha"] & LLX \arrow[dd, Rightarrow, "L\delta_X"] \\
LX \arrow[r, Rightarrow, "\delta_X"] \arrow[d, Rightarrow, "L\alpha"'] & LLX \arrow[dr, Rightarrow, "LL\alpha"] & \\
LLX \arrow[rr, Rightarrow, "\delta_{LX}"'] & & LLLX
\end{tikzcd}$$

where the square in the upper left commutes because of (7.16), the square below the diagonal map because δ is natural, and the square above the diagonal map because it is exactly (7.16) with L applied to it.

Using the same kind of argument as above, it follows that the two equal maps of type $X \Rightarrow LLLX$ formed by the exterior of the above square correspond, via (7.12), to the two maps of type $L \Rightarrow LLL$ from the square in (2.3), which therefore commutes. $\square$

Definition 7.20
If $\mathrm{Lan}_X(X)$ exists, we also say that $\mathcal{C}$ *admits* the comonad $\mathrm{Lan}_X(X)$. $\blacksquare$

We will now show that, for any set X, **Set** admits $\mathrm{Lan}_X(X)$. Consider the product $X \times A$ for two sets X and A. Its universal property is that it it is equivalent to give a map $X \times A \xrightarrow{\tau} B$ and to give a map of type $X \xrightarrow{\tau^\sharp} A{\Rightarrow}B$, where $A{\Rightarrow}B$ denotes the set of all functions from A to B; since this is obtained by the cartesian closed structure on **Set**, this bijection is also natural. More conceptually, we actually use the X-fold *copower* $\bigoplus_X A$ of an object A which, in **Set**, happens to be isomorphic to the product $X \times A$; note that such a copower is a special case of the concept of *tensor* from *enriched category theory* [Kel82]. Using this, we can show the following:

Proposition 7.21
For any set X, the functor $\Lambda \stackrel{\mathrm{df}}{=} (X{\Rightarrow}\_) \times X$ is equal to $\mathrm{Lan}_X(X)$.

Proof: It is obvious that the above assignment makes Λ a functor $\mathbf{Set} \to \mathbf{Set}$. We have to show that it satisfies the universal property of the left Kan extension, viz., given an endofunctor $H : \mathbf{Set} \to \mathbf{Set}$, that there is a natural bijection

$$\frac{X \Rightarrow HX}{\Lambda \Rightarrow H} \tag{7.17}$$

So assume there is a natural transformation $\varphi : \Lambda \Rightarrow H$. By definition, this is the same as, for any sets A, B and a function $f : A \to B$, to give a maps $\varphi_A : \Lambda A \to HA$ and $\varphi_B : \Lambda B \to HB$ which satisfy

$$\begin{tikzcd}
(X \Rightarrow A) \times X \arrow[r, "\tau_A"] \arrow[d, "(X \Rightarrow f)"'] & HA \arrow[d, "Hf"] \\
(X \Rightarrow B) \times X \arrow[r, "\tau_B"'] & HB
\end{tikzcd}$$

By using the cartesian closed structure, we get the diagram

$$\begin{tikzcd}
X \Rightarrow A \arrow[r, "\tau_A^\sharp"] \arrow[d, "X \Rightarrow f"'] & X \Rightarrow HA \arrow[d, "X \Rightarrow Hf"] \\
X \Rightarrow B \arrow[r, "\tau_B^\sharp"'] & X \Rightarrow HB
\end{tikzcd}$$

which commutes because the involved bijection is natural, as remarked above. Consequently, we obtain a natural transformation $\tau^\sharp : (X \Rightarrow \_) \Rightarrow (X \Rightarrow H\_)$ or equivalently, using the hom-set notation instead of function spaces,

$$\tau^\sharp : \mathbf{Set}(X, \_) \Rightarrow \mathbf{Set}(X, H\_)$$

However, applying the *Yoneda Lemma* [Mac97], such natural transformations are in one-to-one correspondence with elements of the set $\mathbf{Set}(X, HX)$, i.e., the set of all H-coalgebras with carrier X. Finally, these are equivalent to natural transformations $X \Rightarrow HX$, regarding X as a functor of type $1 \to \mathbf{Set}$. Thus, since all the bijections involved are natural, we precisely obtain the desired bijection (7.17), showing that Λ is indeed equal to $\mathrm{Lan}_X(X)$. □

Consequently, all the comonads $\mathrm{Lan}_X(X)$ exist in $\mathbf{Set}$; again, we omit an explicit description of the comonad structure since only its existence is important to us. In order to continue our path towards establishing the aforementioned product comonad as the appropriate behaviour comonad for HTSs, we need to introduce the following notion of co-action of a comonad which, as we shall see, generalises the notion of Coalgebra for a comonad:

Definition 7.22
A natural transformation $\sigma : H \Rightarrow DH$ verifying the following diagram

(7.18)
$$\begin{tikzcd} & H \arrow[r, Rightarrow, "\sigma"] \arrow[dl, equal] \arrow[d, Rightarrow, "\sigma"] & DH \arrow[d, Rightarrow, "D\sigma"] \\ H & DH \arrow[l, Rightarrow, "\varepsilon_H"] \arrow[r, Rightarrow, "\delta_H"'] & D^2H \end{tikzcd}$$

is called a *co-action* of the comonad D on the endofunctor H. ■

Instantiating the last definition for functors $X : 1 \to \mathcal{C}$, we obtain:

Proposition 7.23
Given a comonad D on $\mathcal{C}$ and and object X of $\mathcal{C}$, there is a one-to-one correspondence between D-Coalgebras $X \to DX$ and co-actions $X \Rightarrow DX$ of D on $X : 1 \to \mathcal{C}$.

Proof: As previously remarked, a natural transformation $\sigma : X \Rightarrow DX$ has precisely one component, viz., a map $k : X \to DX$, and vice versa. The conditions on σ imposed by (7.18) correspond exactly to the fact that k verifies the two diagrams in (2.5). □

The use of these co-actions will become clear in the next theorem, which refines the bijection (7.12), stating that the comonads $\mathrm{Lan}_X(X)$ have a universal property:

Theorem 7.24
Let $\mathcal{C}$ be a category and, given an object X of $\mathcal{C}$, assume that $L = \mathrm{Lan}_X(X)$ exists; write $\alpha : X \Rightarrow LX$ for its associated natural transformation, and denote its comonad structure by ε^L and δ^L. Let, furthermore, $D = \langle D, \varepsilon, \delta\rangle$ be a comonad on $\mathcal{C}$. Then there is a one-to-one correspondence between co-actions $X \Rightarrow DX$ of D on X and comonad morphisms $L \Rightarrow D$.

Proof: Following the proof of [Dub74, Prop. II.1.4], let $\varphi : L \Rightarrow D$ be natural transformation. That the corresponding map $\varphi^\flat : X \Rightarrow DX$ is a co-action of D on X is expressed in the following two diagrams:

(7.19)
$$\begin{tikzcd}
X \arrow[rr, Rightarrow, "\mathrm{id}"] \arrow[dr, Rightarrow, "\varphi^\flat"'] & & X \\
& DX \arrow[ur, Rightarrow, "\varepsilon_X"'] &
\end{tikzcd}$$

(7.20)
$$\begin{tikzcd}
X \arrow[rrrr, Rightarrow, "\varphi^\flat"] \arrow[dr, Rightarrow, "\alpha"'] \arrow[drr, Rightarrow, "(\delta^L)^\flat"] \arrow[ddd, Rightarrow, "\varphi^\flat"'] & & & & DX \arrow[ddd, Rightarrow, "\delta_X"] \\
& LX \arrow[l, phantom, "(*)"] \arrow[r, Rightarrow, "L\alpha"'] \arrow[ddl, Rightarrow, "\varphi_X"'] & LLX \arrow[d, Rightarrow, "\varphi_{LX}"'] \arrow[ddrr, Rightarrow, "(\varphi\varphi)_X"] & (\diamond) & \\
& (+) & DLX \arrow[d, phantom, "(*)"] \arrow[drr, Rightarrow, "D\varphi_X"] & & \\
DX \arrow[urr, Rightarrow, "D\alpha"'] \arrow[rrrr, Rightarrow, "D\varphi^\flat"'] & & & & DDX
\end{tikzcd}$$

In (7.20), the triangles marked with $(*)$ commute because of the definition of $(\cdot)^\flat$; the square marked with $(+)$ and the triangle on its right commute because φ is natural, and the triangle above it because of the definition of δ^L as $(L\alpha \circ \alpha)^\sharp$, and the fact that $(\cdot)^\sharp$ and $(\cdot)^\flat$ are mutually inverse. Consequently, (7.20) commutes if and only if the following diagram, above marked with $(\diamond)$, commutes:

(7.21)
$$\begin{tikzcd}
X \arrow[r, Rightarrow, "\varphi^\flat"] \arrow[d, Rightarrow, "(\delta^L)^\flat"'] & DX \arrow[d, Rightarrow, "\delta_X"] \\
LLX \arrow[r, Rightarrow, "(\varphi\varphi)_X"'] & DDX
\end{tikzcd}$$

In order for φ to be comonad morphism, it has to satisfy the diagrams

(7.22)
$$\begin{tikzcd}
L \arrow[rr, Rightarrow, "\varepsilon^L"] \arrow[dr, Rightarrow, "\varphi"'] & & \mathrm{Id} \\
& D \arrow[ur, Rightarrow, "\varepsilon"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
L \arrow[r, Rightarrow, "\varphi"] \arrow[d, Rightarrow, "\delta^L"'] & D \arrow[d, Rightarrow, "\delta"] \\
LL \arrow[r, Rightarrow, "\varphi\varphi"'] & DD
\end{tikzcd}$$

By the adjunction $\mathrm{Lan}_X(\_) \dashv (\cdot) \circ X$, it follows:

$$\varepsilon_X \circ \varphi^\flat = (\varepsilon \circ \varphi)^\flat$$
$$\delta_X \circ \varphi^\flat = (\delta \circ \varphi)^\flat$$
$$(\varphi\varphi)_X \circ (\delta^L)^\flat = ((\varphi\varphi) \circ \delta^L)^\flat$$

Together with the fact that $(\varepsilon^L)^\flat = (\mathrm{id}^\sharp)^\flat = \mathrm{id}$, it now follows that (7.19) commutes if and only if the triangle in (7.22) commutes and analogously, (7.21) commutes if and only if the square in (7.22) commutes. In other words: φ is a comonad morphism if and only if $\varphi^\flat$ is a co-action, concluding the proof. □

Theorem 7.25
Let $\mathcal{C}$ be a category and assume that the product $D \times D'$ of the comonads D and D' in $\mathbf{Cmd}(\mathcal{C})$ exists. Then, for an object X of $\mathcal{C}$, if $\mathrm{Lan}_X(X)$ exists, it is equivalent to give a $(D \times D')$-Coalgebra on X, and a pair of Coalgebras $DX \leftarrow X \rightarrow D'X$.

Proof: Let $k : X \to (D \times D')X$ be a $(D \times D')$-Coalgebra. By Proposition 7.23, it is the same as a co-action $X \Rightarrow (D \times D')X$ of $(D \times D')$ on $X : 1 \to \mathcal{C}$. By Theorem 7.24, such a co-action is equivalent to a comonad morphism $\mathrm{Lan}_X(X) \Rightarrow D \times D'$. Since $D \times D'$ is the product of D and D' in $\mathbf{Cmd}(\mathcal{C})$, its universal property states that any map into it can equivalently be regarded as two comonad morphisms with respective types $\mathrm{Lan}_X(X) \Rightarrow D$ and $\mathrm{Lan}_X(X) \Rightarrow D'$. Reversing the previous steps, this is first equivalent to two co-actions of D and D', respectively, on X, and then to two Coalgebras $X \to DX$ and $X \to D'X$. □

Moreover, since all the involved constructions are functorial, and by the definition of $\langle D, D' \rangle$**–Coalg**, we obtain:

Corollary 7.26
There is an isomorphism of categories

$$D \times D'\text{–}\mathbf{Coalg} \cong \langle D, D' \rangle\text{–}\mathbf{Coalg}$$

provided that $D \times D'$ exists, and that $\mathcal{C}$ admits all the comonads $\mathrm{Lan}_X(X)$. □

Combining the last theorem with the results of the previous section, we obtain:

Corollary 7.27 $E(B_A E)^\infty \cong B_A^\infty \times E$ in $\mathbf{Cmd}(\mathbf{Set})$, and consequently

$$(B_A^\infty \times E)\text{–}\mathbf{Coalg} \cong \langle B_A, E \rangle\text{–}\mathbf{Coalg}$$

stating that HTSs over $\langle A, \mathcal{T} \rangle$ are Coalgebras for the product comonad $B_A^\infty \times E$. Moreover, coalgebraic bisimulation for $B_A^\infty \times E$ is exactly heterogeneous bisimulation.

Proof: By Proposition 7.21, all the comonads $\mathrm{Lan}_X(X)$ exist in **Set**, hence, applying Corollary 7.26, we obtain

$$\langle B_A, E\rangle\text{–}\mathbf{Coalg} \cong \langle B_A^\infty, E\rangle\text{–}\mathbf{Coalg} \cong (B_A^\infty \times E)\text{–}\mathbf{Coalg}$$

under the assumption that this product exists. However, Corollary 7.16 states that

$$\langle B_A, E\rangle\text{–}\mathbf{Coalg} \cong E(B_A E)^\infty\text{–}\mathbf{Coalg}$$

so there already *exists* a comonad whose Coalgebras are isomorphic to $\langle B_A, E\rangle$–**Coalg** and, since comonads are determined by their Coalgebras (see [Mac97] for the dual case of monads), it must therefore be the case that

$$E(B_A E)^\infty \cong B_A^\infty \times E$$

in $\mathbf{Cmd}(\mathbf{Set})$, implying the isomorphism between the Coalgebras. The statement about coalgebraic bisimulation then follows trivially. $\square$

Remark 7.28

1. It should be obvious that, under the appropriate assumptions, the last result can be generalised to obtain a characterisation of the product of H^∞ and D for an endofunctor H and a comonad D on some category $\mathcal{C}$, for instance, assuming that $\mathcal{C}$ admits all the comonads $\mathrm{Lan}_X(X)$ and that both H and D are accessible. For the case of two arbitrary comonads, it is not clear what the right assumptions are such that their product always exists. However, if it exists, and additionally also all the comonads $\mathrm{Lan}_X(X)$, the characterisation from Theorem 7.25 is valid, and so the category of Coalgebras for the product comonad is isomorphic to the pullback category $\langle D, D'\rangle$–**Coalg** introduced in (7.7).

2. When also $D = B^\infty$ holds, e.g., in the case of discrete time (for $\mathcal{C} = \mathbf{Set}$, $H = B_A$ and $B = B_{\mathbb{N}}$, i.e., $B_{\mathbb{N}}^\infty = E_{\mathbb{N}}$), it follows that

$$H^\infty \times B^\infty \cong (H \times B)^\infty$$

with $\times$ on the left denoting the product of comonads, and on the right the product of functors, provided that $\mathcal{C}$ admits all the comonads $\mathrm{Lan}_X(X)$, and that both $H^\infty \times B^\infty$ and $(H \times B)^\infty$ exist. In particular, by Proposition 7.21, we get that

$$B_A^\infty \times E_{\mathbb{N}} \cong B_A^\infty \times B_{\mathbb{N}}^\infty \cong (B_A \times B_{\mathbb{N}})^\infty$$

and hence, for discrete time, the two approaches—using either the product functor, or the product comonad—coincide since their categories of coalgebras are isomorphic. $\Diamond$

Corollary 7.29
The comonad $B_A^\infty \times E$ preserves weak pullbacks.

Proof: By the preceding corollary, $B_A^\infty \times E \cong E(B_A E)^\infty$. Furthermore, it is well-known that B_A preserves weak pullbacks (see, e.g., [Tur96]), so does E (see Corollary 4.15). Moreover, by [JPT$^+$01, Lemma 2.8], the forgetful functor U from $B_A E$**–coalg** to **Set** preserves weak pullbacks; hence, so does $(B_A E)^\infty$: the right adjoint to U automatically preserves all limits, and the composition with the forgetful functor is necessarily equal to the cofree comonad $(B_A E)^\infty$. Hence, $B_A^\infty \times E$ is isomorphic to a composition of functors, all of which preserve weak pullbacks, and so the claim follows. $\Box$

7.2.2.3 A Two-Level Approach

In this section, we briefly show the details of a different, but equivalent approach to obtain an appropriate behaviour for HTSs. Given a category $\mathcal{C}$, an endofunctor H and a comonad D, both on $\mathcal{C}$, Section 7.2.2.1 presented the distributive law (7.8) of the comonad D over the *functor* HD. Consequently, we obtained a lifting $\tilde{D}$ of the comonad D to HD**–coalg** such that, by Proposition 7.13, $\tilde{D}$**–Coalg** $\cong \langle H, D\rangle$**–Coalg**.

Instantiating this with B_A and E in place of H and D, respectively, the distributive law lifts E to the comonad $\tilde{E}$ on $B_A E$**–coalg** and we obtain:

Corollary 7.30 $\tilde{E}$**–Coalg** $\cong \langle B_A, E\rangle$**–Coalg** $\Box$

Consequently, by Proposition 7.4, $\tilde{E}$-Coalgebras are the same as HTSs. Recall the definition of an $\tilde{E}$-Coalgebra; it consists of three parts:

- a set X as the carrier,
- a $B_A E$-coalgebra $h : X \to B_A EX$, and
- an E-Coalgebra $k : X \to EX$, i.e., a TTS on X

which additionally satisfy the following ‘coherence condition’:

(7.23)
$$\begin{tikzcd}
X \arrow[r, "k"] \arrow[ddd, "h"'] & EX \arrow[d, "Eh"] & \\
& EB_AEX \arrow[dd, "\ell_X"] \arrow[dr, bend left, "\varepsilon_{B_AEX}"] & \\
& & B_AEX \arrow[dl, bend left, "B_A\delta_X"] \\
B_AEX \arrow[r, "B_AEk"'] & B_AEEX &
\end{tikzcd}$$

Thus, any $\tilde{E}$-Coalgebra ‘contains’ a B_AE-coalgebra h and as such, for $x \in X$, $h(x)$ is equivalent to a map of type $A \to \mathcal{P}_{\mathrm{fi}}(EX)$ such that additionally the two axioms of evolutions (4.1) and (4.2) hold for each evolution e in $h(x)(\alpha)$. This means that h defines *combined* transitions with two labels on X as follows:

$$x \xrightarrow[t]{\alpha} x' \overset{\mathrm{df}}{\Leftrightarrow} (\exists e \in h(x)(\alpha)).\ e \overset{t}{\rightsquigarrow} x'$$

for $t \in \mathcal{T}$ and $\alpha \in A$. Looking at the definition, a slightly different notation for the combined transitions could be $x \overset{\alpha}{\to} \overset{t}{\rightsquigarrow} x'$, putting more emphasis on the fact that first, the B_A-coalgebra is evaluated, and only then the E-coalgebra, as specified by the composite functor B_AE.

Since a $\tilde{E}$-Coalgebra additionally ‘contains’ an E-Coalgebra, which defines *pure* time transitions, one would like the two notions of transitions to be related in a sensible way which is exactly the purpose of the diagram (7.23). In order to ‘decode’ the diagram, note the identity

$$h = \varepsilon_{B_AEX} \circ Eh \circ k$$

obtained from (7.10), and so (7.23) really just states

$$\begin{tikzcd}
X \arrow[r, "h"] \arrow[d, "h"'] & B_AEX \arrow[d, "B_A\delta_x"] \\
B_AEX \arrow[r, "B_AEk"'] & B_AEEX
\end{tikzcd}$$

Translating this into the transition-view, we obtain

(7.24)
$$x \xrightarrow[t]{\alpha} x' \wedge x' \overset{u}{\rightsquigarrow} x'' \ \Leftrightarrow\ x \xrightarrow[t+u]{\alpha} x''$$

where the left-hand side of the equivalence corresponds to the composite $B_A E k \circ h$, the right-hand side to $B_A \delta_X \circ h$. We call the equivalence (7.24) *action continuity* because it is essentially an 'action-decorated' version of axiom (Continuity). Moreover, we can define

$$x \xrightarrow{\alpha} x' \overset{\text{df}}{\Leftrightarrow} x \xrightarrow[0]{\alpha} x'$$

Note that this is exactly the action of the functor F from Proposition 7.13 which, in this particular case, maps an $\tilde{E}$-Coalgebra to an object in $\langle B_A, E\rangle$**–Coalg**, by turning the $B_A E$-coalgebra part into a B_A-coalgebra.

Then, instantiating (7.24) with the trivial identity $0 + t = t$ in $\mathcal{T}$, we obtain that

$$x \xrightarrow[t]{\alpha} x' \;\Leftrightarrow\; x \xrightarrow{\alpha} x'' \wedge x'' \overset{t}{\rightsquigarrow} x' \tag{7.25}$$

i.e., the combined transitions really capture the intuition of describing actions which are no longer necessarily instantaneous, but may take some time $t \in \mathcal{T}$.

Remark 7.31

It is interesting to note that the operational semantics of TiCSP described in [Sch95] uses precisely the two kinds of transitions described by a $\tilde{E}$-Coalgebra. The transition system obtained by the pure time transitions $\overset{t}{\rightsquigarrow}$, incidentally called *evolutions*, is shown to be a TTS, i.e., an E-Coalgebra on the set of terms. However, no result corresponding to (7.24) is obtained. This is, in our opinion, due to the fact that a combined transition $\xrightarrow[t]{\alpha}$ is interpreted as $\overset{t}{\rightsquigarrow}\xrightarrow{\alpha}$, i.e., the time transition is performed *before* the action transition, rather than the other way around, which ideally should validate the symmetric property to (7.25). In order to account for this 'inverted' interpretation, one would have to switch to using EB_A-coalgebras, rather than the $B_A E$-coalgebras we use here; however, we do not see how to obtain a distributive law of E over EB_A, or a lifting of E to EB_A**–coalg**. The only connection between combined and pure transitions that is established in the article is an equivalent formulation of (Persistency):

$$(p \xrightarrow[0]{a} \wedge (\exists p').\, p \overset{t}{\rightsquigarrow} p') \Leftrightarrow p \xrightarrow[t]{a}$$

which, due to the inverted interpretation, is *not* equivalent to (7.25), despite the superficial similarities: in fact, it cannot be equivalent, since the two-level approach is equivalent to using HTSs, and so definitely does not validate (Persistency). $\Diamond$

Having established this very pleasing and intuitive connection between combined $\xrightarrow[t]{\alpha}$-transitions and pure $\overset{u}{\rightsquigarrow}$-transitions in the two-level approach, we would still like to stress that the comonad $\tilde{E}$ 'lives' on the category $B_A E$**–coalg**. This will become important when defining abstract operational rules, as we shall do in the next section.

7.3 Heterogeneous Abstract Rules

We have found different coalgebraic characterisations of HTSs, one given by the product *functor* $B_A \times B_{\mathbb{N}}$ in the case $\mathcal{T} = \mathbb{N}$, where E is cofreely generated, the other given by the product *comonad* $B_A^\infty \times E$, in the general case, which can additionally be characterised by the two-level approach. Accordingly, we can use different approaches to obtaining abstract operational rules: the previous approach of [TP97] for discrete time, and our approach as described in Chapter 5 for the general case.

7.3.1 Discrete Time

Since we have seen in Proposition 7.5 that HTSs over $\langle A, \mathbb{N}\rangle$ are the same as coalgebras for the product functor $B_A \times B_{\mathbb{N}}$, we can simply instantiate the theory of [TP97] for describing abstract rules for heterogeneous timed processes over discrete time, i.e., by using natural transformations

$$\rho : \Sigma(\mathrm{Id} \times (B_A \times B_{\mathbb{N}})) \Rightarrow (B_A \times B_{\mathbb{N}})T \tag{7.26}$$

where Σ is a signature (functor) with freely generated term monad T on **Set**. Furthermore, we have that $(B_A \times B_{\mathbb{N}})T = B_A T \times B_{\mathbb{N}} T$ and so, using the universal property of the product, we can decompose (7.26) into two maps:

$$\rho_A : \Sigma(\mathrm{Id} \times (B_A \times B_{\mathbb{N}})) \Rightarrow B_A T \quad \text{and} \quad \rho_{\mathbb{N}} : \Sigma(\mathrm{Id} \times (B_A \times B_{\mathbb{N}})) \Rightarrow B_{\mathbb{N}} T$$

where ρ_A describes the action transitions, and $\rho_{\mathbb{N}}$ describes the time transitions (in the form of single-step $\rightsquigarrow$-transitions).

Note that, in this fashion, both ρ_A and $\rho_{\mathbb{N}}$ contain a description of the *complete* behaviour of argument processes in the premises, in the sense that the action and time transitions, respectively, of compound processes can depend on both the action and the time transitions of their components. However, we immediately obtain:

Proposition 7.32
Given two natural transformations, one of type $\varphi : \Sigma(\mathrm{Id} \times B_A) \Rightarrow B_A T$ and another one of type $\psi : \Sigma(\mathrm{Id} \times B_{\mathbb{N}}) \Rightarrow B_{\mathbb{N}} T$, we obtain a natural transformation (7.26).

Proof: With the given φ and ψ, define natural transformations

$$\rho_A \stackrel{\mathrm{df}}{=} \Sigma(\mathrm{Id} \times (B_A \times B_{\mathbb{N}})) \stackrel{\Sigma(\mathrm{Id}\times\pi_1)}{\Longrightarrow} \Sigma(\mathrm{Id} \times B_A) \stackrel{\varphi}{\Rightarrow} B_A T$$
$$\rho_{\mathbb{N}} \stackrel{\mathrm{df}}{=} \Sigma(\mathrm{Id} \times (B_A \times B_{\mathbb{N}})) \stackrel{\Sigma(\mathrm{Id}\times\pi_2)}{\Longrightarrow} \Sigma(\mathrm{Id} \times B_{\mathbb{N}}) \stackrel{\psi}{\Rightarrow} B_{\mathbb{N}} T$$

where π_1 and π_2 denote the first and second projections, respectively. Since pairs of natural transformations like ρ_A and $\rho_{\mathbb{N}}$ are in one-to-one correspondence with natural transformation (7.26), the claim follows. □

Proposition 7.32 corresponds to the case that action and time rules of a language are independent: this becomes clear by the fact that φ and ψ only take into account the same kind of behaviour of arguments that is defined for compound terms. However, this need not be the case, the types of ρ_A and $\rho_{\mathbb{N}}$ are general enough to accommodate languages where the two sets of rules are defined mutually depending on each other. In fact, we can model the complete calculi (discrete-time) TiCCS [Wan90], and TPL [HR95].

As a consequence of the general theory from [TP97], also using that coalgebraic bisimulation for $B_A \times B_{\mathbb{N}}$ is heterogeneous bisimulation (see Corollary 7.7), and that $B_A \times B_{\mathbb{N}}$ preserves weak pullbacks (see Corollary 7.9), we obtain:

Proposition 7.33
If the action and time rules of a language can be described by a natural transformation as in (7.26), heterogeneous bisimulation is congruence for the language. □

Example 7.34

1. The action and time rules of the language ATP are covered by Proposition 7.32 (the action rules are GSOS, the time rules is dsl-format, cf. Chapter 6), hence they induce a natural transformation of type (7.26). Consequently, heterogeneous bisimulation is a congruence for ATP.

2. The same holds for the single-step version of TeCCS from Section 6.1.1: the action rules are all GSOS rules, the time rules are in dsl-format. Hence, heterogeneous bisimulation is a congruence. Since, furthermore, by Proposition 6.20, the single-step rules induce the same TTS on the terms of the language, and coalgebraic bisimulation for $B_A \times B_{\mathbb{N}}$ is—see Corollary 7.7—the same as heterogeneous bisimulation, we also obtain the corresponding result for the original version of TeCCS over discrete time. $\Diamond$

Both the above examples are languages where the two sets of rules are independent. We now consider the languages where this is no longer the case, viz., TPL [HR95], and TiCCS [Wan90] over discrete time (for which it is again possible to define a single-step version, very similar to the one for TeCCS). In both calculi, the only operator whose definition actually requires using both action and time transitions in the premises is the parallel operator; its (only) time rule looks as follows:

$$\frac{p \leadsto p' \quad q \leadsto q' \quad p|q \not\xrightarrow{\tau}}{p|q \leadsto p'|q'} \tag{7.27}$$

Note that for TiCCS, the condition on *timed sorts* also degenerates to the statement that p and q, when running in parallel, cannot produce a $\xrightarrow{\tau}$-transition; this is in line with both languages adopting (Maximal Progress), expressing that $\xrightarrow{\tau}$-transitions and time transitions cannot occur together in the same state. Also note that the rule (7.27) only tests for the *absence* of certain action transitions in the premises, i.e., it does not even exploit the full potential given by the combined approach, viz., define the targets of the conclusion by using targets of both action and time transitions.

By the definition of the operational semantics of parallel composition, there are three cases in which a $\xrightarrow{\tau}$-transition of the process $p|q$ can arise: p or, symmetrically, q can themselves already produce such a transition, or there exists an action $a \in (A \setminus \{\tau\})$ such that p can perform an $\xrightarrow{a}$-transition and q can perform a $\xrightarrow{\overline{a}}$-transition. Hence, for a set X of (rule) variables, define the predicate ntau_X, for $\beta, \beta' \in B_A X$, by

$$\mathrm{ntau}_X\langle\beta,\beta'\rangle = \mathtt{tt} \overset{\mathrm{df}}{\Leftrightarrow} \beta(\tau) = \beta'(\tau) = \emptyset \wedge (\forall a \in (A \cup \overline{A}) \setminus \{\tau\}).\ \beta(a) = \emptyset \vee \beta'(\overline{a}) = \emptyset$$

The predicate $\mathrm{ntau}\langle\beta,\beta'\rangle$ consequently holds if and only if the parallel composition of (processes with their action transitions specified by) β and β' cannot perform any $\xrightarrow{\tau}$-transition.

Using ntau_X, (7.27) translates into a map $[\![|]\!]_X : (X \times B_A X \times B_{\mathbb{N}} X)^2 \to B_{\mathbb{N}} TX$ as follows

$$(7.28) \qquad \langle\langle x_1, \beta_1^A, \beta_1^{\mathbb{N}}\rangle, \langle x_2, \beta_2^A, \beta_2^{\mathbb{N}}\rangle\rangle \mapsto \begin{cases} \beta_1^{\mathbb{N}} | \beta_2^{\mathbb{N}} & \text{if } \beta_1^{\mathbb{N}} \neq \star \wedge \beta_2^{\mathbb{N}} \neq \star \wedge \mathrm{ntau}_X \langle \beta_1^A, \beta_2^A\rangle \\ \star & \text{otherwise} \end{cases}$$

We then obtain:

Proposition 7.35 The map $[\![|]\!]_X$ is natural in X.

Proof: This follows since any renaming function $f : X \to Y$ does not affect the 'type' of the arguments, i.e., if, say, $\beta_1^{\mathbb{N}} \neq \star$ then also $B_{\mathbb{N}} f(\beta_1^{\mathbb{N}}) \neq \star$; the analogous statement applies in particular to the B_A-components of the arguments and so, the value of ntau is not affected by the renaming, and the appropriate naturality square commutes. □

Since the other operators can be treated pretty much like in the cases of ATP and TeCCS, we obtain:

Corollary 7.36

Heterogeneous bisimulation is a congruence for TPL and (single-step) TiCCS. □

Remark 7.37

Despite these encouraging positive results for languages with non-independent rules, the problem remains that, in our opinion, a different model already on the level of transition systems should be used for such languages which adopt, e.g., (Maximal Progress) or (Persistency), cf. Remark 7.6. The current solution, somewhat artificially, forces the resulting HTS on programs to satisfy (Maximal Progress) by appropriately defining the *operational semantics*, whereas the underlying HTSs, a priori, do not have to satisfy this property (although not explicitly prohibiting such models either). While we can reluctantly tolerate this discrepancy for discrete time (after all, it works, because of only using one-step behaviours given by behaviour functors), we shall see that, for a general time domain, we do not know how to define appropriate abstract rules as to account for non-independent rules. ◊

7.3.2 The General Case

After presenting abstract rules for HTSs over discrete time, we now consider the general case of an arbitrary time domain. More abstractly, we consider the case of $\mathcal{C}$ being an arbitrary category which admits all the comonads $\mathrm{Lan}_X(X)$, and two comonads D and D' in $\mathbf{Cmd}(\mathcal{C})$ such that their product comonad $D \times D'$ exists. Consequently, as was shown in Section 7.2.2.2, $D \times D'$**–Coalg** is isomorphic to $\langle D, D' \rangle$**–Coalg**, i.e., a $D \times D'$-Coalgebra is the same as a pair consisting of a Coalgebra for D and one for D', respectively, with the same carrier (cf. the definition of $\langle D, D' \rangle$**–Coalg**).

This might be considered very (in fact, too) general: actually, we are only interested in the equivalence between HTSs and $E(B_A E)^\infty$-Coalgebras, based on the characterisation of $B_A^\infty \times E$ as $E(B_A E)^\infty$. However, arguably, nothing can be gained from using the concrete description: what we were able to prove, we proved in all generality; where we did not succeed, we do not see any benefit arising from considering the specific case[3]. This should become more palpable when running into concrete difficulties in the course of this (and later) section(s).

Following the general theory from [TP97], given a monad T (for syntax, usually freely generated by a signature Σ), we want to derive a distributive law

$$T(D \times D') \Rightarrow (D \times D')T \tag{7.29}$$

induced by (a) 'simpler' natural transformation(s). Instantiating the results from Section 5.3 with the behaviour comonad $D \times D'$, we can use abstract temporal rules for $D \times D'$, i.e., a natural transformation of type

$$\Sigma(D \times D') \Rightarrow (D \times D')(\mathrm{Id} + \Sigma)$$

which induces a distributive law like in (7.29) if it respects the structure of $D \times D'$. Alternatively, one could use CSOS rules for $D \times D'$, i.e., a natural transformation

$$\Sigma(D \times D') \Rightarrow (D \times D')T$$

which, again, has to respect the structure of $D \times D'$.

[3]Sure, there might ways to use properties of E and B_A, exploiting the existence of maps of specific types to close diagrams, but all to no avail: such maps might syntactically make diagrams commute but semantically, they do not seem to make sense.

The drawback of this approach is that the comonad structure on $D \times D'$ is rather complex and so, despite the deceptively easy categorical formulation, showing that already existing languages induce such abstract rules, let alone deriving concrete syntactic rule formats, becomes essentially infeasible. In particular, consider instantiating the situation with $D = E$ and $D' = B_A^\infty$: by Corollary 7.27, the product comonad $B_A^\infty \times E$ is isomorphic to $E(B_A E)^\infty$. This is not exactly easy to handle: an element of $E(B_A E)^\infty X$ is an evolution over $(B_A E)^\infty X$, which in turn corresponds to a (potentially infinite) sequence of combined transitions between states in X. Moreover, inspecting the proof of Theorem 7.14 to see how the comonad structure on $E(B_A E)^\infty$ is derived, the respective δ-diagram for either type of abstract rules becomes very hard to check.

Therefore, rather than instantiating the abstract framework with the complex product comonad, one would rather deal with abstract rules for D and for D' separately, in accordance with languages for timed processes: there, one commonly has two sets of rules, one for action transitions, another one for time transitions. Analogously to the situation for discrete time, if these two sets of rules are independent, one can derive rules for the product comonad as follows, for the sake of simplicity formulated in terms of liftings:

Theorem 7.38
Given liftings $\tilde{T}$ and $\tilde{T}'$ of T to D–**Coalg** and D'–**Coalg**, respectively, one obtains a lifting of T to $\langle D, D' \rangle$–**Coalg** $\cong (D \times D')$–**Coalg**.

Proof: Take an object $\langle X, k : X \to DX, k' : X \to D'X \rangle$ in $\langle D, D' \rangle$–**Coalg**. The liftings $\tilde{T}$ and $\tilde{T}'$ provide a way to lift both Coalgebras separately, and so we obtain that

$$\langle TX, \tilde{T}(k) : TX \to DTX, \tilde{T}'(k') : TX \to D'TX \rangle$$

is, in fact, an object in $\langle D, D' \rangle$–**Coalg**: the liftings map Coalgebras to Coalgebras. The operations of T also lift since they lift separately in each component, concluding the proof. $\square$

Since liftings of a monad T to a category of Coalgebras are equivalent to distributive laws of monads over comonads (see Section 2.4), we obtain:

Corollary 7.39

Given distributive laws $\ell : TD \Rightarrow DT$ and $\ell' : TD' \Rightarrow D'T$, we obtain a distributive law of T over $D \times D'$ as in (7.29). $\square$

Remark 7.40

In principle, it should be possible to derive this result directly by using left Kan extensions and their relation to (maps of) comonads (as exemplified in Theorem 7.24), being careful about the necessary conditions guaranteeing existence of the relevant Kan extensions. However, as the proof becomes almost completely trivial when using liftings, we prefer the formulation in terms of liftings, making up in simplicity for what might be lost in terms of elegance and abstraction. $\lozenge$

So, as in the case of discrete time, independent rules can simply be combined to obtain well-behaved abstract rules; for this, note that the product comonad $B_A^\infty \times E$, by Corollary 7.29, preserves weak pullbacks:

Corollary 7.41

If the action rules of a language are given by abstract operational rules as in (5.4), and the times rules independently by either abstract temporal rules or CSOS rules, heterogeneous bisimulation is a congruence. $\square$

These conditions hold for TeCCS (now considered over an arbitrary time domain), so finally, we obtain the congruence result from [MT90], analogously for the more general version ATP_D of ATP presented in [NSY93], where D is a time domain according to their definition, which then also makes it a time domain in our sense (see Section 3.1.3):

Corollary 7.42

Heterogeneous bisimulation is a congruence for TeCCS and for ATP_D. $\square$

However, when trying to obtain abstract rules for non-independent sets of rules, the following problem arises. Inspired by calculi for time processes, consider the case that one set of rules is completely self-contained, while the other set depends in its premises on the first one. More abstractly, we have a distributive law $\ell : TD \Rightarrow DT$ for the self-contained part of the rules, and we would like to use a natural transformation of type

(7.30) $$\ell' : T(D \times D') \Rightarrow D'T$$

defining the second kind of transitions specified by D'-Coalgebras, in the premises depending on the complete behaviour as specified by the product comonad. If there should be any hope of deriving a distributive law like (7.30) by combining ℓ and ℓ', we have to impose conditions on ℓ' to make it 'almost' a distributive law, i.e., it should respect the structures of the monad and the involved comonads. However, this turns out to cause problems, although in a slightly unexpected way. For counit and comultiplication, use

$$\begin{tikzcd} T(D \times D') \arrow[r, Rightarrow, "\ell'"] \arrow[d, Rightarrow, "T\varepsilon"'] & D'T \arrow[d, Rightarrow, "\varepsilon'_T"] \\ T \arrow[r, equal] & T \end{tikzcd} \qquad \begin{tikzcd} T(D \times D') \arrow[rr, Rightarrow, "\ell'"] \arrow[d, Rightarrow, "T\delta"'] & & D'T \arrow[d, Rightarrow, "\delta'_T"] \\ T(D \times D')(D \times D') \arrow[r, Rightarrow, "\ell'_{(D \times D')}"'] & D'T(D \times D') \arrow[r, Rightarrow, "D'\ell'"'] & D'D'T \end{tikzcd}$$

where ε and ε' denote the counit of $D \times D'$ and D', respectively, analogously for the comultiplications δ and δ'. These two conditions are the evident variations on the standard conditions for distributive laws.

Additionally, ℓ' also has to respect the monad structure of T. For the unit η, we obtain

$$\begin{tikzcd} (D \times D') \arrow[r, Rightarrow, "\pi_2"] \arrow[d, Rightarrow, "\eta_{(D \times D')}"'] & D' \arrow[d, Rightarrow, "D'\eta"] \\ T(D \times D') \arrow[r, Rightarrow, "\ell'"'] & D'T \end{tikzcd}$$

However, for the multiplication μ, we *cannot* close the resulting diagram

$$\begin{tikzcd} TT(D \times D') \arrow[r, Rightarrow, "T\ell'"] \arrow[d, Rightarrow, "\mu_{(D \times D')}"'] & TD'T \arrow[r, Rightarrow, "???"] & D'TT \arrow[d, Rightarrow, "D'\mu"] \\ T(D \times D') \arrow[rr, Rightarrow, "\ell'"'] & & D'T \end{tikzcd}$$

because we do not have a map of type $TD'T \Rightarrow D'TT$ at hand. So what is causing problems is, surprisingly, that we do not have enough data to properly relate the non-independent part of the rules with the *monad* structure: we would have expected any problem to arise from the direction of the comonad operations involved.

Remark 7.43

The above approach seems flawed since it does not use the fact that $\langle B_A, E\rangle$**–Coalg**, or more abstractly $\langle H, D\rangle$**–Coalg**, is defined as a pullback category, indicated in (7.1). Using its universal property, and given a distributive law $TH^\infty \Rightarrow H^\infty T$, applying these data to obtain a lifting of T to an endofunctor (and then a monad) on $\langle H, D\rangle$**–Coalg**, we should be able to derive the appropriate conditions such that we indeed get a distributive law of type (7.29). However, due to the lack of time, we cannot investigate this approach more thoroughly, although preliminary considerations (with the assistance of G. Plotkin and J. Power) show that this approach might lead to a, at least formal, solution (it is not at all clear whether the resulting conditions are 'natural' when instantiated for timed processes). $\Diamond$

From a different perspective, starting from existing languages like TiCCS, one would like to use a natural transformation of type

$$\rho : \Sigma E(\mathrm{Id} \times B_A) \Rightarrow ET$$

for defining the time transitions: in order to be able to model the side condition for parallel composition, similar to the one in (7.27) enforcing (Maximal Progress), we need to know not only the targets of time transitions but also potential action transitions after a time transition. However, this time, it is now impossible to close the diagram expressing that ρ respects the comultiplication δ; intuitively, this is not surprising: after applying $\Sigma\delta_{(\mathrm{Id}\times B_A)}$ to the premises, we are stuck at $\Sigma EE(\mathrm{Id} \times B_A)$, i.e., we cannot even apply the rules *once*!

Regardless of that, suppose we were able to apply the rules, one is left with only time transitions in ETX, for some set X, while a second application of the rules would also require knowledge of the action behaviour of processes in X which simply is not there.

Remark 7.44

It seems that something fundamental is missing in the present approaches. This might either be to do with (not yet) exploiting the universal property of the pullback category $\langle H, D\rangle$**–Coalg**, or be rooted in the previously mentioned mismatch between the level of models, where the relation between action and time transitions is completely unspecified, and the level of languages, where the transitions suddenly should be no longer independent. Similarly to Remark 7.6 for discrete time, it might be worthwhile investigating whether we can, under the right circumstances, obtain a form of *tensor comonad* $B_A^\infty \otimes E$ such that its Coalgebras can be regarded as a variant of HTSs where the two kinds of transitions are suitably related. However, at this point, we have to leave this for further research. A reasonable starting point might be to consider replacing the product comonad by an arbitrary comonad C such that there is a comonad map from C to D (the projection of the product case) and then proceed from there. $\Diamond$

Finally, the last possibility for defining abstract rules for HTSs is by using their characterisation as the $\tilde{E}$-Coalgebras, where $\tilde{E}$ is the lifting of E to B_AE**–coalg** defined by the distributive law (7.8). The problem there, however, is that $\tilde{E}$ 'lives' on B_AE**–coalg** and so, in order to use it as a behaviour comonad, we also have to consider syntax on B_AE**–coalg**. For this, we would need a distributive law (of functors, so it is simply a natural transformation)

$$\ell : \Sigma(B_AE) \Rightarrow (B_AE)\Sigma$$

which would allow to lift a standard signature on **Set** to B_AE**–coalg**. We are at the moment not sure how to define such a natural transformations, and also leave that for further research. Once such a distributive law is obtained, we would get a functor $\tilde{\Sigma}$ on B_AE**–coalg** which (ideally) still freely generates a monad $\tilde{T}$. Then this would enable us to instantiate the results from Section 5.3, defining the operational semantics as natural transformations

$$\tilde{\Sigma}\tilde{E} \Rightarrow \tilde{E}(\mathrm{Id}+\tilde{\Sigma}) \quad \text{or} \quad \tilde{\Sigma}\tilde{E} \Rightarrow \tilde{E}\tilde{T} \ ,$$

each of which is then required to respect the structure of $\tilde{E}$ in the appropriate way.

By Corollary 7.30, the $\tilde{E}$-Coalgebras are isomorphic to the $(B_A^\infty \times E)$-Coalgebras, so we would not really expect to obtain any new insights by this re-formulation as regards the problem with incorporating non-independent rules either: even though an $\tilde{E}$-Coalgebra consists not only of an E-Coalgebra but also of a suitably related B_AE-coalgebra (cf. Section 7.2.2.3), the additional information in the B_AE-coalgebra is 'in the wrong order,' we would really need an EB_A-coalgebra, telling about actions *follow-*

ing a time transition, not the other way round; unfortunately, as already remarked in Section 7.2.2.3, we do not know how to obtain a lifting of E to EB_A**–coalg**.

7.4 Towards Heterogeneous Rule Formats

To conclude this chapter, we are going to present some results and remarks concerning syntactic ways of obtaining well-behaved abstract operational rules for HTSs. This is very tentative work; presently, the only, and consequently rather trivial, results pertain to the case that action and time rules are independent. Even so, these results at least account for all the previously described congruence results, exceptions being the ones for TPL and discrete TiCCS, for which we can at least present a conjecture.

7.4.1 Discrete Time

We can now combine previous results from this thesis and elsewhere, yielding an effective characterisation of a large class of languages for timed processes with independent rules over discrete time, together with some preliminary thoughts concerning extensions to the non-independent case. For a fixed countable enumerated set $\mathcal{V}$ of variables (in the sense of Section 6.1), the main result is as follows:

Theorem 7.45
Languages whose action rules are defined by GSOS rules, and whose time rules fit the dsl-format over $\mathcal{V}$, satisfy the property that heterogeneous bisimulation is a congruence.

Proof: From Theorem 5.1, we obtain that the GSOS rules induce (are, in fact, equivalent to) a natural transformation

$$\Sigma(\mathrm{Id} \times B_A) \Rightarrow B_A T$$

Furthermore, in Theorem 6.6, we showed that sets of rules in dsl-format are in one-to-one correspondence with natural transformations

$$\Sigma(\mathrm{Id} \times B_{\mathbb{N}}) \Rightarrow B_{\mathbb{N}} T$$

Finally, by Proposition 7.32, we thus obtain a natural transformation

$$\Sigma(\mathrm{Id} \times (B_A \times B_{\mathbb{N}})) \Rightarrow (B_A \times B_{\mathbb{N}})T$$

and consequently, from Proposition 7.33, we obtain that heterogeneous bisimulation is a congruence for the languages. □

This result also gives a proper proof of the congruence results for the languages ATP [NS94] and TeCCS [MT90], which were previously merely stated.

The analogous results for TPL [HR95] and TiCCS [Wan90] do not follow since neither of them has independent rules. Regarding the kind of dependencies we wish to allow, we argue that, to our knowledge, no language proposed in the literature features an operator whose action transitions depends on time transitions in any way, while the converse case at least exists, e.g., in the mentioned languages which adopt (Maximal Progress). Therefore, we propose to focus on such languages where the action transitions are independent of the time transitions, but not necessarily vice versa.

As an example, consider TiCCS and its parallel operator[4]. In Proposition 7.35, we showed that it is possible to use time rules which only allow time transitions after a successful test for the absence of certain initial actions of its component processes. Inspection of the proof yields that there is nothing special about the fact only τ is tested for being absent, we could equally have used an arbitrary other action in the set A of labels, and still have obtained a natural transformation. The reason for this genericity is the fact that the set of initial actions is not changed by applying renamings, because of the action of $\mathcal{P}_{\mathsf{fi}}(\cdot)^A$ on maps: the A-component is untouched.

Conceptually, since the action successors are not used at all, this corresponds to manipulating the B_A-component of the premises by applying the unique map $X \to 1$ for each set X, i.e., define the semantics as a map

$$\Sigma(X \times (B_A X \times B_{\mathbb{N}} X)) \to \Sigma(X \times (B_A 1 \times B_{\mathbb{N}} X)) \cong \Sigma(X \times (\mathcal{P}(A) \times B_{\mathbb{N}} X)) \to B_{\mathbb{N}} TX$$

[4]Which is actually the only ‘real’ operator where both action and time transitions are present in the time rules: the treatment of τ-prefixes as different from the other action prefixes does not really count since there are no premises.

the isomorphism holding because $\mathcal{P}_{\mathsf{fi}}(1) = 2$, and $2^A \cong \mathcal{P}(A)$. Since 1 is terminal, this results in a natural transformation if and only if the last part of the map

$$\Sigma(X \times (\mathcal{P}(A) \times B_{\mathbb{N}}X)) \to B_{\mathbb{N}}TX$$

is natural: that is exactly what is used in the rule for parallel composition. This leads us to the following conjecture:

Conjecture 7.46
If the action rules of a language are independent of the time rules, and the time rules are in dsl-format but featuring side conditions with predicates on initial actions, then heterogeneous bisimulation is a congruence for the language.

Of course, due to the nature of a conjecture, this is quite vague, in particular it would be interesting to formally define what we mean by 'predicate over initial actions,' but the previous remarks should give at least some intuition as to what we have in mind.

Collapsing the successor states to the one element of 1 also clearly suggests that rules like the above, which only test for action transitions without using the potential successor states, are only sound but not complete: there might be rules which actually use the successors of action transitions, while still inducing a well-defined map which is also natural.

Certainly not arbitrary combinations of action and time transitions in the premises can be allowed: we have to guarantee that the transition relation $\rightsquigarrow$ is *deterministic*. Ignoring the formalities involved with the dsl-format, take the following example of a 'bad' rule for an operator $[\alpha].\_$ for $\alpha \in A$ which, intuitively, allows to substitute a time step for any α-transition:

$$\frac{p \xrightarrow{\alpha} p''}{[\alpha].p \rightsquigarrow p''}$$

This operator does not induce a natural transformation of the appropriate kind because it is not even well-defined: the rule potentially introduces non-determinism for $\rightsquigarrow$, as there might be several distinct α-successors of p. Using a predicate $\det_\alpha(p)$, which holds if and only if there is precisely one α-successor of p, as a side condition to the above rule, then the rule would indeed induce a well-defined map: the predicate can be defined using only the set $B_A X$. However, then map would then no longer be natural: if there are two distinct α-successors in $B_A X$ the rule would not be applicable, while after applying a renaming which identifies the two, it would. This seems to substantiate the claim that tests for action transitions are only allowed if the potential successor states are not used in the conclusion.

Regardless of concrete languages, it is still an interesting open question to obtain a syntactic completeness result, i.e., to give a syntactic characterisation of natural transformations of type

$$\Sigma(\mathrm{Id} \times (B_A \times B_{\mathbb{N}})) \Rightarrow (B_A \times B_{\mathbb{N}})T$$

which would subsume all the previously mentioned rule formats.

7.4.2 The General Case

Since we have not been able to find abstract rules for non-independent sets of rules, we consequently cannot even begin to think about syntactic representation of such natural transformations. However, as shown in Corollary 7.41, independent sets of rules can be treated. Building on the results of [TP97] and Chapter 6, we obtain the following:

Theorem 7.47
Given a language, if its action rules fit within the GSOS format, and its time rules are given by either admissible operators as in Section 6.2.1 or admissible, co-pointed, and continuous sets of meta rules as in Section 6.2.2, then heterogeneous bisimulation is a congruence for the language.

Proof: This follows from Corollary 7.41, together with [TP97, Prop. 5.1] (for the action rules), Theorem 6.19 (for schematic time rules), and Corollary 6.39 (for the mentioned sets of meta rules). □

We can consequently obtain a new proof of the following results:

Corollary 7.48
Heterogeneous bisimulation is a congruence for TeCCS and ATP_D. □

Chapter 8

Conclusion

In this chapter, we wrap up the thesis with some concluding remarks. To this end, we present a summary of our key developments and results, followed by some directions for further research.

8.1 Summary of Results

This thesis presented a formal, abstract model of *timed processes*. Modelling time itself by *time domains* (Definition 3.1), a special kind of monoid, *pure* timed processes (i.e., only considering the timing but not the computational behaviour) were described by means of *TTSs* (Definition 3.23), special labelled transition systems with restrictions on the transition relation which account for some of the intuitive properties of the passage of time; *time bisimulation* (Definition 3.25) was used as a standard notion of equivalences for TTSs.

The adequacy and importance of the monoid structure on the time domain was then illustrated by the additional, more conventionally 'mathematical' (or 'algebraic') characterisation of TTSs as *partial monoid actions* of the time domain (Theorem 3.33), and further underlined by the new notion of *delay operator* (Definition 3.38): intuitively, it models how actions of processes can be postponed and formally, it corresponds to considering *total monoid actions* of the time domain; this also (to some extent) formalised the informal duality between consuming time in the form of time transitions, and 'stacking up' delaying potential. Moreover, these two notions of monoid actions were combined in the novel notion of *biaction* (Definition 3.40) which, despite employing only one axiom relating total and partial monoid action, still captured the essential part of the interplay between the two concepts of delaying and time passing (Proposition 3.41).

Following these concrete descriptions, also categorical characterisations of all the relevant abstract notions were presented. For total monoid actions, i.e., delay operators, well-known characterisations as both Algebras and Coalgebras for a monad and a comonad, respectively, were recalled (Section 4.1). Taking inspiration from the latter, a new comonad E of *evolutions* was introduced (Definitions 4.4 and 4.10), and some of its properties were discussed, in particular that it preserves (weak) pullbacks (Proposition 4.14 and Corollary 4.15) and that is has a rank (Proposition 4.16). As the most important application of this evolution comonad, it was shown that the E-Coalgebras are the same as partial monoid actions (Theorem 4.17), which have already been shown equivalent to TTSs, resulting in a Coalgebraic description of TTSs. Furthermore, *coalgebraic E-bisimulation* was shown to coincide with time bisimulation (Proposition 4.21), yielding a complete match between the transition system-based view and the categorical account.

In the case of discrete time (and more generally for arbitrary *free monoids*), it was then shown that $E = E_{\mathbb{N}}$ is actually *cofreely generated* (Theorem 4.23) which means that, rather than using $E_{\mathbb{N}}$-Coalgebras, one can instead use the simpler, completely unconstrained coalgebras for the *functor $B_{\mathbb{N}}$*. Finally, a distributive law of the monad for total actions over the evolution comonad was introduced and shown that its bialgebras were biactions of the time domain under consideration (Theorem 4.35); for *linear* time domains, it was furthermore shown that also the converse holds, i.e., that any biaction can be described as a bialgebra (Corollary 4.43).

With these results, the scene was set for obtaining an abstract theory of well-behaved operational rules for *behaviour comonads* by suitably adapting, and in the process extending, the framework of *bialgebraic semantics* developed in [TP97]; the extension allows the extra degree of freedom that behaviours no longer have to be functors (cofreely generating comonads) but that arbitrary comonads and their Coalgebras can be used for describing the appropriate kind of *operational models* (i.e., transition systems). Since, in particular, TTSs were Coalgebras for the evolution comonad, this approach applied to timed processes.

For discrete time, where the evolution comonad was cofreely generated, the results from [TP97] could be applied directly, cf. Section 5.2. In the general case, natural transformations of slightly more general kinds were used which, additionally, had to *respect the structure* of the behaviour comonad under consideration, cf. Definitions 5.8 and 5.16. In either case, both abstract and concrete congruence results were obtained, covering (the time rules of) all relevant existing languages (Corollaries 5.13, 5.19 and 5.20).

Careful analysis of the constraints expressed in abstract operational rules for the evolution comonad then allowed to derive several *syntactic rule formats* for timed processes. For discrete time, the *dsl-format* was introduced (Theorem 6.6) which completely captured the relevant abstract rules. It was shown to include all operators from the literature we are aware of. Furthermore, it was sketched how to extend the dsl-format to deal with multiple labels, thus allowing to treat *local qualitative time*, as used in languages like PMC [AM94].

For a general time domain, a 'format' based on *schematic rules* was presented, which was heavily constrained since it only allowed to draw the operational semantics of concrete operators from a small and fixed number of *admissible operators* (Definition 6.15); despite their lack of generality, the admissible operators still covered most important operators from the literature we know of, even excluding a non-desirable one that had previously been proposed. Even so, and not surprisingly, such rules were only *sound* in that, although they allowed to deduce a congruence result (Theorem 6.19), the converse did not hold: (6.7) presented a simple example of well-behaved rules not expressible by admissible operators.

To obtain a format which satisfies also this *completeness* property, a convenient way of specifying *infinite* sets of operational rules with an infinite number of premises, somewhat in between syntax and (abstract categorical) operational semantics, was introduced in the concept of *meta rules* (Definition 6.26). Using special, in general infinite, sets of such meta rules, a complete characterisation of very expressive abstract rules for timed processes was obtained (Corollary 6.39). However, due to the two levels of infinity involved (infinite sets of infinitary rules), this was a rather non-effective result.

Up to this point, all the results and developments so far were solely concerned with time transitions only, i.e., action transitions modelling (instantaneous) *computations* were *not* considered at all. To remedy this, Chapter 7 therefore presented the first steps towards a theory for *heterogeneous* timed processes, based on *HTSs* (Definition 7.1) which *disjointly* combine the two kinds of transitions. This was mirrored in the categorical characterisation as 'mixed' coalgebra/Coalgebra-pairs on the same carrier, obtained by considering the *pullback category* in (7.1). For discrete time, such mixed pairs could immediately be further simplified to pairs of coalgebras, since the evolution comonad was cofreely generated and hence, the *product functor* could be used as the appropriate behaviour functor (Proposition 7.5).

In the general case, no such easy simplification was possible. However, dualising results from [HPP], a Coalgebraic characterisation of HTSs was obtained as Coalgebras for a *composite comonad* (Corollary 7.16). Moreover, using general results on *(left) Kan extensions* from [Dub74], this composite comonad in fact turned out to be the *product comonad* (which is vastly different from the product functor!) of B_A^∞, the cofree comonad on the behaviour B_A for action transitions, and the evolution comonad E (Corollary 7.27), yielding a very pleasing conceptual match between the two cases of discrete and general time domains.

For discrete time, the above results already yielded an abstract theory of well-behaved heterogeneous operational rules, by using pairs of natural transformations; these admit all possible kinds of dependencies between the two kinds of transitions in operational rules. Considering the simple case of *independent* (abstract) rules (Proposition 7.32), the previously obtained results on syntactic rule formats simply carried over, leading to the very beginning of a theory of rule formats for heterogeneous timed processes (Theorem 7.45). Beyond independent rules, it was currently only possible to present (with some justification for its validity) a conjecture allowing the time transitions to depend on the action transitions (which, despite being rather restrictive, is the common practise in existing languages).

As for the general case of an arbitrary time domain, the situation is even more complex. However, independent rules were shown to also work in this case (Corollary 7.41) and thus, it was possible to conceptually retrace the congruence results obtained for existing languages like TeCCS [MT90] (Theorem 7.47).

8.2 Future Work

As always in research, there are many problems left open. We only highlight some of the more interesting and complex ones.

A first question is how to obtain a categorical characterisation of all the relevant examples of time domains. Linear and commutative domains would suggest to use such monoids which are closed when (pre-)ordered by the precedence relation. However, free monoids are *not* closed (though recall the connection between linearity and truncated subtraction being a right adjoint, cf. Proposition 3.22), and since we consider them to be important examples (accounting for qualitative time), closure is not the correct abstract characterisation.

Furthermore, antichain monotonicity seems to play an important rôle both in excluding unwanted models, in particular all the problems arising from products of time domains, and in making some technical results go through: it would therefore be interesting to see how it translates into the categorical view. Maybe this would also clear up why it pops up at two seemingly unrelated points of our development (the distributive law for biactions, and the syntactic approximation of abstract temporal rules).

In a different direction, it would be beneficial to also find an algebraic or monadic (in some sense) characterisation of partial monoid actions since in this way, it might be possible to obtain a more complete categorical description of biactions which, unlike the current one, does not depend on antichain monotonicity.

In addition to that, there are many obvious questions concerned with the syntactic rule formats for a general time domain. If at all, how far can the schematic rules be extended to obtain become more expressive, in particular, is there some leeway to be gained by trying to use the more parametric approach of Section 6.2.1.2? Is there a more finitary way of characterising abstract rules? When using the current model of

TTSs, we are rather pessimistic in this respect: there is at least one level of infinity automatically 'built-in' because TTSs are usually infinitely branching. Consequently, one has to allow infinitely many tests in the premises of rules (this only applies to a *complete* characterisation!). Even so, it would already constitute a drastic improvement to only require finite sets of (infinitary) rules.

On the other hand, another very common model for timed processes are *timed automata* [AD94], and a specific variant called *timed safety automata* [HNSY92]: both are extensions of traditional finite state machines[1] with real-valued clocks in order to include timing information. Based on these timed automata, there are furthermore several timed process algebras, e.g., [WPD94, DB96, LW00b, LW00a]. It might be beneficial to try and fit this model and these languages into the bialgebraic framework; preliminary results, employing a kind of *(timed) powerdomain*, suggest that this is indeed possible and could lead to very different syntactic characterisations (of a potentially more finitary nature).

Similarly, it might also be worthwhile investigating Fiore's approach to hybrid systems introduced in [Fio00]: perhaps this approach could be adjusted so as to fit within the bialgebraic approach, while removing the need to deal with infinite structures. In this way, 'proper' formats might be obtained.

Finally, there are many open questions to do with heterogeneous timed processes, most importantly how to properly deal with non-independent (abstract and concrete) rules. In principle, particularly in the simpler case of discrete time, it should be possible to deal with all kinds of dependencies, resulting in completely new opportunities for specifying the operational semantics of complex operators (in comparison to existing languages which only use minimal dependencies of time transitions on the presence or absence of specific action transitions). A related issue is the sketched idea of using a tensor product, rather than a cartesian product, as the appropriate way of combining the respective behaviours for action and time transitions.

[1] This continues a long-standing dispute in the verification and distributed systems community whether process-algebraic or automata-theoretic techniques should be used.

Appendix A

Total Monoid Actions Revisited

In this appendix, we present some more material on categorical formulations of total monoid actions. The reason we did not include it in Chapter 4 is that it is only recalling well-known results which are not really the main focus of the work described in the thesis. However, we consider the material to be sufficiently interesting to justify its inclusion in an appendix.

A.1 Total Monoid Actions as Presheaves

Apart from the characterisations as Algebras and Coalgebras, there is another well-known presentation of monoid actions, contained, e.g., in [MM92] which, as we shall see, actually subsumes the two mentioned descriptions. For this, recall that any monoid $M = \langle M, +, 0 \rangle$ can regarded as a category $\mathcal{M}$ with precisely one object, commonly written as $\star$, and the elements $m \in M$ correspond to morphisms $m : \star \to \star$ in $\mathcal{M}$. Then the addition $+$ of M becomes composition of morphisms in $\mathcal{M}$, e.g., for $m, n \in M$, $m + n$ becomes the composite $n \circ m$; note how this captures the intuitive functionality of addition: we start with m and then add n to it; correspondingly, we apply (the function) n *after* (the function) m. The neutral element 0, in particular, corresponds to the identity function on $\star$.

The *(contravariant) presheaves* over a category $\mathcal{C}$ are given as the functor category $\mathbf{Set}^{\mathcal{C}^{\mathrm{op}}}$, i.e., the category whose objects are all functors $\mathcal{C}^{\mathrm{op}} \to \mathbf{Set}$, and whose morphisms are all natural transformations between such functors. Categories of presheaves have very rich structure, in particular, they are *elementary topoi*, i.e., categories which, intuitively, are models of set theory (for the precise definition, and a lot of applications, see [MM92]). When instantiating the presheaf construction with $\mathcal{C} = \mathcal{M}$, a very simple description of the resulting category can be obtained.

Any functor $F : \mathcal{M}^{\mathrm{op}} \to \mathbf{Set}$ assigns: to the only object $\star$ of $\mathcal{M}$ a set $X \stackrel{\mathrm{df}}{=} F\star$; to each morphism $m : \star \to \star$ of $\mathcal{M}$, corresponding to $m \in M$, a function $Fm : F\star \to F\star$, i.e., a function $Fm : X \to X$. Intuitively, $Fm : X \to X$ describes how m acts on elements of X; we will write its values $Fm(x)$ as $m \cdot x$. Furthermore, since F is a contravariant functor, the following laws have to hold:

$$F(\mathrm{id}_\star) = \mathrm{id}_{F\star} = \mathrm{id}_X$$
$$F(m \circ n) = (Fn) \circ (Fm)$$

Rewriting them in the action-notation from above, and keeping in mind that $\mathrm{id}_\star$ and $m \circ n$, respectively, represent 0 and $n + m$, we obtain, for all $x \in X$

$$0 \cdot x = x$$
$$(n+m) \cdot x = n \cdot (m \cdot x)$$

i.e., precisely the axioms (3.25) and (3.26) of (left) total M-actions.

Moreover, natural transformations σ between functors $F, F' : \mathcal{M}^{\mathrm{op}} \to \mathbf{Set}$ exactly correspond to homomorphisms of M-actions: since $\mathcal{M}^{\mathrm{op}}$, like $\mathcal{M}$, has only one object, σ consists of precisely one component $\sigma_\star : F\star \to F'\star$, i.e., between the carriers of the two actions, and it is easily verified that naturality precisely corresponds to the homomorphism property for $\sigma_\star$. Therefore:

Proposition A.1
Let M be a monoid, and $\mathcal{M}$ its corresponding one-object category. Then the presheaf category $\mathbf{Set}^{\mathcal{M}^{\mathrm{op}}}$ is equivalent to the category M**–Act** of total M-actions. □

In particular, this shows that M**–Act** is an *elementary topos* [MM92]. As for generalising this characterisation to partial actions, the same problems as in the algebraic approach occur. The obvious thing to do would be to switch to functors $F : \mathcal{M}^{\mathrm{op}} \to \mathbf{pSet}$, i.e., 'presheaves' over **pSet**. Again, natural transformations between such 'partial presheaves' would only describe partial homomorphisms: $\sigma_\star$, as in the above considerations, would 'live' in the base category **pSet**. Finally, the remark about the lacking match between characterisation and established framework, as in the case of algebras for a monad, still applies. Yet, as we shall see next, the above characterisation of total actions as presheaves provides an alternative way to find the coalgebraic characterisation of total actions as presented in Section 4.1.2.

A.2 Varying the Monoid of Time

In this section, we will show how monoid homomorphisms induce especially well-behaved transformations between the actions of the monoids involved. We will then apply this general result in two very special cases in order to, in the next section, finally obtain our desired coalgebraic characterisation of total monoid actions.

Given two monoids M and N and a monoid homomorphism $f : N \to M$, when regarding the monoids as one-object categories $\mathcal{M}$ and $\mathcal{N}$, f precisely corresponds to a functor $\mathcal{N} \to \mathcal{M}$. By pre-composition, f induces a functor $f^\star : \mathbf{Set}^{\mathcal{M}^{\mathrm{op}}} \to \mathbf{Set}^{\mathcal{N}^{\mathrm{op}}}$; note the change in direction. Using the equivalence of categories between $\mathbf{Set}^{\mathcal{M}^{\mathrm{op}}}$ and M**–Act**, this means that $f^\star$ is a 'transformation' of M-actions into N-actions. Concretely, given an M-action $M \times X \to X$, one can define an N-action on X by using f to 'pull back' (not in the categorical sense!) the N-action to the given M-action, viz., $n \cdot x \stackrel{\mathsf{df}}{=} (fn) \cdot x$.

Recall now that all presheaf categories are topoi for which there is the following notion of maps between them. Given two topoi $\mathcal{E}$ and $\mathcal{F}$, a *geometric morphism* [MM92] $f : \mathcal{F} \to \mathcal{E}$ consists a pair of adjoint functors $f^\star \dashv f_\star$

$$\begin{tikzcd} \mathcal{E} \arrow[r, bend left, "f^\star"] \arrow[r, phantom, "\perp"] & \mathcal{F} \arrow[l, bend left, "f_\star"] \end{tikzcd}$$

such that $f^\star$ additionally is *left-exact*, i.e. preserves finite limits. The functors $f^\star$ and $f_\star$ are called the *inverse image* part, and the *direct image* part, respectively, of the geometric morphism f; note that the direction of the geometric morphism is the same as that of the direct image part $f_\star$. Further note, since $f^\star$ has a right adjoint, it automatically preserves all colimits (see e.g., [Mac97]).

In the special situation that $f^\star$ also has a left adjoint $f_! : \mathcal{F} \to \mathcal{E}$, i.e.,

$$\begin{tikzcd} \mathcal{E} \arrow[r, "f^\star" description] & \mathcal{F} \arrow[l, bend right, "f_!"'] \arrow[l, bend left, "f_\star"] \end{tikzcd} \text{ such that } f_! \dashv f^\star \dashv f_\star \,,$$

it follows that $f^\star$ automatically preserves *both* all limits *and* all colimits. So in particular, it is left-exact, hence $f^\star \dashv f_\star$ is a geometric morphism $f : \mathcal{F} \to \mathcal{E}$ which is then called an *essential* geometric morphism. Coming back to the case of monoid actions, the following result holds:

Proposition A.2

Let M, N be monoids with corresponding one-object categories $\mathcal{M}, \mathcal{N}$; let $f : N \to M$ be a monoid homomorphism. Then the induced presheaf functor $f^\star : \mathbf{Set}^{\mathcal{M}^{\mathrm{op}}} \to \mathbf{Set}^{\mathcal{N}^{\mathrm{op}}}$ is the inverse-image part of an essential geometric morphism $f : \mathbf{Set}^{\mathcal{N}^{\mathrm{op}}} \to \mathbf{Set}^{\mathcal{M}^{\mathrm{op}}}$, i.e., $f : N\text{–}\mathbf{Act} \to M\text{–}\mathbf{Act}$.

Proof: The proposition is a special instance of [MM92, §VII.2, Theorem 2]. $\square$

We will now give concrete descriptions of the adjoint functors implicit in Proposition A.2 in two particularly simple, yet interesting and well-known, cases, the first one of which shall be shown to provide the basis of the coalgebraic description of total monoid actions described in Chapter 4.

Example A.3

1. Recall that the trivial monoid $1 = \{0\}$ is the initial object in the category **Mon** of monoids and monoid homomorphisms. Therefore, taking N to be 1 and f to be the (by initiality unique) homomorphism $i : 1 \to M$, which maps $0 \in 1$ to $0 \in M$, and observing that 1-**Act** is equivalent to **Set**, we obtain a functor $i^\star : M\text{–}\mathbf{Act} \to \mathbf{Set}$. Calculating $i^\star$, starting from an M-action $M \times X \to X$, yields a 1-action, i.e., a set, given by

   $$X \cong 1 \times X \to X,\ \langle 0, x\rangle \mapsto 0 \cdot x = x \tag{A.1}$$

   In other words, the 1-action on X is simply given by the identity function, and so the result is simply the carrier X of the given M-action: $i^\star$ simply *forgets* the fact that X carries an M-action, hence it is commonly referred to as the *forgetful functor*.

By Proposition A.2, we know that this forgetful functor $i^\star$ has both a left and a right adjoint, respectively denoted by $i_!$ and $i_\star$, both of which take a set X and construct an M-action $M \times FX \to FX$, where FX denotes the respective resulting carrier. Here are elementary descriptions:

- For the left adjoint, we get that $i_!X = M \times X$, the M-action given by

$$M \times (M \times X) \xrightarrow{\mu_X} M \times X$$

  The map μ_X is the multiplication of the monad T from Proposition 4.1, therefore, this means $m' \cdot \langle m, x\rangle \overset{\mathsf{df}}{=} \langle m' + m, x\rangle$. Effectively, this action on $M \times X$ is simply an extension of the monoid addition, being an M-action on M itself, the X-component is completely ignored. Since it is the left adjoint to the forgetful functor, and because it does not impose any constraints on X, this is called the *free* M-action on X.

- For the right adjoint, one obtains that $i_\star X = X^M$, the set of all functions from M to X. The M-action on X^M, for $m \in M$ and $g : M \to X$, is given by

$$m \cdot g = g(\_ + m) \overset{\mathsf{df}}{=} \lambda n.g(n+m)$$

  Since this construction yields a right adjoint to the forgetful functor, this action is called the *cofree* M-action on X.

Using these definitions, it is easy to check well-definedness, and that the adjunctions hold as stated, i.e., $i_! \dashv i^\star \dashv i_\star$.

2. Dually, one can apply Proposition A.2 by exploiting that 1 is also the terminal object in **Mon** (hence it is a *null object*, see [Mac97]), and one obtains a functor $d^\star$: **Set** $\to M$**–Act**. It induces the *discrete* action [Law89] on a set: the action is simply given by projection, viz., $m \cdot x = x$. Of its two adjoints, the right is sometimes called the *points* [Law89, Law94], and the left the *components* [Law86], respectively. In the case of group actions, the points of the group action are more

commonly known as the *fixed points*, or the *kernel*, of the action, while the components are the *orbits*. In particular, a group action is called *transitive* if there is only one orbit, i.e., if the set of components is a singleton set; see [Lan93] for more details and applications. ◊

A.3 Coalgebras from Geometric Morphisms

In Example A.3, we have seen that the forgetful functor $i^\star : M\text{–}\mathbf{Act} \to \mathbf{Set}$ has both a left and a right adjoint, i.e., we are in the following situation, $i^\star$ being the unlabelled middle arrow:

$$\begin{tikzcd} M\text{–}\mathbf{Act} \arrow[d] \\ \mathbf{Set} \arrow[u, bend left=60, "i_!"] \arrow[u, bend right=60, "i_\star"'] \arrow[u, phantom, shift left=2, "\dashv"] \arrow[u, phantom, shift right=2, "\dashv"] \end{tikzcd}$$

For general reasons (see [Mac97, §VI.1]), this means that we get both a monad T and a comonad D on **Set**, viz., by composing the adjoints to obtain $T \overset{\mathsf{df}}{=} i^\star \circ i_!$ and $D \overset{\mathsf{df}}{=} i^\star \circ i_\star$.

When we concretely calculate T and D, we obtain that $TX = M \times X$, i.e., precisely the monad from Proposition 4.1, and $DX = X^M$, i.e., the one described in Proposition 4.3. Consequently, using the same results from [MM92] as in Section 4.1.2, the same coalgebraic characterisation of total monoid actions as before is obtained.

Bibliography

[Abr91] S. Abramsky. A domain equation for bisimulation. *Information and Computation*, 92:161–218, 191.

[Acz88] P. Aczel. *Non-Well-Founded Sets*. Center for the Study of Language and Information, Stanford University, 1988. CSLI Lecture Notes, Volume 14.

[AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 25 April 1994. Fundamental Study.

[AFV01] L. Aceto, W. Fokkink, and C. Verhoef. Structural operational semantics. In Bergstra et al. [BPS01], chapter 3, pages 197–292.

[AJ94] S. Abramsky and A. Jung. Domain theory. In S. Abramsky, D. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 3, pages 1–168. Clarendon Press, 1994.

[AJM00] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full Abstraction for PCF. *Information and Computation*, 163:409–470, 2000.

[AM89] P. Aczel and P. F. Mendler. A final coalgebra theorem. In D. H. Pitt, D. E. Rydeheard, P. Dybjer, A. M. Pitts, and A. Poigné, editors, *Proceedings of the Conference on Category Theory and Computer Science*, volume 389 of *LNCS*, pages 357–365, Berlin, September 1989. Springer-Verlag.

[AM94] H.R. Andersen and M. Mendler. An asynchronous process algebra with multiple clocks. In D. Sannella, editor, *Proceedings of the 5th European Symposium on Programming (ESOP '94)*, volume 788 of *Lecture Notes* *in Computer Science*, pages 58–73, Edinburgh, UK, April 1994. Springer-Verlag.

[AO97] K. R. Apt and E.-R. Olderog. *Verification of sequential and concurrent programs*. Graduate Texts in Computer Science. Springer-Verlag, 2nd edition, 1997.

[AR94] J. Adamek and J. Rosicky. *Locally Presentable and Accessible Categories*, volume 189 of *London Mathematical Society Lecture Notes Series*. Camebridge University Press, Cambridge, 1994.

[Bar84] H.P. Barendregt. *The Lambda Calculus*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, 1984.

[Bar93] M. Barr. Terminal coalgebras in well-founded set theory. *Theoretical Computer Science*, 144(2):299–315, 1993.

[Bar02] F. Bartels. GSOS for probabilistic transition systems (extended abstract). In Moss [Mos02].

[BB91] J.C.M. Baeten and J.A. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3(2):142–188, 1991.

[Bec69] J. Beck. Distributive laws. In B. Eckmann, editor, *Seminar on Triples and Categorical Homology Theory*, volume 80 of *Lecture Notes in Mathematics*, pages 119–140. Springer-Verlag, 1969.

[BG92a] G. Berry and G. Gonthier. The ESTEREL synchronous programming language: design, semantics, implementation. *Science of Computer Programming*, 19:87–152, 1992.

[BG92b] S. Brookes and S. Geva. Computational comonads and intensional semantics. In M. P. Fourman, P. T. Johnstone, and A. M. Pitts, editors, *Applications of Categories in Computer Science: Proceedings of the London Mathematical Society Symposium, Durham, UK, 1991*, volume 177 of *London Mathematical Society Lecture Notes Series*, Cambridge, 1992. Camebridge University Press.

[BHR84] S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe. A theory of communicating sequential processes. *Journal of the Association for Computing Machinery*, 31(3):560–599, July 1984.

[BIM95] B. Bloom, S. Istrail, and A. R. Meyer. Bisimulation can't be traced. *Journal of the ACM*, 42(1):232–268, January 1995.

[BJ89] G. S. Boolos and R. C. Jeffrey. *Computability and logic*. Camebridge University Press, third edition, 1989.

[BK84] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Computation*, 60:109–137, 1984.

[BK90] J.C.M. Baeten and J.W. Klop, editors. *CONCUR '90 (Concurrency Theory)*, volume 458 of *Lecture Notes in Computer Science*, Amsterdam, The Netherlands, August 1990. Springer-Verlag.

[BM01] J.C.M. Baeten and C.A. Middelburg. Process algebra with timing: real time and discrete time. In Bergstra et al. [BPS01], chapter 10.

[BN98] F. Baader and T. Nipkow. *Term Rewriting and All That*. Camebridge University Press, New York, 1998.

[BPS01] J. A. Bergstra, A. Ponse, and S. A. Smolka, editors. *Handbook of Process Algebra*. North-Holland, 2001.

[BS81] S. Burris and H. P. Sankappanavar. *A Course in Universal Algebra*, volume 78 of *Graduate Texts in Mathematics*. Springer-Verlag, 1981. Available online at `http://www.thoralf.uwaterloo.ca/htdocs/ualg.html`.

[BW85] M. Barr and C. F. Wells. *Toposes, Triples, and Theories*. Springer-Verlag, Berlin, 1985.

[BW90] J.C.M. Baeten and W.P. Weijland. *Process Algebra*, volume 18 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1990.

[CE82] E. M. Clarke and E. A. Emerson. Synthesis of synchronization skeletons from branching time temporal logic. *Lecture Notes in Computer Science*, 131:52–71, 1982.

[CGRH98] A. Corradini, M. Groß-Rhode, and R. Heckel. Structured Transition Systems as Lax Coalgebras. In B. Jacobs, L. Moss, H. Reichel, and J. Rutten, editors, *Coalgebraic Methods in Computer Science (CMCS'1999)*, volume 11 of *Electronic Notes in Theoretical Computer Science*, 1998.

[Che93] L. Chen. *Timed Processes: Models, Axioms and Decidability*. PhD thesis, Department of Computer Science, Edinburgh University, 1993.

[CHM99] A. Corradini, R. Heckel, and U. Montanari. From SOS Specifications to Structured Coalgebras: How to Make Bisimulation a Congruence. In Jacobs and Rutten [JR99].

[CHM02] A. Corradini, R. Heckel, and U. Montanari. Compositional SOS and beyond: a coalgebraic view of open systems. *Theoretical Computer Science*, 280(1–2):163–192, May 2002. Special issue with selected papers from [JR99]. This is an extended version of [CHM99].

[CLM97] R. Cleaveland, G. Lüttgen, and M. Mendler. An algebraic theory of multiple clocks. In A. Mazurkiewicz and J. Winkowski, editors, *CONCUR '97 (Concurrency Theory)*, volume 1243 of *Lecture Notes in Computer Science*, pages 166–180, Warsaw, Poland, July 1997. Springer-Verlag.

[DB96] P.R. D'Argenio and E. Brinksma. A calculus for timed automata (Extended abstract). In B. Jonsson and J. Parrow, editors, *Proceedings of the 4th International School and Symposium on Formal Techniques in Real Time and Fault Tolerant Systems,* Uppsala, Sweden, volume 1135 of *Lecture Notes in Computer Science*, pages 110–129. Springer-Verlag, 1996.

[DH83] R. De Nicola and M.C.B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1983.

[DP01] N. Dershowitz and D. A. Plaisted. Rewriting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 9, pages 535–610. Elsevier Science, 2001.

[dS85] R. de Simone. Higher-level synchronising devices in MEIJE-SCCS. *Theoretical Computer Science*, 37(3):245–267, December 1985.

[DS95] J. Davies and S. Schneider. A brief history of Timed CSP. *Theoretical Computer Science*, 138(2):243–271, 20 February 1995.

[Dub74] E. J. Dubuc. *Kan Extensions in Enriched Category Theory*, volume 145 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1974.

[dV98] E.P. de Vink. On a functor for probabilistic bisimulation and preservation of weak pullbacks. Technical Report IR–444, Vrije Universiteit, 1998. Paper presented at the ETAPS'98 Workshop on Coalgebraic Methods in Computer Science.

[dVR99] E. de Vink and J. Rutten. Bisimulation for probabilistic transition systems: A coalgebraic approach. *Theoretical Computer Science*, 221, 1999.

[EFT96] H.-D. Ebbinghaus, J. Flum, and W. Thomas. *Einführung in die mathematische Logik*. Spektrum, Akademischer Verlag, Heidelberg, 1996. Vierte Auflage.

[EHMR66] S. Eilenberg, D. K. Harrison, S. MacLane, and H. Röhrl, editors. *Proceedings of the Conference on Categorical Algebra, La Jolla, 1965*. Springer-Verlag, 1966.

[EK66] S. Eilenberg and G. M. Kelly. Closed categories. In Eilenberg et al. [EHMR66], pages 421–562.

[Exe98] R. Exel. Partial actions of groups and actions of inverse semigroups. *Proc. AMS*, 126(12), December 1998.

[Fio96] M. Fiore. A coinduction principle for recursive data types based on bisimulation. *Information and Computation*, 127(2):186–198, 1996.

[Fio00] M. Fiore. Fibred models of processes: Discrete, continuous, and hybrid systems. In *IFIP International Conference on Theoretical Computer Science*, volume 1872 of *Lecture Notes in Computer Science*, pages 457–473, 2000.

[FJM+96] M. P. Fiore, A. Jung, E. Moggi, P. O'Hearn, J. Riecke, G. Rosolini, and I. Stark. Domains and denotational semantics: History, accomplishments and open problems. *Bulletin of the EATCS*, 59:227–256, June 1996. Also published as Technical Report CSR-96-2, University of Birmingham School of Computer Science.

[Fok94] W. J. Fokkink. The tyft/tyxt format reduces to tree rules. *Lecture Notes in Computer Science*, 789:440–453, 1994.

[FPT99] M. Fiore, G. Plotkin, and D. Turi. Abstract syntax and variable binding. In *Fourteenth Annual Symposium on Logic in Computer Science (LICS '99)*, pages 193–202, Trento, Italy, 29 June–2 July 1999. IEEE Computer Society Press.

[FT01] M. Fiore and D. Turi. Semantics of name and value passing. In *Sixteenth Annual Symposium on Logic in Computer Science (LICS '01)*, pages 93–104, Boston, MA, USA, June 2001. IEEE Computer Society Press.

[FvG96] W. Fokkink and R. van Glabbeek. Ntyft/ntyxt rules reduce to ntree rules. *Information and Computation*, 126(1):1–10, 10 April 1996.

[Gla01] R.J. van Glabbeek. The linear time – branching time spectrum I; the semantics of concrete, sequential processes. In Bergstra et al. [BPS01], chapter 1, pages 3–99.

[GP02] M. Gabbay and A. Pitts. A New Approach to Abstract Syntax with Variable Binding. *Formal Aspects of Computing*, 13(3–5):341–363, 2002.

[Gro90] J. F. Groote. Specification and verification of real time systems in ACP. In *Protocol Specification, Testing and Verification, X*, Ottawa, Canada, June 1990.

[GS90] C. A. Gunter and D. S. Scott. Semantic domains. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 8, pages 633–674. North-Holland, New York, NY, 1990.

[GTW78] J. A. Goguen, J. W. Thatcher, and E. G. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In Raymond T. Yeh, editor, *Current Trends in Programming Methodology*, volume 4, pages 80–149. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1978.

[Gum01] H. P. Gumm. Functors for coalgebras. *Algebra Universalis*, 45(2–3):135–147, 2001.

[GV92] J. F. Groote and F. Vaandrager. Structured Operational Semantics and Bisimulation as a Congruence. *Information and Computation*, 100:202–260, 1992.

[HdR89] J.J.M. Hooman and W.P. de Roever. Design and verification in real-time distributed computing: an introduction to compositional methods. In *Proceedings of the Ninth International Conference on Protocol Specification, Testing and Verification*, Amsterdam, 1989. North-Holland.

[HNSY92] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. In *Proceedings, Seventh Annual IEEE Symposium on Logic in Computer Science*, pages 394–406, Santa Cruz, California, 22–25 June 1992. IEEE Computer Society Press.

[HO00] J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF. *Information and Computation*, 163:285–407, 2000.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, London, UK, 1985.

[HP79] M. Hennessy and G. Plotkin. Full abstraction for a simple parallel programming language. In *Eighth Symposium on Mathematical Foundations* *of Computer Science*, volume 74 of *Lecture Notes in Computer Science*, pages 108–120, Berlin, 1979. Springer-Verlag.

[HPP] M. Hyland, G. Plotkin, and J. Power. Combining effects: sum and tensor. Submitted. Available from `http://www.dcs.ed.ac.uk/home/gdp`.

[HR95] M. Hennessy and T. Regan. A process algebra for timed systems. *Information and Computation*, 117:221–239, 1995.

[Jac94] B. Jacobs. Semantics of weakening and contraction. *Annals of Pure and Applied Logic*, 69(1):73–106, 6 September 1994.

[Jac95a] B. Jacobs. Bisimulation and Apartness in Coalgebraic Specifications. Notes of lectures given in January1995 at the joint TYPES/CLICS workshop in Gothenburg and at a BRICS seminar in Aarhus, version of 24 February, 1995.

[Jac95b] B. Jacobs. Mongruences and cofree coalgebras. In V.S. Alagar and M. Nivat, editors, *Fourth International Conference on Algebraic Methodology and Software Technology, AMAST'95*, volume 936 of *Lecture Notes in Computer Science*, pages 245–260, Montreal, Canada, 3–7 July 1995. Springer-Verlag.

[Jac96] B. Jacobs. Coalgebraic specifications and models of deterministic hybrid systems. In M. Wirsing and M. Nivat, editors, *Algebraic Methodology and Software Technology*, volume 1101 of *Lecture Notes in Computer Science*, pages 520–535, Munich, Germany, 1–5 July 1996. Springer-Verlag.

[Jac00] B. Jacobs. Object-oriented hybrid systems of coalgebras plus monoid actions. *Theoretical Computer Science*, 239(1):41–95, May 2000.

[Jef91] A. Jeffrey. A linear time process algebra. In Larsen and Skou [LS91b], pages 432–442.

[Joh82] P. T. Johnstone. *Stone Spaces*, volume 3 of *Cambridge Studies in Advanced Mathematics*. Camebridge University Press, Cambridge, 1982.

[Jon90] C. Jones. *Probabilistic Non-Determinism*. PhD thesis, LFCS, Edinburgh University, 1990.

[JP89] C. Jones and G. D. Plotkin. A probabilistic powerdomain of evaluations. In *Proceedings, Fourth Annual Symposium on Logic in Computer Science*, pages 186–195, Asilomar Conference Center, Pacific Grove, California, 5–8 June 1989. IEEE Computer Society Press.

[JPT+98] P. Johnstone, J. Power, T. Tsujishita, H. Watanabe, and J. Worrel. An Axiomatics for Categories of Transition Systems as Coalgebras. In *Thirteenth Annual Symposium on Logic in Computer Science (LICS '98)*, Indianapolis, Indiana, USA, 21–24 June 1998. IEEE Computer Society Press.

[JPT+01] P. Johnstone, J. Power, T. Tsujishita, H. Watanabe, and J. Worrel. On the structure of categories of coalgebras. *Theoretical Computer Science*, 260:87–117, 2001. Preliminary version appeared as [JPT+98].

[JR99] B. Jacobs and J. Rutten, editors. *Coalgebraic Methods in Computer Science (CMCS'1999)*, volume 19 of *Electronic Notes in Theoretical Computer Science*, 1999.

[JSV93] A. S. A. Jeffrey, S. A. Schneider, and F. W. Vaandrager. A comparison of additivity axioms in timed transition systems. Technical Report CS-R9366, CWI - Centrum voor Wiskunde en Informatica, December 31, 1993.

[Kah96] S. Kahrs. Limits of ML-definability. In *Proceedings of PLILP'96*, volume 1140 of *Lecture Notes in Computer Science*, pages 17–31. Springer-Verlag, September 1996.

[Kel82] G. M. Kelly. *Basic concepts of enriched category theory*, volume 64 of *London Mathematical Society lecture note series*. Camebridge University Press, 1982.

[Kic99] M. Kick. Modeling synchrony and asynchrony with multiple clocks. Master's thesis, University of Passau, Passau, Germany, August 1999.

[Kic02a] M. Kick. Bialgebraic modelling of timed processes. In P. Widmayer, F. Triguero, R. Morales, M. Hennessy, S. Eidenbenz, and R. Conejo, editors, *Proceedings ICALP'02*, volume 2380 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002. Available from `http://www.dcs.ed.ac.uk/home/mk`.

[Kic02b] M. Kick. Rule formats for timed processes. In *Proceedings CMCIM'02*, volume 68 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2002. Available from `http://www.dcs.ed.ac.uk/home/mk`.

[KL97] G. M. Kelly and S. Lack. On property-like structures. *Theory and Application of Categories*, 3:213–250, 1997.

[KL02] J. Kellendonk and M. Lawson. Partial actions of groups. Accepted by International Journal of Algebra and Computation, 2002.

[Lan93] S. Lang. *Algebra*. Addison-Wesley, Reading, MA, USA, third edition, 1993.

[Law73] F. W. Lawvere. Metric spaces, generalized logic, and closed categories. In *Rendiconti del Seminario Matematico e Fisico di Milano, XLIII*. Tipografia Fusi, Pavia, 1973.

[Law86] F. W. Lawvere. Taking categories seriously. *Revista Columbiana de Matematicas*, 20(3–4):147–178, 1986.

[Law89] F. W. Lawvere. Qualitative distinctions between some toposes of generalized graphs. In J. W. Gray and A. Scedrov, editors, *Categories in Computer Science and Logic*, volume 92 of *Contemporary Mathematics*. American Mathematical Society, 1989.

[Law94] F. W. Lawvere. Cohesive toposes and Cantor's 'lauter Einsen'. *Philisophia Mathematica*, 2:5–15, 1994.

[Lin66] F. E. J. Linton. Some aspects of equational categories. In Eilenberg et al. [EHMR66], pages 84–94.

[LPW00] M. Lenisa, J. Power, and H. Watanabe. Distributivity for endofunctors, pointed and co-pointed endofunctors, monads and comonads. In H. Reichel, editor, *Coalgebraic Methods in Computer Science (CMCS'2000)*, volume 33 of *Electronic Notes in Theoretical Computer Science*, pages 233–263, 2000.

[LS91a] K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, September 1991.

[LS91b] K.G. Larsen and A. Skou, editors. *Computer Aided Verification (CAV '91)*, volume 575 of *Lecture Notes in Computer Science,* Aalborg, Denmark, July 1991. Springer-Verlag.

[LW00a] H. Lin and Y. Wang. A complete axiomatisation for timed automata. In *Proccedings of Foundations of Software Technology and Theoretical Computer Science*, volume 1974 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.

[LW00b] H. Lin and Y. Wang. A proof system for timed automata. In *Proccedings of Foundations of Software Science and Computation Structures*, volume 1784 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.

[Mac97] S. Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, 2nd edition, 1997. (1st ed., 1971).

[Mal37] A. Malcev. On the immersion of an algebraic ring into a field. *Mathematische Annalen*, 113:686–691, 1937.

[Mal39] A. Malcev. On the immersion of associative systems in groups. *Mat. Sbornik*, 6(48):331–336, 1939. In Russian.

[Mal40] A. Malcev. On the immersion of associative systems in groups. *Mat. Sbornik*, 8(50):251–264, 1940. In Russian.

[Mil72] R. Milner. Logic for computable functions: description of a machine implementation. Technical Report STAN-CS-72-288, Computer Science Department, Stanford University, May 1972.

[Mil77] R. Milner. Fully abstract models of typed lambda-calculus. *Theoretical Computer Science*, 4:1–22, 1977.

[Mil80] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1980.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, London, UK, 1989.

[MM92] S. Mac Lane and I. Moerdijk. *Sheaves in Geometry and Logic: A First Introduction to Topos Theory*. Universitext. Springer-Verlag, New York, 1992.

[Mos02] L. Moss, editor. *Proc. Coalgebraic Methods in Computer Science (CMCS 2002)*, volume 65 of *Electronic Notes in Theoretical Computer Science*, 2002.

[MP89] M. Makkai and R. Pare. *Accessible categories: The Foundations of Categorical Model Theory*. American Mathematical Society, Providence, 1989.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–77, September 1992.

[MS02] M. Megrelishvili and L. Schröder. Globalization of Confluent Partial Actions on Topological and Metric Spaces. Preprint submitted to Elsevier Science, 23 August 2002.

[MT90] F. Moller and C. Tofts. A temporal calculus of communicating systems. In Baeten and Klop [BK90], pages 401–415.

[NPW81] M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures and domains, Part 1. *Theoretical Computer Science*, 13:85–108, 1981.

[NS91] X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In Larsen and Skou [LS91b], pages 376–398.

[NS94] X. Nicollin and J. Sifakis. The algebra of timed processes, ATP: Theory and application. *Information and Computation*, 114:131–178, 1994.

[NSY93] X. Nicollin, J. Sifakis, and S. Yovine. From ATP to timed graphs and hybrid systems. *Acta Informatica*, 30:181–202, 1993.

[Par81] D. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Theoretical Computer Science: 5th GI-Conference, Karlsruhe*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183, Berlin, Heidelberg, and New York, March 1981. Springer-Verlag.

[Pit94] A. M. Pitts. Computational Adequacy via 'Mixed' Inductive Definitions. In *Mathematical Foundations of Programming Semantics, Proc. 9th Int. Conf., New Orleans, LA, USA, April 1993*, volume 802 of *Lecture Notes in Computer Science*, pages 72–82. Springer-Verlag, Berlin, 1994.

[Plo76] G. D. Plotkin. A powerdomain construction. *SIAM Journal on Computing*, 5(3):452–487, 1976.

[Plo77] G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5(3):223–255, 1977.

[Plo81] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, Aarhus, Denmark, September 1981.

[Plo82a] G. Plotkin. A powerdomain for countable non-determinism. In M. Nielsen and E. M. Schmidt, editors, *Automata, Languages and programming*, pages 412–428, Berlin, 1982. EATCS, Springer-Verlag. Lecture Notes in Computer Science Vol. 140.

[Plo82b] G. Plotkin. Probabilistic powerdomains. In *Proceedings CAAP*, 1982.

[Plo83] G. Plotkin. Post-graduate lecture notes in advanced domain theory (incorporating the "Pisa Notes"). LaTeX'ed version available from `http://www.dcs.ed.ac.uk/home/gdp`, 1983.

[Plo85] G. D. Plotkin. Denotational semantics with partial functions. Notes for a course given at the Center for the Study of Language and Information, Stanford, 1985.

[Plo01] G. Plotkin. Bialgebraic Semantics and Recursion (Extended Abstract). In A. Corradini, M. Lenisa, and U. Montanari, editors, *Coalgebraic Methods in Computer Science (CMCS'2001)*, volume 44 of *Electronic Notes in Theoretical Computer Science*, pages 284–287, 2001.

[Pnu85] A. Pnueli. Linear and branching structures in the semantics and logics of reactive systems. In W. Brauer, editor, *Proceedings of the 12th Colloquium on Automata, Languages and Programming*, volume 194 of *Lecture Notes in Computer Science*, pages 15–32, Nafplion, Greece, July 1985. Springer.

[Pra86] V. R. Pratt. Modelling Concurrency with Partial Orders. *International Journal of Parallel Programming*, 15(1):33–71, February 1986.

[PW99] J. Power and H. Watanabe. Distributivity for a monad and a comonad. In Jacobs and Rutten [JR99].

[RB85] D. E. Rydeheard and R. M. Burstall. Monads and theories: a survey for computation. In M. Nivat and J. C Reynolds, editors, *Algebraic methods in semantics*, pages 575–605. Camebridge University Press, 1985. Articles stemming from the seminar on 'The Application of Algebra to Language Definition and Compilation,' Fountainebleau, France, June, 1982.

[Rob02] E. Robinson. Variations on Algebra: Monadicity and Generalisations of Equational Theories. *Formal Aspects of Computing*, 13(3–5):308–326, 2002.

[RR88] G.M. Reed and A.W. Roscoe. A timed model for communicating sequential processes. *Theoretical Computer Science*, 58:249–261, 1988.

[RS02] W. Rounds and H. Song. The phi-calculus - a hybrid extension of the pi-calculus to embedded systems. In *Eighteenth Workshop on the Mathematical Foundations of Programming Semantics*, New Orleans, LA, USA, 23–26 March 2002.

[RT94] J. Rutten and D. Turi. Initial algebra and final coalgebra semantics for concurrency. In J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *A Decade of Concurrency (REX Workshop 1993)*, volume 803 of *Lecture Notes in Computer Science*, pages 530–582. Springer-Verlag, 1994.

[Rut00] J. J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000.

[Sch95] S. Schneider. An Operational Semantics for Timed CSP. *Information and Computation*, 116(2):193–213, 1 February 1995.

[Sco70] D. S. Scott. Outline of a mathematical theory of computation. In *Proceedings, Fourth Annual Princeton Conference on Information Sciences and Systems*, pages 169–176. Princeton University, 1970. Also, Programming Research Group Technical Monograph PRG–2, Oxford University.

[Sco82] D. S. Scott. Domains for Denotational Semantics. In M. Nielson and E. M. Schmidt, editors, *Automata, Languages and Programming: Proceedings 1982*, volume 140 of *Lecture Notes in Computer Science*. Springer-Verlag, 1982.

[Sco93] D. S Scott. A type-theoretical alternative to ISWIM, CUCH, OWHY. *Theoretical Computer Science*, 121:411–440, 1993. First written in 1969 and circulated privately.

[SDJ$^+$91] S. Schneider, J. Davies, D.M. Jackson, G.M. Reed, J.N. Reed, and A.W. Roscoe. Timed CSP: Theory and practice. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, pages 526–548. Springer-Verlag, 1991.

[Sim95] A. K. Simpson. Compositionality via cut-elimination: Hennessy-Milner logic for an arbitrary GSOS. In *Proceedings, Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 420–430, San Diego, California, 26–29 June 1995. IEEE Computer Society Press.

[SP82] M. B. Smyth and G. D. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM Journal on Computing*, 11(4):761–783, November 1982. Also Report D.A.I. 60, University of Edinburgh, Department of Artificial Intelligence, December 1978.

[SS71] D. S. Scott and C. Strachey. Toward a mathematical semantics for computer languages. In *Proceedings Symposium on Computers and Automata*, volume 21 of *Microwave Institute Symposia Series*, pages 19–46. Polytechnic Institute of Brooklyn, 1971.

[Ten91] R. D. Tennent. *Semantics of Programming Languages*. Prentice Hall, New York, 1991.

[TP97] D. Turi and G. Plotkin. Towards a mathematical operational semantics. In *Twelfth Annual Symposium on Logic in Computer Science (LICS '97)*, pages 280–291, Warsaw, Poland, 29 June–2 July 1997. IEEE Computer Society Press.

[TR98] D. Turi and J. Rutten. On the foundations of final coalgebra semantics: non-well-founded sets, partial orders, metric spaces. *Mathematical Structures in Computer Science*, 8(5):481–540, 1998.

[Tur96] D. Turi. *Functorial Operational Semantics and its Denotational Dual*. PhD thesis, Free University, Amsterdam, June 1996. Available from `http://www.dcs.ed.ac.uk/home/dt/`.

[Tur97] D. Turi. Categorical modelling of structural operational rules: Case studies. In *Proc. 7th CTCS Conf.*, volume 1290 of *Lecture Notes in Computer Science*, pages 127–146, 1997.

[Var91] M. Y. Vardi. Verification of concurrent programs: The automata-theoretic framework. *Annals of Pure and Applied Logic*, 51(1–2):79–98, 1991.

[vB89] J. van Benthem. Time, logic and computation. In J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Proceedings of the School/Workshop on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *LNCS*, pages 1–49, Berlin, May 30–June 3 1989. Springer.

[vG96] R.J. van Glabbeek. The meaning of negative premises in transition system specifications II (extended abstract). In F. Meyer auf der Heide and B. Monien, editors, *Automata, Languages and Programming (ICALP '96)*, volume 1099 of *Lecture Notes in Computer Science*, pages 502–513, Paderborn, Germany, July 1996. Springer-Verlag.

[vGSS95] R. J. van Glabbeek, S. A. Smolka, and B. Steffen. Reactive, generative and stratified models of probabilistic processes. *Information and Computation*, 121(1):59–80, 15 August 1995.

[Vic89] S. Vickers. *Topology via Logic*. Camebridge University Press, 1989.

[vW66] B. L. v.d. Waerden. *Modern Algebra*, volume 1 and 2. Frederick Ungar Publishing Co., New York, 1966.

[Wan90] Y. Wang. Real-time behaviour of asynchronous agents. In Baeten and Klop [BK90], pages 502–520.

[Wan91] Y. Wang. CCS + time = an interleaving model for real time systems. In J. Leach Albert, B. Monien, and M. Rodríguez Artalejo, editors, *Automata, Languages and Programming (ICALP '91)*, volume 510 of *Lecture Notes in Computer Science*, pages 217–228, Madrid, Spain, July 1991. Springer-Verlag.

[Wat02] H. Watanabe. Well-behaved Translations between Structural Operational Semantics. In Moss [Mos02].

[Win93] Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction.* Foundations of Computing series. MIT Press, February 1993.

[WN95] G. Winskel and M. Nielsen. Models for Concurrency. In S. Abramsky, D. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 4, pages 1–148. Clarendon Press, 1995.

[WPD94] Y. Wang, P. Pettersson, and M. Daniels. Automatic Verification of Real-Time Communicating Systems By Constraint-Solving. In Dieter Hogrefe and Stefan Leue, editors, *Proc. of the 7*th *Int. Conf. on Formal Description Techniques*, pages 223–238. North–Holland, 1994.

List of Symbols

Index