Finitary Functors and Final Semantics in Type Theory

Niccolò Veltri
Tallinn University of Technology

(Includes joint work with Philipp Joram)

Joint Estonian-Latvian Theory Days 2022, 8 May 2022

Deterministic automata

- S – set of states
- A – alphabet
- $t : S \times A \to S$ – transition function
- $F \subseteq S$ – accept states
- (s_0 – initial state)

Deterministic automata

- S – set of states
- A – alphabet
- $t : S \times A \to S$ – transition function
  Equivalently $t : S \to S^A$
- $F \subseteq S$ – accept states
  Equivalently $F : S \to \mathrm{Bool}$
- (s_0 – initial state)

Deterministic automata

- S – set of states
- A – alphabet
- $t : S \times A \to S$ – transition function
  Equivalently $t : S \to S^A$
- $F \subseteq S$ – accept states
  Equivalently $F : S \to \mathrm{Bool}$
- (s_0 – initial state)

- Alternative specification of deterministic automaton:
  $\langle F, t \rangle : S \to \mathrm{Bool} \times S^A$

Deterministic automata

- S – set of states
- A – alphabet
- $t : S \times A \to S$ – transition function
  Equivalently $t : S \to S^A$
- $F \subseteq S$ – accept states
  Equivalently $F : S \to \mathrm{Bool}$
- (s_0 – initial state)

- Alternative specification of deterministic automaton:
  $\langle F, t \rangle : S \to \mathrm{Bool} \times S^A$

- Nondeterministic automaton:
  $\langle F, t \rangle : S \to \mathrm{Bool} \times \mathcal{P}(S)^A$

Labelled transition systems

- S – set of states
- A – set of labels
- $(\longrightarrow) \subseteq S \times A \times S$ – transition relation
  Styled $s \xrightarrow{a} s'$ instead of $(s, a, s') \in (\longrightarrow)$

Labelled transition systems

- S – set of states
- A – set of labels
- $(\longrightarrow) \subseteq S \times A \times S$ – transition relation
  Styled $s \xrightarrow{a} s'$ instead of $(s, a, s') \in (\longrightarrow)$

- Alternative specification of LTS:
  $t : S \to \mathcal{P}(A \times S)$

  $s \xrightarrow{a} s' \iff (a, s') \in t(s)$

Labelled transition systems

- S – set of states
- A – set of labels
- $(\longrightarrow) \subseteq S \times A \times S$ – transition relation
  Styled $s \xrightarrow{a} s'$ instead of $(s, a, s') \in (\longrightarrow)$

- Alternative specification of LTS:
  $t : S \to \mathcal{P}(A \times S)$

  $s \xrightarrow{a} s' \iff (a, s') \in t(s)$

- If each state has finite number of transitions: $t : S \to \mathcal{P}_f(A \times S)$

Labelled transition systems

- S – set of states
- A – set of labels
- $(\longrightarrow) \subseteq S \times A \times S$ – transition relation
  Styled $s \xrightarrow{a} s'$ instead of $(s, a, s') \in (\longrightarrow)$

- Alternative specification of LTS:
  $t : S \to \mathcal{P}(A \times S)$

  $s \xrightarrow{a} s' \iff (a, s') \in t(s)$

- If each state has finite number of transitions: $t : S \to \mathcal{P}_f(A \times S)$
- An alternative: $t : S \to \mathcal{M}_f(S)$ $\quad (\mathcal{M}_f(S) =$ "finite bags of S")
  $s \xrightarrow{n} s' \iff s' \in t(s)$ with multiplicity n

Transition systems as coalgebras

- Dynamical systems abstractly described by *coalgebras*: $t : S \to F(S)$

Transition systems as coalgebras

- Dynamical systems abstractly described by *coalgebras*:
  $t : S \to F(S)$

- S – set of states
- F – functor on sets, specifying e.g. type of labels, nondeterministic/probabilistic behaviour, termination, ...

  Automata: $F(X) = \mathrm{Bool} \times X^A$ or $F(X) = \mathrm{Bool} \times \mathcal{P}_f(X)^A$
  LTSs: $F(X) = \mathcal{P}_f(A \times X)$ or $F(X) = \mathcal{M}_f(X)$

- $t : S \to F(S)$ – transition function

Transition systems as coalgebras

- The entire computation of a transition system $t : S \to F(S)$, starting from initial state $s_0 : S$, is captured by the *final coalgebra* of F, written $\mathrm{out} : \nu F \to F(\nu F)$
- Intuitively, elements of νF are (possibly-infinite) trees, where the branching, the kind of information stored in the nodes, etc. is specified by F

Transition systems as coalgebras

- The entire computation of a transition system $t : S \to F(S)$, starting from initial state $s_0 : S$, is captured by the *final coalgebra* of F, written $\text{out} : \nu F \to F(\nu F)$
- Intuitively, elements of νF are (possibly-infinite) trees, where the branching, the kind of information stored in the nodes, etc. is specified by F

- For $F(X) = \text{Bool} \times X^A$ define $\mathcal{DA} = \nu F$.
  Then $x \in \mathcal{DA}$ is an infinite tree with Boolean-valued nodes and a branch for each $a \in A$
- For $F(X) = \mathcal{P}_f(A \times X)$ define $\mathcal{LTS} = \nu F$.
  Then $x \in \mathcal{LTS}$ is a possibly-infinite tree with finite unordered branching, where each branch is labelled by some $a \in A$

Transition systems as coalgebras

- The entire computation of a transition system $t : S \to F(S)$, starting from initial state $s_0 : S$, is captured by the *final coalgebra* of F, written $\text{out} : \nu F \to F(\nu F)$
- Intuitively, elements of νF are (possibly-infinite) trees, where the branching, the kind of information stored in the nodes, etc. is specified by F

- For $F(X) = \text{Bool} \times X^A$ define $\mathcal{DA} = \nu F$.
  Then $x \in \mathcal{DA}$ is an infinite tree with Boolean-valued nodes and a branch for each $a \in A$
- For $F(X) = \mathcal{P}_f(A \times X)$ define $\mathcal{LTS} = \nu F$.
  Then $x \in \mathcal{LTS}$ is a possibly-infinite tree with finite unordered branching, where each branch is labelled by some $a \in A$

- The *execution* of a system $t : S \to F(S)$ is given by a canonical function $\text{unfold} : S \to \nu F$

Motivation

- The unified treatment of systems using colgebras and the methods of category theory started >30 years ago by Aczel, Jacobs, Rutten, Turi, etc.
- Various benefits originate from this study, such as common notion of (bi)simulation and equivalence, and new proof techniques, e.g. coinduction principle

Motivation

- The unified treatment of systems using colgebras and the methods of category theory started >30 years ago by Aczel, Jacobs, Rutten, Turi, etc.
- Various benefits originate from this study, such as common notion of (bi)simulation and equivalence, and new proof techniques, e.g. coinduction principle

- *Universal coalgebra* traditionally developed in set theory and reasoning is based on classical logic
- It would be beneficial to develop it in constructive type theory, e.g. in proof assistants such as Agda or Coq:
  - Formalization of programming language semantics
  - Theory of coinductive types and their proof principles

Motivation

- The unified treatment of systems using colgebras and the methods of category theory started >30 years ago by Aczel, Jacobs, Rutten, Turi, etc.
- Various benefits originate from this study, such as common notion of (bi)simulation and equivalence, and new proof techniques, e.g. coinduction principle

- *Universal coalgebra* traditionally developed in set theory and reasoning is based on classical logic
- It would be beneficial to develop it in constructive type theory, e.g. in proof assistants such as Agda or Coq:
  - Formalization of programming language semantics
  - Theory of coinductive types and their proof principles

- In this talk, I will focus on *existence/construction of final coalgebras* in type theory for a selection of functors F

Polynomial functors

- Coq and Agda have some form of support for *coinductive types*
- The latter *are* final coalgebras for some specific functors.

- Det. automata: $F(X) = \mathsf{Bool} \times X^A$

$$\frac{x : \mathcal{D}\mathcal{A}}{\mathsf{accept?}\ x : \mathsf{Bool}}$$

$$\frac{x : \mathcal{D}\mathcal{A}}{\mathsf{next}\ x : A \to \mathcal{D}\mathcal{A}}$$

record $\mathcal{D}\mathcal{A}$: Type where
  coinductive
  field
    accept? : Bool
    next : $A \to \mathcal{D}\mathcal{A}$

$$\mathsf{accept?}\ \mathsf{allZeros} = 0$$
$$\mathsf{next}\ \mathsf{allZeros} = \lambda a.\ \mathsf{allZeros}$$

Polynomial functors

- Coq and Agda have some form of support for *coinductive types*
- The latter *are* final coalgebras for some specific functors.

- Det. automata: $F(X) = \mathsf{Bool} \times X^A$

$$\frac{x : \mathcal{D}\mathcal{A}}{\mathsf{accept?}\ x : \mathsf{Bool}}$$

$$\frac{x : \mathcal{D}\mathcal{A}}{\mathsf{next}\ x : A \to \mathcal{D}\mathcal{A}}$$

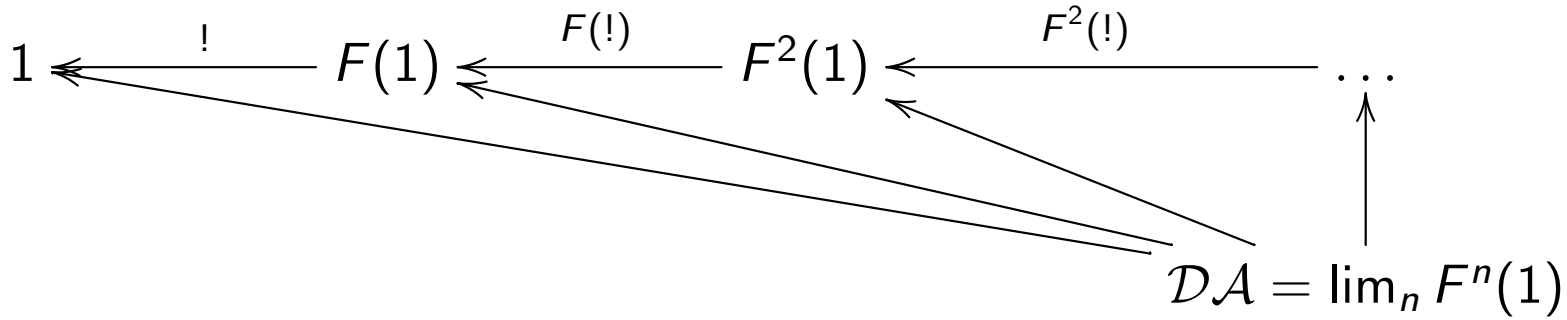
$$\begin{array}{l}\mathsf{record}\ \mathcal{D}\mathcal{A} : \mathsf{Type}\ \mathsf{where}\\ \quad \mathsf{coinductive}\\ \quad \mathsf{field}\\ \quad\quad \mathsf{accept?} : \mathsf{Bool}\\ \quad\quad \mathsf{next} : A \to \mathcal{D}\mathcal{A}\end{array}$$

$$\begin{array}{l}\mathsf{accept?}\ \mathsf{allZeros} = 0\\ \quad \mathsf{next}\ \mathsf{allZeros} = \lambda a.\ \mathsf{allZeros}\end{array}$$

- Many interesting examples of systems are captured, e.g. all *polynomial functors* whose final coalgebras are called *M-types* in type theory

Internalizing M-types

- Ahrens et al.'15: M-types can be constructed *internally* in type theory as an ω-limit, without using coinductive types.
- Det. automata: $F(A) = \mathsf{Bool} \times X^A$



What about NFAs and LTSs?

- How do you represent $\mathcal{P}_f$ and $\mathcal{M}_f$ in type theory?

What about NFAs and LTSs?

- How do you represent $\mathcal{P}_f$ and $\mathcal{M}_f$ in type theory?
- Traditional solution: work with setoids, i.e. types equipped with an equivalence relation

$$\mathcal{P}_f^{\mathsf{s}}\,(A, R) = (\mathsf{List}\,A, \overline{\mathsf{List}}\;R)$$

$$\begin{aligned}\overline{\mathsf{List}}\;R\,s\,t \;=\; & ((x : A) \to x \in s \to \exists y : A.\, y \in t \times R\,x\,y) \\ & \times \\ & ((y : A) \to y \in t \to \exists x : A.\, x \in s \times R\,y\,x)\end{aligned}$$

What about NFAs and LTSs?

- How do you represent $\mathcal{P}_f$ and $\mathcal{M}_f$ in type theory?
- Traditional solution: work with setoids, i.e. types equipped with an equivalence relation

$$\mathcal{P}_f^{\mathsf{s}}\,(A, R) = (\mathsf{List}\,A, \overline{\mathsf{List}}\;R)$$

$$\begin{aligned}\overline{\mathsf{List}}\;R\,s\,t \;= &((x : A) \to x \in s \to \exists y : A.\, y \in t \times R\,x\,y)\\ &\times\\ &((y : A) \to y \in t \to \exists x : A.\, x \in s \times R\,y\,x)\end{aligned}$$

- The final coalgebra of $\mathcal{P}_f^{\mathsf{s}}$ can be constructed (in the category of setoids) using coinductive types:

$$\nu\mathcal{P}_f^{\mathsf{s}} = (\mathsf{Tree}, \mathsf{TreeR})$$

Tree is the final coalgebra of the List functor, and TreeR is the coinductive closure of $\overline{\mathsf{List}}$

What about NFAs and LTSs?

- Different solution: work in *homotopy type theory* (HoTT), which has support for *higher inductive types* (HITs)
- $\mathcal{P}_f(A)$ as a HIT (Frumin et al.'18):

$$\frac{}{\varnothing : \mathcal{P}_f(A)} \qquad \frac{a : A}{\{a\} : \mathcal{P}_f(A)} \qquad \frac{x, y : \mathcal{P}_f(A)}{x \cup y : \mathcal{P}_f(A)}$$

$$\frac{x, y, z : \mathcal{P}_f(A)}{\mathsf{assoc}\ x\ y\ z : (x \cup y) \cup z = x \cup (y \cup z)} \qquad \frac{x, y : \mathcal{P}_f(A)}{\mathsf{comm}\ x\ y : x \cup y = y \cup x}$$

$$\frac{x : \mathcal{P}_f(A)}{\mathsf{nr}\ x : x \cup \varnothing = x} \qquad \frac{x : \mathcal{P}_f(A)}{\mathsf{idem}\ x : x \cup x = x} \qquad + \text{ set-truncation}$$

What about NFAs and LTSs?

- Different solution: work in *homotopy type theory* (HoTT), which has support for *higher inductive types* (HITs)
- $\mathcal{P}_f(A)$ as a HIT (Frumin et al.'18):

$$\frac{}{\varnothing : \mathcal{P}_f(A)} \qquad \frac{a : A}{\{a\} : \mathcal{P}_f(A)} \qquad \frac{x, y : \mathcal{P}_f(A)}{x \cup y : \mathcal{P}_f(A)}$$

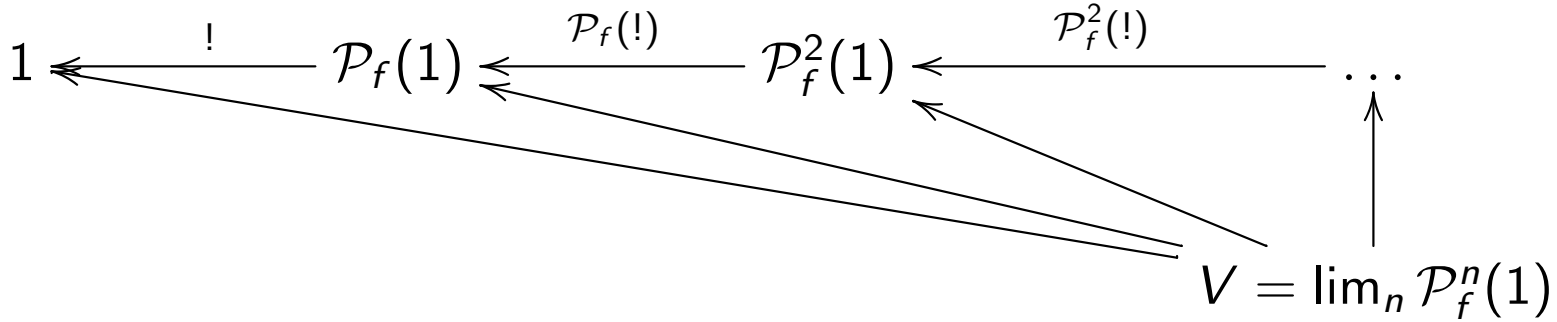
$$\frac{x, y, z : \mathcal{P}_f(A)}{\mathsf{assoc}\ x\ y\ z : (x \cup y) \cup z = x \cup (y \cup z)} \qquad \frac{x, y : \mathcal{P}_f(A)}{\mathsf{comm}\ x\ y : x \cup y = y \cup x}$$

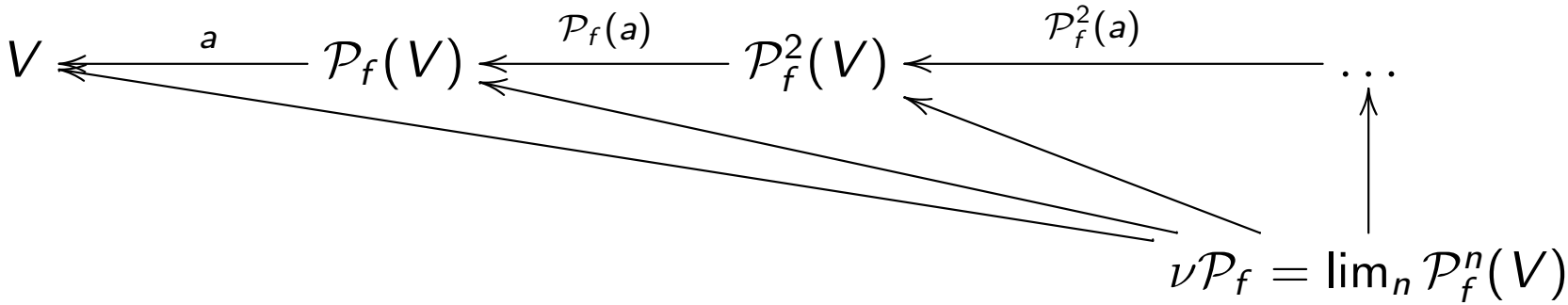
$$\frac{x : \mathcal{P}_f(A)}{\mathsf{nr}\ x : x \cup \varnothing = x} \qquad \frac{x : \mathcal{P}_f(A)}{\mathsf{idem}\ x : x \cup x = x} \qquad + \text{ set-truncation}$$

- Practically, work in *Cubical Agda*, a computational implementation of HoTT
- Cubical Agda has support for coinductive types/final coalgebras of functors specified in terms of HITs, e.g. $\mathcal{LTS}$. This feature is experimental.

Internal construction of final coalgebra of $\mathcal{P}_f$

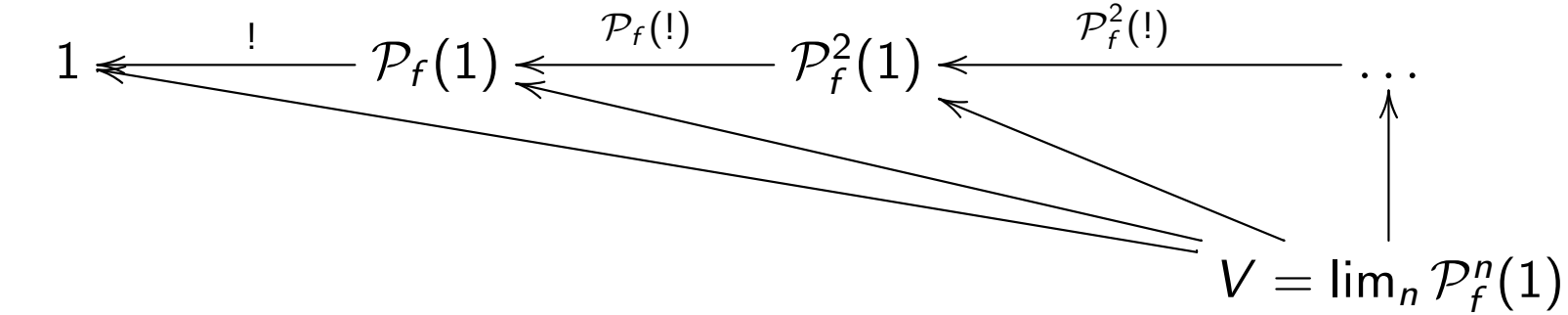
- V.'21: Analysis of Worrell's classical set-theoretic construction of $\nu\mathcal{P}_f$ as an $(\omega + \omega)$-limit in Cubical Agda

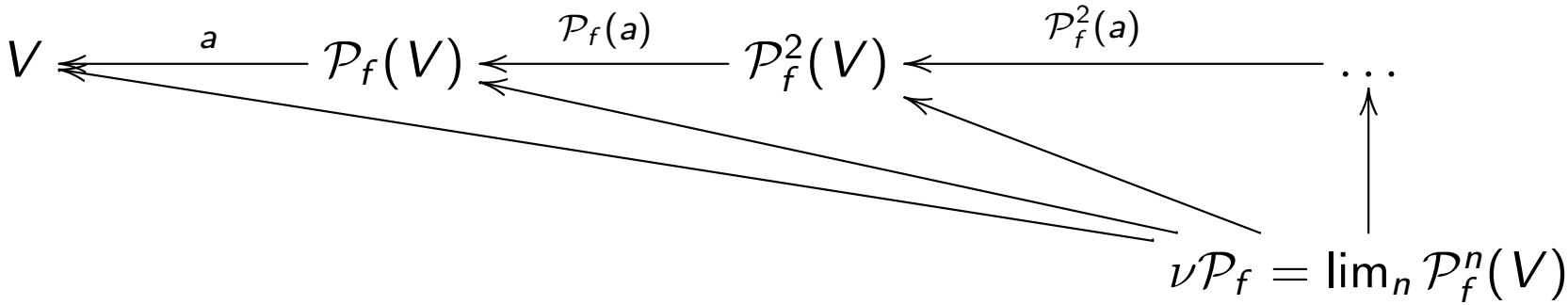




Internal construction of final coalgebra of $\mathcal{P}_f$

- V.'21: Analysis of Worrell's classical set-theoretic construction of $\nu\mathcal{P}_f$ as an $(\omega + \omega)$-limit in Cubical Agda

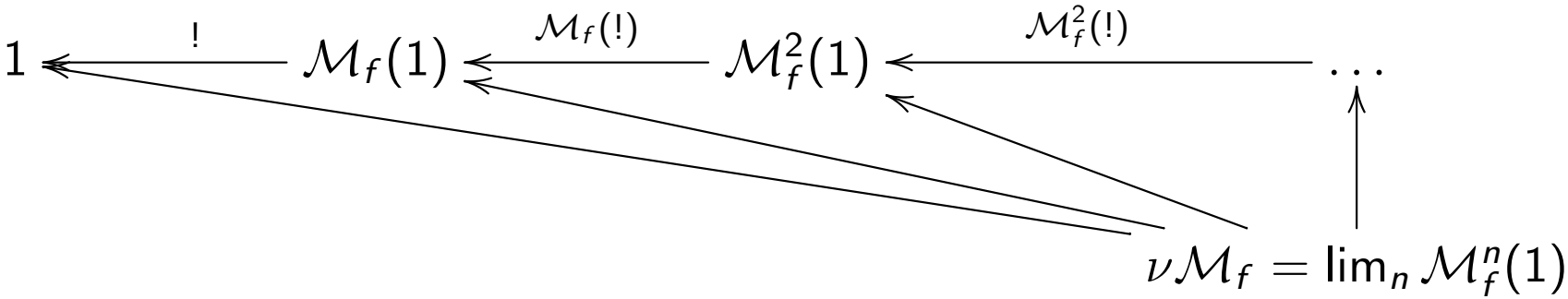




- We found that this construction is *intrinsically* classical: some strong instances of excluded middle and choice needed for it to work

Internal construction of final coalgebra of $\mathcal{M}_f$

- WIP with Philipp Joram: Construction of $\nu\mathcal{M}_f$ as an ω-limit in Cubical Agda



- Generalization to other *analytic functors*, e.g. $F(X) =$ "cyclic lists of elements of X"
- Alternative idea: Take advantage of the higher-dimensional structure of types in HoTT. E.g. $\mathcal{M}_f$ can be defined as a polynomial functor on *groupoids*:

$$\mathcal{M}_f^G(X) = \Sigma(Y : \mathsf{FinSet}).\ \langle Y \rangle \to X$$

Better support for coinduction?

- Support for programming and reasoning with coinductive type in (Cubical) Agda is limited, corecursive definitions need to pass very stringent syntactic checks that ensure their productivity

Better support for coinduction?

- Support for programming and reasoning with coinductive type in (Cubical) Agda is limited, corecursive definitions need to pass very stringent syntactic checks that ensure their productivity

- One solution is to change the metalanguage, moving to a type theory where all corecursive definitions are productive *by construction*
- Ticked Cubical type theory (Møgelberg & V.'19) has support for *guarded recursive types* involving HITs in their specification, e.g.

$$\nu\mathcal{P}_f \simeq \mathcal{P}_f(\triangleright\nu\mathcal{P}_f)$$

- The type $\triangleright X$ contains elements of X available in the next time step

Better support for coinduction?

- Support for programming and reasoning with coinductive type in (Cubical) Agda is limited, corecursive definitions need to pass very stringent syntactic checks that ensure their productivity

- One solution is to change the metalanguage, moving to a type theory where all corecursive definitions are productive *by construction*
- Ticked Cubical type theory (Møgelberg & V.'19) has support for *guarded recursive types* involving HITs in their specification, e.g.

$$\nu\mathcal{P}_f \simeq \mathcal{P}_f(\triangleright\nu\mathcal{P}_f)$$

- The type $\triangleright X$ contains elements of X available in the next time step
- Implementation of Ticked Cubical TT in Agda lead to formalization of fully-abstract model of π-calculus (V. & Vezzosi'20)

Better support for coinduction?

- Support for programming and reasoning with coinductive type in (Cubical) Agda is limited, corecursive definitions need to pass very stringent syntactic checks that ensure their productivity

- One solution is to change the metalanguage, moving to a type theory where all corecursive definitions are productive *by construction*
- Ticked Cubical type theory (Møgelberg & V.'19) has support for *guarded recursive types* involving HITs in their specification, e.g.

$$\nu\mathcal{P}_f \simeq \mathcal{P}_f(\triangleright\nu\mathcal{P}_f)$$

- The type $\triangleright X$ contains elements of X available in the next time step
- Implementation of Ticked Cubical TT in Agda lead to formalization of fully-abstract model of π-calculus (V. & Vezzosi'20)

- Full support for proper coinductive types in recent Clocked type theory (Kristensen, Møgelberg, Vezzosi'22)

Takeaway

Constructing final colalgebras of finitary functors in (variants) of Agda

- Polynomial functors (e.g. DFAs):
  1. Use Agda's primitive coinductive types
  2. Internalization as ω-limit (Ahrens et al.'15)
- Finite powerset (and also LTSs):
  1. Use setoids and coinductive types
  2. Internalization as $\omega + \omega$-limit (V.'21, assuming classical principles)
- Finite bag and analytic functors (WIP):
  1. Use setoids and coinductive types
  2. Internalization as ω-limit
  3. See analytic functors as polynomials on groupoids
- Altenative strategy: use Guarded/Clocked TT

Future work

- Finish the construction of $\nu\mathcal{M}_f$ and final coalgebras of analytic functors in Cubical Agda

Future work

- Finish the construction of $\nu\mathcal{M}_f$ and final coalgebras of analytic functors in Cubical Agda
- Motivate the existence in Cubical Agda of coinductive types employing HITs in their definition, e.g. $\nu\mathcal{P}_f$, $\nu\mathcal{M}_f$, $\mathcal{LTS}$, by constructing them in the cubical set model

Future work

- Finish the construction of $\nu\mathcal{M}_f$ and final coalgebras of analytic functors in Cubical Agda
- Motivate the existence in Cubical Agda of coinductive types employing HITs in their definition, e.g. $\nu\mathcal{P}_f$, $\nu\mathcal{M}_f$, $\mathcal{LTS}$, by constructing them in the cubical set model
- Development of further universal coalgebra in type theory, plus formalization in Cubical Agda of denotational semantics of languages with nondeterministic/probabilistic behaviour