Unifying Semantics for Programming and Verification

Technical Report · April 2018

DOI: 10.13140/RG.2.2.31282.30403

CITATIONS
0

READS
91

1 author:

Sergey Goncharov
Friedrich-Alexander-University of Erlangen-Nürnberg
28 PUBLICATIONS **96** CITATIONS

SEE PROFILE

Unifying Semantics for Programming and Verification

Sergey Goncharov

April 28, 2018

Contents

1 Unifying Semantics: Methodology

Correct well-designed semantics precedes solutions of principal research problems in computer science. As a somewhat allusive, but spectacular illustration of this motto we may view the seminal work of Turing [110], which arguably gave rise to computer science as a separate discipline not entirely reducible to pure mathematics. In his famous result of undecidability of the *halting problem* for what then became known as the *Turing machine* he made two essential steps to achieve the goal:

1. He proposed an idealized formal model of a computing machine and convincingly argued that this notion accurately captures the informal concept of effective computation, however sophisticated (*Church-Turing thesis*).

2. He then proved that there is no efficient procedure in the sense of (1) for deciding if a given Turing machine ever halts on a given input.

While the significance of the second step is apparent from a purely technical perspective – it is a statement with a clear and meaningful mathematical content – the first one is really crucial, and its importance cannot be overestimated. The mere fact that Turing machines became a rather direct prototype of the modern computers and until today qualitatively describe their behavior already speaks for itself. The ontological significance of Turing's approach can be illustrated by comparison to Gödel's recursion theory [39] utilized by Church to give the first example of an undecidable problem [25], which has immediately been realized to be qualitatively the same kind of result as the result of Turing. Both Church's and Turing's results provide negative solutions to Hilbert's famous *Entscheidungsproblem*, and despite the fact that Church's construction came along almost one year earlier than Turing's, the latter is commonly credited as the pivotal step for resolving the issue in the negative, for it not only provided a definite mathematical answer, but it also featured a general method, a far-reaching model and a strong argumentation behind it for reducing the intuitive notion of effective computability to first principles. A *computational interpretation*, or *computational meaning* of mathematical phenomena became a crucial theme in computer science since then, just like *physical meaning* once became and remains to be a crucial theme in physics.

1.1 General View of Semantics

The role of semantics in the above example might become apparent only as an a posteriori reflection – at the time when computer science detached itself from mathematics, it was too early to clearly identify the key distinctions between the two disciplines. Putting it simplistically, in mathematics things are what they denote, e.g. $2 + 2$ *means* 4

in the sense that it *is* 4. In computer science, we say that 4 *denotes* $2+2$, or $2+2$ *evaluates* to 4, for what is hidden behind the equality $2+2=4$ is an actual computational process of obtaining 4 from $2+2$ (which is of course trivial in this example). Semantics is precisely the mathematical term for *meaning*. Hence, generally, semantics is a connection between *languages* and *models*. In retrospect we can view Turing machines as (operational) models, which Turing essentially argued to be adequate w.r.t. the language of mathematical constructs for defining computable numbers. This contrasted with Church's result, couched completely within the language of λ-terms without a convincing model behind it, and in fact finding such a model remained a notorious open problem until finally solved by Scott [98, 96].

When speaking about semantics one usually mentions the classification into *operational*, *denotational* and *axiomatic* semantics, roughly characterized as follows:

- Operational semantics is defined by the agency of a suitable (explicit or implicit) notion of abstract machine (one can again think of Turing machine as an example), so that the meaning of a phrase is given in terms of machine runs triggered by that phrase.
- Denotational semantics interprets a phrase by reference to a suitable semantic domain where the phrase takes a value, called its *denotation*
- Axiomatic semantics describes the meaning of a phrase indirectly via axioms and rules of some logic of properties.

Denotational semantics is also sometimes called *mathematical semantics*, although a running assumption we would like to accept here is that all the above styles of semantics can perfectly be regarded as mathematical.

1.1.1 Denotational vs. Operational Semantics

One may then wonder if the delineations between the above three styles of semantics can be expressed in mathematical terms, e.g. if operational semantics is defined w.r.t. some machine, cannot one regard a suitably defined history of its moves, or the final result, or both as the denotation of the corresponding phrase? The answer to this is that, first, even if it works, this would not reduce denotational semantics to operational semantics, for there can be interesting denotational models obtained in a different way. A second issue is of a non-mathematical nature, and is somewhat akin to the above issue with solutions of the Entscheidungsproblem: not every model is (immediately) satisfactory, and even a specific description of a known model can be more valuable than the model itself – a perfect illustration of this is the dramatic history of the *full abstraction problem for PCF* [80]. The third issue however can to a certain degree be reduced to formal mathematical considerations as follows. One requirement imposed on denotational semantics is that of *compositionality*, i.e. the denotation of a compound phrase must be constructed from the denotations of its parts. Now, if we agree to turn the machine transition history into a denotational semantics, this may not be compositional, e.g. if the phrase of interest describes successive updates of a memory cell: the programs $x := 1; x := 2$ and $x := 2$ must denote the same, but they have different computation histories. One may choose to ignore the computation histories entirely and

regard as the requisite denotation only the final result of the computation if the computation terminates. The problem however is that it need not terminate, and then one would be forced to introduce a designated semantic unit for divergence. If one aims to cover functional abstraction in a compositional manner, one is furthermore forced to include partial function spaces as legitimate semantic domains. By iterating this and further language constructions, the induced notion of semantic domain becomes rather remote from the original idea of using it as a carrier for computed values.

Further reflections in this style potentially lead to a rediscovery of classical Scott/Strachey denotational semantics [107, 99], and more specifically to Scott's formulation of domain theory [97], which is not a goal pursued here. Instead, having said all this, we return to the original idea of generating denotational semantics from the operational semantics, and emphasize that it can be quite productive if the above issues can be circumvented. This is prominently illustrated by Turi and Plotkin's *abstract GSOS semantics* [109], where the outlined passage from operational to denotational semantics is implemented. More concretely, the upshot of this work is that a suitably formalized rule based system for operational semantics does indeed yield a sensible fully abstract denotational model essentially obtained by recording derivation steps of this system run on all possible given input terms. The obtained denotational domain is incidentally a final coalgebra νB of a functor B intuitively capturing the atomic entries of the (branching) computation history, obtained by successive calls of operational semantic rules. The limitations of the GSOS semantics making the Turi and Plotkin construction work are in fact explicit in the name: GSOS = *Guarded Structural Operational Semantics*, where "structural" means that the derivation rules handle input terms inductively over the term structure, and "guarded" means that each rule application inevitably contributes to the prospective denotation of the term at hand. With this in mind, it is clear that denotational semantics thus generated would indeed distinguish programs like $x := 1; x := 2$ and $x := 2$, but this can be seen positively, e.g. if the above programs are executed in a concurrent environment over shared memory, for a third program can alter the value of x in the meantime, hence the above two programs are justifiably non-equivalent [47]. This explains why (abstract) GSOS is particularly successful on first order languages stable under concurrency and non-determinism.

The threshold between "operationally driven" and "genuinely denotational" models sketched above forms the core of the modern discourse in semantics design. One way it unravels is by exploiting the mentioned notion of guardedness and its analogues, which are mathematically more tractable objects than the notion of operational semantics as a whole. For example, contractive maps of complete metric spaces can be justifiably seen as guarded among non-expansive maps [17, 75], which gives rise to *metric* or, more specifically, *ultrametric semantics* [64]. Another direction is *coalgebraic semantics* [92, 57] for which guardedness in an inherent phenomenon. Various concrete examples of this kind reveal trade-offs between incorporating more or less amount of operational behavior in a concrete semantics, to address correspondingly more or less reactive/interactive/concurrent features. This discussion is the centerpiece of *game semantics*, in particular, game semantics of high-order languages such as PCF, where one rather speaks about the trade-off between *intensionality* and *extensionality* in denota-

tional semantics [3].

The game semantics of PCF [49, 4] is in fact an interesting subject for speculating to which extent it can be seen as a conceptual instance of the outlined scenario for constructing operationally driven semantics. Again, from a purely mathematical perspective, the model in question is Milner's familiar inequationally fully abstract order-extensional model [77], which is essentially a term model. Game semantics fundamentally reconceptualizes it and obtains it anew from arguably more suitable universal principles in two steps: by forming a model in an intensional category of games and strategies, and by calculating a certain extensional homomorphic image of it via a procedure known as *extensional collapse*. A factorization via an intensional model is provably inevitable (due to *Loader's theorem* [3]) and the constructed game model bears signs of having operational nature – indeed in [1] Abramsky proposes to think of intensional semantics as "syntax-free operational semantics". This intuition however is rather hard to make precise, for game semantics, although it captures fundamental principles behind processes and their interaction, does not appear to admit a simple mathematical description via primitive mathematical notions and universal constructions.

1.1.2 Denotational vs. Axiomatic Semantics

The connection between denotational and axiomatic semantics appears to be less elusive than the one between denotational and operational semantics, largely thanks to Abramsky's thesis [2] where the fundamental *Stone duality* is interpreted as the duality between domains (seen as topological spaces under the Scott topology) and the corresponding algebras of properties of domain elements, i.e. program denotations. A standard example of axiomatic semantics is *Hoare logic*, which can be used to define the meaning of a program p via the Hoare triples $\{\phi\}p\{\psi\}$ it satisfies where ϕ and ψ are state assertions constraining p. A salient feature of Abramsky's framework is that the precise form of such constraints does not matter – what matters is that they encode Scott open sets of the domain and thus yield an indirect description of its elements.

The main expository case elaborated by Abramsky is the one of *bifinite domains*, which includes Scott domains. Even within the realm of domain theory the restriction to bifinite domains is significant [60] (not to mention that there is no consensus about the right notion of domain within domain theory [61, 51, 13]), and, as we briefly discussed above, denotational semantics is not confined to domain semantics only. However the general idea of relating denotational and axiomatic semantics via a duality seems to be a generally appropriate powerful guiding principle, whether or not it can be implemented immediately and/or literally. A concrete implementation can vary and one immediate modification is to replace duality with a contravariant adjunction [82].

One peculiar fact about the outlined correspondence between axiomatic and denotational semantics is related to the use of axiomatic semantics for verification where this instructive duality can be quickly lost to sight. As for verification purposes, especially for model checking, one usually restricts to decidable properties and finite models, the topological intuition is neglected more often than not. For example, the Plotkin powerdomain, denotationally responsible for finite non-determinism, dually corresponds to

the $\Box$ and $\Diamond$ operators of intuitionistic normal modal logic. By enforcing the law of excluded middle, the logic becomes classical and the corresponding frame of opens becomes a Boolean algebra, whose topological duals are *Stone spaces*. From the modal perspective, topological spaces correspond to carriers of Kripke frames and, if such a frame is finite, then as a Stone space it is discrete, i.e. essentially a finite set. Now, without knowing the global context, it is highly nontrivial to come up with the idea that the right generalization starting from finite sets is not to infinite sets but to infinite Stone spaces.

In [G8, G10] we utilize the ideas about the relation between topology and logics to revisit and generalize the Cook's classical result [27] on soundness and relative completeness of Hoare logic. Our analysis reveals that under general assumptions on the model, the object of truth values for state assertions becomes a *frame*, which is a Boolean algebra only in specific cases.

1.2 Towards Categorical Semantics

In the previous discussion of semantic phenomena, we have tried to keep in focus their mathematical extent. We proceed to further distill and classify the relevant mathematical devices in the context of category theory, and to that end first and foremost adopt the categorical viewpoint. It is commonplace nowadays to say that categorical language is appropriate for dealing with semantics: categories of domains are standardly required to be Cartesian closed, as it is necessary for them to be models of simply typed λ-calculus, algebra and co-algebra make their entrance via GSOS semantics, and numerous Stone-like dualities are really dualities of certain topological and algebraic categories. We would like to emphasize however that sticking to the categorical language is a too general commitment, and does not alone identify the approach and the role of categorical models.

1.2.1 Algebraic vs. Domain Semantics

For illustration, we compare *domain semantics* and *algebraic semantics*. Roughly, the distinction is seated in the fact that domain categories have a rather pure categorical structure, e.g. it is easy to argue that total recursion operators inherent to domain categories conflict with coproducts, specifically with internal Boolean algebra objects $2 = 1 + 1$ (because calculating the fixpoint of the negation causes havoc [48]). A domain theoretic solution is also well-known – to use *smash products* instead of coproducts, or more compellingly distinguish between domains and *predomains* where the former have fixpoints, the latter have coproducts and both are interconnected in a conceptually highly satisfactory manner [33]. Manes and Arbib [73] introduce a versatile notion of *partially additive category* as a basis for algebraic semantics, and in this light criticize postulates of domain theory as being too restrictive [73, p. 294]. Somewhat paradoxically, by pursuing the goal of overcoming the limitations of domain theory, the authors base their framework on a rather specific feature of set-based models, namely countable coproducts,

which are not available e.g. in Lawvere's axiomatization of the category of sets [66], and generally conflict with realizability-based universes [81]. Further aspects of algebraic semantics exposed in [73], such as modelling Hoare-style reasoning via Boolean algebras, reveal a general commitment to classical logic, which is at odds with the fact of undecidability of program termination, suggesting that the logic of state assertions must be properly intuitionistic [104, 116]. These considerations indicate that algebraic semantics, at least as presented in [73], is rather more suitable for decidable properties of programs. Contrastingly, domain theory is in harmony with constructivity and intuitionism and relies on them in a substantial manner as signified by the *synthetic domain theory* program [91, 83, 50]. Another witness of the deep connection between program semantics and intuitionism is the fact that polymorphic (second-order) λ-calculus has no *classical* set-theoretic models [90], but it does have *intuitionistic* set-theoretic models [84].

The conclusion we would like to draw from this comparison is that category theory provides a spacious realm for hosting and interconnecting various styles of semantics, even if they would otherwise come into collision. The following quote from [73]:

> We hope the reader will not misconstrue our claims that other avenues of approach are possible as arguing against the merits of the ordered approach. Rather, we are reacting to the acceptance, certainly championed by some, of ordered semantics as the *only* mathematical foundation for the theory of program semantics.
>
> — E.G. Manes, M.A. Arbib, *Algebraic approaches to program semantics*

we would like to complement with the *genericity principle*, which we understand as the requirement that categorical modelling should be consistent with various aspects of semantics by allowing for broad classes of models in an axiomatic manner, so that more specific models can be potentially obtained by restriction via suitable categorical constructions. In other words, we aim to see categorical semantics as a means for establishing *more general models* rather than *alternative models*.

1.2.2 Domains and Universes

The explosive development of domain theory and the recurring quest for the right category of domains somewhat overshadow the fact that categories of domains are only suitable for modelling programs and not for reasoning about them, although this becomes more explicit in axiomatic domain theory and even more so in synthetic domain theory. Categorically speaking, this implies the necessity of (at least) two categories for program semantics where one (domains) is supposed to be constructible within another, the latter being some sufficiently broad universe. Abstracting away from the domain case, and following the principles of Abramsky [2] of approaching the issue from an ontological perspective, we witness two kinds of categories appearing in semantics: those for modelling computations, and those for modelling logics. For convenience, let us call the latter kind *logical universes*. A rough classification of categories relevant to

semantics into these two families can be made based on the presence/absence of a total recursion operator, which is typically inherent to categories of computations, but not to logical universes. As we argued in Section 1.1.2, there is evidence that logical universes must support some version of intuitionistic logic. The diversity of candidates for the role of a logical universe includes intuitionistic Zermelo–Fraenkel set theory (IZF), realizability toposes [81], the topos of trees [16], nominal sets [70], predicative toposes [14] and various flavors of intensional type theory. On the other hand, the Stone duality perspective suggests that the category for program logics must be dual to the category of programs. This is not in conflict with the previous idea, for both categories can coexist within a large logical universe as demonstrated in Taylor's *abstract Stone duality* [108].

The generic approach to semantics in the outlined dichotomy of categorical models suggests an agnostic view of assuming as few categorical properties as possible without a strict commitment to any specific setup. The same principle applies to categories for modelling computations, and it may not even be necessary to assume them to be domain-like, for instead of recursion they might be confined to supporting *iteration*, or no fixpoints at all. A particularly interesting task is the one of identifying semantic notions that can be flexibly instantiated across various categories on both sides of the dichotomy. As one example of such generic notion we name *guarded recursion* [75]. For the sake of further presentation we distinguish two further generic categorical devices of this kind of fundamental importance for semantics, which are *monads* and *coalgebras*.

1.2.3 Extensional Monads

Monads crop up in categorical semantics on numerous occasions, which seems to explain some inconsistency of their use. One principled way of using monads stems from the seminal thesis of Lawvere [67] who introduced finitary monads as a categorical counterpart of *clones* of equational theories, in particular, the *free monad* over an algebraic signature captures the term languages over this signature, and this is the role monads play in Turi and Plotkin's abstract GSOS semantics [109]. The view of finitary monads as algebraic theories was exploited in numerous ways since Lawvere's initial work, notably by Elgot [30] and subsequently by Boom and Ésik [19] to axiomatize and reason about *monad-based iteration*. In [38] Giry (using further insights by Lawvere) introduced a monad for probability distributions, which subsequently (in simplified form) was accommodated in *probabilistic semantics*, and, more generally, *quantitative semantics* [45]. An important example of a monad that fails to be algebraic in Lawvere's original sense is the *lifting monad* for (pre-)domains in constructive logical universes, which is a monad for adjoining a fresh bottom element (the lifting monad can however be seen as algebraic in an extended, specifically enriched, sense [54]). A crucial insight about the role of monads in semantics is due to Moggi [79], who identified (strong) monads as carriers of *computational effects*, e.g. randomness for probabilistic monads, divergence for lifting monads, etc. Later, Plotkin and Power established a connection between computational monads and algebraic theories in their theory of *algebraic effects* [85, 87] and so, in a sense, closed the circle by drawing on Lawvere's original treatment of equational theories. A remarkable role in Moggi's presentation is played by strength: a monad

is strong if it is equipped with a special binary natural transformation called *strength*, satisfying a number of technical coherence conditions [78]. Although this notion has been known for a long time, Moggi gives it a principled interpretation as a semantic mechanism for interpreting programs in a multivariable context.

Approaches to semantics based on monads are highly general and undemanding to the host category. This generality can be flexibly refined by restricting to special classes of monads, e.g. monads supporting iteration.

The core idea of Moggi's approach [79] is the factorization of concrete categorical program semantics through the generic *computational metalanguage*, which is essentially the *internal language* of strong monads. The main construct of this language is imperative-style program composition

$$\mathsf{do}\, x \leftarrow p;\, q,$$

which binds the result of executing a side effecting program p to x and subsequently executes a side effecting program q depending on x. In the terminology of Abramsky [3] we can qualify this approach as belonging to the *extensional paradigm*, for we can interact with programs only by calling them on given arguments and using the output result values, but not by communicating with them at runtime or using any knowledge about their internal structure.

1.2.4 Intensional Coalgebras

In contrast to monads, coalgebras [57] morally belong to the *intensional paradigm*, because program semantics based on them explicitly distinguishes computation steps taken during program runs. This allows one, e.g., to count these steps, read the intermediate results of computations and interact with them at run time before they terminate (no matter if they terminate). Essentially, coalgebra-based semantics is the semantics of *computation processes*. The fact that in contrast to monads, coalgebras were not so deeply and readily accommodated in program semantics seems to be again due to the overwhelming impact of domain theory: in domain theory one uses fixpoint objects for modelling data types which are traditionally seen as initial algebras, but the standard categories of domains are algebraically compact [35, 36], i.e. initial algebras and final coalgebras coincide. As a result, any occasional entrance of coalgebras in domain-based semantics, at least in the form of final coalgebras, becomes obscured. We would like to emphasize however that the described technical effect of algebra-coalgebra coincidence is not the reason for marginalization of final coalgebras in domain theory. Quite to the contrary, this effect itself is a natural consequence of the core postulates of domain theory in which only positive information is regarded as observable and stable, contrary to negative information. Therefore, in contrast to coalgebras in sets, domain coalgebras may on the one hand contain more elements (because of directed completeness), but on the other hand may be coarser, e.g. by not distinguishing infinite traces and complete sets of their finite approximations [69], for the statement "there is an infinite trace" is an example of negative information, in contrast to the positive statement "there are arbitrarily long traces".

This discussion explains why coalgebras occur in semantics primarily in the context of logical universes, specifically in type theory [114, 23, 22].

1.2.5 Extensional Collapse

We focus our attention on monads and coalgebras as primary multi-purpose tools in the semantic toolset. We regard semantics based on these concepts as a scheme that can be instantiated in various settings and further enriched to obtain a more and more refined view. A simple example of a feature that, to our best knowledge, cannot be couched in terms of monads and coalgebras is general total recursion, unlike more specific iteration, which can be conveniently formulated in terms of monads. Intriguingly, total recursion seems to have something to do with monads, for it subsumes partiality, which is a monadic notion. Existing axiomatic approaches to defining recursion via monads derive it from an additional assumption of nontriviality of certain equalizers [29], which is close to the requirement on the category to be algebraically compact [103]. This indicates that the notion of recursion is global, i.e. it cannot be added to an existing category as a new ingredient – it can be only obtained by purposefully constructing a special kind of category supporting recursion out of the box.

Compared to recursion, the case of iteration is remarkable because not only can it be intrinsically *described* in terms of monads and coalgebras, but a specific monad for iteration can be *constructed* in a suitable universe assuming certain principles not directly tailored to iteration. In particular, this was realized recently in intensional type theory under the assumption of the *axiom of countable choice* [24, 113]. The construction obtained is a quotient of Capretta's *delay monad* [23], which is the pointwise final coalgebra $\nu\gamma.(- + \gamma)$. The resulting monad can be seen as an *extensional collapse* of the delay monad, by analogy with the corresponding concept in game semantics. In the same sense this is used by Escardó in his work on a metric model of PCF [31]. An important distinction between quotienting monads and standard constructions of extensional collapse is however that the latter are coherent homomorphic transformations of a whole category, while the former affects only a family of objects. Although quotienting a monad is a structurally more basic procedure, we feel that it is an instance of the same general phenomenon of relating intensional and extensional concepts. Formally, a connection can prospectively be made via enrichments of categories in final coalgebras. Previous work on categories of processes enriched in final coalgebras [65] seems to pave the way to such view.

In our category-independent perspective, the same procedure of collapsing the delay monad can perfectly well be carried out in the classical Zermelo–Fraenkel set theory (possibly without choice), where the construction simplifies drastically and produces the *maybe monad* $(- + 1)$, which follows from our results discussed in more detail later: in [G6] we show that the maybe monad is an *initial complete Elgot monad* in **Set** (and more generally in any hyperextensive category [7]), and in [G9] we demonstrate that if the initial complete Elgot monad exists then it is a retract of the delay monad. The general categorical construction of the extensional collapse of the delay monad is currently open.

1.2.6 Unifying Semantics

Moggi's seminal paper [79] opened a novel perspective on semantics: the notion of computation formalized by a suitable monad can be singled out as a parameter, to a large degree independent of other semantic features, arguably not captured by monads, such as recursion, polymorphism or linearity (in the sense of linear logic and stable domain theory), to name just a few. One can trace all the developments in semantics all the way back to Turing machines and Gödel's recursive functions in the light of Moggi's notion of computation, and find out that *partiality* was the first computational effect encountered. One can then argue that the second computational effect was *non-determinism*, for it was also introduced by Turing in the same seminal paper on 1936 [110], although he did it only in passing. The main subject of Turing's narrative is the notion of *a-machine* (i.e. "automatic-machine"), while what we now call non-deterministic Turing machines briefly appears under name *c-machine* (i.e. "choice-machine"). It is fair to say that while a-machines were proposed as an accurate abstraction of a physical computation processes, c-machines were meant to serve as auxiliary objects, and it seems that Turing was on the one hand aware of the fact that deterministic machines can simulate non-deterministic ones, but tried to avoid the latter notion. From today's perspective non-determinism is a classical semantic feature adopted in various settings, most notably used for delimiting complexity classes, for improving versatility and adequacy of machine models w.r.t. various language classes, and for modelling concurrency. Numerous reasons can be drawn to argue that nondeterminism is a desirable and convenient notion, but the core fact ensuring that models involving it remain in touch with some physical computational processes is that, after all, we know that in the world of Turing computability nondeterminism can be eliminated.

The example of nondeterminism is instructive, for it illustrates the fact that in a rather basic case semantics of languages for computations does not only aim to cover phenomena (not necessarily computational effects) directly corresponding to computational processes in the real world but it also aims to model mathematical abstractions, as long as they are both useful and well-justified in the sense that they agree with our general physical intuition. An example of an abstraction for which the latter criterion is debatable is *unbounded nondeterminism* [10, 26] (contrasting *bounded nondeterminism* of Turing machines).

By approaching semantics from a generic perspective, we do not necessarily aim to exclude possible instantiations without any computational meaning but we certainly aim to maintain those that are potentially useful. The strategic reason for this decision is that the strands of semantics are diverse and fragmented. On the one hand, various mathematical constructions are being reproduced in isolation, without recognizing their common nature, while formalizing the underlying connections (which is what category theory helps us to achieve) allows for transferring knowledge between instances. For illustration we refer to our work [G4] where a generalization of Kleene's regular expressions is developed, motivated by the view of various machine models as coalgebras whose transitions are side-effecting in sense of Moggi. Another case from

our experience is the work [G5] where we develop a unified notion of *weak bisimulation* in terms of monads and coalgebras, spanning, in particular, classical CCS weak bisimulation, probabilistic and resource weak bisimulations. It turns out that reasoning purely in terms of the mathematical principles behind these notions is in perfect agreement with their practical motivation, even though the latter varies considerably. On the other hand, semantic theories are always in motion: ideas from the past are revisited, recycled and, give rise to new models. Notions and constructions that did not receive much attention in the past are being rethought and utilized for the purposes of addressing todays new challenges. A good example are Manes and Arbib's partially additive categories, as mentioned in Section 1.2.1. These were originally proposed as a semantic alternative to the mainstream domain semantics, however recently they were revisited in the light of Girard's *geometry of interaction* program [37] via Haghverdi's *unique decomposition categories* [43].

Having set up the context, we proceed to present the author's contribution in detail.

2 Unifying Semantics: Worked Scenarios

2.1 Coalgebraic Machines and Process Algebra

2.1.1 Generic Observations and Coalgebraic Chomsky Hierarchy

A popular motive in semantics is to involve two functors, one of which is a monad, and a distributive law between them. An important source of examples following this pattern is identified in the seminal work by Turi and Plotkin on *abstract GSOS* [109], which identifies a *behavior functor* B, a *syntax functor* Σ and a natural transformation $\Sigma(\mathrm{Id} \times B) \to B\Sigma^\star$ giving the operational semantics induced by a set of rules in GSOS format. The crucial outcome of this approach is that every δ as above induces a *denotational semantics* $\hat{\delta} : \Sigma B^\infty \to B^\infty$ of the algebraic language generated by signature Σ over the final B-coalgebra B^∞, and the problem of *adequacy* of denotational semantics w.r.t. the operational one resolves automatically. Abstract GSOS are equivalent to distributive laws $\delta : \Sigma^\star(\mathrm{Id} \times B) \to \Sigma^\star \times B\Sigma^\star$ of the free monad $\Sigma^\star$ over the pointed functor $\mathrm{Id} \times B$ preserving the point, i.e. satisfying $\mathsf{fst}\,\delta = \Sigma^\star\,\mathsf{fst}$ [68]. More generally, one can consider arbitrary distributive laws of a monad over a functor, as well as distributive laws of functors over monads (this leads to a distinction between *Eilenberg-Moore* and *Kleisli distributive laws* [58]) in order to capture further computationally relevant examples. Numerous variations of this theme occur in the literature [46, 62, 12, 55].

A more recent take on distributive laws of the form $\delta : \mathbb{T}M \to M\mathbb{T}$ for a monad $\mathbb{T}$ and a functor M is proposed in [102, 100], where F is associated with the transition type of a system and the monad $\mathbb{T}$ with the underlying computational effect. The motivating example of this situation is the language semantics of non-deterministic automata: the

monad $\mathbb{T}$ is the powerset monad (referring to the effect of non-determinism) and M is the functor $2 \times (-)^A$ where A is the input alphabet and 2 is the two element Boolean algebra used to classify accepting and non-accepting states. By using a suitable distributive law and applying a standard characterization of distributive laws [59] one can view any coalgebra $X \to MTX$, i.e. a non-deterministic automaton, as an M-coalgebra, i.e. a deterministic automaton, in the category of $\mathbb{T}$-algebras, which also yields the language semantics via the standard finality argument. The summarized construction was dubbed the *generalized powerset construction (GPC)*, for restricted to the motivating example it precisely realizes the standard Rabin-Scott determinization [89]. By varying the monad $\mathbb{T}$, one obtains further instances of the determinization procedure, e.g. determinization of probabilistic and weighted automata seamlessly fits into this scenario.

The idea behind the analysis of the GPC in [G2] is to recognize GPC within a broader context as follows. Given two categories, $\mathbf{C}$ and $\mathbf{D}$ related by a functor $H : \mathbf{C} \to \mathbf{D}$, and endofunctors $F : \mathbf{C} \to \mathbf{C}$ and $G : \mathbf{D} \to \mathbf{D}$, one can view F-coalgebras as systems to be observed and G-coalgebras as their images under a suitable notion of observation abstracting away some aspects of the original system. This generalizes the GPC case by taking $\mathbf{D}$ to be an Eilenberg-Moore category $\mathbf{C}^{\mathbb{T}}$ of some monad $\mathbb{T}$ on $\mathbf{C}$, the free $\mathbb{T}$-algebra functor as H, $F = MT$, and an extension of M to $\mathbf{D}$ as G. The effect of applying this construction to a non-deterministic automaton in $\mathbf{C}$ is that the semantic equivalence on its image in $\mathbf{D}$ (language equivalence) is strictly coarser than the original one (bisimilarity). Intuitively, this captures the intuition that the models of systems of interest as F-coalgebras are finer than the corresponding models as G-coalgebras, due to a possible information loss through the act of observation.

A new implication of the approach in [G2] is the recognition of the fact that the GPC case corresponds to the *real-time* observations in the sense that every transition step of the original system is translated to one step of the target system; more generally, one obtains *lookahead observations*, i.e. those which can arbitrary long follow chains of F-transitions before a G-transition is triggered. Lookahead observations are thus particularly well-suited for modelling features that are typically difficult to encompass in coalgebraic frameworks, like silent transitions, unproductive divergence and weak bisimilarity. The relevant examples are presented in [G2].

A further principal idea contributed by [G2] is the connection to the theory of algebraic effects of Plotkin and Power [85, 87, 86]. Specifically, the following (axiomatization of) the *stack theory* underlying (deterministic) push-down automata was identified:

$$\begin{aligned} push_i(pop(x_1, \ldots, x_n, y)) &= x_i \\ pop(push_1(x), \ldots, push_n(x), x) &= x \\ pop(x_1, \ldots, x_n, pop(y_1, \ldots, y_n, z)) &= pop(x_1, \ldots, x_n, z) \end{aligned}$$

Here, the operations pop and $push$ should be understood as commands of a state machine over a memory stack, and the above axioms completely describe the operational equivalence over feasible stack transformers (i.e. those that can alter the stack only up to a finite statically determined depth). Formally, the above stack theory is equivalent to a *stack monad*, which is a submonad of the store monad (aka state monad) finite se-

quences of memory cells capable of storing n different symbols each, but with only those store transformations allowed that alter statically determined prefixes of these sequences. This perspective is taken further in subsequent work [G4] where a unified bialgebraic perspective of classical automata theory is presented, covering moreover some less standard models such as *weighted automata* and *valence automata*. This perspective includes the following aspects:

(a) a notion of $\mathbb{T}$-*automaton*, i.e. a generalized *Moore automaton* with computational effects carried by a monad $\mathbb{T}$;

(b) a language of fixpoint expressions, generalizing standard regular expressions over an algebraic theory underlying the corresponding monad $\mathbb{T}$;

(c) a generalized *Kleene theorem* providing a two-way translation between $\mathbb{T}$-automata w.r.t. a finitary monad $\mathbb{T}$ on **Set** and the corresponding fixpoint expressions.

Many variations of the program summarized in (a)–(c) occurred previously in the literature, starting from the classical case of non-deterministic finite state machines [63], which are $\mathcal{P}_\omega$-automata w.r.t. the finite set monad $\mathcal{P}_\omega$, standard regular expressions and the standard Kleene theorem relating them. A notable difference to the classical case is the style of fixpoint expressions: the expressions in [G4] involve a fixpoint operator μ, instead of the usual *Kleene star*, they generally allow neither for sequential composition nor for *unguarded recursion*, i.e. at least one input symbol must be consumed along any recursion loop. In the case of non-deterministic finite state machines both expression formats are mutually convertible but in general the format of μ-expression is preferable because it accurately captures the expressive power of the corresponding class of machines, e.g. if the underlying monad is the above mentioned stack monad then the corresponding expressions with unguarded recursion would address a wider class of languages (context-free) than the expressions with guarded recursion only (deterministic context-free). A legitimate way of increasing expressiveness implied by our approach is by switching to a more sophisticated underlying monad supporting more effects, e.g. to capture context-free languages, one can use a combination of the stack monad with nondeterminism by *tensoring* with the finite powerset monad $\mathcal{P}_\omega$ [34].

Previous generalization of Kleene theorem in coalgebraic setting were established by Silva and collaborators [102, 101, 100], and differ from the contribution of [G4] in that the former were focused on varying the transition functor, and restricted to specific examples of monads, while in our work, conversely, the format of the transition functor is restricted to $B \times -^A$ but the monad becomes a generic ingredient, only required to be equationally presentable.

A notable feature of the approach summarized in (a)–(c) is that one does not necessarily obtain exact textbook definitions of the expected machines, but rather their equivalent categorical analogues. For example, instead of non-deterministic push-down automata, one obtains non-deterministic automata over stacks whose one-step transitions are allowed to alter a finite prefix of the underlying stack at once (by popping and then pushing finitely many elements), while the standard non-deterministic push-down automata can pop *at most* one element of the stack at a time.

The theory of monad tensors going back to Freyd [34] has a direct impact on $\mathbb{T}$-automata and their fixpoint expressions. On the one hand, various types of store can be combined by forming tensors. For example, $\mathbb{T}$-automata over n-fold tensor products of the stack monad with itself are precisely (deterministic) machines over the n-stack memory. On the other hand, every finitary monad can be tensored with the finite powerset monad, which underlies the transition from given $\mathbb{T}$-automata to their corresponding non-deterministic modifications. One of the results of [G4] is the complete characterization of languages $\mathcal{L}_i$ recognized by non-deterministic i-stack $\mathbb{T}$-automata. Specifically, the following is shown:

- $\mathcal{L}_1$ is the class of context-free languages;
- $\mathcal{L}_2$ is properly between $\mathcal{L}_1$ and $\mathcal{L}_3$;
- for all $i > 2$, $\mathcal{L}_i$ is the class of non-deterministic linear time languages $\mathsf{NTIME}(n)$.

Another important class of machines explored in [G4] are *tape* $\mathbb{T}$*-automata*, which are the ones for which the underlying theory is the theory of the *Turing tape*. These are the expected operations for moving the machine head one position to the left, and one position to the right, as well as the operations for reading and writing the current tape symbol. The (Turing) tape monad is identified as the submonad of a suitable store monad consisting of realizable tape transformers. Like in the case of stack transformations, admissible tape transformations are those where the head can move only within a statically predetermined tape interval and alter only the cells visited. We show that (unlike the stack monad) the induced tape monad is not finitely axiomatizable. Roughly, the reason for this is that a complete axiomatization must ensure that writing a cell does not impact any other cell, no matter how far it is from the original one. Formally, this amounts to the following axiom scheme:

$$wr_i(mv_k(wr_j(mv_{-k}(x)))) = mv_k(wr_j(mv_{-k}(wr_i(x))))$$

where mv_i stands for moving the head i cells to the right ($-i$ cells to the left if i negative) and wr_i stands for writing the symbol indexed by i at the current head position.

The described approach is genuinely coalgebraic, that is, every step of a $\mathbb{T}$-automaton consumes an input alphabet symbol. Such machines are known as *real-time* [20]. The above characterization of languages $\mathcal{L}_i$ recognized by non-deterministic multi-stack machines indicates that the restriction to real-time machines implies a principal bound on the expressiveness of the languages recognized by computationally feasible $\mathbb{T}$-automata, and this bound is the class of non-deterministic linear time languages $\mathsf{NTIME}(n)$. In order to overcome this constraint, an *observational semantics* for a broad class of $\mathbb{T}$-automata is developed in [G4], i.e. a semantics allowing for ε-transitions that do not contribute to the language being recognized. This resulted in a proof that tape $\mathbb{T}$-automata with silent transitions recognize precisely the *recursively enumerable languages*.

2.1.2 Coalgebraic Weak Bisimulation

Weak bisimulation and *weak bisimilarity* are central concepts in Milner's CCS process algebra [76] built on top of the auxiliary notion of *strong bisimulation*. Coalgebraic modelling has accommodated many ideas and notions from process algebra, in particular, strong bisimulation, albeit in somewhat varied forms [106], but generally in a rather satisfactory manner, especially for weak pullback preserving functors, for which the relational form of bisimulation coincides with the functional one, called *observational equivalence*. More specifically, here we stick to the notion of strong bisimulation for F-coalgebras as *kernel pairs* of F-coalgebra morphisms, also called *kernel bisimulations*.

In [G5] we tackle the problem of defining a generic notion of weak bisimulation for coalgebras. The starting point is Milner's "double arrow construction" [76] for reducing weak bisimulation to strong bisimulation in the following way. Given a labeled transition system $(X, (\rightarrow_a \subseteq X \times X)_{a \in A})$, Milner constructs a labeled transition system $(X, (\Rightarrow_a \subseteq X \times X)_{a \in A})$ for which a binary relation on X is a strong bisimulation iff it is a weak bisimulation for the original system. By comparison to the previous work by Baier and Herrmanns [11] on weak bisimulation for *fully probabilistic systems* (i.e. labeled transition systems whose transitions are additionally weighted by probabilities) we conclude that Milner's construction cannot serve as a generic pattern for coalgebraic weak bisimulation. The insight provided by the probabilistic case is that weak point-to-point transitions no longer form a probability distribution, and therefore cannot be used as strong probabilistic transitions of a modified system. For example, in a system where $x \rightarrow_a y$ with probability 0.5 and $x \rightarrow_\tau x$ with probability 0.5 we would obtain both $x \Rightarrow_a y$ with probability 1 and $x \Rightarrow_\tau x$ with probability 1, which is explained by the fact that the events of staying at x and moving to y are not independent (we can move from x to y after staying at x for a while). The solution due to Baier and Hermanns [11] is to draw on *point-to-set* transitions instead of *point-to-point* transitions, and therefore introduce *total probabilities* $P(x, \Lambda, S)$ where x is a state, X is a set of states and Λ is a set of traces from x to the states in Λ, so that $P(x, \Lambda, S)$ is the probability of reaching S from x via Λ. The main technical difficulty confronted within [11] is that of calculating total probabilities by involving nontrivial measure-theoretic arguments. The approach we present in [G5] abstracts away from these specific constructions and from total probabilities as such, in favor of a generic solution via recursive equations over monads.

More specifically, we consider the general case of coalgebras of the form $X \rightarrow T(A \times X)$ where $\mathbb{T}$ is a monad whose Kleisli category is enriched over the category of pointed complete partial orders (a similar assumption occurs in previous work on coalgebraic trace semantics [46]) and A is a set of actions, which should be thought of as labels of the corresponding generalized labeled transition system. The enrichment assumption is needed for calculating solutions of recursive equations w.r.t. $\mathbb{T}$ via least fixpoints, and our main contribution in [G5] is to develop a theory of coalgebraic weak bisimulation based on these fixpoints. We show that various concepts of weak bisimulation in the literature become unified with this pattern. Perhaps most strikingly this is illustrated by comparison probabilistic bisimulation as discussed earlier and resource

bisimulation [28]. Although mathematically, *probabilities* and the *multiplicities* used in resource semantics are seemingly close in that the first involve the complete ordered semiring of real numbers, while the second involve the complete ordered semiring $N \cup \{\infty\}$ of extended natural numbers, the units of these semirings play rather different roles w.r.t. to ordering: on the one hand, 1 is the maximal probability, but on the other hand, it is the smallest multiplicity. This distinction contributes quite significantly to the original diverse formulations of weak bisimulation for both types of systems. Nevertheless, our general formula for weak bisimulation covers both cases seamlessly.

Besides the monad $\mathbb{T}$ capturing the transition effects of coalgebras in question, our framework receives an additional parameter, a binary continuous operation $\oplus$ for $\mathbb{T}$. Intuitively, the role of $\oplus$ is to join computational effects triggered by separate transition paths, e.g. in the probabilistic case this operation calculates the pointwise maximum of two probability distributions, and it is under the hood of the above-mentioned total probabilities $P(x, \Lambda, S)$, which are the probabilities of reaching S from x via Λ, i.e. the *maximal* probabilities of reaching *some* element of S from x via Λ. The crucial fact about $\oplus$ is that the construction of weak bisimulation simplifies drastically when $\oplus$ is an algebraic operation of T in the sense of Plotkin and Power [85, 87, 86], for as we prove, whenever this is the case, weak bisimulation can be constructed as strong bisimulation via a generic version of Milner's double arrow construction. The sheds new light on previous developments by Brengos [21], who successfully extended the double arrow construction, e.g., to *Segala systems*, i.e. systems combining probability and non-determinism (for which $\oplus$ is algebraic non-deterministic choice), but not to fully probabilistic systems as above (for which $\oplus$ is non-algebraic pointise maximum of distribution), and contributes to clarifying the role of algebraicity in coalgebraic modelling.

For non-algebraic operations $\oplus$ (e.g. for fully probabilistic systems) we provide a generic construction for extending the original monad T to a monad $\widehat{T}$ with algebraic $\oplus$ in such a way that an original system is transformed to a system over $\widehat{T}$ and the relevant notions of weak bisimulation agree. In summary, the double arrow construction can be applied also to weak bisimulation over non-algebraic $\oplus$, but after a suitable extension of the system type.

2.2 Monad-Based Program Logics

2.2.1 Hoare Logic for Order-Enriched Effects

In [G8] we present a further development of previous work [95, 94] on monad-based Hoare logic. The main idea we put forward is that a general formulation of a sound and relatively complete calculus for verifying partial assertions can be achieved under two chief assumptions: programs are realized as Kleisli morphisms of a strong monad $\mathbb{T}$ addressing a computational effect of interest, and the Kleisli category of $\mathbb{T}$ is suitably enriched over the category of bounded complete directed partial orders (bdcpo's) and continuous maps. The obvious goal behind the enrichment assumption is to ensure that

while loops can be introduced to the language and interpreted using the standard recipe from domain theory [5]. In fact, an enrichment over directed complete partial orders, or even over ω-complete partial orders would already suit this goal [105], however, by additionally assuming bounded completeness we achieve that not only programs, but also a logic of *state assertions* can be defined. Note that bounded completeness and directed completeness are two of the standard condition in the definition of Scott domains [5]. The third condition is *algebraicity*, or more generally, *continuity*, which we do not assume, because it does not play a role in our developments.

Motivated by [95, 94], we postulate a submonad $\mathbb{P}$ of $\mathbb{T}$ consisting of well-behaved programs treated as state assertions. Specifically, $\mathbb{P}$ is required to be *innocent* i.e. to satisfy the following laws.

- every $f : X \to PY$ and $g : X \to PZ$ commute:

  $$\mathsf{do}\, y \leftarrow f(x); z \leftarrow g(x);\ \mathsf{ret}\langle y, z\rangle = \mathsf{do}\, z \leftarrow g(x); y \leftarrow f(x);\ \mathsf{ret}\langle y, z\rangle,$$

  i.e. $\mathbb{P}$ is a *commutative monad*;
- every $f : X \to PY$ is *copyable*:

  $$\mathsf{do}\, y \leftarrow f(x); z \leftarrow f(x);\ \mathsf{ret}\langle y, z\rangle = \mathsf{do}\, y \leftarrow f(x);\ \mathsf{ret}\langle y, y\rangle;$$
- every $f : X \to PY$ is *weakly discardable*: $\mathsf{do}\, y \leftarrow f(x);\ \mathsf{ret}\star \sqsubseteq \mathsf{ret}\star$.

That is, well-behaved computations, i.e. those that commute with each other, can be run several times with the same effect, and have various degrees of divergence as the only computational effect, isolated by voiding the return value.

These conditions, together with bounded completeness, ensure that the object $P1$ can serve as a *truth value object* for state assertions, formally, the functor $\mathsf{Hom}(-, P1)$ factors through the category of frames, and conjunction of assertions is induced by Kleisli composition, i.e. in order to obtain the conjunction of two programs, we simply run them one after another. In view of the latter, the above conditions identifying innocent monads function as follows: commutativity ensures that conjunction is commutative, copyability that it is idempotent, and weak discardability that $\mathsf{ret}\star$ is the totally true assertion. The described approach allows us to cover the following classes of example categories $\mathbf{C}$ and strong monads $\mathbb{T}$.

- $\mathbf{C}$ is the category $\mathbf{Set}$ of sets and functions. In this case, one has largely the standard examples of effects as long as they account for non-termination. E.g. the total store monad $TX = S \to (S \times X)$ is not order-enriched, for it does not accommodate the bottom element, but the partial store monad $TX = S \rightharpoonup (S \times X)$ is. The corresponding innocent submonad is then the partial reader monad $TX = S \rightharpoonup X$, and the induced truth value object is the set $\mathcal{P}(S)$ of all store predicates.
- $\mathbf{C}$ is a category of predomains (e.g. bottomless bounded-complete dcpo's); then a monad on $\mathbf{C}$ is order-enriched if computational types accommodate bottom elements and binding respects existing finite joins on the left. This allows for accommodating various examples from domain theory, with the most simple one being the lifting monad $(-)_\perp$ as $\mathbb{T}$. In this case, $\mathbb{T}$ itself is innocent and $T1$ is the *Sierpiński space*.

- $\mathbf{C}$ is a category of presheaves such as $\mathbf{C} = [I, \mathbf{Set}]$ where I is the category of finite sets and injections; one monad of interest here is the *local store monad* (aka *local state monad*) [87], which we modify to allow for partiality (and hence order-enrichment):

$$(TX)n = V^n \rightharpoonup \int^{m \in n/I} V^m \times Xm$$

where the integral sign denotes a categorical *coend*, which is a special form of colimit. The intuitive meaning of the integral here is a sum over $m \supseteq n$ of pairs representing the new state and the output parametrized by the set m of allocated locations; each component of this sum corresponds to allocating $m - n$ new memory cells. The corresponding natural innocent submonad is the partial local reader monad: $(PX)n = V^n \rightharpoonup \int^{m \in n/I} Xm$, which induces a truth value object consisting of all predicates on the store configurations V^n.

In the terminology of Section 1.2.2, the category $\mathbf{C}$ figuring in the first and the third class of examples, can be seen as a logical universe, but not in the second class, for the induced truth value object $P1$ in that case is indeed only an external frame, and not a Heyting algebra, because existence of *Heyting implication*, which is antitone in the first argument, would contradict continuity of morphisms in $\mathbf{C}$.

In the development a generic Hoare calculus over a pair $(\mathbb{T}, \mathbb{P})$ consisting of an order-enriched monad $\mathbb{T}$ and its innocent submonad $\mathbb{P}$, we start from the celebrated relative completeness result by Cook [27], which is essentially a reduction of the problem of checking a Hoare triple to the problem of checking a formula of the assertion language. Cook's proof is based a technical trick for eliminating fixpoint assertions using Gödel's β-function, which is avalaible in first order arithmetic serving as the assertion language in Cook's model. It has later been observed that the full strength of the first order arithmetic is not necessary for the proof – instead one can make do with a weaker system containing fixpoints as first class citizens [18]. The latter approach naturally arises in our case, for the truth value object $P1$, being an external frame, automatically supports greatest fixpoints as needed for the completeness proof (indeed also the least fixpoints). Just as in Cook's proof, relative completeness is achieved by expressing weakest (liberal) preconditions inductively over the program structure, e.g. the relevant clause of while-loops is handled as follows:

$$\mathsf{wp}(x \leftarrow (\mathsf{while}\, b;\, x \leftarrow p), \psi) = \big(\nu X.\, \lambda x.\ \mathsf{if}\, b\, \mathsf{then}\, \mathsf{wp}(x \leftarrow p, X(x))\, \mathsf{else}\, \psi\big)(x)$$

where $\mathsf{wp}(x \leftarrow p, \phi)$ stands for the weakest precondition for program p w.r.t. the post-condition ϕ and $\nu X.\, \phi$ calculates the greatest fixpoint of $(X : A \to P1) \mapsto (\phi : A \to P1)$ by the Knaster-Tarski theorem. By successively applying further inductive clauses, we eventually reduce to the level of atomic programs, for which weakest preconditions are assumed to be expressible. Concretely, an atomic program represents by a morphism of the form $A \to TB$ from a postulated effect signature. In the terminology of Plotkin and Power [87], these are *generic effects* of the monad. For example, the effect of coin *toss*ing $toss : 1 \to T2$ underlies nondeterminism: by calling it we obtain a nondeterministic

value from $2 = 1 + 1$, and so the nondeterministic binary choice operation is expressible as $p + q = \mathsf{do}\, x \leftarrow \mathit{toss};\ \mathsf{case}\, x\, \mathsf{of}\, \mathsf{inl}\, \star \mapsto p;\ \mathsf{inr}\, \star \mapsto q$. The corresponding defining clause for the weakest precondition operator is

$$\mathsf{wp}(x \leftarrow \mathit{toss}, \psi) = \psi[\mathsf{inl}\, \star / x] \wedge \psi[\mathsf{inr}\, \star / x]$$

expressing the fact that ψ is satisfied after executing a nondeterministic program if it is satisfied after executing any of its deterministic branches. A more familiar example is the following one:

$$\mathsf{wp}(- \leftarrow \mathit{write}_i(v), \psi) = \psi[\mathsf{ret}\, v / \mathit{read}_i]$$

where $\mathit{write}_i : V \to T1$ updates the store location indexed by i with v and $\mathit{read}_i : 1 \to PV$ returns the content of the location indexed by i.

Our proof of the relative completeness result relies on a property of the weakest precondition operator that we call *sequential compatibility*:

$$\mathsf{wp}(x \leftarrow (\mathsf{do}\, y \leftarrow p;\ q), \psi) \iff \mathsf{wp}(y \leftarrow p, \mathsf{wp}(x \leftarrow q, \psi)).$$

In subsequent work [44], Hasuo proposes an alternative approach to a generic categorical Hoare calculus resting on a different collection of postulates. Specifically, instead of deriving $P1$ from $\mathbb{T}$, he introduces a truth value object Ω axiomatically, requiring it to be a $\mathbb{T}$-algebra, while the submonad $\mathbb{P}$ does not arise at all in his approach. This, in fact, yields a slicker reformulation of our assumptions, in particular our weakest precondition operators are in one to one correspondence with $\mathbb{T}$-algebra structures on $P1$, and moreover the above sequential comparability condition corresponds precisely to an axiom of $\mathbb{T}$-algebras.

In [G10] we extend the results of [G8] to the case of monads with exception throwing/catching facilities. It is known from previous work [93] that a monad satisfying a natural set of axioms for catching/throwing exceptions must have the form $T_E = T(- + E)$ where E is an object of exceptions and $\mathbb{T}$ is monad capturing further possible effects, e.g. $T = \mathrm{Id}$ if there are none. The approach we develop in [G10] starts from the assumption that $\mathbb{T}$ is an order-enriched monad (so, e.g. T cannot be the identity – at least it must cope with partiality), and proving that so is the transformed monad $\mathbb{T}(- + E)$. This implies that the previous completeness result does instantiate to the case at hand. However, the novelty is that the syntax of the program language is extended not only by and algebraic operations $\mathit{throw} : E \to T_E 1$, but also by a non-algebraic operation $\mathit{catch} : T_E X \to T_E(- + E)$. This considerably impacts the whole calculus of Hoare triples, which become *Hoare quadruples* [93, 56]: these consists of a precondition ϕ, a program p, a postcondition ψ, and an *abnormal postcondition* ϵ, in summary we obtain constructs of the form $\{\phi\}\ x \leftarrow p\ \{\psi \mid \epsilon\}$ where the postcondition ψ depends on x and ϵ corresponds to a morphism $E \to P1$, that is ϵ addresses the situation when an exception is raised, and therefore no return value is available. The weakest precondition construct is thus modified to cover the abnormal postcondition alongside the normal postcondition. The key additional inductive clause in the definition of the weakest precondition operator is the one handling the catch-operator and is as follows:

$$\mathsf{wp}(y \leftarrow \mathsf{catch}\, p, \psi \mid \epsilon) = \mathsf{wp}(x \leftarrow p, \psi[\mathsf{inl}\, x / y] \mid \lambda e.\psi[\mathsf{inr}\, e / y]).$$

That is, if an exception is caught then the declared abnormal termination postcondition does not apply, instead the normal postcondition is used twice. Modulo these distinctions, the results of [G8] are reestablished in the extended setting.

2.2.2 Calculi for Asynchronous Side-Effecting Processes

The use of monads as a semantic base for side-effecting programs in the previous section fits the extensional paradigm (cf. Section 1.2.3) in which a monad is used to encapsulate a computational process not providing any interface to the intermediate stages, but only possibly delivering a final result. The language of programs we used is thus essentially Moggi's *computational metalanguage* [79], i.e. a generic imperative language covering two essential features: sequential composition of programs via *binding*, and the return operator. As shown by Moggi, the computational metalanguage is sound and complete over strong monads on categories with finite products. Categorically speaking, the computational metalanguage is the *internal language* of strong monads. By selecting a monad from a more specific class one can respectively extend the metalanguage with more features, notably with *algebraic operations* (e.g. reading/writing a store, non-determinism), *effect handlers* [88] (e.g. exception handling), recursion/iteration operators [29, 42, 40]. In [G7] we introduce an extension of Moggi's computational metalanguage for programming communicating and concurrent processes in the style of Milner [76], parametrized by a monad carrying the underlying computational effect of atomic process steps. This equips the language of programs with intensional features (cf. Section 1.2.4) and thus effectively turns it is into a *language of processes*. More specifically, we speak about *asynchronous side-effecting processes* because the principle of synchronizing events (e.g. as in Milner's CCS) is not an inherent monadic feature: a handshake synchronization amounts to executing an action of one process and subsequently executing a dual action of another process. The CCS semantics abstracts away from a possible time lag between these events, but in general, it cannot be neglected, e.g. our monad-based abstraction is equally suitable for shared variable concurrency, for which the non-commutativity of read-write events is a key feature.

The key semantic gadget we add to the standard axiomatic setup is the *coalgebraic resumption monad transformer*

$$T_\nu = \nu\gamma.\, T(- + \gamma).$$

The idea of using resumptions in semantics goes back to Hennessy and Plotkin [47] who obtained them as solutions of a certain recursive domain equation involving stores, non-termination and non-determinism (in the form of the Plotkin powerdomain). In the context of domain theory, one looks for initial solutions of domain equations, because they correspond to least fixpoints [105], and the latter are justified by interpreting the underlying partial order as the order of information increase. Since domain categories are typically *algebraically compact*, the domains of resumptions calculated as initial and as final solutions coincide, but in other categories, e.g. in **Set**, they might be essentially different, and the choice of the coalgebraic resumption monad transformer is preferable over the algebraic one, for the former captures possibly infinite behavior as required in

process algebra in a category-independent way.

Moggi's computational metalanguage is thus extended in [G7] to a *concurrent computational metalanguage* by adding coproducts, resumption types $T_\nu X$ with the corresponding term formation rules, and the non-deterministic choice operator for computations over $\mathbb{T}$. The most important new ingredient is, of course, the resumption types and the associated operations, which are

- $\mathsf{out} : T_\nu X \to T(X + T_\nu X)$, for unfolding $T_\nu X$ to a computation returning either a value of type X or a value of type $T_\nu X$;
- $\mathsf{init}\, x := (-)\ \mathsf{unfold}\ (-)$ producing a value of type $T_\nu X$ from its arguments of type Y and $Y \to T_\nu(Y + X)$ correspondingly by *coiterating* the latter.

The Moggi's sound and complete equational calculus [79] is therefore completed essentially by axiomatizing the fact that each $T_\nu X$ is a final coalgebra whose structure map is out and for which the above unfold construct calculates the unique final morphism from any $T(X + -)$-coalgebra to $T_\nu X$. Some effort is required to extend Moggi's soundness result, because the axiomatized finality principle is parametric in term contexts, and does not exactly correspond to the unparameterized one, semantically characterizing final coalgebras.

The soundness and completeness results allow us to work entirely within the equational calculus of the concurrent computational metalanguage, which we successively show to be sufficient to express key concurrent primitives. As a preparatory step we show that the postulated principle of defining by iteration implies a much richer and versatile definitional principle by *mutual corecursion*. Intuitively, this guarantees that any finite system of equations recursively specifying functions $f_i : A_i \to T_\nu B_i$ has a unique solution as long the right hands sides of equations satisfy a *guardedness condition*, ensuring that every recursive call properly grows the defined part of the function being defined. We revisit the problem of finding solutions of guarded definitions from a syntax-independent perspective in Section 2.3.

Using mutual corecursion, we extend the concurrent metalanguage with the following features:

- the return and binding operators for T_ν making it into a strong monad (the fact that T_ν is a monad was already known from [111]);
- conditional branching;
- conditional looping;
- exception throwing and catching;
- process interleaving (with the corresponding mutual corecursion definition, inspired by the ACP approach of defining the parallel composition and the *left merge* operator by mutual recursion [15]).

The remaining part of [G7] is devoted to developing a program verification framework on top of the concurrent computational metalanguage. This is achieved by introducing a sound calculus of *resumptive Hoare tuples* extending previous work in [95, 94]. This

differs from the perspective presented in Section 2.2.1 in that no assumptions of order-enrichment for $\mathbb{T}$ are made, and instead of an innocent submonad of $\mathbb{T}$ we use a *pure submonad* $\mathbb{P}$, which is subject to the same set of axioms, except that in weak discardablity, inequality is replaced by equality as follows:

$$\mathsf{do}\, y \leftarrow f(x);\ \mathsf{ret}\star = \mathsf{ret}\star,$$

and the obtained axiom is called *discardability*. As the truth value object we use $P2$ instead of $P1$, and thus obtain an internal Boolean algebra structure on $P2$, induced by $2 = 1 + 1$, instead of a Heyting algebra structure on $P1$ as discussed in Section 2.2.1. Morally, these changes amount to viewing the monad $\mathbb{T}$ as a monad of decidable programs, which is in agreement with its use for capturing atomic computational steps, in contrast to $\mathbb{T}_\nu$ capturing entire computation processes. As a result, we inherit from [94] a sound Hoare calculus for programs over $\mathbb{T}$. A calculus of *resumptive Hoare tuples* is defined on top of the latter, to address processes, i.e. programs over $\mathbb{T}_\nu$. A resumptive Hoare tuple is thus a construct of the form $\{\{\phi\}\}\ x \leftarrow p\ \{\{\psi \mid \xi\}\}$ where p corresponds to a morphism of type $X \to T_\nu Y$, equivalently seen as having type $X \to T(Y + T_\nu Y)$ via the final coalgebra structure on $T_\nu Y$. The postconditions ψ and ξ thus address the corresponding components of the coproduct $Y + T_\nu Y$ (note that we change the order of the postconditions in the tuples, compared to [G7], in order to maintain consistency of the present text). The postcondition ψ is interpreted as a morphism of type $Y \to P2$ and specifies values obtained after performing one step of the process p (not necessarily the first one), provided the precondition ϕ is satisfied directly before this step. The postcondition ξ address the resumptive part of this scenario: it must be satisfied after an atomic step whenever ϕ is satisfied directly before and the outcome of the step is a resumption of type $T_\nu Y$. The postcondition ξ does not depend on the resumption itself, i.e. it only handles the fact of its presence. In summary, resumptive Hoare tuples are similar to Hoare tuples for monads with exceptions from [G10], as discussed in Section 2.2.1.

We prove soundness of a calculus of resumptive Hoare tuples designed analogously to the case of $\mathbb{T}$ and illustrate its use by a toy example by mimicking an approach to verification of safety properties by Manna and Pnueli [74] based on program invariants and program step annotations.

2.3 Guarded and Unguarded Iteration

2.3.1 Unguarded Recursion via Complete Elgot Monads

One of the earliest motivating examples for coalgebraic modelling in semantics is process algebra [92]: a (non-determinstic) process can be seen as a coalgebra whose transitions are labeled by process actions. Note that in process algebra, infinite behavior is considered desirable: the fact that a process can evolve forever ensures that it remains responsive or in any way keeps producing information about its run. This idea crucially factors into the general coalgebraic methodology of solving systems of recursive equations by *corecursion*: under the assumption that the system is *guarded*, i.e. the recursive calls are preceded by at least one action call, the system has a unique solution

```
Class ElgotMonad (T: Obj → Obj) :=
{
     (* Monad structure: unit *)
     η : forall x, x ⇝ T x;

     (* Monad structure: Kleisli lifting *)
     lifting : forall x y (f: x ⇝ T y), T x ⇝ T y
         where "f ⋆" := (lifting _ _ f);

     map x y (f: x ⇝ y) : T x ⇝ T y := ((η _) ∘ f)⋆;

     (* Kleisli triple laws *)
     kl1 : forall x, (η x)⋆ = id;
     kl2 : forall x y (f: x ⇝ T y), f⋆ ∘ (η _) = f;
     kl3 : forall x y z (f : y ⇝ T z) (g : x ⇝ T y), (f⋆ ∘ g)⋆ = f⋆ ∘ g⋆;

     (* Elgot iteration  *)
     iter : forall x y (f: x ⇝ T (y ⊕ x)), x ⇝ T y
         where "f †" := (iter _ _ f);

     unfolding : forall x y (f: x ⇝ T (y ⊕ x)),
         [(η _) , f †]⋆ ∘ f = f †;

     naturality : forall x y z (f: x ⇝ T (y ⊕ x)) (g: y ⇝ T z),
         g⋆ ∘ f † = ([ map _ _ inl ∘ g  , (η _) ∘ inr ]⋆ ∘ f)†;

     dinaturality : forall x y z (g : x ⇝ T (y ⊕ z)) (h: z ⇝ T (y ⊕ x)),
         ([(η _) ∘ inl, h]⋆ ∘ g)† = [(η _) , ([(η _) ∘ inl, g]⋆ ∘ h)† ]⋆ ∘ g;

     codiagonal : forall x y (g: x ⇝ T (y ⊕ x ⊕ x)),
         ((map _ _ [id, inr]) ∘ g)† = g† †;

     uniformity : forall x y z (f: x ⇝ T (y ⊕ x)) (g: z ⇝ T (y ⊕ z)) (h: z ⇝ x),
         f ∘ h = (map _ _ (id ⊕⊕ h)) ∘ g →  f † ∘ h = g†;
}.
```

Figure 2.1: Elgot monad type class in Coq.

w.r.t. the semantics in a final coalgebra. Guardedness is typically ensured by referring to a concrete language, enforcing some sufficient (but maybe not necessary) syntactic constraints to ensure it. Constraints of this kind are instrumental in our work on side-effecting process calculi discussed in Section 2.2.2. A more semantic approach is to move from individual coalgebras to monads generated by coalgebras and to define guardedness via the structure of the underlying functor. An example of this approach is given by the notion of *completely iterative monad* [6], which are monads admitting unique *solutions* of guarded morphisms $f : X \to T(Y + X)$ where a solution is a morphism $f^\dagger : X \to TY$ such that $[\eta, f^\dagger]^\star f = f^\dagger$. The latter equation describes $f^\dagger$ as an *iteration* of f, and indeed, in the present context the terms "solution" and "iteration" are interchangeable. The *free completely iterative monad* induced by an endofunctor Σ is determined by a universal property in the standard sense, and it is shown in [6] to have the explicit form $\nu\gamma.(- + \Sigma\gamma)$

Crudely speaking, guardedness is used to separate the good morphisms from the

bad ones, and it can be argued that the notion in [6] is too restrictive, e.g. it would not cover guarded recursive equation systems in process algebra like

$$x = y + a.x$$

In [111] Uustalu introduced a more general definition of guardedness for *parametrized monads*, covering this and other examples. One outcome of his work is that the above monad $\nu\gamma.(- + \Sigma\gamma)$ becomes generalized to $\nu\gamma.- \# \gamma$ where $\#$ is a given parametrized monad, i.e. a bifunctor, which is a monad in the first argument and an endofunctor in the second. On the one hand this covers the above case with $X \# Y = X + \Sigma Y$, but on the other hand it moreover covers process-algebra-like examples with $X \# Y = T(X + \Sigma Y)$ for an arbitrary monad $\mathbb{T}$. Both the notion of guardedness and complete iterativity are generalized accordingly.

Aside from the distinctions in formulations of guardedness for monads, we pose a more radical question in [G6]: how to extend the theory of monad-based definitions to the unguarded case in a principled way, maintaining agreement with the previous results on guarded solutions? In order to answer this question, we draw on *complete Elgot monads* [9], which are a class of monads with an axiomatically defined iteration of the same profile as above, but with no guardedness restrictions. The iteration operator is thus total, and is required to satisfy the standard laws of iteration according to Bloom and Ésik [19], except that we additionally require the *uniformity principle*, called the *functorial dagger implication law* [19]. A large source of complete Elgot monads is domain theory: any monad on a suitable domain category supports iteration via least fixpoints and is thus completely Elgot. More generally, monads whose Kleisli category is enriched over pointed complete partial orders are complete Elgot, for the iteration via least fixpoint can be calculated in just the same way. Therefore, e.g. the powerset monad $\mathcal{P}$ is completely Elgot on **Set**.

The central result we establish in [G6] is that given a complete Elgot monad $\mathbb{T}$ and any endofunctor Σ, the functor $T_\Sigma = \nu\gamma.T(- + \Sigma\gamma)$ extends to a complete Elgot monad, and the latter is freely generated on $\mathbb{T}$ by Σ in the category of complete Elgot monads (in fact, in [G6] we only treated the case $\Sigma = a \times (-)^b$ but that restriction proved to be irrelevant [41]). The main technical difficulty we encountered in establishing this result is the proof that T_Σ yields a complete Elgot monad, i.e. an iteration operator for it can be defined and indeed satisfies the laws of iteration. The iteration operator is defined in two stages:

1. Given $f : X \to T_\Sigma(Y + X)$, we switch to the isomorphic morphism

$$\hat{f} : X \to T\big(X + Y + T_\Sigma(Y + X)\big)$$

and solve it by calling the iteration of $\mathbb{T}$ to obtain

$$\hat{f}^\dagger : X \to T\big(Y + \Sigma T_\Sigma(Y + X)\big).$$

2. The latter morphism can be converted back to the original type $X \to T_\Sigma(Y + X)$, but now this morphism is guarded, and therefore its unique solutioncan be found using the previous approach [111].

In order to define iteration on $\mathbb{T}_\Sigma$, we thus combine the axiomatically given iteration on $\mathbb{T}$ and guarded iteration, i.e. *corecursion*. The verification of the laws of iteration for $\mathbb{T}_\Sigma$ turned out to be a rather nontrivial task – the proof was therefore machine checked using the Coq proof assistant by a *deep encoding* of the necessary fragment of category theory. This formalization is available at `https://git8.cs.fau.de/redmine/projects/corque`. The (slightly simplified) formalization of complete Elgot monads as a Coq type class is shown in Fig. 2.1.

A salient feature of the constructed iteration operator on $\mathbb{T}_\Sigma$ is that even when the iteration operator for $\mathbb{T}$ is calculated as the least fixpoint, the resulting iteration operator on $\mathbb{T}_\Sigma$ is in general neither the least nor the unique solution of the corresponding fixpoint identity. This can be easily seen by instantiating $\mathbb{T}$ with the countable powerset monad $\mathcal{P}_{\omega_1}$ (finite powerset does not support iteration, and for unbounded powerset the final coalgebra T_Σ does not exist) and Σ with $(A \times -)$. A set $T_\Sigma X = \nu\gamma.\mathcal{P}_{\omega_1}(X + A \times \gamma)$ is a natural semantic domain for nondeterministic possibly infinite processes returning eventual results of type X and performing atomic actions from A. On the one hand, solutions of systems of process definitions are not unique, e.g. because the trivial equation $x = x$ has arbitrary solutions, and on the other hand they are also not the least solutions, for processes cannot be ordered – the coarsest precongruence generating process bisimilarity is *similarity*, which is not a partial order.

A further issue we investigate in [G6] is the one of existence of the *initial complete Elgot monad* $\mathbb{L}$. This is related to the previous discussion in that the transformer $\mathbb{T} \mapsto \mathbb{T}_\Sigma$ only gives a way to obtain a complete Elgot monad from an existing one, and so it is natural to ask if there is a canonical starting point to this process. A positive result is obtained under the assumption of *hyperextensivity* [7] of the underlying category, which means that it supports countable coproducts and those are disjoint and stable under pullbacks, and coproduct injections are closed under countable unions. As we indicated in Section 1.2.1, countable products are not compatible with some existing models of interest including some rather rich variants of set theory, and yet there are many interesting examples of hyperextensive categories. For such categories we show that the maybe monad $(- + 1)$ is the initial complete Elgot monad. Remarkably, this holds true in the category of directed complete partial orders and continuous maps, which is hyperextensive. In this category, the maybe monad is the initial complete Elgot monad, despite the existence of a seemingly more suitable candidate, the *lifting monad*. As a consequence, there is a unique morphism from the maybe monad to the lifting monad, but no morphism in the opposite direction – this morally corresponds to the fact that nontermination (captured by lifting monad) is undecidable.

2.3.2 Unifying Guarded and Unguarded Iteration

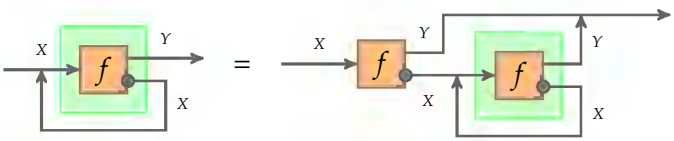
As we indicated in Section 2.3.1, guarded and unguarded monad-based iteration are intimately related: $T_\Sigma = \nu\gamma.T(- + \Sigma\gamma)$ is a monad supporting guarded iteration whenever $\mathbb{T}$ is a monad, but if $\mathbb{T}$ itself supports unguarded iteration then this canonically extends to the transformed monad, and the iteration operator of the latter calls both on the guarded iteration operator, which exists unconditionally, and on the unguarded

iteration operator of $\mathbb{T}$, which can be defined e.g. via least fixpoints. In [G9], we further study the connection between guarded and unguarded iteration, in particular by presenting a notion that unifies both cases in an axiomatic manner. Recall that iteration of complete Elgot monads is already given axiomatically – it is just an operator satisfying a certain collection of axioms. Here, we additionally axiomatize the rules determining when this operator is applicable. One corner case is that it is alway applicable, i.e. the iteration operator is *total*, as in the case of complete Elgot monads. Otherwise, the iteration operator is *partial*, and is defined precisely on the morphisms that are provably guarded w.r.t. an axiomatically given notion of guardedness. The core idea we put forward can be contrasted to the one behind Malherbe et.al.'s *partially traced categories* [71] where the trace operator (of which iteration is a special case) is directly introduced as a family of partial maps over suitable hom-sets, and axioms of trace are given in term of (directed) *Kleene equality*, that is, validity of such an equality depends on definiteness of its sides, which is not known a priori. Our approach makes more stringent assumptions but offers more compositionality in that guardedness propagates across all programs, and not and not only those that can be iterated.

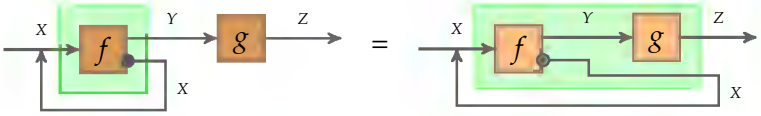
Given a morphism of the form $f : X \to TY$ we see it, as usual, as a side-effecting program with input of type X and output of type Y. Our notion of guardedness governs the output, and thus we aim to distinguish a portion of the output in which the program is guarded. In order to make this precise, we involve *coproduct summands* $\sigma : Y' \hookrightarrow Y$ to judge if f is guarded w.r.t. σ or not. The notion of guardedness in question is then induced by a simple rule-based system. For example, one rule states that $(T\,\mathsf{inl})\,f : X \to T(Y + Z)$ is guarded w.r.t. $\mathsf{inr} : Z \to Y + Z$ for any $f : X \to TY$. We then introduce *guarded pre-iterative monads* as those monads for which an iteration operator exists, sending any $f : X \to T(Y + X)$ guarded in $\sigma + \mathsf{id}$ to $f^\dagger : X \to TY$ and satisfying the fixpoint law $f^\dagger = [\eta, f^\dagger]^\star f$. If moreover $f^\dagger$ is a unique solution of this equation for every f as above, we call the monad *guarde iterative*. In any case, it follows from the axioms of guardedness that the resulting morphism $f^\dagger$ is guarded in σ. Another immediate consequence of our axioms is that besides the *greatest* notion of guardedness, identifying all morphisms as guarded, there is the *least* one: roughly it is induced by the above rule identifying a morphism $f : X \to T(Y + Z)$ as guarded w.r.t. $\mathsf{inr} : Z \to Y + Z$ if the whole program flow runs through Y.

We summarize the axioms of iteration adapted from complete Elgot monads (Section 2.3.1) in graphical form in Figure 2.2. This style of presentation goes back to Elgot (e.g. [30]): the diagrams are to be read from left to right; the inputs and the outputs of orange boxes correspond to inputs and outputs of programs correspondingly and are respectively bunched by coproducts; feedback loops graphically represent iteration. We additionally use green boxes to indicate program fragments to which iteration is applied, and blue boxes in the uniformity axiom to distinguish side-effect-free morphisms of the form ηf. The novel ingredient are the bullets indicating guarded outputs. It is easy to see from the diagrams that the axioms are stated in such a way that feedback loops are always cut by bullets. Our background axiomatization of the notion of guardedness precisely takes care of the fact that guardedness judgments extends from atomic (orange) boxes to compound diagrams, which is why we do not have to re-

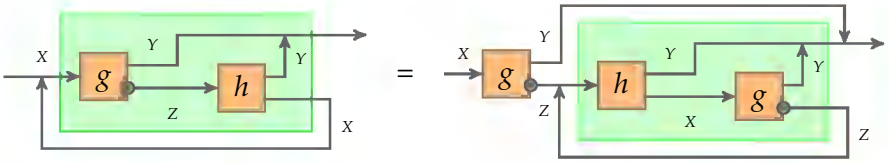
Fixpoint:



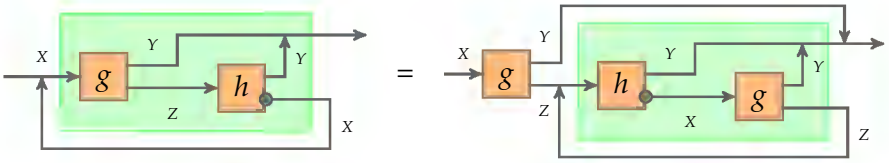
Naturality:



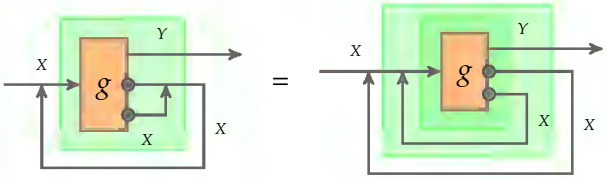
Dinaturality I:



Dinaturality II:



Codiagonal:



Uniformity:

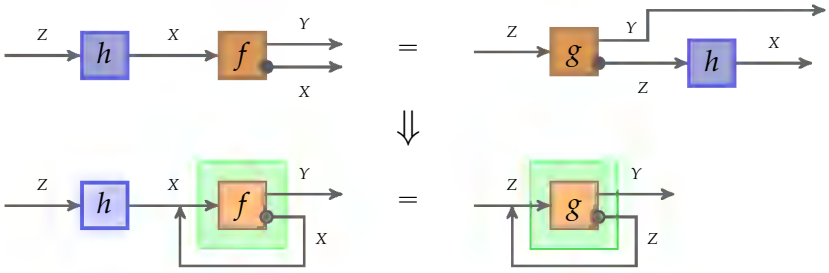


Figure 2.2: Axioms of guarded iteration.

peat bullets along a loop. A notable feature of our axioms are the two versions of the dinaturality axiom, which have a single classical prototype, and in fact both equations

express the same fact about loops whenever they are guarded, but the distinction comes from the fact they are guarded for different reasons: in Dinaturality I, guardedness of both loops is implied by guardedness of g in Z, while in Dinaturality II it is implied by guardedness of h in X. As we prove in the paper, both versions of guardedness are, in fact, derivable from the remaining axioms, and the crucial role in this proof is played by Uniformity. This is a generalization of recently discovered analogous phenomenon that takes place on the level of abstract iteration theories [32].

Our notion of guardedness is designed is such a way that it covers both unguarded iteration (as in complete Elgot monads) and guarded iteration (in the sense of Uustalu [111]). The first outcome of this unification is that extension of unguarded iteration along the monad transformer $\mathbb{T} \mapsto \mathbb{T}_\Sigma$ discussed in Section 2.3.1 is complemented by an analogous result for guarded iteration. That is, whenever $\mathbb{T}$ is guarded (pre-iterative) then so is $\mathbb{T}_\Sigma$ under a notion of guardedness canonically extended from $\mathbb{T}$ to $\mathbb{T}_\Sigma$. An example showing why this is of interest can be obtained by taking $\mathbb{T}$ to be the finite powerset monad $\mathcal{P}_\omega$ on **Set** and $\Sigma = A_\tau \times -$ where A_τ is the disjoint union of some set of actions A and a distinguished internal action τ. The sets $\mathbb{T}_\Sigma X = \nu\gamma.\mathcal{P}_\omega(X + A_\tau \times \gamma)$ model finitely branching nondeterministic processes possibly delivering final results in X in the style of Milner's CCS, and we can distinguish two notions of guardedness for them: the weaker one requiring any recursive process call to be preceded by an action from A_τ, and the stronger one requiring any recursive process call to be preceded by an action from A. We reconcile this with our framework as follows. Using the isomorphism

$$\nu\gamma.\mathcal{P}_\omega(X + A_\tau \times \gamma) \cong \nu\gamma.\nu\gamma'.\mathcal{P}_\omega(X + A \times \gamma + \gamma')$$

we can equivalently view every process as an inhabitant of the left-hand side under the standard notion of guardedness, which is the weaker one from the process algebra perspective, or as an inhabitant of the right-hand side, with the notion of guardedness canonically extended from the inner monad $\nu\gamma'.\mathcal{P}_\omega(- + \gamma')$ for which guardedness is selected to be trivial (i.e. no morphism is guarded, because guardedness w.r.t. the τ-action is discounted) to the outer one – this yields the stronger guardedness principle. The stronger notion of guardedness thus becomes modular, which can be illustrated more concretely as follows:

$$x = a.x + \tau.x \qquad x = a.x + y \qquad y = \tau.x$$

Here the defining term of the equation on the left hand side is guarded in the weak sense but not in the strong sense, and the latter fact is established by replacing it equivalently with a pair of definitions according to the above isomorphism where unguardedness is witnessed by the fact that in the first equation the definition of x involves an unguarded call of y, whose definition itself is unguarded.

Another basic fact established in [G9] is that when a monad is guarded iterative (i.e. suitably typed guarded morphisms possess unique solutions) then all the laws of iteration in Fig. 2.2 are satisfied automatically. The first main result of [G9] is a characterization of complete Elgot monads in terms of guarded iterative ones: it turns out

that the complete Elgot monads are precisely the *iteration-congruent retracts* of guarded iterative ones, and for a given complete Elgot monad $\mathbb{T}$ the corresponding guarded iterative monad can be chosen concretely as $\nu\gamma.\, T(- + \gamma)$. In other words, a complete Elgot monad $\mathbb{T}$ is a certain collapse of $\nu\gamma.\, T(- + \gamma)$ (cf. Section 1.2.5) under a monad morphism ρ that has an objectwise left inverse and respects iteration, i.e. $\rho f = \rho g$ implies $\rho f^\dagger = \rho g^\dagger$. This characterization greatly facilitates working with complete Elgot monads (which are hard) by reduction to the corresponding guarded iterative monads (which are easy). For example, this can be used to obtain as an easy consequence the fact that Elgotness propagates along the monad transformer $\mathbb{T} \mapsto \mathbb{T}_\Sigma$ (which we previously shown in [G6] under subtly more general assumptions).

The second main result of [G9] is a characterization of complete Elgot monads as certain algebras in a category of monadic transformations of monads. Specifically, the transformation $T \mapsto \nu\gamma.\, T(- + \gamma)$ can itself be seen as a monad in the category of monads, and complete Elgot monads are algebras of this monad, for, as we indicated above, they are characterized by collapsing monad morphisms $\rho : \nu\gamma.\, T(- + \gamma) \to T$. We characterize complete Elgot monads among all such algebras: these are precisely the algebras $(\mathbb{T}, \rho)$ satisfying the delay-cancellation condition $\rho \triangleright = \rho$ where $\triangleright$ is an endomorphism on each $\nu\gamma.\, T(X + \gamma)$ intuitively adding a new T-layer. This characterization is of importance to intensional type theory where collapses of Capretta's *delay monad* $\nu\gamma.\, (- + \gamma)$ are recently being explored [113, 115]. On the one hand, the delay monad is the initial algebra w.r.t. $T \mapsto \nu\gamma.\, T(- + \gamma)$, but it is not delay cancellative. Suitable collapses of the delay monad are rather difficult to construct and study explicitly, besides classical settings when a canonical and rather simplistic choice is available in the shape of the maybe monad [G6] (cf. Section 1.2.5). Our analysis implies that the *initial complete Elgot monad* a reasonable candidate for a collapse of the delay monad.

2.3.3 Monads for Iteration vs. Algebras for Iteration

Our treatment of guarded and unguarded monad-based iteration in conjunction with our results on monad-based Hoare calculi set the scene for discussing our work on Elgot algebras [G3], which roughly speaking contributes to relating monad-based iteration and algebra-based iteration. As we saw in Sections 2.2.1 and 2.2.2, one can build a Hoare calculus for effectful programs based either on the assumption that the monad is suitably order-enriched, or on the assumption that it supports unique solutions of guarded definitions by corecursion. These two assumptions are in conflict: in order-enriched semantics iteration is only a least fixpoint and not a unique one, while in process algebra, where guarded definitions can indeed be solved uniquely, processes cannot be ordered in a way consistent with strong bisimilarity. A "grand unification" of ordered semantics and coalgebraic semantics in the context of Hoare logics is thus potentially possible either by dropping the orderedness assumption via complete Elgot monads, or by switching to partial iteration via guarded iterative monads. Both approaches are connected in a formal way [G9], which we reiterate as a slogan: *complete Elgot monads are iteration-congruent retracts of guarded iterative monads*.

Recall that another critical ingredient of the design of our Hoare calculus discussed

in Section 2.2.1 is a truth value object $P1$ used in the semantics of assertions. More abstractly, we can attempt to replace it with some $\mathbb{T}$-algebra Ω supporting enough additional structure for interpreting the necessary logical connectivities (e.g. assume it to be an internal Heyting algebra). Depending on the choice of the monad $\mathbb{T}$ as a complete Elgot monad or as a guarded iterative monad we obtain different notions of algebras with iteration. In [G3], we compare these two notions under a specific choice of guarded iterative monads as being of the form $\nu\gamma.\, T(- + \Sigma\gamma)$ with T being the functor part of a monad and Σ being an arbitrary endofunctor (in fact, the setting of [G3] is slightly more general and involves fixpoints $\nu\gamma.\, (- \# \gamma)$ of *parametrized monads* [112], but we stick to the concrete instance $X \# Y = T(X + \Sigma Y)$, for expository purposes). We therefore obtain

- (Eilenberg-Moore-)algebras of $\nu\gamma.\, T(- + \Sigma\gamma)$ for an arbitrary monad $\mathbb{T}$;
- (Eilenberg-Moore-)algebras of $\mathbb{T}$ for a complete Elgot monad $\mathbb{T}$.

Regarding the first option, in [G3] we obtain an intrinsic characterization of the requisite algebras using an axiomatization adapted from previous work by Adámek et.at. [8]. Specifically, given a monad $\mathbb{T}$ and an endofunctor Σ, we introduce a class of *complete Elgot algebras* corresponding to them, characterized by the following data: every such algebra A carries an algebra structure α w.r.t. the monad $\mathbb{T}$ and an algebra structure β w.r.t. the functor Σ, and A is equipped with an iteration operator

$$(f : X \to T(A + \Sigma X)) \;\mapsto\; (f^\dagger : X \to A)$$

satisfying three axioms, e.g. the following *solution axiom*: $f^\dagger = \alpha \circ T[\mathsf{id}, \beta \circ (\Sigma f^\dagger)] \circ f$. Under the assumption that all final coalgebras $\nu\gamma.\, T(X + \Sigma\gamma)$ exist, we show that the category of complete Elgot algebras is equivalent to the category of $\nu\gamma.\, T(- + \Sigma\gamma)$-algebras. We note, however, that the notion of complete Elgot algebra makes perfect sense also when the requisite final coalgebras do not exist. Also, the axioms of iteration for complete Elgot algebras are more light-weight compared to those for complete Elgot monads in that the former do not include any form of the rather powerful *Bekić law*, or, equivalently, the Codiagonal axiom as in Fig. 2.2. This distinction is in fact critical for relating the situations from the first and the second clauses of the above list. Since each complete Elgot monad $\mathbb{T}$ is an iteration-congruent retract of $\nu\gamma.\, T(- + \gamma)$, we can convert each $\mathbb{T}$-algebra to a $\nu\gamma.\, T(- + \gamma)$-algebra, by composing the relevant algebra structure with the retraction morphism. This identifies $\mathbb{T}$-algebras as certain complete Elgot algebras w.r.t. $\mathbb{T}$ and $\Sigma = \mathrm{Id}$, and it turns out they are exactly those for which the Σ-algebra structures are identity (in [G3] we obtain this directly without assuming existence of final coalgebras). We also show a converse to this statement, i.e. a monad $\mathbb{T}$ is completly Elgot if the corresponding category of complete Elgot algebras satisfies an analogue of the Codiagonal axiom.

2.4 Effect Combination

One advantage of monad-based semantics is that it automatically provides a basic compositionality principle that allows composing computations sequentially. This contrasts the monad based style of semantics with the purely algebraic or coalgebraic ones.

The sequential compositionality principle is only valid for computations w.r.t. to the same monad. In practice, it is also important to be able to compose computations (i) belonging to different monads; (ii) not only sequentially (e.g. in parallel, or non-deterministially). This leads to the problem of composing monads, which dates back to Freyd [34] and which reemerged recently in the context of semantics [53, 52]. Two operations on monads are of particular importance for computer science applications: *sum* (or coproduct) and *tensor*. The sum of two monad is their coproduct in the category of monads over the given category. The idea behind a sum $\mathbb{S} + \mathbb{T}$ is to give the smallest common domain for computations w.r.t. $\mathbb{S}$ and $\mathbb{T}$, not assuming any laws connecting the effects from $\mathbb{S}$ with the effects from $\mathbb{T}$. Thus, by combining reading/writing effects with non-determinism one obtains domains for communicating non-deterministic processes for which the distributive law

$$a.(x + y) = a.x + a.y \tag{2.1}$$

does not hold, as expected in process algebra. Sometimes, however, laws like (2.1) are desirable, in which case one may involve the *tensor product* $\mathbb{S} \otimes \mathbb{T}$ of two monads, which is essentially the coarsening of the sum $\mathbb{S} + \mathbb{T}$ under the law of noninterference of effects, or *tensor law*. The tensor law requires intuitively that the effects from $\mathbb{S}$ and $\mathbb{T}$ commute over each other, and so by combining reading/writing with non-determinsm by tensoring, one obtains in fact the equation (2.1), which is desirable in the context of trace semantics, or language semantics for automata.

The problem that arises is that the sum and the tensor in general need not exist. In the case of sum, a negative example can be constructed easily if one of the monads is *unranked*. Two computationally relevant unranked monads are the *powerset monad* and the *continuation monad*. The above example of combining reading/writing with non-determinism already yields a counterexample if non-determinism is unbounded, i.e. carried by the powerset monad, for the potential monad sum is too large to fit the category of sets. The case of tensors is much more intricate, due to the presence of tensor laws like (2.1), which potentially ameliorate the size issue. The problem of existence of the tensor product of monads in the category of sets dates back at least to [72]. It was shown in [52] that tensor products exists at least if both monads are ranked and it was relatively clear that the unranked powerset monad would not help construct a counterexample due to the preempting power of the tensor law. In [52] a hope was raised that a counterexample could be constructed with help of the continuation monad. However, quite to the contrary, in [G8] we identified a class of *uniform monads* tensors with which always exist, and showed that the continuation monad (and in fact also the unbounded powerset monad) is uniform. The first counterexample is proposed by the author in [G8]. Latter, Bowler and Levy came up with a different counterexample, which resulted in joint publication [G1]. In summary, two pairs $(\mathbb{S}, \mathbb{T})$ of monads are presented in [G8] for which the tensor product $\mathbb{S} \otimes \mathbb{T}$ on **Set** does not exist:

- $\mathbb{S}$ is the *well-order monad* for the first time introduced in [G8] with

$$SX = \{(Y, \rho) \mid \emptyset \neq Y \subseteq X,\ \rho \text{ a well-order on } Y\} \cup \{\bot\}$$

and $\mathbb{T}$ is the free monad on two binary operation symbols. The monad structure on $\mathbb{S}$ is defined similarly to the one on the list monad, so, intuitively, the elements of SX are generalized non-empty lists over X, of any ordinal-length. The error-value $\perp$ is triggered by repetitive list elements, and propagates upwards.

- $\mathbb{S}$ is the finite powerset monad $\mathcal{P}_\omega X = \{Y \mid Y \textit{ a finite subset of } X\}$ and $\mathbb{T}$ is a sophisticated monad constructed by a painstaking manipulation of ordinals.

In summary, the first pair of examples is easy to understand by drawing on a suitable computational intuition, while in the second example pair, one of the monads is very sophisticated but the partner monad is more interesting from the semantic perspective than the two monads from the first pair. The latter non-tensorability result also carries a stronger message, for the finite powerset monad is subject to nontrivial identities, potentially helping tensor products with it to exist.

3 Auhtor's Publications

[G1] Nathan Bowler, Sergey Goncharov, Paul Blain Levy, and Lutz Schröder. "Exploring the Boundaries of Monad Tensorability on Set". In: *Logical Methods in Computer Science* 9.3 (2013).

[G2] Sergey Goncharov. "Trace Semantics via Generic Observations". In: *Proc. Algebra and Coalgebra in Computer Science: 5th International Conference (CALCO 2013)*. Ed. by Reiko Heckel and Stefan Milius. Lecture Notes in Computer Science. Springer, 2013, pp. 158–174.

[G3] Sergey Goncharov, Stefan Milius, and Christoph Rauch. "Complete Elgot Monads and Coalgebraic Resumptions". In: *Proc. 32st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXII)*. Ed. by Lars Birkedal. Vol. 325. Electronic Notes in Theoretical Computer Science. 2016, pp. 147–168.

[G4] Sergey Goncharov, Stefan Milius, and Alexandra Silva. "Towards a Coalgebraic Chomsky Hierarchy". In: *Proc. Theoretical Computer Science: 8th IFIP TC 1/WG 2.2 International Conference (TCS 2014)*. Ed. by Josep Diaz, Ivan Lanese, and Davide Sangiorgi. Vol. 8705. Springer, 2014, pp. 265–280.

[G5] Sergey Goncharov and Dirk Pattinson. "Coalgebraic Weak Bisimulation from Recursive Equations over Monads". In: *Proc. Automata, Languages, and Programming – 41st International Colloquium, (ICALP 2014)*. Ed. by Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias. Vol. 8573. Lecture Notes in Computer Science. 2014, pp. 196–207.

[G6] Sergey Goncharov, Christoph Rauch, and Lutz Schröder. "Unguarded Recursion on Coinductive Resumptions". In: *Proc. 31st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXI)*. Vol. 319. Electronic Notes in Theoretical Computer Science. 2015, pp. 183 –198.

[G7] Sergey Goncharov and Lutz Schröder. "A coinductive calculus for asynchronous side-effecting processes". In: *Information and Computation* 231 (2013), pp. 204 –232.

[G8] Sergey Goncharov and Lutz Schröder. "A Relatively Complete Generic Hoare Logic for Order-Enriched Effects". In: *Proc. 28th Annual Symposium on Logic in Computer Science (LICS 2013)*. IEEE Computer Society, 2013, pp. 273–282.

[G9] Sergey Goncharov, Lutz Schröder, Christoph Rauch, and Maciej Piróg. "Unifying Guarded and Unguarded Iteration". In: *Proc. 20th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2017)*. Ed. by Javier Esparza and Andrzej Murawski. Vol. 10203. 2017.

[G10] Christoph Rauch, Sergey Goncharov, and Lutz Schröder. "Generic Hoare Logic for Order-Enriched Effects with Exceptions". In: *Proc. 23rd International Workshop on Algebraic Development Techniques (WADT 16)*. Vol. 10644. Lecture Notes in Computer Science. Springer, 2017.

4 References

[1] S. Abramsky. "Semantics of Interaction: an introduction to Game Semantics". In: *Proceedings of the 1996 CLiCS Summer School, Isaac Newton Institute*. Ed. by P. Dybjer and A. Pitts. Cambridge University Press, 1997, pp. 1–31.

[2] Samson Abramsky. "Domain Theory in Logical Form". In: *Annals of Pure and Applied Logic* 51 (1991), pp. 1–77.

[3] Samson Abramsky. "Intensionality, Definability and Computation". In: *Johan van Benthem on Logic and Information Dynamics*. Ed. by Alexandru Baltag and Sonja Smets. Cham: Springer International Publishing, 2014, pp. 121–142.

[4] Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. "Full Abstraction for PCF". In: *Information and Computation* 163.2 (2000), pp. 409 –470.

[5] Samson Abramsky and Achim Jung. "Domain Theory". In: *Handbook of Logic in Computer Science*. Vol. 3. Oxford University Press, 1994, pp. 1–168.

[6] Peter Aczel, Jiří Adámek, Stefan Milius, and Jiří Velebil. "Infinite trees and completely iterative theories: a coalgebraic view". In: *Theoretical Computer Science* 300.1 (2003), pp. 1 –45.

[7] Jiří Adámek, Reinhard Börger, Stefan Milius, and Jiří Velebil. "Iterative algebras: How iterative are they?" In: *Theory Appl. Cat.* 19 (2008), pp. 61–92.

[8] Jiří Adámek, Stefan Milius, and Jiří Velebil. "Elgot Algebras: (Extended Abstract)". In: *Electronic Notes in Theoretical Computer Science* 155 (2006), pp. 87–109.

[9] Jiří Adámek, Stefan Milius, and Jiří Velebil. "Equational properties of iterative monads". In: *Inf. Comput.* 208 (2010), pp. 1306 –1348.

[10] Krzysztof Apt and Gordon Plotkin. "A Cook's Tour of Countable Nondeterminism". In: *DAIMI Report Series* 10.133 (1981).

[11] Christel Baier and Holger Hermanns. "Weak Bisimulation for Fully Probabilistic Processes". In: *Proc. CAV 1997*. Springer, 1997, pp. 119–130.

[12] Falk Bartels. "Generalised coinduction". In: *Math. Struct. Comput. Sci.* 13 (2003), pp. 321–348.

[13] Ingo Battenfeld, Matthias Schröder, and Alex Simpson. "A Convenient Category of Domains". In: *Electronic Notes in Theoretical Computer Science* 172.Supplement C (2007). Computation, Meaning, and Logic: Articles dedicated to Gordon Plotkin, pp. 69–99.

[14] Benno van den Berg. "Predicative toposes". In: *ArXiv e-prints* (July 2012).

[15] Jan Bergstra. *Handbook of Process Algebra*. Ed. by A. Ponse and Scott A. Smolka. New York, NY, USA: Elsevier Science Inc., 2001.

[16] Lars Birkedal, Rasmus Møgelberg, Jan Schwinghammer, and Kristian Støvring. "First Steps in Synthetic Guarded Domain Theory: Step-Indexing in the Topos of Trees". In: *Logical Methods in Computer Science* 8.4:1 (2012), pp. 1–45.

[17] Lars Birkedal, Jan Schwinghammer, and Kristian Støvring. "A Metric Model of Lambda Calculus with Guarded Recursion". In: *FICS*. Laboratoire d'Informatique Fondamentale de Marseille, 2010, pp. 19–25.

[18] Andreas Blass and Yuri Gurevich. "The Underlying Logic of Hoare Logic". In: *Current Trends in Theoretical Computer Science*. 2001, pp. 409–436.

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

[20] Ronald Book and Sheila Greibach. "Quasi-Realtime Languages." In: *Math. Systems Theory* 4.2 (1970), pp. 97–111.

[21] Tomasz Brengos. "Weak bisimulation for coalgebras over order enriched monads". In: *Logical Methods in Computer Science* 11.2 (2015).

[22] Venanzio Capretta. "Coalgebras in functional programming and type theory". In: *Theoretical Computer Science* 412.38 (2011). CMCS Tenth Anniversary Meeting, pp. 5006–5024.

[23] Venanzio Capretta. "General recursion via coinductive types". In: *Logical Methods in Computer Science* 1.2 (2005).

[24] James Chapman, Tarmo Uustalu, and Niccolò Veltri. "Quotienting the Delay Monad by Weak Bisimilarity". In: *ICTAC*. Vol. 9399. Lecture Notes in Computer Science. Springer, 2015, pp. 110–125.

[25] Alonzo Church. "An Unsolvable Problem of Elementary Number Theory". In: *American Journal of Mathematics* 58.2 (Apr. 1936), pp. 345–363.

[26] William Douglas Clinger. "Foundations of Actor Semantics". PhD dissertation. Massachusetts Institute of Technology, 1981.

[27] Stephen A. Cook. "Soundness and Completeness of an Axiom System for Program Verification". In: *SIAM J. Comput.* 7.1 (1978), pp. 70–90.

[28] Flavio Corradini, Rocco De Nicola, and Anna Labella. "Graded Modalities and Resource Bisimulation". In: *Proc. FSTTCS 1999*. Ed. by C. Pandu Rangan, Venkatesh Raman, and Ramaswamy Ramanujam. Vol. 1738. LNCS. Springer, 1999, pp. 381–393.

[29] Roy Crole and Andrew Pitts. "New Foundations for Fixpoint Computations". In: *Logic in Computer Science, LICS 1990*. IEEE Computer Society, 1990, pp. 489–497.

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

[31] Martín Hötzel Escardó. "A metric model of PCF". In: *Realizability Semantics and Applications*. 1999.

[32] Zoltán Ésik and Sergey Goncharov. "Some Remarks on Conway and Iteration Theories". In: *CoRR* abs/1603.00838 (2016). URL: http://arxiv.org/abs/1603.00838.

[33] Marcelo Fiore. *Axiomatic Domain Theory in Categories of Partial Maps*. Distinguished Dissertations in Computer Science. Cambridge University Press, 2004.

[34] Peter Freyd. "Algebra valued functors in general and tensor products in particular". In: *Colloq. Math.* 14 (1966), pp. 89–106.

[35] Peter Freyd. "Algebraically complete categories". In: *Category Theory: Proceedings of the International Conference held in Como*. Ed. by Aurelio Carboni, Maria Cristina Pedicchio, and Guiseppe Rosolini. Vol. 1488. Lecture Notes in Math. Springer, 1991, pp. 95–104.

[36] Peter Freyd. "Remarks on algebraically compact categories". In: *Applications of category theory in computer science: Proceedings of the London Mathematical Society Symposium, Durham 1991*. Ed. by M. P. Fourman, P. T. Johnstone, and A. M. Pitts. Vol. 177. London Mathematical Society Lecture Note Series. Cambridge University Press, 1992, pp. 95–106.

[37] Jean-Yves Girard. "Towards a geometry of interaction". In: *Contemporary Mathematics* 92.69-108 (1989), p. 6.

[38] Michèle Giry. "A categorical approach to probability theory". In: *Categorical Aspects of Topology and Analysis* (1982), pp. 68–85.

[39] Kurt Gödel. *On undecidable propositions of formal mathematical systems*. Lecture notes taken by Kleene and Rosser at Princeton, reprinted in The Undecidable, ed.˜Martin Davis, pp.˜39–74. 1934.

[40] Sergey Goncharov. "Kleene monads". PhD thesis. Universität Bremen, 2010.

[41] Sergey Goncharov, Christoph Rauch, Lutz Schröder, and Julian Jakob. "Unguarded recursion on coinductive resumptions". In: *Logical Methods in Computer Science* (submitted).

[42] Sergey Goncharov, Lutz Schröder, and Till Mossakowski. "Kleene monads: handling iteration in a framework of generic effects". In: *Algebra and Coalgebra in Computer Science, CALCO 2009*. Vol. 5728. LNCS. Springer, 2009, pp. 18–33.

[43] Esfandiar Haghverdi and Philip Scott. "A categorical model for the geometry of interaction". In: *Theoretical Computer Science* 350.2 (2006). Automata, Languages and Programming: Logic and Semantics (ICALP-B 2004), pp. 252 –274.

[44] Ichiro Hasuo. "Generic weakest precondition semantics from monads enriched with order". In: *Theoretical Computer Science* 604 (2015). Coalgebraic Methods in Computer Science, pp. 2 –29.

[45] Ichiro Hasuo, Bart Jacobs, and Ana Sokolova. "Generic Trace Semantics via Coinduction". In: *Logical Methods in Computer Science* 3.4 (2007).

[46] Ichiro Hasuo, Bart Jacobs, and Ana Sokolova. "Generic trace theory". In: *International Workshop on Coalgebraic Methods in Computer Science (CMCS 2006), volume 164 of Elect. Notes in Theor. Comp. Sci*. Elsevier, 2006, pp. 47–65.

[47] Matthew Hennessy and Gordon Plotkin. "Full Abstraction for a Simple Parallel Programming Language". In: *MFCS*. 1979, pp. 108–120.

[48] Hagen Huwig and Axel Poigné. "A Note on Inconsistencies Caused by Fixpoints in a Cartesian Closed Category". In: *Theor. Comput. Sci.* 73 (1990), pp. 101–112.

[49] J.M.E. Hyland and C.-H.L. Ong. "On Full Abstraction for PCF: I, II, and III". In: *Information and Computation* 163.2 (2000), pp. 285 –408.

[50] Martin Hyland. "First steps in Synthetic Domain Theory". In: *Category Theory*. Vol. 1144. LNM. Springer, 1992, pp. 131–156.

[51] Martin Hyland. "Some reasons for generalising domain theory". In: *Mathematical Structures in Computer Science* 20.2 (2010), 239–265.

[52] Martin Hyland, Paul Blain Levy, Gordon Plotkin, and John Power. "Combining algebraic effects with continuations". In: *Theoret. Comput. Sci.* 375.1-3 (2007). Festschrift for John C. Reynolds's 70th birthday, pp. 20 –40.

[53] Martin Hyland, Gordon Plotkin, and John Power. "Combining effects: Sum and tensor". In: *Theoret. Comput. Sci.* 357 (2006), pp. 70–99.

[54] Martin Hyland and John Power. "Discrete Lawvere theories and computational effects". In: *Theoret. Comput. Sci.* 366 (2006), pp. 144–162.

[55] Bart Jacobs. "Distributive laws for the coinductive solution of recursive equations". In: *Information and Computation* 204.4 (2006). Seventh Workshop on Coalgebraic Methods in Computer Science 2004, pp. 561 –587.

[56] Bart Jacobs and Erik Poll. "A Logic for the Java Modeling Language JML". In: *Fundamental Approaches to Software Engineering*. Vol. 2029. LNCS. Springer, 2001, pp. 284–299.

[57] Bart Jacobs and Jan Rutten. "A Tutorial on (Co)Algebras and (Co)Induction". In: *EATCS Bulletin* 62 (1997), pp. 222–259.

[58] Bart Jacobs, Alexandra Silva, and Ana Sokolova. "Trace Semantics via Determinization". In: *CMCS'12*. Vol. 7399. LNCS. Springer, 2012, pp. 109–129.

[59] Peter Johnstone. "Adjoint Lifting Theorems for Categories of Algebras". In: *Bulletin of the London Mathematical Society* 7.3 (1975), pp. 294–297.

[60] Achim Jung. "Continuous domain theory in logical form". In: *Computation, Logic, Games, and Quantum Foundations. The Many Facets of Samson Abramsky*. Ed. by B. Coecke, L. Ong, and P. Panangaden. Vol. 7860. Lecture Notes in Computer Science. Springer Verlag, 2013, pp. 166–177.

[61] Achim Jung and Regina Tix. "The troublesome probabilistic powerdomain". In: *Computation and Approximation*. Vol. 13. ENTCS. Elsevier, 1998.

[62] Bartek Klin. "Bialgebras for structural operational semantics: An introduction". In: *Theor. Comput. Sci.* 412.38 (2011), pp. 5043–5069.

[63] Dexter Kozen. *Automata and Computability*. New York: Springer-Verlag, 1997.

[64] Krishnaswami Krishnaswami and Nick Benton. "Ultrametric Semantics of Reactive Programs". In: *Logic in Computer Science, LICS 2011*. IEEE Computer Society, 2011, pp. 257–266.

[65] Sava Krstic, John Launchbury, and Dusko Pavlovic. "Categories of Processes Enriched in Final Coalgebras". In: *Foundations of Software Science and Computation Structures, FOSSACS 2001*. Vol. 2030. LNCS. Springer, 2001, pp. 303–317.

[66] William Lawvere. "An elementary theory of the category of sets (long version) with commentary". In: *Reprints in Theory and Applications of Categories* 11 (2005). Reprinted and expanded from Proc. Nat. Acad. Sci. U.S.A. **52** (1964), With comments by the author and Colin McLarty, pp. 1–35.

[67] William Lawvere. "Functorial Semantics of Algebraic Theories". In: *Proc. Natl. Acad. Sci. USA* 50.5 (1963), pp. 869–872.

[68] Marina Lenisa, John Power, and Hiroshi Watanabe. "Distributivity for endofunctors, pointed and co-pointed endofunctors, monads and comonads". In: *Electronic Notes in Theoretical Computer Science* 33.Complete (2000), pp. 230–260.

[69] Paul Blain Levy. "Infinite trace equivalence". In: *Annals of Pure and Applied Logic* 151.2 (2008), pp. 170 –198.

[70] Steffen Lösch and Andrew Pitts. "Denotational Semantics with Nominal Scott Domains". In: *J. ACM* 61.4 (July 2014), 27:1–27:46.

[71] Octavio Malherbe, Philip Scott, and Peter Selinger. "Partially traced categories". In: *J. Pure Appl. Algebra* 216 (2012), pp. 2563 –2585.

[72] Ernest Manes. "A triple theoretic construction of compact algebras". In: *Seminar on Triples and Categorical Homology Theory*. Vol. 80. Lect. Notes Math. Springer, 1969, pp. 91–118.

[73] Ernest Manes and Michael Arbib. *Algebraic approaches to program semantics*. Texts and monographs in computer science. Springer-Verlag, 1986.

[74] Zohar Manna and Amir Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer, 1995.

[75] Stefan Milius and Tadeusz Litak. "Guard Your Daggers and Traces: On The Equational Properties of Guarded (Co-)recursion". In: *Fixed Points in Computer Science, FICS 2013*. Vol. 126. EPTCS. 2013, pp. 72–86.

[76] Robin Milner. *Communication and concurrency*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989.

[77] Robin Milner. "Fully abstract models of typed λ-calculi". In: *Theoretical Computer Science* 4.1 (1977), pp. 1 –22.

[78] Eugenio Moggi. *An Abstract View of Programming Languages*. Tech. rep. ECS-LFCS-90-113. Lecture Notes for course CS 359, Stanford Univ. Edinburgh Univ., Dept. of Comp. Sci., June 1989.

[79] Eugenio Moggi. "Notions of Computation and Monads". In: *Inf. Comput.* 93 (1991), pp. 55–92.

[80] Luke Ong. "Correspondence between operational and denotational semantics: the full abstraction problem for PCF". In: *Handb. Log. Comput. Sci.* 4.38 (Sept. 1995), pp. 269–356.

[81] Jaap van Oosten. *Realizability: An Introduction to its Categorical Side*. Studies in Logic and the Foundations of Mathematics. Elsevier Science, 2008.

[82] Dusko Pavlovic, Michael Mislove, and James Worrell. "Testing Semantics: Connecting Processes and Process Logics". In: *AMAST'06*. Vol. 4019. LNCS. Springer, 2006, pp. 308–322.

[83] Wesley Phoa. "Domain Theory in Realizability Toposes". PhD thesis. Cambridge University, 1990.

[84] Andrew Pitts. "Polymorphism is Set Theoretic, Constructively". In: *Category Theory and Computer Science*. London, UK, UK: Springer-Verlag, 1987, pp. 12–39.

[85] Gordon Plotkin and John Power. "Adequacy for Algebraic Effects". In: *Foundations of Software Science and Computation Structures (FoSSaCS'01), Proceedings*. Ed. by Furio Honsell and Marino Miculan. Vol. 2030. LNCS. 2001, pp. 1–24.

[86] Gordon Plotkin and John Power. "Algebraic Operations and Generic Effects". In: *Appl. Cat. Struct.* 11 (2003), pp. 69–94.

[87] Gordon Plotkin and John Power. "Notions of Computation Determine Monads". In: *Foundations of Software Science and Computation Structures (FoSSaCS'02), Proceedings*. Ed. by Mogens Nielsen and Uffe Engberg. Vol. 2303. LNCS. Springer, 2002, pp. 342–356.

[88] Gordon Plotkin and Matija Pretnar. "Handlers of Algebraic Effects". In: *European Symposium on Programming, ESOP 2009*. Vol. 5502. LNCS. Springer, 2009, pp. 80–94.

[89] Michael Rabin and Dana Scott. "Finite Automata and Their Decision Problems". In: *IBM J. Res. Dev.* 3.2 (Apr. 1959), pp. 114–125.

[90] John Reynolds. "Polymorphism is not Set-Theoretic". In: *Semantics of Data Types*. Ed. by Gilles Kahn, David B. MacQueen, and Gordon D. Plotkin. Vol. 173. Lecture Notes in Computer Science. Springer, 1984, pp. 145–156.

[91] Guiseppe Rosolini. "Continuity and Effectiveness in Topoi". PhD thesis. University of Oxford, 1986.

[92] Jan J. M. M. Rutten and Daniele Turi. "Initial Algebra and Final Coalgebra Semantics for Concurrency". In: *Proc. REX School/Symposium 1993*. Ed. by J. W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg. Vol. 803. Lecture Notes in Computer Science. Springer, 1994, pp. 530–582.

[93] Lutz Schröder and Till Mossakowski. "Generic exception handling and the Java monad". In: *Algebraic Methodology and Software Technology, AMAST 2004*. Vol. 3116. LNCS. Springer, 2004, pp. 443–459.

[94] Lutz Schröder and Till Mossakowski. "HasCASL: Integrated Higher-Order Specification and Program Development". In: *Theoret. Comput. Sci.* 410 (2009), pp. 1217–1260.

[95] Lutz Schröder and Till Mossakowski. "Monad-independent Hoare logic in HasCASL". In: *Fundamental Aspects of Software Engineering, FASE 2003*. Vol. 2621. LNCS. 2003, pp. 261–277.

[96] Dana Scott. "Continuous lattices". In: *Toposes, Algebraic Geometry and Logic: Dalhousie University, Halifax, January 16–19, 1971*. Ed. by F. W. Lawvere. Berlin, Heidelberg: Springer Berlin Heidelberg, 1972, pp. 97–136.

[97] Dana Scott. "Domains for denotational semantics". In: *Automata, Languages and Programming: Ninth Colloquium Aarhus, Denmark, July 12–16, 1982*. Ed. by Mogens Nielsen and Erik Meineche Schmidt. Berlin, Heidelberg: Springer Berlin Heidelberg, 1982, pp. 577–610.

[98] Dana Scott. *Outline of a Mathematical Theory of Computation*. Tech. rep. PRG–2. Oxford, England, 1970.

[99] Dana Scott and Christopher Strachey. *Toward a mathematical semantics for computer languages*. Vol. 1. Oxford University Computing Laboratory, Programming Research Group, 1971.

[100] Alexandra Silva, Filippo Bonchi, Marcello Bonsangue, and Jan Rutten. "Generalizing determinization from automata to coalgebras". In: *LMCS* 9.1 (2013).

[101] Alexandra Silva, Filippo Bonchi, Marcello Bonsangue, and Jan Rutten. "Quantitative Kleene coalgebras". In: *Information and Computation* 209.5 (2011). Special Issue: 20th International Conference on Concurrency Theory (CONCUR 2009), pp. 822 –849.

[102] Alexandra Silva, Filippo Bonchi, Marcello Bonsangue, and Jan J. M. M. Rutten. "Generalizing the powerset construction, coalgebraically". In: *FSTTCS*. Vol. 8. LIPIcs. 2010, pp. 272–283.

[103] Alex Simpson. *Recursive Types in Kleisli Categories*. Tech. rep. University of Edinburgh, 1992.

[104] Michael Smyth. "Power Domains and Predicate Transformers: A Topological View". In: *Automata, Languages and Programming, ICALP 1983*. Vol. 154. LNCS. Springer, 1983, pp. 662–675.

[105] Michael Smyth and Gordon Plotkin. "The category-theoretic solution of recursive domain equations". In: *Foundations of Computer Science, FOCS 1977*. IEEE Computer Society, 1977, pp. 13–17.

[106] Sam Staton. "Relating coalgebraic notions of bisimulation". In: *Logical Methods in Computer Science* 7.1 (2011).

[107] Christopher Strachey. "Towards a Formal Semantics". In: *Formal Language Description Languages for Computer Programming*. North Holland, 1966, pp. 198–220.

[108] Paul Taylor. "Geometric and Higher Order Logic in terms of Abstract Stone Duality". In: *Theory and Applications of Categories* 7.15 (2000), pp. 284–338.

[109] Daniele Turi and Gordon Plotkin. "Towards a Mathematical Operational Semantics". In: *Logic in Computer Science, LICS 1997*. 1997, pp. 280–291.

[110] Alan Turing. "On Computable Numbers, with an Application to the Entscheidungsproblem". In: *Proceedings of the London Mathematical Society* s2-42.1 (1936), pp. 230–265.

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

[112] Tarmo Uustalu. "Generalizing Substitution". In: *ITA* 37 (2003), pp. 315–336.

[113] Tarmo Uustalu and Niccolò Veltri. "The Delay Monad and Restriction Categories". In: *Theoretical Aspects of Computing – ICTAC 2017: 14th International Colloquium, Hanoi, Vietnam, October 23-27, 2017, Proceedings*. Ed. by Dang Van Hung and Deepak Kapur. Cham: Springer International Publishing, 2017, pp. 32–50.

[114] Tarmo Uustalu and Varmo Vene. "Primitive (Co)Recursion and Course-of-Value (Co)Iteration, Categorically". In: *Informatica (Lithuanian Academy of Sciences)* 10.1 (1999), pp. 5–26.

[115] Niccoló Veltri. "A Type-Theoretical Study of Nontermination". PhD thesis. Tallinn University of Technology, 2017.

[116] Steven Vickers. *Topology via logic*. Cambridge tracts in theoretical computer science. Cambridge: University Press, 1996.