Symmetric Monoidal Automata
Extended Abstract

Elena Di Lavore and Mario Román

Tallinn University of Technology, Ehitajate tee 5, 12616 Tallinn, Estonia

Introduction. Automata appear across the computer science literature in different flavours: deterministic, non-deterministic, or stochastic; and each of these flavours has its own notions of transition, bisimulation or trace equivalence. This text shows that we do not need to repeat these notions each time we change the flavour of automata: all these automata are particular cases of what we call *monoidal automata*; we start studying bisimulation and trace equivalence for monoidal automata; and we suggest that these generalise the multiple classical notions.

Monoidal Automata. We depart from a monoidal analogue of transition system that comes from the work of Katis, Sabadini and Walters [KSW97]: *monoidal transition system*. Input and output types, automata, and automata morphisms assemble into a monoidal bicategory. Previous work of these authors, together with Gianola, Sabadini and Sobociński [DLGR+23] has studied the universal properties of these monoidal transition systems.

Definition 1. An *automaton* in a symmetric monoidal category $(\mathbb{C}, \otimes, I)$, taking inputs on $A \in \mathsf{Obj}(\mathbb{C})$ and producing outputs in $B \in \mathsf{Obj}(\mathbb{C})$, is a triple (S, s_0, f) consisting of a *state space* $S \in \mathsf{Obj}(\mathbb{C})$, an initial state $s_0 \colon I \to S$ and a *transition morphism*, $f : S \otimes A \to S \otimes B$.

Definition 2. Monoidal automata over a symmetric monoidal category $(\mathbb{C}, \otimes, I)$ form a monoidal bicategory, $\mathsf{Aut}(\mathbb{C}, \otimes)$, whose 0-cells the objects of $\mathbb{C}$; whose 1-cells $(S, s_0, f) \colon A \to B$ are monoidal automata $f \colon S \otimes A \to S \otimes B$; and whose 2-cells $\alpha \colon (S, s_0, f) \to (T, t_0, g)$ are morphisms $\alpha : S \to T$ such that $f \fatsemi (\alpha \otimes \mathrm{id}) = (\alpha \otimes \mathrm{id}) \fatsemi g$ and $s_0 \fatsemi \alpha = t_0$. The composition and

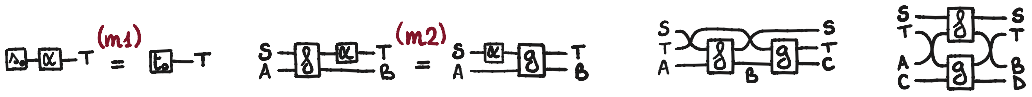


Figure 1: Morphisms, composition and tensor of monoidal automata.

tensoring of two automata tensors both state spaces and both initial states. The only difference is on how the transition functions interact: composition uses the output of the first as the input of the second, tensoring keeps them independent (Figure 1, right). The composition and the tensor of morphisms of automata is exactly its tensor in the original monoidal category.

Example 3 (Deterministic automata). Monoidal automata in the monoidal category of sets and functions are deterministic automata [BK08, Definition 2.5], i.e. a function $f \colon S \times A \to S \times B$ with an initial state $s_0 \in S$. Homomorphisms of deterministic automata [Rut95, Section 2] are precisely the 2-cells of $\mathsf{Aut}(\mathsf{Set}, \times)$.

Example 4 (Stochastic automata). Consider the category of stochastic maps with the cartesian product, $(\mathsf{Stoch}, \times)$. Morphisms from A to B, are functions $f(\bullet|\bullet) \colon A \times B \to [0, 1]$ such that $f(\bullet|a)$ has finite support and moreover $\sum_b f(b|a) = 1$ for each $a \in A$. Stochastic automata are monoidal automata over the category of stochastic maps, $\mathsf{Aut}(\mathsf{Stoch}, \times)$, i.e. an initial distribution, $d_0 \colon A \to [0, 1]$, together with a function $f(s', b|s, a)$ returning the probability of outputting b and transitioning to s' from the state s with an input a.

Example 5 (Non-deterministic automata). Non-deterministic automata [BK08, Definition 2.1] are monoidal automata over the category of relations, $\mathsf{Aut}(\mathsf{Rel}, \times)$. Deterministic morphisms between them coincide with the classical notion of morphism of automata [Rut95, Rut00].

Example 6. Automata can be represented as labelled graphs with an initial node. The vertices are states and the edges are transitions. In a deterministic automaton, for each vertex and each input, there is exactly one outgoing edge that produces one output, while, in a stochastic automaton, the weights of the outgoing edges must sum to 1.

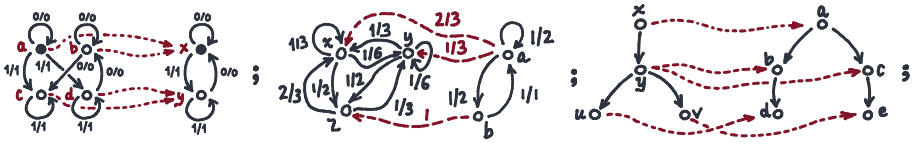
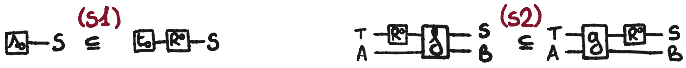


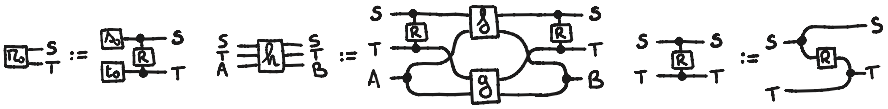
Figure 2: Homomorphisms of deterministic, stochastic and non-deterministic automata.

Morphisms and bisimulation. The classical definition of simulation can be rewritten as a pair of inequations in Rel. A relation $R \subseteq S \times T$ is a simulation from (S, s_0, f) to (T, t_0, g) if it satisfies $(s1)$ and $(s2)$ below, where $R^\circ \colon T \to S$ indicates the opposite of the relation R.



This observation allows us to characterise simulations in terms of morphisms of automata.

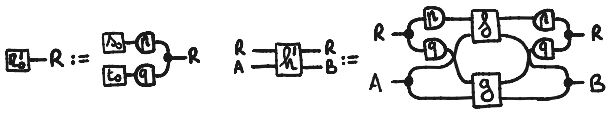
Proposition 7. *Let $(S, s_0, f), (T, t_0, g) \colon A \to B$ be two automata in* Rel *and $R \colon S \to T$ be a relation between their sets of states. Then, R is a simulation from (S, s_0, f) to (T, t_0, g) if and only if the relation $(\mathrm{id}_S \times R^\circ) \fatsemi \Delta$ is a morphism of automata $(S \times T, r_0, h) \to (S, s_0, f)$ in* Rel*, for the automaton $(S \times T, r_0, h)$ defined below.*



(1)

A bisimulation is a relation R such that both R and R° are simulations. We use this observation to characterise bisimulations with the following result, which is essentially a restatement of [Rut95, Lemma 2.2] with added inputs and outputs. We give a string diagrammatic proof.

Proposition 8. *Let $(S, s_0, f), (T, t_0, g) \colon A \to B$ be two automata in* Rel*, and $R \colon S \to T$ be a relation between their sets of states. Then, R is a bisimulation if and only if the projections $p \colon R \to S$ and $q \colon R \to T$ are morphisms of automata, $p \colon (R, r_0', h') \to (S, s_0, f)$ and $q \colon (R, r_0', h') \to (T, t_0, g)$, where (R, r_0', h') is the automaton defined below.*



(2)

Monoidal Streams and Monoidal Traces. Automata define an execution trace: a controlled stream of the visible outputs of the automaton. In the same way, monoidal automata define an execution trace, which must be now a *monoidal stream*. The theory of monoidal streams [DLdFR22] is an analogue of the coinductive construction of streams using morphisms in a symmetric monoidal category. We now employ it to study traces of monoidal automata.

A classical stream, $\mathbb{A} = (A_0, A_1, \ldots)$, is coinductively defined to be *(i)* an element $\mathtt{now}(\mathbb{A}) = A_0$, *(ii)* together with a stream, representing its tail, $\mathtt{later}(\mathbb{A}) = (A_1, A_2, \ldots)$. When the stream is formed by objects of a symmetric monoidal category, we can act on the first element of the stream $\mathtt{now}(M \cdot \mathbb{A}) = M \otimes \mathtt{now}(\mathbb{A})$ leaving the rest of the stream the same, $\mathtt{later}(M \cdot \mathbb{A}) = \mathtt{later}(\mathbb{A})$. Monoidal streams use this to introduce a new element: an explicit memory that allows to connect each piece of the stream to the next one.

Definition 9. A *monoidal stream* $\mathbb{f} \colon \mathbb{A} \to \mathbb{B}$ between two families of objects $\mathbb{A} = (A_0, A_1 \ldots)$ and $\mathbb{B} = (B_0, B_1, \ldots)$ is a triple $(\mathtt{M}(\mathbb{f}), \mathtt{now}(\mathbb{f}), \mathtt{later}(\mathbb{f}))$ consisting of *(i)* an object of the symmetric monoidal category, $\mathtt{M}(\mathbb{f})$, called the *memory*; *(ii)* a morphism of the symmetric monoidal category, $\mathtt{now}(\mathbb{f}) \colon \mathtt{now}(\mathbb{X}) \to \mathtt{M}(\mathbb{f}) \otimes \mathtt{now}(\mathbb{Y})$, called the *first action*; and *(iii)* a monoidal stream $\mathtt{later}(\mathbb{f}) \colon \mathtt{M}(\mathbb{f}) \cdot \mathtt{later}(\mathbb{A}) \to \mathtt{later}(\mathbb{B})$, called the *rest of the action*. Monoidal streams are quotiented by a dinaturality equivalence relation: the minimal equivalence relation that relates $(\mathtt{M}(\mathbb{f}), \mathtt{now}(\mathbb{f}), \mathtt{later}(\mathbb{f})) \sim (\mathtt{M}(\mathbb{g}), \mathtt{now}(\mathbb{g}), \mathtt{later}(\mathbb{g}))$, for each morphism $r \colon \mathtt{M}(\mathbb{g}) \to \mathtt{M}(\mathbb{f})$ such that $\mathtt{now}(\mathbb{g}) \fatsemi r = \mathtt{now}(\mathbb{f})$ and such that $r \cdot \mathtt{later}(\mathbb{f}) \sim \mathtt{later}(\mathbb{g})$. Monoidal streams form a symmetric monoidal category, $\mathsf{Stream}(\mathbb{C}, \otimes)$.

A single object A can be repeated to form a stream $[\![A]\!]$, defined by $\mathtt{now}([\![A]\!]) = A$ and $\mathtt{later}([\![A]\!]) = [\![A]\!]$. Analogously, a transition function, $f \colon S \otimes A \to S \otimes B$ with state space S, can be repeated to form a monoidal stream $[\![f]\!] \colon S \cdot [\![A]\!] \to [\![B]\!]$. When the transition function has an initial state, s_0, and forms a monoidal automata, we can use it to construct a monoidal stream that represents the execution trace of the automaton.

Definition 10. The *trace* of a monoidal automaton $(S, s_0, f) \colon A \to B$ is the monoidal stream, $\mathbf{tr}(S, s_0, f) \colon [\![A]\!] \to [\![B]\!]$, defined by $\mathbf{tr}(S, s_0, f) = s_0 \cdot [\![f]\!]$. We say that two automata are *trace-equivalent* if their traces coincide.

Example 11 (Deterministic traces). The traces of a deterministic automaton (S, s_0, f) are the streams of elements in the image of the monoidal stream $\mathbf{tr}(S, s_0, f)$ in $\mathsf{Stream}(\mathsf{Set}, \times)$. The explicit form of functional monoidal streams are *causal stream functions* [SJ19] and can be described as a coKleisli category [UV08].

Related work. Our work is mostly based on the work of Katis, Sabadini and Walters on processes in monoidal categories [KSW97], and our previous work, first presented at NWPT'21, which coinductively defined *monoidal streams* [DLdFR22] as a monoidal counterpart for dataflow programming [UV08]. We expand this work: we now add notions of bisimulation and trace equivalence for monoidal automata. We reason coinductively; Kozen and Silva [KS17] have given an informal exposition of practical coinductive reasoning. Most of the categorical literature on automata has been separated from monoidal categories [CP20, SBBR13], or has been restricted to the cartesian case [BLLL23]. Notions of probabilistic and metric bisimulation have been already advanced [BDEP97, DDLP06, BBLM17, BSS21, BSV22]. The term "monoidal automata" is also used to indicate automata whose language is monoidal [ES22, ES23].

Acknowledgements. Elena Di Lavore and Mario Román were supported by the European Social Fund Estonian IT Academy research measure (project 2014-2020.4.05.19-0001).

References

[BBLM17] Giovanni Bacci, Giorgio Bacci, Kim G Larsen, and Radu Mardare. On the metric-based approximate minimization of markov chains. 2017.

[BDEP97] Richard Blute, Josée Desharnais, Abbas Edalat, and Prakash Panangaden. Bisimulation for labelled markov processes. In *Proceedings of Twelfth Annual IEEE Symposium on Logic in Computer Science*, pages 149–158, 1997. doi:10.1109/LICS.1997.614943.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[BLLL23] G. Boccali, A. Laretto, F. Loregian, and S. Luneia. Completeness for categories of generalized automata. In Paolo Baldan and Valeria de Paiva, editors, *10th Conference on Algebra and Coalgebra in Computer Science (CALCO 2023)*, volume 270 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:14, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CALCO.2023.20.

[BSS21] Filippo Bonchi, Alexandra Silva, and Ana Sokolova. Distribution Bisimilarity via the Power of Convex Algebras. *Logical Methods in Computer Science*, 17(3), 07 2021. doi:10.46298/lmcs-17(3:10)2021.

[BSV22] Filippo Bonchi, Ana Sokolova, and Valeria Vignudelli. The Theory of Traces for Systems with Nondeterminism, Probability, and Termination. *Logical Methods in Computer Science*, 18(2), 06 2022. doi:10.46298/lmcs-18(2:21)2022.

[CP20] Thomas Colcombet and Daniela Petrişan. Automata minimization: a functorial approach. *Logical Methods in Computer Science*, 16, 2020.

[DDLP06] Vincent Danos, Josée Desharnais, François Laviolette, and Prakash Panangaden. Bisimulation and cocongruence for probabilistic systems. *Information and Computation*, 204(4):503–523, 2006.

[DLdFR22] Elena Di Lavore, Giovanni de Felice, and Mario Román. Monoidal streams for dataflow programming. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '22, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3531130.3533365.

[DLGR+23] Elena Di Lavore, Alessandro Gianola, Mario Román, Nicoletta Sabadini, and Paweł Sobociński. Span(Graph): a canonical feedback algebra of open transition systems. *Software and System Modeling*, 22(2):495–520, 2023. doi:10.1007/s10270-023-01092-7.

[ES22] Matthew Earnshaw and Paweł Sobociński. Regular Monoidal Languages. In Stefan Szeider, Robert Ganian, and Alexandra Silva, editors, *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, volume 241 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 44:1–44:14, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2022.44.

[ES23] Matthew Earnshaw and Paweł Sobociński. String Diagrammatic Trace Theory. In Jérôme Leroux, Sylvain Lombardy, and David Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023)*, volume 272 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 43:1–43:15, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2023.43.

[KS17] Dexter Kozen and Alexandra Silva. Practical coinduction. *Mathematical Structures in Computer Science*, 27(7):1132–1152, 2017. doi:10.1017/S0960129515000493.

[KSW97] Piergiulio Katis, Nicoletta Sabadini, and Robert FC Walters. Bicategories of processes. *Journal of Pure and Applied Algebra*, 115(2):141–178, 1997. doi:10.1016/S0022-4049(96)00012-6.

[Rut95] Jan Rutten. A calculus of transition systems (towards universal coalgebra). In *Modal logic and process algebra : a bisimulation perspective*, pages 231–256, 01 1995.

[Rut00] Jan Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*,

249(1):3–80, 2000. Modern Algebra. doi:10.1016/S0304-3975(00)00056-6.

[SBBR13] Alexandra Silva, Filippo Bonchi, Marcello Bonsangue, and Jan Rutten. Generalizing determinization from automata to coalgebras. *Logical Methods in Computer Science*, 9, 2013.

[SJ19] David Sprunger and Bart Jacobs. The differential calculus of causal functions. *CoRR*, abs/1904.10611, 2019. URL: http://arxiv.org/abs/1904.10611, arXiv:1904.10611.

[UV08] 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.