The Power of Abstract Mathematics in Computer Science

Compositional Model Checking via Monoidal Categories

From (Abstract) Category Theory to (Concrete) Fast and Scalable Algorithms

Kazuki Watanabe, Clovis Eberhart, Ichiro Hasuo (NII & SOKENDAI), Kazuyuki Asada (Tohoku U)

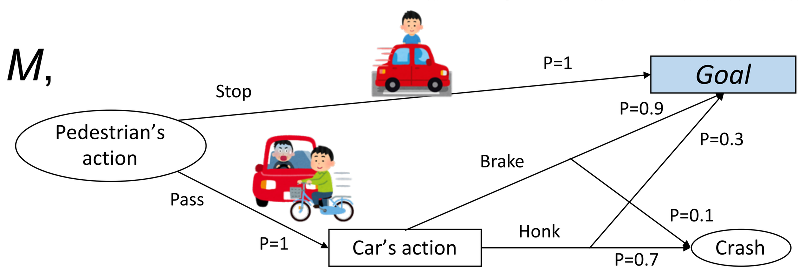
Compositional Probabilistic Model Checking with String Diagrams of MDPs, Proc. CAV 2023. See also [Watanabe+, TACAS'24 & CAV'24]

Abstract. We present the first MDP model-checking algorithm that is fully compostional and automatic. It is a rare example of abstract category theory leading to concrete algorithmic merits.

Model Checking: Graph-Based Automated Analysis

We study the following problem (**probabilistic model checking**):

- given a **Markov decision process (MDP)** M,
- compute its **optimal expected reward**.
  - "expected": an action leads to a randomized next state
  - "optimal": for the best choice of actions (*strategy*)



An important problem, tacked also in reinforcement learning:

- finding the optimal strategy ➔ **decision making**
- precise computation of exp. reward ➔ **quality guarantee**

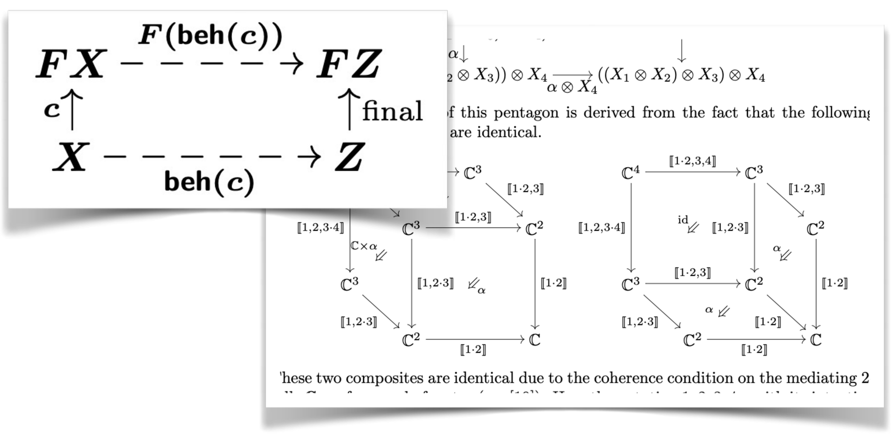
A topic heavily studied in the formal verification community (e.g. CAV).
Baier, C., Katoen, J.: Principles of model checking. MIT Press (2008)

Scalability is a challenge—sometimes an MDP has ~10^8 states (or more). "State explosion." **We want a scalable algorithm**.

Category Theory: an Abstract Language of Modern Math

Category theory is a language of modern mathematics
(Mac Lane, Eilenberg, Grothendieck, ...).

- It describes abstract structures using arrows,
- thus identifying essential similarities in different fields (e.g. algebra and geometry)



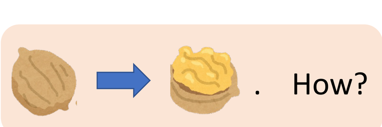
Use of category theory has been pursued in **computer science**, too
Functional programming and logic (Lawvere, Lambek, ..., cf. Haskell)
Coalgebra and process theory (Jacobs, Rutten, ...)

- "Get the setting right, then the problem solves itself"

However, the use of category theory has been mostly as a **theoretical backend**

- It helps to come up with a theory (definitions, correctness theorems)
- But **concrete algorithmic benefits have been rare**

Grothendieck's pursuit of simplicity and elegance: Nut opening example

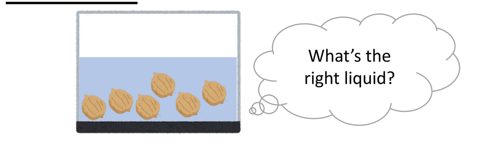


Alexander Grothendieck (1928-2014)
Leading figure of modern algebraic geometry
Categories, sheaves, ...
Fields medal, 1966

Method 1

✅ Quicker to get to a solution
❌ The solution is specific, not transportable. It does not scale

Method 2



❌ Takes time
✅ The solution, once found, is generic, widely applicable (other kind of nuts, shellfish, ...), scalable (use a bigger pool)

We pursue this with **category theory** (as Grothendieck did)

→ Once we establish the **right environment** (~ **formulation, definition, language**), a problem solves by itself

Compositionality: Algebraic "Divide-and-Conquer" for Scalability

Compositionality is a paradigm eagerly pursued in computer science.
It comes with performance advantage as well as mathematical blessing

The solution of a composition is obtained by ...

$$\mathcal{S}(\mathcal{A} \star \mathcal{B}) = \mathcal{S}(\mathcal{A}) \star \mathcal{S}(\mathcal{B})$$

first solving the components $\mathcal{A}$, $\mathcal{B}$ and then composing their solutions

Composition of *systems* (seqComp, parComp, sum, ...) — Composition of *semantics*

Conceptual Value

- "**Divide-and-Conquer**": simplifies a problem into smaller subproblems
- $\mathcal{S}(\mathcal{A})$, $\mathcal{S}(\mathcal{B})$ are **summaries** of components $\mathcal{A}$, $\mathcal{B}$. Unnecessary details get abstracted away

Performance Advantage

- Clear adv. when there are **duplicates** (reuse $\mathcal{S}(\mathcal{A})$!)

$$\mathcal{S}(\mathcal{A} \star \cdots \star \mathcal{A}) = \mathcal{S}(\mathcal{A}) \star \cdots \star \mathcal{S}(\mathcal{A})$$

- (In some cases you don't need duplicates, e.g. mergesort)

Mathematical Blessing

- Compositionality means that the solution

$$\mathcal{S}\colon \mathbb{M} \longrightarrow \mathbb{S}$$

is a **homomorphism**, preserving the operation ★

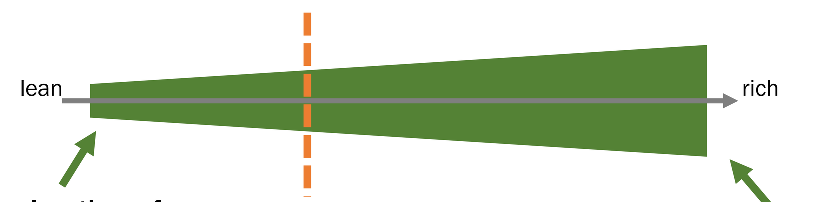
"Semantical Granularity" as a Challenge

Compositionality is not easy to achieve, though...
one has to strike the right "**granularity of semantics**"

$$\mathcal{S}(\mathcal{A} \star \mathcal{B}) = \mathcal{S}(\mathcal{A}) \star \mathcal{S}(\mathcal{B})$$

Imagine: estimating the output of a team of two

Q. What information should the summaries $\mathcal{S}(\mathcal{A})$, $\mathcal{S}(\mathcal{B})$ carry?



the **original notion of solution** (only) →
- enough when $\mathcal{A}$, $\mathcal{B}$ operate separately;
- not enough otherwise

find minimal enrichment that addresses interaction

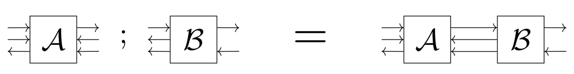
the **component itself** → no suppressed details, no performance advantage

Monoidal Categories Come to the Rescue

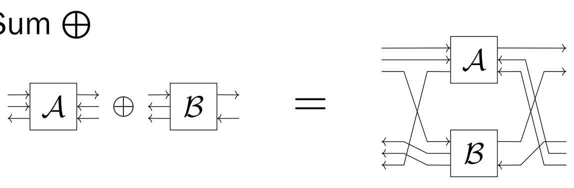
We use **string diagrams** for composing MDPs. They come from monoidal categories

String Diagram of MDPs

- Sequential composition ;



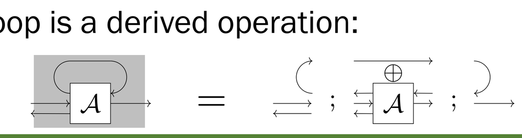
- Sum ⊕



- and some "constants" , , , ...

➔ **planar composition** of MDPs (mostly *sequential* composition; not parallel)

- Loop is a derived operation:



Background: Monoidal Categories

- Well-established field of category theory (Mac Lane, Kelly, Joyal, Street, ...)
- Used for many applications:
  quantum field theory (Khavanov, ...),
  quant. computation (Abramsky, Coecke, Vicary, Heunen, ...),
  linguistics (Sadrzadeh, Coecke, ...),
  signal flow diagrams (Bonchi, Sobocinski, Zanasi, ...), ...
- String diagrams as an *internal language* for monoidal categories [Joyal & Street, Adv. Math. 1991]
  - nicely expressive (planar composition, see left)
  - comes with a rich metatheory (see later)
- In particular, we use string diagrams for **compact closed categories** (compCCs).
  - Dual object A^* for each A
  - Example: fin.-dim. vector spaces, MDPs, ...

Use of monoidal categories comes with two notable benefits.

Benefit 1: Settling "Granularity of Semantics"

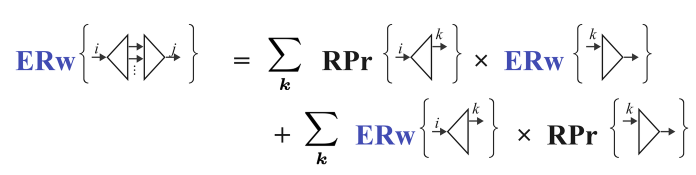
- Compositional solution is a homomorphism

$$\mathcal{S}\colon \mathbb{M} \longrightarrow \mathbb{S}$$

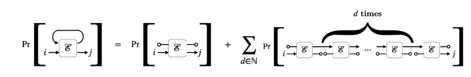
- The domain $\mathbb{M}$ is a compact closed category (compCC) of MDPs
  ➔ **We need $\mathbb{S}$ to be a compCC too!**
  - Minimally enrich the original solution domain (optimal expected reward) till it becomes a compCC
- Turns out: it suffices to additionally compute **reachability probabilities**
  - "decomposition equalities" (right)

- *Reachability probabilities* decompose into themselves:

- Similar decomposition of *expected rewards* requires more data:

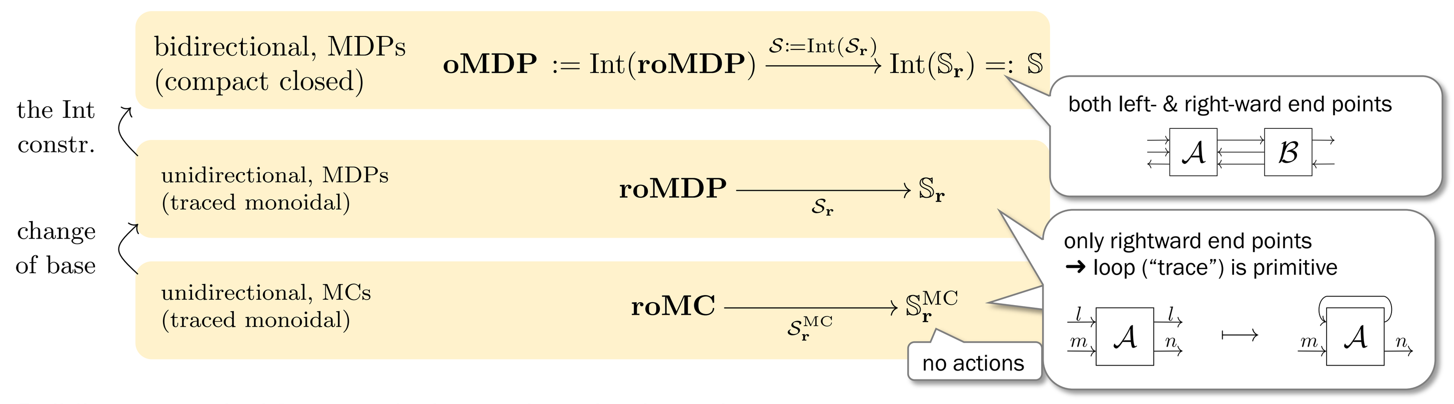


- The same for loops



- (NB: no actions so far, this is about Markov chains → next slide)

Benefit 2: Framework Upgrades for Free (uni-dir. MC ➔ uni-dir. MDP ➔ bi-dir. MDP)



- Building semantical frameworks is easy in a simple setting (*rightward-open MCs*; no action, end points are all rightward)
- Upgrades are **for free**, exploiting the categorical metatheory change of base (Eilenberg, Kelly, ...), the Int construction (Joyal, Street, Verity)

Algorithmic Outcomes

- The first MDP model checking algorithm that is fully compositional and automatic
  inspired by [Junges+, CAV'22] (not fully compositional), [Kwiatkowska+, Inf. Comp. '13] (not fully automatic)
- Can be **arbitrarily faster** than non-compositional algorithms (increase duplicates!);
  we observed max. x 6000 speed-up
- Many large systems are built compostionally
  ➔ practical potential

benchmark	$\lvert Q\rvert$	$\lvert E\rvert$	exec. time [s] DI-high	DI-mid	DI-low
Patrol1	10^8	10^8	21	42	83
Patrol2	10^8	10^8	23	48	90
Patrol3	10^9	10^9	22	43	89
Patrol4	10^9	10^9	30	60	121
Wholesale1	10^8	$2 \cdot 10^8$	130	260	394
Wholesale2	10^8	$2 \cdot 10^8$	92	179	274
Wholesale3	$2 \cdot 10^8$	$4 \cdot 10^8$	6	12	23
Wholesale4	$2 \cdot 10^8$	$4 \cdot 10^8$	129	260	393

benchmark	$\lvert Q\rvert$	$\lvert E\rvert$	exec. time [s] FZ-none	FZ-int.	FZ-all (PRISM)
Packets1	$2.5 \cdot 10^5$	$5 \cdot 10^5$	TO	1	65
Packets2	$2.5 \cdot 10^5$	$5 \cdot 10^5$	TO	3	64
Packets3	$2.5 \cdot 10^5$	$5 \cdot 10^5$	TO	1	56
Packets4	$2.5 \cdot 10^5$	$5 \cdot 10^5$	TO	3	56
Patrol5	10^8	10^8	22	22	TO
Wholesale5	$5 \cdot 10^7$	10^8	TO	14	TO

$\lvert Q\rvert$ is the number of positions; $\lvert E\rvert$ is the number of transitions (only counting action branching, not probabilistic branching); execution time is the average of five runs, in sec.; timeout (TO) is 1200 sec.

Contact: Ichiro Hasuo, National Institute of Informatics & SOKENDAI
URL: https://group-mmm.org/eratommsd/ Email: hasuolab-secr@nii.ac.jp