Monoidal Streams and Probabilistic Dataflow with DisCoPy

Alexis Toumi Richie Yeung Boldizsár Poór Giovanni de Felice
Quantinuum – Quantum Compositional Intelligence
17 Beaumont street, OX1 2NA Oxford, UK
firstname@discopy.org

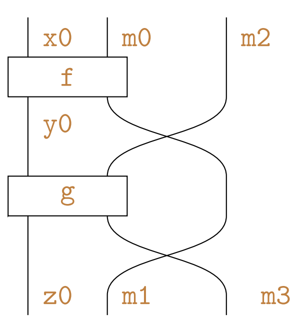
Monoidal streams give semantics to signal flow diagrams over any base monoidal category. Over the category of functions, they give rise to causal stream functions used in dataflow programming. Over categories of probabilistic functions, they give rise to different kinds of controlled stochastic processes. Here, we report on an implementation of monoidal streams in DisCoPy, the Python library for computing with string diagrams. The tool allows to specify dataflow programs in the syntax of Markov categories with delayed feedback. It then performs the semantic evaluation of these programs by unrolling their time-evolution in the chosen semantic category. Example uses include computing the Fibonacci sequence, sampling from a random walk or simulating quantum channels with memory.

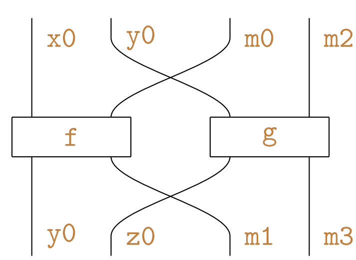
Extended Abstract

DisCoPy [3] is a Python toolkit for computing with *string diagrams*, a graphical calculus for representing sequential and parallel composition of processes. The library allows for string diagrams to be defined either as algebraic formulae or as Python programs. They can then be plotted, compared, rewritten and evaluated as code, be it for a quantum circuit, a probabilistic program or a neural network. In many cases however, we are not interested in one fixed process but in a family of processes, e.g. circuits indexed by the size of their input or communication protocols indexed by a discrete time step.

Monoidal streams [4] are infinite families of processes where each process may depend on the output of the previous one. Formally, for a symmetric monoidal category $\mathbf{C}$ and three countable sequences of objects $X, Y, Z \in \mathrm{Ob}(\mathbf{C})^{\mathbb{N}}$ we define $\mathbf{Stream}(\mathbf{C})(X,Y,Z) = \mathbf{C}(X_0 \otimes Z_0, Y_0 \otimes Z_1) \times \mathbf{Stream}(\mathbf{C})(X^+, Y^+, Z^+)$ i.e. a monoidal stream[1] $f : X \to Y$ with $\texttt{memory}(f) = Z$ is a process $\texttt{now}(f) : X_0 \times Z_0 \to Y_0 \times Z_1$ and a monoidal stream $\texttt{later}(f) : X^+ \to Y^+$ with $\texttt{memory}(\texttt{later}(f)) = Z^+ = (Z_1, Z_2, \dots)$. This gives a symmetric monoidal category $\mathbf{Stream}(\mathbf{C})(X,Y) = \coprod_{Z \in \mathrm{Ob}(C)^{\mathbb{N}}} \mathbf{Stream}(\mathbf{C})(X,Y,Z)$ where:

- $X \otimes Y = (X_0 \otimes Y_0, X_1 \otimes Y_1, \dots)$ and $\texttt{memory}(f \circ g) = \texttt{memory}(f \otimes g) = \texttt{memory}(f) \otimes \texttt{memory}(g)$
- $\texttt{now}(f \circ g)$ and $\texttt{now}(f \otimes g)$ are given by the following composition in $\mathbf{C}$:



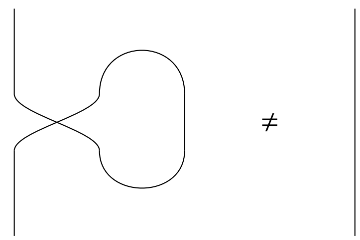


[1] Here we define *intentional* streams which can later be quotiented by *extensional* and *observational* equivalence.

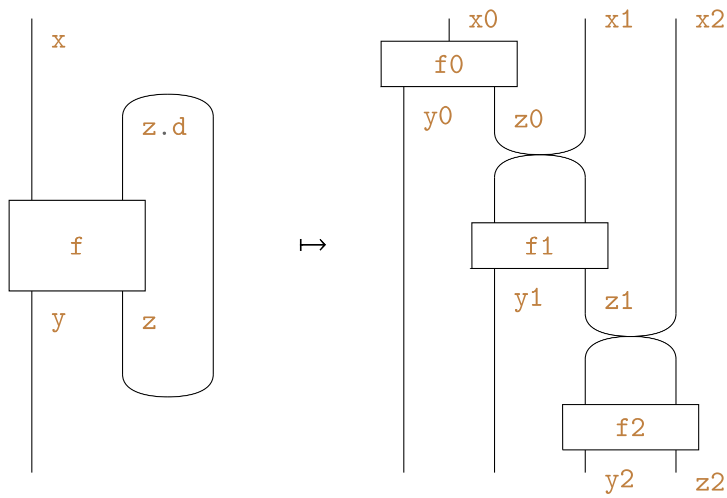
Submitted to:
ACT 2023

- $\texttt{later}(f \circ g) = \texttt{later}(f) \circ \texttt{later}(g)$ and $\texttt{later}(f \otimes g) = \texttt{later}(f) \otimes \texttt{later}(g)$

The category of monoidal streams comes with a *delayed feedback*, [6] i.e. an endofunctor δ called `delay` and an operation from $f : X \otimes \delta(Z) \to Y \otimes Z$ to $\texttt{feedback}_Z(f) : X \to Y$ which satisfies all the axioms of a traced monoidal category [5] except *yanking*, i.e. in general the feedback of a swap is not the identity.



Thus, we can take monoidal functors from the free category with delayed feedback over a monoidal signature (where the morphisms are string diagrams with feedback loops) to the category of monoidal streams. In effect, we are unrolling the feedback loop:



When we take $\mathbf{C} = \mathbf{Set}$ the category of sets and functions, we get an implementation of *dataflow programming*. For instance, in Appendix A we implement the Fibonacci sequence as a feedback diagram together with a functor into the category of monoidal streams of Python functions. DisCoPy implements *linear type systems* inside of Python which follow the hierarchy of graphical languages for monoidal categories [8]. That is, it can either prevent variable copying and deleting or make it explicit as structural morphisms in a *copy-discard category* [2]. In particular, this allows to implement streams of probabilistic processes e.g. in $\mathbf{C} = \mathbf{Stoch}$ the category of measurable spaces and Markov kernels. This gives an implementation of *probabilistic dataflow programming* which we showcase in Appendix B.

Another natural application would be to build upon the quantum computing features of DisCoPy [7] to implement quantum dataflow programming [1]. We also plan to develop heuristics to simplify dataflow programs via string diagram rewriting based on DisCoPy's hypergraph data structure.

References

[1] Titouan Carette, Marc de Visme & Simon Perdrix (2021): *Graphical Language with Delayed Trace: Picturing Quantum Computing with Finite Memory*. In: *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pp. 1–13, doi:10.1109/LICS52264.2021.9470553. Available at https://ieeexplore.ieee.org/abstract/document/9470553.

[2] Kenta Cho & Bart Jacobs (2019): *Disintegration and Bayesian Inversion via String Diagrams*. *Mathematical Structures in Computer Science* 29(7), pp. 938–971, doi:10.1017/S0960129518000488. arXiv:1709.00322.

[3] Giovanni de Felice, Alexis Toumi & Bob Coecke (2020): *DisCoPy: Monoidal Categories in Python*. In: *Proceedings of the 3rd Annual International Applied Category Theory Conference, ACT*, 333, EPTCS, doi:10.4204/EPTCS.333.13.

[4] Elena Di Lavore, Giovanni de Felice & Mario Román (2022): *Monoidal Streams for Dataflow Programming*. In: *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '22, Association for Computing Machinery, New York, NY, USA, pp. 1–14, doi:10.1145/3531130.3533365. Available at https://doi.org/10.1145/3531130.3533365.

[5] André Joyal, Ross Street & Dominic Verity (1996): *Traced Monoidal Categories*. *Mathematical Proceedings of the Cambridge Philosophical Society* 119(3), pp. 447–468, doi:10.1017/S0305004100074338. Available at https://www.cambridge.org/core/journals/mathematical-proceedings-of-the-cambridge-philosophical-society/article/abs/traced-monoidal-categories/2BE85628D269D9FABAB41B6364E117C8.

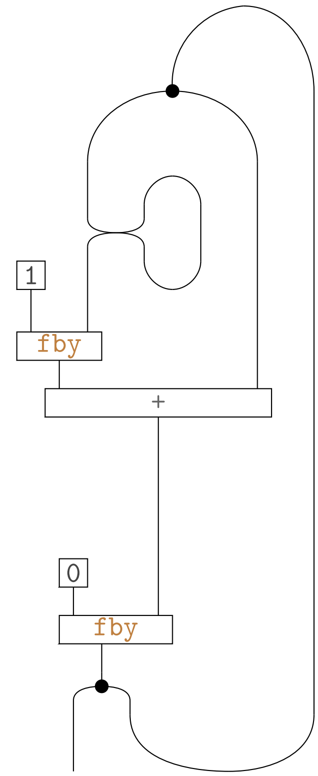
[6] P. Katis, Nicoletta Sabadini & Robert F. C. Walters (2002): *Feedback, Trace and Fixed-Point Semantics*. *RAIRO - Theoretical Informatics and Applications* 36(2), pp. 181–194, doi:10.1051/ita:2002009. Available at https://www.rairo-ita.org/articles/ita/abs/2002/02/ita0217/ita0217.html.

[7] Alexis Toumi, Giovanni de Felice & Richie Yeung (2022): *DisCoPy for the Quantum Computer Scientist*. *QPL*. arXiv:2205.05190.

[8] Alexis Toumi, Richie Yeung, Boldizsár Poór & Giovanni de Felice (2023): *DisCoPy: The Hierarchy of Graphical Languages in Python*, doi:10.48550/arXiv.2311.10608. arXiv:2311.10608.

A The Fibonacci sequence as a feedback diagram

```
from discopy.feedback import *

Diagram.use_hypergraph_equality = True

X = Ty('X')
fby, wait = FollowedBy(X), Swap(X, X.d).feedback()
zero, one = Box('0', Ty(), X), Box('1', Ty(), X)
copy, plus = Copy(X), Box('+', X @ X, X)

@Diagram.feedback
@Diagram.from_callable(X.d, X @ X)
def fib(x):
    x = fby(zero.head(), plus.d(
      fby.d(one.head.d(), wait.d(x)), x))
    return (x, x)

assert fib == (copy.d >> one.head.d @ wait.d @ X.d
                      >> fby.d @ X.d
                      >> plus.d
                      >> zero.head @ X.d
                      >> fby >> copy).feedback()

F = Functor(
    ob={x: int},
    ar={zero: lambda: 0,
        one: lambda: 1,
        plus: lambda x, y: x + y},
    cod=stream.Category(python.Ty, python.Function))

assert F(fib).unroll(10).now() == (0, 1, 1, 2, 3, 5, 8, 13, 21, 34)
```



B A random walk as a feedback diagram

```
from random import choice, seed; seed(420)
from discopy import stream, python
from discopy.feedback import *

x, fby = Ty('x'), FollowedBy(Ty('x'))
zero, rand, plus = Box('0', Ty(), x), Box('rand', Ty(), x), Box('+', x @ x, x)

@Diagram.feedback
@Diagram.from_callable(x.d, x @ x)
def walk(x0):
    x1 = plus.d(rand.d(), x0)
    x2 = fby(zero.head(), x1)
    return (x2, x2)

F = Functor(
    ob={x: int},
    ar={zero: lambda: 0,
        rand: lambda: choice([-1, +1]),
        plus: lambda x, y: x + y},
    cod=stream.Category(python.Ty, python.Function))

assert F(walk).unroll(10).now() == [0, -1, 0, 1, 2, 1, 0, -1, 0, 1]
assert F(walk).unroll(10).now() == [0, -1, 0, 1, 2, 1, 2, 3, 2, 1]
assert F(walk).unroll(10).now() == [0, 1, 0, 1, 0, -1, 0, -1, 0, -1]
```