Retracing some paths in categorical semantics: From process-propositions-as-types to categorified reals and computers

Dusko Pavlovic*
University of Hawaii, Honolulu HI
dusko@hawaii.edu

Abstract

The logical parallelism of propositional connectives and type constructors extends beyond the static realm of predicates, to the dynamic realm of processes. Understanding the logical parallelism of *process* propositions and *dynamic* types was one of the central problems of the semantics of computation, albeit not always clear or explicit. It sprung into clarity through the early work of Samson Abramsky, where the central ideas of denotational semantics and process calculus were brought together and analyzed by categorical tools, e.g. in the structure of *interaction categories*. While some logical structures borne of dynamics of computation immediately started to emerge, others had to wait, be it because the underlying logical principles (mainly those arising from coinduction) were not yet sufficiently well-understood, or simply because the research community was more interested in other semantical tasks. Looking back, it seems that the process logic uncovered by those early semantical efforts might still be starting to emerge and that the vast field of results that have been obtained in the meantime might be a valley on a tip of an iceberg.

In the present paper, I try to provide a logical overview of the gamut of interaction categories and to distinguish those that model computation from those that capture processes in general. The main coinductive constructions turn out to be of the latter kind, as illustrated towards the end of the paper by a compact category of all real numbers as processes, computable and uncomputable, with polarized bisimulations as morphisms. The operation of addition of the reals arises as the biproduct, real vector spaces are the enriched bicompletions, and linear algebra arises from the enriched kan extensions. At the final step, I sketch a structure that characterizes the computable fragment of categorical semantics.

*Supported by NSF and AFOSR.

Contents

1 Introduction: On categorical logics and propositions-as-types **4**
- 1.1 Logics of types . . . 5
- 1.2 Categorical proof theory . . . 6
  - 1.2.1 Entailments as morphisms . . . 6
  - 1.2.2 Conjunction and disjunction as product and coproduct . . . 7
  - 1.2.3 Function abstraction and cartesian closed categories . . . 8
- 1.3 Modalities as monads and comonads . . . 9
  - 1.3.1 Possibility and side-effects . . . 9
  - 1.3.2 Necessity and reductions . . . 11
- 1.4 Labelled sequents, commutative monads, and surjections . . . 12

2 Process logics **13**
- 2.1 Idea of process . . . 13
- 2.2 Process propositions and implications . . . 14
  - 2.2.1 Process sequents must be labelled . . . 14
  - 2.2.2 Machine abstraction and process-closed categories . . . 14
  - 2.2.3 Process propositions . . . 17
- 2.3 Relating process implications and static implications . . . 17
  - 2.3.1 History types . . . 17
  - 2.3.2 Retractions and idempotents . . . 18
  - 2.3.3 Proposition . . . 19
- 2.4 The problem of cut in process logics . . . 19

3 Functions extended in time **21**
- 3.1 Dynamic elements as streams . . . 21
- 3.2 Functions extended in time as deterministic channels . . . 21
- 3.3 History monad and comonad . . . 22
- 3.4 Category of functions extended in time . . . 22

4 Partial functions extended in time **23**
- 4.1 Output deletions and process deadlocks . . . 23
- 4.2 Safety and synchronicity . . . 24
  - 4.2.1 Synchronous safe functions . . . 24
  - 4.2.2 Asynchronous safe functions . . . 24

5 Relations extended in time **26**
- 5.1 External and internal nondeterminism . . . 26
- 5.2 Internal nondeterminism . . . 27
  - 5.2.1 Synchronous safe relations . . . 27
  - 5.2.2 Asynchronous safe relations . . . 28
- 5.3 External nondeterminism . . . 29

5.3.1 Synchronous dynamic relations . . . 29
5.3.2 Internalising the labels . . . 31
5.3.3 Synchronous dynamic relations as hypersets . . . 32
5.3.4 Asynchronous dynamic relations . . . 33

6 Integers, interactions, and real numbers 33
6.1 The common denominator of integers and interactions . . . 33
6.2 Games as labelled polarized relations extended in time . . . 35
6.3 Polarized dynamics . . . 35
6.3.1 Synchronous case . . . 36
6.3.2 Asynchronous case . . . 37
6.4 A category of real numbers . . . 38
6.4.1 Coalgebra of reals . . . 38
6.4.2 Numbers extended in time: Conway's version of Dedekind cuts . . . 39
6.4.3 Real numbers as processes . . . 40
6.5 Where is computation? . . . 42

7 Categorical semantics as a programming language 42
7.1 Computability-as-programmability . . . 42
7.2 Categorical semantics of intensional computation . . . 43
7.3 Computability as an intrinsic property . . . 46

8 Summary 48

Appendices 57

A Category R of sets and relations 57

B Proof of Prop. 2.3.3 58

C Proof sketch for Lemma 5.1 61

D Proof of Corollary 5.2 61

E Traces and the Int-construction 62

F The extended reals as alternating dyadics 63

Personal introduction

I first learned about Samson Abramsky's work from his invited plenary lecture at the International Category Theory Meeting in Montreal in 1991. It was the golden age of category theory, and Montreal was at the heart of it, and I got to be a postdoc there. Just a few years earlier, I was a dropout freelance programmer, but had become a mathematician, and was uninterested in computers. I was told, however, that Abramsky had constructed some categories that no one had seen before, so I came to listen to his talk. I also had a talk to give later that day myself, but for some reason, I do not recall how that went. At the end of Abramsky's plenary lecture, Saunders Mac Lane stood up, one of the two fathers of category theory, high up near the ceiling of the amphitheater, and spoke for a long time. He criticized computer science in general. After that, Bill Lawvere stood up, and provided some friendly comments, suggesting directions for progress and improvement.

Two years later, I became an *"EU Human Capital Mobility"* fellow within the Theory Group at the Imperial College in London, led by Samson Abramsky. I started learning computer science and spent a lot of time trying to understand Samson's *interaction categories* [3]. In the meantime, he had constructed more categories that no one had seen before. My fellowship ended after a year or two, and the human capital mobility turned out to be much greater than anyone could imagine, but I continued to think about interaction categories for years. Here I try to summarize some of that thinking.

1 Introduction: On categorical logics and propositions-as-types

The category of sets or types. This is a paper about categorical semantics. It is written for a collection intended for logicians. If you are reading this, then you are presumed to be interested in categorical logic, although you may not be interested in categories in general. To ease this tension, I will avoid abstract categories, and mostly stick with the category **S** of sets and functions. It is presented, however, as a universe of types, by specifying which type constructors are used in each construction. Initially, we just need the cartesian products, but later we need more. The naive set theory used to be presented incrementally. Nowadays most mathematicians think of types as sets, and most programmers think of sets as types, so it seems reasonable for logicians and computer scientists to identify the two. To keep the naive-set-theory flavor, we usually call the type inhabitants *elements*, where type theorists use the term *terms*.

When a set is constructed as a type, then it can also be construed as a proposition: since its elements are constructions, they can be viewed as proofs [71]. Such interpretations originate from logic, where the idea of propositions-as-types was first encountered along the paths of proofs-as-constructions [30, 55, 70]. We retrace these paths first, and proceed throughout with propositions-as-types, types-as-sets, terms-as-elements, elements-as-morphisms [62, 64].

Naming names. While sets and types signal different approaches, many concepts are studied in different communities under different names even if there are no significant differences. This is useful to place narratives in their contexts or to authenticate speakers' allegiances. It is not easy to avoid such connotations when they are undesired. In some cases, I resorted to renaming. E.g., the

histories from Sec. 2.3.1 onwards are known as non-empty lists, or words, or strings. There are other examples. I am not trying to reinvent them but to dissociate them from narrow contexts. I hope they will not be too distracting.

1.1 Logics of types

Bertrand Russell proposed his *ramified theory of types* [104] as a logical framework for paradox prevention. Alonzo Church and Stephen Kleene advanced type theory into a model of computation [30, 52]. Dana Scott adopted type theory as the foundation for a mathematical approach to the semantics of computation [105]. The semantics of programming languages were built steadily upon that foundation [45, 102]. Process semantics also arose from that foundation [73], but had to undergo a substantive conceptual evolution before the types could capture dynamics. I followed these developments through Samson Abramsky's work.

The propositions-as-types paradigm was discovered many times. In logic and computer science, it is attributed to Haskell Curry and William Howard [106, 42, Ch. 3]. Howard got the idea from Georg Kreisel [114], and Kreisel's goal was to formalize Brouwer's concept of proofs-as-constructions [56]. An early formalization of Brouwer's concept goes back to Kolmogorov [55].

The structural reason why propositions and types obey analogous laws was offered by Lawvere [65], who pointed out that the propositional and the typing rules are instances of analogous categorical *adjunctions*; and that the proof constructions and the term derivations arise from the adjunction units and counits. This gave rise to the idea of categorical proof theory, pursued by Lambek [58, 60, 61], and to the basic structures of categorical semantics, succinctly described in [63, and the references therein]. In the preface to his seminal report [105], Dana Scott explained that

> "a category represents the 'algebra of types', just as abstract rings give us the algebra of polynomials, originally understood to concern only integers or rationals. One can of course think only of particular type systems, but, for a full understanding, one needs also to take into account the general theory of types, and especially *translations* or *interpretations* of one system in another."

Samson Abramsky spearheaded the efforts towards expanding the categorical semantics of program abstraction, as formalized in type theory and merge it with a categorical semantics of process abstraction and interaction, as formalized in the theory of concurrency and process calculi. This led to interaction categories [3, 31, 84, 86], specification structures [11, 95], and a step further to geometry of interaction [13] and game semantics [5, 12, 14, 15, and many other publications]. As the realm of program abstraction expanded, e.g. into quantum computation and protocols [10], the semantical apparatus also expanded [7, 8], the tree branched [33, 91], some branches crossed[1]. In the present paper, however, we are only concerned with the root. And even that might be overly ambitious.

[1] E.g., [90] used the methods of [95] to expand the models of [10].

1.2 Categorical proof theory

Proofs-as-constructions. The Curry-Howard isomorphism is one of the conceptual building blocks of type theory, built deep into the foundation of computer science and functional programming [42, ch. 3]. The fact that it is an *isomorphism* means that the type constructors on one side obey the same laws as the propositional connectives on the other side; and these laws are expressed as a bijection between the terms and the proofs.

1.2.1 Entailments as morphisms

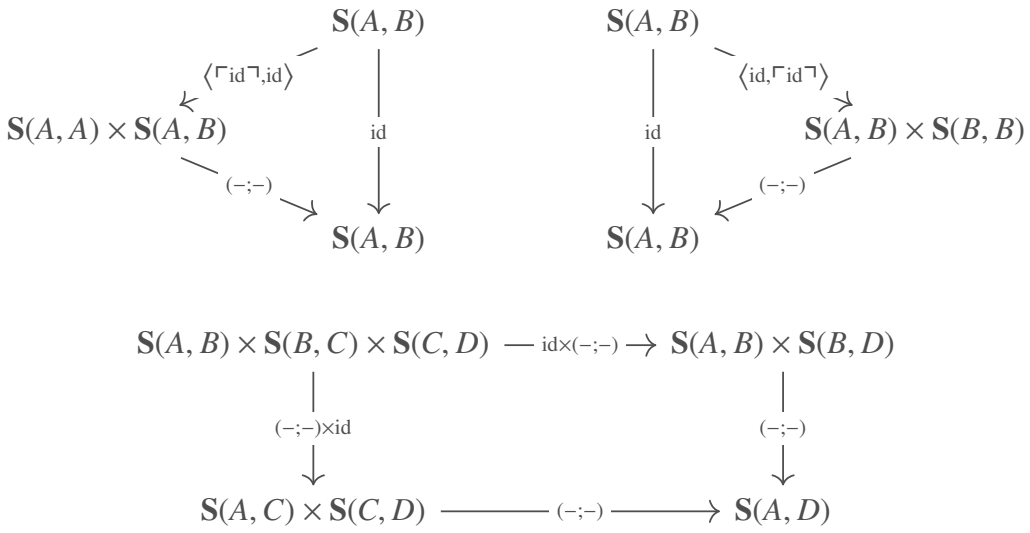
In categorical proof theory, logical sequents are treated as arrows in a category [58, 60, 61, 65]. The reflexivity and the transitivity of the entailment relation then correspond to the main categorical structures: the identities and the composition.

$$A \vdash A \qquad\qquad \begin{array}{c} 1 \\ \downarrow \\ \ulcorner \mathrm{id} \urcorner \\ \downarrow \\ \mathbf{S}(A, A) \end{array} \tag{1}$$

$$\begin{array}{ccc} A \vdash B & & B \vdash C \\ & A \vdash C & \end{array} \qquad\qquad \begin{array}{c} \mathbf{S}(A, B) \times \mathbf{S}(B, C) \\ \downarrow \\ (-;-) \\ \downarrow \\ \mathbf{S}(A, C) \end{array}$$

But while there is at most one sequent $A \vdash B$ for given A and B, there can be many arrows between A and B in a category. Categorical semantics of the logical entailment must therefore be imposed

by equations:



1.2.2 Conjunction and disjunction as product and coproduct

Algebraically, the conjunction and the disjunction are the meet and the join in the proposition lattice. Categorically, they are the product and the coproduct:

$$\frac{X \vdash A \qquad X \vdash B}{X \vdash A \wedge B} \qquad \begin{array}{c} \mathbf{S}(X, A) \times \mathbf{S}(X, B) \\ {\scriptstyle \langle -,- \rangle}\big\downarrow \quad \big\uparrow{\scriptstyle \langle \pi_A \circ -, \pi_B \circ - \rangle} \\ \mathbf{S}(X, A \times B) \end{array} \tag{2}$$

$$\frac{A \vdash X \qquad B \vdash X}{A \vee B \vdash X} \qquad \begin{array}{c} \mathbf{S}(A, X) \times \mathbf{S}(B, X) \\ {\scriptstyle [-,-]}\big\downarrow \quad \big\uparrow{\scriptstyle \langle -\circ \iota_A, -\circ \iota_B \rangle} \\ \mathbf{S}(A + B, X) \end{array} \tag{3}$$

Definition 1.1 *A category with the product constructor* × *supporting the correspondence* (2) *is called* cartesian*. A category with the coproduct constructor* +*supporting the correspondence* (3) *is called* cocartesian*.*

The difference between the algebraic and the categorical view is that in the first case there is at most one entailment $X \vdash A$, whereas in the second case there can be many arrows $X \to A$, usually labelled, and viewed as functions in the category $\mathbf{S}$. The mapping in (2) on the right establishes the bijection between the proofs or functions $X \to A \times B$ and the pairs of proofs or functions $X \to A$ and $X \to B$. The proof transformations thus become function manipulations. If the elements of sets or entries of data types, witness the corresponding propositions, then the logical operations are operations on data. E.g., proof of conjunction becomes a pair of data entries. It often comes as a surprise that such simple-minded analogies can be effective tools in functional programming [93]. They also have far-reaching logical consequences, some of which are pursued in the present paper.

1.2.3 Function abstraction and cartesian closed categories

The fact that the conjunction $A \wedge (-)$ is the right adjoint to the implication $A \supset (-)$ [65] means that the implication introduction and elimination can be expressed as the reversible logical rule in (4) on the left.

$$\frac{(A \wedge X) \vdash B}{X \vdash (A \supset B)}\,\supset \qquad\qquad \begin{array}{c} \mathbf{S}(A \times X, B) \\ {\scriptstyle (A\Rightarrow -)\circ \eta_X}\Big(\quad \Big)_{\scriptstyle \varepsilon_X \circ (A \times -)} \\ \mathbf{S}(X, (A \Rightarrow B)\,) \end{array} \tag{4}$$

The corresponding type-theoretic correspondence in (4) on the right was the first example of the propositions-as-types phenomenon. This bijection between two sets of proofs-as-terms was noticed by Haskell Curry back in the 1930s. The operation corresponding to the implication introduction, i.e. going down, is called the *abstraction*. The operation corresponding to the implication elimination, i.e. going up, is called the *application*. The categorical view of the resulting correspondence captures its uniformity with respect to all indexing types X, i.e. its *polymorphism*, as the *naturality* with respect to the type constructors $(A \times -)$ and $(A \Rightarrow -)$. A correspondence between two constructors is natural if it is preserved under all substitutions. For (4) every $f \in \mathbf{S}(X, Y)$ induces the two squares in (5), one formed by ηs, the other by εs.

$$\begin{array}{ccc} \mathbf{S}(A \times Y, B) & \xrightarrow{\;-\circ(A\times f)\;} & \mathbf{S}(A \times X, B) \\ {\scriptstyle (A\Rightarrow -)\circ \eta_Y}\Big(\quad \Big)_{\scriptstyle \varepsilon_Y \circ (A \times -)} & & {\scriptstyle (A\Rightarrow -)\circ \eta_X}\Big(\quad \Big)_{\scriptstyle \varepsilon_X \circ (A \times -)} \\ \mathbf{S}(Y, (A \Rightarrow B)\,) & \xrightarrow{\;-\circ f\;} & \mathbf{S}(X, (A \Rightarrow B)\,) \end{array} \tag{5}$$

The naturality of these squares means that they are commutative. The commutativity of these squares captures the type-theoretic *conversion rules* imposed on the abstraction operation and the application operation:

$$\begin{array}{ccccc} & & A \times (A \Rightarrow (A \times X)) & & \\ & {\scriptstyle A\times\eta_X}\nearrow & & \searrow{\scriptstyle \varepsilon_{A\times X}} & \\ A \times X & & \xrightarrow{\quad \mathrm{id} \quad} & & A \times X \\ A \;\Rightarrow\; X & & \xrightarrow{\quad \mathrm{id} \quad} & & A \;\Rightarrow\; X \\ & {\scriptstyle \eta_{(A\Rightarrow X)}}\searrow & & \nearrow{\scriptstyle (A\Rightarrow\varepsilon_X)} & \\ & & A \;\Rightarrow\; (A \times (A \Rightarrow X)) & & \end{array} \qquad \begin{array}{l} \Big(\lambda a.\; f_x(a)\Big) \cdot b = f_x(b) \qquad (\beta) \\ \\ \lambda a.\; \Big(g_x \cdot a\Big) = g_x \qquad (\eta) \end{array}$$

The application operation is derived from the adjunction counit $\varepsilon_X : A \times (A \Rightarrow X) \longrightarrow X$ in the form $g \cdot a = \varepsilon(a, g)$. The abstraction operation is written in type theory it is written using the variables, like in the rules (β) and (η) above, but can also derived from the adjunction unit $\eta_X : X \longrightarrow (A \Rightarrow (A \times X))$, in the form $\lambda(f) = (A \Rightarrow f) \circ \eta_X$.

Definition 1.2 *A* cartesian closed *category is a cartesian category* $\mathbf{S}$ *with the static implication* $(A \Rightarrow B)$ *for every pair of types* A, B *and the* X*-natural (function) abstraction operation*

$$\mathbf{S}(A \times X, B) \xrightarrow[\sim]{\lambda_X^{AB}} \mathbf{S}(X, (A \Rightarrow B)) \tag{6}$$

Remark. Towards aligning with Def. 2.1, note that the function abstraction λ_{AB} is with respect to the functors $H_{AB}, \nabla_{AB} \colon \mathbf{S}^o \longrightarrow \mathbf{S}$ defined

$$H_{AB}X = \mathbf{S}(A \times X, B) \qquad\qquad \nabla_{AB}X = \mathbf{S}(X, (A \Rightarrow B)$$

The arrow parts are induced by precomposition.

1.3 Modalities as monads and comonads

1.3.1 Possibility and side-effects

A possibility modality can be introduced by the rules on the left.

$$A \vdash \Diamond A \qquad\qquad \begin{array}{c} A \wedge B \vdash \Diamond C \\ \Diamond A \wedge \Diamond B \vdash \Diamond C \end{array} \qquad\qquad \begin{array}{c} \mathbf{S}(A \times B, MC) \\ {\scriptstyle \#\phi}\big(\quad \big\uparrow{\scriptstyle \eta\eta} \\ \mathbf{S}(MA \times MB, MC) \end{array} \tag{7}$$

Each of the logical rules corresponds to one of the categorical transformations on the right, where $\eta\eta$ precomposes $A \times B \xrightarrow{\eta_A \times \eta_B} MA \times MB \longrightarrow MC$, whereas $\#\phi$ first #-lifts $A \times B \xrightarrow{g} MC$, and then precomposes to $MA \times MB \xrightarrow{\phi} M(A \times B) \xrightarrow{g^\#} MC$. The quadruple $(M, \eta, \#, \phi)$ is the structure of a *commutative monad* [23, 53, 54, 68]. The type constructor M, the unit $\eta : A \longrightarrow MA$ and the lifting # of $A \xrightarrow{f} MC$ to $MA \xrightarrow{f^\#} MB$ satisfy the equations

$$\eta_A^\# = \mathrm{id}_{MA} \qquad\qquad f^\# \circ \eta_{A\times B} = f \qquad\qquad (f^\# \circ t)^\# = f^\# \circ t^\# \tag{8}$$

This triple is one of the equivalent presentations of the structure of a monad [68]. Most presentations [23, 63] define a monad as a triple (M, η, μ), where $\mu_A : MMA \longrightarrow MA$ are the (cochain) evaluators. The lifting # is derivable from the evaluators by setting $f^\# = \left(MA \xrightarrow{Mf} MMB \xrightarrow{\mu} MB\right)$, whereas the evaluators are derivable as the liftings of the identities, in the form $\mu_A = \left(MMA \xrightarrow{\mathrm{id}^\#} MA\right)$. The lifting operation # seems more convenient for programming. The last component ϕ of the structure $(M, \eta, \#, \phi)$ (or of the equivalent form (M, η, μ, ϕ)) is the *bilinearity* $\phi_{AB} : MA \times MB \longrightarrow M(A \times B)$, which makes the monad commutative [53, 54]. This natural family satisfies

$$\phi_{AC} \circ (\eta_A \times \eta_C) = \eta_{A\times C} \qquad\qquad (\phi_{BD} \circ (f \times g))^\# \circ \phi_{AC} = \phi_{BD} \circ (f^\# \times g^\#) \tag{9}$$

for all pairs $f : A \longrightarrow MB$ and $g : C \longrightarrow MD$. Similar equations are valid for all tuples. The equations in (8) define the identities and the composition in the category of free algebras (in the Kleisli form)

$$\begin{aligned} |\mathbf{S}_M| &= |\mathbf{S}| \\ \mathbf{S}_M(A, B) &= \mathbf{S}(A, MB) \end{aligned}$$

While correspondences (3) persist in $\mathbf{S}_M$, the natural bijection in (2) does not, and $\mathbf{S}_M$ is not a cartesian category any more. However, the equations in (9) assure that the product $\times$ from $\mathbf{S}$ persists as a monoidal structure in $\mathbf{S}_M$ [54]. Intuitively, a function $A \longrightarrow MB$ produces not just the outputs in B, but also some *side-effects* [73], represented in the type MB. E.g., the fact that computations may not terminate means that they implement functions in the form $A \longrightarrow B_\perp$ where the monad

$$\begin{aligned} (-)_\perp : \mathbf{S} &\longrightarrow \mathbf{S} \\ X &\mapsto X \cup \{\perp\} \end{aligned} \tag{10}$$

adjoins to each set a fresh element $\perp$. This is the *maybe* monad, corresponding to the algebraic theory with a single constant and no operations or equations. The category $\mathbf{S}_\perp$ is (equivalent to) the category of sets and partial functions.

Another side-effect of interest is the *nondeterminism*. Some computations may depend on the states of the computer, which may depend on the environment. Running the same program on the same inputs may therefore produce different outputs at different times, for no unobservable reason. Such computations implement functions in the form $A \longrightarrow \wp B$, where the monad

$$\begin{aligned} \wp : \mathbf{S} &\longrightarrow \mathbf{S} \\ X &\mapsto \{V \subseteq X\} \end{aligned} \tag{11}$$

maps each set to the set of its subsets, a.k.a. its *powerset*. This is the *nondeterminism* (or powerset) monad. It maps to every function $X \xrightarrow{g} Y$ the function $\wp X \xrightarrow{\wp g} \wp Y$, which takes $V \subseteq X$ to $\wp g(V) = \{g(x) \in Y \mid x \in V\}$. The unit $X \xrightarrow{\eta} \wp X$ maps $x \in X$ to $\eta(x) = \{x\}$. The lifting maps a function $A \xrightarrow{f} \wp B$ to $\wp A \xrightarrow{f^\#} \wp B$ where $f^\#(V) = \cup_{v \in V} f(v)$. For reasons discussed in Appendix A, it satisfies

$$\mathbf{S}(A, \wp B) \cong \mathbf{S}(B, \wp A)$$

which makes the category $\mathbf{S}_\wp$ of nondeterminisic functions self-dual, along the natural bijection $\mathbf{S}_\wp(A, B) \cong \mathbf{S}_\wp(B, A)$. The idea is that, given a nondeterministic function $A \longrightarrow \wp B$, knowing all possible B-outputs for each A-input allows us to extract all possible A-inputs for each B-output, which yilelds just another nondeterministic function $B \longrightarrow \wp A$. See Appendix A for more.

Notation. Since they will be cast in leading roles, the above categories of functions with effects are abbreviated to:

- $\mathbf{P} = \mathbf{S}_\perp$ — category of partial functions, and
- $\mathbf{R} = \mathbf{S}_\wp$ — category of relations.

Background. In mathematics, monads emerged as a "standard construction" of free algebras involving topologies [23, 68]. The observation that the type constructors M that add side-effects also carry the monad structure goes back to [78]. Initially proposed as a semantical tool, monads turned out to be a powerful and convenient programming tool. Nowadays, monads' popularity among programmers drives interest in semantics. Mathematically, a monad M is a saturated view of an algebraic theory, presented not by operations and equations, but as a mapping from any set of generators B to the free algebra MB. The unit η maps each generator to its place in the free algebra. The lifting # expands the assignment $A \xrightarrow{f} MB$ from the generators A to the algebra homomorpism $MA \xrightarrow{f^\#} MB$. Any monad corresponds to an algebraic theory, albeit with infinitary operations. The semantical assumption that all computational side-effects can be captured by algebraic operations has deep repercussions on the concept of computation.

1.3.2 Necessity and reductions

Dually, a necessity modality can be introduced by

$$\Box A \vdash A \qquad \Box A \vdash B \vee C \qquad \Box A \vdash \Box B \vee \Box C \qquad \begin{array}{c} \mathbf{S}(GA, B+C) \\ {\scriptstyle \#}\big(\quad \big)_{-\circ\varepsilon_{B+C}} \\ \mathbf{S}(GA, GB+GC) \end{array} \tag{12}$$

This time the triple $(G, \varepsilon, \#)$ is made into a comonad by the equations:

$$\varepsilon_A^\# = \mathrm{id}_{GA} \qquad \varepsilon_{B+C} \circ f^\# = f \qquad (f \circ t^\#)^\# = f^\# \circ t^\#$$

The third equation defines the composition in the category of coffee coalgebras, in the *Kleisli* form again:

$$\begin{array}{rcl} |\mathbf{S}_G| & = & |\mathbf{S}| \\ \mathbf{S}_G(A,B) & = & \mathbf{S}(GA, B) \end{array}$$

Computational interpretations of comonads are less standard, but overviews can be found in [25, 108]. We will need a *history comonad* to capture the time extension of processes in Sec. 2.3.1. For the moment, let us just mention the *indexing* comonads

$$\begin{array}{rcl} A \times (-) : \mathbf{S} & \longrightarrow & \mathbf{S} \\ X & \mapsto & A \times X \end{array} \tag{13}$$

which exist for each $A \in \mathbf{S}$, with the counits $A \times X \xrightarrow{\varepsilon} X$ realized by the projections, and the lifting $A \times X \xrightarrow{h} Y + Z$ defined to be $A \times X \xrightarrow{\langle \mathrm{id}_A, h \rangle} A \times (Y+Z) \cong (A \times Y) + (A \times Z)$. The Kleisli category $\mathbf{S}_{A\times}$ freely adjoins an indeterminate arrow $1 \xrightarrow{x} A$ to $\mathbf{S}$, and plays the role of the polynomial extension $\mathbf{S}[x : A]$ [63, 88]. Like any Kleisli category, $\mathbf{S}_{A\times}$ provides a *resolution* of its comonad, in the sense that it factors through the functors

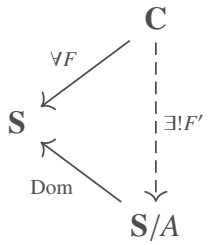
$$A \times (-) \quad = \quad \left(\mathbf{S} \xrightarrow{-\circ\varepsilon} \mathbf{S}_{A\times} \xrightarrow{\#} \mathbf{S} \right)$$

as displayed in (12). While the Kleisli resolution is *initial* among the resolutions of the comonad $A \times (-)$, some of the constructions in this paper are built upon the fact that the resolution

$$A \times (-) \quad = \quad \left(\mathbf{S} \xrightarrow{\Pi_A} \mathbf{S}/A \xrightarrow{\mathrm{Dom}} \mathbf{S}\right)$$

is *final* among all resolutions. Here $\mathbf{S}/A$ is the category of $\mathbf{S}$-morphisms into A, the functor Π_A functor maps X to the projection $A \times X \xrightarrow{\pi_A} A$, whereas the Dom functor Dom takes the $\mathbf{S}/A$-objects, which are the $\mathbf{S}$-morphisms with the codomain A, to their domains $\mathrm{Dom}(X \longrightarrow A) = X$.

Lemma 1.3 *The domain functor* $\mathrm{Dom} : \mathbf{S}/A \longrightarrow \mathbf{S}$ *is final among all functors* $F : \mathbf{C} \longrightarrow \mathbf{S}$ *which map the terminal object* 1 *into* A.



Proof. Given F with $F1 = A$, the unique F' with $\mathrm{Dom} \circ F' = F$ is $F'X = F(X \xrightarrow{!} 1)$. □

1.4 Labelled sequents, commutative monads, and surjections

In propositional logic, a sequent $X \vdash Y$ transforms proofs of X into proofs of Y. If there are several different ways to derive one from the other, the sequent $X \vdash Y$ identifies them all. This leads to a mismatch within the propositions-as-types interpretation because it implies that there is at most one proof $X \vdash A \supset B$, while there can be many different terms of type $X \xrightarrow{t} (A \Rightarrow B)$. This mismatch is resolved by labelling the sequents, writing $X \overset{t}{\vdash\!\!\rightarrow} A \supset B$ for the former sequent. We use the symbol $\vdash\!\!\rightarrow$ (and not $\vdash$) for labelled sequents, to be able to write $X \vdash\!\!\rightarrow Y$ instead of $X \overset{f}{\vdash\!\!\rightarrow} Y$ when the label f is irrelevant. The categorical proof theory originates from studies of labelled sequents in [58, 60, 61]. A non-categorical theory of labelled sequents was developed in [39].

For a modality $\Diamond$, the sequents between the propositions $\Diamond A \wedge \Diamond B$ and $\Diamond(A \wedge B)$ are derivable both ways, and the two are considered equivalent. The proposition $\Diamond\top$ is also equivalent to the truth $\top$. For a monad M, the maps $M(A \times B) \xrightarrow{\langle M\pi_A, M\pi_B\rangle} MA \times MB$ and $M1 \xrightarrow{!} 1$ are derivable from the cartesian structure, and the maps $MA \times MB \xrightarrow{\phi} M(A \times B)$ and $1 \xrightarrow{\eta} M1$ are given by the monad structure. However, these pairs of functions both ways are generally not inverse to one another. The type $M1$ is generally not final, and the type $M(A \times B)$ is generally not a product. The side-effects of type $M(A \times B)$ are different from the side-effects that arise when the outputs are received into MA and MB separately.

While this state of affairs is justified in algebra, where $M1$ is the free algebra over a single generator, it is not justified in the semantics of computation, where the trivial outputs of type 1 should not cause nontrivial side effects contained in type $M1$. Viewing the monad M as an

algebraic theory shows that the nontrivial elements of $M1$ arise from the constants of the algebraic theory. This requirement is not satisfied either by the maybe monad, or by the nondeterminism monad, as the former gives the universe $\mathbf{P} = \mathbf{S}_\perp$ of sets and partial maps, the latter the universe $\mathbf{R} = \mathbf{S}_\wp$ of sets and relations. The former is the category of free algebras for the theory with a single constant $\perp$, and no other operations. The latter is the category of free join semilattices, where the lattice unit is a constant again.

Lemma 1.3 says that making 1 into the unit type (final object) in $\mathbf{R} = \mathbf{S}_\wp$ leads to the slice category $\mathbf{tR} = \mathbf{R}/1$, which boils down to

$$\begin{aligned} |\mathbf{tR}| &= \coprod_{A \in |\mathbf{S}|} \wp A \\ \mathbf{tR}(S_{\subseteq A}, T_{\subseteq B}) &= \Big\{ R \in \mathbf{R}(A,B) \mid (x \in S \iff \exists y \in T.\ xRy) \ \wedge \\ &\qquad \wedge\ (y \in T \iff \exists x \in S.\ xRy) \Big\} \end{aligned} \tag{14}$$

Since the $\Rightarrow$-direction of each of the conjuncts in (14) implies the $\Leftarrow$-direction of the other conjunct, the requirement boils down to $\forall x \in S\, \exists y \in T.\ xRy$ and $\forall y \in T \exists x \in S.\ xRy$. The category $\mathbf{tR}$ is thus equivalent to the subcategory of $\mathbf{R}$ comprised of the relations that are total in both directions. Proceeding in a similar way to make 1 into the final type in the category $\mathbf{S}_\perp = \mathbf{P}$ leads to the slice category $\mathbf{tP} = \mathbf{P}/1$, which is equivalent to the subcategory $\mathbf{tS}$ of $\mathbf{S}$ spanned by the surjective functions:

$$\begin{aligned} |\mathbf{tP}| &= \coprod_{A \in |\mathbf{S}|} \wp A \\ \mathbf{tP}(S_{\subseteq A}, T_{\subseteq B}) &= \Big\{ f \in \mathbf{S}(S,T) \mid y \in T \Rightarrow \exists x \in S.\ f(x) = y \Big\} \end{aligned} \tag{15}$$

Remark for the category theorist. The forgetful functor $\mathbf{tP} \longrightarrow \mathbf{tS}$, where $\mathbf{tS}$ is the category of sets and surjections, is an equivalence because it is surjective on the objects, and full and faithful on the morphisms. However, for each set $S \in \mathbf{S}$ there is a proper class of sets A such that $S_{\subseteq A} \in \mathbf{tP}$ is mapped to $S \in \mathbf{tS}$. Constructing the adjoint equivalence $\mathbf{tS} \longrightarrow \mathbf{tP}$ thus involves a choice from these proper classes of objects.

2 Process logics

2.1 Idea of process

The alignment of logic and type theory remains stable as long as the world is stable: the true propositions remain true, and the data types remain as given. The problems arise when processes need to be modeled, and their dynamic aspects need to be taken into account.

There are physical processes, chemical processes, mental processes, social processes. The common denominator is that they change state: a physical process changes the state of matter; a mental process changes the state of mind. Computation is also a process. Although already a local execution of a program changes the local states of a computer, it seems that the crucial aspects of

processes of computation arise from their interleaving with the processes of communication, from the resulting computational interactions, and only emerge into light when the problem of concurrency is taken into account. That is why the semantics of computational processes, formalized in process calculi, initially forked off from the main branch of the semantics of programming languages. The main part of Samson Abramsky's work, which I am trying to reconstruct in logical terms, was concerned with bringing the two branches together.

2.2 Process propositions and implications

2.2.1 Process sequents must be labelled

Process logics involve modeling states. There are many different ways to model states, but within a propositions-as-types framework, state spaces occur together with the data types, subject to the same derivation rules. Although the two must be treated differently within the rules (as we shall see already in Sec. 2.4), both require *labelled* sequents. For state spaces, this is clearly unavoidable. As mentioned in Sec. 1.4, an unlabelled sequent $X \vdash Y$ identifies all different proofs that X entails Y. In particular, there is just one entailment $X \vdash X$, the trivial one. But if X is a state space, then modeling state transitions requires nontrivial sequents $X \overset{\xi}{\mapsto} X$. The labels allow distinguishing the nontrivial sequents, where the states change, from the trivial one, where they do not.

2.2.2 Machine abstraction and process-closed categories

A process implication $[A, B]$ asserts not just that A implies B, but also that A implies $[A, B]$. Under the propostions-as-types interpretation, the type $[A, B]$ thus comes with two functions

- $A \wedge [A, B] \overset{\varepsilon}{\mapsto} B$ $(\upsilon^\bullet)$
- $A \wedge [A, B] \overset{\zeta}{\mapsto} [A, B]$ (υ°)

The label says that the latter is not an instance of the propositional conjunction elimination, a.k.a. projection on the types side. The sequent ζ is a *coinductive* clause, saying that $[A, B]$ is true on its own whenever it is true together with A as a *guard* [35, 89]. This is a typical *impredicative* claim, of kind which is often used mathematical analysis [97, 98, 100]. The general idea is that, whenever a proposition X, guarded by a proposition A, entails a proposition B, *and moreover also itself*, i.e. whenever X comes with the sequents

- $A \wedge X \mid\!\!\rightarrow B$ $(\llbracket - \rrbracket^\bullet)$
- $A \wedge X \mid\!\!\rightarrow X$ $(\llbracket - \rrbracket^\circ)$

then X also entails the process implication $[A, B]$. Putting it all together, we get the following rules:

$$A \wedge [A, B] \overset{\upsilon}{\mapsto} B \wedge [A, B] \;{}^{\upsilon} \qquad \frac{A \wedge X \overset{\varphi}{\mapsto} B \wedge X}{X \overset{\llbracket \varphi \rrbracket}{\longmapsto} [A, B]}\;{}^{\llbracket - \rrbracket} \qquad \begin{array}{c} \mathbf{S}(A \times X, B \times X) \\ \Big\downarrow{\scriptstyle \llbracket - \rrbracket_X} \\ \mathbf{S}\,(X, [A, B]) \end{array} \tag{16}$$

Terminology. A function in the form $\xi : A \times X \longrightarrow B \times X$ is often called a *machine*, and the set X is construed as its *state space*. The induced description $[\![\xi]\!] : X \longrightarrow [A, B]$ is called *anamorphism*.[2]

Naturality. Comparing the $[\![-]\!]$-rule with the $(\supset)$-rule in Sec. 1.2.3, shows the sense in which $[A, B]$ is a dynamic version of the implication $(A \supset B)$. But note that the rule $(\supset)$ is reversible, whereas the rule $[\![-]\!]$ is not; and that the X-natural bijection in (4) on the right boils down to an X-natural transformation on the right in (16). Moreover, since X occurs on both sides of the sequent $A \wedge X \overset{\varphi}{\vdash\!\!\rightarrow} B \wedge X$, and thus in both covariant and contravariant position in $\mathbf{S}(A \times X, B \times X)$, the naturality of $[\![-]\!]_X$ is not as simple as in (5), and it genuinely adds to the story. This time the naturality is in the form

$$\begin{array}{ccccc} \mathbf{S}\,(A \times Y, B \times Y) & \longleftarrow & \Theta_{AB}f & \longrightarrow & \mathbf{S}\,(A \times X, B \times X) \\ \Big\downarrow{\scriptstyle [\![-]\!]_Y} & & & & \Big\downarrow{\scriptstyle [\![-]\!]_X} \\ \mathbf{S}\,(Y, [A, B]) & \longrightarrow & (-\circ f) & \longrightarrow & \mathbf{S}\,(X, [A, B]) \end{array} \tag{17}$$

where Θ_{AB} is the functor

$$\begin{array}{rcl} \Theta_{AB} \;:\; \mathbf{S}^o & \longrightarrow & \mathbf{R} \\ X & \mapsto & \mathbf{S}(A \times X, B \times X) \end{array} \tag{18}$$

where $\mathbf{R}$ is the category of sets and relations, described in Appendix A. The arrow part of this functor transforms a function $f \in \mathbf{S}(X, Y)$ into the relation $\Theta_{AB}f = (f)$ which is a subset of $\mathbf{S}(A \times Y, B \times Y) \times \mathbf{S}(A \times X, B \times X)$ defined by

$$\zeta(f)\xi \quad \Longleftrightarrow \quad \begin{array}{ccc} A \times Y & \leftarrow {\scriptstyle A\times f} - & A \times X \\ \Big\downarrow{\scriptstyle \zeta} & & \Big\downarrow{\scriptstyle \xi} \\ B \times Y & \leftarrow {\scriptstyle B\times f} - & B \times X \end{array} \tag{19}$$

The relation $(- \circ f)$ in (17) is the arrow part of the functor

$$\begin{array}{rcl} \nabla_{AB} : \mathbf{S}^o & \longrightarrow & \mathbf{R} \\ X & \mapsto & \mathbf{S}(X, [A, B]) \end{array} \tag{20}$$

[2]Anamorphisms are the coalgebra homomorphisms into final coalgebras. The name is due, I believe, to Lambert Meertens. It seems to have caught on without having been introduced in a publication. Many functional programmers call them *unfolds*, generalizing the special case of lists. A machine $A \times X \longrightarrow B \times X$ can be viewed as a coalgebra $X \longrightarrow (A \Rightarrow (B \times X))$.

where $\nabla_{AB} f = (- \circ f)$ is the subset of $\mathbf{S}(Y, [A, B]) \times \mathbf{S}(X, [A, B])$ defined

$$y(- \circ f)x \quad \Longleftrightarrow \quad y \circ f = x \tag{21}$$

∇_{AB} is, of course, just homming into $[A, B]$, i.e. a functor to the category of sets $\mathbf{S}$ extended along the inclusion $\mathbf{S} \hookrightarrow \mathbf{R}$ of functions as special relations. The naturality of $[\![-]\!] : \Theta_{AB} \longrightarrow \nabla_{AB}$ genuinely depends on this casting. It says that $[\![-]\!]$ must preserve the machine (i.e. coalgebra) homomorphisms specified in (19). The concept of an AB-machine homomorphism has herewith been reconstructed logically, from the properties of the dynamic implication $[A, B]$ in (16).

To reconstruct the structure of final AB-machine, substitute $[A, B]$ for Y in (16), to get the outer square in

$$\begin{array}{ccccc}
\mathbf{S}\left(A \times [A,B]\,, B \times [A,B]\right) & \longleftarrow & ([\![\varphi]\!]) & \longrightarrow & \mathbf{S}\left(A \times X, B \times X\right) \\
\Big\downarrow{\scriptstyle [\![-]\!]} & & \upsilon \longmapsto \varphi & & \Big\downarrow{\scriptstyle [\![-]\!]} \\
 & & \Big\downarrow \quad \Big\downarrow & & \\
 & & \mathrm{id}_{[A,B]} \longmapsto [\![\varphi]\!] & & \\
\mathbf{S}\left([A,B]\,, [A,B]\right) & \longrightarrow & (-\circ[\![\varphi]\!]) & \longrightarrow & \mathbf{S}\left(X, [A,B]\right)
\end{array} \tag{22}$$

The inner square says that, if we bind together the two left-hand rules in (16) by requiring that

$$[\![\upsilon]\!]_{[A,B]} \quad = \quad \mathrm{id}_{[A,B]}$$

then the naturality requirement in (16) implies that $A \times [A, B] \xrightarrow{\upsilon} B \times [A, B]$ is final among all AB-machines. This is conveniently summarized in the next definition, intended for the readers with categorical background.

Definition 2.1 *A* process closed *category is a cartesian category* $\mathbf{S}$ *with a process implication* $[A, B]$ *for any pair of types* A, B *and the* X*-natural* machine abstraction *operaton*

$$\mathbf{S}(A \times X, B \times X) \quad \xrightarrow{[\![-]\!]^{AB}_X} \quad \mathbf{S}(X, [A, B]) \tag{23}$$

The naturality of $[\![-]\!]^{AB}$ *is with respect to the functors* $\Theta_{AB}, \nabla_{AB} \colon \mathbf{S}^o \longrightarrow \mathbf{R}$ *defined in (18–21).*

Remark. Def. 2.1 can be viewed as a lifting of Def. 1.2 to process logics. While the latter is the categorical setting of the static propositions-as-types paradigm, the former recasts categories with final AB-machines in a logical form. The simple logical relation between the two structures will be spelled out in Prop. 2.3.3.

2.2.3 Process propositions

A static proposition B is equivalent with the static implication $\top \supset X$, where $\top$ is the true proposition. All propositions can thus be viewed as special implications: namely the implications from the truth. A dynamic proposition $[B]$ can thus be defined in the form $[B] = [\top, B]$. Since the conjunctions $\top \wedge X$ are also equivalent with X, dynamic propositions can be defined by the rules

$$[B] \overset{\upsilon}{\vdash\!\!\to} B \wedge [B] \quad {}^{\upsilon} \qquad\qquad X \overset{\beta}{\vdash\!\!\to} B \wedge X \quad {}^{[\![-]\!]} \qquad\qquad \begin{array}{c} \mathbf{S}(X, B \times X) \\ \big| \\ {}^{[\![-]\!]} \\ \downarrow \\ \mathbf{S}(X, [B]) \end{array}$$

$$X \overset{[\![\beta]\!]}{\longmapsto} [B]$$

Retracing the analysis from Sec. 2.2.2 now presents a proposition $[B]$ with a structure map $[B] \overset{\upsilon}{\to} B \times [B]$, as final among all maps in the form $X \longrightarrow B \times X$. The structure map is thus a pair $\upsilon = \langle \upsilon^\bullet, \upsilon^\circ \rangle$, where $\upsilon^\bullet : [B] \longrightarrow B$ gives an output of the process proposition, or an action, and $\upsilon^\circ : [B] \longrightarrow [B]$ gives a resumption. It is thus a stream of elements in B.

2.3 Relating process implications and static implications

The static implication is defined by the rules and the correspondence in (4). The process implication is defined by the rules and the correspondence in (16). How are they related? Under which conditions are both sets of rules supported? Prop. 2.3.3 provides an answer. Sections 2.3.1 and 2.3.2, introduce the structures involved in the answer.

2.3.1 History types

A *process of A-histories over a state space X* is a pair of functions $\kappa = \langle \kappa_{(-)}, \kappa_{(::)} \rangle$ typed

$$A \overset{\kappa_{(-)}}{\longrightarrow} X \overset{\kappa_{(::)}}{\longleftarrow} A \times X \tag{24}$$

The idea is that,

- $\kappa_{(-)}(a) \in X$ is the initial state of a process that starts with $a \in A$;
- $\kappa_{(::)}(x, a) \in X$ is the next state of a process after the state $x \in X$ and event or action $a \in A$.

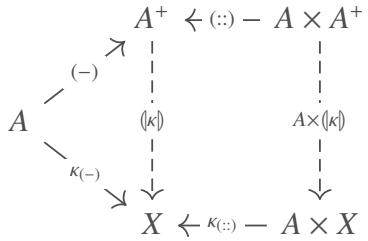
A history $a^n = (\, a_1\ a_2 \cdots a_n\,)$ thus takes the process κ to the state

$$x_n \quad = \quad \kappa_{(::)}(a_n, \kappa_{(::)}(a_{n-1}, \ldots \kappa_{(::)}(a_1, \kappa_{(-)}(a_0)) \cdots))$$

Each string of n actions, construed as an A-history is thus mapped to a unique element of X. If the histories $(\, a_1 \cdots a_n\,)$ are viewed as the elements of A^n, then the disjoint union (coproduct)

$$A^+ \quad = \quad \coprod_{n=1}^{\infty} A^n$$

is the type of all A-histories. This is what we call a *history type*. For any process of A-histories κ over X there is a unique *banana-function* (a.k.a. *fold*, or *catamorphism*) $A^+ \xrightarrow{(\!|\kappa|\!)} X$ that makes following diagram commute.

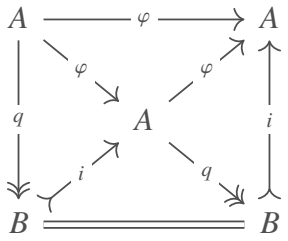


Hence the history type constructor, the functor

$$\begin{array}{rccc} (-)^+ : & \mathbf{S} & \longrightarrow & \mathbf{S} \\ & A & \mapsto & A^+ \end{array} \tag{25}$$

2.3.2 Retractions and idempotents

A *retraction* is a pair of maps $A \underset{i}{\overset{q}{\rightleftarrows}} B$ such that $q \circ i = \mathrm{id}_B$. The type B *is a retract of* A when there is such a pair. It is easy to see that the composite $\varphi = i \circ q$ is then idempotent, and the retraction $A \underset{i}{\overset{q}{\rightleftarrows}} B$ is its *splitting*. The following diagram summarizes a retraction



It is easy to see that i is then an equalizer of φ and the identity; and that q is a coequalizer of the same pair. Since any functor preserves retractions, they provide an exampe of an *absolute limit and colimit*. A categorical construction is *absolute* when it is preserved by all functors. It turns out that all absolute limits and colimits boil down to retractions [82]. A category where all idempotents split into retractions is thus *absolutely complete*. The *absolute completion* of a category takes its idempotents as the objects, and a morphism f between the idempotents φ and ψ is required to preserve them, in the sense that $f = \psi \circ f \circ \varphi$. This is the weakest form of a categorical completion. Retractions, or idempotent splittings[3], are thus an instance of the (co)limit operation.

The following proposition is a first step towards expanding the propositions-as-types paradigm to processes, promised in the title of this paper.

[3]While the term *idempotent splitting* is well-established in category theory, the term *retraction* is familiar in mathematics at large. They refer to the same fundamental operation [96, Sec. 11].

2.3.3 Proposition

Let **S** *be a cartesian category. Then the static implications and the process implications induce each other in the presence of the history types and the retractions. More precisely,*

a) a cartesian closed category is process closed whenever it has the history types;

b) a process closed category is cartesian closed whenever it has the retractions.

The **proof** is given in the Appendix.

Process abstraction is function abstraction over history types. Prop. 2.3.3 says that a cartesian closed category with history types has final AB-machines for all types A and B, and that their state spaces $[A, B] = (A^+ \Rightarrow B)$ support rules (16). A final AB-state machine can be constructed as a final coalgebras for the functor

$$\begin{array}{rcl} E_{AB} : \mathbf{S} & \longrightarrow & \mathbf{S} \\ X & \mapsto & (A \Rightarrow (B \times X)) \end{array}$$

i.e. as a limit of the tower in the form

$$\begin{array}{ccccccccc} 1 & \xleftarrow{\;!\;} & (A \Rightarrow B) & \xleftarrow{(A \Rightarrow (B \times !))} & (A \Rightarrow (B \times (A \Rightarrow B))) & \leftarrow \cdots & & & \\ & & \vdots & & & & & & \\ & & E^n_{AB}(1) & \xleftarrow{E^n_{AB}(!)} & E^{n+1}_{AB}(1) & \leftarrow \cdots & (A^+ \Rightarrow B) & & \end{array} \tag{26}$$

The process implications $[A, B]$ are thus modeled together with the static implications $(A \Rightarrow B)$, and both sets of rules (4) and (16) are supported. Processes can thus be modeled as machines. This was indeed the starting idea of process semantics [73]. However, early on along this path, it becomes clear that many different machines implement indistinguishable processes. The problem of process equivalence arises [75]. The input and the output types A and B of a process are observable, but the state space X may not be. In fact, any observable behavior can be realized over many different, unobservable state spaces.

2.4 The problem of cut in process logics

The fact that a process model may not support process composition is not just a conceptual shortcoming, but an obstacle to applications. Engineering tasks are in principle made tractable by decomposing the required processes into components, implementing the components, and composing the implementations. The process component models thus usually encapsulate and hide implementations, and display the interfaces. This methodology is conceptualized in *full abstraction*, one of the tenets of semantics of computation ever since [73, Sec. 4].

The first logical requirement of compositionality is that the state spaces must be factored out. This is necessary if the composition is to comply with a cut rule (1). If the process sequents are

state machines in the form $X \times A \overset{\varphi}{\longmapsto} X \times B$ and $Y \times B \overset{\psi}{\longmapsto} Y \times C$, then the cut rule would be something like

$$\begin{array}{c} X \wedge A \overset{\varphi}{\longmapsto} X \wedge B \qquad\qquad Y \wedge B \overset{\psi}{\longmapsto} Y \wedge C \\ Z \wedge A \overset{(\varphi;\psi)}{\longmapsto} Z \wedge C \end{array} \tag{27}$$

The mismatch between the state spaces X and Y needs to be somehow resolved by composite state space Z. How should processes pass their internal states to one another?

The intuitive difference between data and states is that data are processed, whereas the states enable the processing. The structural difference is that data can be copied and sent in messages, whereas the states are internal, and may not be communicable. The problem of process composition is thus that the *observable* aspects of processes, that get passed in process composition from one process to another, need to be separated from the unobservable aspects, that remain hidden from the compositions. The same problem arises in applying processes as dynamic functions on sources as dynamic elements.The latter can, of course, be viewed as a special case of the former, just like process propositions are viewed as a special case of process implications.

The idea towards a solution is that the observable aspects are presented as data types, the unobservable aspects as state spaces. Processes should thus keep their internal states for themselves, as any dynamics aspects of their interactions can be communicated using the process implications. This follows from the fact, spelled out at the end of Sec. 2.2.2, that the process implications are the state spaces of the final state machines. Dispensing with the internal states, the process composition should thus be defined as a sequent in the form

$$[A, B] \wedge [B, C] \quad \overset{\gamma}{\longmapsto} \quad [A, C] \tag{28}$$

In the static logics, the sequents that establish the transitivity of implication are equivalent with the cut rule from (1). In the process logics, the sequents like (28) solve the problem with (27). The final machine and coalgebra structures carried by the process implications have been used to define composition in a variety of final-coalgebra-enriched categories [3, 11, 95, 57]. The general derivation pattern behind the composition sequents in the form (28) is something like this:

$$\begin{array}{c} A \wedge [A, B] \overset{\upsilon}{\longmapsto} B \wedge [A, B] \quad {}^{\upsilon} \qquad\qquad B \wedge [B, C] \overset{\upsilon}{\longmapsto} C \wedge [B, C] \quad {}^{\upsilon} \\ A \wedge [A, B] \wedge [B, C] \overset{\alpha}{\longmapsto} B \wedge [A, B] \wedge [B, C] \qquad B \wedge [A, B] \wedge [B, C] \overset{\beta}{\longmapsto} C \wedge [A, B] \wedge [B, C] \\ A \wedge [A, B] \wedge [B, C] \overset{(\alpha;\beta)}{\longmapsto} C \wedge [A, B] \wedge [B, C] \quad {}^{[\![-]\!]} \\ [A, B] \wedge [B, C] \overset{\gamma = [\![\alpha;\beta]\!]}{\longmapsto} [A, C] \end{array} \tag{29}$$

The task of composing processes thus boils down to interpreting the process implications $[A, B]$. The task of applying processes to sources boils down to interpreting the process propositions $[A] = [\top, A]$.

3 Functions extended in time

3.1 Dynamic elements as streams

The outputs of a machine $a = \left(X \xrightarrow{\langle a^\bullet, a^\circ\rangle} A \times X\right)$ are observable as a stream $a^\omega = (\, a_0\, a_1 \cdots a_n \cdots)$. Starting from an initial state $x_0 \in X$ the process

- outputs $a_0 = a^\bullet_{x_0}$ and updates the state to $x_1 = a^\circ_{x_0}$; then it
- outputs $a_1 = a^\bullet_{x_1}$ and updates the state to $x_2 = a^\circ_{x_1}$; after n steps, it
- outputs $a_n = a^\bullet_{x_n}$ and updates the state to $x_{n+1} = a^\circ_{x_n}$; and so on.

A dynamic[4] element can thus be construed as a stream of outcomes of a repeated measurement or count. Such data streams arise in science, and they are the subject of statistical inference [36]. If the outcomes are the truth values, then these streams are the process propositions. When the frequencies are counted, then they are the streams of random variables called *sources* in information theory [20, Ch. 6].

3.2 Functions extended in time as deterministic channels

A dynamic function from A to B is generated by a machine in the form $f = \left(A \times X \xrightarrow{\langle f^\bullet, f^\circ\rangle} B \times X\right)$. Starting from an initial state $x_0 \in X$ the process consists of the following data maps and state updates:

$$\begin{array}{rr} a_0 \mapsto b_0 = f^\bullet_{x_0}(a_0) & a_0 \mapsto x_1 = f^\circ_{x_0}(a_0) \\ a_0\, a_1 \mapsto b_1 = f^\bullet_{x_1}(a_1) & a_0\, a_1 \mapsto x_2 = f^\circ_{x_1}(a_1) \\ \cdots & \cdots \\ a_0\, a_1 \cdots a_n \mapsto\ b_n = f^\bullet_{x_n}(a_n) & a_0\, a_1\ \cdots a_n \mapsto x_{n+1} = f^\circ_{x_n}(a_n) \\ \cdots & \cdots \end{array}$$

A dynamic function can thus be viewed as a stream of functions in the form

$$f^\omega \;=\; \left(\, f_0\, f_1\, \cdots f_n \cdots \right) \qquad \text{where} \qquad f_n = f^\bullet_{x_n} : A^n \longrightarrow B$$

The propositions-as-types interpretation of the process implication is based on such streams of functions. Streams of random functions are studied in information theory as channels. Those considered here correspond to the *deterministic* channels [20, Sec. 3.2].

[4]We use the terms *"dynamic"* and *"extended in time"* interchangeably. A distinguishing aspect, that justifies keeping both in traffic, will emerge later.

3.3 History monad and comonad

The construction $(-)^+ : \mathbf{S} \longrightarrow \mathbf{S}$, described in Sec. 2.3.1, supports the monad structure

$$A \xrightarrow{\eta} A^+ \qquad\qquad A^+ \xleftarrow{g^\#} B^+$$

$$a \longmapsto \Big(a\Big) \qquad\qquad g(b_1) \cdot g(b_2) \cdots g(b_n) \longmapsfrom \Big(a_1\ a_2 \cdots a_n\Big)$$

where g is a function from B to A^+, and $\cdot$ is the string concatenation. The algebras for this monad are semigroups: the set of finite A-sequences (words, nonempty lists) A^+ is the free semigroup over A.

For our concerns, it is more interesting that the construction $(-)^+ : \mathbf{S} \longrightarrow \mathbf{S}$ also supports the comonad structure

$$A \xleftarrow{\varepsilon} A^+ \xrightarrow{f^\#} B^+$$

$$a_n \longmapsfrom (a_1 a_2 \cdots a_n) \longmapsto \Big(f(a_1)\ f(a_1 a_2)\ \cdots\ f(a_1 \cdots a_n)\Big)$$

Thinking of the sequences $(a_1 a_2 \cdots a_n)$ as sequences of events makes them into *histories*. The cumulative functions $f^\#$ thus capture the functions extended in time. Prop. 2.3.3 says that proofs of the process implications $[A, B]$ correspond to such functions. This correspondence makes process implications into hom-sets of a category.

3.4 Category of functions extended in time

The category of free coalgebras for the comonad $(-)^+$ is

$$\begin{aligned} |\mathbf{S}_+| &= |\mathbf{S}| \\ \mathbf{S}_+(A, B) &= \mathbf{S}(A^+, B) \end{aligned}$$

The lifting # gives rise to the composition in this category:

$$A^+ \xrightarrow{f} B$$

$$A^+ \xrightarrow{f^\#} B^+ \qquad\qquad B^+ \xrightarrow{g} C$$

$$(f; g) = \left(A^+ \xrightarrow{f^\#} B^+ \xrightarrow{g} C\right)$$

The counit $A^+ \xrightarrow{\varepsilon} A$ plays the role of the identity for this composition. The correspondence

$$\mathbf{S}_+(A, B) \cong \mathbf{S}(1, [A, B])$$

means that $\mathbf{S}_+$, in a sense, externalizes the process implications as functions extended in time, and makes their proofs composable. The time extension of their composition unfolds in their cumulative form. Since A^+ is the disjoint union of $\coprod_{n=1}^{\infty} A^n$, a function $f : A^+ \longrightarrow B$ can be viewed as the stream $f^\omega = (f_1\ f_2 \cdots f_n \cdots)$ of functions $f_n : A^n \longrightarrow B$, like in Sec. 3.2. The corresponding cumulative function $f^\# : A^+ \longrightarrow B^+$ can then be viewed as the stream $f^\# = \left(f^1\ f^2 \cdots f^n \cdots\right)$ of functions $f^n : A^n \longrightarrow B^n$ which commute in the following diagram

$$\begin{array}{ccccccccc}
A & \xleftarrow{\overleftarrow{\pi}} & A^2 & \xleftarrow{\overleftarrow{\pi}} & A^3 & \xleftarrow{\overleftarrow{\pi}} & A^4 & \cdots\!\!\leftarrow & A^i & \cdots\!\!\leftarrow \\
\downarrow{\scriptstyle f^1} & & \downarrow{\scriptstyle f^2} & & \downarrow{\scriptstyle f^3} & & \downarrow{\scriptstyle f^4} & & \downarrow{\scriptstyle f^i} & \\
B & \xleftarrow[\overleftarrow{\pi}]{} & B^2 & \xleftarrow[\overleftarrow{\pi}]{} & B^3 & \xleftarrow[\overleftarrow{\pi}]{} & B^4 & \cdots\!\!\leftarrow & B^i & \cdots\!\!\leftarrow
\end{array} \tag{30}$$

Each $\overleftarrow{\pi}$ projects away the rightmost component. The components f^n are:

$$f^1 \;=\; f_1 \qquad\qquad f^{i+1} \;=\; \left\langle f^i \circ \overleftarrow{\pi}, f_{i+1} \right\rangle$$

4 Partial functions extended in time

4.1 Output deletions and process deadlocks

Recall from Sec. 1.3.1(10) that the partiality monad $(-)_\perp : \mathbf{S} \longrightarrow \mathbf{S}$ adjoins a fresh element $\perp$ to every type. A partial function $f : A \rightharpoonup B$ can be viewed as the total function $A \longrightarrow B_\perp$, which sends to $\perp$ the elements where f is undefined. There are two logically different ways to lift this to processes:

$$\begin{array}{cc}
A \wedge X \,|\!\!\rightarrow B_\perp \wedge X & A \wedge X \,|\!\!\rightarrow (B \wedge X)_\perp \\
X \,|\!\!\rightarrow [A, B_\perp] & X \,|\!\!\rightarrow [A, B]_\perp
\end{array} \tag{31}$$

On the left, the process may *delete* some of the outputs, but it always proceeds to the next state, whether if has produced the output or not. On the right, the process may *deadlock* and fail to produce either the output or the next state. The meanings of the two implications $[A, B_\perp]$ and $[A, B]_\perp$ are captured, respectively, by the final coalgebras of the two functors

$$\begin{array}{ll}
D_{AB\perp} : \mathbf{S} \longrightarrow \mathbf{S} & D_{A\perp B} : \mathbf{S} \longrightarrow \mathbf{S} \\
\quad X \mapsto (A \Rightarrow (B_\perp \times X)) & \quad X \mapsto (A \Rightarrow (B \times X)_\perp)
\end{array}$$

The state spaces of the final coalgebras of these two functors are then the hom-sets of the two categories of partial functions extended in time:

$$\begin{array}{ll}
|\mathbf{S}_{+\perp}| = |\mathbf{S}| & |\mathbf{S}_{\perp+}| = |\mathbf{S}| \\
\mathbf{S}_{+\perp}(A, B) = \mathbf{S}(A^+, B_\perp) & \mathbf{S}_{\perp+}(A, B) = \coprod_{S \in \mathcal{Y}A^+} \mathbf{S}(S, B)
\end{array} \tag{32}$$

where $\curlyvee A^+$ is the set of safety specifications in A [3, 19, 95]

$$\curlyvee A^+ \quad = \quad \{S \subseteq A^+ \mid \vec{x} \preccurlyeq \vec{y} \in S \implies \vec{x} \in S\} \tag{33}$$

and where the prefix relation $\vec{x} \preccurlyeq \vec{y}$ means that there is $\vec{z}$ such that $\vec{x}\vec{z} = \vec{y}$. An $\mathbf{S}_{\perp+}$-morphism is a ladder like (34), but with partial functions f_i as rungs. The commutativity requirement imples that $f_i(\vec{s})$ must be defined whenever $f_{i+1}(\vec{s}a)$ is defined for some a. Hence $S \in \curlyvee A^+$ in (32).

4.2 Safety and synchronicity

For $B = 1$, the right-hand part of (32) boils down to $\mathbf{S}_{\perp+}(A, 1) \cong \curlyvee A^+$. The safety properties in $\curlyvee A^+$ can thus be viewed as the objects of categories of *safe* dynamic functions. The morphisms may be *synchronous* or *asynchronous*, depending on whether the outputs are always observable. They become asynchronous if some outputs may be hidden or deleted.

4.2.1 Synchronous safe functions

The category SFun of *safe dynamic functions* has all safety specifications as its objects. Combining the $\mathbf{S}_+$-ladders from (30) with the $\mathbf{S}_\perp/1$-surjections from (15) shows that the safe dynamic functions are ladders in the form

$$\begin{array}{ccccccccccc}
S_1 & \xleftarrow{\overleftarrow{\pi}} & S_2 & \xleftarrow{\overleftarrow{\pi}} & S_3 & \xleftarrow{\overleftarrow{\pi}} & S_4 & \leftarrow\cdots & S_i & \leftarrow\cdots \\
| & \llcorner^w & | & \llcorner^w & | & \llcorner^w & | & & | & \\
f^1 & & f^2 & & f^3 & & f^4 & & f^i & \\
\downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & \\
T_1 & \xleftarrow[\overleftarrow{\pi}]{} & T_2 & \xleftarrow[\overleftarrow{\pi}]{} & T_3 & \xleftarrow[\overleftarrow{\pi}]{} & T_4 & \leftarrow\cdots & T_i & \leftarrow\cdots
\end{array} \tag{34}$$

The functions f^i are not mere surjections, in the sense that for every history $\vec{t} \in T$ there is a history $\vec{s} \in S$ such that $\vec{t} = f^\#(\vec{s})$. They are surjections *extended in time*, in the sense that the prefixes of $\vec{t}$ must have been the image of the prefixes of $\vec{s}$, i.e. $\overleftarrow{\pi}\left(\vec{t}\right) = f^\#\left(\overleftarrow{\pi}\vec{s}\right)$. Categorically, this amounts to saying that the squares in (34) are weak pullbacks. Logically, the commutativity of (34) uncovers a general coinductive pattern:

$$f^\#(\vec{s}) = \vec{t} \iff \forall b \in B \left(\vec{t}b \in T \implies \exists a \in A.\ \vec{s}a \in S \wedge f^\#(\vec{s}a) = \vec{t}b\right) \tag{35}$$

Such coinductive surjections lie at the heart of process theory as components of *bisimulations*, which we shall encounter in the next section. Before that, note that the dynamic surjections satisfying (35) must be *synchronous*, in the sense that they preserve the length of the histories: the time ticks steadily up the ladder. If there are silent actions, i.e. if functions may delete their outputs, this synchronicity may be breached.

4.2.2 Asynchronous safe functions

The functions extended in time *asynchronously* inhabit the category $\mathbf{S}_{\perp+}$. The element $\perp$ added to the outputs plays the role of the *silent*, unobservable action [47, 76]. In synchronous models, the

observer is assumed to have global testing capabilities [1]. The asynchrony arises when some of the actions of the Environment may not be observable for the System. Viewed as channels, the asynchronous functions extended in time become the deterministic *deletion* channels [77]. This leads to coarser process equivalences. Combining both of the constructions (32) allows capturing both forms of the partiality in

$$\begin{aligned} |\mathbf{S}_{\perp+\perp}| &= |\mathbf{S}| \\ \mathbf{S}_{\perp+\perp}(A,B) &= \coprod_{S \in \Downarrow A^+} \mathbf{S}(S, B_\perp) \end{aligned} \tag{36}$$

A function $f \in \mathbf{S}(S, B_\perp)$ can be viewed as a stream of functions $f = (f_n : S_{\leq n} \longrightarrow B_\perp)_{i=1}^{\infty}$, where $S_{\leq n}$ are safe histories of length up to n, including the empty history, i.e.

$$S_{\leq n} \quad = \quad (S \cap A^{\leq n}) + \{()\} \tag{37}$$

Here $A^{\leq n}$ is the disjoint union (coproduct) $\coprod_{i=0}^{n} A^i$. The cumulative form $f^{\#} = (f^{\leq n} : S_{\leq n} \longrightarrow B^{\leq n})_{n=1}^{\infty}$ is now defined by

$$f^{\leq 1}() = () \qquad\qquad f^{\leq 1}(a) = \begin{cases} () & \text{if } f_1(a) = \perp \\ f_1(a) & \text{otherwise} \end{cases}$$

$$f^{\leq n+1}() = () \qquad\qquad f^{\leq n+1}(a\vec{x}) = \begin{cases} f^{\leq n}(\vec{x}) & \text{if } f_{n+1}(a\vec{x}) = \perp \\ f^{\leq n}(\vec{x}) :: f_{n+1}(a\vec{x}) & \text{otherwise} \end{cases}$$

Its components are this time the rungs of the ladder

$$\begin{array}{ccccccccc} S_{\leq 1} & \xleftarrow{\overleftarrow{\pi}} & S_{\leq 2} & \xleftarrow{\overleftarrow{\pi}} & S_{\leq 3} & \xleftarrow{\overleftarrow{\pi}} & S_{\leq 4} & \longleftarrow\!\cdots & S_{\leq i} \longleftarrow\!\cdots \\ \downarrow{\scriptstyle f^{\leq 1}} & & \downarrow{\scriptstyle f^{\leq 2}} & & \downarrow{\scriptstyle f^{\leq 3}} & & \downarrow{\scriptstyle f^{\leq 4}} & & \downarrow{\scriptstyle f^{\leq i}} \\ B^{\leq 1} & \xleftarrow[\overleftarrow{\pi}]{} & B^{\leq 2} & \xleftarrow[\overleftarrow{\pi}]{} & B^{\leq 3} & \xleftarrow[\overleftarrow{\pi}]{} & B^{\leq 4} & \longleftarrow\!\cdots & B^{\leq i} \longleftarrow\!\cdots \end{array} \tag{38}$$

where each $\overleftarrow{\pi}$ again projects away the last component. The category $\mathsf{ASFun} = \mathbf{S}_{\perp+\perp}/1$ of asynchronous safe functions has the safety specifications as its objects again, and a morphism $f \in \mathsf{ASFun}(S_{\pm A}, T_{\pm B})$ is a tower in the form

$$\begin{array}{ccccccccc} S_{\leq 1} & \xleftarrow{\overleftarrow{\pi}} & S_{\leq 2} & \xleftarrow{\overleftarrow{\pi}} & S_{\leq 3} & \xleftarrow{\overleftarrow{\pi}} & S_{\leq 4} & \longleftarrow\!\cdots & S_{\leq i} \longleftarrow\!\cdots \\ \downarrow{\scriptstyle f^{\leq 1}} & \llcorner^{w} & \downarrow{\scriptstyle f^{\leq 2}} & \llcorner^{w} & \downarrow{\scriptstyle f^{\leq 3}} & \llcorner^{w} & \downarrow{\scriptstyle f^{\leq 4}} & \llcorner^{w} & \downarrow{\scriptstyle f^{\leq i}} \\ T_{\leq 1} & \xleftarrow[\overleftarrow{\pi}]{} & T_{\leq 2} & \xleftarrow[\overleftarrow{\pi}]{} & T_{\leq 3} & \xleftarrow[\overleftarrow{\pi}]{} & T_{\leq 4} & \longleftarrow\!\cdots & T_{\leq i} \longleftarrow\!\cdots \end{array} \tag{39}$$

This tower differs from (38) in that the squares are weak pullbacks, and the rungs of the ladder are surjective[5]. It shows that the *asynchronous* surjections exended in time satisfy the following

[5]Formally, in any regular category $\mathbf{S}$, the fact that all rungs are surjective can be derived from the assumption that the starting component is a surjection, and that the squares are weak pullbacks.

condition:

$$f^{\#}(\vec{s}) = \vec{t} \iff \left(\forall b \in B.\ \vec{tb} \in T \Rightarrow \exists \vec{a} \in A^{+}.\ \vec{sa} \in S \wedge f^{\#}(\vec{sa}) = \vec{tb}\right) \tag{40}$$

This condition differs from (35) in that each step up the T-side by $b \in B$ may be followed on the S-side by a string of steps $\vec{a} \in A^{+}$, rather than just a single step $a \in A$.

5 Relations extended in time

5.1 External and internal nondeterminism

We saw in Sec. 1.3.1 that nondeterminism is modeled using the powerset monad $\wp : \mathbf{S} \longrightarrow \mathbf{S}$. Since a subset $U \subseteq A$ corresponds to an element $U \in \wp A$, a binary relation $R \subseteq A \times B$, viewed as a set of subsets $aR \subseteq B$ indexed over $a \in A$ corresponds to the function $\bullet R \colon A \longrightarrow \wp B$. The same relation, viewed as a set of subsets $Rb \subseteq A$, indexed over $b \in B$ also corresponds to the function $R\bullet \colon B \longrightarrow \wp A$. See Appendix A for more details.

There are two ways again in which the side-effect, this time nondeterminism, may affect processes. Internal nondeterminism affects the outputs, whereas external nondeterminism may also affect the states:

$$\begin{array}{ll} A \times X \xrightarrow{\xi} (\wp B \times X)_{\perp} & A \times X \xrightarrow{\zeta} \wp(B \times X) \\ X \xrightarrow{[\![\xi]\!]} [A, \wp B]_{\perp} & X \xrightarrow{[\![\zeta]\!]_{\wp}} [A, B]_{\wp} \end{array} \tag{41}$$

The external nondeterminism on the right incorporates the partiality as the empty outcome $\emptyset \in \wp(B \times X)$. The partiality monad $(-)_{\perp}$ is explicitly added to the internal nondeterminism on the left since they would otherwise never deadlock, which is problematic both conceptually and technically. If a process ξ on the left, e.g. involving some guessing that leads to internal nondeterminism, never deadlocks at a state $x \in X$ and on an input $a \in A$, then it determines a unique next state $\xi^{\circ}(a, x) \in X$, and may produce an output from the set $\xi^{\bullet}(a, x) \in \wp B$. For an externally nondeterministic process ζ on the right, both the outputs and the state transitions are impacted by the nondeterminism, and any pair from $\zeta(a, x) \in \wp(B \times X)$ may be produced when the input a is consumed at state x. The intended meanings of the two process implications $[A, \wp B]_{\perp}$ and $[A, B]_{\wp}$ are captured, respectively, as the final coalgebras of the functors

$$\begin{array}{ll} P_{AB} : \mathbf{S} \longrightarrow \mathbf{S} & Q_{AB} : \mathbf{S} \longrightarrow \mathbf{S} \\ \quad X \mapsto (A \Rightarrow (\wp B \times X))_{\perp} & \quad X \mapsto \wp(A \times B \times X) \end{array} \tag{42}$$

The expression on the right is based on the bijection $\wp(A \times B \times X) \cong (A \Rightarrow \wp(B \times X))$. The state spaces of the final coalgebras of these two functors are quite different. We consider them separately, in the next two sections.

5.2 Internal nondeterminism

5.2.1 Synchronous safe relations

The state space of the final coalgebra of the functor $P_{A \wp B}$ can be constructed within $\mathbf{S}$ as a limit of the tower like (26)

$$\begin{array}{ccccccc} 1 & \xleftarrow{!} & (A \Rightarrow \wp B)_\perp & \xleftarrow{(A \Rightarrow (\wp B \times !))} & (A \Rightarrow (\wp B \times (A \Rightarrow \wp B)_\perp))_\perp & \leftarrow \cdots & \\ & & \cdots & & & & \\ & & P^n_{AB}(1) & \xleftarrow{P^n_{A \wp B}(!)} & P^{n+1}_{AB}(1) & \leftarrow \cdots & [A, \wp B]_\perp \end{array} \tag{43}$$

or presented simply as

$$\begin{aligned} |\mathbf{S}_{+\wp}| &= |\mathbf{S}| \\ \mathbf{S}_{+\wp}(A, B) &= \coprod_{S \in \downarrow A^+} \mathbf{S}(S, \wp B) \end{aligned} \tag{44}$$

A morphism from A to B in $\mathbf{S}_{+\wp}$ is thus a pair $\langle S, R \rangle$, where S is a safety specification, i.e. a prefix-closed set of A-histories from (33), and R is a stream of relations, presented as a stream of functions $\bullet R = \left(S_n \xrightarrow{\bullet R_n} \wp B \right)_{n=1}^{\infty}$, where $S_n = S \cap A^n$, or viewed cumulatively as

$$\bullet R^{\#} = \left(S_n \xrightarrow{\bullet R^n} (\wp B)^n \right)_{n=1}^{\infty}$$

The inductive definition is analogous to the one at the end of Sec. 3. On any input $(a_1\, a_2\, \cdots a_n) \in S$ the n-th component of $\bullet R^{\#}$ thus produces an n-tuple of subsets of B:

$$(a_1\, a_2\, \cdots a_n) R^n = \left\langle a_1 R_1,\ (a_1 a_2) R_2,\ \ldots,\ (a_1 \cdots a_{n-1}) R_{n-1},\ (a_1 \cdots a_{n-1} a_n) R_n \right\rangle \tag{45}$$

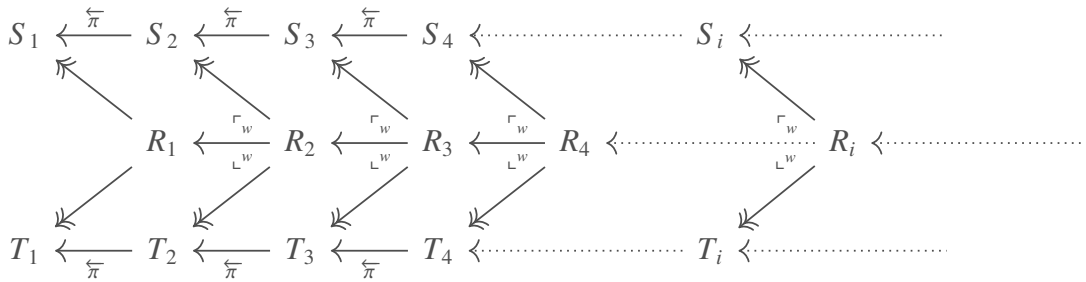
If each each function $S_n \xrightarrow{\bullet R^n} (\wp B)^n$ is viewed as a relation $S_n \xleftrightarrow{R^n} B^n$, then (45) says that they make the following tower commute

$$\begin{array}{ccccccccccc} S_1 & \xleftarrow{\overleftarrow{\pi}} & S_2 & \xleftarrow{\overleftarrow{\pi}} & S_3 & \xleftarrow{\overleftarrow{\pi}} & S_4 & \leftarrow \cdots & S_i & \leftarrow \cdots & \\ & \nwarrow & & \nwarrow & & \nwarrow & & \nwarrow & & \nwarrow & \\ & & R_1 & \xleftarrow{} & R_2 & \xleftarrow{} & R_3 & \xleftarrow{} & R_4 \leftarrow \cdots & R_i & \leftarrow \cdots \\ & \swarrow & & \swarrow & & \swarrow & & \swarrow & & \swarrow & \\ B & \xleftarrow[\overleftarrow{\pi}]{} & B^2 & \xleftarrow[\overleftarrow{\pi}]{} & B^3 & \xleftarrow[\overleftarrow{\pi}]{} & B^4 & \leftarrow \cdots & B^i & \leftarrow \cdots & \end{array} \tag{46}$$

To preclude any nontrivial side-effects of processes with trivial outputs, we slice over the trivial type 1 again, and define the category of *safe synchronous relations extended in time* as

$$\mathsf{SProc} = \mathbf{S}_{+\wp}/1 \tag{47}$$

This is the original *interaction category*, introduced in [3], and further studied in [11, 95]. The descriptions were different, but it is easy to see that the objects coincide, since the morphisms $S \in \mathbf{S}_{+\wp}(A, 1)$ are the prefix-closed sets $S \subseteq A^+$. Reasoning like in Sec. 4.2.1, a morphism $S_{\trianglelefteq A} \overset{R}{\longleftrightarrow} T_{\trianglelefteq B}$ in $\mathbf{S}_{+\wp}/1$ is now reduced to a ladder of spans



(48)

Like in (14), we have relations that are total in both directions, which means that the projections $R \to S$ and $R \to T$ are surjective, in this case componentwise. Like in (34), the surjections are extended in time, in the sense that all rhombi in (48) are weak pullbacks. Putting it all together, this tower says that R satisfies

$$\begin{aligned} \vec{s}\, R\, \vec{t} \quad \Longleftrightarrow \quad & \forall a \in A \ \left(\vec{s}a \in S \Rightarrow \exists b \in B.\, \vec{t}b \in T \ \wedge \ \vec{s}a\, R\, \vec{t}b\right) \ \wedge \\ & \forall b \in B \ \left(\vec{t}b \in T \Rightarrow \exists a \in A.\, \vec{s}a \in S \ \wedge \ \vec{s}a\, R\, \vec{t}b\right) \end{aligned} \tag{49}$$

This condition means that $S_{\trianglelefteq A} \overset{R}{\longleftrightarrow} T_{\trianglelefteq B}$ is a *strong* or *synchronous bisimulation* relation [76, 83], as required in the original definition of SProc in [3].

Bisimulations are intrinsic. The notion of bisimulation was introduced in process theory as an imposed equivalence of the processes that are intended to be semantically indistinguishable [75, 83]. The logical reconstruction of synchronous bisimulation from process-types-as-propositions shows that the same notion also arises as a property of morphisms in a category. The coinductive reconstructions of the whole gamut of bisimulations comprise a well-studied field of research. The present reconstruction, combining the nondeterminism monad $\wp$, the history comonad $(-)^+$, and the slicing over 1, displays the synchronous bisimulations as a logical property of processes arising from nondeterministic choices extended in time, provided that that nontrivial side-effects only arise when there are nontrivial outputs.

5.2.2 Asynchronous safe relations

Including in the model the silent, unobservable actions leads to asynchronicity, and to the notion of *weak* or *observational* bisimulation [47, 76]. Proceeding like in Sec. 4.2.2, we consider the final coalgebras of the functors

$$\begin{aligned} P_{AB\perp} : \mathbf{S} \ &\longrightarrow \ \mathbf{S} \\ X \ &\mapsto \ (A \Rightarrow (\wp(B_\perp) \times X))_\perp \end{aligned} \tag{50}$$

as the hom-sets of the category

$$\begin{aligned} |\mathbf{S}_{+\wp\perp}| &= |\mathbf{S}| \\ \mathbf{S}_{+\wp\perp}(A,B) &= \coprod_{S\in\bigvee A^+} \mathbf{S}(S, \wp(B_\perp)) \end{aligned} \tag{51}$$

The morphism tower is like (46), but with each S_n, R_n and B^n replaced replaced with $S_{\leq n}$, $R_{\leq n}$ and $B^{\leq n}$, as in (37) and (38). The category of *safe asynchronous relations extanded in time* is now

$$\mathsf{ASProc} \;=\; \mathbf{S}_{+\wp\perp}/1$$

and the morphism tower is like (48), with the same modification of the subscripts and the superscripts. This modified tower characterizes the following logical property of the *asynchronous* relation R extended in time:

$$\begin{aligned} \vec{s}\,R\,\vec{t} \iff \forall a\in A\,\Big(\,\vec{s}a\in S \;\Rightarrow\; & \exists b\in B\,(\vec{t}b\in T \;\wedge\; \vec{s}a\,R\,\vec{t}b) \;\vee \\ & \exists \vec{x}\in A^*\,(\vec{s}a\vec{x}\in S \;\wedge\; \vec{s}a\vec{x}\,R\,\vec{t}\,)\Big) \quad \wedge \\ \forall b\in B\,\Big(\,\vec{t}b\in T \;\Rightarrow\; & \exists a\in A\,(\,\vec{s}a\in S \;\wedge\; \vec{s}a\,R\,\vec{t}b) \;\vee \\ & \exists \vec{y}\in B^*\,(\vec{t}b\vec{y}\in T \;\wedge\; \vec{s}\,R\,\vec{t}b\vec{y})\Big) \end{aligned} \tag{52}$$

This characterizes the *weak* or *observationsl* bisimulations of [47, 76]. The category ASProc is equivalent to the one introduced and studied under the same name in [3, 86, 95].

5.3 External nondeterminism

5.3.1 Synchronous dynamic relations

The state space of the final coalgebra of the functor Q_{AB} from (42) should again come with a tower like

$$\begin{array}{ccccccc} 1 & \xleftarrow{\;!\;} & \wp(A\times B) & \xleftarrow{\wp(A\times B\times !)} & \wp(A\times B\times \wp(A\times B) & \leftarrow\cdots & \\ & & \cdots & & & & \\ & & Q^n_{AB}(1) & \xleftarrow{P^n_{\wp AB}(!)} & Q^{n+1}_{AB}(1) & \leftarrow\cdots & [A,B]_\wp \end{array} \tag{53}$$

The trouble is that such a tower never stabilizes within a universe of sets, since there is no set X such that $X\cong \wp X$. If we take $A = B = 1$, the tower boils down to

$$1 \xleftarrow{\;\cup\;} \wp 1 \xleftarrow{\;\cup\;} \wp\wp 1 \leftarrow\cdots \wp^n 1 \xleftarrow{\;\cup\;} \wp^{n+1}(1) \leftarrow\cdots [1,1]_{\mathcal{P}} = \mathfrak{H} \tag{54}$$

where the coinductive fixpoint $\mathfrak{H}$ is the class of *hypersets*, or *non-wellfounded sets* [16]. It is dual to von Neumann's class of well-founded sets [113, 115], which arises as the inductive fixpoint $\mathfrak{V}$ along the tower

$$\emptyset \xhookrightarrow{\;\epsilon\;} \wp\emptyset = 1 \xhookrightarrow{\;\epsilon\;} \wp\wp 1 \xhookrightarrow{\;\epsilon\;} \wp^n 1 \xhookrightarrow{\;\epsilon\;} \wp^{n+1} 1 \xhookrightarrow{\;\epsilon\;} \mathfrak{V} \tag{55}$$

Von Neumann, of course, did not draw categorical diagrams like (55), but specified his construction using the transfinite induction

$$V_0 = \emptyset \qquad V_\beta = \bigcup_{\alpha<\beta} \wp(V_\alpha) \qquad \mathfrak{V} = \bigcup_{\alpha\in\mathsf{Ord}} V_\alpha \tag{56}$$

The class Ord of ordinals, over which the union in the third clause is indexed, is assumed to be given. The construction thus provides an *inner* model of set theory within a given universe of sets and classes [16], or equivalently within a universe with an inaccessible cardinal playing the role of the class Ord [21].[6]. In any case, reach a fixpoint within a given universe, the constructor $\wp$ must be restricted to stay within a smaller universe. Early on, Gödel restricted it to the subsets definable in the language of set theory, and constructed the universe $\mathfrak{L}$ of constructible sets, proving the independence of the Continuum Hypothesis, and launching the whole industry of the independence proofs [43]. Inner models of set-theory have also been constructed over topological spaces, concrete or abstract [49].

The above constructions also restrict to finite sets. The set theorists often explicitly exclude $\aleph_0$ from the definition of inaccessible cardinals, but the inequalities $2^n < \aleph_0$ and $\cup n < \aleph_0$ hold for all for all $n < \aleph_0$, and that makes $\aleph_0$ inaccessible from the universe $\mathbf{fS}$ of finite sets. Formally, $\mathbf{fS}$ can be viewed as the subcategory of $\mathbf{S}$ spanned by $U \in \mathbf{S}$ such that $\#U < \aleph_0$, where $\#U$ denotes the cardinality of U. Since computation is mostly concerned with finite sets, $\mathbf{fS}$ is often by the computer scientists to be the universe of "small sets", and $\mathbf{S}$ is interpreted as the universe of "classes". The powerset construction $\wp : \mathbf{S} \longrightarrow \mathbf{S}$ where $\wp X = \{U \subset X\}$ is then replaced with $\mathcal{P} : \mathbf{S} \longrightarrow \mathbf{S}$ where

$$\mathcal{P}X \;=\; \wp_{<\omega}X \;=\; \{U \subset X \mid \#U < \aleph_0\} \tag{57}$$

which restricts to $\mathcal{P} : \mathbf{fS} \longrightarrow \mathbf{fS}$. The tower (54) with $\mathcal{P}$ replacing $\wp$ thus lies in $\mathbf{fS}$, and reaches a fixpoint $\mathcal{H} \cong \mathcal{P}\mathcal{H}$ in $\mathbf{S}$ after countably many steps. Since $\mathcal{P}$ does not preserve limits, the tower does not stabilize at its limit, but it turns out to stabilize at a retract of its limit [17, 21, 59, 89]. The projections from the fixpoint down the tower are still jointly monic, and support inductive reasoning about the universe of finite hypersets $\mathcal{H} = [1, 1]_{\mathcal{P}}$, which arises in the finite version of (54), and about the finite AB-relations extended in time $[A, B]_{\mathcal{P}}$ which arises in the finite version of (53). Continuing with the workflow from the preceding sections, we use the process implications arising from these finite versions of (53) to define the universe of sets with synchronous dynamic relations:

$$\begin{aligned} |\mathbf{S}^{\mathcal{P}}| &= |\mathbf{S}| \\ \mathbf{S}^{\mathcal{P}}(A, B) &= [A, B]_{\mathcal{P}} \end{aligned} \tag{58}$$

Like before, we factor out any nontrivial side-effects of processes with trivial outputs by slicing over the trivial type 1 again and define the *category synchronous dynamic relations*

$$\mathsf{DProc} \;=\; \mathbf{S}^{\mathcal{P}}/1 \tag{59}$$

[6] A universe of sets and classes is a model of the *NBG* set theory, whereas the one with an inaccessible cardinal can be interpreted in terms of the *ZFC* axioms [72, Ch. 4].

But now something new happens, and a path beyond the workflow from the previous sections opens up. When nondeterminism is internalized, the powerset constructor $\mathcal{P}$ generates types with enough structure to play the role of the labels. More precisely, the states built along the towers (53) to be cumulatively stored and distinguished using the intrinsic structure, making the label sets $A, B \in \mathbf{S}$ dispensable. In the constructions so far, the labels were used to identify the same action when it occurs in different processes. Now action can be identified by its history, which the type constructor, that generates the action, stores in the constructed type.

5.3.2 Internalising the labels

All process universes presented up to so far have been built starting from a given universe $\mathbf{S}$ of labels. The coinductive construction leading to DProc has a novel feature that it can be built starting from nothing: the role of the label sets $A \in \mathbf{S}$ can be played by structures arising from the construction itself. The role of the labels $a \in A$ is to identify the same action when it occurs in different observations, or safety specifications S or T. This is assured by modeling them as subsets $S, T \subseteq A^+$. The upshot is that there can be at most one label-preserving function $S \longrightarrow T$, namely the inclusion $S \hookrightarrow T$.

When all actions arise in a cumulative hierarchy, by iterating the constructor $\mathcal{P}$, be it inductively (55) or coinductively (54), they are always given as sets with the element relation ϵ, which records the elements of each set, their elements, and so on. The axiom of extensionality

$$a = b \quad \Longleftrightarrow \quad (\forall x.\, x \,\epsilon\, a \iff x \,\epsilon\, b) \tag{60}$$

says that this ϵ-structure completely determines the identity of each set: two sets with the same elements are the same set. In the cumulative hierarchy, the elements of sets are sets too, so the same elements are also the sets with the same elements. If such hereditary ϵ-relations are unfolded into trees, the extensionality axiom means that these trees must be *irredundant*: they have no nontrivial automorphisms. In other words, they cannot contain isomorphic subtrees at the same level [84]. The ϵ-structures that arise from the cumulative processes in (55) and (54) are extensional, thus irredundant, because the powerset constructors impose $\{a, a, b, c, \ldots\} = \{a, b, c, \ldots\}$. The other way around, Mostowski's *Collapse Lemma* [80] says that every well-founded extensional relation corresponds to the ϵ-structure of a set somewhere in $\mathfrak{V}$. Aczel's crucial observation in [16] is that the well-foundedness assumption can be dropped: any extensional relation, including non-wellfounded, can be reconstructed as the ϵ-relation of a hyperset, somewhere in $\mathfrak{H}$, or for finite sets somewhere in $\mathcal{H}$. The upshot is that any two hypersets $S, T \in \mathcal{H}$, there is at most one ϵ-preserving function $S \longrightarrow T$, or else nontrivial automorphisms arise. The role of the label sets can now be played by the ϵ-structures.

Lemma 5.1 *For every countable $A \in \mathbf{S}$, i.e. such that $\#A \leq \aleph_0$, there are dynamic relations $A \overset{m}{\to} 1$ and $1 \overset{e}{\to} A$ in $\mathbf{S}^{\mathcal{P}}$ which make A into a retract of 1, i.e. their composite in $\mathbf{S}^{\mathcal{P}}$ is*

$$\mathrm{id}_A \;=\; \left(A \overset{e}{\twoheadrightarrow} 1 \overset{m}{\rightarrowtail} A\right)$$

A **proof** is sketched in Appendix C. To a category theorist, Lemma 5.1 says that the subcategory $\mathbf{S}^{\mathcal{P}}_{\leq\aleph_0} \hookrightarrow \mathbf{S}^{\mathcal{P}}$ spanned by the countable sets is the idempotent completion within $\mathbf{S}^{\mathcal{P}}$ of the endomorphism monoid $\boldsymbol{H} = \mathbf{S}^{\mathcal{P}}(1,1)$. The underlying set of this monoid is the set $\mathcal{H}$ of finite hypersets. The monoid operation is the dynamic synchronous relational composition, spelled out below. For the categories

$$\mathsf{dProc} \;=\; \boldsymbol{H}/1 \qquad\qquad \mathsf{DProc}_{\leq\aleph_0} \;=\; \mathbf{S}^{\mathcal{P}}_{\leq\aleph_0}/1 \tag{61}$$

we have the following corollary, proved in Appendix D.

Corollary 5.2 *The inclusion*

$$\mathsf{dProc} \;\hookrightarrow\; \mathsf{DProc}_{\leq\aleph_0} \tag{62}$$

is an equivalence of categories.

Remark. The equivalence in the preceding corollary means that the embedding is full and faithful, and essentially surjective, i.e. that every type in $\mathsf{DProc}_{\leq\aleph_0}$ is isomorphic to a type in the image of dProc. This notion of equivalence allows finding an adjoint functor in the opposite direction *provided* that the axiom of choice is given, in this case for classes. The equivalence in (62) therefore does not provide an effective global representation of $\mathsf{DProc}_{\leq\aleph_0}$ in dProc. Locally, however, any structure present in DProc can be found in dProc, as long as we do not need uncountable sets of labels. In the rest of the paper, we elide the labels, and work in dProc.

5.3.3 Synchronous dynamic relations as hypersets

The objects of the category dProc boil down the elements of the universe of finite $\mathcal{H}$, that arises as the coinductive fixpoint of the tower like (54), but with $\wp$ restricted to $\mathcal{P} = \wp_{\leq\aleph_0}$. Since $\mathcal{H} \cong \mathcal{P}\mathcal{H}$, an element of $\mathcal{H}$ can also be viewed as its finite subset, which unfolds it into a tower

$$\begin{array}{ccccccccccc}
S_1 & \xleftarrow{\ni} & S_2 & \xleftarrow{\ni} & S_3 & \xleftarrow{\ni} & S_4 & \leftarrow\!\cdots & S_n & \leftarrow\!\cdots & S \\
\downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow \\
\mathcal{P}1 & \xleftarrow{\ni} & \mathcal{P}^2 1 & \xleftarrow{\ni} & \mathcal{P}^3 1 & \xleftarrow{\ni} & \mathcal{P}^4 1 & \leftarrow\!\cdots & \mathcal{P}^n 1 & \leftarrow\!\cdots & \mathcal{H}
\end{array} \tag{63}$$

where all S_n and $\mathcal{P}^n 1$ are in $\mathbf{fS}$. This seems like the most convenient presentation of the objects of dProc. A tower corresponding to a morphism $R \in \mathsf{dProc}(S,T)$ looks just like (48) in Sec. 5.2.1, except that the projections $\overleftarrow{\pi}$ are replaced by the set-theoretic operation $\cup$. The bisimulation condition (49) now becomes

$$s\,R\,t \quad\Longleftrightarrow\quad \forall s' \in s\;\; \exists t' \in t.\;\; s'\,R\,t' \quad\wedge\quad \forall t' \in t\;\; \exists s' \in s.\;\; s'\,R\,t' \tag{64}$$

5.3.4 Asynchronous dynamic relations

So far, the asynchrony has been modeled using a silent action $\perp$, which enabled waiting. When the actions are modeled using the element relation ϵ, i.e. each state transition is a choice of an element, then waiting can be enabled by allowing sets to contain and choose themselves, i.e. by making the relation ϵ reflexive, satisfying $x \epsilon x$ for all x. The objects of the category aProc of *asynchronous dynamic relations* are now the reflexive finite hypersets, conveniently viewed as towers of finite subsets

$$\begin{array}{ccccccccc} S_{\leq 1} & \overset{\ni}{\longleftarrow} & S_{\leq 2} & \overset{\ni}{\longleftarrow} & S_{\leq 3} & \longleftarrow\!\cdots & S_{\leq n} & \longleftarrow\!\cdots & S \\ \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow \\ \mathcal{P}^{\leq 1}1 & \overset{\ni}{\longleftarrow} & \mathcal{P}^{\leq 2}1 & \overset{\ni}{\longleftarrow} & \mathcal{P}^{\leq 3}1 & \longleftarrow\!\cdots & \mathcal{P}^{\leq n}1 & \longleftarrow\!\cdots & \mathcal{H}^{\cup} \end{array} \tag{65}$$

where $\mathcal{P}^{\leq n}X = \coprod_{i=0}^{n} \mathcal{P}^i X$, and $\mathcal{H}^{\cup}$ is the universe of reflexive finite hypersets. A morphism $R \in \mathsf{aProc}(S, T)$ is now a reflexive hyperset relation, satisfying the following property

$$\begin{aligned} s\,R\,t \quad \Longleftrightarrow \quad & \forall s' \epsilon s \ (\exists t' \epsilon t.\ s'\,R\,t' \ \vee \ \exists s'' \epsilon s'.\ s''R\,t) \ \wedge \\ & \forall t' \epsilon t \ (\exists s' \epsilon s.\ s'\,R\,t' \ \vee \ \exists t'' \epsilon t'.\ s\,R\,t'') \end{aligned} \tag{66}$$

The computational origins of this simulation strategies were studied in [110, 111]. Like before, it also arises from the mathematical structure of final coalgebras, and can be logically reconstructed from the paradigm of process-types-as-propositions.

6 Integers, interactions, and real numbers

6.1 The common denominator of integers and interactions

Counting generates the ordinals [113], but the integers arise from the duality of counting up and down. Geometric and algebraic transformations generate monoids, but capturing the symmetries requires groups. Interactions between the system and the environment generate process universes, some of which we studied; but the dual interactions between the environment and the system were not captured. The duality inherent in process interactions was noted, albeit in passing, very early on in process theory:

> "The *whole* meaning of any computing agent [would be that it is] a transducer, whose input sequence consists of enquiries by, or responses from, its environment, and whose output sequence consists of enquiries of, or responses to, its environment" [73, p. 160].

A similar vision of dual interactions between the system and the environment, as an ongoing session of a question-answer protocol, re-emerged in linear logic [41]. It was formalized categorically in [13], and retraced in [4]. The mathematical underpinning turned out to be the Int-construction, generating free compact categories over traced monoidal categories [50]. The name does not refer to ***int****eractions* but to ***int****egers*. Appying the Int-construction to the additive monoid $\mathbb{N}$ of natural numbers, viewed as a discrete monoidal category, gives rise to the additive group $\mathbb{Z}$ of integers,

viewed as a discrete compact category. The the trace structure on the monoid $\mathbb{N}$ is the cancellation property:

$$m + k = n + k \quad \Longrightarrow \quad m = n$$

The set of integers is defined as the quotient

$$\mathbb{Z} \;=\; \mathsf{Int}_{\mathbb{N}} \;=\; \mathbb{N}_- \times \mathbb{N}_+ / \sim$$

where the equivalence relation $\sim$ is:

$$\langle m_-, m_+ \rangle \sim \langle n_-, n_+ \rangle \quad \Longleftrightarrow \quad m_- + n_+ = n_- + m_+$$

The two components of the product are annotated for convenience, e.g. as $\mathbb{N}_- = \{''-''\} \times \mathbb{N}$ and $\mathbb{N}_+ = \{''+''\} \times \mathbb{N}$. The cancellation property guarantees that each $\sim$-equivalence class contains a unique canonical representative in the form $\langle n, 0 \rangle$ or $\langle 0, n \rangle$. The former can be written as $-n$, the latter as $+n$.

The structural common denominator of integers and interactions, which makes the Int-construction applicable to both, is the *trace* operation. It will also take us from relations extended in time to the reals. Towards that goal, we spell out how the trace operation arises in categories of relations. This will makes the Int-construction applicable to the interaction categories.

Since the categories of relations, described in Appendix A, are self-dual, the coproducts + from the universe of sets and functions **S** become biproducts $\oplus$ in the category **R** of sets and relations. As the coproduct lifts to the universes of functions extended in time, the biproducts lift to the universes SProc, ASProc, dProc and aProc of relations extended in time. By definition, the biproducts are both products and coproducts. Since the relation biproducts are induced by the function coproducts, their unit is the function coproduct unit 0. For every type X, the biproduct structure consists of

- a monoid $0 \xrightarrow{!} X \xleftarrow{[\mathrm{id},\mathrm{id}]} X \oplus X$, and
- a comonoid $0 \xleftarrow{!} X \xrightarrow{\langle \mathrm{id},\mathrm{id} \rangle} X \oplus X$,

which are natural with respect to all morphisms in and out of X. The projections $X \xleftarrow{\pi} X \oplus Y \xrightarrow{\pi'} Y$ and the injections $X \xrightarrow{\iota} X \oplus Y \xleftarrow{\iota'} Y$ are derived from the comonoid counits and from the monoid units respectively. A propositions-as-types interpretation of biproducts is tenuous but a process category with the biproducts and the hom-sets $[A, B]$ supporting a coinductive rule

$$A \oplus X \xrightarrow{\xi} B \oplus X$$

$$X \xrightarrow{[\![\xi]\!]} [A, B]$$

comes with the trace structure Tr derived by

$$A \xrightarrow{\iota} A \oplus Y \qquad A \oplus Y \oplus [A \oplus Y, B \oplus Y] \xrightarrow{\upsilon} B \oplus Y \oplus [A \oplus Y, B \oplus Y] \qquad B \oplus Y \xrightarrow{\pi} B$$

$$A \oplus [A \oplus Y, B \oplus Y] \xrightarrow{\iota;\upsilon;\pi} B \oplus [A \oplus Y, B \oplus Y]$$

$$[A \oplus Y, B \oplus Y] \xrightarrow{\mathrm{Tr}=[\![\iota;\upsilon;\pi]\!]} [A, B]$$

Each of the categories of relations, **R**, SProc, dProc, etc., is easily seen to give rise to the trace structure in this way. See Appendix E for more.

6.2 Games as labelled polarized relations extended in time

The biproducts in ASProc are in the form

$$\begin{array}{ccccccc} (S \oplus T)_{\leq 1} & \longleftarrow & (S \oplus T)_{\leq 2} & \longleftarrow & (S \oplus T)_{\leq 3} & \leftarrow\!\cdots & (S \oplus T)_{\leq i} \leftarrow\!\cdots \\ \downarrow & & \downarrow & & \downarrow & & \downarrow \\ (A+B)^{\leq 1} & \xleftarrow{\pi} & (A+B)^{\leq 2} & \xleftarrow{\pi} & (A+B)^{\leq 3} & \leftarrow\!\cdots & (A+B)^{\leq i} \leftarrow\!\cdots \end{array} \tag{67}$$

where $(S \oplus T)_{\trianglelefteq A+B}$ are all shuffles of $S_{\trianglelefteq A}$ and $T_{\trianglelefteq B}$.

$$(S \oplus T)_{\leq i} \quad = \quad \left\{ \vec{x} \in (A+B)^{\leq i} \mid \vec{x} \restriction_A \in S \ \wedge \ \vec{x} \restriction_B \in T \right\}$$

The trace structure of categories of relations with respect to the biproducts as the monoidal structure was analyzed already in the final section of [50], and explained in more detail for the interaction categories in [4]. The analysis presented in that paper suggests that the *AJM-games* [12, 14, 15] should be construed in terms of the Int-construction. The AJM-games are, of course, one of the crowning achievements of the quest for fully abstract models of PCF, and a tool of many other semantical results. They appeared in many different semantical contexts [5, 14, 109], with many refinements and different presentation details. A crude common denominator can be obtained by applying the Int-construction from Appendix E to the category ASProc, leading to

$$\begin{array}{rcl} |\mathsf{Gam}| & = & |\mathsf{ASProc}|_- \times |\mathsf{ASProc}|_+ \\ \mathsf{Gam}(S,T) & = & \mathsf{ASProc}(S_- \oplus T_+, T_- \oplus S_+) \end{array}$$

Some of the crucial features of game semantics, such as the copycat strategy, and the various switching and starting conditions, arise in such reconstructions as abstract mathematical properties, like the notions of bisimulations arose before.

6.3 Polarized dynamics

Since $\mathcal{P}(A+B) \cong \mathcal{P}A \times \mathcal{P}B$, applying the powerset constructor on polarized sets $X_- + X_+$ leads to the functor

$$\begin{array}{rcl} Q \colon \mathbf{S} & \longrightarrow & \mathbf{S} \\ X & \mapsto & \mathcal{P}_- X \times \mathcal{P}_+ X \end{array}$$

where the subscripts are still just annotations, and we can take, e.g., $\mathcal{P}_- X = \{''-''\} \times \mathcal{P}X$ and $\mathcal{P}_+ X = \{''+''\} \times \mathcal{P}X$.

6.3.1 Synchronous case

The universe of *signed* finite hypersets can be constructed just like the universe of hypersets in Sec. 5.3, but bifurcating at each step into positive and negative hypersubsets:

$$\begin{array}{l} 1 \xleftarrow{\ !\ } \mathcal{P}_-1 \times \mathcal{P}_+1 \xleftarrow{Q!} \mathcal{P}_-(\mathcal{P}_-1 \times \mathcal{P}_+1) \times \mathcal{P}_+(\mathcal{P}_-1 \times \mathcal{P}_+1) \leftarrow \cdots \\ \qquad \cdots \quad Q^n1 \xleftarrow{\quad Q^n! \quad} \mathcal{P}_-(Q^n1) \times \mathcal{P}_+(Q^n1) \leftarrow \cdots \quad \mathcal{H}_\pm \end{array} \tag{68}$$

The final coalgebra structure still maps each hyperset to its elements, but this time they can be positive or negative

$$\mathcal{H}_\pm \underset{\langle\, \ni_-,\, \ni_+ \rangle}{\overset{\cup}{\underset{\cong}{\rightleftarrows}}} \mathcal{P}_-\mathcal{H}_\pm \times \mathcal{P}_+\mathcal{H}_\pm \tag{69}$$

Notation. Given $s \in \mathcal{H}_\pm$, we write $s^- = \ni_-(s)$ for the negative part and $s^+ = \ni_+(s)$ for the positive part. We often tacitly identify $\mathcal{H}_\pm$ with $\mathcal{P}_-\mathcal{H}_\pm \times \mathcal{P}_+\mathcal{H}_\pm$, in which case $s \in \mathcal{H}_\pm$ becomes a pair $s = \langle s^-|s^+\rangle$, where $s^- = \ni_-(s)$ and $s^+ = \ni_+(s)$. We follow [34] and denote a generic element of s^- by s_-, and a generic element of s^+ by s_+, and abbreviate $s_- \in \ni_-(s)$ and $s_+ \in \ni_+(s)$ to $s_-, s_+ \in s$. Writing $s = \{s_- \mid s_+\}$ instead of $s = \langle s^-, s^+\rangle$ is yet another well-established notational abuse, used to great effect by John Conway in [34]. E.g., instead of $\cup s = \langle \cup s^-, \cup s^+\rangle$, the polarized union operation, pointing left in (69), can be written in the form

$$\cup s \;\; = \;\; \{s_{--}, s_{-+} \mid s_{+-}, s_{++}\}$$

Other coinductive definitions become even simpler, e,g.

$$\ominus\, s \;=\; \{\ominus s_+ \mid \ominus\, s_-\} \qquad\qquad s \oplus t = \{s_- \oplus t, s \oplus t_- \mid s_+ \oplus t, s \oplus t_+\} \tag{70}$$

Synchronous hypergames. The objects of the category gam are the signed finite hypersets from the universe $\mathcal{H}_\pm$. The final coalgebra structure (69) separates their elements into a negative and a positive part. In the game semantics, this is interpreted as separating a game $s \in \mathcal{H}_\pm$ into a pair $s = \langle s^-, s^+\rangle$, where $s^- = \{s_- \in s\} \in \mathcal{P}_-(\mathcal{H}_\pm)$ are the moves available to the player $-$, whereas $s^+ = \{s_+ \in s\} \in \mathcal{P}_+(\mathcal{H}_\pm)$ are the moves available to the player $+$. The projections $\mathcal{H}_\pm \overset{q_i}{\to} Q^n1$ down the tower (68) represent each game $s \in \mathcal{H}_\pm$ as a stream $[s^1, s^2, s^3, \ldots, s^{n+1}, \ldots]$, where $s^{n+1} = q_{n+1}(s) \in Q^{n+1}1 = \mathcal{P}_-(Q^n1) \times \mathcal{P}_+(Q^n1)$, and thus $s^{n+1} = \langle s^{n+1}_-, s^{n+1}_+\rangle$, where $s^{n+1}_-, s^{n+1}_+ \subseteq Q^n1$.

A morphism $R \in \mathsf{gam}(s,t)$ should be a **synchronous hyperstrategy**. It is a *hyper*strategy because the players $-$ and $+$ play not one, but two games, s and t, distinguished by the dual goals that the two players have in each of them:

$$s\,R\,t \quad\Longleftrightarrow\quad \forall s_- \in s\; \exists t_- \in t.\;\; s_-\,R\,t_- \;\;\wedge\;\; \forall t_+ \in t\; \exists s_+ \in s.\;\; s_+\,R\,t_+ \tag{71}$$

The player − is thus tasked with simulating every s-step by a t-step, whereas the player + is tasked with simulating every t-step by an s-step. A hyperstrategy into a polarized version of a synchronous bisimulation (64). While a bisimulation relation between two processes provides a simulation relation of each of them in the other one, both ways, the polarization of a hyperstrategy separates the two simulation tasks, and each player is tasked with one.

6.3.2 Asynchronous case

Using the functor $Q : \mathbf{S} \longrightarrow \mathbf{S}$ where $QX = X + QX$, the tower in (68) becomes

$$1 \xleftarrow{\ !\ } Q^{\leq 1}(1) \xleftarrow{\ Q!\ } Q^{\leq 2}(1) \leftarrow\cdots\cdots Q^{\leq n}(1) \xleftarrow{\ Q^n!\ } Q^{\leq n+1}(1) \leftarrow\cdots\cdots \mathfrak{R} \tag{72}$$

where $Q^{\leq n}(1) = \coprod_{i=0}^{n} Q^i(1)$. The final coalgebra structure is thus

$$\mathfrak{R} \quad \underset{\ni}{\overset{\cup}{\underset{\cong}{\rightleftarrows}}} \quad \mathfrak{R} \ + \ \mathcal{P}_-\mathfrak{R} \times \mathcal{P}_+\mathfrak{R} \tag{73}$$

The coalgebra structure $\ni$ maps $s = \langle s^-, s^+ \rangle$ to $s = \ni(s)$ if $s^- \in s^-$ and $s^+ \in s^+$. Otherwise it unfolds its elements into $s^- = \ni_-(s)$ and $s^+ = \ni_+(s)$ like before. A straightforward induction along the tower gives the following.

Lemma 6.1 *Every $s \in \mathfrak{R}$ is ϵ-transitive, in the sense that for all $s_-, s_+ \in s$ holds*

$$s_-^- \subseteq s^- \subseteq s_+^- \qquad\qquad s_+^+ \subseteq s^+ \subseteq s_-^+ \tag{74}$$

The elements of the universe $\mathfrak{R}$ of transitive finite signed hypersets can be thought of as **asynchronous hypergames**. They are the objects of the category $\mathcal{R}$. An **asynchronous hyperstrategy** $R \in \mathcal{R}(s,t)$ resembles a branching bisimulation from (66), except that the two simulation tasks are again separated, like in (71), and assigned to the two players:

$$\begin{aligned} s\,R\,t \quad \Longleftrightarrow \quad & \forall s_- \epsilon s \ (\exists t_- \epsilon t.\ \ s_-\,R\,t_- \ \vee \ \exists s_{-+} \epsilon s_-.\ s_{-+} R\,t) \ \wedge \\ & \forall t_+ \epsilon t \ (\exists s_+ \epsilon s.\ \ s_+\,R\,t_+ \ \vee \ \exists t_{+-} \epsilon t_+.\ s\,R\,t_{+-}) \end{aligned} \tag{75}$$

Lemma 6.1 makes the relations induced by the coalgebra structure on $\mathfrak{R}$ into hyperstrategies. Remember that $s_- \epsilon s$ abbreviates $s_- \in \ni(s) \in \mathcal{P}_-\mathfrak{R}$, whereas $s \ni s_+$ abbreviates $s_+ \in \ni(s) \in \mathcal{P}_+\mathfrak{R}$.

Lemma 6.2 (75) *holds when sRt is instantiated to $s_- \epsilon s$ and $s \ni s_+$, for any $s \in \mathfrak{R}$ and $s_-, s_+ \epsilon s$.*

Proof. $s_-^- \subseteq s^-$ implies that for every s_{--} there is s'_- with $s_{--} \epsilon s'_-$. $s^+ \subseteq s_+^+$ implies that for every s_+ there is some s_{-+} with $s_{-+} \epsilon s_+$. Hence (75) for $s_- \epsilon s$. $s^- \subseteq s_+^-$ implies that for every s_- there is s_{+-} with $s - \ni s_{+-}$. $s_+^+ \subseteq s^+$ implies that for every s_{++} there is some s'_+ with $s'_+ \ni s_{++}$. Hence (75) for $s \ni s_+$. □

Remark. The property in (75) is not self-dual under the relational converse, but under the polarity change $\ominus$ in (70). In game semantics, the polarity change switches the roles of the player and the opponent. The game-theoretic concept of *equilibrium*, where both players play their best responses, reimposes the *bi*simulation requirement: that the *same* relation is a winning strategy (simulation) in *both* directions. The equilibrium strategies are thus fixed under two dualities: under the polarity change (switching the players $-$ and $+$), and under the relational converse (switching the component games s and t). The two dualities are generally not independent, as there are situations when they do not commute. However, for games where they do commute, they induce a *dagger-compact* structure, akin to the adjunctions over the complex linear operators, which is induced by two commuting dualities: the complex conjugation and the matrix transposition. This structure also arises in many other areas of abelian and nonabelian geometry. It was not used in the game semantics, but it emerged in the Abramsky-Coecke models of quantum protocols and has been explored in other areas of the semantics of computation [10, 32, 90].

6.4 A category of real numbers

In closing this section, we encounter a remarkable and somewhat disturbing fact: that the posetal collapse of the category $\mathcal{R}$ boils down to the ordered field $\mathbb{R}$ of the real numbers. It is disturbing because it shows that the described process logic and game semantic constructions impose on the processes no computability restrictions whatsoever since they include all real numbers. On one hand, this observation should not be surprising, since John Conway reconstructed numbers from games a long time ago [34], and game semantics was inspired by his ideas and informed by his constructions [12]. On the other hand, it should be surprising, because game semantics has been developed as the semantics of *computational* processes, albeit as a quotient of an undecidable term calculus [24, 67, 74].

6.4.1 Coalgebra of reals

We adapt the alternating dyadics from [97, Sec. 3.2][7] to present the real numbers. Consider the alphabet $\Sigma = \{-, +\}$, and denote by $\Sigma^{\circledast}$ the set of finite and infinite strings over it. It comes with the coalgebra structure

$$\Sigma^{\circledast} \underset{\chi}{\overset{(::)}{\cong}} 1 + \Sigma \times \Sigma^{\circledast} \tag{76}$$

where χ maps the empty string () into 1 and each nonempty strings into its head symbol and the tail string. Equivalently, this coalgebra can be written in the form

$$\Sigma^{\circledast} \underset{\kappa}{\overset{[o,h_-,h_+]}{\cong}} 1 + \Sigma^{\circledast}_- + \Sigma^{\circledast}_+ \tag{77}$$

[7] See also [98, 100] for a broader context.

Where the product $\Sigma\times\Sigma^{\circledast}$, which is $\{-,+\}\times\Sigma^{\circledast}$ is expanded into $\{-\}\times\Sigma^{\circledast}+\{+\}\times\Sigma^{\circledast}$, and the products with the singletons are abbreviated as subscripts. The structure map κ now maps the empty string into 1, and the strings in the form $\pm :: \vec{x}$ as $\vec{x}$ into $\Sigma^{\circledast}_{\pm}$, whereas the components h_- and h_+ add $-$ and $+$ as the head, while o maps the singleton from 1 into the empty string.

Each Σ-string encodes a unique real number. The idea is that we count the first string of $-$s or $+$s in the unary, and after that proceed in the alternating dyadics, e.g.

$$\begin{array}{rcl} +\,+\,-\,-\,-\,+\,-\,- & \mapsto & +1+1-\frac{1}{2}-\frac{1}{4}-\frac{1}{8}+\frac{1}{16}-\frac{1}{32}-\frac{1}{64} \\ -\,-\,-\,-\,+\,-\,+\cdots & \mapsto & -1-1-1-1+\frac{1}{2}-\frac{1}{4}+\frac{1}{8}\cdots \end{array}$$

Since the infinite strings of $-$s and of $+$s encode the two infinities, we will have a map into the extended reals $\mathbb{R}=\mathbb{R}\cup\{\infty,-\infty\}$. The bijection $\Sigma^{\circledast}\cong\mathbb{R}$ is described in Appendix F. We henceforth identify the two, and use both names interchangeably, since $\Sigma^{\circledast}$ refers to the encoding, and $\mathbb{R}$ says what is encoded.

Ordering. The usual ordering of the reals in $\mathbb{R}$ corresponds to the lexicographic ordering of $\Sigma^{\circledast}$. When the finite strings are padded by 0s, the symbol ordering is $-<0<+$.

6.4.2 Numbers extended in time: Conway's version of Dedekind cuts

Theorem 6.3 *There are functors*

$$\mathbb{R} \quad \underset{\Gamma}{\overset{\Upsilon}{\rightleftarrows}} \quad \mathcal{R} \tag{78}$$

which make the extended continuum $\mathbb{R}$ into the posetal collapse of the category $\mathcal{R}$ of asynchronous hypergames. In particular,

- *for every real number $\varsigma\in\mathbb{R}$ holds $\Upsilon\Gamma(\varsigma)=\varsigma$;*
- *for every asynchronous hypergame $s\in\mathcal{R}$ there are natural hyperstrategies*

$$s\xrightarrow{\eta}\Gamma\Upsilon(s) \qquad \textit{and} \qquad \Gamma\Upsilon(s)\xrightarrow{\varepsilon}s$$

Proof (sketch) . The functor Γ can be obtained from the anamorphism $[\![\kappa]\!]$

$$\begin{array}{ccc} \mathbb{R} & \xrightarrow{\ \kappa\ } & \mathbb{R}\ +\ \mathcal{P}_-\mathbb{R}\ \times\ \mathcal{P}_+\mathbb{R} \\ \Big\downarrow{\scriptstyle [\![\kappa]\!]} & & \Big\downarrow{\scriptstyle [\![\kappa]\!]+\mathcal{P}_-[\![\kappa]\!]+\mathcal{P}_+[\![\kappa]\!]} \\ \mathfrak{R} & \xrightarrow[\ \mathfrak{z}\ ]{} & \mathfrak{R}\ +\ \mathcal{P}_-\mathfrak{R}\ \times\ \mathcal{P}_+\mathfrak{R} \end{array}$$

where κ is derived from (77), by mapping the empty string to the empty string, the Σ-strings in the form $(-::\varsigma)$ to the pair $\langle\{\varsigma\},\emptyset\rangle$, and the strings in the form $(+::\varsigma)$ to $\langle\emptyset,\{\varsigma\}\rangle$. Setting $\Gamma\varsigma=[\![\kappa]\!]\,\varsigma$,

the functoriality of Γ boils down to the observation that the lexicographic order $\varsigma \leq \vartheta$ on $\Sigma^{\circledast}$ lifts to a relation $s \leq t$ on $s = \Gamma\varsigma$ and $t = \Gamma\vartheta$ which satisfies (75), i.e.

$$\begin{aligned} s \leq t \quad \Longleftrightarrow \quad & \forall s_- \in s \ (\exists t_- \in t. \ s_- \leq t_- \ \vee \ \exists s_{-+} \in s_-. \ s_{-+} \leq t) \ \wedge \\ & \forall t_+ \in t \ (\exists s_+ \in s. \ s_+ \leq t_+ \ \vee \ \exists t_{+-} \in t_+. \ s \leq t_{+-}) \end{aligned} \tag{79}$$

As long as ς and ϑ are unpadded by 0s, their lexicographic ordering leads to $s = \Gamma\varsigma$ and $t = \Gamma\vartheta$ satisfying the synchronous comparison clauses $s_- \leq t_-$ and $s_+ \leq t_+$ of (79). If ϑ is padded by 0s, then (79) is satisfied because the lexicographic ordering induces $s_{-+} \leq t$. If ς is padded by 0s, then it induces $s \leq t_{+-}$. This completes the definition of Γ.

The functor Υ arises from Conway's *simplicity theorem* [34, Thm. 11]. It picks the simplest representatives of the equivalence classes of the posetal collapse of $\mathcal{R}$, where the simplicity is measured in [34] by the "birthday ordinal", which for our finite hypersets, signed or not, boils down each element's position on its coinduction tower. The simplicity theorem plays a central role in all presentations of surreal numbers, and suitable versions have been proved in detail in [18, 44]. The arrow part of Υ collapses the $\mathcal{R}$-morphisms to the lexicographic order on $\Sigma^{\circledast}$. Conway shortcuts his proof of the simplicity theorem by imposing the posetal collapse directly signed hypersets by

$$s \leq t \quad \Longleftrightarrow \quad \forall s_- \in s \, \forall t_+ \in t. \ \ t \not\leq s_- \ \ \wedge \ \ t_+ \not\leq s \tag{80}$$

Instantiating this definition to $t \leq s_-$ and to $t_+ \leq s$ (80) gives

$$\begin{aligned} t \not\leq s_- \quad & \Longleftrightarrow \quad \exists t_- \in t. \ s_- \leq t_- \quad \vee \ \exists s_{-+} \in s_-. \ s_{-+} \leq t \\ t_+ \not\leq s \quad & \Longleftrightarrow \quad \exists t_{+-} \in t_+. \ s \leq t_{+-} \quad \vee \ \exists s_+ \in s. \ s_+ \leq t_+ \end{aligned}$$

and shows that (80) implies (79). The converse, spelled out along the lines of the proofs of the simplicity theorem that can be found in [18, 44], involves extensive but routinely case reasoning. The equivalence classes of the posetal quotient of $\mathcal{R}$ are thus ordered by (80), which on $\Sigma^{\circledast}$ boils down to the lexicographic order. □

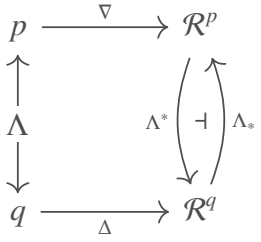
Remarks. Conway's proof of the simplicity theorem demonstrates coinduction in action, not only at the formal level in (80), but also at the meta-level. In order to define the $\mathbb{R}$-ordering of the minimal representatives of the equivalence classes of his games, reduced to numbers, he imposes the sought ordering as a preorder on arbitrary representatives and then uses that preorder to prove the existence of the minimal representatives. Lemma 6.2 also shows how the simplicity follows from the coinductive construction, as it implies $\Upsilon(s_-) \leq \Upsilon(s) \leq \Upsilon(s_+)$, and steers the coinductive descent towards the simplest representative.

6.4.3 Real numbers as processes

Thm. 6.3 says that the real numbers can be viewed as processes; and the other way around, that the asynchronous, polarized, reflexive processes boil down to real numbers. The heart of the theorem is in the "boil down" part of the second statement. Its precise meaning is that the simulations

between the asynchronous, polarized, reflexive processes implement (and are thus consistent with) the real number ordering. If these processes are thought of as the processes of observing, then the reals are the outcomes of the measurements. On the other hand, computing with the reals involves some embedding into a universe where each number is the outcome of many processes. This is a consequence of the observation, going back to Brouwer [26], that the irredundant representations of the reals, where each real number corresponds to a unique stream of digits, there are always basic arithmetical operations, and easily defined inputs, where no finite prefix suffices to determine a finite prefix of the output. Such operations are obviously not computable.

Dropping the infinite strings $-\infty = (---\cdots)$ and $\infty = (+++\cdots)$ on the left-hand side of the retraction $\mathbb{R} \rightleftarrows \mathcal{R}$ in (78), and the signed hypersets bisimilar to $-\infty = \{-\infty|\}$ and $\infty = \{|\infty\}$ on the right-hand side, we get the retraction $\mathbb{R} \rightleftarrows \mathcal{R}$. It lifts to $\mathbb{R}^n \rightleftarrows \mathcal{R}^n$, i.e. it makes the real vector spaces into retracts of the discrete functor categories. A real matrix $L \in \mathbb{R}^{p\times q}$ becomes an $\mathcal{R}$-profunctor $\Lambda = \left(p \overset{\Gamma L}{\longleftrightarrow} q\right)$, and the linear operators $\mathbb{R}^p \overset{L}{\to} \mathbb{R}^q$ and $\mathbb{R}^q \overset{L^\ddagger}{\to} \mathbb{R}^p$ become the $\mathcal{R}$-extensions of $\Lambda = \Gamma L$ along the Yoneda embeddings, in the enriched-category sense.[8]



The left Kan extension Λ^* maps the functor $\alpha \in \mathcal{R}^p$ into the coend, which is the colimit along α of its tensors with the left transpose of Λ. The right Kan extension Λ_* maps the functor $\beta \in \mathcal{R}^q$ into the end, which is the limit along β of its cotensors with the right transpose of Λ. But since α and β are discrete, the colimits boil down to coproducts, and the limits boil down to products. And since $\mathcal{R}$ is self-dual, the products and the coproducts coincide as the biproducts, which we write $\oplus$; and the tensors and the cotensors also coincide as $\otimes$. The Kan extensions thus become

$$\Lambda^*(\alpha) = \left(\bigoplus_{i=1}^{p} \alpha_i \otimes \Lambda_{ij}\right)_{j=1}^{q} \qquad \Lambda_*(\beta) = \left(\bigoplus_{j=1}^{q} \Lambda_{ij} \otimes \beta_j\right)_{i=1}^{p} \tag{81}$$

where

$$\begin{aligned} s \oplus t &= \left\{ s_- \oplus t,\ s \oplus t_- \;\middle|\; s_+ \oplus t,\ s \oplus t_+ \right\} \\ s \otimes t &= \left\{ (s_- \otimes t) \oplus (s \otimes t_+) \ominus (s_- \otimes t_+),\ (s_+ \otimes t) \oplus (s \otimes t_-) \ominus (s_+ \otimes t_-) \;\middle|\right. \\ &\qquad \left. (s_- \otimes t) \oplus (s \otimes t_-) \ominus (s_- \otimes t_-),\ (s_+ \otimes t) \oplus (s \otimes t_+) \ominus (s_+ \otimes t_+) \right\} \end{aligned}$$

correspond respectively to Conway's addition and multiplication operations [34]. Formally, this correspondence means that the retraction Υ satisfies

$$\Upsilon(s \oplus t) = \Upsilon s + \Upsilon t \qquad \Upsilon(s \otimes t) = \Upsilon s \cdot \Upsilon t$$

[8]The reader unfamiliar with what any of this means is welcome to skip the next paragraph paragraph.

The usual matrix operations are thus "rediscovered" as the Υ-image of the Kan extensions in (81) of $\mathcal{R}$-profunctors (corresponding to the real matrices) along the Yoneda embeddings of the bases into their $\mathcal{R}$-completions (corresponding to the real vector spaces).

6.5 Where is computation?

As exciting as it is to see the real numbers arising from the categorical structure of processes, it also suggests that we lost the computation from sight somewhere along the way, while retracing the paths of the categorical semantics of computation. The process universe $\mathcal{R}$ contains a representative $\Gamma\varsigma$ of every real number ς from the field $\mathbb{R}$. Whatever can be computed on such process representatives of the reals in $\mathcal{R}$ can be projected back into $\mathbb{R}$ along Υ. Any real number can ς can be obtained in that way, since $\Upsilon\Gamma\varsigma = \varsigma$. But most real numbers are not computable. Arbitrarily long prefixes of uncomputable reals can be defined by enumerating all computations and avoiding all computable reals by a diagonal argument. This idea has been refined in many directions, showing that almost all real numbers are uncomputable, whichever way we quantify them [29, 40, 69]. And they all live in $\mathcal{R}$. Everything that any oracle can tell any computer is already there. Somewhere on the path from propositions-as-types, through process-propositions-as-types-extended-in-time, to dynamic interactions, the idea of process-computability-as-programmability got lost, and we got all processes.

In the final section, we retrace the path back to one of the original questions of categorical semantics: *How can **intensional** computation be characterized semantically?*

7 Categorical semantics as a programming language

7.1 Computability-as-programmability

A process is computable if it is programmable.[9] In a universe of processes, types are used to specify requirements and to impose constraints. In a universe of *computable* processes, there is also a type $\mathbb{P}$ of *programs*. Since any Turing-complete language can encode its own interpreter, any model of a Turing-complete language must contain[10] the type $\mathbb{P}$ of programs in that language.

A model of computable processes is *extensional* if it only describes the extensions of computations, i.e. their input-output functions, and does not say anything about the process of computation. Each computable function is thus assigned a unique "program". Type-theoretically, this unique "program" is captured by the (cartesian) abstraction operation, which fold a function $f_x(a) : A \times X \longrightarrow B$ with parameters from X to the X-indexed family of abstract functions $\lambda a.f_x(a) : X \longrightarrow (A \Rightarrow B)$. The application operation applies an abstraction to its inputs and recovers the corresponding function. The bijection between the abstractions and their applications was displayed in (4) and formalized in Def. 1.2 using the structure of *cartesian closed* categories.

[9] Network processes are sometimes also called computations, although they are not globally controllable, and thus not programmable. They can be steered by interacting programs and protocols, but that is a different story. The notion of computability was originally defined as computability by computers, and the term is still used in that sense.

[10] The tacit assumption is that a model of a programming language contains all types recognizable in that language.

If "programs" do not specify some input-output mappings, but also how they change during computation, then the X-indexing becomes a state dependency, and the computations are presented as state machines $\xi : A\times X \longrightarrow B\times X$, producing the outputs by $\xi^\bullet : A\times X \longrightarrow B$ and updating the states by $\xi^\circ : A \times X \longrightarrow X$. The bijection between the parametrized functions and their abstractions (4) becomes a mapping (16) of machines $\xi : A \times X \longrightarrow B \times X$ to the anamorphisms $[\![\xi]\!] : X \longrightarrow [A, B]$ assigning to each state in X a dynamic function as the induced computational behavior. This *machine abstraction* was formalized in Def. 2.1 using the structure of *process closed* categories. The machine abstraction is not injective because many different machines realize the same behaviors; and it is not surjective because some dynamic functions are not implementable by machines. A categorical structure capturing how actual computable functions are specified by actual programs (without the quotation marks) is formalized in Def. 7.1. It characterizes computable functions using the language of Definitions 1.2 and 2.1, but not in terms of an abstraction operation, since program abstraction is not an operation.

The conceptual distinction between the static view of the function abstraction in (4), and the dynamic view of the process abstraction in (16) is echoed to some extent by the technical distinction between the *denotational* and the *operational* semantics of computation [5, 9, 27]. Overarching all such distinction is the logical distinction between the *extensional* and the *intensional* models of meaning, going back to Frege, Carnap, Church and Martin-Löf [37]. All models of computation that capture abstraction *as an operation* fall squarely on the extensional side. The intuitive reason is that abstraction as an operation readily produces a "program" to each computation; but programming is not such an easy operation. It is a process that involves programmers and evolves other processes.

In contrast with the denotational models of the λ-abstraction of functions, and with the operational models of the $[\![-]\!]$-abstraction of processes, the intensional models of computations are based on the operations for evaluating programs and executing computations. There are many programs for each computation, but there is no operation that transforms computations into programs.

7.2 Categorical semantics of intensional computation

The logical schema of intensional computation is dual to (16):

$$\begin{array}{ccc} X \overset{p}{\vdash\!\!\to} \mathbb{P} & & \mathbf{S}\,(X, \mathbb{P}) \\ & \dashv\!\vdash\!\diamond & \Big| \\ A \wedge X \overset{\{\!|p|\!\}}{\longmapsto} \diamond(B \wedge X) & & \{\!|-|\!\} \\ & & \Big\downarrow \\ & & \mathbf{S}_M(A \times X, B \times X) \end{array} \tag{82}$$

The idea of computability-as-programmability is expressed by the requirement that the maps $\{\!|-|\!\}$ are surjective: for any computation $A\times X \overset{g}{\to} M(B\times X)$ there is a program ρ such that $\{\!|\rho|\!\} = g$. Computations are presented as state machines to help capturing the dynamics of computation. Prop. 7.2 shows that this view of computation is equivalent to the standard view in terms of acceptable enumerations.

The naturality of the program executions $\{\!|-|\!\}$ in (82) can be described, *mutatis mutandis*, in a similar way like the naturality of $[\![-]\!]$ in (16). An X-indexed family of functions $\{\!|-|\!\}^{AB}_X : \mathbf{S}(X, \mathbb{P}) \longrightarrow \mathbf{S}_M(A \times X, B \times X)$ constitutes a natural transformation $\{\!|-|\!\}^{AB} : \nabla_{\mathbb{P}} \longrightarrow \Theta_{AB}$ between the functors

$$\begin{array}{ll} \nabla_{\mathbb{P}} : \mathbf{S}^o \longrightarrow \mathbf{R} & \Theta_{AB} : \mathbf{S}^o \longrightarrow \mathbf{R} \\ \quad X \mapsto \mathbf{S}(X, \mathbb{P}) & \quad X \mapsto \mathbf{S}_M(A \times X, B \times X) \end{array} \tag{83}$$

See (19) and (21) in Sec. 2.2.2 for the arrow parts of these functors. The naturality requirement is dual to (17). It implies that the diagram here on the left commutes for every $p \in \mathbf{S}(X, \mathbb{P})$.

$$\begin{array}{ccc} \mathbf{S}\,(\mathbb{P}, \mathbb{P}) & \xrightarrow{\quad (-\circ p) \quad} & \mathbf{S}\,(X, \mathbb{P}) \\ \Big\downarrow{\scriptstyle \{\!|-|\!\}^{AB}_{\mathbb{P}}} & & \Big\downarrow{\scriptstyle \{\!|-|\!\}^{AB}_{X}} \\ \mathbf{S}_M\,(A \times \mathbb{P}, B \times \mathbb{P}) & \xleftarrow{\quad (p) \quad} & \mathbf{S}_M\,(A \times X, B \times X) \end{array} \qquad \begin{array}{ccc} A \times X & \xrightarrow{\{\!|\rho|\!\}} & M(B \times X) \\ \Big\downarrow{\scriptstyle A \times \rho} & & \Big\downarrow{\scriptstyle M(B \times \rho)} \\ A \times \mathbb{P} & \xrightarrow[\{\!|\mathrm{id}|\!\}]{} & M(B \times \mathbb{P}) \end{array} \tag{84}$$

The diagram on the right arises by chasing $\mathrm{id} \in \mathbf{S}(\mathbb{P}, \mathbb{P})$ around the diagram on the left. The left-hand diagram says that $\{\!|\mathrm{id}|\!\}^{AB}_{\mathbb{P}}$ and $\{\!|p|\!\}^{AB}_X$ are related under $\Theta_{AB}p$, which by the definition in (19) means that the right-hand square commutes. Since the naturality implies that

$$\{\!|pf|\!\}_Y \;=\; \{\!|p|\!\}_X\,(A \times f) \;=\; \{\!|\mathrm{id}|\!\}_{\mathbb{P}}\,(A \times pf)$$

holds for all $f \in \mathbf{S}(Y, X)$ and $p \in \mathbf{S}(X, \mathbb{P})$, droping the subcripts X from $\{\!|-|\!\}_X$ seldom causes confusion. The other way around, by the surjectivity of $\{\!|-|\!\}$, for every computation $g \in \mathbf{S}_M(A \times X, B \times X)$ there is an X-indexed program $\rho \in \mathbf{S}(X, \mathbb{P})$ such that $\{\!|\rho|\!\}^{AB}_X = g$, making the right-hand square in (84) commute. Since this is true for all A and B, the claim is thus that $\mathbb{P}$ is the state space of a weakly[11] final AB-machine $\{\!|\mathrm{id}|\!\}^{AB}_{\mathbb{P}} \in \mathbf{S}_M(A \times \mathbb{P}, B \times \mathbb{P})$ — for *all types* A and B in $\mathbf{S}$. The categories of computable-as-programmable functions, induced by (84), are thus process-closed in a suitable intensional sense that is both weaker and stronger than the extensional process-closed structure (16). It is weaker in the sense that the abstractions are not unique, but it is stronger in the sense that all abstractions, over all types, are of the same type $\mathbb{P}$. They are the programs. Hence the intensional cousin of the cartesian-closed and the process-closed categories defined in 1.2 and 2.1:

Definition 7.1 *A* categorical computer *is a cartesian category* $\mathbf{S}$ *with a commutative monad* $M : \mathbf{S} \longrightarrow \mathbf{S}$*, a fixed type* $\mathbb{P}$ *of* programs*, and for any pair of types* A, B *an* X*-natural family of surjections, called* program executions*:*

$$\mathbf{S}(X, \mathbb{P}) \quad \xrightarrow{\{\!|-|\!\}^{AB}_X} \!\!\!\!\rightarrow \quad \mathbf{S}_M(A \times X, B \times X) \tag{85}$$

The naturality of the program executions $\{\!|-|\!\}^{AB}$ *is with respect to the functors* $\nabla_{\mathbb{P}}, \Theta_{AB} : \mathbf{S}^o \longrightarrow \mathbf{R}$ *from* (83).

[11]The word "weakly" refers to the fact that the programs ρ_g are not unique: each machine g can be represented by many of them; in fact infinitely many.

Proposition 7.2 *Let* $\mathbf{S}$ *be a cartesian category,* $\mathbb{P} \in \mathbf{S}$ *a fixed type, and* $M : \mathbf{S} \longrightarrow \mathbf{S}$ *a commutative monad. Specifying the the program executions* $\{\!|-|\!\}$ *in* (85)*, and establishing* $\mathbf{S}$ *as a categorical computer, is equivalent to specifying the following data for all types* A, B, X*:*

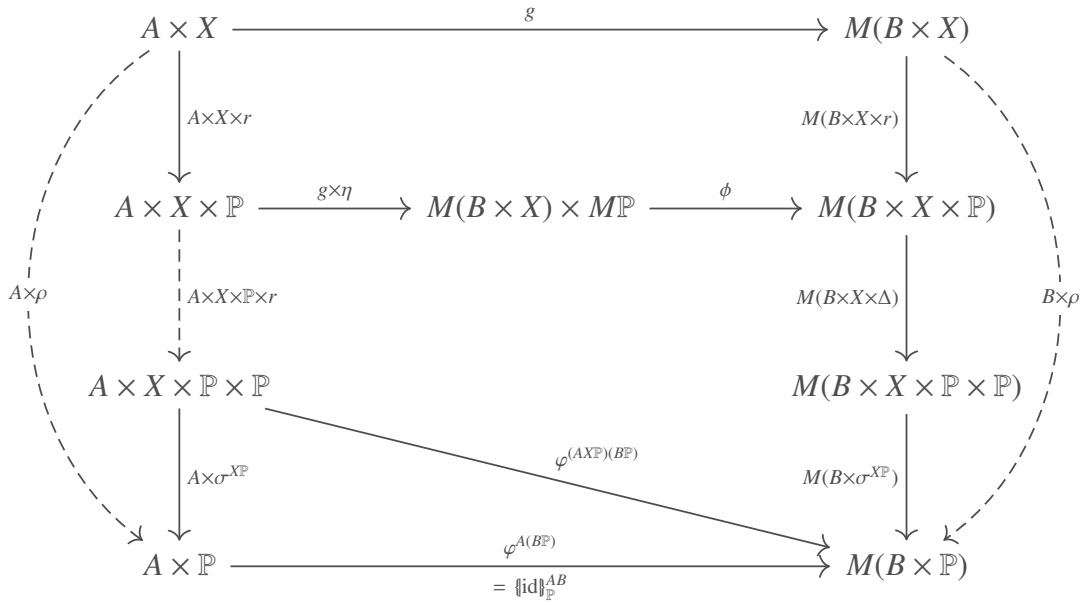
a) a universal evaluator *(or* interpreter*)* $\varphi^{AB} \in \mathbf{S}_M(A \times \mathbb{P}, B)$ *and*

b) a partial evaluator *(or* specializer*)* $\sigma^X \in \mathbf{S}(X \times \mathbb{P}, \mathbb{P})$

such that for any $f \in \mathbf{S}_M(A, B)$ *there is* $p \in \mathbf{S}(1, \mathbb{P})$ *with*

$$\begin{aligned} f &= \varphi^{AB} \circ \big(A \times p\big) \\ \varphi^{(AX)B} &= \varphi^{AB} \circ \big(A \times \sigma^X\big) \end{aligned} \qquad \begin{array}{ccc} A & \xrightarrow{\quad f \quad} & B \\ {\scriptstyle A\times p}\downarrow & \nearrow_{\varphi^{AB}} & \uparrow{\scriptstyle \varphi^{(AX)B}} \\ A \times \mathbb{P} & \xleftarrow{A\times\sigma^X} & A \times X \times \mathbb{P} \end{array} \tag{86}$$

Proof (sketch). Given a categorical computer, the interpreters are $\varphi^{AB} = \pi_B \circ \{\!|\mathrm{id}|\!\}^{AB}_{\mathbb{P}}$ and the specializers σ^X are chosen using the surjectivity of $\{\!|-|\!\}^{AB}_{(X\mathbb{P})}$. Showing that the same σ^X can be chosen for all A and B is the only part which requires work[12]. Towards he converse, setting $\{\!|p|\!\}^{AB}_X = \varphi^{A(BX)} \circ (A \times p)$ defines a natural transformation. To show that its components are surjective, for an arbitrary computation $A \times X \xrightarrow{g} M(B \times X)$, set $\rho(x) = \sigma^{X\mathbb{P}}(x, r, r)$ using in the following diagram.



(87)

The program r is defined by the commutative trapezoid in the middle. It encodes the computation where the state output $A \times X \xrightarrow{g^\circ} MX$ is fed into the function $X \times \mathbb{P} \xrightarrow{X\times\Delta} X \times \mathbb{P} \times \mathbb{P} \xrightarrow{\sigma^{XP}} \mathbb{P}$ where $\sigma^{X\mathbb{P}}$ partially evaluates any program on itself. This computation is the composite of the arrows going from $A \times X \times \mathbb{P}$ right along the top and down along the right side of the trapezoid. Some programs

[12]Most computability theory goes through with non-uniform specializers, which may vary with the context A, B.

r that make the trapezoid commute when substituted as the left dashed side exist by Prop. 7.2(a). The top rectangle is obtained by feeding some such r as the input to to the partial evaluator, to evaluate it on itself. The triangle at the bottom commutes by Prop. 7.2(b). The commutativity of the whole diagram gives $\{\!|\rho|\!\}_X^{AB} = g$. □

Historic background. Prop. 7.2 says that the structure of categorical computer is a categorical version of the standard concept of *acceptable enumeration* [103]. In the standard notation, the enumeration would be a sequence $(\varphi_x^n)_{x\in\mathbb{P}}^{n\in\mathbb{N}}$, where x is the program index, and n is the arity of the computable function φ_x. While the computable functions are usually modeled over natural numbers, and the arity n means that the function takes the inputs of type $\mathbb{N}^n$, and always produces a single output of type $\mathbb{N}$, the categorical treatment is over abstract types, so we write φ^{AB} to specify the input type A and the output type B.

Programming background. The construction in the proof of Prop. 7.2 is easily seen to be a version of Kleene's construction of the fixpoint in his Second Recursion Theorem [103, Ch. 11]. The partial evaluator evaluating all programs on themselves plays the central role. This capability of self-evaluation lies at the heart of many computational constructions [79]. While the diagram chase above elides many equations, the string diagrammatic versions do not just abridge the constructions but display the geometric patterns behind many of them. They support a diagrammatic programming language with convenient implementations of computable logic and arithmetic, program schemas, abstract metaprogramming concepts like compilation, supercompilation, synthesis, and to derive static, dynamic, and algorithmic complexity measures [94, 99].

The λ-calculus and the underlying type theories have been used as abstract programming languages in the semantics of computation from the outset [105], and remained at the heart of the semantical investigations [14, 48]. Programming in abstract programming languages has also been pursued since early on [101]. It led to functional programming, which now permeates programming practices beyond the realm of. However, the mere presence of the abstraction operations makes the underlying type systems essentially extensional. Dropping the extensional λ-conversions allows that multiple programs may correspond to a single computation, but still provides a canonical choice among them, maintains a canonical extensional core of the type system [46]. This has been the main obstacle to studying genuinely intensional algorithmic phenomena, such as complexity, within the semantics of computation.

7.3 Computability as an intrinsic property

A poset may be a monoid in many different ways: e.g., the reals are a monoid for addition, for multiplication, and for many other operations. But a poset may be a lattice (an idempotent monoid) in at most one way: the joins are the least upper bounds, the meets are the greatest lower bounds, and if they exist, they are uniquely determined by the order. A category can be monoidal in many different ways, but it can be cartesian in at most one way because the cartesian products are uniquely determined. The lattice structure of a poset and the cartesian structure of a category are unique, and they are therefore the *properties* of their carriers. When the meets in a poset have the

right adjoints, the implications that arise are also unique, and the structure of a Heyting algebra in is also a property. For the same reason, the cartesian-closed structure from Def. 1.2 is a property of a category, as is the process-closed structure from Def. 2.1.

The structure of a categorical computer from Def. Def. 7.1 is also essentially unique and thus a property of the category that carries it. Proving this requires a little more work than the simple arguments above, but not much more. It boils down to a categorical encoding of the theorem that all parametrized interpreters isomorphically interpret one another [103]. A categorical computer thus displays computability as a categorical property: that all of its morphisms are programmable functions.

On the other hand, it was explained in [9, Sec. 1.2.3] that the notion of computability, *as defined in the standard Church-Turing approach*, is *extrinsic*, in the sense that a particular computable function is recognized as such only by referring to a particular external model of computation, say a Turing machine or a definitional schema. The invoked model then describes a particular process of computing the function, which is not recorded or recognizable on the function itself. It was thus argued in [9] that the standard definitions do not specify computability as an intrinsic structure, even less a property of a function. In contrast, (82) expresses the idea of *computability-as-programmability* as a logical structure; and by the virtue of uniqueness of that structure, as a logical property. Whatever programming language $\mathbb{P}$ might be used to encode programs, they are always assigned semantics along some program executions $\mathbf{S}(X, \mathbb{P}) \twoheadrightarrow \mathbf{S}_M(A \times X, B \times X)$, or along some equivalent mappings. The Rogers' isomorphism theorem says that all programming languages are isomorphic along semantics-preserving computable functions. Whichever Church-Turing model of computation might be used to define computability, the underlying execution model will map its process descriptions to the corresponding computational processes, and this mapping will make it into a categorical computer. This structure provides a *"canonical form witnessing computability"*, sought in [9, Sec. 1.2.3].

Many languages of logic claim universality and establish their universality on their own terms. The set theory proves that it is the foundation of all mathematics, first-order logic is the language of predicates, category theory is the language of structures. The statement that logic is tasked with discovering the universal laws of logic is a tautology, in a logic of logic. A universal law should not be misunderstood as the last word about anything, but as the first word about something else. The idea that *computability-as-programmability* is a model-invariant, syntax-independent, device-free concept, and a property intrinsic to all computable objects and processes, is broader than any particular structure, categorical or otherwise, in which it may be expressed. The idea of computability-as-programmability lurks behind Kolmogorov's invariance theorem [66, Sec. 2.1]. While recognizing a particular function as computable depends on encodings in a particular model, the invariance theorem is built upon the fact that the encodings and their transformations are programmable, and that the programs are of constant lengths. Kolmogorov's invariance theorem can be construed as a quantitative counterpart of Rogers' isomorphism theorem [28, Thm. 2.4.14]. Both theorems characterize computability as an intrinsic property. Computability-as-programmability is not just testable by any of the equivalent models of computation, as claimed by the Church-Turing thesis, but it is also *quantifiable*, in Kolmogorov's formulation by the length of programs. Kolmogorov's *algorithmic complexity* is thus the quantitative view of the intrinsic property of computability-as-

programmability. By displaying programmability as a structure, categorical semantics provides the qualitative view of this property.

It should be noted that the qualitative and the quantitative views of computability as an intrinsic property of processes come about in disguise in many arenas of science. Although the search for a program that makes a process computable is generally not a computable process, its average algorithmic complexity is an intrinsic quantity again: the Shannon entropy [81, 116]. Information theory as the theory of information processing has been viewed as a theory of computation in microsystems, averaged out in thermodynamics. Domain theory has been viewed as a theory of computability-as-approximation in suitable topologies [2, 107, Sec. 5.1]. A natural task for categorical semantics is to bring such conceptual threads together. That is the message that I got from Samson Abramsky's categories that no one had seen before.

8 Summary

In the propositions-as-types view, the extensional operations of abstraction and application, *viz* the structure of cartesian closed categories, correspond to the introduction and the elimination of the propositional implication:

$$\frac{(A \wedge X) \vdash B}{X \vdash (A \supset B)} \supset \qquad \begin{array}{c} \mathbf{S}(A \times X, B) \\ {\scriptstyle (A \Rightarrow -) \circ \eta_X} \big(\quad \big) {\scriptstyle \varepsilon_X \circ (A \times -)} \\ \mathbf{S}(X, (A \Rightarrow B)\,) \end{array}$$

In process logics, the process implication introduction rule corresponds to the coinductive interpretation of arbitrary states as process behaviors, captured in the final machine:

$$\begin{array}{l} A \wedge X \overset{\varphi}{\vdash\!\!\rightarrow} \Diamond(B \wedge X) \\ \qquad\qquad {\scriptstyle [\![-]\!]} \\ X \overset{[\![\varphi]\!]}{\longmapsto} [A, B]_\Diamond \end{array} \qquad \begin{array}{c} \mathbf{S}_M(A \times X, B \times X) \\ \big| \\ {\scriptstyle [\![-]\!]_X} \\ \downarrow \\ \mathbf{S}\,(X, [A, B]_M) \end{array}$$

In terms of dynamic types, computation corresponds to program execution. In terms of process propositions, computability-as-programmability is thus an elimination rule, mapping programs, as intensional proofs of the universal proposition, the programming language, into computations as their extensions:

$$\begin{array}{c} X \overset{p}{\vdash\!\!\rightarrow} \mathbb{P} \\ {\scriptstyle \{-\}} \\ A \wedge X \overset{\{p\}}{\longmapsto} \Diamond(B \wedge X) \end{array} \qquad \begin{array}{c} \mathbf{S}\,(X, \mathbb{P}) \\ \big| \\ {\scriptstyle \{-\}} \\ \twoheaddownarrow \\ \mathbf{S}_M(A \times X, B \times X) \end{array}$$

Categorical semantics provides convenient and sometimes effective tools for reasoning about types and processes. Samson Abramsky led many of us through its vast landscape. I followed him to the best of my ability. The present paper is an attempt at a travel report. But the territory is largely uncharted, and there were times when I lost sight of Samson, probably somewhere far ahead. It is thus likely that the travel report is not just about what I learned from Samson, but also about what I misunderstood by getting lost, and maybe most of all about what I did not learn at all. Categorical semantics of computational processes is a computational process itself, and it is the nature of such processes that they may terminate, or not.

References

[1] Samson Abramsky. Observation equivalence as a testing equivalence. *Theor. Comput. Sci.*, 53(2-3):225–241, 1987.

[2] Samson Abramsky. Domain theory in logical form. *Ann. Pure Appl. Log.*, 51(1-2):1–77, 1991.

[3] Samson Abramsky. Interaction categories. In Geoffrey L. Burn, Simon J. Gay, and Mark Ryan, editors, *Theory and Formal Methods 1993*, Workshops in Computing, pages 57–69. Springer, 1993.

[4] Samson Abramsky. Retracing some paths in process algebra. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR*, volume 1119 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 1996.

[5] Samson Abramsky. Semantics of interaction: an introduction to game semantics. In *Semantics and logics of computation*, volume 14 of *Publications of the Newton Institute*, pages 1–31. Cambridge University Press, 1997.

[6] Samson Abramsky. Abstract scalars, loops, and free traced and strongly compact closed categories. In J.L. Luiz Fiadeiro *et al.*, editor, *Proceedings of the First International Conference on Algebra and Coalgebra in Computer Science (CALCO)*, volume 3629 of *Lecture Notes in Computer Science*, pages 1–29. Springer, 2005.

[7] Samson Abramsky. Coalgebras, chu spaces, and representations of physical systems. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 411–420. IEEE Computer Society, 2010.

[8] Samson Abramsky. Big toy models: Representing physical systems as Chu spaces. *Synthese*, 186(3):697–718, 2012.

[9] Samson Abramsky. Intensionality, definability and computation. In Alexandru Baltag and Sonja Smets, editors, *Johan van Benthem on Logic and Information Dynamics*, pages 121–142. Springer, 2014.

[10] Samson Abramsky and Bob Coecke. A categorical semantics of quantum protocols. In *Proceedings of 19th IEEE Symposium on Logic in Computer Science (LICS 2004)*, 2004.

[11] Samson Abramsky, Simon J. Gay, and Rajagopal Nagarajan. Specification structures and propositions-as-types for concurrency. In Faron Moller and Graham M. Birtwistle, editors, *Logics for Concurrency (Proceedings of 8th Banff Higher Order Workshop)*, volume 1043 of *Lecture Notes in Computer Science*, pages 5–40. Springer, 1995.

[12] Samson Abramsky and Radha Jagadeesan. Games and full completeness for multiplicative linear logic (extended abstract). In R. K. Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science, 12th Conference, New Delhi, India, December 18-20, 1992, Proceedings*, volume 652 of *Lecture Notes in Computer Science*, pages 291–301. Springer, 1992.

[13] Samson Abramsky and Radha Jagadeesan. New Foundations for the Geometry of Interaction. *Inf. Comput.*, 111(1):53–119, 1994.

[14] Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Full abstraction for PCF. *Information and Computation*, 163(2):409–470, 2000.

[15] Samson Abramsky, Pasquale Malacaria, and Radha Jagadeesan. Full abstraction for PCF. In Masami Hagiya and John C. Mitchell, editors, *TACS*, volume 789 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 1994.

[16] Peter Aczel. *Non-well-founded Sets*. Center for the Study of Language and Information Publication Lecture Notes. Cambridge University Press, 1988.

[17] Jirí Adámek and Václav Koubek. On the greatest fixed point of a set functor. *Theor. Comput. Sci.*, 150(1):57–75, 1995.

[18] Norman L. Alling. *Foundations of Analysis over Surreal Number Fields*, volume 141 of *Notas de Matemática*. North-Holland, 1987.

[19] Bowen Alpern and Fred B. Schneider. Recognizing safety and liveness. *Distributed Comput.*, 2(3):117–126, 1987.

[20] Robert B. Ash. *Information Theory*. Dover Publications, 1990.

[21] Michael Barr. Terminal coalgebras in well-founded set theory. *Theoretical Computer Science*, 114(2):299 – 315, 1993.

[22] Michael Barr, Pierre A. Grillet, and Donovan H. van Osdol. *Exact Categories and Categories of Sheaves*, volume 236 of *Lecture Notes in Mathematics*. Springer Verlag, 1971.

[23] Michael Barr and Charles Wells. *Toposes, Triples, and Theories*. Number 278 in Grundlehren der mathematischen Wissenschaften. Springer-Verlag, 1985. Republished in: Reprints in Theory and Applications of Categories, No. 12 (2005) pp. 1-287.

[24] Gerard Berry and Pierre-Louis Curien. Sequential algorithms on concrete data structures. *Theoretical Computer Science*, 20(3):265 – 321, 1982.

[25] Stephen D. Brookes and Shai Geva. Computational comonads and intensional semantics. In M. P. Fourman et al., editor, *pplications of Categories in Computer Science*, volume 177 of *London Math.Society Lecture Note Series*, pages 1–44. Cambridge Univ. Press, 1992.

[26] L. E. J. Brouwer. Besitzt jede reelle Zahl eine Dezimalbruchentwicklung? *Mathematische Annalen*, 83(3):201–210, 1921.

[27] Roberto Bruni and Ugo Montanari. *Models of Computation*. Texts in Theoretical Computer Science. An EATCS Series. Springer International, 2017.

[28] Cristian Calude. *Theories of Computational Complexity*, volume 35 of *Annals of Discrete Mathematics*. North-Holland, 1988.

[29] Gregory Chaitin. On the length of programs for computing finite binary sequences: statistical considerations. *J. ACM*, 16:145–159, 1969.

[30] Alonzo Church. A formulation of the simple theory of types. *The Journal of Symbolic Logic*, 5(2):56–68, 1940.

[31] J.Robin B. Cockett and D.A. Spooner. Categories for synchrony and asynchrony. *Electronic Notes in Theoretical Computer Science*, 1:66 – 90, 1995. MFPS XI, Mathematical Foundations of Programming Semantics, Eleventh Annual Conference.

[32] Bob Coecke, Éric Oliver Paquette, and Dusko Pavlovic. Classical and quantum structuralism. In Simon Gay and Ian Mackie, editors, *Semantical Techniques in Quantum Computation*, pages 29–69. Cambridge University Press, 2009.

[33] Bob Coecke and Dusko Pavlovic. Quantum measurements without sums. In G. Chen, L. Kauffman, and S. Lamonaco, editors, *Mathematics of Quantum Computing and Technology*, page 36pp. Taylor and Francis, 2007. arxiv.org/quant-ph/0608035.

[34] John H. Conway. *On numbers and games*. A K Peters, 2001. (2. ed.).

[35] Thierry Coquand. Infinite objects in type theory. In *International Workshop on Types for Proofs and Programs*, volume 806 of *Lecture Notes in Computer Science*, pages 62–78. Springer, 1993.

[36] Ronald A. Fisher. *Statistical methods and scientific inference*. Hafner Press, 1973.

[37] Melvin Fitting. Intensional logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, spring 2020 edition, 2020.

[38] Peter Freyd and Andre Scedrov. *Categories, Allegories*. Number 39 in Mathematical Library. North-Holland, 1990.

[39] Dov M. Gabbay. *Labelled Deductive Systems*. Labelled Deductive Systems. Clarendon Press, 1996.

[40] Peter Gacs. Every sequence is reducible to a random one. *Information and Control*, 70(2):186 – 192, 1986.

[41] Jean-Yves Girard. Towards a geometry of interaction. In J.W. Gray and A. Scedrov, editors, *Categories in computer science and logic*, volume 92 of *Contemporary Mathematics*, pages 69–108. American Mathematical Society, 1989.

[42] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*. Number 7 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1989.

[43] Kurt Gödel. *The Consistency of the Axiom of Choice and of the Generalized Continuum-hypothesis with the Axioms of Set Theory*. Number 3 in Annals of Mathematics Studies. Princeton University Press, 1940.

[44] Harry Gonshor. *An Introduction to the Theory of Surreal Numbers*, volume 110 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 1986.

[45] Carl Gunter. *Semantics of Programming Languages: Structures and Techniques*. Foundations of Computing. MIT Press, 1992.

[46] Susumu Hayashi. Adjunction of semifunctors: Categorical structures in nonextensional lambda calculus. *Theor. Comput. Sci.*, 41:95–104, 1985.

[47] Matthew Hennessy and Robin Milner. On observing nondeterminism and concurrency. In J. W. de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherlands, July 14-18, 1980, Proceedings*, volume 85 of *Lecture Notes in Computer Science*, pages 299–309. Springer, 1980.

[48] J. Martin E. Hyland and C.-H. Luke Ong. On full abstraction for PCF: I, II, and III. *Inf. Comput.*, 163(2):285–408, 2000.

[49] André Joyal and Ieke Moerdijk. *Algebraic Set Theory*. Number 220 in London Mathematical Society Lecture Notes. Cambridge University Press, 1995.

[50] Andre Joyal, Ross Street, and Dominic Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119(3):447–468, 1996.

[51] G. Max Kelly and Manuel L. Laplaza. Coherence for compact closed categories. *Journal of Pure and Applied Algebra*, 19:193 – 213, 1980.

[52] Stephen C. Kleene. A theory of positive integers in formal logic. *Amer. J. of Math.*, 57:153–173, 1935.

[53] Anders Kock. Closed categories generated by commutative monads. *J. of the Australian Mathematical Society*, 12(4):405–424, 1971.

[54] Anders Kock. Commutative monads as a theory of distributions. *Theory and Applications of Categories*, 26(4):97–131, 2012.

[55] Andrej Kolmogoroff. Zur Deutung der intuitionistischen Logik. *Mathematische Zeitschrift*, 35(1):58–65, 1932.

[56] Georg Kreisel. A survey of proof theory. *Journal of Symbolic Logic*, 33:321–388, 1968.

[57] Sava Krstić, John Launchbury, and Dusko Pavlović. Categories of processes enriched in final coalgebras. In Furio Honsell, editor, *Proceedings of FoSSaCS 2001*, volume 2030 of *Lecture Notes in Computer Science*, pages 303–317. Springer Verlag, 2001.

[58] Joachim Lambek. Deductive Systems and Categories I: Syntactic Calculus and Residuated Categories. *Mathematical Systems Theory*, 2:287–318, 1968.

[59] Joachim Lambek. A fixpoint theorem for complete categories. *Mathematische Zeitschrift*, 103:151–161, 1968.

[60] Joachim Lambek. Deductive Systems and Categories II: Standard Constructions and Closed Categories. In P. Hilton, editor, *Category Theory, Homology Theory and their Applications*, number 86 in Lecture Notes in Mathematics, pages 76–122. Springer-Verlag, 1969.

[61] Joachim Lambek. Deductive Systems and Categories III: Cartesian Closed Categories, Intuitionist Propositional Calculus, and Combinatory Logic. In F. William Lawvere, editor, *Toposes, Algebraic Geometry, and Logic*, number 274 in Lecture Notes in Mathematics, pages 57–82. Springer-Verlag, 1972.

[62] Joachim Lambek. From types to sets. *Adv. in Math.*, 36:113–164, 1980.

[63] Joachim Lambek and Philip Scott. *Introduction to Higher Order Categorical Logic*. Number 7 in Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1986.

[64] F. William Lawvere. An elementary theory of the category of sets. *Proceedings of the National Academy of Sciences of the United States of America*, 52:1506–1511, 1964. Reprinted in *Theory and Applications of Categories* 11(2005) 1–35.

[65] F. William Lawvere. Adjointness in foundations. *Dialectica*, 23:281–296, 1969. reprint in Theory and Applications of Categories, No. 16, 2006, pp.1–16.

[66] Ming Li and Paul M. B. Vitányi. *An introduction to Kolmogorov complexity and its applications (2. ed.)*. Graduate texts in computer science. Springer, 1997.

[67] Ralph Loader. Finitary PCF is not decidable. *Theoretical Computer Science*, 266(1):341 – 364, 2001.

[68] Ernest Manes. *Algebraic Theories*. Number 26 in Graduate Texts in Mathematics. Springer-Verlag, 1976.

[69] Per Martin-Löf. The definition of random sequences. *Information and Control*, 9(6):602 – 619, 1966.

[70] Per Martin-Löf. An intuitionistic theory of types: Predicative part. In Harvey Rose and John Sheperdson, editors, *Logic Colloquium '73*, number 80 in Studies in Logic and the Foundations of Mathematics, pages 73–118. North-Holland, 1975.

[71] Per Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, Naples, 1984.

[72] Elliott Mendelson. *Introduction to Mathematical Logic*. Discrete Mathematics and Its Applications. Taylor & Francis, 6 edition, 2015.

[73] Robin Milner. Processes: a mathematical model of computing agents. In *Proceedings of the Logic Colloquium. Bristol, July 1973*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 157–173. Elsevier, 1975.

[74] Robin Milner. Fully abstract models of typed λ-calculi. *Theoretical Computer Science*, 4(1):1 – 22, 1977.

[75] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Heidelberg, 1982.

[76] Robin Milner. *Communication and concurrency*, volume 84 of *Series in Computer Science*. Prentice Hall, New York, 1989.

[77] Michael Mitzenmacher. A survey of results for deletion channels and related synchronization channels. *Probability Surveys*, 6:1–33, 2009.

[78] Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55 – 92, 1991. Selections from 1989 IEEE Symposium on Logic in Computer Science.

[79] Yiannis N. Moschovakis. Kleene's amazing Second Recursion Theorem. *Bulletin of Symbolic Logic*, 16(2):189–239, 2010.

[80] Andrzej Mostowski. An undecidable arithmetical statement. *Fundamenta Mathematicæ*, 36:143–164, 1949.

[81] Anatol Muchnik and Nikolai Vereshchagin. Shannon Entropy vs. Kolmogorov Complexity. In Dima et al. Grigoriev, editor, *Computer Science — Theory and Applications*, pages 281–291, Berlin, Heidelberg, 2006. Springer.

[82] Robert Paré. On absolute colimits. *J. Alg.*, 19:80–95, 1971.

[83] David Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Theoretical computer science*, number 104 in Lecture Notes in Computer Science, pages 167–183. Springer, 1981.

[84] Dusko Pavlovic. Convenient categories of processes and simulations I: modulo strong bisimilarity. In D. Pitt et al., editor, *Category Theory and Computer Science '95*, volume 953 of *Lecture Notes in Computer Science*, pages 3–24. Springer Verlag, 1995.

[85] Dusko Pavlovic. Maps I: relative to a factorisation system. *J. Pure Appl. Algebra*, 99:9–34, 1995.

[86] Dusko Pavlovic. Convenient categories of processes and simulations II: modulo weak and branching bisimilarities. In A. Edalat et al., editor, *Theory and Formal Methods of Computing 96*, pages 156–167. World Scientific, 1996.

[87] Dusko Pavlovic. Maps II: Chasing diagrams in categorical proof theory. *J. of the IGPL*, 4(2):1–36, 1996.

[88] Dusko Pavlovic. Categorical logic of names and abstraction in action calculus. *Math. Structures in Comp. Sci.*, 7:619–637, 1997.

[89] Dusko Pavlovic. Guarded induction on final coalgebras. *E. Notes in Theor. Comp. Sci.*, 11:143–160, 1998.

[90] Dusko Pavlovic. Relating toy models of quantum computation: comprehension, complementarity and dagger autonomous categories. *E. Notes in Theor. Comp. Sci.*, 270(2):121–139, 2011. arxiv.org:1006.1011.

[91] Dusko Pavlovic. Geometry of abstraction in quantum computation. *Proceedings of Symposia in Applied Mathematics*, 71:233–267, 2012. arxiv.org:1006.1010.

[92] Dusko Pavlovic. Tracing Man-in-the-Middle in monoidal categories. In Dirk Pattinson and Lutz Schroeder, editors, *Proceedings of CMCS 2012*, volume 7399 of *Lecture Notes in Computer Science*, pages 191–217. Springer Verlag, 2012. arXiv:1203.6324.

[93] Dusko Pavlovic. Logic of fusion. In C. Talcott et al., editor, *Proceedings of the Symposium in Honor of Andre Scedrov*, volume 12300 of *Lecture Notes in Computer Science*. Springer, 2020. pp 32.

[94] Dusko Pavlovic. *We, Computer*. textbook manuscript; available on request, 2021.

[95] Dusko Pavlovic and Samson Abramsky. Specifying interaction categories. In E. Moggi and G. Rosolini, editors, *Category Theory and Computer Science '97*, volume 1290 of *Lecture Notes in Computer Science*, pages 147–158. Springer Verlag, 1997.

[96] Dusko Pavlovic and Dominic J.D. Hughes. The nucleus of an adjunction and the Street monad on monads. *CoRR*, abs/2004.07353:87 pp, 2020. to appear in *Theory and Applications of Category Theory*.

[97] Dusko Pavlovic and Vaughan Pratt. On coalgebra of real numbers. *E. Notes in Theor. Comp. Sci.*, 19:133–148, 1999.

[98] Dusko Pavlovic and Vaughan Pratt. The continuum as a final coalgebra. *Theor. Comp. Sci.*, 280(1-2):105–122, 2002.

[99] Dusko Pavlovic and Muzamil Yahia. Monoidal computer III: A coalgebraic view of computability and complexity. In Corina Cîrstea, editor, *Coalgebraic Methods in Computer Science (CMCS) 2018 — Selected Papers*, volume 11202 of *Lecture Notes in Computer Science*, pages 167–189. Springer, 2018. arxiv:1704.04882.

[100] Duško Pavlović and Martín Escardó. Calculus in coinductive form. In Vaughan Pratt, editor, *Proceedings. Thirteenth Annual IEEE Symposium on Logic in Computer Science*, pages 408–417. IEEE Computer Society, 1998.

[101] Gordon Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.

[102] John Reynolds. *Theories of Programming Languages*. Cambridge University Press, 1998.

[103] Hartley Rogers, Jr. *Theory of recursive functions and effective computability*. MIT Press, Cambridge, MA, USA, 1987.

[104] Bertrand Russell. Mathematical logic based on the theory of types. *American Journal of Mathematics*, 30:222–262, 1908. Reprinted in [112], pages 150–182.

[105] Dana S. Scott. A type-theoretical alternative to ISWIM, CUCH, OWHY. *Theoretical Computer Science*, 121(1-2):411–440, 1993. technical report written in 1969.

[106] Jonathan P. Seldin and J. Roger Hindley, editors. *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*. Academic Press, London, 1980.

[107] Michael B. Smyth. Topology. In S. Abramsky and D. Gabbay, editors, *Handbook of Logic in Computer Science (Vol. 1). Background: Mathematical Structures*, pages 641–761. Oxford University Press, Inc., USA, 1993.

[108] Tarmo Uustalu and Varmo Vene. Comonadic notions of computation. *Electronic Notes in Theoretical Computer Science*, 203(5):263–284, 2008.

[109] Matthijs Vákár, Radha Jagadeesan, and Samson Abramsky. Game semantics for dependent types. *Inf. Comput.*, 261(Part):401–431, 2018.

[110] Rob J. van Glabbeek. The Linear Time – Branching Time Spectrum II. The semantics of sequential processes with silent moves. In Eike Best, editor, *Proceedings of CONCUR '93*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81. Springer, 1993.

[111] Rob J. van Glabbeek and W. P. Weijland. Branching time and abstraction in bisimulation semantics. *J. ACM*, 43(3):555–600, 1996.

[112] Jan van Heijenoort, editor. *From Frege to Gödel: a Source Book in Mathematical Logic, 1879–1931*. Harvard University Press, 1967. Reprinted 1971, 1976.

[113] Johann von Neumann. Zur Einführung der transfiniten Zahlen. *Acta litt. Acad. Sc. Szeged*, X(1):199–208, 1923. English translation, "On the introduction of transfinite numbers" in [112], pages 393–413.

[114] Philip Wadler. Propositions as types. *Commun. ACM*, 58(12):75–84, November 2015.

[115] Ernst Zermelo. Über Grenzzahlen Und Mengenbereiche: Neue Untersuchungen Über Die Grundlagen der Mengenlehre. *Fundamenta Mathematicæ*, 16:29–47, 1930.

[116] Wojciech H. Zurek. Algorithmic randomness and physical entropy. *Physical Review A*, 40(8):4731, 1989.

Appendices

A Category R of sets and relations

Relations $A \longleftarrow R \longrightarrow B$ arise in two ways:

a) as subsets $R \overset{r}{\rightarrowtail} A \times B$, so that

$$aRb \iff \exists x \in X.\ a = r_A(x) \wedge r_B(x) = b$$

b) as a nondeterministic functions $A \xrightarrow{\varrho} \wp B$ and $B \xrightarrow{\varrho^o} \wp A$, so that

$$aRb \iff \varrho(a) \ni b \iff a \in \varrho^o(b)$$

where $\wp : \mathbf{S} \longrightarrow \mathbf{S}$ is the powerset monad.

The equivalence between the two views lies at the heart of the elementary structure of topos [23, 38, 63], which can be defined in terms of the correspondece between the subsets $R \rightarrowtail A \times B$ and the elements $\chi_R \in \wp(A \times B)$, and the natural bijections

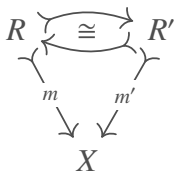
$$\mathbf{S}(X \times A, \wp B) \ \cong\ \mathbf{S}(X, \wp(A \times B)) \ \cong\ \mathbf{S}(X \times B, \wp A) \tag{88}$$

A relational calculus can, however, be developed entirely in terms of subobjects $R \rightarrowtail A \times B$, in type universes without the powerset monad. Process relations are presented from this angle. The universe $\mathbf{S}$ only needs to be *regular* [22, 85]. In addition to the cartesian structure, it is thus also assumed to have the equalizers (i.e., the subsets characterized by equations), which induce the pullback squares. The final assumption, crucial for the relational calculus, is that every function $f : A \longrightarrow B$ has an epi-mono (surjective-injective) factorization: it can be decomposed in the form $f = \left(A \overset{e_f}{\twoheadrightarrow} A' \overset{m_f}{\rightarrowtail} B\right)$, where $e_f \in \mathcal{E}$ and $m_f \in \mathcal{M}$. The family $\mathcal{E}$ can be thought of as the quotient maps (coequalizers), whereas $\mathcal{M}$ are all monics. The family $\mathcal{E}$ is required to be stable under the pullbacks. The category of relations in $\mathbf{S}$ is then defined to be

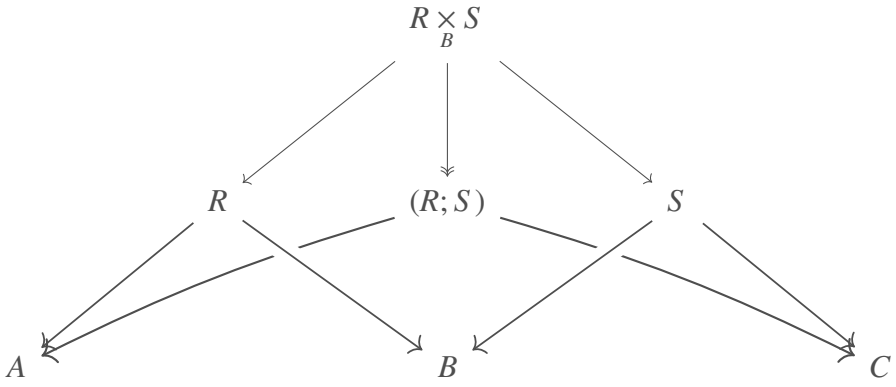
$$\begin{aligned} |\mathbf{R}| &= |\mathbf{S}| \\ \mathbf{R}(A, B) &= \mathcal{M}_{\cong}/(A \times B) \end{aligned} \tag{89}$$

where $\mathcal{M}_{\cong}$ is the set of the equivalence classes modulo the relation

$$m \cong m' \iff$$



Without this quotienting, $\mathbf{R}(A, B)$ would in general be a proper class. The composition of relations $A \overset{R}{\longleftrightarrow} B$ and $B \overset{S}{\longleftrightarrow} C$, viewed as the $\mathcal{M}$-monics $R \rightarrowtail A \times B$ and $S \rightarrowtail B \times C$, is defined using the pullback $R \underset{B}{\times} S$ and the factorization in the following diagram.



The identity $A \longleftrightarrow A$ in $\mathbf{R}_{\mathbf{S}}$ is the diagonal $A \longrightarrow A \times A$ in $\mathbf{S}$. More general categories of relations can be defined in more general situations using technically different but conceptually similar constructions [85, 87]. If $\mathbf{S}$ has the coproducts $+$, they become biproducts in $\mathbf{R}$. The products $\times$ from $\mathbf{S}$ induce a canonical monoidal structure in $\mathbf{R}$, with the compact structure $\eta : 1 \leftrightarrow A \leftrightarrow A \times A$ and $\varepsilon : A \times A \leftrightarrow A \leftrightarrow 1$ on every A [51].

B Proof of Prop. 2.3.3

a) Suppose that $\mathbf{S}$ is a cartesian closed category with the static implication $(A \Rightarrow B)$, and with the process of A-histories $A \overset{(-)}{\longrightarrow} A^+ \overset{(::)}{\longleftarrow} A \times A^+$ for every A. Then $[A, B] = (A^+ \Rightarrow B)$ is the state space of the final AB-machine with the structure map

$$A \times (A^+ \Rightarrow B) \quad \xrightarrow{\upsilon = \langle \upsilon^\bullet, \upsilon^\circ \rangle} \quad B \times (A^+ \Rightarrow B) \tag{90}$$

where the components are derived by evaluating along the components of the A-history process

$$A \times (A^+ \Rightarrow B) \xrightarrow{(A\times(-)\Rightarrow B)} A \times (A \Rightarrow B) \xrightarrow{\varepsilon} B$$

$$A \times (A^+ \Rightarrow B) \xrightarrow{\upsilon^\bullet} B$$

$$A^+ \times A \times (A^+ \Rightarrow B) \;\cong\; A \times A^+ \times (A^+ \Rightarrow B) \xrightarrow{(::)\times(A^+\Rightarrow B)} A^+ \times (A^+ \Rightarrow B) \xrightarrow{\varepsilon} B$$

$$A \times (A^+ \Rightarrow B) \xrightarrow{\upsilon^\circ} (A^+ \Rightarrow B)$$

To show that (90) is a final machine, first note that every AB-machine $A\times X \xrightarrow{\xi\,=\,\langle\xi^\bullet,\xi^\circ\rangle} B\times X$ induces an A-history process

$$A \xrightarrow{\kappa(-)} (X \Rightarrow B) \xleftarrow{\kappa(::)} A \times (X \Rightarrow B) \tag{91}$$

with the components

$$A \times X \xrightarrow{\xi^\bullet} B \qquad\qquad X \times A \times (X \Rightarrow B) \;\cong\; A \times X \times (X \Rightarrow B) \xrightarrow{\xi^\circ\times(X\Rightarrow B)} X \times (X \Rightarrow B) \xrightarrow{\varepsilon} B$$

$$A \xrightarrow{\kappa(-)} (X \Rightarrow B) \qquad\qquad A \times (X \Rightarrow B) \xrightarrow{\kappa(::)} (X \Rightarrow B)$$

By Sec. 2.3.1, the A-history process κ induces the catamorphism (i.e. fold, banana-function) $(\!|\kappa|\!)$

$$\begin{array}{ccccc} & & A^+ & \xleftarrow{\;(::)\;} & A \times A^+ \\ & \overset{(-)}{\nearrow} & \vdots & & \vdots \\ A & & (\!|\kappa|\!) & & A\times(\!|\kappa|\!) \\ & \underset{\kappa(-)}{\searrow} & \downarrow & & \downarrow \\ & & (X \Rightarrow B) & \xleftarrow{\;\kappa(::)\;} & A \times (X \Rightarrow B) \end{array} \tag{92}$$

On the other hand, the transposition

$$A^+ \xrightarrow{(\!|\kappa|\!)} (X \Rightarrow B)$$

$$X \xrightarrow{[\![\xi]\!]} (A^+ \Rightarrow B)$$

induces the anamorphism (unfold, lens-function) $[\![\xi]\!]$

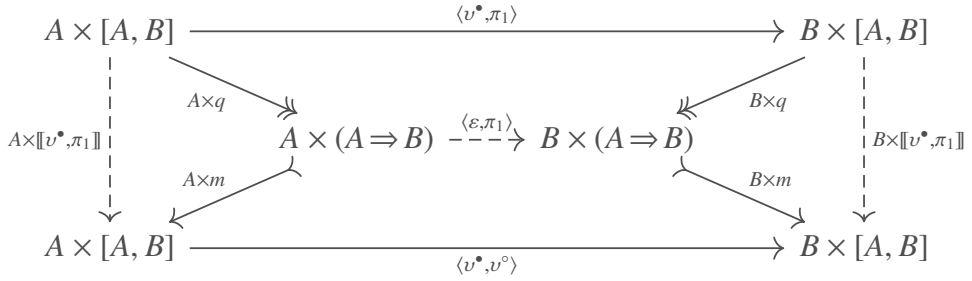
$$\begin{array}{ccc} X \times A & \xrightarrow{\;\xi\;} & X \times B \\ \vdots & & \vdots \\ [\![\xi]\!]\times A & & [\![\xi]\!]\times B \\ \downarrow & & \downarrow \\ (A^+ \Rightarrow B) \times A & \xrightarrow[\;\upsilon\;]{} & (A^+ \Rightarrow B) \times B \end{array} \tag{93}$$

which shows that υ makes $[A, B]$ into the process implication as in Sec. 2.2.2. The diagram chase showing that the catamorphism (92) commutes if and only if (93) commutes is an instructive exercise.

b) The assumption is that **S** has final AB-machines

$$A \times [A,B] \xrightarrow{\upsilon\,=\,\langle \upsilon^\bullet, \upsilon^\circ\rangle} B \times [A,B]$$

Replacing the second component by the projection gives the machine which induces the anamorphism $[\![\upsilon^\bullet, \pi_1]\!]$, which makes the outer square in the following diagram commute.



(94)

Since $[\![\upsilon^\bullet, \pi_1]\!]$ is also endomorphism on the AB-machine $A \times [A,B] \xrightarrow{\langle \upsilon^\bullet, \pi_1\rangle} B \times [A,B]$, the uniqueness of $[\![\upsilon^\bullet, \pi_1]\!]$ as an AB-machine homomorphism from $\langle \upsilon^\bullet, \pi_1\rangle$ to $\langle \upsilon^\bullet, \upsilon^\circ\rangle$ implies that it is an idempotent:

$$[\![\upsilon^\bullet, \pi_1]\!] \circ [\![\upsilon^\bullet, \pi_1]\!] \;=\; [\![\upsilon^\bullet, \pi_1]\!]$$

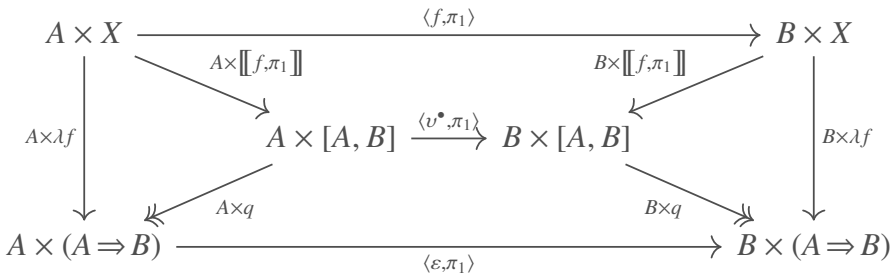
Here we use the assumption that the idempotents split in **S**, and define $(A \Rightarrow B)$ as the splitting

$$[\![\upsilon^\bullet, \pi_1]\!] \;=\; \left([A,B] \overset{q}{\twoheadrightarrow} (A \Rightarrow B) \overset{m}{\rightarrowtail} [A,B]\right)$$

also displayed in (94). The component $A \times (A \Rightarrow B) \xrightarrow{\varepsilon} B$ of the factoring defined there is the counit of the adjunction $A \times (-) \dashv (A \Rightarrow -)$, defined

$$\begin{array}{rcl} \mathbf{S}(X \times A, B) & \xrightarrow{\lambda} & \mathbf{S}(X, (A \Rightarrow B)) \\ f & \longmapsto & \lambda f = q \circ [\![f, \pi_1]\!] \end{array}$$

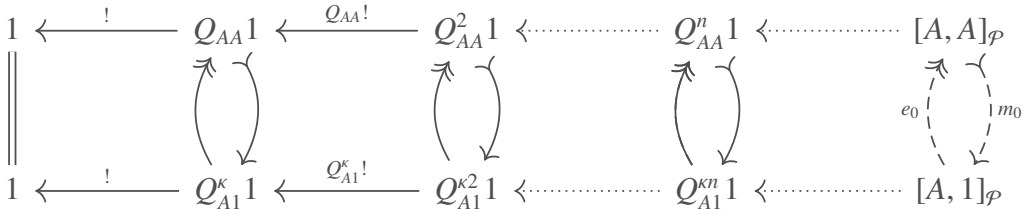
To show that $\varepsilon \circ (\lambda A \times f) \;=\; f$, chase the following diagram:



□

C Proof sketch for Lemma 5.1

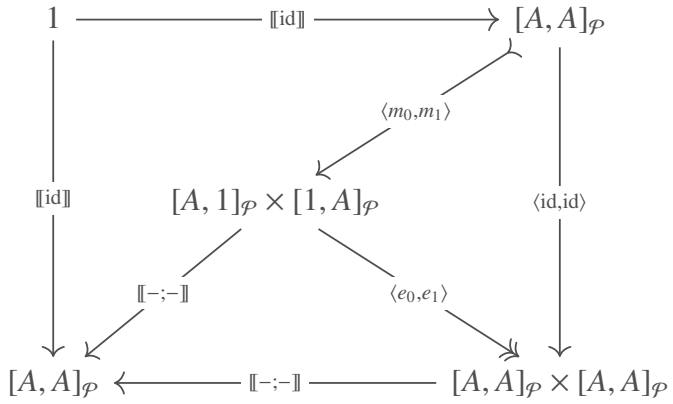
Since $\#A \leq \aleph_0$, there is an ordinal number $\kappa \leq \omega$ large enough to support an retraction $\mathcal{P}(A \times A) \rightarrowtail \mathcal{P}^\kappa(A) \twoheadrightarrow \mathcal{P}(A \times A)$, and thus also $Q_{AA}1 \rightarrowtail Q^\kappa_{A1}1 \twoheadrightarrow Q_{AA}1$ for the functor Q defined in (42). Hence the tower of retractions:



The symmetry $A \times 1 \cong 1 \times A$ lifts to a smilar retraction

$$[A,A]_{\mathcal{P}} \overset{m_1}{\rightarrowtail} [1,A]_{\mathcal{P}} \overset{e_1}{\twoheadrightarrow} [A,A]_{\mathcal{P}}$$

With these retractions, the proof boils down to showing the commutativity of the following diagram



where $[\![-;-]\!]$ are the enriched compositions, constructed like in (29) (or see [57] for more details), whereas $[\![\mathrm{id}]\!]$ is the enriched identity, constructed as the anamorphism (final coalgebra homomorphism) from the identity machine $A \times 1 \overset{\eta}{\rightarrow} \mathcal{P}(A \times 1)$, where η is the unit of the monad $\mathcal{P}$. This diagram says that $j = m_0\,[\![\mathrm{id}]\!] \in \mathbf{S}^{\mathcal{P}}(A,1)$ and $r = m_1\,[\![\mathrm{id}]\!] \in \mathbf{S}^{\mathcal{P}}(1,A)$ display A as a retract of 1 in $\mathbf{S}^{\mathcal{P}}$, i.e. that they compose to

$$\mathrm{id}_A \;=\; \left(A \xrightarrow{j=m_0[\![\mathrm{id}]\!]} 1 \xrightarrow{r=m_1[\![\mathrm{id}]\!]} A\right) \tag{95}$$

□

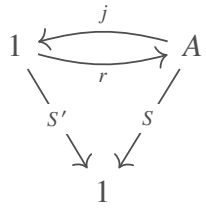
D Proof of Corollary 5.2

Since the embedding (62) is full and faithful by definition, we only need to prove that it is essentially surjective: for an arbitrary object $S \in \mathsf{DProc}_{\leq \aleph_0}$ we must find $S' \in \mathsf{dProc}$ such that $S \cong S'$ in

$\mathsf{DProc}_{\leq\aleph_0}$. An object of $\mathsf{DProc}_{\leq\aleph_0}$ is a dynamic relation $A \overset{S}{\longleftrightarrow} 1$ in $\mathbf{S}^{\mathcal{P}}$, where $\#A \leq \aleph_0$. An object of dProc is a hyperset S', viewed as a dynamic relation $1 \overset{S'}{\longleftrightarrow} 1$ in $\mathbf{S}^{\mathcal{P}}$. By Lemma 5.1, there are the relations $j \in \mathbf{S}^{\mathcal{P}}(A, 1)$ and $r \in \mathbf{S}^{\mathcal{P}}(1, A)$ such that $(j; r) = \mathrm{id}_A$. Setting

$$S' \;=\; \left(1 \overset{r}{\longleftrightarrow} A \overset{S}{\longleftrightarrow} 1\right)$$

assures that the inner triangle in the following diagram commutes.



The outer triangle commutes because $S' \circ j = S \circ r \circ j = S$ by (95). So we have the morphisms $r \in \mathsf{DProc}_{\leq\aleph_0}(S', S)$ and $j \in \mathsf{DProc}_{\leq\aleph_0}(S, S')$. They form an isomorphism because $r \circ j = \mathrm{id}_S$ by (95) again, and $j \circ r \in \mathsf{DProc}_{\leq\aleph_0}(S', S')$ must be an identity because S' is a subobject of the terminal object in $\mathsf{DProc}_{\leq\aleph_0}$. □

E Traces and the Int-construction

The *trace* operation on a symmetric (or braided) monoidal category $(C, \otimes, I)$ is typed by the rule

$$\frac{A \otimes Y \overset{f}{\to} B \otimes Y}{A \overset{\mathrm{Tr}_Y(f)}{\longrightarrow} B}$$

The equations for this operation, with some examples and explanations can be found in [6, 50, 92]. The free compact category over any traced monoidal C

$$\begin{array}{rcl} |\mathsf{Int}_C| & = & |C|_- \times |C|_+ \\ \mathsf{Int}_C(A, B) & = & C\,(A_- \otimes B_+, B_- \otimes A_+) \end{array} \tag{96}$$

where $X_- = \{-\} \times X$ and $X_+ = \{+\} \times X$. The composition of $\mathsf{Int}_C(A, B) \times \mathsf{Int}_C(B, C) \overset{\bullet}{\to} C(A, C)$ is defined by

$$A_- \otimes B_+ \overset{f}{\longrightarrow} B_- \otimes A_+ \qquad\qquad B_- \otimes C_+ \overset{g}{\longrightarrow} C_- \otimes B_+$$

$$A_- \otimes C_+ \otimes B_- \otimes B_+ \overset{\sigma}{\cong} A_- \otimes B_+ \otimes B_- \otimes C_+ \overset{f\otimes g}{\longrightarrow} B_- \otimes A_+ \otimes C_- \otimes B_+ \overset{\sigma}{\cong} C_- \otimes A_+ \otimes B_- \otimes B_+$$

$$g \bullet f \;=\; \left(A_- \otimes C_+ \overset{\mathrm{Tr}_{B_-\otimes B_+}(\sigma\circ(g\otimes f)\circ\sigma)}{\longrightarrow} C_- \otimes A_+\right)$$

F The extended reals as alternating dyadics

Recall from Sec. 6.4.1 that $\mathbb{R} = \mathbb{R} \cup \{\infty, -\infty\}$ is the extended real continuum, and that $\Sigma^{\circledast} = \coprod_{i=0}^{\omega+1} \Sigma^i$ is the set of finite or infinite (countable) strings of symbols from $\Sigma = \{-, +\}$, which are treated in (97) as $\{-1, 1\}$.

Define the value of the function $\Phi : \Sigma^{\circledast} \longrightarrow \mathbb{R}$ on an arbitrary string $\varsigma = (\varsigma_0\, \varsigma_1\, \varsigma_2, \ldots)$ to be

$$\Phi(\varsigma) \quad = \quad z \cdot \varsigma_0 + \sum_{i=z+1}^{\infty} \frac{\varsigma_i}{2^{i-z}} \tag{97}$$

where $z = \mu n.\, \varsigma_n \neq \varsigma_{n+1}$ is the length of the initial segment before the sign flips. If ς is the infinite string of either one sign or the other, then z is infinite, and the value of $\Phi(\varsigma)$ is either ∞ or $-\infty$. Leaving the two infinities aside, Φ establishes a bijection between the remaining Σ-stings, where the sign eventually flips, and the finite real numbers from $\mathbb{R}$. For an arbitrary $x \in \mathbb{R}$, the string $\upsilon \in \Sigma^{\circledast}$ such that $x = \Phi(\upsilon)$ can be constructed as follows:

- Decompose the real line as the disjoint union of the closed-open and open-closed intervals

$$\mathbb{R} \quad = \quad \coprod_{n=1}^{\infty} [-n, -n+1) + \{0\} + \coprod_{n=1}^{\infty} (n-1, n]$$

  leaving the 0 on its own. Then there are 3 cases:

- (0) If $x = 0$ then υ is the empty string ().
- (-) If $x \in [-n_0, -n_0 + 1)$, then υ begins with $\underbrace{- - \cdots -}_{n_0}$.
- (+) If $x \in [n_0 - 1, n_0)$, then υ begins with $\underbrace{+ + \cdots +}_{n_0}$.
- In case (-), find
  - the smallest n_1 such that $x \leq -n_0 + \sum_{i=1}^{n_1} \frac{1}{2^i}$ and append $\underbrace{+ \cdots +}_{n_1}$ to υ;
  - the smallest n_2 such that $x \geq -n_0 + \sum_{i=1}^{n_1} \frac{1}{2^i} - \sum_{i=1}^{n_2} \frac{1}{2^{n_1+i}}$ and append $\underbrace{- \cdots -}_{n_2}$ to υ;
  - the smallest n_3 such that $x \leq \cdots$, etc.
- In case (+), find
  - the smallest n_1 such that $x \geq n_0 - \sum_{i=1}^{n_1} \frac{1}{2^i}$ and append $\underbrace{- \cdots -}_{n_1}$ to υ;
  - the smallest n_2 such that $x \leq \cdots$, etc.
- If you ever reach a sum equal to x, then halt and leave υ finite. Otherwise υ is infinite.

In any case, it is easy to see that $\Phi(\upsilon) = x$ and that $\Phi(\upsilon) = \Phi(\zeta)$ implies $\upsilon = \zeta$. So Φ is an injection. And we have just shown that it is a surjection by constructing for an arbitrary $x \in \mathbb{R}$ a $\upsilon \in \Sigma^{\circledast}$ such that $x = \Phi(\upsilon)$. The function Φ defined by (97) is thus the claimed bijection.