Transductions in affine logic

Nguyễn Lê Thành Dũng (a.k.a. Tito) — nltd@nguyentito.eu
LIPN, Université Paris 13

Trends in Linear Logic and Applications, Dortmund, June 30th, 2019

Transductions in typed λ-calculi

1. **Motivation: the simply-typed λ-calculus functions on Church encodings and automata theory**
   - Claim:
     λ-definable string functions are related to *transducers*,
     i.e. automata with output (many possible variants).
2. Capturing a class of transductions with *affine typing*,
   via linearity in *streaming string transducers*

Details on (2) in my 5-page abstract.
I've come to think that (1) is more important.

Transductions in typed λ-calculi

1. **Motivation: the simply-typed λ-calculus functions on Church encodings and automata theory**
   - Claim:
     λ-definable string functions are related to *transducers*,
     i.e. automata with output (many possible variants).
2. Capturing a class of transductions with *affine typing*,
   via linearity in *streaming string transducers*

Details on (2) in my 5-page abstract.
I've come to think that (1) is more important.

But (2) came first, triggered by a question by M. Bojańczyk.
$\longrightarrow$ Counterexample to "provinciality" of linear logic.

Simply typed functions on Church numerals (1)

Church encodings of (unary) natural numbers:

- $\mathsf{Nat} = o \to (o \to o) \to o$
- $n \in \mathbb{N} \rightsquigarrow \overline{n} = \lambda x.\, \lambda f.\, f\,(\ldots\ (f\,x)\ldots) : \mathsf{Nat}$ with n times f

For $t : \mathsf{Nat} \to \mathsf{Nat}$, $\mathcal{F}(t)(n) = m \iff t\,\overline{n} =_\beta \overline{m}$.

Theorem (Schwichtenberg 1975)

The functions definable by simply-typed λ-terms of type $\mathsf{Nat} \to \mathsf{Nat}$ *are the* extended polynomials (*generated by 0,1,+,×,*`ifzero`).

Simply typed functions on Church numerals (1)

Church encodings of (unary) natural numbers:

- $\mathsf{Nat} = o \to (o \to o) \to o$
- $n \in \mathbb{N} \rightsquigarrow \overline{n} = \lambda x.\, \lambda f.\, f\,(\ldots\ (f\,x)\ldots) : \mathsf{Nat}$ with n times f

For $t : \mathsf{Nat} \to \mathsf{Nat}$, $\mathcal{F}(t)(n) = m \iff t\,\overline{n} =_\beta \overline{m}$.

Theorem (Schwichtenberg 1975)

The functions definable by simply-typed λ-terms of type $\mathsf{Nat} \to \mathsf{Nat}$ *are the* extended polynomials (*generated by 0,1,+,×,*`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

$\{\mathcal{F}(t) \mid A \text{ simple type},\ t : \mathsf{Nat}[A] \to \mathsf{Nat}\} = ?$

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!

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 (3)

Towers of exponentials of any fixed height $\mathsf{Nat}[T[A]] \to \mathsf{Nat}[A]$.
On the other hand:

Theorem (Statman 198?)

Subtraction *cannot be defined as a simply typed λ-term of type* $\mathsf{Nat}[A] \to \mathsf{Nat}[B] \to \mathsf{Nat}$.

More weirdness: some "easy" 1-variable functions,
e.g. $n \mapsto \lfloor \sqrt{n} \rfloor$, are also undefinable.

Simply typed functions on Church numerals (3)

Towers of exponentials of any fixed height $\mathsf{Nat}[T[A]] \to \mathsf{Nat}[A]$.
On the other hand:

Theorem (Statman 198?)

Subtraction *cannot be defined as a simply typed λ-term of type* $\mathsf{Nat}[A] \to \mathsf{Nat}[B] \to \mathsf{Nat}$.

More weirdness: some "easy" 1-variable functions,
e.g. $n \mapsto \lfloor \sqrt{n} \rfloor$, are also undefinable.

$\longrightarrow$ Does this really correspond to any interesting class, or is our open problem a "typical misguided natural question"?
(cf. Gromov, *Spaces and questions*)
Might explain lack of progress on this question...

A first interesting result: predicates on Church integers

My POV: the question is actually interesting.
Illustration on restricted case: *predicates* ($\mathsf{Bool} = o \to o \to o$).

Theorem (Joly 2001)

A subset of $\mathbb{N}$ *is decidable by some* $t : \mathsf{Nat}[A] \to \mathsf{Bool}$ *if and only if it is* ultimately periodic.

A first interesting result: predicates on Church integers

My POV: the question is actually interesting.
Illustration on restricted case: *predicates* ($\mathsf{Bool} = o \to o \to o$).

Theorem (Joly 2001)

A subset of $\mathbb{N}$ *is decidable by some* $t : \mathsf{Nat}[A] \to \mathsf{Bool}$ *if and only if it is* ultimately periodic.

Corollary

For all simple types A *and all* $t : \mathsf{Nat}[A] \to \mathsf{Nat}$,
$X \subseteq \mathbb{N}$ *ultimately periodic* $\implies$ $\mathcal{F}(t)^{-1}(X)$ *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 $\Sigma = \{a_1, \ldots, a_{|\Sigma|}\}$:

- $\mathsf{Str}_\Sigma = o \to (o \to o) \to \ldots |\Sigma| \text{ times} \ldots \to (o \to o) \to o$
- $w \in \Sigma^* \rightsquigarrow \overline{w} = \lambda x.\ \lambda f_1.\ \ldots\ \lambda f_{|\Sigma|}.\ f_{i_1}\ (\ldots\ (f_{i_n}\ x)\ldots) : \mathsf{Str}_\Sigma$ where $w = a_{i_1} \ldots a_{i_n}$

Characterizations of $\{\mathcal{F}(t) \mid t : \mathsf{Str}_\Gamma \to \mathsf{Str}_\Sigma\}$ (no subst.) exist[1].

Open question

$\{\mathcal{F}(t) \mid A \text{ simple type},\ t : \mathsf{Str}_\Gamma[A] \to \mathsf{Str}_\Sigma\} = ?$

[1] e.g. Zaionc 1987, *Word operation definable in the typed λ-calculus*

Simply typed predicates on Church-encoded strings

Theorem (Hillebrand & Kanellakis 1995)

A subset of Σ^* *is decidable by some* $t : \mathsf{Str}_\Sigma[A] \to \mathsf{Bool}$ *iff it is a* regular language.

(Note: $X \subseteq \{1\}^*$ regular $\iff$ $X \subseteq \mathbb{N}$ ultimately periodic)

Corollary

$L \subseteq \Sigma^*$ *regular* $\implies$ $\mathcal{F}(t)^{-1}(L)$ *regular for* $t : \mathsf{Str}_\Gamma[A] \to \mathsf{Str}_\Sigma$.

$\longrightarrow$ look for an *automata-theoretic* function class with regularity preservation + hyperexponential growth

Register transducers

For input alphabet Γ, output alphabet Σ:

- 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* $u : \Gamma \to (R \to (\Sigma \cup R)^*)$ e.g.

$$a \mapsto \begin{cases} X \mapsto Xa \\ Y \mapsto aY \end{cases} \qquad b \mapsto \begin{cases} X \mapsto Xb \\ Y \mapsto bY \end{cases}$$

- "output function" $\in (\Sigma \cup R)^*$ e.g. out $= XY$

Register transducers

For input alphabet Γ, output alphabet Σ:

- 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* $u : \Gamma \to (R \to (\Sigma \cup R)^*)$ e.g.

$$a \mapsto \begin{cases} X := Xa \\ Y := aY \end{cases} \qquad b \mapsto \begin{cases} X := Xb \\ Y := bY \end{cases}$$

- "output function" $\in (\Sigma \cup R)^*$ e.g. out $= XY$

Register transducers

For input alphabet Γ, output alphabet Σ:

- 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* $u : \Gamma \to (R \to (\Sigma \cup R)^*)$ e.g.

$$a \mapsto \begin{cases} X := Xa \\ Y := aY \end{cases} \qquad b \mapsto \begin{cases} X := Xb \\ Y := bY \end{cases}$$

- "output function" $\in (\Sigma \cup R)^*$ e.g. out $= XY$

Execution over *abaa*: start with

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

Register transducers

For input alphabet Γ, output alphabet Σ:

- 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* $u : \Gamma \to (R \to (\Sigma \cup R)^*)$ e.g.

$$a \mapsto \begin{cases} X := Xa \\ Y := aY \end{cases} \qquad b \mapsto \begin{cases} X := Xb \\ Y := bY \end{cases}$$

- "output function" $\in (\Sigma \cup R)^*$ e.g. out $= XY$

Execution over $\underline{a}baa$:

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

Register transducers

For input alphabet Γ, output alphabet Σ:

- 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* $u : \Gamma \to (R \to (\Sigma \cup R)^*)$ e.g.

$$a \mapsto \begin{cases} X := Xa \\ Y := aY \end{cases} \qquad b \mapsto \begin{cases} X := Xb \\ Y := bY \end{cases}$$

- "output function" $\in (\Sigma \cup R)^*$ e.g. out $= XY$

Execution over $a\underline{b}aa$:

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

Register transducers

For input alphabet Γ, output alphabet Σ:

- 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* $u : \Gamma \to (R \to (\Sigma \cup R)^*)$ e.g.

$$a \mapsto \begin{cases} X := Xa \\ Y := aY \end{cases} \qquad b \mapsto \begin{cases} X := Xb \\ Y := bY \end{cases}$$

- "output function" $\in (\Sigma \cup R)^*$ e.g. out $= XY$

Execution over $ab\underline{a}a$:

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

Register transducers

For input alphabet Γ, output alphabet Σ:

- 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* $u : \Gamma \to (R \to (\Sigma \cup R)^*)$ e.g.

$$a \mapsto \begin{cases} X := Xa \\ Y := aY \end{cases} \qquad b \mapsto \begin{cases} X := Xb \\ Y := bY \end{cases}$$

- "output function" $\in (\Sigma \cup R)^*$ e.g. out $= XY$

Execution over $aba\underline{a}$:

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

Register transducers

For input alphabet Γ, output alphabet Σ:

- 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* $u : \Gamma \to (R \to (\Sigma \cup R)^*)$ e.g.

$$a \mapsto \begin{cases} X := Xa \\ Y := aY \end{cases} \qquad b \mapsto \begin{cases} X := Xb \\ Y := bY \end{cases}$$

- "output function" $\in (\Sigma \cup R)^*$ e.g. out $= XY$

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

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

Register transducers

For input alphabet Γ, output alphabet Σ:

- 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* $u : \Gamma \to (R \to (\Sigma \cup R)^*)$ e.g.

$$a \mapsto \begin{cases} X := Xa \\ Y := aY \end{cases} \qquad b \mapsto \begin{cases} X := Xb \\ Y := bY \end{cases}$$

- "output function" $\in (\Sigma \cup R)^*$ e.g. out $= XY$

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

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

Register transducers with states

- Finite set of Σ^*-valued *registers* R, initial values $R \to \Sigma^*$
- *finite state machine* (*i.e. DFA*) $(Q, q_{\text{init}} \in Q, \delta : \Gamma \times Q \to Q)$
- updates $\Gamma \times Q \to (\Sigma \cup R)^*$, "output function" $Q \to (\Sigma \cup R)^*$

Register transducers with states

- Finite set of Σ^*-valued *registers* R, initial values $R \to \Sigma^*$
- *finite state machine* (*i.e. DFA*) $(Q, q_{\text{init}} \in Q, \delta : \Gamma \times Q \to Q)$
- updates $\Gamma \times Q \to (\Sigma \cup R)^*$, "output function" $Q \to (\Sigma \cup R)^*$

Theorem (Filiot & Reynier 2017)

Let $f : \Gamma^* \to \Sigma^*$. The following are equivalent:

- *f can be computed by a register transducer*
- *f can be computed by a register transducer with states*
- *f can be specified by a* HDT0L system
  (*variant of* Lindenmayer systems *from the 1960s*)

Furthermore, L regular $\implies$ $f^{-1}(L)$ regular for such f.

Exponential growth of HDT0L transductions

$$\Gamma = \Sigma = \{1\} \quad R = \{X\} \qquad X_{\text{init}} = 1 \quad X := XX$$

$$\Longrightarrow f : \Gamma^* \to \Sigma^* \text{ such that } |f(w)| = 2^{|w|}$$

Exponential growth of HDT0L transductions

$$\Gamma = \Sigma = \{1\} \quad R = \{X\} \qquad X_{\text{init}} = 1 \quad X := XX$$
$$\Longrightarrow f : \Gamma^* \to \Sigma^* \text{ such that } |f(w)| = 2^{|w|}$$

Proposition (Exponential upper bound)

Conversely, for any HDT0L $f : \Gamma^* \to \Sigma^*$, $|f(w)| = 2^{O(|w|)}$.

Corollary

HDT0L transductions are not closed under composition.

Their closure under comp. has elementary *growth.*

Exponential growth of HDT0L transductions

$$\Gamma = \Sigma = \{1\} \quad R = \{X\} \qquad X_{\text{init}} = 1 \quad X := XX$$
$$\Longrightarrow f : \Gamma^* \to \Sigma^* \text{ such that } |f(w)| = 2^{|w|}$$

Proposition (Exponential upper bound)

Conversely, for any HDT0L $f : \Gamma^* \to \Sigma^*$, $|f(w)| = 2^{O(|w|)}$.

Corollary

HDT0L transductions are not closed under composition.
Their closure under comp. has elementary *growth.*

HDT0L+composition $\supset$ all transduction classes that I know
(to be fair, I don't know many)

See also: Sénizergues 2007, *Sequences of level 1,2,...,k,...*

Returning to λ-definable functions

Theorem

The class of functions defined by simply typed λ-terms of type $\mathsf{Str}_\Gamma[A] \to \mathsf{Str}_\Sigma$ contains HDT0L+composition.

Proof.

It is closed under composition and can encode register transducers (programming exercise). $\square$

Returning to λ-definable functions

Theorem

The class of functions defined by simply typed λ-terms of type $\mathsf{Str}_\Gamma[A] \to \mathsf{Str}_\Sigma$ contains HDT0L+composition.

Proof.

It is closed under composition and can encode register transducers (programming exercise). □

STλ subsumes many classes of regularity-preserving functions!

Open question (concrete proposal)

What about the converse?

That is, STλ-definable = HDT0L+composition?

Possible connections with *iterated pushdown automata*.

Linearity in register transducers

$X := XX$ is a *non-linear* register update
$\longrightarrow$ as usual, duplication causes complexity ($2^{O(n)}$ growth)

Whereas *affine* updates lead to $O(n)$ growth,
e.g. $|f(w)| = 2|w|$ for $f : w \mapsto w \cdot \texttt{reverse}(w)$

$$\begin{cases} X := Xa \\ Y := aY \end{cases} \quad \leftrightsquigarrow \quad x \otimes y \mapsto xa \otimes ay$$

(In the automata theory literature: “copyless assignments”.)

Linearity in register transducers

$X := XX$ is a *non-linear* register update
$\longrightarrow$ as usual, duplication causes complexity ($2^{O(n)}$ growth)

Whereas *affine* updates lead to $O(n)$ growth,
e.g. $|f(w)| = 2|w|$ for $f : w \mapsto w \cdot \texttt{reverse}(w)$

$$\begin{cases} X := Xa \\ Y := aY \end{cases} \quad \leftrightsquigarrow \quad x \otimes y \mapsto xa \otimes ay$$

(In the automata theory literature: “copyless assignments”.)

Definition

A *streaming string transducer* (*SST*) is a register transducer with states and with *affine register updates*.

Regular functions

Definition

A *streaming string transducer* (*SST*) is a register transducer with states and with *affine register updates*.

Theorem (Alur & Černý 2010)

The functions computed by SSTs are the regular functions.

Closed under composition, linear growth, f^{-1}(reg. lang) is reg.
Many other characterizations:

- *two-way* finite state transducers without registers
- Monadic Second-Order Logic
- basic functions + combinators (at least 3 possible variants)
- etc.

Inspiration from implicit complexity in linear logic

Goal: regular functions in an affine programming language.

We use the *elementary affine λ-calculus* (EAλ).

Theorem (Baillot, De Benedetti & Ronchi Della Rocca 2014)

The languages defined by EAλ*-terms* with type fixpoints *of type* $!\mathsf{Str}_\Sigma \multimap !!\mathsf{Bool}$ *are exactly those decidable in* polynomial time.

Where $\mathsf{Str}_\Sigma = \forall\alpha.\ (!(\alpha \multimap \alpha))^{|\Sigma|} \multimap !(\alpha \multimap \alpha)$

Inspiration from implicit complexity in linear logic

Goal: regular functions in an affine programming language.

We use the *elementary affine λ-calculus* (EAλ).

Theorem (Baillot, De Benedetti & Ronchi Della Rocca 2014)

The languages defined by EAλ*-terms* with type fixpoints *of type* $!\mathsf{Str}_\Sigma \multimap !!\mathsf{Bool}$ *are exactly those decidable in* polynomial time.

Where $\mathsf{Str}_\Sigma = \forall\alpha.\ (!(\alpha \multimap \alpha))^{|\Sigma|} \multimap !(\alpha \multimap \alpha)$

Theorem (my talk at TLLA 2018!)

The languages defined by EAλ*-terms of type* $!\mathsf{Str}_\Sigma \multimap !!\mathsf{Bool}$ *are exactly the* regular languages *over* Σ.

Good news: $!\mathsf{Str}_\Gamma \multimap !\mathsf{Str}_\Sigma \rightsquigarrow$ regularity-preserving functions.

A characterization of regular functions

Good news: $!\mathsf{Str}_\Gamma \multimap !\mathsf{Str}_\Sigma \rightsquigarrow$ regularity-preserving functions.

Bad news: they have super-linear growth ($\Theta(n^k)$ for any k).

A characterization of regular functions

Good news: $!\mathsf{Str}_\Gamma \multimap !\mathsf{Str}_\Sigma \rightsquigarrow$ regularity-preserving functions.
Bad news: they have super-linear growth ($\Theta(n^k)$ for any k).

Idea: drop the exponentials on the Str.

Theorem

$f : \Gamma^* \to \Sigma^*$ *regular* $\iff$ f *defined by* EAλ *term* $t : \mathsf{Str}_\Gamma \multimap \mathsf{Str}_\Sigma$.

A characterization of regular functions

Good news: $!\mathsf{Str}_\Gamma \multimap !\mathsf{Str}_\Sigma \rightsquigarrow$ regularity-preserving functions.
Bad news: they have super-linear growth ($\Theta(n^k)$ for any k).

Idea: drop the exponentials on the Str.

Theorem

$f : \Gamma^* \to \Sigma^*$ *regular* $\iff$ f *defined by* EAλ *term* $t : \mathsf{Str}_\Gamma \multimap \mathsf{Str}_\Sigma$.

Proof of ($\Longrightarrow$).

Encode the streaming string transducer in EAλ. □

Proof of ($\Longleftarrow$) **(with Pistone, Seiller & Tortora de Falco).**

Much more difficult: we need to construct a kind of
"*finitely parameterized semantics*" of *2nd-order* affine MLL
(+ constants) via combinatorial study of *proof nets*
(looks like concurrent games?) □

Relationship with implicit complexity in linear logic, again

Theorem (Simplification of a result by Baillot et al. 2014)

The functions defined by EAλ*-terms* with type fixpoints *of type* $!\mathsf{Str}_\Gamma \multimap !\mathsf{Str}_\Sigma$ *are exactly those computable in* polynomial time.

So, what about $!\mathsf{Str}_\Gamma \multimap !\mathsf{Str}_\Sigma$ in EAλ w/o type fixpoints?

Some class of functions that must be regularity-preserving...

Relationship with implicit complexity in linear logic, again

Theorem (Simplification of a result by Baillot et al. 2014)

The functions defined by EAλ*-terms* with type fixpoints *of type* $!\mathsf{Str}_\Gamma \multimap !\mathsf{Str}_\Sigma$ *are exactly those computable in* polynomial time.

So, what about $!\mathsf{Str}_\Gamma \multimap !\mathsf{Str}_\Sigma$ in EAλ w/o type fixpoints?
Some class of functions that must be regularity-preserving...

$$\text{regular} \subsetneq \text{this} \subseteq \textit{polyregular} \text{ functions (Bojańczyk 2018)}$$

(last inclusion probably strict; also, polyregular $\subsetneq$ HDT0L+composition)

"Slogan: Polyregular functions are the polynomial growth finite state transducers." Unpublished paper (`arXiv:1810.08760`).

Conclusion

This is only the start! Our concrete results for now:

- register transducers (HDT0L) + composition
  can be encoded in simply typed λ-calculus (STλ)
- regular functions can be characterized
  in the elementary affine λ-calculus
- not mentioned here: tree transducers vs additive '&'

Conclusion

This is only the start! Our concrete results for now:

- register transducers (HDT0L) + composition
  can be encoded in simply typed λ-calculus (STλ)
- regular functions can be characterized
  in the elementary affine λ-calculus
- not mentioned here: tree transducers vs additive '&'

Why this might be interesting:

- reviving an old question on STλ
- recent surge of interest in transductions with super-linear growth (HDT0L systems, polyregular functions)
  - STλ gives the largest (?) such "natural" class known

Conclusion

This is only the start! Our concrete results for now:

- register transducers (HDT0L) + composition
  can be encoded in simply typed λ-calculus (STλ)
- regular functions can be characterized
  in the elementary affine λ-calculus
- not mentioned here: tree transducers vs additive '&'

Why this might be interesting:

- reviving an old question on STλ
- recent surge of interest in transductions with super-linear growth (HDT0L systems, polyregular functions)
  - STλ gives the largest (?) such "natural" class known

Paper: `https://nguyentito.eu/tmp/transducer-p1.pdf`