Monoidal Streams for Dataflow Programming

Extended abstract

Elena Di Lavore
Tallinn University of Technology

Giovanni de Felice
University of Oxford
Quantinuum

Mario Román
Tallinn University of Technology

Abstract

We introduce monoidal streams: a generalization of causal stream functions to monoidal categories. In the same way that streams provide semantics to dataflow programming with pure functions, monoidal streams provide semantics to dataflow programming with theories of processes represented by a symmetric monoidal category. At the same time, monoidal streams form a feedback monoidal category, which can be used to interpret signal flow graphs. As an example, we study a stochastic dataflow language.

This is an extended abstract of "Monoidal Streams for Dataflow Programming", to appear in LiCS '22.

Keywords: Monoidal stream, Stream, Monoidal category, Dataflow programming, Feedback, Signal flow graph, Coalgebra, Stochastic process.

1 Introuction

Dataflow languages. Dataflow (or *stream-based*) programming languages, such as Lucid [11, 22], follow a paradigm in which every declaration represents an infinite list of values: a *stream* [2, 20]. The following program in a Lucid-like language (Figure 1) computes the Fibonacci sequence, thanks to a Fby ("followed by") operator.

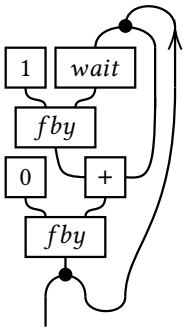
$$fib = 0 \text{ Fby } (fib + (1 \text{ Fby } \text{Wait}(fib)))$$

Figure 1. *The Fibonacci sequence is 0 followed by the Fibonacci sequence plus the Fibonacci sequence preceded by a 1.*

The control structure of dataflow programs is inspired by *signal flow graphs* [2, 15, 18]. Signal flow graphs are diagrammatic specifications of processes with feedback loops, widely used in control system engineering. In a dataflow program, feedback loops represent how the current value of a stream may depend on its previous values. For instance, the previous program (Figure 1) corresponds to the signal flow graph in Figure 2.

Feedback monoidal categories. Signal flow graphs are the graphical syntax for *feedback monoidal categories* [5, 6, 9, 10, 14]: they are the *string diagrams* for any monoidal theory, extended with *feedback*.

Yet, semantics of dataflow languages have been mostly restricted to theories of pure functions [2, 7, 8, 16, 19, 21]: what are called *cartesian* monoidal categories. We claim that this restriction is actually inessential; dataflow programs may take semantics in non-cartesian monoidal categories, exactly as their signal flow graphs do.



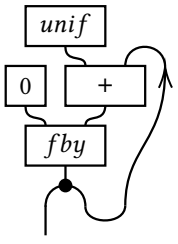
$$\begin{aligned}\text{fib} = \textbf{fbk}(&\text{copy};\\ &\partial(1 \times \text{wait}) \times \text{id};\\ &\partial(\text{fby}) \times \text{id};\\ &\partial(+);\\ &0 \times \text{id};\\ &\text{fby};\\ &\text{copy})\end{aligned}$$

Figure 2. Fibonacci: signal flow graph and morphism.

Contributions. The present work provides this missing semantics: we construct *monoidal streams* (Theorem 2.3) over a symmetric monoidal category, which form a *feedback monoidal category* (Theorem 3.3). Monoidal streams model the values of a *monoidal dataflow language*, in the same way that streams model the values of a classical dataflow language. This opens the door to stochastic, effectful, or quantum dataflow languages. In particular, we give semantics and string diagrams for a *stochastic dataflow programming language*, where the following code can be run.

$$walk = 0 \text{ Fby } (\text{Uniform}(-1, 1) + walk)$$

Figure 3. *A stochastic dataflow program. A random walk is 0 followed by the random walk plus a stochastic stream of steps to the left (-1) or to the right (1), sampled uniformly.*



$$\begin{aligned}\text{walk} = \textbf{fbk}(&\\ &\partial(\text{unif}) \otimes \text{id};\\ &0 \otimes \partial(+);\\ &\text{fby};\\ &\text{copy})\end{aligned}$$

Figure 4. Random walk: signal flow graph and morphism.

2 Monoidal Streams

The manuscript contains three main definitions in terms of universal properties: *intensional [19], extensional and observational monoidal streams* (Figure 5). The latter are our

definitive notion of streams and we refer to them as simply *monoidal streams*. The present extended abstract directly introduces the coinductive definition of monoidal streams (Definition 2.4) and their explicit construction: *observational sequences* (Definition 2.2).

Monoidal streams. Classically, type-variant streams have a neat coinductive definition [12, 17] that says:

> *"A stream of type $\mathbb{A} = (A_0, A_1, \dots)$ is an element of A_0 together with a stream of type $\mathbb{A}^+ = (A_1, A_2, \dots)$".*

Formally, streams are the final fixpoint of the equation

$$\mathrm{S}(A_0, A_1, \dots) \cong A_0 \times \mathrm{S}(A_1, A_2, \dots);$$

and this fixpoint is computed to be $\mathrm{S}(\mathbb{A}) = \prod_{n\in\mathbb{N}}^{\infty} A_n$.

In the same vein, we want to introduce not only streams but *stream processes* over a fixed theory of processes.

> *"A stream process from $\mathbb{X} = (X_0, X_1, \dots)$ to $\mathbb{Y} = (Y_0, Y_1, \dots)$ is a process from X_0 to Y_0 communicating along a channel M with a stream process from $\mathbb{X}^+ = (X_1, X_2, \dots)$ to $\mathbb{Y}^+ = (Y_1, Y_2, \dots)$."*

Streams are recovered as stream processes on an empty input, so we take this more general slogan as our definition of *monoidal stream* (in Theorem 2.3). Formally, they are the final fixpoint of the equation in Figure 5.

$$\mathbf{Q}(\mathbb{X}, \mathbb{Y}) \cong \int^{M\in\mathrm{C}} \hom(X_0, M \otimes Y_0) \times \mathbf{Q}(M \cdot \mathbb{X}^+, \mathbb{Y}^+).$$

Figure 5. Fixpoint equation for monoidal streams.

Remark 2.1 (Notation). Let $\mathbb{X} \in [\mathbb{N}, \mathrm{C}]$ be a sequence of objects $(X_0, X_1, \dots)$. We write $\mathbb{X}^+$ for its *tail* $(X_1, X_2, \dots)$. Given $M \in \mathrm{C}$, we write $M \cdot \mathbb{X}$ for the sequence $(M \otimes X_0, X_1, X_2, \dots)$; As a consequence, we write $M \cdot \mathbb{X}^+$ for $(M \otimes X_1, X_2, X_3, \dots)$.

Definition 2.2 (Observational sequence)**.** The set of observational sequences is

$$\mathbf{Obs}(\mathbb{X}, \mathbb{Y}) \cong \left(\int^{M\in[\mathbb{N},\mathrm{C}]} \prod_{i=0}^{\infty} \hom(M_{i-1} \otimes X_i, M_i \otimes Y_i) \right) \Big/ \approx$$

where $(\approx)$ is observational equivalence.

Theorem 2.3. *In a productive category, the final fixpoint of the equation in Figure 5, which is the set of* monoidal streams*, coincides with the set of observational sequences.*

The final fixpoint of a functor does not need to exist in general. However, when C satisfies some extra conditions, which we call *productivity*, the final fixpoint does exist and can be computed by Adámek's theorem [1]. This allows us to recast the definition of monoidal streams in coinductive terms.

Definition 2.4 (Monoidal stream)**.** A *monoidal stream* $f \in \mathrm{Stream}(\mathbb{X}, \mathbb{Y})$ is a triple consisting of

- $M(f) \in \mathrm{Obj}(\mathrm{C})$, the *memory*,
- $\mathrm{now}(f) \in \hom(X_0, M(f) \otimes Y_0)$, the *first action*,
- $\mathrm{later}(f) \in \mathrm{Stream}(M(f) \cdot \mathbb{X}^+, \mathbb{Y}^+)$, the *rest of the action*,

quotiented by dinaturality in M.

Explicitly, monoidal streams are quotiented by the equivalence relation $f \sim g$ generated by

- the existence of $r \colon M(g) \to M(f)$,
- such that $\mathrm{now}(f) = \mathrm{now}(g); r$,
- and such that $r \cdot \mathrm{later}(f) \sim \mathrm{later}(g)$.

Here, $r \cdot \mathrm{later}(f) \in \mathrm{Stream}(M(g) \cdot \mathbb{X}^+, \mathbb{Y}^+)$ is obtained by precomposition of the first action of $\mathrm{later}(f)$ with r.

3 Delayed feedback for streams

Monoidal streams form a *feedback* monoidal category with respect to the "delay functor" $\partial \colon [\mathbb{N}, \mathrm{C}] \to [\mathbb{N}, \mathrm{C}]$.

Definition 3.1 (Delay functor)**.** Let $\partial \colon [\mathbb{N}, \mathrm{C}] \to [\mathbb{N}, \mathrm{C}]$ be the endofunctor defined on objects $\mathbb{X} = (X_0, X_1, \dots)$, as $\partial(\mathbb{X}) = (I, X_0, X_1, \dots)$; and on morphisms $\mathbb{f} = (f_0, f_1, \dots)$ as $\partial(\mathbb{f}) = (\mathrm{id}_I, f_0, f_1, \dots)$.

Given some stream in $\mathrm{Stream}(\partial\mathbb{S} \otimes \mathbb{X}, \mathbb{S} \otimes \mathbb{Y})$, we can create a new stream in $\mathrm{Stream}(\mathbb{X}, \mathbb{Y})$ that passes the output in $\mathbb{S}$ as a memory channel that gets used as the input in $\partial\mathbb{S}$. As a consequence, the category of monoidal streams has a graphical calculus given by that of feedback monoidal categories [3, 4, 9, 13, 14]. This graphical calculus is complete for extensional equivalence.

Definition 3.2. A *feedback monoidal category* is a symmetric monoidal category $(\mathrm{C}, \otimes, I)$ endowed with a monoidal endofunctor $\mathbf{F} \colon \mathrm{C} \to \mathrm{C}$ and an operation

$$\mathbf{fbk}_S \colon \hom(\mathbf{F}(S) \otimes X, S \otimes Y) \to \hom(X, Y)$$

for all S, X and Y objects of C; this operation needs to satisfy the following axioms.

(A1). Tightening: $u;\mathbf{fbk}_S(f);v = \mathbf{fbk}_S((\mathrm{id}_{FS} \otimes u);f;(\mathrm{id}_S \otimes v))$.
(A2). Vanishing: $\mathbf{fbk}_I(f) = f$.
(A3). Joining: $\mathbf{fbk}_T(\mathbf{fbk}_S(f)) = \mathbf{fbk}_{S\otimes T}(f)$
(A4). Strength: $\mathbf{fbk}_S(f) \otimes g = \mathbf{fbk}_S(f \otimes g)$.
(A5). Sliding: $\mathbf{fbk}_S((\mathbf{F}h \otimes \mathrm{id}_X)\,;f) = \mathbf{fbk}_T(f\,;(h \otimes \mathrm{id}_Y))$.

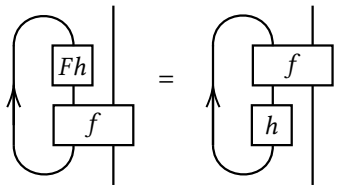


Figure 6. The sliding axiom (A5).

Theorem 3.3. *Monoidal streams over a symmetric monoidal category form a ∂-feedback monoidal category.*

Acknowledgments

Elena Di Lavore and Mario Román were supported by the ESF funded Estonian IT Academy research measure (project 2014-2020.4.05.19-0001) and the Estonian Research Council grant PRG1210. We thank Paweł Sobociński, Edward Morehouse, Fosco Loregian, Niels Voorneveld, George Kaye, Chad Nester, and the reviewers for an earlier version presented at NWPT'21.

References

[1] Jiří Adámek. Free algebras and automata realizations in the language of categories. *Commentationes Mathematicae Universitatis Carolinae*, 015(4):589–602, 1974. URL: http://eudml.org/doc/16649.

[2] Albert Benveniste, Paul Caspi, Paul Le Guernic, and Nicolas Halbwachs. Data-flow synchronous languages. In J. W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg, editors, *A Decade of Concurrency, Reflections and Perspectives, REX School/Symposium, Noordwijkerhout, The Netherlands, June 1-4, 1993, Proceedings*, volume 803 of *Lecture Notes in Computer Science*, pages 1–45. Springer, 1993. doi:10.1007/3-540-58043-3_16.

[3] Stephen L. Bloom and Zoltán Ésik. *Iteration Theories - The Equational Logic of Iterative Processes*. EATCS Monographs on Theoretical Computer Science. Springer, 1993. doi:10.1007/978-3-642-78034-9.

[4] Filippo Bonchi, Joshua Holland, Robin Piedeleu, Paweł Sobociński, and Fabio Zanasi. Diagrammatic algebra: from linear to concurrent systems. *Proc. ACM Program. Lang.*, 3(POPL):25:1–25:28, 2019. doi:10.1145/3290338.

[5] Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. A categorical semantics of signal flow graphs. In *International Conference on Concurrency Theory*, pages 435–450. Springer, 2014.

[6] Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. Full abstraction for signal flow graphs. *ACM SIGPLAN Notices*, 50(1):515–526, 2015.

[7] Patrick Cousot. Syntactic and semantic soundness of structural dataflow analysis. In Bor-Yuh Evan Chang, editor, *Static Analysis - 26th International Symposium, SAS 2019, Porto, Portugal, October 8-11, 2019, Proceedings*, volume 11822 of *Lecture Notes in Computer Science*, pages 96–117. Springer, 2019. doi:10.1007/978-3-030-32304-2_6.

[8] Antonin Delpeuch. A complete language for faceted dataflow programs. In John Baez and Bob Coecke, editors, *Proceedings Applied Category Theory 2019, ACT 2019, University of Oxford, UK, 15-19 July 2019*, volume 323 of *EPTCS*, pages 1–14, 2019. doi:10.4204/EPTCS.323.1.

[9] Elena Di Lavore, Alessandro Gianola, Mario Román, Nicoletta Sabadini, and Paweł Sobociński. A canonical algebra of open transition systems. In Gwen Salaün and Anton Wijs, editors, *Formal Aspects of Component Software*, pages 63–81, Cham, 2021. Springer International Publishing.

[10] Dan R. Ghica, George Kaye, and David Sprunger. Full abstraction for digital circuits, 2022. arXiv:2201.10456.

[11] Nicolas Halbwachs, Fabienne Lagnier, and Christophe Ratel. Programming and verifying real-time systems by means of the synchronous data-flow language LUSTRE. *IEEE Trans. Software Eng.*, 18(9):785–793, 1992. doi:10.1109/32.159839.

[12] Bart Jacobs. *Introduction to Coalgebra: Towards Mathematics of States and Observation*, volume 59 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2016. doi:10.1017/CBO9781316823187.

[13] Piergiulio Katis, Nicoletta Sabadini, and Robert F. C. Walters. On the algebra of feedback and systems with boundary. In *Rendiconti del Seminario Matematico di Palermo*, 1999.

[14] Piergiulio Katis, Nicoletta Sabadini, and Robert F. C. Walters. Feedback, trace and fixed-point semantics. *RAIRO-Theor. Informatics Appl.*, 36(2):181–194, 2002. doi:10.1051/ita:2002009.

[15] S. J. Mason. Feedback Theory - Some properties of signal flow graphs. *Proceedings of the Institute of Radio Engineers*, 41(9):1144–1156, 1953. doi:10.1109/JRPROC.1953.274449.

[16] José Nuno Oliveira. *The formal semantics of deterministic dataflow programs*. PhD thesis, University of Manchester, UK, 1984. URL: http://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.376586.

[17] Jan J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000. doi:10.1016/S0304-3975(00)00056-6.

[18] Claude E. Shannon. *The Theory and Design of Linear Differential Equation Machines*. Bell Telephone Laboratories, 1942.

[19] David Sprunger and Shin-ya Katsumata. Differentiable causal computations via delayed trace. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–12. IEEE, 2019. doi:10.1109/LICS.2019.8785670.

[20] Tarmo Uustalu and Varmo Vene. The essence of dataflow programming. In Kwangkeun Yi, editor, *Programming Languages and Systems, Third Asian Symposium, APLAS 2005, Tsukuba, Japan, November 2-5, 2005, Proceedings*, volume 3780 of *Lecture Notes in Computer Science*, pages 2–18. Springer, 2005. doi:10.1007/11575467_2.

[21] Tarmo Uustalu and Varmo Vene. Comonadic notions of computation. In Jiří Adámek and Clemens Kupke, editors, *Proceedings of the Ninth Workshop on Coalgebraic Methods in Computer Science, CMCS 2008, Budapest, Hungary, April 4-6, 2008*, volume 203 of *Electronic Notes in Theoretical Computer Science*, pages 263–284. Elsevier, 2008. doi:10.1016/j.entcs.2008.05.029.

[22] William W Wadge, Edward A Ashcroft, et al. *Lucid, the dataflow programming language*, volume 303. Academic Press London, 1985.