Implicit automata in typed λ-calculi

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

Irif, December 16th, 2020

The idea

Suppose the programs of type T in a programming language $\mathcal{P}$ all compute *languages*, something like $T = \texttt{String} \to \texttt{Bool}$. (Or *functions* $\texttt{String} \to \texttt{String}$.)

What class of languages? Depends on $\mathcal{P}$ and T.
Many theoretical PLs *not* Turing-complete, especially *typed λ-calculi*

The idea

Suppose the programs of type T in a programming language $\mathcal{P}$
all compute *languages*, something like $T = \texttt{String} \to \texttt{Bool}$.
(Or *functions* $\texttt{String} \to \texttt{String}$.)

What class of languages? Depends on $\mathcal{P}$ and T.
Many theoretical PLs *not* Turing-complete, especially *typed λ-calculi*
Implicit complexity: machine-free characterizations of complexity classes
using high-level programming languages
Big project: the same thing for automata instead of complexity

The idea

Suppose the programs of type T in a programming language $\mathcal{P}$
all compute *languages*, something like $T = \texttt{String} \to \texttt{Bool}$.
(Or *functions* $\texttt{String} \to \texttt{String}$.)

What class of languages? Depends on $\mathcal{P}$ and T.
Many theoretical PLs *not* Turing-complete, especially *typed λ-calculi*
Implicit complexity: machine-free characterizations of complexity classes
using high-level programming languages
Big project: the same thing for automata instead of complexity

Plan

1. Alternative justification:
   internal motivations from *simply typed λ-calculus*
2. A concrete result: *regular string-to-string functions* in an *affine* λ-calculus
   (+ brief mention of *star-free languages* vs *non-commutative types*)
3. Some abstract nonsense on monoidal closed categories

Simply typed functions on Church numerals (1)

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`).

Simply typed functions on Church numerals (1)

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 numerals (2)

Take $\mathsf{mult} = \lambda n.\lambda m.\lambda f.\, n\,(m\,f) : \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat}$.
$\mathsf{mult}\ \overline{2} : \mathsf{Nat} \to \mathsf{Nat}$ can be iterated by a $\mathsf{Nat}[\mathsf{Nat}]$...

$$\longrightarrow \mathsf{exp2} = \lambda n.\, n\,(\mathsf{mult}\ \overline{2})\ \overline{1} : \mathsf{Nat}[\mathsf{Nat}] \to \mathsf{Nat}$$

which cannot be iterated!

Simply typed functions on Church numerals (2)

Take $\mathsf{mult} = \lambda n.\lambda m.\lambda f.\, n\,(m\,f) : \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat}$.
$\mathsf{mult}\ \overline{2} : \mathsf{Nat} \to \mathsf{Nat}$ can be iterated by a $\mathsf{Nat}[\mathsf{Nat}]$...

$$\longrightarrow \mathsf{exp2} = \lambda n.\, n\,(\mathsf{mult}\ \overline{2})\ \overline{1} : \mathsf{Nat}[\mathsf{Nat}] \to \mathsf{Nat}$$

which cannot be iterated! Smaller types, still heterogenous:

$$\mathsf{exp2} = \lambda n.\, n\ \overline{2} : \mathsf{Nat}[o \to o] \to \mathsf{Nat}$$

Simply typed functions on Church numerals (2)

Take $\mathsf{mult} = \lambda n.\lambda m.\lambda f.\, n\,(m\,f) : \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat}$.
$\mathsf{mult}\;\overline{2} : \mathsf{Nat} \to \mathsf{Nat}$ can be iterated by a $\mathsf{Nat}[\mathsf{Nat}]$...

$$\longrightarrow \mathsf{exp2} = \lambda n.\, n\,(\mathsf{mult}\;\overline{2})\;\overline{1} : \mathsf{Nat}[\mathsf{Nat}] \to \mathsf{Nat}$$

which cannot be iterated! Smaller types, still heterogenous:

$$\mathsf{exp2} = \lambda n.\, n\;\overline{2} : \mathsf{Nat}[o \to o] \to \mathsf{Nat}$$

Towers of exponentials of *any fixed height* $\mathsf{Nat}[T[A]] \to \mathsf{Nat}[A]$.
This is the fastest possible growth for simply typed λ-terms.

Simply typed functions on Church numerals (2)

Take $\mathsf{mult} = \lambda n.\lambda m.\lambda f.\, n\,(m\,f) : \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat}$.
$\mathsf{mult}\ \overline{2} : \mathsf{Nat} \to \mathsf{Nat}$ can be iterated by a $\mathsf{Nat}[\mathsf{Nat}]$...

$$\longrightarrow \mathsf{exp2} = \lambda n.\, n\,(\mathsf{mult}\ \overline{2})\ \overline{1} : \mathsf{Nat}[\mathsf{Nat}] \to \mathsf{Nat}$$

which cannot be iterated! Smaller types, still heterogenous:

$$\mathsf{exp2} = \lambda n.\, n\ \overline{2} : \mathsf{Nat}[o \to o] \to \mathsf{Nat}$$

Towers of exponentials of *any fixed height* $\mathsf{Nat}[T[A]] \to \mathsf{Nat}[A]$.
This is the fastest possible growth for simply typed λ-terms.

On the other hand:

Theorem (Statman 198X)

Subtraction cannot be defined by any simply typed $t : \mathsf{Nat}[A] \to \mathsf{Nat}[B] \to \mathsf{Nat}$.

Simply typed functions on Church numerals (3)

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?

Subtraction cannot be defined; some "easy" 1-variable functions, e.g. $n \mapsto \lfloor \sqrt{n} \rfloor$, are also undefinable.

Does this even admit a *satisfying* answer?

Simply typed functions on Church numerals (3)

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?

Subtraction cannot be defined; some “easy” 1-variable functions, e.g. $n \mapsto \lfloor \sqrt{n} \rfloor$, are also undefinable.

Does this even admit a *satisfying* answer? There is one for *predicates*!

Theorem (Joly 2001)

A subset of $\mathbb{N}^k$ *is decidable by some* $t : \mathsf{Nat}[A_1] \to \cdots \to \mathsf{Nat}[A_n] \to \mathsf{Bool}$ (*where* $\mathsf{Bool} = o \to o \to o$) *if and only if it is ultimately periodic.*

Simply typed functions on Church numerals (3)

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?

Subtraction cannot be defined; some "easy" 1-variable functions, e.g. $n \mapsto \lfloor \sqrt{n} \rfloor$, are also undefinable.

Does this even admit a *satisfying* answer? There is one for *predicates*!

Theorem (Joly 2001)

A subset of $\mathbb{N}^k$ is decidable by some $t : \mathsf{Nat}[A_1] \to \cdots \to \mathsf{Nat}[A_n] \to \mathsf{Bool}$ (where $\mathsf{Bool} = o \to o \to o$) if and only if it is ultimately periodic.

Corollary

If $t : \mathsf{Nat}[A] \to \mathsf{Nat}$ defines $f_t : \mathbb{N} \to \mathbb{N}$, then
$X \subseteq \mathbb{N}$ ultimately periodic $\implies f_t^{-1}(X)$ ultimately periodic.

A not quite trivial necessary condition!

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

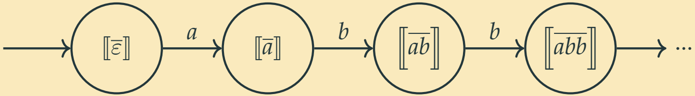
Theorem (Hillebrand & Kanellakis, LICS'96)

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 $Q = [\![\mathsf{Str}_\Sigma[A]]\!]$ ($\mathrm{Card}(Q)$ depends on A), acceptation as $[\![t]\!](-) = [\![\mathtt{true}]\!]$.



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

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

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

Towards implicit automata

Recap: in the simply typed λ-calculus,
$\mathsf{Str}_\Gamma[A] \to \mathsf{Bool}$ = regular languages; $\mathsf{Str}_\Gamma[A] \to \mathsf{Str}_\Sigma$ = ???

So what's next?

Towards implicit automata

Recap: in the simply typed λ-calculus,
$\mathsf{Str}_\Gamma[A] \to \mathsf{Bool}$ = regular languages; $\mathsf{Str}_\Gamma[A] \to \mathsf{Str}_\Sigma$ = ???

So what's next? Use restricted types to:

- get an easier problem for string-to-string functions

Towards implicit automata

Recap: in the simply typed λ-calculus,
$\mathsf{Str}_\Gamma[A] \to \mathsf{Bool}$ = regular languages; $\mathsf{Str}_\Gamma[A] \to \mathsf{Str}_\Sigma$ = ???

So what's next? Use restricted types to:

- get an easier problem for string-to-string functions

- characterize smaller classes of languages
  $\longrightarrow$ *star-free languages* (regular expressions without repetition star, but with complementation)
  using *non-commutative* types (functions must use their arguments in the order that they are given in)
  (not covered here)

Towards implicit automata

Recap: in the simply typed λ-calculus,
$\mathsf{Str}_\Gamma[A] \to \mathsf{Bool}$ = regular languages; $\mathsf{Str}_\Gamma[A] \to \mathsf{Str}_\Sigma$ = ???

So what's next? Use restricted types to:

- get an easier problem for string-to-string functions
  $\longrightarrow$ **regular functions in an affine λ-calculus**
- characterize smaller classes of languages
  $\longrightarrow$ *star-free languages* (regular expressions without repetition star, but with complementation)
  using *non-commutative* types (functions must use their arguments in the order that they are given in)
  (not covered here)

A λ-calculus with affine types

Typing judgments: {non-affine variables} | {affine variables} $\vdash t : A$

- $\lambda^{\to} x.\, t : A \to B$ unrestricted function
- $\lambda^{\circ} x.\, t : A \multimap B$ affine function (at most one x in t)

$$\frac{}{\Gamma, x : A \mid \varnothing \vdash x : A} \qquad \frac{}{\Gamma \mid x : A \vdash x : A}$$

$$\frac{\Gamma \mid \Delta \vdash t : A \to B \qquad \Gamma \mid \varnothing \vdash u : A}{\Gamma \mid \Delta \vdash t\, u : B} \qquad \frac{\Gamma, x : A \mid \Delta \vdash t : B}{\Gamma \mid \Delta \vdash \lambda^{\to} x.\, t : A \to B}$$

$$\frac{\Gamma \mid \Delta \vdash t : A \multimap B \qquad \Gamma \mid \Delta' \vdash u : A}{\Gamma \mid \Delta, \Delta' \vdash t\, u : B} \qquad \frac{\Gamma \mid \Delta, x : A \vdash t : B}{\Gamma \mid \Delta \vdash \lambda^{\circ} x.\, t : A \multimap B}$$

The above = Dual Intuitionistic Linear Logic

$$\frac{\Gamma \mid \Delta \vdash t : A}{\Gamma \mid \Delta' \vdash t : A} \text{ when } \Delta \subseteq \Delta' \text{: \textit{weakening} rule}$$

Regular functions

Church encoding with linear/affine 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

Church encoding with linear/affine 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 our affine λ-calculus with A *purely affine*, i.e. containing no `$\to$'

Regular functions

Church encoding with linear/affine 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 our affine λ-calculus with A *purely affine*, i.e. containing no '$\to$'

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

beware: sequential functions $\neq$ rational functions $\neq$ regular functions

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

Single-state streaming string transducers

Definition

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

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. $a \mapsto \begin{cases} X := Xa \\ Y := aY \end{cases} \quad b \mapsto \begin{cases} X := Xb \\ Y := bY \end{cases}$
- "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. $X_{\mathrm{init}} = Y_{\mathrm{init}} = \varepsilon$
- *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. $\mathrm{out} = XY$

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. $a \mapsto \begin{cases} X := Xa \\ Y := aY \end{cases} \quad b \mapsto \begin{cases} X := Xb \\ Y := bY \end{cases}$
- "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. $X_{\text{init}} = Y_{\text{init}} = \varepsilon$
- *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. $\texttt{out} = XY$

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

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

Single-state streaming string transducers

Definition

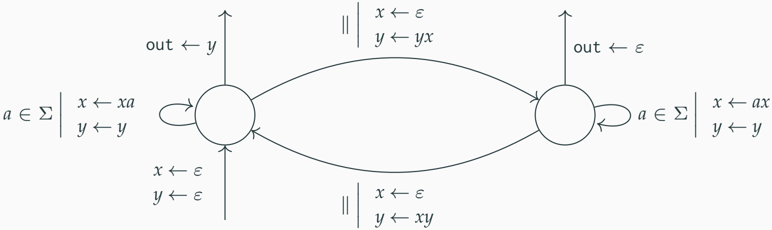
- Finite set of Σ^*-valued *registers* e.g. $R = \{X, Y\}$
- Initial values $R \to \Sigma^*$ e.g. $X_{\text{init}} = Y_{\text{init}} = \varepsilon$
- *Register update function* e.g. $a \mapsto \begin{cases} X := Xa \\ Y := aY \end{cases}$ $b \mapsto \begin{cases} X := Xb \\ Y := bY \end{cases}$
- "output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

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

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

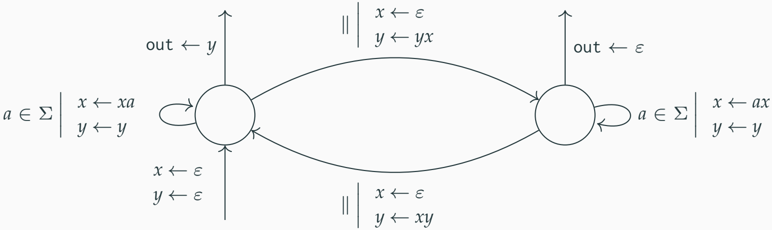
Stateful streaming string transducers

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 :=

(for each fixed input letter, at most once among all the associated :=)

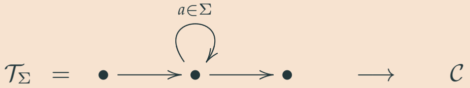
Intuition: memory $M = Q \otimes \Sigma^* \otimes \ldots \otimes \Sigma^*$, transitions $M \multimap M$

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

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

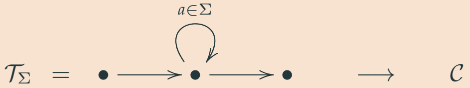


- DFA = automata over the category of finite sets
- Copyless SSTs $\approx$ start from a category $\mathcal{R}$ of copyless register updates
  + add states by *free finite coproduct completion* $(-)_\oplus$

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



- 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* $(-)_\oplus$

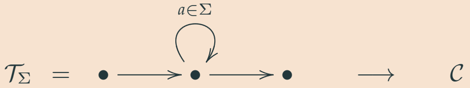
Definition of the free finite coproduct completion $\mathcal{C}_\oplus$

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

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



- 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* $(-)_\oplus$

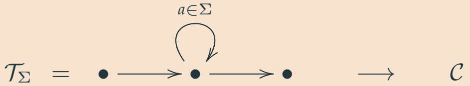
Definition of the free finite coproduct completion $\mathcal{C}_\oplus$

- **Objects:** formal finite sums $\bigoplus_{u \in U} C_u$ of objects of $\mathcal{C}$
  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 $\mathcal{T}_\Sigma \to \mathcal{C}$)



- 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 affine λ-calculus can be turned into automata over a category $\mathcal{L}$ of purely affine 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 affine λ-calculus can be turned into automata over a category $\mathcal{L}$ of purely affine 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 affine λ-calculus can be turned into automata over a category $\mathcal{L}$ of purely affine 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 affinely 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 *affine 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 affine 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.

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

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.

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

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.

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

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

Conclusion

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

Conservativity

Lemma

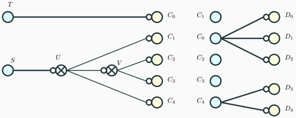
$(\mathcal{C}_{\&})_{\oplus}$ automata are equivalent to non-deterministic $\mathcal{C}_{\oplus}$ automata.

A determinization theorem is enough to conclude

Conservativity

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

- Determinization 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., $\mathcal{C}_{\oplus}$-automata can be determinized

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 affine ILL with A purely affine

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 affine ILL with A purely affine

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 affine ILL with A purely affine

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 affine ILL with A purely affine

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 affine ILL with A purely affine

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

Additive connectives: why (not)?

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 (not)?

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!

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

Additive connectives: why (not)?

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”.

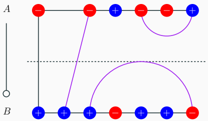
Principled explanation via linear logic:
just allow the *additive conjunction* in the internal memory!

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

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

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 affine (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 affine (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 affine (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!