Fibrational perspectives on determinization of finite-state automata

Thea Li
Université Paris Cité
Paris, France
thea.li@etu.u-paris.fr

Colcombet and Petrişan argued that automata may be usefully considered from a functorial perspective, introducing a general notion of "$\mathscr{V}$-automaton" based on functors into $\mathscr{V}$. This enables them to recover different standard notions of automata by choosing $\mathscr{V}$ appropriately, and they further analyzed the determinization for **Rel**-automata using the Kleisli adjunction between **Set** and **Rel**. In this paper, we revisit Colcombet and Petrişan's analysis from a fibrational perspective, building on Melliès and Zeilberger's recent alternative but related definition of categorical automata as functors $p : \mathscr{Q} \to \mathscr{C}$ satisfying the finitary fiber and unique lifting of factorizations property. In doing so, we improve the understanding of determinization in three regards: Firstly, we carefully describe the universal property of determinization in terms of forward-backward simulations. Secondly, we generalize the determinization procedure for **Rel** automata using a local adjunction between **Span(Set)** and **Rel**, which provides us with a canonical forward simulation. Finally we also propose an alterative determinization based on the multiset relative adjunction which retains paths, and we leverage this to provide a canonical forward-backward simulation.

1 Introduction

One motivation for studying automata from a categorical point of view, is that it enables us to reason about properties of automata abstractly. In particular, this is useful when considering constructions where canonicity might be desirable, such as for determinization of non-deterministic automata. Commonly one considers automata as algebras or coalgebras, see among others [1], [4], [8], [2].

A different approach is taken by Colcombet and Petrişan [3], they introduced a general notion of $\mathscr{V}$-automaton based on functors into $\mathscr{V}$, which enabled them to recover standard notions of automata by instantiating $\mathscr{V}$ appropriately. In particular, they show that non-deterministic automata can be seen as **Rel**-automata, and they further provide a determinization for **Rel**-automata using the Kleisli-adjunction between **Set** and **Rel**. The **Rel**-automata approach to non-deterministic automata is practical for studying logical aspects of automata, yet it does not retain some of the computational content. Consequently, we revisit Colcombet and Petrişan analysis of determinization from the point of view of Mellies and Zeilberger's [5] alternative approach, using functors with the *unique lifting of factorizations* (ULF) property, that ameliorates the loss of computational content and generalizes **Rel**-automata. Their approach describes non-deterministic automata as ULF-functors over a category of labels, with finite fibers.

Both [5] and [3] provide a definition of deterministic automata, which turns out to be equivalent under the Grothendieck construction. As mentioned, determinization categorically has previously been studied by Colcombet and Petrişan for **Rel**-automata, but is yet to be done Melliès and Zeilberger's approach. In this work, we detail the determinization procedure for **Rel** and provide a generalization of this to the setting of [5], recovering a similar universal property. We then discuss the links between our construction and the usual algorithmic procedure first presented by Rabin and Scott [7]. As this procedure identifies

To appear in EPTCS.

paths with the same label, which is not always desirable, we construct an alternative determinization using multisets that is path-relevant.

1.1 Outline

We recall in Section 2 the two definitions of non-deterministic automata as functors and the common definition of a deterministic automaton. In Section 3 we revisit the determinization procedure for **Rel**-automata and in Section 4 we construct a generalization of this procedure to the **Span(Set)** case. We then construct an alternative determinization in Section 5 with a stronger universal property, though the downside of this one is a significant increase in size.

2 Preliminaries

2.1 Automata as functors

In [3], Colcombet and Petrişan consider functors as generalized automata. They view the domain as the category of input words and the codomain as the category of outputs, and the functor itself as specifying the behavior of the automaton. Additionally, to specify the accepted input, they require a full subcategory of the category of inputs, that in practice amounts to choosing initial and final states.

Definition 2.1. [3, Def. 2.1] A $\mathscr{V}$-automaton is a functor $\mathscr{A} : \mathscr{I} \to \mathscr{V}$, together with a full subcategory inclusion $i : \mathscr{O} \to \mathscr{I}$

This definition has the advantage to be very simple categorically and easy to manipulate. Moreover, the accepted language can be defined naturally as the composition of the automata and its full subcategory inclusion.

Definition 2.2. [3, Def. 2.1] The language accepted by a $\mathscr{V}$-automaton $\mathscr{A} : \mathscr{I} \to \mathscr{V}$ is the composition $\mathscr{A} \circ i : \mathscr{O} \to \mathscr{V}$.

To model classical non-deterministic automata without epsilon transitions they particularize the definition to the case where $\mathscr{V}$ is **Rel**, the category of sets and relations. Given a fixed automaton over the alphabet Σ, they take $\mathscr{I}$ to be the one object category associated to the free monoid Σ^*, to which they add two objects in and out and two morphisms that acts as begin- and end-input markers. $\mathscr{O}$ is then the full subcategory generated by in and out, we denote this by $\mathscr{I}_{\text{word}}$. The functor maps the base object of the free monoid to the set of states, each letter to the relation generated by the transition function, in and out are both mapped to 1 and the begin- and end-input markers to relations pointing out initial and final states.

In the remainder of the paper, we will call automata with codomain **Rel**, **Rel**-automata, regardless of domain.

2.2 Non-deterministic automata as ULF functors

The **Rel**-automata approach to non-deterministic automata is language oriented and models very well the logical aspects of automata. Yet, as the functor is mapping into **Rel**, it only interprets transitions as ways of relating states under a label and does not account for which states it passed along the way, nor does it account for the different ways of doing so. It thus equates different paths, and we loose some of the computational content of automata.

Mellies and Zeilberger [5] introduces a categorification of non-deterministic automata (without epsilon transitions) using ULF-functors, that is closer to the classical definition, ameliorates the loss of computational content, and is actually a generalization of **Rel**-automata. The idea is for the domain to be the underlying directed graph of the automaton and the codomain to be the alphabet, with the ULF functor specifying the label of a path. The ULF-property then describes an important feature of automata, that a path α labelled by a word w can be factored in subpaths along the letters of w.

Definition 2.3 (ULF-functor)**.** A functor p is ULF if for each factorization of a morphism $p(\alpha) = uv$ there are unique morphisms β and γ such that $p(\beta) = u$ and $p(\gamma) = v$.

Definition 2.4 (ULF-automaton [5, Def. 3.3])**.** An automaton over a category $\mathscr{C}$ is a tuple $(\mathscr{C}, \mathscr{Q}, p : \mathscr{Q} \to \mathscr{C}, q_0, Q_f)$ where p is a ULF functor with finite fibers, q_0 is an object of $\mathscr{Q}$, and Q_f is a set of objects in $\mathscr{Q}$.

The objects of $\mathscr{Q}$ corresponds to states, q_0 to the initial state, and Q_f to the final states, while unfactorizable morphisms in $\mathscr{C}$ corresponds to letters of our alphabet, and consequently arbitrary morphism to words. This leads us to naturally to consider the following as its language.

Definition 2.5. [5, Def. 3.3] The language recognized by a ULF-automaton M with labelling functor p, is $L_M := \{w \mid \exists \alpha : q_0 \to q_f,\ \exists q_f \in Q_f,\ p(\alpha) = w\}$.

It turns out that ULF-automata generalizes **Rel**-automata, indeed any functor $\mathscr{I} \to$ **Rel** can be extended to a functor $\mathscr{I} \to$ **Span(Set)**, yet by a Grothendieck like correspondence any pseudofunctor into **Span(Set)** corresponds to a ULF-functor [5], hence any **Rel**-automaton induces a ULF-automaton over $\mathscr{I}$. We can in fact characterize these as a particular type of ULF-functors.

Proposition 2.6. *A functor $F : \mathscr{C} \to$ **Span(Set)** factors through **Rel** if and only if its corresponding functor lying over $\mathscr{C}$ (by the grothendieck construction) is ULF and faithful.*

Corollary 2.7. ***Rel**-automata $\mathscr{C} \to$ **Rel** factoring through **FinRel** are equivalent to faithful ULF-automata over $\mathscr{C}$.*

We can, as noted in [5], represent ULF-automata as pseudofunctors $\mathscr{C} \to$ **Span(Set)** factoring through **Span(Fin)**, up to choice of initial and final states. In practice, we will often directly work with the equivalent **Span(Set)**-functor rather than with of the ULF-automaton, as it makes it easier to relate automata over the same category. We will henceforth refer to them as **Span(Set)**-automata.

We remark that we on occasion drop the condition that our **Span(Set)**-automata factors through **Span(Fin)** as it is not always important. In Section 4 all constructions preserve finiteness, and consequently the construction naturally restricts to **Span(Fin)**, thus we will consider general **Span(Set)**-automata. However, in Section 5 whether constructions are finite is important and we adapt our definitions accordingly.

2.3 Deterministic Automata

Both [3] and [5] give definitions of a deterministic automaton. These definitions turn out to be equivalent under the Grothendieck construction given the right instantiations, and being able to reason using both approaches is useful.

By definition an automaton is deterministic when its transition relation is a functional relation, consequently, [3] models deterministic automata as **Rel**-automata that factors through **Set**.

Definition 2.8 ([3])**.** A deterministic automaton is a functor $\mathscr{I}_{\text{word}} \to$ **Set**, which maps *in* to 1 and *out* to 2.

For a ULF-functor to model a deterministic automaton it not only needs a unique lift of factorizations of words, as noted in [6], it requires every word to have a unique path given a starting state, meaning that any word $w : p(q) \to c$ has a unique lift $\alpha : q \to q'$.

Definition 2.9 (Discrete opfibration)**.** A functor $p : \mathscr{Q} \to \mathscr{C}$ is a discrete opfibration if any morphism $w : p(q) \to c$ has a unique lift, i.e. there exists a unique $\alpha : q \to q'$ such that $p(\alpha) = w$.

Definition 2.10. [6, Def. 2.9] A deterministic automaton over a category $\mathscr{C}$ is a tuple $(\mathscr{C}, \mathscr{Q}, p : \mathscr{Q} \to \mathscr{C}, q_0, Q_f)$ where p is a discrete opfibration with finite fibers, q_0 is an object of $\mathscr{Q}$, and Q_f is a set of objects in $\mathscr{Q}$.

It turns out that these two definitions are equivalent given the right instantiations.

Proposition 2.11. *By the Grothendieck construction, functors $\mathscr{C} \to$ **Set** are equivalent to discrete opfibrations over $\mathscr{C}$.*

Corollary 2.12. ***Set**-automata $\mathscr{I}_{\text{word}} \to$ **Set** are equivalent to deterministic automata over $\mathscr{I}_{\text{word}}$ where the fiber over* in *has one element and the fiber over* out *has two.*

3 Determinisation for Rel

In order to determinize **Span(Set)**-automata, we first focus on the particular case of **Rel**-automata, previously studied in [3], this procedure will constitute in one building stone for our **Span(Set)**-automata determinization. We recall the determinization procedure for **Rel**-automata in Section 3.1, transpose it to faithful ULF-automata. Finally we discuss the difficulties of extending the construction to **Span(Set)**.

3.1 Determinisation for Rel

Determinizing a **Rel**-automaton $\mathscr{I} \to$ **Rel** consists in transforming it into a functor $\mathscr{I} \to$ **Set**; to obtain such, one can postcompose by a functor **Rel** $\to$ **Set**. As explained in [3], such a functor naturally arises, as **Rel** is the Kleisli-category of the powerset monad on **Set**. Consequently, there is an adjunction $L :$ **Set** $\rightleftarrows$ **Rel** $: R$. The functor R sends objects A in **Rel** to their powersets $\mathscr{P}(A)$ and relations $r \subseteq A \times B$ to the function $\mathscr{P}(A) \to \mathscr{P}(B)$ that sends a subset $S \subseteq A$ to the subset of all elements in B related to some element in S.

Construction 3.1. [3, Sec 3.3] Given a **Rel**-automaton $A : \mathscr{I} \to$ **Rel**, its deteminization is given by

$$Det(F) := \mathscr{C} \xrightarrow{A} \textbf{Rel} \xrightarrow{R} \textbf{Set}$$

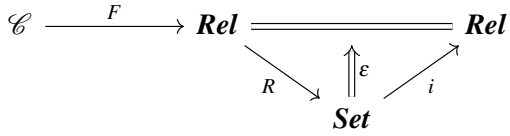
To be an appropriate determinization procedure the deterministic automaton should simulate the original automaton and recognize the same language. Simulations can be represented categorically as the following, which is inspired by simulations of labelled transition systems presented in [9] and similar to the morphisms of automata presented in [3]. Moreover, this gives a suitable notion of morphism of automata.

Definition 3.2. Given two automata $F, F' : \mathscr{C} \to$ **Rel**, a *forward-backward simulation* from F to F' is a natural transformation $\alpha : F' \Rightarrow F$.

Definition 3.3. **Rel**-automata with domain $\mathscr{C}$ and simulations form a category Aut($\mathscr{C}$).

We can then justify the canonicity of the determinization by the existence of a canonical *forward-backwards* simulation from an automaton to its determinization, in which the initial state is simulated by an initial state and the final states are simulated by final states.

Proposition 3.4. *Let $F : \mathscr{C} \to$ **Rel** be an automaton, then there is a canonical forward-backward simulation from F to Det(F) given by the counit of the Kleisli-adjunction between **Rel** and **Set**.*



Having a canonical simulation is not enough to justify that the determinization preserves the language, as the simulation is one-sided, and as we have yet to address how initial and final states are determined for Det(F). Under Grothendieck construction Det(F) is

$$\mathrm{Det}(M) := (\int RF, \mathscr{C}, \Pi : \int RF \to \mathscr{C}, q_0', Q_f')$$

We take as initial state $q_0' = (p^{-1}(c), \{q_0\})$, where p is the ULF-functor corresponding to F, q_0 is the initial state of F and $c = p(q_0)$. The set of final states Q_f' are all states $(p^{-1}(q_f), S)$ such that q_f is a final state in F and $q_f \in S$. Given this it is straight forward to check that Det(F) recognizes the same language as F.

3.2 Universal Property

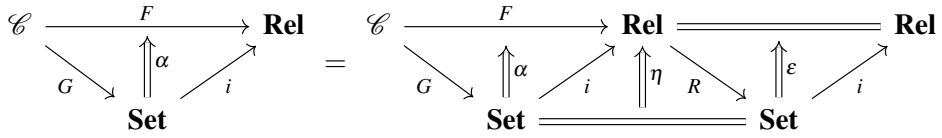
Now that we have justified the canonicity of our determinization procedure, we would like to be able to relate it to other determinizations and other deterministic automata, that is, equip it with a universal property. To characterize the universal property we make use of the notion of bisimulation, for this purpose, recall that **Rel** is a dagger category taking $(-)^\dagger$ as the converse relation.

Definition 3.5. If $\alpha : F' \Rightarrow F$ is a natural transformation such that $(\alpha)^\dagger$ also is a natural transformation, then α is called a *bisimulation*.

As our canonical simulation is built out of a unit we make use of the triangle identity for unit and counit of adjoint functors to obtain the following universal property.

Proposition 3.6. *Let $G : \mathscr{C} \to$ **Set** be some deterministic automaton then any* forward-backward *simulation from F to G factors uniquely through the universal simulation to Det(F) and a bisimulation between Det(F) and G.*

Proof. By one of the triangle identities for adjoint functors we have the following equality.



The simulation between Det(M) and G is a bisimulation as any natural transformation generated by a natural transformation between functors into **Set** and an inclusion into **Rel**. □

3.3 Difficulties in generalization to Span(Set)-automata

This determinization procedure has the advantage to be direct and fairly simple, but does not scale easily to **Span(Set)** as it is less well behaved than **Rel**. First, we do not have an adjunction between **Span(Set)** and **Set**, which was essential in the **Rel**-case for determinization and to establish the universal property. Second, as morphisms in **Span(Set)** are composed using pullbacks, functors out of or to **Span(Set)** tend to be lax or pseudo as pullbacks are only unique up to isomorphism.

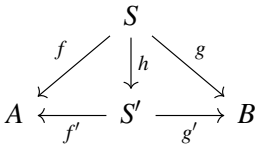
4 Powerset determinization for Span(Set)-automata

In this section, we construct local adjunction between **Span(Set)** and **Rel** seen as 2-categories, and we show that this is sufficient to recover a determinization procedure and universal property analogous to the **Rel**-determinization for ULF-automata/**Span(Set)**-automata. We then discuss the link between the categorical determinization procedure and the algorithmic one.

4.1 Determinization procedure

As ULF-automata over $\mathscr{C}$ are equivalent to pseudofunctors $\mathscr{C} \to$ **Span(Set)** and deterministic ULF-automata are equivalent to functors $\mathscr{C} \to$ **Set**, it suffices to be able to build a functor $\mathscr{C} \to$ **Set** out of a pseudo-functor $\mathscr{C} \to$ **Span(Set)** to determinize ULF-automata. As we know how to determinize **Rel**-automata, it is sufficient for us to be able to turn a pseudo-functor $\mathscr{C} \to$ **Span(Set)** into a functor $\mathscr{C} \to$ **Rel** in a canonical way. That is, it suffices to construct a lax functor **Span(Set)** $\to$ **Rel** that strictifies the pseudo-functor to a strict functor $\mathscr{C} \to$ **Rel** by postcomposition. To do so, we must work in a 2-categorical setting as we deal with pseudo-functors, and view **Span(Set)** and **Rel** as a 2-categorical construction.

Definition 4.1. **Span(Set)** is a 2-category with 2-morphisms given by morphisms of spans $\langle f,g\rangle : S \rightrightarrows \langle A,B\rangle$ to $\langle f',g'\rangle : S' \rightrightarrows \langle A,B\rangle$ which are functions $h : S \to S'$ such that:



In this context, we can view **Rel** as a 2-category in a similar way, 2-morphisms would be given by inclusions of relations, which indeed is just a morphism of spans.

There is a "forgetful" lax functor from **Span(Set)** to **Rel** that is identity on objects and maps spans to the relation generated by the image of the legs, we call this functor Im for image. As this functor identifies spans that generate the same relation, by post-composing it to a pseudo-functor such as our automaton F we obtain a functor into **Rel** that is strict. Post-composing an automaton $F : \mathscr{C} \to$ **Span(Set)** with $R \circ \mathrm{Im}$ does indeed give us a determinization of F that preserves the language, as the image functor only identifies all paths with the same label between two given states.

Construction 4.2. Given a **Span(Set)**-automaton $F : \mathscr{C} \to$ **Span(Set)**, its determinization is given by

$$Det(F) := \mathscr{C} \xrightarrow{F} \textbf{Span(Set)} \xrightarrow{\mathrm{Im}} \textbf{Rel} \xrightarrow{R} \textbf{Set}$$

As for **Rel**-automata, for our determinization to be appropriate, the deterministic automaton must simulate the original automaton and recognize the same language. This requires us to adapt our notion of simulation to the 2-categorical setting and to **Span(Set)**.

Definition 4.3. Given two automata $F, F' : \mathscr{C} \to$ **Span(Set)**, a *forward simulation* from F to F' is a lax natural transformation $\alpha : F' \Rightarrow F$.

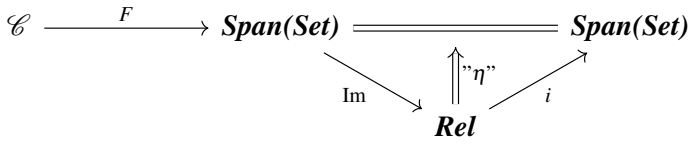
Definition 4.4. Given two automata $F, F' : \mathscr{C} \to$ **Span(Set)**, a *forward-backward simulation* from F to F' is a pseudo natural transformation $\alpha : F' \Rightarrow F$.

Definition 4.5. **Span(Set)**-automata with domain $\mathscr{C}$ and forward simulations form a category $\mathrm{Aut}(\mathscr{C})$.

Though the image functor does not have any strict adjoints, it is the left adjoint of a local adjunction between **Rel** and **Span(Set)** seen as 2-categories, with the right adjoint being the inclusion. Using this local adjunction properly will enable us the prove the canonicity of our determinization procedure, as well as recover a universal property similar to one of **Rel**. To construct a canonical simulation we begin by constructing a canonical simulation from a **Span(Set)**-automaton and its **Rel**-ification, for this purpose we will make use of the following definition.

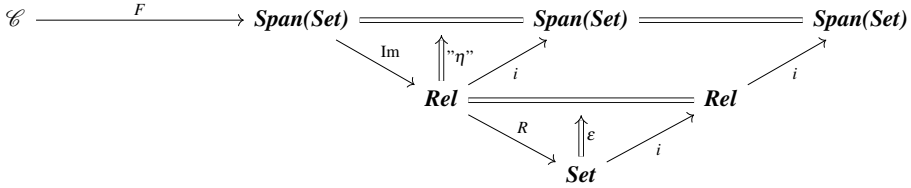
Definition 4.6. We denote by "η" : $i\mathrm{Im} \Rightarrow \mathrm{Id}_{\mathbf{Span(Set)}}$ the lax natural transformation that is the identity on each component with lax naturality given by the η the unit of the local adjunction $\mathrm{Im} : \mathbf{Span(Set)}(A,B) \rightleftarrows \mathbf{Rel}(A,B) : i$. We respectively denote by "ε" : $\mathrm{Id}_{\mathbf{Rel}} \Rightarrow \mathrm{Im}i$ the lax natural transformation associated to the counit.

Lemma 4.7. *Let $F : \mathscr{C} \to$ **Span(Set)** be an automaton, then there is a canonical forward simulation from F to $iImF$ given by the lax natural transformation "η" defined above.*



Using this we can now obtain a canonical simulation from an automaton to its determinization similar to the one for **Rel**.

Proposition 4.8. *Let $F : \mathscr{C} \to$ **Span(Set)** be an automaton, then there is a canonical forward simulation from F to Det(F) given by pasting the forward simulation 4.7 and the counit of the Kleisli-adjunction between **Rel** and **Set**.*



Moreover, it indeed recognizes the same language as postcomposing with Im does not affect the language and the analogous result for **Rel**-automata. Hence this determinization procedure is a fitting determinization procedure.

4.2 Universal Property for Span(Set)

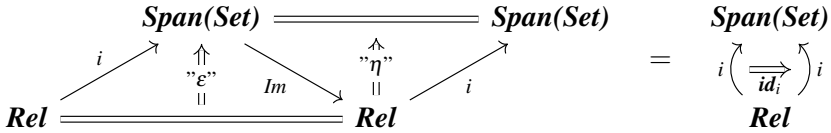
Having proven the canonicity of our determinization, we now construct the universal property of the determinization of **Span(Set)**-automata. As before, we rely on bisimulations, which we need to adapt to **Span(Set)**. For this purpose, recall **Span(Set)** is a dagger category.

Proposition 4.9. ***Span(Set)** is a dagger category taking $(-)^\dagger$ as the converse span.*

Definition 4.10. A a natural transformation $\alpha : F' \Rightarrow F$ such that $(\alpha)^\dagger : F \Rightarrow F'$ is also a natural transformation is a *bisimulation*, any such alpha must incidentally be strict or pseudo.

To recover a similar universal property as for **Rel**, we additionally need a pasting inverse to the simulation "η". To construct such an inverse we leverage the fact that our construction is based on a local adjunction.

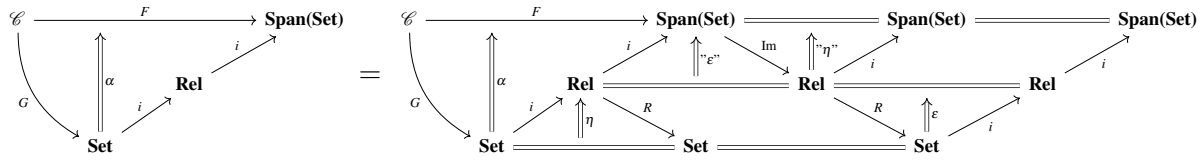
Lemma 4.11. *The forward simulation "ε" is the pasting inverse of the forward simulation "η"*



Using this "triangle-identity" for "ε" and "η" and the universal property for **Rel**-automata we can now obtain a universal property for **Span(Set)**-automata.

Proposition 4.12. *Let $G : \mathscr{C} \to$ **Set** be some deterministic automaton, then any forward simulation from F to G factors through the canonical simulation to Det(F) and a bisimulation between Det(F) and G.*

Proof. By Proposition 3.6 and Lemma 4.11 we have that the following equality of diagrams:

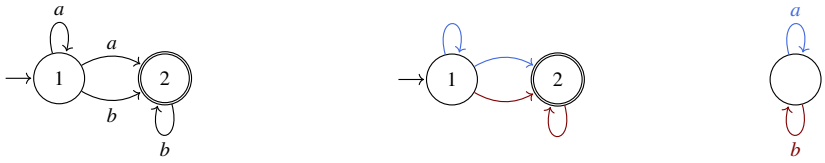


Consequently this gives a unique factorization of α through the universal simulation between M and Det(M). □

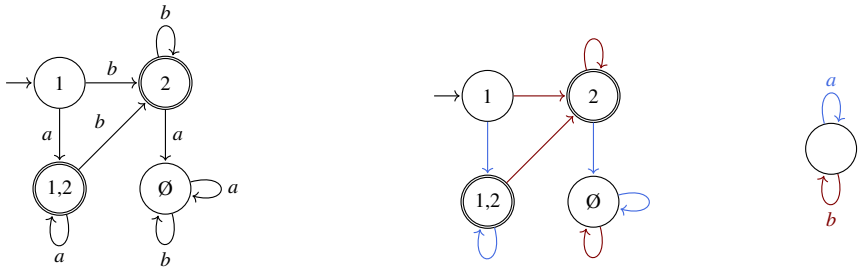
4.3 Relation to the classical determinization algorithm

To see how this version of determinization presented here relates to the classical determinization algorithm we start by working through an example.

Example 4.13. Consider the following automaton:



Using the determinization process presented in the previous section the determinization of this automaton is the ULF-automaton pictured to the right, which is equivalent to what we would get using the algorithmic procedure, pictured to the left.

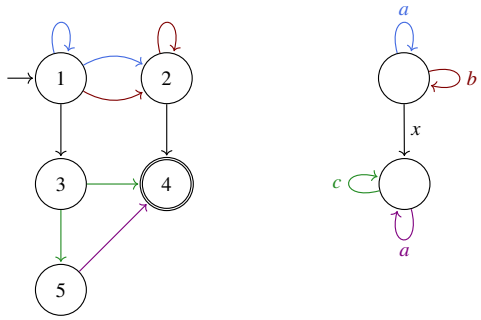


Let us recall the algorithmic procedure for determinizing an automaton. Given an automaton $M = (\Sigma, Q, \delta : Q \times \Sigma \to \mathscr{P}(Q), q_0, Q_f)$ we begin by taking the powerset of the set of states $\mathscr{P}(Q)$, this will be the states of our deterministic machine. Then for the transition function we just take the function $\delta' : \mathscr{P}(Q) \times \Sigma \to \mathscr{P}(Q)$, generated by δ, mapping $S \subseteq Q \mapsto \{q \in Q \mid \exists q'(q \in \delta(q'))\}$.

When we have a free monoid as base category for our automaton the determinization we obtain is analogous to the algorithmic one. This is because postcomposition by R replaces the objects in $\mathscr{C}$ with the powersets of their fibers, if we only have one fiber, then the determinization has its powerset as set of states.

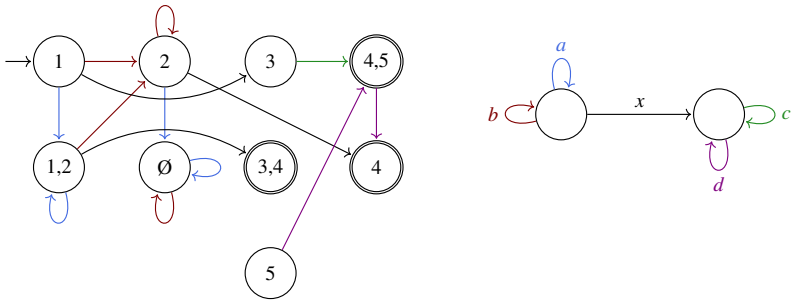
For a general base category we do not get the full powerset of the original set of states, as the states of our deterministic machine. Instead the determinization process takes to a certain extent into account which subsets would be unreachable. The limiting factor here is that these are only the subsets comprised of states lying over different fibers and consequently have no chance of a path to or from them.

Example 4.14. Consider the following automaton M:



It recognizes the language $\{a^n b^m xcd\} \cup \{a^n b^m xc\}$, in particular has two "disjoint" alphabets $\{a, b\}$ and $\{c, d\}$ with a connective x, meaning that we have no paths combining them without an x in the middle, and once we have "put" the x after a possibly empty word $a^n b^m$ we cannot go back and only have $\{c, d\}$ available.

The following is the determinization of M using our determinization:



We see here that we do not get states containing states from different fibers, which makes our determinized automaton smaller that the powerset.

5 Multiset determinization for Span(Set)-automata

One step in the determinization process presented above that might feel unjustified is the identification of paths between states having the same label (i.e. postcomposition with the image functor into **Rel**). To make amends for this we might consider a path-relevant version of determinization using multisets. It is worth noting that this significantly increases the size of our final machine, however, it strengthens the universal property.

5.1 Procedure

Multisets only correspond to finite spans, consequently we will restrict ourselves to **Span(Fin)**-automata. This is not unjustified as we in the definition of ULF-automaton have the condition that the fibers of the ULF-functor are finite, that is, its corresponding **Span(Set)**-functor factors through **Span(Fin)**. To define the multiset determinization use that the multiset functor forms a relative monad $M : \mathbf{Fin} \to \mathbf{Set}$ and the fact that the "relative" Kleisli-category on **Fin** is the skeleton of **Span(Fin)** on the 2-categorical level.

Proposition 5.1. *Multisets form a relative monad from **Fin** to **Set**, defined for all $A, B : \mathbf{Fin}$ and $\varphi : A \to \mathbb{N}^B$ by*

$$\begin{array}{cccc} M : \mathbf{Fin} & \to & \mathbf{Set} \\ A & \mapsto & \mathbb{N}^A \end{array} \qquad \begin{array}{cccc} \eta_A : A & \to & \mathbb{N}^A \\ a & \mapsto & \Sigma_{a' \in A} \chi_a a' \end{array} \qquad \begin{array}{cccc} \varphi^* : \mathbb{N}^A & \to & \mathbb{N}^B \\ f & \mapsto & \Sigma_{a \in A} f(a) \cdot \varphi(a, -) \end{array}$$

Lemma 5.2. *There is a local equivalence of categories between **Span(Fin)** and $\mathbf{Fin}_M$ that is identity on objects defined by*

$$\begin{array}{rcl} U : \mathbf{Span(Fin)}(A,B) & \rightleftarrows & \mathbf{Fin}_M(A,B) : \amalg \\ \langle f, g \rangle : S \rightrightarrows \langle A, B \rangle & \mapsto & (a \mapsto (b \mapsto \left| \langle f, g \rangle^{-1}(a,b) \right|)) \\ \langle \pi_1, \pi_2 \rangle : \amalg_{(a,b) \in A \times B} f(a,b) \rightrightarrows \langle A, B \rangle & \mapsfrom & f : A \to \mathbb{N}^B \end{array}$$

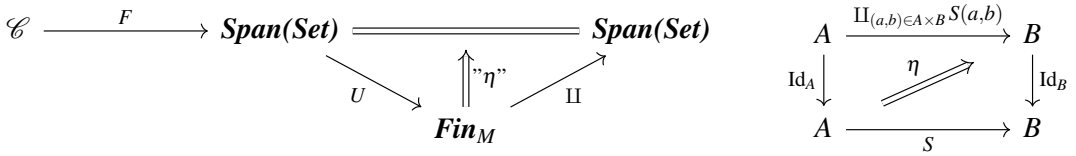
Construction 5.3. Given a **Span(Set)**-automaton $F : \mathscr{C} \to \mathbf{Span(Set)}$, its multiset determinization is given by the following, where $\mathbf{Fin}_M$ is the relative Kleisli-category associated to M.

$$\mathrm{MDet}(F) := \mathscr{C} \xrightarrow{F} \mathbf{Span(Set)} \xrightarrow{U} \mathbf{Fin}_M \xrightarrow{M} \mathbf{Set}$$

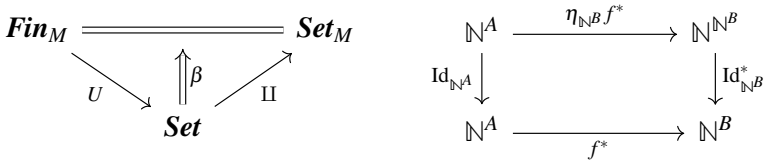
As MDet(M) is based on the multiset monad it can be seen as a determinization that remembers which states it passed and how many different ways it passed it.

We must again make sure that there is a canonical simulation between an automaton and its determinization, as well as the fact that it recognizes the same language. Here the use of the local equivalence of categories and relative Kleisli-category assures us of having a canonical simulation that is forward-backward, instead of just forward.

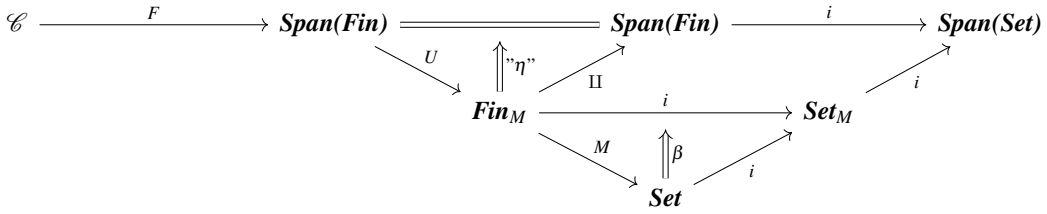
Lemma 5.4. *Let $F : \mathscr{C} \to \mathbf{Span(Set)}$ be an automaton, then there is a canonical forward-backward simulation from F to $\amalg UF$ with pseudo naturality given by the η of the local equivalence of categories.*



Lemma 5.5. *There is a natural transformation β, with the following naturality square (commuting in **Set**)*



Proposition 5.6. *Let $F : \mathscr{C} \to$ **Span(Set)** be an automaton, then there is a canonical forward-backward simulation from F to MDet(F).*



5.2 Universal Property

We again rely on constructing pasting inverses to obtain a universal property. To construct these we rely on the fact that our construction is based on a local equivalence of categories as well as a relative adjunction induced by a relative monad.

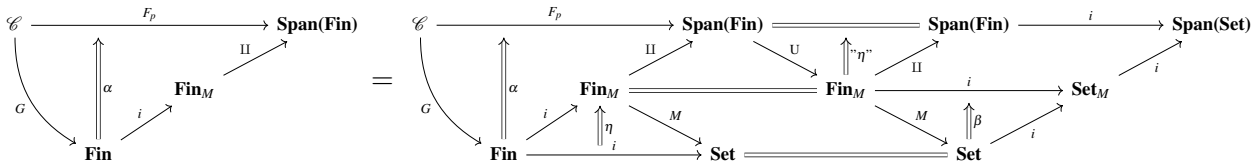
Lemma 5.7. *The natural transformation α is the identity when restricted to $\mathbf{Fin}_M$, as the ε from the local equivalence of categories Lemma 5.2 is the identity.*

Lemma 5.8. *The natural transformation β is the pasting inverse of the simulation η from the relative adjunction*

Using these lemmas we find the following corresponding universal property:

Proposition 5.9. *Let $G : \mathscr{C} \to$ **Set** be some deterministic automaton, then any simulation from F to G factors through the canonical forward-backward simulation to MDet(F) and a bisimulation between Det(F) and G.*

Proof. By Lemma 5.7 and Lemma 5.8 we have that the following diagram:



Consequently this gives a unique factorization of α through the universal simulation between M and Det(M). □

6 Conclusion

In this paper, we studied determinizations for **Rel**-automata and ULF-automata from a categorical point of view. Determinization for **Rel**-automata has previously been investigated by Colcombet and Petrişan

in [3], as ULF-automata is a generalization of **Rel**-automata, through the **Span(Set)** intermediary, we consider how it might be generalized to **Span(Set)**-automata and thus also to ULF-automata.

To do so, we gave a detailed presentation of the determinization procedure for **Rel**, which relies on the Kleisli adjunction between **Set** and **Rel**, as well as its correctness and universal property as previously described by Colcombet and Petrişan.

Generalizing this construction to **Span(Set)** turned out to be non-trivial, as we neither have an adjunction between **Span(Set)** and **Set**, nor one between **Span(Set)** and **Rel**. Moreover, as **Span(Set)**-automata corresponds to pseudofunctors, it forced us to work in a 2-categorical setting and adapt definitions to it. We have shown that it is possible to recover a fitting determinization procedure by first turning a pseudofunctor $\mathscr{C} \to$ **Span(Set)** into $\mathscr{C} \to$ **Rel** using the local adjunction and between **Span(Set)** and **Rel** then using the **Rel**-determinization to get back a functor $\mathscr{C} \to$ **Set**. To prove correctness and equip the procedure with a similar universal property we used the local adjunction and the correctness and universal property for **Rel**-determinization. The determinized automaton we obtained has subsets of states as states and a transition function, which given a letter maps a subset to the set of reachable states in the original automaton, analogously to usual algorithmic determinization procedure. We showed that our procedure do not always get whole powerset as the set of states, but does as soon as we consider automata with a free monoid as codomain.

This determinization procedure has the particularity to identify the paths taken, as this may not be wanted, we provided an alternative determinization based on the multiset relative monad that retains them. Compared to the one before, we no longer rely on the **Rel** determinization procedure as a middle step, as factoring through **Rel** identifies paths. Instead, we use the relative Kleisli category of the relative multiset monad M, $\mathbf{Fin}_M$, as a middle step, and then postcompose by the functor from $\mathbf{Fin}_M$ to **Set** provided by Kleisli construction of $\mathbf{Fin}_M$. Moreover, we have proven the correctness of this procedure and provided it with a universal property, leveraging the relative monad and that there is a local equivalence of categories between $\mathbf{Fin}_M$ and **Span(Fin)**. The multiset determinization enabled us to achieve path relevance, and to obtain a stronger universal property that encompasses forward-backward simulation, as we now get a canonical forward-backward simulation between an automaton F and its multiset determinization $\mathrm{MDet}(F)$.

There are many potential ways to expand the work on categorical automata, one potential possibility is to explore the multiset determinization from the point of view of transducers where path relevance might be desired as we in addition to the input also have an output to take into account. Another possibility would be minimization, as this has been done for **Rel**-automata, we might consider how to generalize this to to our setting.

References

[1] J. Adámek and V. Trnková. *Automata and Algebras in Categories*. Mathematics and its Applications. Springer Netherlands, 1990. ISBN: 9780792300106. URL: https://books.google.fr/books?id=MkADy_WxInsC.

[2] Michael A. Arbib and Ernest G. Manes. "Machines in a category". In: *Journal of Pure and Applied Algebra* 19 (1980), pp. 9–20. ISSN: 0022-4049. DOI: https://doi.org/10.1016/0022-4049(80)90090-0. URL: https://www.sciencedirect.com/science/article/pii/0022404980900900.

[3] Thomas Colcombet and Daniela Petrişan. “Automata Minimization: a Functorial approach”. In: *Logical Methods in Computer Science* (Mar. 2020), pp. 1–32. DOI: 10.23638/LMCS-16(1:32)2020. URL: https://hal.science/hal-03105616.

[4] Samuel Eilenberg and Jesse B. Wright. “Automata in general algebras”. In: *Information and Control* 11.4 (1967), pp. 452–470. ISSN: 0019-9958. DOI: https://doi.org/10.1016/S0019-9958(67)90670-5. URL: https://www.sciencedirect.com/science/article/pii/S0019995867906705.

[5] Paul-André Melliès and Noam Zeilberger. “Parsing as a lifting problem and the Chomsky-Schützenberger representation theorem”. In: *MFPS 2022 - 38th conference on Mathematical Foundations for Programming Semantics*. Ithaca, NY, United States, July 2022. URL: https://hal.science/hal-03702762.

[6] Paul-André Melliès and Noam Zeilberger. “The categorical contours of the Chomsky-Schützenberger representation theorem”. This is a thoroughly revised and expanded version of a paper with a similar title (hal-03702762, arXiv:2212.09060) presented at the 38th Conference on the Mathematical Foundations of Programming Semantics (MFPS 2022). 62 pages, including a 13 page Addendum on ”gCFLs as initial models of gCFGs”, and a table of contents. Dec. 2023. URL: https://hal.science/hal-04399404.

[7] M. O. Rabin and D. Scott. “Finite Automata and Their Decision Problems”. In: *IBM Journal of Research and Development* 3.2 (1959), pp. 114–125. DOI: 10.1147/rd.32.0114.

[8] J.J.M.M. Rutten. “Universal coalgebra: a theory of systems”. In: *Theoretical Computer Science* 249.1 (2000). Modern Algebra, pp. 3–80. ISSN: 0304-3975. DOI: https://doi.org/10.1016/S0304-3975(00)00056-6. URL: https://www.sciencedirect.com/science/article/pii/S0304397500000566.

[9] Pawel Sobociński. “Relational Presheaves as Labelled Transition Systems”. In: *11th International Workshop on Coalgebraic Methods in Computer Science (CMCS)*. Ed. by Dirk Pattinson and Lutz Schröder. Vol. LNCS-7399. Coalgebraic Methods in Computer Science. Tallinn, Estonia: Springer, Mar. 2012, pp. 40–50. DOI: 10.1007/978-3-642-32784-1_3. URL: https://inria.hal.science/hal-01539889.