Research Statement

Todd Schmid

My research is in mathematical logic, specifically universal algebra and coalgebra, and the applications of these disciplines to theoretical computer science and other areas of mathematics. I am interested in formal notions of behaviour arising in the study of state machines and other discrete event dynamical systems, particularly when behaviours can be composed and manipulated algebraically. My work is largely motivated by mathematical beauty, like many topics in the European computer science canon, but it is also deeply rooted in practical problems in the study of automata and the formal semantics of programming languages, concurrency, and model checking.

Previous Work

In the last few years, my work has been motivated by open axiomatization problems in the semantics of programming languages.

A programming language consists of a set of blocks of code that can be composed and manipulated algebraically and that stand for instructions that a computer can follow. Different sequences of instructions can produce the same computational outcome, and we call blocks of code that denote equivalent instructions *behaviourally equivalent*. For example, for any condition b and any two programs p_1 and p_2, the code snippets "if b then p_1 else p_2" and "if not b then p_2 else p_1" are intuitively behaviourally equivalent. The formal equation

$$\text{if } b \text{ then } p_1 \text{ else } p_2 = \text{if not } b \text{ then } p_2 \text{ else } p_1$$

is therefore said to be *sound* with respect to behavioural equivalence, and might be included in an axiomatization of (a set of equations capturing) behavioural equivalence.

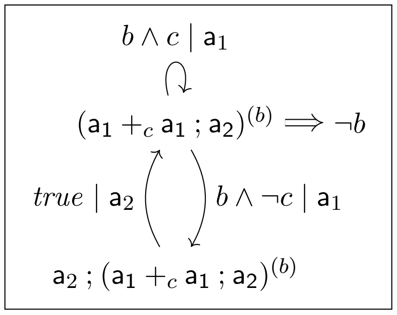
Definition. If every formal equation between behaviourally equivalent programs can be derived from an axiomatization, the axiomatization is said to be *complete* with respect to behavioural equivalence.

The search for complete axiomatizations of behavioural equivalence in programming languages is one of the oldest sources of open problems in theoretical computer science [Kle51; Mil84]. A complete axiomatization can often be turned into a decision procedure for program equivalence. Showing that an axiomatization is complete is often extremely difficult, and only a few tools exist that are both general and powerful for proving completeness theorems. During my PhD, the central aims of my research have been to elucidate the mechanics of completeness proofs that exist in the literature, reapply them to prove new completeness results, and provide general frameworks for constructing programming languages where existing axiomatization techniques apply.

Coalgebraic completeness theorems. Equational axiomatizations and their properties are studied in universal algebra [Coh81], but the appropriate notions of "instruction" and "behaviour" can more fruitfully be formalized in the language of *universal coalgebra* [Rut00].

Definition. Given an endofunctor F on a category $\mathbf{C}$, an *F-coalgebra* is a pair (X, γ) consisting of an object X, called the *state space*, and an arrow $\gamma : X \to FX$, called the *transition structure*. If the objects of $\mathbf{C}$ are sets, then the elements of a state space are called its *states*. A *homomorphism* $h : (X, \gamma) \to (Y, \vartheta)$ of F-coalgebras is an arrow $h : X \to Y$ such that $F(h) \circ \gamma = \vartheta \circ h$.

Language:	
$\mid b$	$(b \in 2^{\mathsf{At}})$
$\mid \mathsf{a}$	$(\mathsf{a} \in \mathsf{Act})$
$\mid e_1 +_b e_2$	(if b then e_1 else e_2)
$\mid e_1 \,;\, e_2$	
$\mid e^{(b)}$	(while b do e)


$b \wedge c \mid \mathsf{a}_1$
$(\mathsf{a}_1 +_c \mathsf{a}_1 \,;\, \mathsf{a}_2)^{(b)} \Longrightarrow \neg b$
$true \mid \mathsf{a}_2$
$b \wedge \neg c \mid \mathsf{a}_1$
$\mathsf{a}_2 \,;\, (\mathsf{a}_1 +_c \mathsf{a}_1 \,;\, \mathsf{a}_2)^{(b)}$

$L \to (\{0,1\} + \mathsf{Act} \times L)^{\mathsf{At}}$

Figure 1: The syntax of GKAT and an example of a state machine (a coalgebra) specified by a GKAT program. Above, At is a set of elements interpreted as atomic elements in a Boolean algebra of "tests".

Coalgebras are general state-based systems. By varying the category $\mathbf{C}$ and the endofunctor F, one can obtain deterministic, nondeterministic, and probabilistic automata [Rut98], pushdown automata [Sil+13], and Turing machines [Jac11; Gon+22] as examples of F-coalgebras.

The blocks of code that make up a programming language are states in an F-coalgebra that encodes their instructional information. The algebra of system behaviours expressible in a programming language consists of blocks of code up to *behavioural equivalence*, which is determined by $\mathbf{C}$ and F.

Definition. If x and y are states in the F-coalgebras (X_1, γ_1) and (X_2, γ_2) respectively, we say that x and y are *behaviourally equivalent* if there exist homomorphisms $h_1 : (X_1, \gamma_i) \to (Z, \zeta)$ and $h_2 : (X_2, \gamma_i) \to (Z, \zeta)$ such that $h_1 x = h_2 y$.

For specific $\mathbf{C}$ and F, behavioural equivalence instantiates to notions of equivalence from logic and computer science, including bisimilarity of Kripke frames/models from modal logic [Cir+11] and Myhill-Nerode equivalence from formal language theory [BCR15].

In [SRS21], my coauthors and I study axiomatizations of behavioural equivalence for set-based coalgebras in general. Fix an endofunctor F on the category of sets and functions.

Definition. A subset $V \subseteq X$ is *open* in an F-coalgebra (X, γ) if γ restricts to an F-coalgebra $\gamma|_V : V \to FV$.

Definition. Let $\mathcal{V}$ be a class of F-coalgebras. An F-coalgebra (Z, ζ) is called *locally final in* $\mathcal{V}$ if

1. for any $z \in Z$ there is a $V \subseteq Z$ open in (Z, ζ) such that $z \in V$ and $(V, \gamma|_V) \in \mathcal{V}$, and
2. every F-coalgebra (X, γ) in $\mathcal{V}$ admits a unique homomorphism $(X, \gamma) \to (Z, \zeta)$ of F-coalgebras.

Theorem 1 (§5 of [SRS21])**.** *Let F be an endofunctor on the category of sets and functions. In an F-coalgebra (X, γ), behavioural equivalence is equality if and only if there exists a class $\mathcal{V}$ of F-coalgebras such that (X, γ) is locally final in $\mathcal{V}$ and $\mathcal{V}$ is closed under quotients.*

In other words, an axiomatization of behavioural equivalence is complete in a programming language if and only if the coalgebra consisting of blocks of code modulo the axioms satisfies a certain universal property. This generalizes the approach to completeness proofs found in a number of historically significant works in the Kleene algebra [Sal66; Jac06; Sil10; Mil10; Sch+21] and process algebra literature [Mil84; GF20], starting with Salomaa's complete axiomatizations of the algebra of regular events in the 1960s [Sal66].

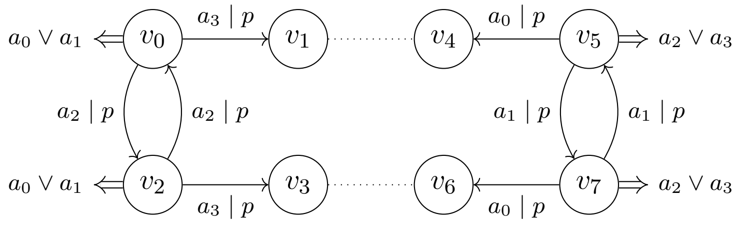


Figure 2: As depicted, this automaton is well-nested. However, identifying v_1 with v_4, and v_3 with v_6, we obtain an automaton that is not well-nested.

GKAT. *Guarded Kleene algebra with tests* (*GKAT*) [KT08] is a deceptively simple programming language designed to capture control flow (if-then-else and loop constructs) in imperative programming languages like Java, C++, and python. In [Smo+20], it was shown that behaviourally equivalent blocks of GKAT code can be algorithmically identified extremely efficiently. An axiomatization of equivalence appears in [Smo+20], but their proof of completeness requires an undesirable axiom called the *uniqueness axiom*. Hoping to avoid the uniqueness axiom, the authors of [Smo+20] made the following conjecture.

Conjecture 1. A certain class of coalgebras generated from GKAT programs, the class of *well-nested automata*, is closed under quotients.

The significance of Conjecture 1 is that the provable equivalence classes of GKAT programs form a coalgebra that is locally final in the class of well-nested automata. By Theorem 1, Conjecture 1 implies

Conjecture 2. The uniqueness axiom can be derived from the other axioms of GKAT (equivalently, the axiomatization without the uniqueness axiom is complete).

In [Sch+21], my coauthors and I observe that the automaton in fig. 2 is a counterexample to Conjecture 1: it is a well-nested automaton with a quotient that is not well-nested. We also offer a fresh take on the algebra of behaviours of GKAT programs by identifying it with an algebra of what we call *nested trees*.

Theorem (§5 of [Sch+21])**.** *The algebra of behaviours of GKAT programs over atomic tests* At *and actions* Σ *is a subalgebra of the algebra of* At*-branching* Σ*-labelled trees with leaves in* At.

The set of all trees forms a compact metric space, in which the nested trees are dense. Among our contributions is a characterization of nested trees as certain limits.

Figure 2 disproves Conjecture 1, but Conjecture 2 and the completeness problem for GKAT remain open. In a recent breakthrough, my coauthors and I proved that the conjectured axiomatization of GKAT in [Smo+20] is complete for a large fragment of GKAT programs called the *skip-free fragment* [SKS22].

Theorem ([SKS22])**.** *If* e *and* f *are behaviourally equivalent skip-free GKAT expressions, then* $\vdash e = f$ *is provable in GKAT without using the uniqueness axiom.*

This settles Conjecture 2 for that fragment. Our approach reduces the problem to a completeness theorem due to Clemens Grabmayer and Wan Fokkink [GF20] for a different language.

Processes Parametrized. Specialized programming languages are designed to capture particular aspects of computing. GKAT is a good example of this: its constructs are restricted to *control flow*, ie. conditional statements like if-then-else and loop statements like while-do. This makes GKAT perfect for studying how algebraic manipulations affect the control flow structure of real programs, since it abstracts away from irrelevant details like memory management. Other specialized programming languages include probabilistic programming languages [BSV19] and languages for concurrency [Mil80].

In [Sch+22], my coauthors and I propose a general framework for designing and studying specialized programming languages that capture computational effects like control flow, uncertainty, concurrency, and more. Formally, given an algebraic theory and two sets Act and Var, our *processes parametrized* framework generates a programming language Exp, a functor B, a B-coalgebra structure (Exp, ϵ), and a complete axiomatization of behavioural equivalence in (Exp, ϵ).

Definition. Let S be an algebraic signature and EQ a set of formal equations between S-terms. Define the *expression language* Exp to be the set of terms generated by the grammar

$$\mathsf{v} \mid \mathsf{a}e \mid \sigma(e_1, \dots, e_n) \mid \mu\mathsf{v}\ e \qquad (\mathsf{v} \in \mathsf{Var},\ \mathsf{a} \in \mathsf{Act},\ \sigma \in S, \text{ and } e, e_i \in \mathsf{Exp})$$

The variable v is *free* in e if it does not appear within the scope of any $\mu\mathsf{v}$, *unguarded* in e if it appears outside the scope of every $\mathsf{a}(-)$, and *guarded* (*gdd.*) in e if it is not unguarded in e.

If M constructs the free S-algebra satisfying EQ on every set, define the functor

$$B = M(\mathsf{Var} + \mathsf{Act} \times (-))$$

on the category of sets and functions.

Theorem (See [Sch+22])**.** *There is a B-coalgebra structure (Exp, ϵ) on the expression language such that for any finite B-coalgebra (X, γ), and any $x \in X$, there is an expression $e \in \mathsf{Exp}$ such that x and e are behaviourally equivalent. Furthermore, the equations* EQ *and the fixed-point rules*

$$\frac{\mathsf{w} \text{ not free in } e}{\mu\mathsf{v}\ e = \mu\mathsf{w}\ e[\mathsf{v} := \mathsf{w}]} \qquad \frac{\mathsf{v} \text{ is gdd. in } e}{\mu\mathsf{v}\ e = e[\mathsf{v} := \mu\mathsf{v}\ e]} \qquad \frac{g = e[\mathsf{v} := g] \quad \mathsf{v} \text{ is gdd. in } e}{g = \mu\mathsf{v}\ e}$$

are a complete axiomatization of behavioural equivalence in (EQ, ϵ).

The processes parametrized framework captures many existing examples of specialized programming languages, like in fig. 3, and even GKAT appears as a fragment of one of these languages.

Furthermore, in [Sch22a], I extend the processes parametrized framework to a setting where a partial order on states is embedded in the coalgebra structure specified by a program (the category $\mathbf{C}$ is the category of partially ordered sets and monotone maps). This captures examples of process calculi in the literature that were not adequately described by the processes parametrized framework, such as Stark and Smolka's probabilistic variation of Milner's algebra of processes in fig. 3 [SS00].

Each of the programming languages covered by the processes parametrized framework has an order-theoretic version, and in the category of partially ordered sets, the behaviours of ordered coalgebras are themselves ordered. I compare the classic notions of *similarity* [HJ04] with the behavioural order and give sufficient conditions on (S, EQ) for the two to coincide. This led me to a characterization of the algebraic theories (S, EQ) for which two-way similarity and bisimilarity to coincide (see §8 of [Sch22a]).

Monad presentations. In the processes parametrized framework, one of the ingredients in the definition of the functor B is a construction of the free algebra satisfying a given algebraic theory. Free algebra constructions are examples of (finitary) monads [Mac88]. Determining whether a particular monad M is a free-algebra construction involves finding a *presentation* for it (see, for example, [BSV19]), a natural

Theory of semilattices (with 0):

$$S = \{0, +\}$$

$$\mathsf{EQ} = \left\{ \begin{array}{c} 0 + x = x \\ x + x = x \\ x + y = y + x \\ x + (y + x) = (x + y) + x \end{array} \right\} \rightsquigarrow$$

Language:

$\mid \mathsf{v}$	$(\mathsf{v} \in \mathsf{Var})$
$\mid \mathsf{a}e_1$	$(\mathsf{a} \in \mathsf{Act})$
$\mid 0$	
$\mid e_1 + e_2$	
$\mid \mu\mathsf{v}\ e$	$(\mathsf{v} \in \mathsf{Var})$

$$\begin{tikzcd} \mu\mathsf{v}\ (\mathsf{av} + \mathsf{abv} + \mathsf{u}) \arrow[loop above, "\mathsf{a}"] \arrow[r, Rightarrow] \arrow[d, bend left, "\mathsf{a}"] & \mathsf{u} \\ \mathsf{b}(\mu\mathsf{v}\ (\mathsf{av} + \mathsf{abv} + \mathsf{u})) \arrow[u, bend left, "\mathsf{b}"] & \end{tikzcd}$$

$$\mathsf{Exp} \to \mathcal{P}(\mathsf{Var} + \mathsf{Act} \times \mathsf{Exp})$$

Figure 3: A concurrent programming language first explored in [Mil84], generated from the algebraic theory of semilattices (with 0) in the processes parametrized framework. Here, $\mathcal{P}X$ is the finitary powerset of X, which constructs the free semilattice (with 0) on X.

algebraic structure on MX that satisfies the necessary universal property. Monad presentations in the classical setting, where algebraic theories consist of equations, exist for all the canonical examples and are well-studied. The ordered setting is a different story, particularly when it comes to probabilities.

In [Sch22b], I provide a number of presentations for monads on the category of partially ordered sets, particularly focusing on free ordered modules and ordered probability distributions. For example, consider the ordered semiring $\mathbb{R}^+$ consisting of nonnegative real numbers.

Definition. A subset $U \subseteq X$ of a partially ordered set $(X, \leq)$ is *upper* if $x \in U$ and $x \leq y$ implies $y \in U$.

Theorem ([Sch22b])**.** *The free ordered $\mathbb{R}^+$-module on a partially ordered set $(X, \leq)$ is the set of finitely supported functions $X \to \mathbb{R}^+$ equipped with the partial order*

$$f_1 \sqsubseteq f_2 \iff (\forall \textit{ upper } U \subseteq X) \sum_{x \in U} f_1(x) \leq \sum_{x \in U} f_2(x)$$

Future Work

Going forward, I will develop general strategies for constructing and axiomatizing programming languages. Moreover, I will take my "algebra of behaviours" perspective out of the realm of programming language theory and apply it in other areas of mathematics.

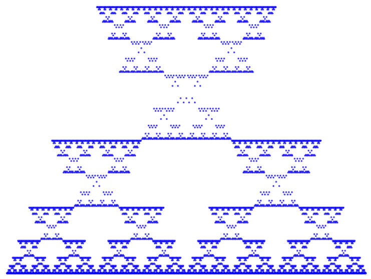
Open problems. Several related completeness problems exist in the literature on programming languages.

Project: The question of whether the axiomatization of GKAT (without the uniqueness axiom) in [Smo+20] is complete remains open. In [SKS22], my coauthors and I were able to reduce the completeness problem for a certain fragment of GKAT to a famous problem posed by Milner [Mil84], which was recently solved [GF20; Gra22]. I will resolve the full completeness conjecture for GKAT by reducing it to Milner's completeness problem in its entirety.

Project: In [Sch+22], my coauthors and I showed that GKAT is an example of a *star fragment*, a generalization of regular expressions suggested by the processes parametrized framework. Examples of star fragments include the GKAT completeness problem and Milner's completeness problem, but they also include new examples, such as regular expressions modelling probabilistic computing tree-search algorithms. To date,

$$\mu \mathsf{v}\ (\mathsf{av} + \mathsf{bv} + \mathsf{dcv})$$

$$\boxed{\begin{aligned} X_1 &= \mathsf{a}(X_2) \cup \mathsf{b}(X_2) \cup \mathsf{d}(X_1) \\ X_2 &= \mathsf{c}(X_1) \end{aligned}}$$



$$\begin{aligned} \mathsf{a}(x,y) &= (x/2 - 1/2, y/2 - \sqrt{3}/4) \\ \mathsf{b}(x,y) &= (x/2 + 1/2, y/2 - \sqrt{3}/4) \\ \mathsf{c}(x,y) &= (x/2, y/2 + \sqrt{3}/4) \\ \mathsf{d}(x,y) &= (-x, -y) \end{aligned}$$

Figure 4: A Sierpinski gasket with a recursively rotated triangle. Generated by a recipe specified by a term in a process calculus, where the actions have been interpreted as endomaps on $\mathbb{R}^2$.

the only star fragment with a solved completeness problem is Milner's [Mil84; Gra22]. I will investigate the completeness problems of probabilistic star fragments, as they provide intuitive syntaxes and algebraic reasoning tools for Markov chains/decision processes and other models of probabilistic computing.

Fractal semantics. A surprising application area of coalgebra is in the geometry of fractal or self-similar sets. Many famous examples of fractal sets can be characterized as final F-coalgebras on some category [Fre08; Bha+14; Rat+21; NM21]. A systematic approach to realizing fractals as final coalgebras involves writing them down as solutions to *recursive program specifications*, systems of equations such as

$$\begin{aligned} X_1 &= t_1(X_1, \dots, X_n, Y_1, \dots, Y_m) \\ &\vdots \\ X_n &= t_n(X_1, \dots, X_n, Y_1, \dots, Y_m) \end{aligned}$$

for terms $t_1, \dots, t_n$ in a free algebra [MM09b; Lei11]. Recursive program specifications give recipes for the construction of self-similar sets as in fig. 4. Recursive program specifications generalize the *iterated function systems* of Hutchinson [Hut81], which are operators on the space of nonempty compact subsets of a complete metric space M of the form

$$X \longmapsto \sigma_1(X) \cup \cdots \cup \sigma_n(X)$$

where each σ_i is a contraction on M, from "one-variable recipes" to "n-variable recipes".

Process calculi provide syntaxes for recursive specifications. Thus, a term in a process calculus is a recipe for constructing self-similar sets. As we can see from fig. 4, there are pairs of terms that are *fractal equivalent*, meaning they produce equivalent recipes. Axiomatizing fractal equivalence would give an algebraic structure to fractal recipes, deriving equivalences between fractal sets without having to construct isometries or homeomorphisms explicitly.

Project: In collaboration with Lawrence Moss and Victoria Noquez, I am working to axiomatize fractal equivalence for Milner's process calculus (obtained from the theory of semilattices with 0–see fig. 3), which has been employed in fig. 4. Our starting point is an observation due to Moss and Milius [MM09a], that typical axioms for recursive program specifications (like those in [Hur+98]) are sound with respect to fractal equivalence.

Project: Other process calculi given by the processes parametrized framework [Sch+22] allow for the stochastic specification of self-similar sets and the specification of self-similar sets that are constructed in more complex ways (like through gluings [Lei11]). I will investigate fractal equivalence in these settings.

More processes parametrized. At present, the processes parametrized framework can produce a programming language for specifying coalgebras in the category of sets from an algebraic theory, as well as coalgebras in the category of partially ordered sets from an ordered algebraic theory. These are only two of many base categories that appear in coalgebraic models of computation.

Project: I will extend the processes parametrized framework to be able to handle other algebraic theories, particularly the metric theories of Mardare, Panagaden, and Plotkin [MPP21]. A metric version of the processes parametrized framework produces coalgebras with a metric structure. This would capture programs written with a standard notion of *behavioural distance* in mind [Bal+18]. More generally, I will develop a general recipe for producing processes parametrized-like frameworks in other categories using monads and monad presentations.

Broader applications. The processes parametrized framework is a general enough framework to find applications in areas outside of computer science and mathematics. Processes described by the framework have uninterpreted *actions* that can be replaced by physical events like chemical reactions or biophysical events [Plo13; PP08]. I am currently working with an undergraduate student at Ryerson University to implement an interpretation of the process calculus that is useful in media arts applications, specifically generative art. We use the framework to compose generative musical pieces and animations, and we expect apply it also to recursive image filtering and nature simulations. Research axiomatizing process equivalences that track any one of these interpretations would be fascinating interdisciplinary work touching on mathematics, computer science, and fine arts.

Conclusion

Theoretical computer science is a rapidly evolving field that reaches in many directions and is in a constant state of outgrowing its foundations. New applications demand novel mathematical formalisms, and the situation is always better if the new formalism can be incorporated into existing theories. General theoretical tools like universal algebra and coalgebra have made an important impact in computer science for this reason, and I am excited to be a part of the mathematical developments in this youthful area.

References

[Bal+18] Baldan, P. "Coalgebraic Behavioral Metrics". In: *Log. Methods Comput. Sci.* 14.3 (2018). DOI: 10.23638/LMCS-14(3:20)2018. URL: https://doi.org/10.23638/LMCS-14(3:20)2018.

[BCR15] Ballester-Bolinches, A., Cosme-Llópez, E., Rutten, J. J. M. M., "The dual equivalence of equations and coequations for automata". In: *Inf. Comput.* 244 (2015), pp. 49–75. DOI: 10.1016/j.ic.2015.08.001. URL: https://doi.org/10.1016/j.ic.2015.08.001.

[Bha+14] Bhattacharya, P. "Fractal Sets as Final Coalgebras Obtained by Completing an Initial Algebra". In: *Horizons of the Mind. A Tribute to Prakash Panangaden - Essays Dedicated to Prakash Panangaden on the Occasion of His 60th Birthday*. Ed. by Franck van Breugel et al. Vol. 8464. Lecture Notes in Computer Science. Springer, 2014, pp. 146–167. DOI: 10.1007/978-3-319-06880-0_7. URL: https://doi.org/10.1007/978-3-319-06880-0%5C_7.

[BSV19] Bonchi, F., Sokolova, A., Vignudelli, V., “The Theory of Traces for Systems with Nondeterminism and Probability”. In: *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*. IEEE, 2019, pp. 1–14. DOI: 10.1109/LICS.2019.8785673. URL: https://doi.org/10.1109/LICS.2019.8785673.

[Cir+11] Cirstea, C. “Modal Logics are Coalgebraic”. In: *Comput. J.* 54.1 (2011), pp. 31–41. DOI: 10.1093/comjnl/bxp004. URL: https://doi.org/10.1093/comjnl/bxp004.

[Coh81] Cohn, P. M. *Universal Algebra*. Springer Dordrecht, 1981. ISBN: 9789027712134. DOI: 10.1007/978-94-009-8399-1.

[Fre08] Freyd, P. “Algebraic real analysis”. In: *Theory and Applications of Categories [electronic only]* 20 (2008), pp. 215–306.

[GF20] Grabmayer, C., Fokkink, W. J., “A Complete Proof System for 1-Free Regular Expressions Modulo Bisimilarity”. In: *LICS ’20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*. Ed. by Holger Hermanns et al. ACM, 2020, pp. 465–478. DOI: 10.1145/3373718.3394744. URL: https://doi.org/10.1145/3373718.3394744.

[Gon+22] Goncharov, S. *Towards a Higher-Order Mathematical Operational Semantics*. 2022. DOI: 10.48550/ARXIV.2210.13387. URL: https://arxiv.org/abs/2210.13387.

[Gra22] Grabmayer, C. A. “Milner’s Proof System for Regular Expressions Modulo Bisimilarity is Complete: Crystallization: Near-Collapsing Process Graph Interpretations of Regular Expressions”. In: *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS ’22. Haifa, Israel: Association for Computing Machinery, 2022. ISBN: 9781450393515. DOI: 10.1145/3531130.3532430. URL: https://doi.org/10.1145/3531130.3532430.

[HJ04] Hughes, J., Jacobs, B., “Simulations in coalgebra”. In: *Theor. Comput. Sci.* 327.1-2 (2004), pp. 71–108. DOI: 10.1016/j.tcs.2004.07.022. URL: https://doi.org/10.1016/j.tcs.2004.07.022.

[Hur+98] Hurkens, A. J. C. “The logic of recursive equations”. In: *The Journal of Symbolic Logic* 63.2 (1998), pp. 451–478.

[Hut81] Hutchinson, J. E. “Fractals and Self Similarity”. In: *Indiana University Mathematics Journal* 30.5 (1981), pp. 713–747. ISSN: 00222518, 19435258. URL: http://www.jstor.org/stable/24893080 (visited on 10/31/2022).

[Jac06] Jacobs, B. “A Bialgebraic Review of Deterministic Automata, Regular Expressions and Languages”. In: *Algebra, Meaning, and Computation, Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday*. Ed. by Kokichi Futatsugi, Jean-Pierre Jouannaud, and José Meseguer. Vol. 4060. Lecture Notes in Computer Science. Springer, 2006, pp. 375–404. DOI: 10.1007/11780274_20.

[Jac11] Jacobs, B. “Coalgebraic Walks, in Quantum and Turing Computation”. In: *Foundations of Software Science and Computational Structures - 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*. Ed. by Martin Hofmann. Vol. 6604. Lecture Notes in Computer Science. Springer, 2011, pp. 12–26. DOI: 10.1007/978-3-642-19805-2_2. URL: https://doi.org/10.1007/978-3-642-19805-2%5C_2.

[Kle51] Kleene, S. “Representation of Events in Nerve Nets and Finite Automata”. In: 1951.

[KT08] Kozen, D., Tseng, W. D., "The Böhm-Jacopini Theorem Is False, Propositionally". In: *Mathematics of Program Construction, 9th International Conference, MPC 2008, Marseille, France, July 15-18, 2008. Proceedings*. Ed. by Philippe Audebaud and Christine Paulin-Mohring. Vol. 5133. Lecture Notes in Computer Science. Springer, 2008, pp. 177–192. DOI: 10.1007/978-3-540-70594-9_11. URL: https://doi.org/10.1007/978-3-540-70594-9%5C_11.

[Lei11] Leinster, T. "A general theory of self-similarity". In: *Advances in Mathematics* 226.4 (2011), pp. 2935–3017.

[Mac88] MacLane, S. "Categories for the working mathematician. 4th corrected printing". In: *Graduate texts in mathematics* 5 (1988).

[Mil10] Milius, S. "A Sound and Complete Calculus for Finite Stream Circuits". In: *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*. IEEE Computer Society, 2010, pp. 421–430. DOI: 10.1109/LICS.2010.11.

[Mil80] Milner, R. *A Calculus of Communicating Systems*. Vol. 92. Lecture Notes in Computer Science. Springer, 1980. ISBN: 3-540-10235-3. DOI: 10.1007/3-540-10235-3. URL: https://doi.org/10.1007/3-540-10235-3.

[Mil84] Milner, R. "A Complete Inference System for a Class of Regular Behaviours". In: *J. Comput. Syst. Sci.* 28.3 (1984), pp. 439–466. DOI: 10.1016/0022-0000(84)90023-0. URL: https://doi.org/10.1016/0022-0000(84)90023-0.

[MM09a] Milius, S., Moss, L. S., "Equational properties of recursive program scheme solutions". In: *Cahiers de topologie et géométrie différentielle catégoriques* 50.1 (2009), pp. 23–66.

[MM09b] Milius, S., Moss, L. S., "The Category Theoretic Solution of Recursive Program Schemes". In: (2009). DOI: 10.48550/ARXIV.0904.2385. URL: https://arxiv.org/abs/0904.2385.

[MPP21] Mardare, R., Panangaden, P., Plotkin, G., "Fixed-Points for Quantitative Equational Logics". In: *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)* (June 2021). DOI: 10.1109/lics52264.2021.9470662. URL: http://dx.doi.org/10.1109/LICS52264.2021.9470662.

[NM21] Noquez, V., Moss, L. S., *The Sierpinski Carpet as a Final Coalgebra*. 2021. DOI: 10.48550/ARXIV.2110.06404. URL: https://arxiv.org/abs/2110.06404.

[Plo13] Plotkin, G. D. "A calculus of chemical systems". In: *In search of elegance in the theory and practice of computation*. Springer, 2013, pp. 445–465.

[PP08] Pedersen, M., Plotkin, G., "A language for biochemical systems". In: *International Conference on Computational Methods in Systems Biology*. Springer. 2008, pp. 63–82.

[Rat+21] Ratnayake, J. *Presenting the Sierpinski Gasket in Various Categories of Metric Spaces*. 2021. DOI: 10.48550/ARXIV.2110.06916. URL: https://arxiv.org/abs/2110.06916.

[Rut00] Rutten, J. J. M. M. "Universal coalgebra: a theory of systems". In: *Theor. Comput. Sci.* 249.1 (2000), pp. 3–80. DOI: 10.1016/S0304-3975(00)00056-6. URL: https://doi.org/10.1016/S0304-3975(00)00056-6.

[Rut98] Rutten, J. J. M. M. "Automata and Coinduction (An Exercise in Coalgebra)". In: *CONCUR '98: Concurrency Theory, 9th International Conference, Nice, France, September 8-11, 1998, Proceedings*. Ed. by Davide Sangiorgi and Robert de Simone. Vol. 1466. Lecture Notes in Computer Science. Springer, 1998, pp. 194–218. DOI: 10.1007/BFb0055624.

[Sal66] Salomaa, A. “Two Complete Axiom Systems for the Algebra of Regular Events”. In: *J. ACM* 13.1 (1966), pp. 158–169. DOI: 10.1145/321312.321326. URL: https://doi.org/10.1145/321312.321326.

[Sch+21] **Schmid, Todd,** “Guarded Kleene Algebra with Tests: Coequations, Coinduction, and Completeness”. In: *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*. Ed. by Nikhil Bansal, Emanuela Merelli, and James Worrell. Vol. 198. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 142:1–142:14. DOI: 10.4230/LIPIcs.ICALP.2021.142. URL: https://doi.org/10.4230/LIPIcs.ICALP.2021.142.

[Sch+22] **Schmid, Todd,** “Processes Parametrised by an Algebraic Theory”. In: LIPIcs 229 (2022). Ed. by Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, 132:1–132:20. DOI: 10.4230/LIPIcs.ICALP.2022.132. URL: https://doi.org/10.4230/LIPIcs.ICALP.2022.132.

[Sch22a] **Schmid, Todd,** “A (Co)Algebraic Framework for Ordered Processes”. In: *CoRR* abs/2209.00634 (2022). DOI: 10.48550/arXiv.2209.00634. arXiv: 2209.00634. URL: https://doi.org/10.48550/arXiv.2209.00634.

[Sch22b] **Schmid, Todd,** “Presenting with Quantitative Inequational Theories”. In: *CoRR* abs/2207.11629 (2022). DOI: 10.48550/arXiv.2207.11629. arXiv: 2207.11629. URL: https://doi.org/10.48550/arXiv.2207.11629.

[Sil+13] Silva, A. “Generalizing determinization from automata to coalgebras”. In: *Logical Methods in Computer Science* 9.1 (Mar. 2013). Ed. by Prakash Panangaden. DOI: 10.2168/lmcs-9(1:9)2013. URL: https://doi.org/10.2168%2Flmcs-9%281%3A9%292013.

[Sil10] Silva, A. “Kleene coalgebra”. PhD thesis. University of Nijmegen, 2010.

[SKS22] **Schmid, Todd,** Kappé, T., Silva, A., *A Complete Inference System for Skip-free Guarded Kleene Algebra with Tests*. Forthcoming. 2022.

[Smo+20] Smolka, S. “Guarded Kleene algebra with tests: verification of uninterpreted programs in nearly linear time”. In: *Proc. ACM Program. Lang.* 4.POPL (2020), 61:1–61:28. DOI: 10.1145/3371129. URL: https://doi.org/10.1145/3371129.

[SRS21] **Schmid, Todd,** Rot, J., Silva, A., “On Star Expressions and Coalgebraic Completeness Theorems”. In: *Proceedings 37th Conference on Mathematical Foundations of Programming Semantics, MFPS 2021, Hybrid: Salzburg, Austria and Online, 30th August - 2nd September, 2021*. Ed. by Ana Sokolova. Vol. 351. EPTCS. 2021, pp. 242–259. DOI: 10.4204/EPTCS.351.15. URL: https://doi.org/10.4204/EPTCS.351.15.

[SS00] Stark, E. W., Smolka, S. A., “A complete axiom system for finite-state probabilistic processes”. In: *Proof, Language, and Interaction, Essays in Honour of Robin Milner*. Ed. by Gordon D. Plotkin, Colin Stirling, and Mads Tofte. The MIT Press, 2000, pp. 571–596.