Implicit automata in typed λ-calculi

Pierre Pradic
Oxford University
j.w.w. Nguyễn Lê Thành Dũng (a.k.a. Tito) (Paris 13)

LHC, February 5th, 2021

Simply typed functions on Church numerals

Church encodings of (unary) natural numbers:

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

Theorem (Schwichtenberg 1975)

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

Simply typed functions on Church numerals

Church encodings of (unary) natural numbers:

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

Theorem (Schwichtenberg 1975)

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

Let's add a bit of (meta-level) polymorphism: $t = \mathsf{Nat}[A] \to \mathsf{Nat}$
where $\mathsf{Nat}[A] = \mathsf{Nat}[A/o] = (A \to A) \to A \to A$

Open question

Choose some simple type A and some term $t : \mathsf{Nat}[A] \to \mathsf{Nat}$.
What functions $\mathbb{N} \to \mathbb{N}$ can be defined this way?

Simply typed functions on Church-encoded strings

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

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

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

More generally $\mathsf{Str}_\Sigma = (o \to o) \to \ldots |\Sigma| \text{ times} \ldots \to (o \to o) \to o \to o$

Open question

Choose some simple type A and some term $t : \mathsf{Str}_\Gamma[A] \to \mathsf{Str}_\Sigma$.
What functions $\Gamma^* \to \Sigma^*$ 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*! $\mathsf{Nat} = \mathsf{Str}_{\{1\}}$

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

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

More generally $\mathsf{Str}_\Sigma = (o \to o) \to \ldots |\Sigma| \text{ times} \ldots \to (o \to o) \to o \to o$

Open question

Choose some simple type A and some term $t : \mathsf{Str}_\Gamma[A] \to \mathsf{Str}_\Sigma$.
What functions $\Gamma^* \to \Sigma^*$ 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 $t : \mathsf{Str}_\Sigma[A] \to \mathsf{Bool}$
if and only if it is a *regular language*.

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

λ-definable functions are regular

Theorem (Hillebrand & Kanellakis, LICS'96)

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

Proof by semantic evaluation.

Let $[\![-]\!]$ stand for the denotational semantics in the *CCC of finite sets*.

We build an automaton with *finite* set of states $Q = [\![\mathsf{Str}_\Sigma[A]]\!]$

$$\begin{tikzcd} {} \arrow[r] & {[\![\varepsilon]\!]} \arrow[r, "a"] & {[\![\overline{a}]\!]} \arrow[r, "b"] & {\left[\!\!\left[\overline{ab}\right]\!\!\right]} \arrow[r, "b"] & {\left[\!\!\left[\overline{abb}\right]\!\!\right]} \arrow[r] & \cdots \end{tikzcd}$$

$$t\,\overline{w} =_\beta \mathtt{true} \iff [\![t]\!]([\![\overline{w}]\!]) = [\![\mathtt{true}]\!] \iff w \text{ accepted}$$

(Proof of $(\Leftarrow)$: if $\mathrm{Card}([\![o]\!]) \geq 2$ then $[\![\mathtt{true}]\!] \neq [\![\mathtt{false}]\!]$) $\square$

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

Regular functions

Assume a λ-calculus for linear intuitionistic logic with additives

- $\lambda^{\to} x.\, t : A \to B$ 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]:

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

Regular functions

Assume a λ-calculus for linear intuitionistic logic with additives

- $\lambda^{\to}x.\, t : A \to B$ 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]:

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

Today's main theorem [Nguyễn & P.]

$$f : \Gamma^* \to \Sigma^* \text{ is a } \textit{regular function}$$

$$\iff$$

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

Regular functions

Assume a λ-calculus for linear intuitionistic logic with additives

- $\lambda^{\to} x.\, t : A \to B$ 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]:

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

Today's main theorem [Nguyễn & P.]

$$f : \Gamma^* \to \Sigma^* \text{ is a } \textit{regular function}$$

$$\Longleftrightarrow$$

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

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 $R \to \Sigma^*$ 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. $\mathsf{out} = XY$

Definition

- Finite set of Σ^*-valued *registers* e.g. $R = \{X, Y\}$
- Initial values $R \to \Sigma^*$ 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. $\mathrm{out} = XY$

Execution over *abaa*: start with

$$X = \varepsilon \qquad Y = \varepsilon$$

Single-state streaming string transducers

Definition

- Finite set of Σ^*-valued *registers* e.g. $R = \{X, Y\}$
- Initial values $R \to \Sigma^*$ 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. $\mathrm{out} = XY$

Execution over *abaa*:

$$X = a \qquad Y = a$$

Definition

- Finite set of Σ^*-valued *registers* e.g. $R = \{X, Y\}$
- Initial values $R \to \Sigma^*$ 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. $\mathsf{out} = XY$

Execution over *abaa*:

$$X = ab \qquad Y = ba$$

Single-state streaming string transducers

Definition

- Finite set of Σ^*-valued *registers* e.g. $R = \{X, Y\}$
- Initial values $R \to \Sigma^*$ 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. $\mathsf{out} = XY$

Execution over *abaa*:

$$X = aba \qquad Y = aba$$

Single-state streaming string transducers

Definition

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

Execution over *abaa*:

$$X = abaa \qquad Y = aaba$$

Single-state streaming string transducers

Definition

- Finite set of Σ^*-valued *registers* e.g. $R = \{X, Y\}$
- Initial values $R \to \Sigma^*$ e.g. $X_{\mathrm{init}} = Y_{\mathrm{init}} = \varepsilon$
- *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. $\mathtt{out} = XY$

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

$$X = abaa \qquad Y = aaba$$

Single-state streaming string transducers

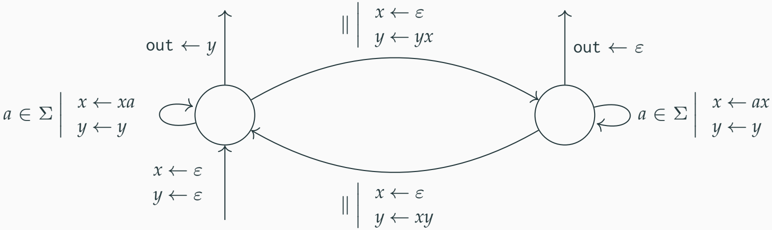
Definition

- Finite set of Σ^*-valued *registers* e.g. $R = \{X, Y\}$
- Initial values $R \to \Sigma^*$ 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. $\mathsf{out} = XY$

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

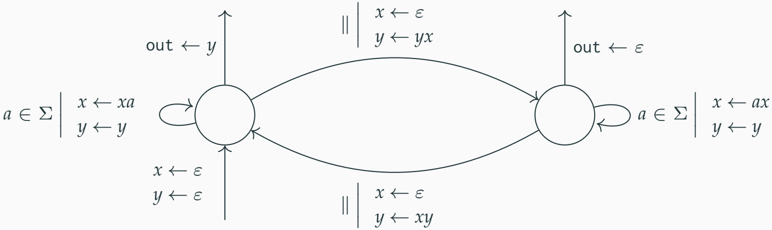
$$X = abaa \qquad Y = aaba$$

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 $M = Q \otimes \Sigma^* \otimes \ldots \otimes \Sigma^*$, transitions $M \multimap M$

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

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 $\mathcal{T}_\Sigma \to \mathcal{C}$)

$$\mathcal{T}_\Sigma \quad = \quad \begin{tikzcd} \bullet \arrow[r] & \bullet \arrow[loop above, "a\in\Sigma"] \arrow[r] & \bullet \end{tikzcd} \qquad \longrightarrow \qquad \mathcal{C}$$

- 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* $(-)_\oplus$

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

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

$$\mathcal{T}_\Sigma \quad = \quad \begin{tikzcd} \bullet \arrow[r] & \bullet \arrow[r] \arrow[loop above, "a \in \Sigma"] & \bullet \end{tikzcd} \qquad \longrightarrow \qquad \mathcal{C}$$

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

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

- **Objects:** formal finite sums $\bigoplus_{u \in U} C_u$ of objects of $\mathcal{C}$
- **Morphisms:** $\mathrm{Hom}_{\mathcal{C}_\oplus}\left(\bigoplus_u C_u, \bigoplus_v D_v\right) = \prod_u \sum_v \mathrm{Hom}_{\mathcal{C}}(C_u, D_v)$

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 $\mathcal{T}_\Sigma \to \mathcal{C}$)

$$\mathcal{T}_\Sigma \quad = \quad \begin{tikzcd} \bullet \arrow[r] & \bullet \arrow[r] \arrow[loop above, "a\in\Sigma"] & \bullet \end{tikzcd} \qquad \longrightarrow \qquad \mathcal{C}$$

- 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* $(-)_\oplus$

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:** $\mathrm{Hom}_{\mathcal{C}_\oplus}\left(\bigoplus_u C_u, \bigoplus_v D_v\right) = \prod_u \sum_v \mathrm{Hom}_{\mathcal{C}}(C_u, D_v)$

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 $\mathcal{T}_\Sigma \to \mathcal{C}$)

$$\mathcal{T}_\Sigma \quad = \quad \begin{tikzcd} \bullet \arrow[r] & \bullet \arrow[loop above, "a\in\Sigma"] \arrow[r] & \bullet \end{tikzcd} \quad \longrightarrow \quad \mathcal{C}$$

- 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* $(-)_\oplus$

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:** $\mathrm{Hom}_{\mathcal{C}_\oplus}\left(\bigoplus_u C_u, \bigoplus_v D_v\right) = \prod_u \sum_v \mathrm{Hom}_{\mathcal{C}}\left(C_u, D_v\right)$
  $\cong \sum_f \prod_u \mathrm{Hom}_{\mathcal{C}}\left(C_u, D_{f(u)}\right)$

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/ 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/ 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/ 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* $\mathcal{L} \to \mathcal{R}_\oplus$ preserving enough structure

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

Unfortunately $R_\oplus$ is **not** monoidal closed...

Toward a monoidal closed category

So far, we encountered:

- $\mathcal{L}$: category of purely linear λ-terms (w/ 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: $\mathcal{C} \mapsto \mathcal{C}_\& = ((\mathcal{C}^{\mathrm{op}})_\oplus)^{\mathrm{op}}$

  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

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

Tensorial products can be lifted to the completions

- The new tensorial products satisfy the additional laws

$$A \otimes (B \,\&\, C) \equiv (A \otimes B) \,\&\, (A \otimes C) \qquad A \otimes (B \oplus C) \equiv (A \otimes B) \oplus (A \otimes C)$$

- 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

$$A \otimes (B \mathbin{\&} C) \equiv (A \otimes B) \mathbin{\&} (A \otimes C) \qquad A \otimes (B \oplus C) \equiv (A \otimes B) \oplus (A \otimes C)$$

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

$$A \mathbin{\&} (B \oplus C) \equiv (A \mathbin{\&} B) \oplus (A \mathbin{\&} C)$$

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.

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

Structure (2): combinatorics on strings

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.

$$\begin{cases} X := abXcY \\ Y := ba \end{cases} \quad \rightsquigarrow \quad \text{shape} \begin{cases} X := Z_1 X Z_2 Y \\ Y := Z_3 \end{cases} \quad + \quad \text{parameters } Z_1 = ab, \ldots$$

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

Structure (2): combinatorics on strings

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.

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

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

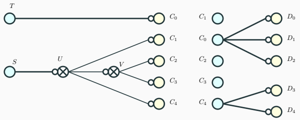
$(\mathcal{C}_{\&})_{\oplus}$ automata are equivalent to non-deterministic $\mathcal{C}_{\oplus}$ automata.

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

Main results

I have just discussed

Today's main theorem [Nguyễn & P.]

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

Specific ingredients:

- Bottom-up categorical tree automata over SMCs
- A comparison of $\mathcal{C}_\&$ with a kind of *coherence completion* — similar to [Hu, Joyal]
- A reasonably elegant multicategory of tree registers transition

Conclusion

Today:

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

Broader picture

$\mathsf{Str}_\Sigma[A] \multimap \mathsf{Bool}$ 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	✓

$\mathsf{Str}_\Gamma[A] \multimap \mathsf{Str}_\Sigma$ 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???	???

Conclusion

Today:

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

Broader picture

$\mathsf{Str}_\Sigma[A] \multimap \mathsf{Bool}$ 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	✓

$\mathsf{Str}_\Gamma[A] \multimap \mathsf{Str}_\Sigma$ 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 $\mathsf{Str}[A] \to \mathsf{Str}$ **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

$\mathsf{Str}_\Sigma[A] \multimap \mathsf{Bool}$ 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	✓

$\mathsf{Str}_\Gamma[A] \multimap \mathsf{Str}_\Sigma$ 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 $\mathsf{Str}[A] \to \mathsf{Str}$ as comparison-free polyregular functions

Thanks for listening!