Implicit automata in typed $\lambda$-calculi

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

GdT Plume, 8 Mars 2021
Simply typed functions on Church numerals

Church encodings of (unary) natural numbers:
- $\text{Nat} = (o \rightarrow o) \rightarrow o \rightarrow o$
- $n \in \mathbb{N} \mapsto \bar{n} = \lambda f. \lambda x. f \ldots (f \ x) \ldots : \text{Nat}$ with $n$ times $f$
- all inhabitants of $\text{Nat}$ are equal to some $\bar{n}$ up to $\equiv_{\beta\eta}$

**Theorem (Schwichtenberg 1975)**
The functions $\mathbb{N} \rightarrow \mathbb{N}$ definable by simply-typed $\lambda$-terms of type $\text{Nat} \rightarrow \text{Nat}$ are the extended polynomials (generated by $0, 1, +, \times, \text{id}$ and $\text{ifzero}$).
Simply typed functions on Church numerals

*Church encodings of (unary) natural numbers:*
  - \( \text{Nat} = (o \to o) \to o \to o \)
  - \( n \in \mathbb{N} \leadsto \bar{n} = \lambda f. \lambda x. f (\ldots (f x) \ldots) : \text{Nat} \) with \( n \) times \( f \)
  - all inhabitants of \( \text{Nat} \) are equal up to \( =_{\beta\eta} \)

**Theorem (Schwichtenberg 1975)**

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

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

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

**Open question**

Choose some simple type \( A \) and some term \( t : \text{Nat}[A] \to \text{Nat} \).
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! \( \text{Nat} = \text{Str}_{\{1\}} \)

Church encodings of strings over alphabet \( \Sigma = \{a, b\} \):

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

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

**Open question**

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

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

Without input type substitutions, an answer is known [Zaionc 1987].
Simply typed functions on Church-encoded strings

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

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

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

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

Open question

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

Note: unary regular languages $\cong$ ultimately periodic subsets of $\mathbb{N}$
\(\lambda\)-definable functions are regular

**Theorem (Hillebrand & Kanellakis, LICS’96)**

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

**Proof by semantic evaluation.**

Let \([-\cdot-]\) stand for the denotational semantics in the CCC of finite sets.

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

\[
\begin{array}{c}
\text{[e]} \\
\rightarrow \\
\text{[a]} \\
\text{[ab]} \\
\rightarrow \\
\text{[abb]} \\
\rightarrow \ldots
\end{array}
\]

\(t \bar{w} \beta \text{ true} \iff [[t]](\bar{w}) = [\text{true}] \iff w \text{ accepted}\)

(Proof of (\(\Leftarrow\)):
if \(\text{Card}([\cdot]) \geq 2\) then \([\text{true}] \neq [\text{false}]\)

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

Assume a $\lambda$-calculus for linear intuitionistic logic with additives

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

Church encoding with linear types [Girard 1987]:

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

Assume a $\lambda$-calculus for linear intuitionistic logic with additives

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

Church encoding with linear types [Girard 1987]:

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

Today’s main theorem [Nguyễn & P.]

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

$\iff$

$f$ is defined by some $t : \text{Str}_{\Gamma}[A] \rightarrow \text{Str}_\Sigma$ in the intuitionistic linear $\lambda$-calculus
with $A$ purely linear, i.e. containing no ‘$\rightarrow$’
Regular functions

Assume a λ-calculus for linear intuitionistic logic with additives

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

Church encoding with linear types [Girard 1987]:

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

Today’s main theorem [Nguyễn & P.]

$$f : \Gamma^* \to \Sigma^* \text{ is a regular function} \iff f \text{ is defined by some } t : \text{Str}_\Gamma[A] \to o \text{ in the intuitionistic linear } \lambda\text{-calculus with } A \text{ purely linear, i.e. containing no } \to$$

Regular functions are a classical topic, many equivalent definitions...
One of them: **copyless streaming string transducers** [Alur & Černý 2010]
\sim \text{ sounds suspiciously like affine types!}
Definition

- Finite set of $\Sigma^*$-valued registers e.g. $R = \{X, Y\}$
- Initial values $R \rightarrow \Sigma^*$ e.g. $X_{init} = Y_{init} = \varepsilon$
- 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. $\text{out} = XY$
Single-state streaming string transducers

**Definition**

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

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

$$X = \varepsilon \quad Y = \varepsilon$$
Single-state streaming string transducers

Definition

- Finite set of $\Sigma^*$-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. $\text{out} = XY$

Execution over $abaa$:

$$X = a \quad Y = a$$
**Single-state streaming string transducers**

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

Execution over $abaa$:

$$X = ab \quad Y = ba$$
Single-state streaming string transducers

**Definition**

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

Execution over $abaa$:

$X = aba \quad Y = aba$
Definition

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

Execution over $abaa$:

\[
X = abaa \quad Y = aaba
\]
Single-state streaming string transducers

**Definition**

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

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

$$X = abaa \quad Y = aaba$$
Single-state streaming string transducers

**Definition**

- Finite set of $\Sigma^*$-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. 
  \[
  \begin{align*}
  a &\mapsto \begin{cases} 
  X &\mapsto Xa \\
  Y &\mapsto aY 
  \end{cases} \\
  b &\mapsto \begin{cases} 
  X &\mapsto Xb \\
  Y &\mapsto bY 
  \end{cases}
  \end{align*}
  \]
- “output function” e.g. $\text{out} = XY$

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

\[
X = abaa \quad Y = aaba
\]
SSTs can also have states: their memory is $Q \times (\Sigma^*)^R$ (with $|Q| < \infty$)
SSTs can also have *states*: their memory is $Q \times (\Sigma^*)^R$ (with $|Q| < \infty$)

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

(Q \cong 1 \oplus \ldots \oplus 1, \text{concat} : \Sigma^* \otimes \Sigma^* \rightarrow \Sigma^*)
Categorical automata

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

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

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

- 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 $C$
- transitions = morphisms (and [letter $\rightarrow$ transition] = functor $T_\Sigma \rightarrow C$)

$$T_\Sigma = \bullet \xrightarrow{a \in \Sigma} \bullet \xrightarrow{\bullet} \rightarrow C$$

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

Formally

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

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

Notion of $\mathcal{C}$-automaton (abusively called $\mathcal{C}$-automata in the sequel)
**SSTs as categorical automata**

**The register category with output alphabet \( \Sigma \)**

- **Objects**: finite sets \( R, S \)
  - Think register variables
- **Morphisms**: \( \text{Hom}_R (R, S) = \text{maps } S \to (R + \Sigma)^* \) 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 \( \varepsilon, a : I \to \Sigma^* \) for \( a \in \Sigma \) and
  - \( \text{cat} : \Sigma^* \otimes \Sigma^* \to \Sigma^* \)
- **For the streaming setting**, take \( \top = I = 0 \) and \( \bot = \Sigma^* = \{\bullet\} \)
SSTs as categorical automata

The register category with output alphabet $\Sigma$

- **Objects:** finite sets $R, S$
  
- **Morphisms:** $\text{Hom}_R (R, S) = \text{maps } S \to (R + \Sigma)^*$ 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 $\varepsilon, a : I \to \Sigma^*$ for $a \in \Sigma$ and $\text{cat} : \Sigma^* \otimes \Sigma^* \to \Sigma^*$
- For the streaming setting, take $\top = I = 0$ and $\bot = \Sigma^* = \{\bullet\}$

Definition of the free finite coproduct completion $C_{\oplus}$

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

  $\cong \sum_f \prod_u \text{Hom}_C (C_u, D_{f(u)})$

- Morphisms $\bigoplus_{q \in Q} R \to \bigoplus_{q \in Q} R$ correspond to transitions in a SST
- Canonical embedding $C \to C_{\oplus}$ allows to lift streaming settings
Transductions definable in linear $\lambda$-calculus can be turned into automata over a category $\mathcal{L}$ of purely linear $\lambda$-terms (w/ $\text{const } f_c : o \rightarrow o$ for $c \in \Sigma$)

**Claim**

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

Proof: syntactic analysis of normal forms
Transductions definable in linear $\lambda$-calculus can be turned into automata over a category $\mathcal{L}$ of purely linear $\lambda$-terms (w/ $\text{const} f_c : o \to o$ for $c \in \Sigma$)

**Claim**

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

Proof: syntactic analysis of normal forms
Transductions definable in linear $\lambda$-calculus can be turned into automata over a category $\mathcal{L}$ of purely linear $\lambda$-terms (w/ const $f_c : o \to o$ for $c \in \Sigma$)

**Claim**

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

Proof: syntactic analysis of normal forms

**Proof strategy for linear $\lambda$-definable $\implies$ regular function**

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

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

Unfortunately $\mathcal{R}_{\oplus}$ is not monoidal closed...
Toward a monoidal closed category

So far, we encountered:

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

Now consider:

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

- the composite completion $\mathcal{C} \mapsto \mathcal{C}_\& \mapsto (\mathcal{C}_\&)_{\oplus}$
  
  **Objects:** formal sums of products $\bigoplus_u \&_x C_{u,x}$

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

Goals toward our main theorem

- Structure: $(\mathcal{R}_\&)_{\oplus}$ has finite products and is monoidal closed
- Conservativity: $(\mathcal{R}_\&)_{\oplus}$-automata and $\mathcal{R}_\oplus$-automata are equivalent
Tensorial products can be lifted to the completions

- The new tensorial products satisfy the additional laws
  \[ A \otimes (B \& C) \equiv (A \otimes B) \& (A \otimes C) \quad A \otimes (B \oplus C) \equiv (A \otimes B) \oplus (A \otimes C) \]

- In particular, \((C \&) \oplus\) 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 \((C_\&)_\oplus\)

Tensorial products can be lifted to the completions

- The new tensorial products satisfy the additional laws
  \[
  A \otimes (B \& C) \equiv (A \otimes B) \& (A \otimes C) \quad A \otimes (B \oplus C) \equiv (A \otimes B) \oplus (A \otimes C)
  \]

- In particular, \((C_\&)_\oplus\) 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 \(C\) is symmetric monoidal and \((C_\&)_\oplus\) has the internal homs \(A \multimap B\)
for all \(A, B \in C\), then \((C_\&)_\oplus\) is symmetric monoidal closed.

\[
\left( \bigoplus_{u \in U} \bigotimes_{x \in X_u} A_x \right) \multimap \left( \bigoplus_{v \in V} \bigotimes_{y \in Y_v} B_y \right) = \bigotimes_{u \in U} \bigoplus_{v \in V} \bigotimes_{y \in Y_v} x \in X_u A_x \multimap B_y
\]
Lemma

\( \mathcal{R} \oplus \) has the internal homs \( A \twoheadrightarrow B \) for all \( A, B \in \mathcal{R} \).

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

\[
\begin{align*}
X &:= abXcY \\
Y &:= ba \\
\end{align*}
\]

\( \Rightarrow \) shape

\[
\begin{align*}
X &:= Z_1XZ_2Y \\
Y &:= Z_3 + \text{parameters } Z_1 = ab, \ldots
\end{align*}
\]

copyless SST \( \Rightarrow \) finitely many shapes: use as states; registers for params
Lemma

\( \mathcal{R} \oplus \) has the internal homs \( A \to B \) for all \( A, B \in \mathcal{R} \).

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

\[
\begin{align*}
X &:= abXcY \\
Y &:= ba \\
\end{align*}
\]

\( \leadsto \) shape

\[
\begin{align*}
X &:= Z_1XZ_2Y \\
Y &:= Z_3 \\
\end{align*}
\]

+ parameters \( Z_1 = ab, \ldots \)

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

**Conclusion**

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

Lemma

\((C \&)_{\oplus} \) automata are equivalent to non-deterministic \(C_{\oplus} \) 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 \(C\), if \(C_{\oplus}\) has all the internal homsets \(A \rightarrow B\) for \(A, B \in C\), then \((C \&)_{\oplus}\)-automata and \(C_{\oplus}\)-automata are equivalent.

i.e., ND \(C_{\oplus}\)-automata can be uniformized
Main results

I have just discussed

**Today’s main theorem [Nguyễn & P.]**

regular string function $\iff$ definable by some $t : \text{Str}_\Gamma[A] \to \text{Str}_\Sigma$

in ILL with $A$ purely linear
Main results

I have just discussed

**Today’s main theorem [Nguyễn & P.]**

\[ \text{regular string function } \iff \text{definable by some } t : \text{Str}_\Gamma[A] \rightarrow \text{Str}_\Sigma \]

in ILL with \( A \) purely linear

Using similar tools, analogous result for trees over ranked alphabets

**Main theorem for trees [Nguyễn & P.]**

\[ \text{regular tree function } \iff \text{definable by some } t : \text{Tree}_\Gamma[A] \rightarrow \text{Tree}_\Sigma \]

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 $t : \text{Str}_\Gamma[A] \rightarrow \text{Str}_\Sigma$

in ILL with $A$ purely linear

Using similar tools, analogous result for trees over ranked alphabets

**Main theorem for trees [Nguyễn & P.]**

regular tree function $\iff$ definable by some $t : \text{Tree}_\Gamma[A] \rightarrow \text{Tree}_\Sigma$

in ILL with $A$ purely linear

Specific ingredients:

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

Additives are required for trees

*Copyless* streaming tree transducers ⊆ 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!

\[ M = Q \Sigma^* (\Sigma^* \& \Sigma^*) = \bigoplus_{q \in Q \Sigma^* (\Sigma^* \& \Sigma^*)} \]

String functions without additive

\[ \begin{align*}
\text{Still an equivalence, but non-trivial} \\
\text{(solution via Krohn–Rhodes)} \\
\text{Allows GoI-style interpretation in categories of diagrams} \\
\text{Interpretation as bidirectional automata (w/o registers)}
\end{align*} \]
Additive connectives: why?

**Additives are required for trees**

*Copyless* streaming tree transducers $\subset$ regular tree functions;
conjectured to be a *strict inclusion*.
To recover an equality: ad-hoc relaxation called “single use restriction”.

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

\[ M = Q \otimes \Sigma^* \otimes (\Sigma^* \& \Sigma^*) = \bigoplus_{q \in Q} \Sigma^* \otimes (\Sigma^* \& \Sigma^*) \]

String functions without additive

• Still an equivalence, but non-trivial (solution via Krohn–Rhodes)
• Allows GoI-style interpretation in categories of diagrams
\[ \Rightarrow \]
Interpretation as bidirectional automata (w/o registers)
Additive connectives: why?

Additives are required for trees

*Copyless* streaming tree transducers \(\subset\) regular tree functions;
conjectured to be a *strict inclusion*.
To recover an equality: ad-hoc relaxation called “single use restriction”.

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

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

String functions without additive

- Still an equivalence, but non-trivial (solution via Krohn–Rhodes)
- Allows GoI-style interpretation in categories of diagrams

\(\rightsquigarrow\) Interpretation as bidirectional automata (w/o registers)

Planar diagrams

\(\rightsquigarrow\)
FO fragments
Conclusion

Today:

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

Broader picture

<table>
<thead>
<tr>
<th>$\text{Str}_{\Sigma}[A] \to \text{Bool}$ with $A$ linear (adapted as needed):</th>
<th>$\lambda$-calculus</th>
<th>languages</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>simply typed</td>
<td>regular</td>
<td>✓ [Hillebrand &amp; Kanellakis 1996]</td>
<td></td>
</tr>
<tr>
<td>linear or affine</td>
<td>regular</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>non-commutative linear or affine</td>
<td>star-free</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\text{Str}<em>{\Gamma}[A] \to \text{Str}</em>{\Sigma}$ with $A$ affine (adapted as needed):</th>
<th>$\lambda$-calculus</th>
<th>transducers</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear (without additives)</td>
<td>nothing interesting (?)</td>
<td>✓ (?)</td>
<td></td>
</tr>
<tr>
<td>affine</td>
<td>regular functions</td>
<td>✓ (coming soon)</td>
<td></td>
</tr>
<tr>
<td>non-commutative affine</td>
<td>first-order regular fn.</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>linear/affine with additives</td>
<td>regular functions</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>parsimonious</td>
<td>polyregular</td>
<td>??</td>
<td></td>
</tr>
<tr>
<td>simply typed</td>
<td>variant of CPDA??</td>
<td>??</td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

Today:

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

Broader picture

<table>
<thead>
<tr>
<th>$\text{Str}_\Sigma[A] \to \text{Bool}$ with $A$ linear (adapted as needed):</th>
<th>$\lambda$-calculus</th>
<th>languages</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>simply typed</td>
<td>regular</td>
<td>✓ [Hillebrand &amp; Kanellakis 1996]</td>
<td></td>
</tr>
<tr>
<td>linear or affine</td>
<td>regular</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>non-commutative linear or affine</td>
<td>star-free</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\text{Str}<em>\Gamma[A] \to \text{Str}</em>\Sigma$ with $A$ affine (adapted as needed):</th>
<th>$\lambda$-calculus</th>
<th>transducers</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear (without additives)</td>
<td>nothing interesting (?)</td>
<td>✓ (?)</td>
<td></td>
</tr>
<tr>
<td>affine</td>
<td>regular functions</td>
<td>✓ (coming soon)</td>
<td></td>
</tr>
<tr>
<td>non-commutative affine</td>
<td>first-order regular fn.</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>linear/affine with additives</td>
<td>regular functions</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>parsimonious</td>
<td>polyregular</td>
<td>??</td>
<td></td>
</tr>
<tr>
<td>simply typed</td>
<td>variant of CPDA??</td>
<td>??</td>
<td></td>
</tr>
</tbody>
</table>

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

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

**Broader picture**

<table>
<thead>
<tr>
<th>$\text{Str}_\Sigma[A] \rightarrow \text{Bool}$ with $A$ linear (adapted as needed):</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda$-calculus</td>
<td>languages</td>
<td>status</td>
</tr>
<tr>
<td>simply typed</td>
<td>regular</td>
<td>✓</td>
</tr>
<tr>
<td>linear or affine</td>
<td>regular</td>
<td>✓</td>
</tr>
<tr>
<td>non-commutative linear or affine</td>
<td>star-free</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\text{Str}<em>\Gamma[A] \rightarrow \text{Str}</em>\Sigma$ with $A$ affine (adapted as needed):</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda$-calculus</td>
<td>transducers</td>
<td>status</td>
</tr>
<tr>
<td>linear (without additives)</td>
<td>nothing interesting (?)</td>
<td>✓  (?)</td>
</tr>
<tr>
<td>affine</td>
<td>regular functions</td>
<td>✓ (coming soon)</td>
</tr>
<tr>
<td>non-commutative affine</td>
<td>first-order regular fn.</td>
<td>✓</td>
</tr>
<tr>
<td>linear/affine with additives</td>
<td>regular functions</td>
<td>✓</td>
</tr>
<tr>
<td>parsimonious</td>
<td>polyregular</td>
<td>??</td>
</tr>
<tr>
<td>simply typed</td>
<td>variant of CPDA??</td>
<td>??</td>
</tr>
</tbody>
</table>

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

---

Thanks for listening!