Implicit automata in typed λ-calculi

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

Structure meets Power Workshop, June 28th 2021

Simply typed functions on Church numerals

Church encodings of (unary) natural numbers:

- $\mathsf{Nat} = (o \to o) \to o \to o$
- $n \in \mathbb{N} \rightsquigarrow \overline{n} = \lambda f.\ \lambda x.\ f\,(\dots\ (f\,x)\dots) : \mathsf{Nat}$ 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 $0, 1, +, \times$, `id` and `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* $0, 1, +, \times,$ `id` *and* `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 $\mathsf{Str}_\Sigma = (o \to o) \to \dots |\Sigma| \text{ times} \dots \to (o \to o) \to o \to o$

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

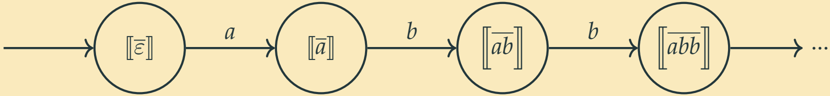
Theorem [Hillebrand & Kanellakis, LICS'96]

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

Proof by semantic evaluation.

Let $[\![-]\!]$ stand for a denotational semantics in the *CCC of finite sets*. (determined by $[\![o]\!]$)

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



λ-definable functions are regular

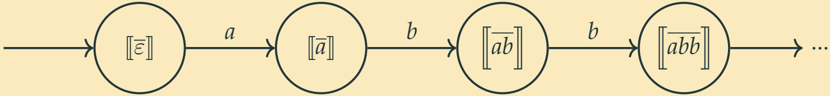
Theorem [Hillebrand & Kanellakis, LICS'96]

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

Proof by semantic evaluation.

Let $[\![-]\!]$ stand for a denotational semantics in the *CCC of finite sets*. (determined by $[\![o]\!]$)

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



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

λ-definable functions are regular

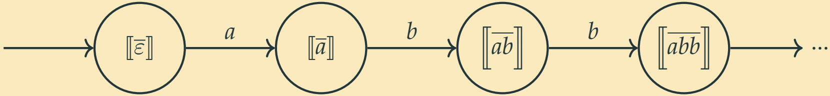
Theorem [Hillebrand & Kanellakis, LICS'96]

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

Proof by <u>semantic evaluation</u>.

Let $[\![-]\!]$ stand for a denotational semantics in the *CCC of finite sets*. (determined by $[\![o]\!]$)

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



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

To conclude: if $\mathrm{Card}([\![o]\!]) \geq 2$ then $[\![\mathtt{true}]\!] \neq [\![\mathtt{false}]\!]$) □

λ-definable functions are regular

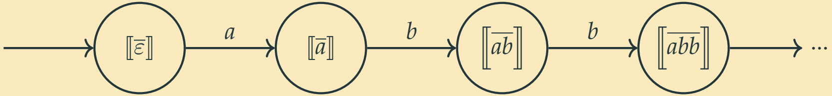
Theorem [Hillebrand & Kanellakis, LICS'96]

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

Proof by semantic evaluation.

Let $[\![-]\!]$ stand for a denotational semantics in the *CCC of finite sets*. (determined by $[\![o]\!]$)

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



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

To conclude: if $\mathrm{Card}([\![o]\!]) \geq 2$ then $[\![\texttt{true}]\!] \neq [\![\texttt{false}]\!]$) □

Similar ideas in higher-order model checking (see 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.]

$f : \Gamma^* \to \Sigma^*$ is a *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 $A \mathbin{\&} B$

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!

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. $a \mapsto \begin{cases} X := Xa \\ Y := aY \end{cases}$ $b \mapsto \begin{cases} X := Xb \\ Y := bY \end{cases}$ $c \mapsto \begin{cases} X := aba \\ Y := YabaX \end{cases}$
- "output function" e.g. $\text{out} = XY$

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. $a \mapsto \begin{cases} X := Xa \\ Y := aY \end{cases} \quad b \mapsto \begin{cases} X := Xb \\ Y := bY \end{cases} \quad c \mapsto \begin{cases} X := aba \\ Y := YabaX \end{cases}$
- "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. $a \mapsto \begin{cases} X := Xa \\ Y := aY \end{cases}$ $\quad b \mapsto \begin{cases} X := Xb \\ Y := bY \end{cases}$ $\quad c \mapsto \begin{cases} X := aba \\ Y := YabaX \end{cases}$
- "output function" e.g. $\text{out} = XY$

Execution over *abaa*:

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

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. $a \mapsto \begin{cases} X := Xa \\ Y := aY \end{cases}$ $b \mapsto \begin{cases} X := Xb \\ Y := bY \end{cases}$ $c \mapsto \begin{cases} X := aba \\ Y := YabaX \end{cases}$
- "output function" e.g. $\text{out} = XY$

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. $a \mapsto \begin{cases} X := Xa \\ Y := aY \end{cases}$ $\quad b \mapsto \begin{cases} X := Xb \\ Y := bY \end{cases}$ $\quad c \mapsto \begin{cases} X := aba \\ Y := YabaX \end{cases}$
- "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. $X_{\text{init}} = Y_{\text{init}} = \varepsilon$
- *Register update function* e.g. $a \mapsto \begin{cases} X := Xa \\ Y := aY \end{cases} \quad b \mapsto \begin{cases} X := Xb \\ Y := bY \end{cases} \quad c \mapsto \begin{cases} X := aba \\ Y := YabaX \end{cases}$
- "output function" e.g. $\text{out} = XY$

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. $X_{\text{init}} = Y_{\text{init}} = \varepsilon$
- *Register update function* e.g. $a \mapsto \begin{cases} X := Xa \\ Y := aY \end{cases}$ $b \mapsto \begin{cases} X := Xb \\ Y := bY \end{cases}$ $c \mapsto \begin{cases} X := aba \\ Y := YabaX \end{cases}$
- "output function" e.g. $\texttt{out} = XY$

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

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

Single-state streaming string transducers

Definition

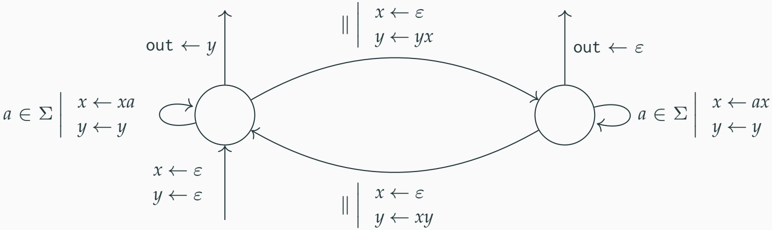
- Finite set of Σ^*-valued *registers* e.g. $R = \{X, Y\}$
- Initial values $R \to \Sigma^*$ e.g. $X_{\text{init}} = Y_{\text{init}} = \varepsilon$
- *Register update function* e.g. $a \mapsto \begin{cases} X := Xa \\ Y := aY \end{cases} \quad b \mapsto \begin{cases} X := Xb \\ Y := bY \end{cases} \quad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

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

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

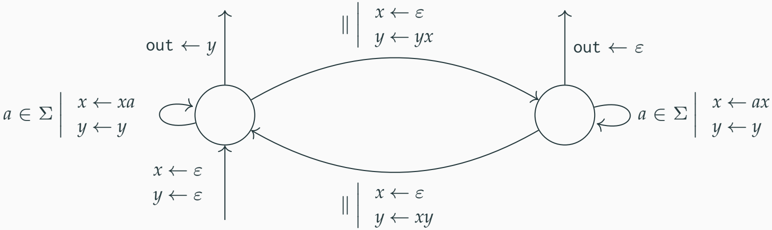
f restricted to $\{a, b\}^*$: corresponds to $w \mapsto w \cdot \mathtt{reverse}(w)$

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 $M \multimap M$

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

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

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

$$\mathcal{T}_\Sigma \quad = \quad \bullet \longrightarrow \overset{a \in \Sigma}{\overset{\circlearrowright}{\bullet}} \longrightarrow \bullet \qquad \longrightarrow \qquad \mathcal{C}$$

- 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 = $  $\quad \longrightarrow \quad \mathcal{C}$

- 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* $(-)_\oplus$

Formally

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

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

Notion of $\mathfrak{C}$-automaton (abusively called $\mathcal{C}$-automata in the sequel)

SSTs as categorical automata

The register category with output alphabet Σ

- **Objects:** finite sets R, S — think register variables
- **Morphisms:** $\mathrm{Hom}_{\mathcal{R}}(R, S) = $ maps $S \to (R + \Sigma)^*$ 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 $\varepsilon, a : \mathbf{I} \to \Sigma^*$ for $a \in \Sigma$ and $cat : \Sigma^* \otimes \Sigma^* \to \Sigma^*$
- For the streaming setting, take $\top = \mathbf{I} = 0$ and $\perp = \Sigma^* = \{\bullet\}$

SSTs as categorical automata

The register category with output alphabet Σ

- **Objects:** finite sets R, S think register variables
- **Morphisms:** $\mathrm{Hom}_{\mathcal{R}}(R, S) =$ maps $S \to (R + \Sigma)^*$ 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 $\varepsilon, a : \mathbf{I} \to \Sigma^*$ for $a \in \Sigma$ and $cat : \Sigma^* \otimes \Sigma^* \to \Sigma^*$
- For the streaming setting, take $\top\!\!\top = \mathbf{I} = 0$ and $\perp\!\!\perp = \Sigma^* = \{\bullet\}$

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

- Morphisms $\bigoplus_{q \in Q} R \quad \to \quad \bigoplus_{q \in Q} R$ correspond to transitions in a SST
- Canonical embedding $\mathcal{C} \to \mathcal{C}_\oplus$ allows to lift streaming settings

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* with (co)products

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/ 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

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

- 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

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.

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

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 parameters

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 parameters

Conclusion

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

Conservativity

Lemma

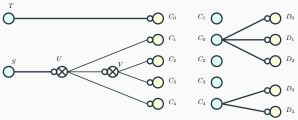
$(\mathcal{C}_{\&})_{\oplus}$ automata are equivalent to non-deterministic $\mathcal{C}_{\oplus}$ automata.

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.

equivalently: 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:

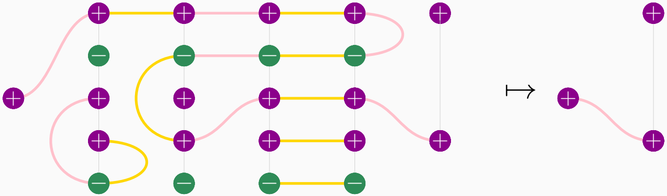
- Bottom-up categorical tree automata over SMCs
- A reasonably elegant multicategory of tree registers transition $\mathcal{R}$
  - Generated from the correponding PROP in a principled way (reminiscent from the notion of clone)
  - Argument for $\mathcal{R}$-monoidal closure argument generalizes to trees
- Regular functions already known to correspond to $\mathcal{R}_{\oplus\&}$-automata!

Dropping the additives

- Allows GoI-style interpretation in categories of diagrams ($\cong$ **Int**(FinPartInj))

$\rightsquigarrow$ Interpretation as two-way automata [Hines 2003]

Define regular languages



Consequence (not interesting)

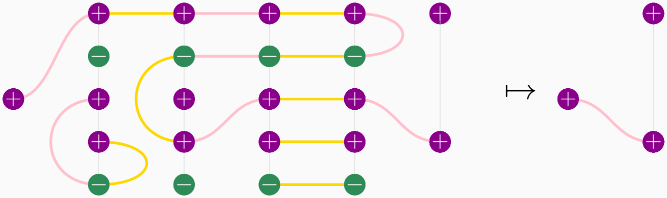
Every linear term $t : \mathsf{Str}_\Sigma[A] \multimap \mathsf{Bool}$ with A $\to$-free defines a regular language.

Dropping the additives and commutativity

- Allows GoI-style interpretation in categories of **planar** diagrams

$\rightsquigarrow$ Interpretation as two-way **planar** automata [Hines 2003,2006]

Define **star-free** languages



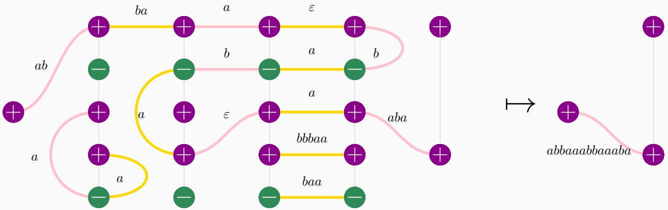
Consequence [Nguyễn, P. 2020]

Every **planar** linear term $t : \mathsf{Str}_\Sigma[A] \multimap \mathsf{Str}$ with A $\to$-free defines a star-free language.

- Allows GoI-style interpretation in categories of planar **labelled** diagrams

$\rightsquigarrow$ Interpretation as two-way planar **transducers** (2DFTs; w/o registers) [Hines 2003,2006]

Define **first-order** regular functions



Consequence

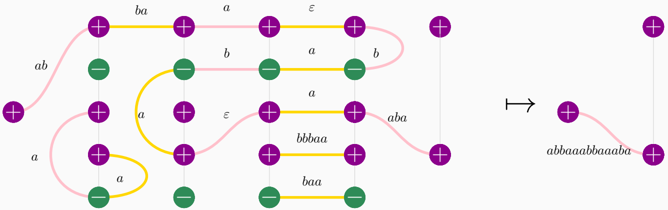
Every **planar** linear term $t : \mathsf{Str}_\Sigma[A] \multimap \mathsf{Str}$ with A $\to$-free defines a FO-transduction.

Dropping the additives and commutativity

- Allows GoI-style interpretation in categories of planar **labelled** diagrams

⇝ Interpretation as two-way planar **transducers** (2DFTs; w/o registers) [Hines 2003,2006]

Define **first-order** regular functions



Consequence

Every **planar** linear term $t : \mathsf{Str}_\Sigma[A] \multimap \mathsf{Str}$ with A $\to$-free defines a FO-transduction.

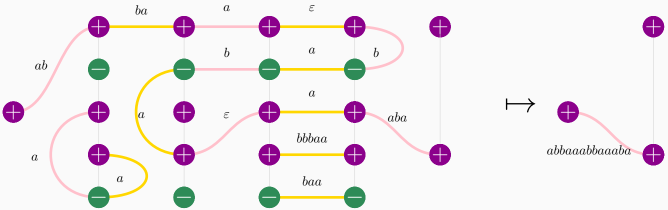
Alas, planar linear terms are much weaker than FO-transductions (preserve Parikh images)

Dropping the additives and commutativity

- Allows GoI-style interpretation in categories of planar **labelled** diagrams

$\rightsquigarrow$ Interpretation as two-way planar **transducers** (2DFTs; w/o registers) [Hines 2003,2006]

Define **first-order** regular functions



Conjecture

Every planar **affine** term $t : \mathsf{Str}_\Sigma[A] \multimap \mathsf{Str}$ with A $\rightarrow$-free defines a FO-transduction.

The converse holds (main ingredient for the proof: the Krohn-Rhodes theorem)

Conclusion

What happened here:

- Connections between Church encodings and automata
- Application of categorical semantics (Dialectica, geometry of interaction (GoI))
- A generic uniformization-like construction $(\mathcal{C}_{\&})_{\oplus} \to \mathcal{C}_{\oplus}$ for monoidal $\mathcal{C}$ with certain homsets

Some take-aways:

- Important ingredient in uniformization: monoidal closure

- Additive connectives are important for trees
- Links between planar GoI, two-way transducers and first-order fragments

Conclusion

What happened here:

- Connections between Church encodings and automata
- Application of categorical semantics (Dialectica, geometry of interaction (GoI))
- A generic uniformization-like construction $(\mathcal{C}_{\&})_{\oplus} \to \mathcal{C}_{\oplus}$ for monoidal $\mathcal{C}$ with certain homsets

Some take-aways:

- Important ingredient in uniformization: monoidal closure
  - Slick formalization w/o categories (using e.g. transition monoids)?
- Additive connectives are important for trees
- Links between planar GoI, two-way transducers and first-order fragments

Conclusion

What happened here:

- Connections between Church encodings and automata
- Application of categorical semantics (Dialectica, geometry of interaction (GoI))
- A generic uniformization-like construction $(\mathcal{C}_{\&})_{\oplus} \to \mathcal{C}_{\oplus}$ for monoidal $\mathcal{C}$ with certain homsets

Some take-aways:

- Important ingredient in uniformization: monoidal closure
  - Slick formalization w/o categories (using e.g. transition monoids)?
- Additive connectives are important for trees
- Links between planar GoI, two-way transducers and first-order fragments
  - Further links with tree-walking automata?

Conclusion

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)	weird (?)	✓ (?)
affine	regular functions	✓
non-commutative affine	first-order regular fn.	✓?
linear/affine with additives	regular functions	✓
parsimonious	polyregular	??
simply typed	variant of CPDA???	???

Conclusion

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)	weird (?)	✓ (?)
affine	regular functions	✓
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

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)	weird (?)	✓(?)
affine	regular functions	✓
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! Questions?

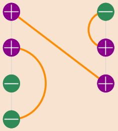
A category of planar diagrams

- Interpret purely linear non-commutative λ-terms in a monoidal closed category
- We consider a non-commutative refinement of Geometry of Interaction

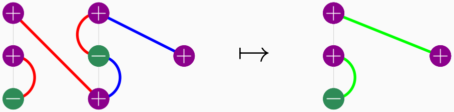
(well-known model of linear logic)

A compact closed category of planar diagrams

- **Objects:** words in $\{+,-\}^*$
- **Morphisms** $u \to v$ **:** graphs over $|u| + |v|$ with
  - degree ≤ 1 for every node
  - polarity restrictions
  - planarity restriction



To compute the composition of two morphisms, follow the paths (and forget the middle component)

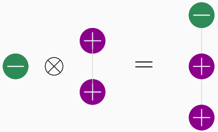


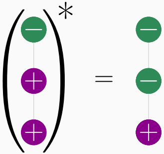
Compact-closure and interpretation of the λ-calculus

Structure to interpret the linear λ-calculus

- Monoidal product $A \otimes B$ given by concatenation
- Duals A^*: reverse and flip polarities
- Monoidal closure by setting $A \multimap B \; = \; A^* \otimes B$
- Interpretation of types $[\![A]\!]$ by induction with $[\![o]\!] = +$

(injective interpretation of booleans)

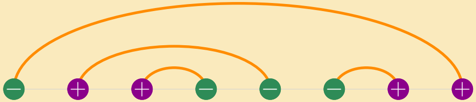




Examples

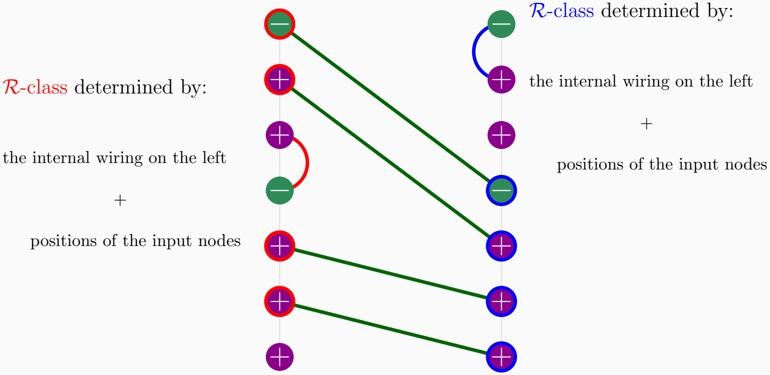
$$[\![((o \multimap o) \multimap o \multimap o) \multimap ((o \multimap o) \multimap o) \multimap o]\!] \; = \; - + + - - - + +$$

$$[\![\lambda f.\lambda g.\, f\,(\lambda x.\, x)\,(g\,(\lambda x.\, x))]\!] \; =$$



Aperiodicity

To conclude, we need to show that every $(\mathsf{Hom}(A, A), \circ)$ is finite and aperiodic for every A

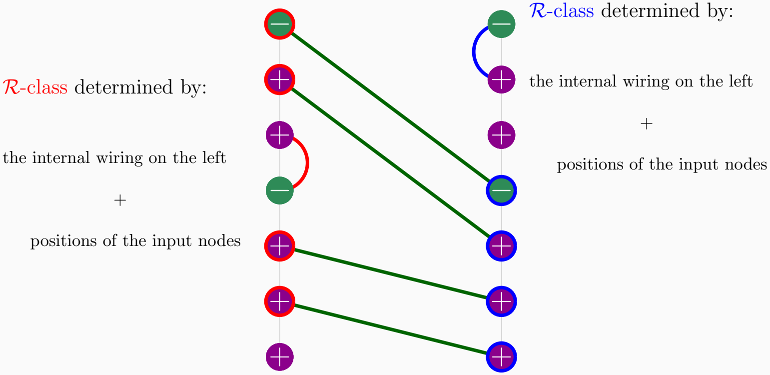


Therefore: planar $\Longrightarrow$ $\mathcal{H}$-trivial

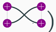
- More elementary proofs w/o Green relations possible (e.g. order+Kleene's theorem)

Aperiodicity

To conclude, we need to show that every $(\mathsf{Hom}(A, A), \circ)$ is finite and aperiodic for every A

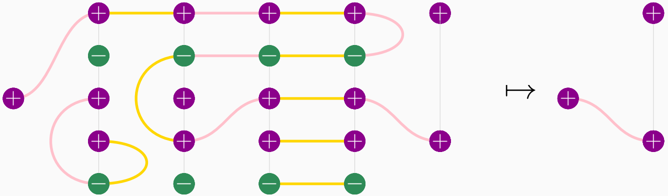


Therefore: planar $\Longrightarrow$ $\mathcal{H}$-trivial

- More elementary proofs w/o Green relations possible (e.g. order+Kleene's theorem)
- Planarity restriction is essential (consider )

Diagrams and two-way automata

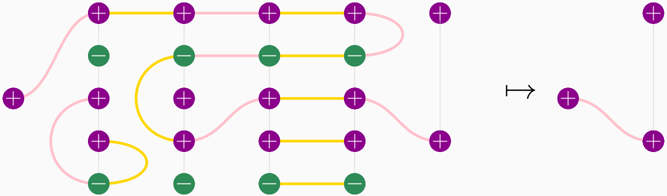
Non-planar diagrams (with crossings): reminiscent of runs in 2DFAs!



- Transition functions $\delta : \Sigma \to \mathsf{Hom}(Q, Q)$ for some object Q $\qquad$ $Q \approx$ set of directed states
- (actually, should also incorporate boundary morphisms $\mathsf{Hom}(+, Q)$ and $\mathsf{Hom}(Q, F)$)

Diagrams and two-way automata

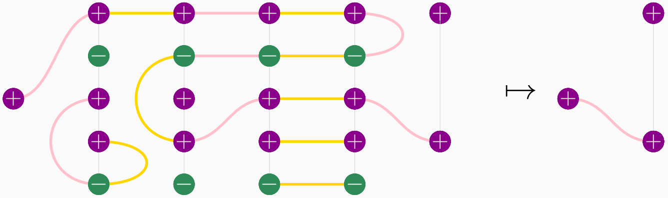
Non-planar diagrams (with crossings): reminiscent of runs in 2DFAs!



- Transition functions $\delta : \Sigma \to \mathsf{Hom}(Q, Q)$ for some object Q — $Q \approx$ set of directed states
- (actually, should also incorporate boundary morphisms $\mathsf{Hom}(+, Q)$ and $\mathsf{Hom}(Q, F)$)
- Planarity restriction $\Rightarrow$ the transition flow monoid is aperiodic
- (links between GoI and planar 2DFAs already considered by (Hines 2003))

Diagrams and two-way automata

Non-planar diagrams (with crossings): reminiscent of runs in 2DFAs!



- Transition functions $\delta : \Sigma \to \mathsf{Hom}(Q, Q)$ for some object Q — $Q \approx$ set of directed states
- (actually, should also incorporate boundary morphisms $\mathsf{Hom}(+, Q)$ and $\mathsf{Hom}(Q, F)$)
- Planarity restriction $\Rightarrow$ the transition flow monoid is aperiodic
- (links between GoI and planar 2DFAs already considered by (Hines 2003))

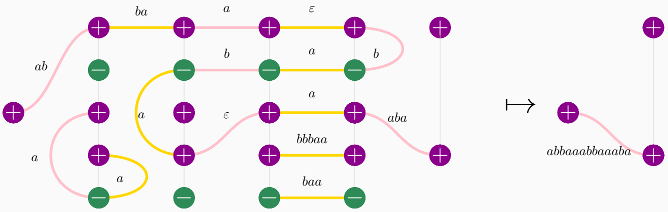
Theorem

Star-free languages are exactly those recognized by planar 2DFAs.

More generally: first-order transductions

Consider a richer category of diagrams where edges are labelled by output words

(labels of compositions given by concatenation)



Much like before, corresponding notion of (planar) 2DFTs.

Theorem

First-order transduction (FO regular functions) are those computed by reversible planar 2DFTs.

- 2DFTs with aperiodic transition monoid = FO regular functions [Carton&Dartois 2015]

  (hence reversible planar 2DFTs ⊆ FO-transductions)

- FO transduction ⊆ reversible planar 2DFTs: closure by composition and Krohn–Rhodes

  (see `http://nguyentito.eu/2021-01-links.pdf`)