

Implicit automata in typed λ -calculi

Pierre PRADIC

Oxford University

j.w.w. NGUYỄN Lê Thành Dũng (a.k.a. Tito) (Paris 13)

GdT Plume, 8 Mars 2021

Simply typed functions on Church numerals

Church encodings of (unary) natural numbers:

- $\text{Nat} = (o \rightarrow o) \rightarrow o \rightarrow o$
- $n \in \mathbb{N} \rightsquigarrow \bar{n} = \lambda f. \lambda x. f (\dots (f x) \dots) : \text{Nat}$ with n times f
- all inhabitants of Nat are equal to some \bar{n} up to $=_{\beta\eta}$

Theorem (Schwichtenberg 1975)

The functions $\mathbb{N} \rightarrow \mathbb{N}$ definable by simply-typed λ -terms of type $\text{Nat} \rightarrow \text{Nat}$ are the extended polynomials (generated by $0, 1, +, \times, \text{id}$ and ifzero).

Simply typed functions on Church numerals

Church encodings of (unary) natural numbers:

- $\text{Nat} = (o \rightarrow o) \rightarrow o \rightarrow o$
- $n \in \mathbb{N} \rightsquigarrow \bar{n} = \lambda f. \lambda x. f (\dots (f x) \dots) : \text{Nat}$ with n times f
- all inhabitants of Nat are equal to some \bar{n} up to $=_{\beta\eta}$

Theorem (Schwichtenberg 1975)

The functions $\mathbb{N} \rightarrow \mathbb{N}$ definable by simply-typed λ -terms of type $\text{Nat} \rightarrow \text{Nat}$ are the extended polynomials (generated by $0, 1, +, \times, \text{id}$ and ifzero).

Let's add a bit of (meta-level) polymorphism: $t = \text{Nat}[A] \rightarrow \text{Nat}$

where $\text{Nat}[A] = \text{Nat}[A/o] = (A \rightarrow A) \rightarrow A \rightarrow A$

Open question

Choose some simple type A and some term $t : \text{Nat}[A] \rightarrow \text{Nat}$.

What functions $\mathbb{N} \rightarrow \mathbb{N}$ can be defined this way?

Simply typed functions on Church-encoded strings

To gain more insight, let's *generalize!* $\text{Nat} = \text{Str}_{\{1\}}$

Church encodings of *strings* over alphabet $\Sigma = \{a, b\}$:

- $\text{Str}_{\{a,b\}} = (o \rightarrow o) \rightarrow (o \rightarrow o) \rightarrow o \rightarrow o$
- $abb \in \{a, b\}^* \rightsquigarrow \overline{abb} = \lambda f_a. \lambda f_b. \lambda x. f_a (f_b (f_b x)) : \text{Str}_\Sigma$

More generally $\text{Str}_\Sigma = (o \rightarrow o) \rightarrow \dots |\Sigma| \text{ times } \dots \rightarrow (o \rightarrow o) \rightarrow o \rightarrow o$

Open question

Choose some simple type A and some term $t : \text{Str}_\Gamma[A] \rightarrow \text{Str}_\Sigma$.

What functions $\Gamma^* \rightarrow \Sigma^*$ can be defined this way?

Without input type substitutions, an answer is known [Zaionc 1987].

Simply typed functions on Church-encoded strings

To gain more insight, let's *generalize!* $\text{Nat} = \text{Str}_{\{1\}}$

Church encodings of *strings* over alphabet $\Sigma = \{a, b\}$:

- $\text{Str}_{\{a,b\}} = (o \rightarrow o) \rightarrow (o \rightarrow o) \rightarrow o \rightarrow o$
- $abb \in \{a, b\}^* \rightsquigarrow \overline{abb} = \lambda f_a. \lambda f_b. \lambda x. f_a (f_b (f_b x)) : \text{Str}_\Sigma$

More generally $\text{Str}_\Sigma = (o \rightarrow o) \rightarrow \dots |\Sigma| \text{ times } \dots \rightarrow (o \rightarrow o) \rightarrow o \rightarrow o$

Open question

Choose some simple type A and some term $t : \text{Str}_\Gamma[A] \rightarrow \text{Str}_\Sigma$.

What functions $\Gamma^* \rightarrow \Sigma^*$ can be defined this way?

Without input type substitutions, an answer is known [Zaionc 1987].

An answer for predicates [Hillebrand & Kanellakis 1996]

A subset of Σ^* is decidable by some $t : \text{Str}_\Sigma[A] \rightarrow \text{Bool}$

if and only if it is a *regular language*.

Note: unary regular languages \cong ultimately periodic subsets of \mathbb{N}

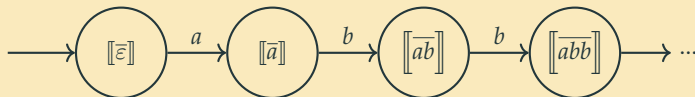
Theorem (Hillebrand & Kanellakis, LICS'96)

For any type A and any simply typed λ -term $t : \text{Str}_\Sigma[A] \rightarrow \text{Bool}$, the language $\{w \in \Sigma^* \mid t \bar{w} =_\beta \text{true}\}$ is regular.

Proof by semantic evaluation.

Let $\llbracket - \rrbracket$ stand for the denotational semantics in the CCC of finite sets.

We build an automaton with finite set of states $Q = \llbracket \text{Str}_\Sigma[A] \rrbracket$



$$t \bar{w} =_\beta \text{true} \iff \llbracket t \rrbracket(\llbracket \bar{w} \rrbracket) = \llbracket \text{true} \rrbracket \iff w \text{ accepted}$$

(Proof of (\Leftarrow) : if $\text{Card}(\llbracket o \rrbracket) \geq 2$ then $\llbracket \text{true} \rrbracket \neq \llbracket \text{false} \rrbracket$)

□

Similar ideas in higher-order model checking, e.g. Grellois & Mellies

Regular functions

Assume a λ -calculus for linear intuitionistic logic with additives

- $\lambda^\rightarrow x. t : A \rightarrow B$ unrestricted function
- $\lambda^\circ x. t : A \multimap B$ linear function (exactly one x in t)
- coproducts $A \oplus B$ and products $A \& B$

Church encoding with linear types [Girard 1987]:

$$\overline{abb} = \lambda^\rightarrow f_a. \lambda^\rightarrow f_b. \lambda^\circ x. f_a (f_b (f_b x)) : \text{Str}_{\{a,b\}} = (o \multimap o) \rightarrow (o \multimap o) \rightarrow o \multimap o$$

Regular functions

Assume a λ -calculus for linear intuitionistic logic with additives

- $\lambda^\rightarrow x. t : A \rightarrow B$ unrestricted function
- $\lambda^\circ x. t : A \multimap B$ linear function (exactly one x in t)
- coproducts $A \oplus B$ and products $A \& B$

Church encoding with linear types [Girard 1987]:

$$\overline{abb} = \lambda^\rightarrow f_a. \lambda^\rightarrow f_b. \lambda^\circ x. f_a (f_b (f_b x)) : \text{Str}_{\{a,b\}} = (o \multimap o) \rightarrow (o \multimap o) \rightarrow o \multimap o$$

Today's main theorem [Nguyễn & P.]

$f : \Gamma^* \rightarrow \Sigma^*$ is a *regular function*

\iff

f is defined by some $t : \text{Str}_\Gamma[A] \multimap \text{Str}_\Sigma$ in the intuitionistic linear λ -calculus with A *purely linear*, i.e. containing no ' \rightarrow '

Regular functions

Assume a λ -calculus for linear intuitionistic logic with additives

- $\lambda^\rightarrow x. t : A \rightarrow B$ unrestricted function
- $\lambda^\circ x. t : A \multimap B$ linear function (exactly one x in t)
- coproducts $A \oplus B$ and products $A \& B$

Church encoding with linear types [Girard 1987]:

$$\overline{abb} = \lambda^\rightarrow f_a. \lambda^\rightarrow f_b. \lambda^\circ x. f_a (f_b (f_b x)) : \text{Str}_{\{a,b\}} = (o \multimap o) \rightarrow (o \multimap o) \rightarrow o \multimap o$$

Today's main theorem [Nguyễn & P.]

$f : \Gamma^* \rightarrow \Sigma^*$ is a *regular function*

\iff

f is defined by some $t : \text{Str}_\Gamma[A] \multimap \text{Str}_\Sigma$ in the intuitionistic linear λ -calculus with A *purely linear*, i.e. containing no ' \rightarrow '

Regular functions are a classical topic, many equivalent definitions...

One of them: **copyless streaming string transducers** [Alur & Černý 2010]

\rightsquigarrow sounds suspiciously like affine types!

Definition

- Finite set of Σ^* -valued *registers* e.g. $R = \{X, Y\}$
- Initial values $R \rightarrow \Sigma^*$ e.g. $X_{\text{init}} = Y_{\text{init}} = \varepsilon$
- *Register update function* e.g. $a \mapsto \begin{cases} X := Xa \\ Y := aY \end{cases} \quad b \mapsto \begin{cases} X := Xb \\ Y := bY \end{cases}$
- “output function” e.g. $\text{out} = XY$

Definition

- Finite set of Σ^* -valued *registers* e.g. $R = \{X, Y\}$
- Initial values $R \rightarrow \Sigma^*$ e.g. $X_{\text{init}} = Y_{\text{init}} = \varepsilon$
- *Register update function* e.g. $a \mapsto \begin{cases} X := Xa \\ Y := aY \end{cases} \quad b \mapsto \begin{cases} X := Xb \\ Y := bY \end{cases}$
- “output function” e.g. $\text{out} = XY$

Execution over *abaa*: **start** with

$$X = \varepsilon \quad Y = \varepsilon$$

Definition

- Finite set of Σ^* -valued *registers* e.g. $R = \{X, Y\}$
- Initial values $R \rightarrow \Sigma^*$ e.g. $X_{\text{init}} = Y_{\text{init}} = \varepsilon$
- *Register update function* e.g. $a \mapsto \begin{cases} X := Xa \\ Y := aY \end{cases} \quad b \mapsto \begin{cases} X := Xb \\ Y := bY \end{cases}$
- “output function” e.g. $\text{out} = XY$

Execution over $abaa$:

$$X = a \quad Y = a$$

Definition

- Finite set of Σ^* -valued *registers* e.g. $R = \{X, Y\}$
- Initial values $R \rightarrow \Sigma^*$ e.g. $X_{\text{init}} = Y_{\text{init}} = \varepsilon$
- *Register update function* e.g. $a \mapsto \begin{cases} X := Xa \\ Y := aY \end{cases} \quad b \mapsto \begin{cases} X := Xb \\ Y := bY \end{cases}$
- “output function” e.g. $\text{out} = XY$

Execution over $abaa$:

$$X = ab \quad Y = ba$$

Definition

- Finite set of Σ^* -valued *registers* e.g. $R = \{X, Y\}$
- Initial values $R \rightarrow \Sigma^*$ e.g. $X_{\text{init}} = Y_{\text{init}} = \varepsilon$
- *Register update function* e.g. $a \mapsto \begin{cases} X := Xa \\ Y := aY \end{cases} \quad b \mapsto \begin{cases} X := Xb \\ Y := bY \end{cases}$
- “output function” e.g. $\text{out} = XY$

Execution over $abaa$:

$$X = aba \quad Y = aba$$

Definition

- Finite set of Σ^* -valued *registers* e.g. $R = \{X, Y\}$
- Initial values $R \rightarrow \Sigma^*$ e.g. $X_{\text{init}} = Y_{\text{init}} = \varepsilon$
- *Register update function* e.g. $a \mapsto \begin{cases} X := Xa \\ Y := aY \end{cases} \quad b \mapsto \begin{cases} X := Xb \\ Y := bY \end{cases}$
- “output function” e.g. $\text{out} = XY$

Execution over aba :

$$X = abaa \quad Y = aaba$$

Definition

- Finite set of Σ^* -valued *registers* e.g. $R = \{X, Y\}$
- Initial values $R \rightarrow \Sigma^*$ e.g. $X_{\text{init}} = Y_{\text{init}} = \varepsilon$
- *Register update function* e.g. $a \mapsto \begin{cases} X := Xa \\ Y := aY \end{cases} \quad b \mapsto \begin{cases} X := Xb \\ Y := bY \end{cases}$
- “output function” e.g. $\text{out} = XY$

Execution over $abaa$: $f(abaa) = abaaaaba$

$$X = abaa \quad Y = aaba$$

Definition

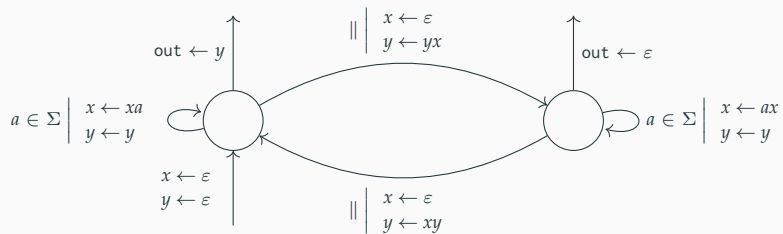
- Finite set of Σ^* -valued *registers* e.g. $R = \{X, Y\}$
- Initial values $R \rightarrow \Sigma^*$ e.g. $X_{\text{init}} = Y_{\text{init}} = \varepsilon$
- *Register update function* e.g. $a \mapsto \begin{cases} X := Xa \\ Y := aY \end{cases} \quad b \mapsto \begin{cases} X := Xb \\ Y := bY \end{cases}$
- “output function” e.g. $\text{out} = XY$

Execution over $abaa$: $f(abaa) = abaaaaba$, $f: w \mapsto w \cdot \text{reverse}(w)$

$$X = abaa \quad Y = aaba$$

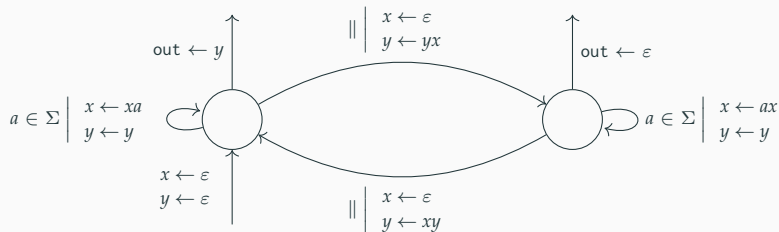
Stateful streaming string transducers

SSTs can also have *states*: their memory is $Q \times (\Sigma^*)^R$ (with $|Q| < \infty$)



Stateful streaming string transducers

SSTs can also have *states*: their memory is $Q \times (\Sigma^*)^R$ (with $|Q| < \infty$)



Copylessness restriction

Each register appears *at most once* on RHS of \leftarrow

(for each fixed input letter, at most once among all the associated \leftarrow)

Intuition: memory $M = Q \otimes \Sigma^* \otimes \dots \otimes \Sigma^*$, transitions $M \multimap M$

($Q \cong 1 \oplus \dots \oplus 1$, $\text{concat} : \Sigma^* \otimes \Sigma^* \multimap \Sigma^*$)

A framework for “single-pass” automata [Colcombet & Petrişan 2017]

- internal memory = object of a *category* \mathcal{C}
- transitions = morphisms (and [letter \mapsto transition] = functor $\mathcal{T}_\Sigma \rightarrow \mathcal{C}$)

$$\mathcal{T}_\Sigma = \bullet \longrightarrow \bullet \xrightarrow{a \in \Sigma} \bullet \longrightarrow \bullet \longrightarrow \mathcal{C}$$

- DFA = automata over the category of finite sets
- Copyless SSTs \approx start from a category \mathcal{R} of copyless register updates
+ add states by *free finite coproduct completion* $(-)_\oplus$

Categorical automata

A framework for “single-pass” automata [Colcombet & Petrişan 2017]

- internal memory = object of a *category* \mathcal{C}
- transitions = morphisms (and [letter \mapsto transition] = functor $\mathcal{T}_\Sigma \rightarrow \mathcal{C}$)

$$\mathcal{T}_\Sigma = \begin{array}{c} a \in \Sigma \\ \curvearrowright \\ \bullet \longrightarrow \bullet \longrightarrow \bullet \end{array} \longrightarrow \mathcal{C}$$

- DFA = automata over the category of finite sets
- Copyless SSTs \approx start from a category \mathcal{R} of copyless register updates
+ add states by *free finite coproduct completion* $(-)_\oplus$

Formally

A streaming setting \mathfrak{C} with output X is a tuple $(\mathcal{C}, \top, \perp, out)$ with

- \mathcal{C} a category
- \top and \perp objects of \mathcal{C}
- $out : \text{Hom}_{\mathcal{C}}(\top, \perp) \rightarrow X$ a set-theoretic-map

Notion of \mathfrak{C} -automaton

(abusively called \mathcal{C} -automata in the sequel)

The register category with output alphabet Σ

- **Objects:** finite sets R, S think register variables
- **Morphisms:** $\text{Hom}_{\mathcal{R}}(R, S) = \text{maps } S \rightarrow (R + \Sigma)^*$ corresponding to copyless register affectations
$$\sum_{s \in S} |f(s)|_r \leq 1$$
- Monoidal with $\otimes = +$
- Free affine monoidal category over an object $\Sigma^* = \{\bullet\}$, morphisms $\varepsilon, a : \mathbf{I} \rightarrow \Sigma^*$ for $a \in \Sigma$ and $\text{cat} : \Sigma^* \otimes \Sigma^* \rightarrow \Sigma^*$
- For the streaming setting, take $\top = \mathbf{I} = 0$ and $\perp = \Sigma^* = \{\bullet\}$

The register category with output alphabet Σ

- **Objects:** finite sets R, S think register variables
- **Morphisms:** $\text{Hom}_{\mathcal{R}}(R, S) = \text{maps } S \rightarrow (R + \Sigma)^*$ corresponding to copyless register affectations $\sum_{s \in S} |f(s)|_r \leq 1$
- Monoidal with $\otimes = +$
- Free affine monoidal category over an object $\Sigma^* = \{\bullet\}$, morphisms $\varepsilon, a : \mathbf{I} \rightarrow \Sigma^*$ for $a \in \Sigma$ and $\text{cat} : \Sigma^* \otimes \Sigma^* \rightarrow \Sigma^*$
- For the streaming setting, take $\top = \mathbf{I} = 0$ and $\perp = \Sigma^* = \{\bullet\}$

Definition of the free finite coproduct completion \mathcal{C}_{\oplus}

- **Objects:** formal finite sums $\bigoplus_{u \in U} C_u$ of objects of \mathcal{C} formally pairs $(U, (C_u)_{u \in U})$, U a finite set, $C_u \in \mathcal{C}_0$
- **Morphisms:** $\text{Hom}_{\mathcal{C}_{\oplus}}(\bigoplus_u C_u, \bigoplus_v D_v) = \prod_u \sum_v \text{Hom}_{\mathcal{C}}(C_u, D_v)$ $\cong \sum_f \prod_u \text{Hom}_{\mathcal{C}}(C_u, D_{f(u)})$
- Morphisms $\bigoplus_{q \in Q} R \rightarrow \bigoplus_{q \in Q} R$ correspond to transitions in a SST
- Canonical embedding $\mathcal{C} \rightarrow \mathcal{C}_{\oplus}$ allows to lift streaming settings

Transductions definable in linear λ -calculus can be turned into automata over a category \mathcal{L} of purely linear λ -terms (w/ $\text{const } f_c : o \multimap o$ for $c \in \Sigma$)

Claim

\mathcal{L} -automata compute the same string functions as λ -terms.

Proof: syntactic analysis of normal forms

Transductions definable in linear λ -calculus can be turned into automata over a category \mathcal{L} of purely linear λ -terms (w/ $\text{const } f_c : o \multimap o$ for $c \in \Sigma$)

Claim

\mathcal{L} -automata compute the same string functions as λ -terms.

Proof: syntactic analysis of normal forms

Compiling into higher-order transducers

Transductions definable in linear λ -calculus can be turned into automata over a category \mathcal{L} of purely linear λ -terms (w/ $\text{const } f_c : o \multimap o$ for $c \in \Sigma$)

Claim

\mathcal{L} -automata compute the same string functions as λ -terms.

Proof: syntactic analysis of normal forms

Proof strategy for linear λ -definable \implies regular function

Define a *functor* $\mathcal{L} \rightarrow \mathcal{R}_\oplus$ preserving enough structure

Useful fact: there is a canonical functor from \mathcal{L} to any *symmetric monoidal closed category*

Unfortunately \mathcal{R}_\oplus is **not** monoidal closed...

Toward a monoidal closed category

So far, we encountered:

- \mathcal{L} : category of purely linear λ -terms (w/ $\text{const } f_c : o \multimap o$ for $c \in \Sigma$)
- \mathcal{R} : category of finite sets of registers and copyless assignments
- \mathcal{R}_\oplus : free finite coproduct completion of the latter (add states)

Now consider:

- the free finite *product* completion: $\mathcal{C} \mapsto \mathcal{C}_\& = ((\mathcal{C}^{\text{op}})_\oplus)^{\text{op}}$

Objects: formal products $\&_x C_x$

- the composite completion $\mathcal{C} \mapsto \mathcal{C}_\& \mapsto (\mathcal{C}_\&)_\oplus$

Objects: formal sums of products $\bigoplus_u \&_x C_{u,x}$

similar to de Paiva's *Dialectica* categories **DC**, think $\exists u. \forall x. \varphi(u, x)$

Goals toward our main theorem

- Structure: $(\mathcal{R}_\&)_\oplus$ has finite products and is monoidal closed
- Conservativity: $(\mathcal{R}_\&)_\oplus$ -automata and \mathcal{R}_\oplus -automata are equivalent

Tensorial products can be lifted to the completions

- The new tensorial products satisfy the additional laws

$$A \otimes (B \& C) \equiv (A \otimes B) \& (A \otimes C) \quad A \otimes (B \oplus C) \equiv (A \otimes B) \oplus (A \otimes C)$$

- In particular, $(\mathcal{C}_{\&})_{\oplus}$ has distributive cartesian products

$$A \& (B \oplus C) \equiv (A \& B) \oplus (A \& C)$$

When embedded in (co)presheafs \cong Day convolution

Structure (1): generic remarks $(\mathcal{C}_{\&})_{\oplus}$

Tensorial products can be lifted to the completions

- The new tensorial products satisfy the additional laws

$$A \otimes (B \& C) \equiv (A \otimes B) \& (A \otimes C) \quad A \otimes (B \oplus C) \equiv (A \otimes B) \oplus (A \otimes C)$$

- In particular, $(\mathcal{C}_{\&})_{\oplus}$ has distributive cartesian products

$$A \& (B \oplus C) \equiv (A \& B) \oplus (A \& C)$$

When embedded in (co)presheafs \cong Day convolution

Lemma (folklore observation about dependent Dialectica categories?)

If \mathcal{C} is symmetric monoidal and $(\mathcal{C}_{\&})_{\oplus}$ has the internal homs $A \multimap B$ for all $A, B \in \mathcal{C}$, then $(\mathcal{C}_{\&})_{\oplus}$ is symmetric monoidal closed.

$$\left(\bigoplus_{u \in U} \&_{x \in X_u} A_x \right) \multimap \left(\bigoplus_{v \in V} \&_{y \in Y_v} B_y \right) = \&_{u \in U} \bigoplus_{v \in V} \&_{y \in Y_v} \bigoplus_{x \in X_u} A_x \multimap B_y$$

Lemma

\mathcal{R}_\oplus has the internal homs $A \multimap B$ for all $A, B \in \mathcal{R}$.

The construction appears in the original SST paper [Alur & Černý 2010] without the categorical vocabulary.

$$\begin{cases} X := abXcY \\ Y := ba \end{cases} \rightsquigarrow \text{shape } \begin{cases} X := Z_1XZ_2Y \\ Y := Z_3 \end{cases} + \text{parameters } Z_1 = ab, \dots$$

copyless SST \implies finitely many shapes: use as states; registers for params

Lemma

\mathcal{R}_\oplus has the internal homs $A \multimap B$ for all $A, B \in \mathcal{R}$.

The construction appears in the original SST paper [Alur & Černý 2010] without the categorical vocabulary.

$$\begin{cases} X := abXcY \\ Y := ba \end{cases} \rightsquigarrow \text{shape } \begin{cases} X := Z_1XZ_2Y \\ Y := Z_3 \end{cases} + \text{ parameters } Z_1 = ab, \dots$$

copyless SST \implies finitely many shapes: use as states; registers for params

Conclusion

$(\mathcal{R}_{\&})_\oplus$ is symmetric monoidal closed (and almost affine).

Conservativity

Lemma

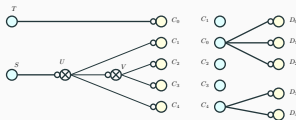
$(\mathcal{C}_{\&})_{\oplus}$ automata are equivalent to non-deterministic \mathcal{C}_{\oplus} automata.

A uniformization (\sim determinization) theorem is enough to conclude

Conservativity

$(\mathcal{R}_{\&})_{\oplus}$ -automata are equivalent to standard SSTs.

- Uniformization already known [Alur & Deshmuk 2011]
- Argument implicitly based on monoidal closure!



Theorem

For any monoidal category \mathcal{C} , if \mathcal{C}_{\oplus} has all the internal homsets $A \multimap B$ for $A, B \in \mathcal{C}$, then $(\mathcal{C}_{\&})_{\oplus}$ -automata and \mathcal{C}_{\oplus} -automata are equivalent.

i.e., ND \mathcal{C}_{\oplus} -automata can be uniformized

I have just discussed

Today's main theorem [Nguyễn & P.]

regular string function \iff definable by some $t : \text{Str}_\Gamma[A] \multimap \text{Str}_\Sigma$
in ILL with A purely linear

Main results

I have just discussed

Today's main theorem [Nguyễn & P.]

regular string function \iff definable by some $t : \text{Str}_\Gamma[A] \multimap \text{Str}_\Sigma$
in ILL with A purely linear

Using similar tools, analogous result for trees over ranked alphabets

Main theorem for trees [Nguyễn & P.]

regular *tree* function \iff definable by some $t : \text{Tree}_\Gamma[A] \multimap \text{Tree}_\Sigma$
in ILL with A purely linear

I have just discussed

Today's main theorem [Nguyễn & P.]

regular string function \iff definable by some $t : \text{Str}_\Gamma[A] \multimap \text{Str}_\Sigma$
in ILL with A purely linear

Using similar tools, analogous result for trees over ranked alphabets

Main theorem for trees [Nguyễn & P.]

regular *tree* function \iff definable by some $t : \text{Tree}_\Gamma[A] \multimap \text{Tree}_\Sigma$
in ILL with A purely linear

Specific ingredients:

- Bottom-up categorical tree automata over SMCs
- A reasonably elegant multicategory of tree registers transition \mathcal{R}
- Regular functions already known to correspond to $\mathcal{R}_{\oplus \&}$ -automata!

Additive connectives: why?

Additives are required for trees

Copyless streaming *tree* transducers \subset regular *tree* functions;
conjectured to be a *strict inclusion*.

To recover an equality: ad-hoc relaxation called “single use restriction”.

Additive connectives: why?

Additives are required for trees

Copyless streaming tree transducers \subset regular *tree* functions;
conjectured to be a *strict inclusion*.

To recover an equality: ad-hoc relaxation called “single use restriction”.

Principled explanation via linear logic:

just allow the *additive conjunction* in the internal memory!

$$\text{e.g. } M = Q \otimes \Sigma^* \otimes (\Sigma^* \& \Sigma^*) = \bigoplus_{q \in Q} \Sigma^* \otimes (\Sigma^* \& \Sigma^*)$$

Additive connectives: why?

Additives are required for trees

Copyless streaming tree transducers \subset regular *tree functions*;
conjectured to be a *strict inclusion*.

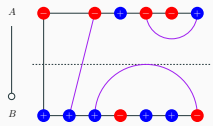
To recover an equality: ad-hoc relaxation called "single use restriction".

Principled explanation via linear logic:
just allow the *additive conjunction* in the internal memory!

$$\text{e.g. } M = Q \otimes \Sigma^* \otimes (\Sigma^* \& \Sigma^*) = \bigoplus_{q \in Q} \Sigma^* \otimes (\Sigma^* \& \Sigma^*)$$

String functions without additive

- Still an equivalence, but non-trivial (solution via Krohn–Rhodes)
 - Allows GoI-style interpretation in categories of diagrams
- \rightsquigarrow Interpretation as bidirectional automata (w/o registers)



Planar diagrams
 \rightsquigarrow
FO fragments

Today:

- Church encodings lead to connections with automata
- Additive connectives are important for trees
- Application of categorical semantics (Dialectica, GoI)

Broader picture

$\text{Str}_\Sigma[A] \multimap \text{Bool}$ with A linear (adapted as needed):

λ -calculus	languages	status
simply typed	regular	✓ [Hillebrand & Kanellakis 1996]
linear or affine	regular	✓
non-commutative linear or affine	star-free	✓

$\text{Str}_\Gamma[A] \multimap \text{Str}_\Sigma$ with A affine (adapted as needed):

λ -calculus	transducers	status
linear (without additives)	nothing interesting (?)	✓ (?)
affine	regular functions	✓ (coming soon)
non-commutative affine	first-order regular fn.	✓ ?
linear/affine with additives	regular functions	✓
parsimonious	polyregular	??
simply typed	variant of CPDA???	???

Today:

- Church encodings lead to connections with automata
- Additive connectives are important for trees
- Application of categorical semantics (Dialectica, GoI)

Broader picture

$\text{Str}_\Sigma[A] \multimap \text{Bool}$ with A linear (adapted as needed):

λ -calculus	languages	status
simply typed	regular	✓ [Hillebrand & Kanellakis 1996]
linear or affine	regular	✓
non-commutative linear or affine	star-free	✓

$\text{Str}_\Gamma[A] \multimap \text{Str}_\Sigma$ with A affine (adapted as needed):

λ -calculus	transducers	status
linear (without additives)	nothing interesting (?)	✓ (?)
affine	regular functions	✓ (coming soon)
non-commutative affine	first-order regular fn.	✓ ?
linear/affine with additives	regular functions	✓
parsimonious	polyregular	??
simply typed	variant of CPDA???	???

+ a characterization of $\text{Str}[A] \rightarrow \text{Str}$ as comparison-free polyregular functions

Conclusion

Today:

- Church encodings lead to connections with automata
- Additive connectives are important for trees
- Application of categorical semantics (Dialectica, GoI)

Broader picture

$\text{Str}_\Sigma[A] \multimap \text{Bool}$ with A linear (adapted as needed):

λ -calculus	languages	status
simply typed	regular	✓ [Hillebrand & Kanellakis 1996]
linear or affine	regular	✓
non-commutative linear or affine	star-free	✓

$\text{Str}_\Gamma[A] \multimap \text{Str}_\Sigma$ with A affine (adapted as needed):

λ -calculus	transducers	status
linear (without additives)	nothing interesting (?)	✓ (?)
affine	regular functions	✓ (coming soon)
non-commutative affine	first-order regular fn.	✓ ?
linear/affine with additives	regular functions	✓
parsimonious	polyregular	??
simply typed	variant of CPDA???	???

+ a characterization of $\text{Str}[A] \rightarrow \text{Str}$ as comparison-free polyregular functions

Thanks for listening!