Monoidal Computer III: A coalgebraic view of computational complexity

Dusko Pavlovic and Muzamil Yahia

CMCS 2018
Thessaloniki, April 2018

Intro
CC to MC
MMC to CMC
Step counting
Speedup etc
Work

Outline

Introduction

Background: Monoidal computer

Approach: Coalgebraic view

Result: Complexity evaluators

Applications: Speedup, gap, approximation, . . . coalgebraically

Work: One-way and trapdoor

Outline

Introduction

Background: Monoidal computer

Approach: Coalgebraic view

Result: Complexity evaluators

Applications: Speedup, gap, approximation, . . . coalgebraically

Work: One-way and trapdoor

Coauthor

Muzamil Yahia

Intro

CC to MC

MMC to CMC

Step counting

Speedup etc

Work

Message of the paper

Background

Coalgebra captures dynamics and stateful behavior

News

Coalgebra provides gauges to measure complexity

Outline

Introduction

Background: Monoidal computer

Definition and structure

Examples

Fundamental Theorem

Some consequences

Approach: Coalgebraic view

Result: Complexity evaluators

Cartesian closed category

$$\mathcal{C}(X, [A, B]) \underset{\lambda_X^{AB}}{\overset{\varepsilon_X^{AB}}{\rightleftarrows}} \mathcal{C}(X \times A, B) \quad (\cong)$$

Monoidal computer

$$\frac{\mathcal{C}(X, [A, B]) \underset{\lambda_X^{AB}}{\overset{\varepsilon_X^{AB}}{\underset{\cong}{\rightleftarrows}}} \mathcal{C}(X \times A, B)}{\mathsf{C}^{\bullet}(X, \mathbb{P}) \xrightarrow{\gamma_X^{AB}} \mathsf{C}(X \otimes A, B)}$$

Complexity by Coalgebra

DP and MY

Intro

CC to MC

Structure

Examples

Fundamental Theorem

Some consequences

MMC to CMC

Step counting

Speedup etc

Work

Monadic monoidal computer

$$\frac{\mathcal{C}(X, [A, B]) \underset{\lambda_X^{AB}}{\overset{\varepsilon_X^{AB}}{\underset{\longleftarrow}{\overset{\longrightarrow}{\cong}}}} \mathcal{C}(X \times A, B)}{\mathbb{C}(X, \mathbb{P}) \overset{\gamma_X^{AB}}{\twoheadrightarrow} \mathbb{C}(X \times A, MB)}$$

where M is a commutative monad on $\mathbb{C}$

Monadic monoidal computer coalgebraically

$$\frac{\mathcal{C}(X,[A,B]) \underset{\lambda_X^{AB}}{\overset{\varepsilon_X^{AB}}{\underset{\leftarrow}{\overset{\rightarrow}{\cong}}}} \mathcal{C}(X\times A,B)}{\mathbb{C}(X,\mathbb{P}) \xrightarrow{\nu_X^{AB}} \mathbb{C}\big(X\times A, M(X\times B)\big)}$$

where $M : \mathbb{C} \to \mathbb{C}$ is a commutative monad

Complexity by Coalgebra

DP and MY

Intro

CC to MC

Structure

Examples

Fundamental Theorem

Some consequences

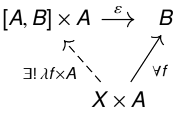
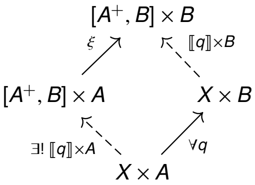
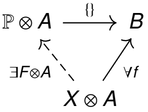
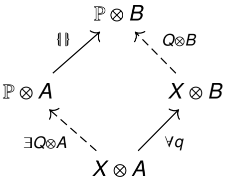
MMC to CMC

Step counting

Speedup etc

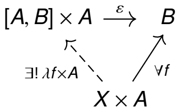
Work

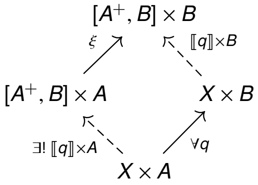
From CC to MC

models	static	dynamic
extensional models: Cartesian Closed	 abstractions $\overset{\varepsilon}{\underset{\lambda}{\longleftrightarrow}}$ applications	 behaviors $\xleftarrow{[\![-]\!]}$ machines
intensional models: Monoidal Computers	 programs $\twoheadrightarrow$ computations	 adaptive prog's $\rightharpoonup$ processes

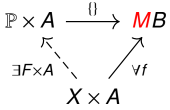
From CC to MMC

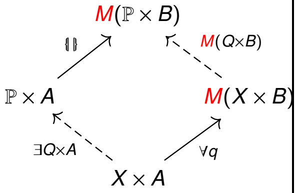
<table>
<tr><th>models</th><th>static</th><th>dynamic</th></tr>
<tr>
<td>extensional models:

Cartesian Closed</td>
<td>

abstractions $\overset{\varepsilon}{\underset{\lambda}{\longleftrightarrow}}$ applications</td>
<td>

behaviors $\overset{[\![-]\!]}{\longleftarrow}$ machines</td>
</tr>
<tr>
<td>intensional models:

Monadic Monoidal Computers</td>
<td>

programs $\twoheadrightarrow$ computations</td>
<td>

adaptive prog's $\rightharpoonup$ processes</td>
</tr>
</table>

MC structure

Proposition

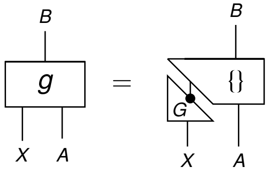
The following structures are equivalent

a) X-natural transformation $\mathbb{C}(X,\mathbb{P}) \xrightarrow{\gamma_X^{AB}} \mathbb{C}(X\times A, MB)$

b) a *universal evaluator* $\{\} \in \mathbb{C}(\mathbb{P}\times A, MB)$, such that

- $\forall g \in \mathbb{C}(X\times A, MB)\ \ \exists G \in \mathbb{C}(X,\mathbb{P})$

$$g(x,a) \quad = \quad \big\{G(x)\big\}a$$



MC structure

Proposition

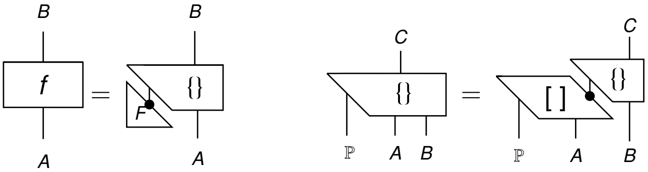
The following structures are equivalent

a) X-natural transformation $\mathbb{C}(X, \mathbb{P}) \xrightarrow{\gamma_X^{AB}} \mathbb{C}(X \times A, MB)$

b) a *universal evaluator* $\{\}^{AB} \in \mathbb{C}(\mathbb{P} \times A, MB)$, and
a *partial evaluator* $[]^{(AB)C} \in \mathbb{C}(\mathbb{P} \times A, \mathbb{P})$, such that

- $\forall f \in \mathbb{C}(A, MB)\ \exists F \in \mathbb{C}(X, \mathbb{P})$

$$f(a) \;=\; \{F\}a \qquad\qquad \{G\}(a,b) \;=\; \{[G]\,a\}b$$



Complexity by Coalgebra

DP and MY

Intro

CC to MC

Structure

Examples

Fundamental Theorem

Some consequences

MMC to CMC

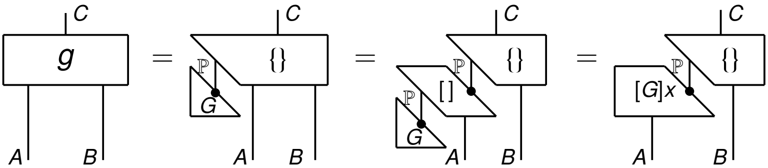
Step counting

Speedup etc

Work

MC structure

Overview



$$g(x, y) = \{G\}(x, y) = \{[G]\, x\}y$$

Complexity by Coalgebra

DP and MY

Intro

CC to MC

Structure

Examples

Fundamental Theorem

Some consequences

MMC to CMC

Step counting

Speedup etc

Work

MC structure

Definition

A *monoidal computer (MC)* is a

- monoidal category $\mathbb{C}$ with
- commutative comonoids $A \otimes A \xleftarrow{\Delta} A \xrightarrow{\top} I$ for all A
- a distinguished type $\mathbb{P}$ of programs
- equivalent structures from the Proposition.

MMC structure

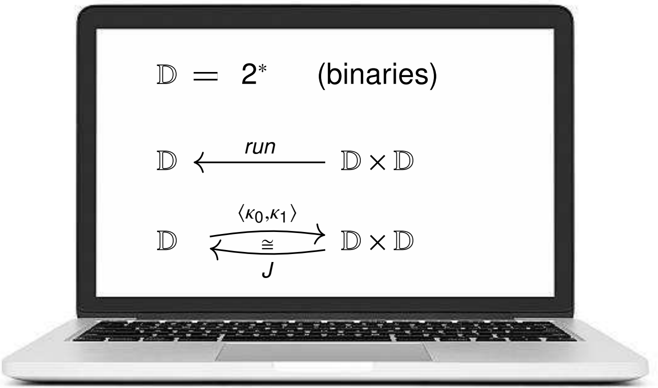
Definition

An *monadic monoidal computer (MMC)* is a

- cartesian category $\mathbb{C}$ with a
- commutative monad $M : \mathbb{C} \to \mathbb{C}$
- a distinguished type $\mathbb{P}$ of programs
- equivalent structures from the Proposition.

Examples?

Any computer is monoidal



Intro
CC to MC
Structure
Examples
Fundamental Theorem
Some consequences
MMC to CMC
Step counting
Speedup etc
Work

Computable universe $\mathbb{C}$

- **types:** sets where each element is tagged by some $\mathbb{D}$-labels

$$|\mathbb{C}| \quad = \quad \coprod_{A \in |\mathcal{S}|} \left\{ A \overset{\rho_A}{\twoheadleftarrow} |A| \subseteq \mathbb{D} \right\}$$

- **morphisms:** functions that are traced on the tags by $\mathbb{D}$-implementable computations

$$\mathbb{C}(A,B) \quad = \quad \left\{ f \in \mathcal{S}(A,B) \;\middle|\; \exists F \in \mathbb{D}. \begin{array}{ccc} |X| & \xrightarrow{run(F)} & |B| \\ \rho_A \downarrow\!\!\downarrow & & \downarrow\!\!\downarrow \rho_B \\ A & \xrightarrow{f} & B \end{array} \right\}$$

Computable monads: Maybe

$$\begin{array}{rcl} ? : \mathbb{C} & \to & \mathbb{C} \\ A & \longmapsto & \left(1 + A \xleftarrow{run(-,0)} |?A| \right) \end{array}$$

where

$$|?A| \quad = \quad \left\{ F \in \mathbb{D} \mid run(F,0)\downarrow \Longrightarrow \; run(F,0) \in A \right\}$$

Computable monads: Power

$$\wp : \mathbb{C} \to \mathbb{C}$$
$$A \longmapsto \left(\wp A \twoheadleftarrow |\wp A| \right)$$

where

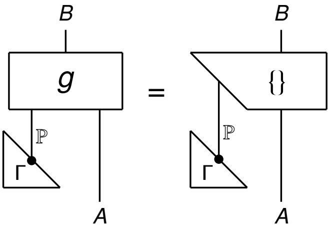
$$|\wp A| = \left\{ F \in \mathbb{D} \mid \forall a \in A.\ run(F, a)\downarrow \ \wedge\ run(F, a) \in \{0, 1\} \right\}$$
$$\wp A = |\wp A| \big/ \equiv \quad \text{where}$$
$$F \equiv G \iff run(F) = run(G)$$

The Fundamental Theorem of Computability

Theorem

For every computation $g \in \mathbb{C}(\mathbb{P} \times A, MB)$ there is a program $\Gamma \in \mathbb{C}(\mathbb{P})$ such that

$$g(\Gamma, a) \quad = \quad \{\Gamma\}\, a$$

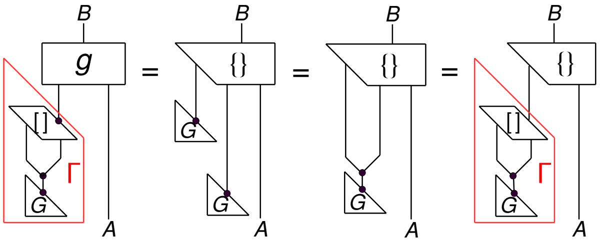


Γ is *Kleene's fixed point* of g.

The Fundamental Theorem of Computability

Proof

$$g(\Gamma, a) = g\Big([G, G], a\Big) = \{G\}(G, a) = \Big\{[G, G]\Big\}a = \{\Gamma\}\, a$$



Complexity by Coalgebra

DP and MY

Intro

CC to MC

Structure

Examples

Fundamental Theorem

Some consequences

MMC to CMC

Step counting

Speedup etc

Work

Complete MCs

Proposition about idempotents

An MC is finitely complete and cocomplete if and only if it is Cauchy complete, i.e. if the idempotents split in it.

Computability monads are partial

Proposition about partiality

Every MC contains partial maps.

Computability monads are partial

Proposition about partiality

Every MC contains partial maps.

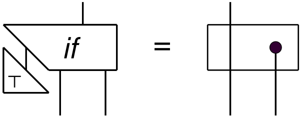
If there is an MMC over a monad $M : \mathbb{C} \to \mathbb{C}$, then

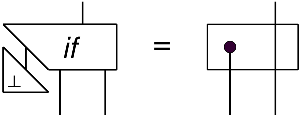
$$? \quad \subseteq \quad M$$

Natural numbers in MCs

Branching

$$if(b,x,y) = \begin{cases} x & \text{if } b = \top \\ y & \text{if } b = \bot \end{cases}$$

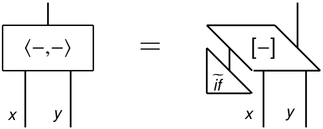




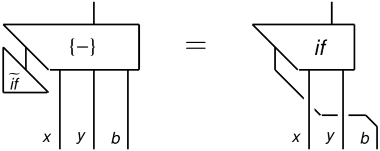
Natural numbers in MCs

Pairing

Define $\langle -, - \rangle : \mathbb{P} \times \mathbb{P} \to \mathbb{P}$



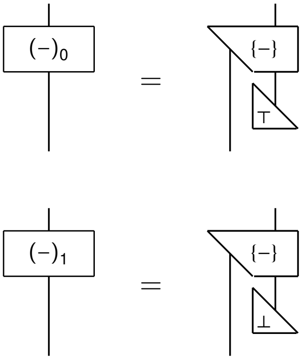
where



Natural numbers in MCs

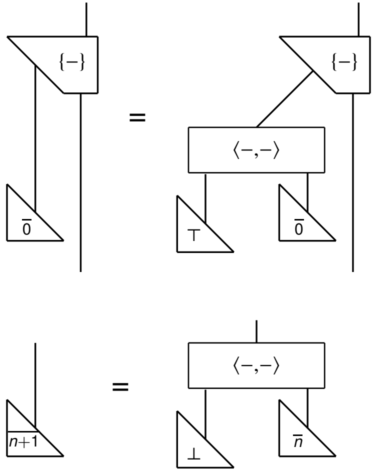
Projections

Define $(-)_0, (-)_1 : \mathbb{P} \to \mathbb{P}$



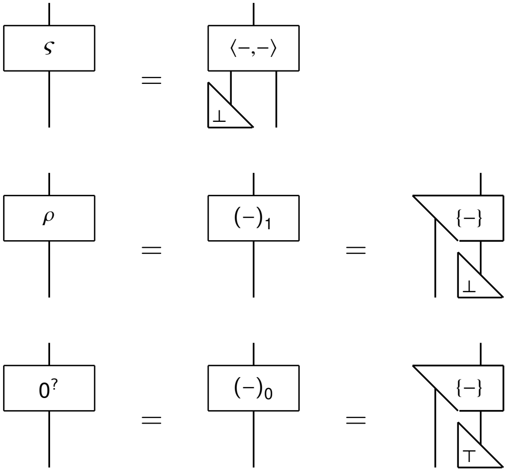
Natural numbers in MCs

Numbers as programs



Natural numbers in MCs

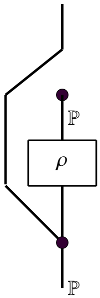
Successor, predecessor, and zero test



Natural numbers in MCs

$\mathbb{N}$ as idempotent splitting

The type $\mathbb{N}$ can be defined as the domain of ρ.



Natural numbers in MCs

Proposition about $\mathbb{N}$

Every complete MC has a natural numbers object.

Outline

Introduction

Background: Monoidal computer

Approach: Coalgebraic view

Result: Complexity evaluators

Applications: Speedup, gap, approximation, . . . coalgebraically

Work: One-way and trapdoor

Step computations

Definition

An *M-step computation* from A to B is a morphism

$$X \times A \xrightarrow{q} M(X \times B)$$

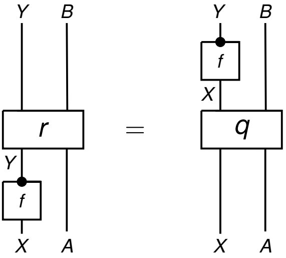
in an MMC $\mathbb{C}$ with a commutative monad M, where

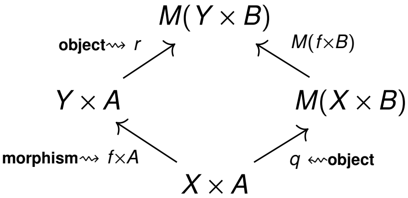
- A is the type of inputs
- B is the type of outputs
- X is the state space

Step computations

Definition

A *step evaluation* of the M-step computation $X \times A \xrightarrow{q} M(X \times B)$ in another M-step computation $Y \times A \xrightarrow{r} M(Y \times B)$, is a morphism $f \in \mathbb{C}(X, Y)$ with

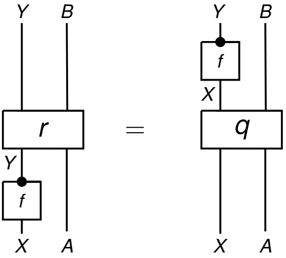


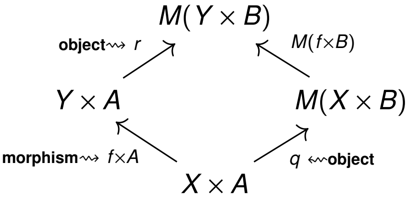


Step computations

Definition

A *step evaluation* of the M-step computation $X \times A \xrightarrow{q} M(X \times B)$ in another M-step computation $Y \times A \xrightarrow{r} M(Y \times B)$, is a morphism $f \in \mathbb{C}(X, Y)$ with



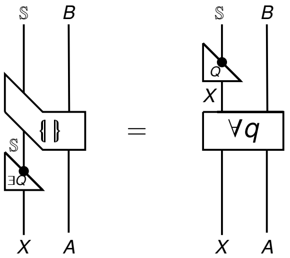


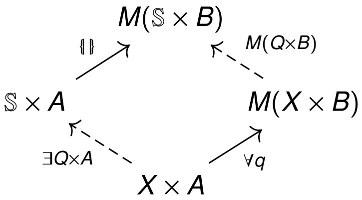
$\mathbb{C}_{MAB}$ denotes the category of M-step computations from A to B with step evaluations as morphisms.

Universal state space

Definition

A type $\mathbb{S}$ in an MMC $\mathbb{C}$ is a *universal state space* if for every A and B there is a weakly final M-step computation $\{\!\mid\, \mid\!\} \in \mathbb{C}\big(\mathbb{S} \times A, M(\mathbb{S} \times B)\big)$

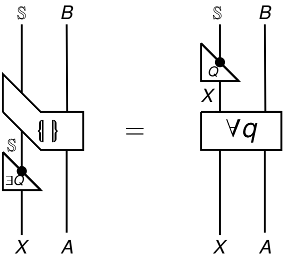


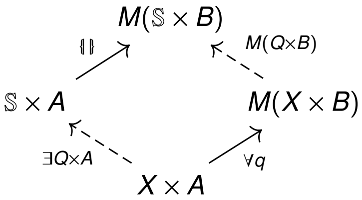


Universal step evaluator

Definition

A type $\mathbb{S}$ in an MMC $\mathbb{C}$ is a *universal state space* if for every A and B there is a weakly final M-step computation $\{\!|\,|\!\} \in \mathbb{C}\big(\mathbb{S} \times A, M(\mathbb{S} \times B)\big)$





This weakly final step computation $\{\!|\,|\!\}$ is a *universal step evaluator*.

Monoidal computer coalgebraically

Proposition

A cartesian category $\mathbb{C}$ with a commutive monad M is an MMC if and only if it has universal step evaluators.

Monoidal computer coalgebraically

Proposition

A cartesian category $\mathbb{C}$ with a commutive monad M is an MMC if and only if it has universal step evaluators.

There is a bijective correspondence between the families

- universal evaluators $\{\} \in \mathbb{C}(\mathbb{P} \times A, MB)$
- universal step evaluators $\{\!|\,|\!\} \in \mathbb{C}\big(\mathbb{S} \times A, M(\mathbb{S} \times B)\big)$

indexed over A and B in $\mathbb{C}$.

Monoidal computer coalgebraically

Proposition

A cartesian category $\mathbb{C}$ with a commutive monad M is an MMC if and only if it has universal step evaluators.

There is a bijective correspondence between the families

- universal evaluators $\{\} \in \mathbb{C}(\mathbb{P} \times A, MB)$
- universal step evaluators $\{\!|\,|\!\} \in \mathbb{C}\big(\mathbb{S} \times A, M(\mathbb{S} \times B)\big)$

indexed over A and B in $\mathbb{C}$.

The types $\mathbb{P}$ and $\mathbb{S}$ can be taken to coincide.

Monoidal computer coalgebraically

Proof (1)

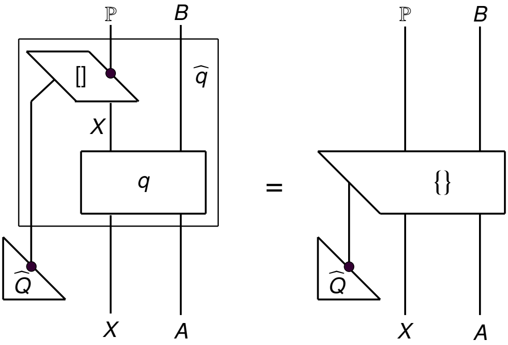
Given

- universal evaluator $\{\} \in \mathbb{C}(\mathbb{P} \times A, MB)$
- partial evaluator $[] \in \mathbb{C}(\mathbb{P} \times A, \mathbb{P})$ derived from $\{\}$,
- a step computation $q \in \mathbb{C}\big(X \times A, M(X \times B)\big)$

Monoidal computer coalgebraically

Proof (2)

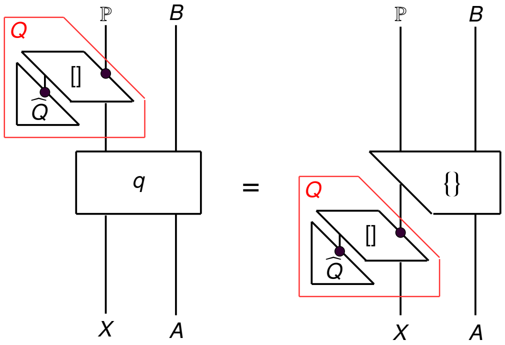
…construct a Kleene fixed point $\widehat{Q}$ of $\widehat{q}$



Monoidal computer coalgebraically

Proof (3)

The step evaluation $Q \in \mathbb{C}(X, \mathbb{P})$ is $Q(x) = \left[\widehat{Q}\right] x$.



Outline

Introduction

Background: Monoidal computer

Approach: Coalgebraic view

Result: Complexity evaluators

- Beyond time and space
- Kleene's normal form
- Recursion and approximation theorems

Applications: Speedup, gap, approximation, . . . coalgebraically

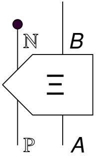
Ideas beyond Turing machine complexity

- **Kleene's *normal form***: traces as a hardness measure
- **Cobham's *intrinsic hardness***: it should only depend on functions, not on models
- **Blum's *abstract complexity***: machine independent complexity through *step counting*
  - time, space, alternations...
  - Kleene's trace
  - project to universal evaluators!

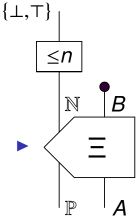
Complexity evaluators

Definition

A *complexity evaluator* is a computation $\Xi^{AB} \in \mathbb{C}\big(\mathbb{P} \times A, M(\mathbb{N} \times B)\big)$, such that

- 

is a universal evaluator

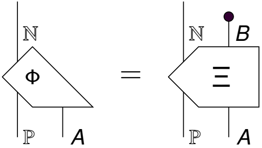
- 

is decidable

Step-counting functions

Definition

A *step-counting function* $\Phi^{AB} \in \mathbb{C}\big(\mathbb{P} \times A, M\mathbb{N}\big)$ is the first projection of a comlexity evaluator $\Xi^{AB} \in \mathbb{C}\big(\mathbb{P} \times A, M(\mathbb{N} \times B)\big)$



Step-counting functions project to evaluators

Theorem 1

There is a family of step-counting functions K such that

a) for every step-counting function Φ there is a primitive recursive function f with

$$\Phi \quad = \quad f \circ K$$

b) for every universal evaluator $\{-\}$ there is a primitive recursive function u with

$$\{-\} \quad = \quad u \circ K$$

Complexity evaluators

Theorem 2

Every complete MC contains all step counting functions.

Not presented

Outline

Introduction

Background: Monoidal computer

Approach: Coalgebraic view

Result: Complexity evaluators

Applications: Speedup, gap, approximation, . . . coalgebraically

Work: One-way and trapdoor

Not presented

Outline

Introduction

Background: Monoidal computer

Approach: Coalgebraic view

Result: Complexity evaluators

Applications: Speedup, gap, approximation, . . . coalgebraically

Work: One-way and trapdoor

Work

Question

Why yet another model of computation?

Work

Question

Why yet another model of computation?

Answer

Why is it that

- practice of computation has been revolutionized by high level programming languages, but
- theory of computation is still confined to low level machine languages

?

Work

Task 1

Teach computability and complexity to 2nd year students:
`http://www.asecolab.org/courses/ics-222/`

Task 2

Develop a reasonable theory of *absolute one-way functions*.