Implicit automata in typed λ-calculi

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

LHC, February 5th, 2021
Simply typed functions on Church numerals

Church encodings of (unary) natural numbers:
- $\text{Nat} = (o \to o) \to o \to o$
- $n \in \mathbb{N} \mapsto \overline{n} = \lambda f. \lambda x. f (\ldots (f x)\ldots) : \text{Nat}$ with $n$ times $f$
- all inhabitants of $\text{Nat}$ are equal to some $\overline{n}$ 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 $\text{ifzero}$).
Simply typed functions on Church numerals

Church encodings of (unary) natural numbers:

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

**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, id and ifzero).

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

where \( \text{Nat}[A] = \text{Nat}[^A] = (A \rightarrow A) \rightarrow A \rightarrow A \)

**Open question**

Choose some simple type \( A \) and some term \( t : \text{Nat}[A] \rightarrow \text{Nat} \).
What functions \( \mathbb{N} \rightarrow \mathbb{N} \) 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 \( \Sigma = \{a, b\} \):

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

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

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! \( \text{Nat} = \text{Str}_{\{1\}} \)

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 \( \mathbb{N} \)
\(\lambda\text{-definable functions are regular}\)

**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 \([\_]\) stand for the denotational semantics in the CCC of finite sets.

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

\[
\begin{array}{cccccc}
[\varepsilon] & \xrightarrow{a} & [\bar{a}] & \xrightarrow{b} & [ab] & \xrightarrow{b} [abb] & \cdots \\
\end{array}
\]

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

(Proof of \(\leftarrow\): if \(\text{Card}([\varepsilon]) \geq 2\) 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

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

Church encoding with linear types [Girard 1987]:

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

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

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

Church encoding with linear types [Girard 1987]:

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

Today’s main theorem [Nguyễn & P.]

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

$$\iff$$

$f$ 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 ‘$\rightarrow$’
Assume a λ-calculus for linear intuitionistic logic with additives

- $\lambda^\rightarrow x. t : A \to B$ 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]:

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

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

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

Regular functions are a classical topic, many equivalent definitions...
One of them: **copyless streaming string transducers** [Alur & Černý 2010]

$\rightsquigarrow$ sounds suspiciously like affine types!
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. $\text{out} = XY$
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. $\text{out} = XY$

Execution over $abaa$: start with

$$X = \varepsilon \quad Y = \varepsilon$$
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 $abaa$:

\[
X = a \quad Y = a
\]
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. $a \mapsto \begin{cases} X := X^a \\ Y := aY \end{cases}$, $b \mapsto \begin{cases} X := X^b \\ Y := bY \end{cases}$
- “output function” e.g. $\text{out} = XY$

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 $R \rightarrow \Sigma^*$ 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. $\text{out} = XY$

Execution over $abaa$:

\[ X = aba \quad Y = aba \]
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. $\text{out} = XY$

Execution over $abaa$:

$$X = abaa \quad Y = aaba$$
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. $\text{out} = XY$

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 $R \rightarrow \Sigma^*$ 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. $\text{out} = XY$

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

$X = abaa \quad Y = aaba$
Stateful streaming string transducers

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 $|Q| < \infty$).

**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 $M \rightarrow M$

($Q \cong 1 \oplus \ldots \oplus 1$, 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 $T_\Sigma \to C$)

$\mathcal{T}_\Sigma = \bullet \quad \bullet \quad \bullet \quad \to \quad C$

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

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

- 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 $\mathcal{C}$
- **Morphisms**: $\text{Hom}_{\mathcal{C}_\oplus} \left( \bigoplus_{u} C_u, \bigoplus_{v} D_v \right) = \prod_{u} \sum_{v} \text{Hom}_{\mathcal{C}} \left( C_u, D_v \right)$
Categorical automata

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

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

\[
T_\Sigma = \bullet \xrightarrow{a \in \Sigma} \bullet \xrightarrow{} \bullet \xrightarrow{} C
\]

- 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 \((-)\oplus\)

Definition of the free finite coproduct completion \( C_\oplus \)

- Objects: formal finite sums \( \bigoplus_{u \in U} C_u \) of objects of \( C \)
  formally pairs \((U, (C_u)_{u \in U}), U \) a finite set, \( C_u \in C_0\)
- Morphisms: \( \text{Hom}_{C_\oplus} \left( \bigoplus_{u \in U} C_u, \bigoplus_{v} D_v \right) = \prod_{u} \sum_{v} \text{Hom}_C \left( C_u, D_v \right) \)
Categorical automata

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

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

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

- 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 $(\_ \_ \_)$

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:** $\text{Hom}_{\mathcal{C}_\oplus} \left( \bigoplus_{u} C_u, \bigoplus_{v} D_v \right) = \prod_{u} \sum_{v} \text{Hom}_{\mathcal{C}} (C_u, D_v)$
  $$\approx \sum_f \prod_{u} \text{Hom}_{\mathcal{C}} (C_u, D_{f(u)})$$
Compiling into higher-order transducers

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/ $\text{const} f_c : o \rightarrow o$ for $c \in \Sigma$)

**Claim**

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

**Proof:** syntactic analysis of normal forms

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

Define a functor $\mathcal{L} \rightarrow \mathcal{R}_{\oplus}$ 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 \( f_c : o \to o \) 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 \( \&_x \mathcal{C}_x \)

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

  similar to de Paiva’s *Dialectica* categories \( \text{DC} \), 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 \((\text{co})\text{presheafs} \cong \text{Day convolution}\)
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 \((\text{co})\text{presheafs} \cong \text{Day convolution}\)

**Lemma** \(((\text{folklore observation about dependent Dialectica categories?}))\)

*If \(C\) is symmetric monoidal and \((C_\&)_{\oplus}\) has the internal homs \(A \to B\) for all \(A, B \in C\), then \((C_\&)_{\oplus}\) is symmetric monoidal closed.*

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

\[ \mathcal{R} \oplus \text{ has the internal homs } A \rightarrow B \text{ for all } A, B \in \mathcal{R}. \]

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

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

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

*copyless* SST \[ \implies \] 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.

\[
\begin{aligned}
X &:= abXcY \\
Y &:= ba
\end{aligned}
\quad \Rightarrow \quad
\begin{aligned}
X &:= Z_1XZ_2Y \\
Y &:= Z_3
\end{aligned}
\quad + \quad \text{parameters } Z_1 = ab, \ldots
\]

\textit{copyless} SST \quad \Rightarrow \quad \text{finitely many shapes: use as states; registers for params}

Conclusion

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

**Lemma**

\((C\&)_+\) automata are equivalent to non-deterministic \(C_+\) automata.

A uniformization (\(\sim\) determinization) theorem is enough to conclude

**Conservativity**

\((\mathcal{R}\&)_+\)-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 \(C_+\) has all the internal homsets \(A \rightarrow B\) for \(A, B \in C\), then \((C\&)_+\)-automata and \(C_+\)-automata are equivalent.*

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

I have just discussed

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

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

in ILL with $A$ purely linear
Main results

I have just discussed

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

<table>
<thead>
<tr>
<th>regular string function</th>
<th>$\iff$</th>
<th>definable by some $t : \text{Str}<em>\Gamma[A] \to \text{Str}</em>\Sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>in ILL with $A$ purely linear</td>
</tr>
</tbody>
</table>

Using similar tools, analogous result for trees over ranked alphabets

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

<table>
<thead>
<tr>
<th>regular tree function</th>
<th>$\iff$</th>
<th>definable by some $t : \text{Tree}<em>\Gamma[A] \to \text{Tree}</em>\Sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>in ILL with $A$ purely linear</td>
</tr>
</tbody>
</table>
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
```

Using similar tools, analogous result for trees over ranked alphabets

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

```
regular tree function \iff definable by some \( t : \text{Tree}_\Gamma[A] \to \text{Tree}_\Sigma \)
in ILL with \( A \) purely linear
```

Specific ingredients:

- Bottom-up categorical tree automata over SMCs
- A comparison of \( C_\& \) with a kind of *coherence completion* similar to [Hu, Joyal]
- A reasonably elegant multicategory of tree registers transition
Conclusion

Today:

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

**Broader picture**

\[ \text{Str}_\Sigma[A] \to \text{Bool} \]

with \( A \) linear (adapted as needed):

<table>
<thead>
<tr>
<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>
</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>

\[ \text{Str}_\Gamma[A] \to \text{Str}_\Sigma \]

with \( A \) affine (adapted as needed):

<table>
<thead>
<tr>
<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>
</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>

Thanks for listening!
Conclusion

Today:

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

Broader picture

\( \text{Str}_\Sigma[A] \rightarrow \text{Bool} \) with \( A \) linear (adapted as needed):

<table>
<thead>
<tr>
<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>
</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>

\( \text{Str}_\Gamma[A] \rightarrow \text{Str}_\Sigma \) with \( A \) affine (adapted as needed):

<table>
<thead>
<tr>
<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>
</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
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>$\text{Str}_\Sigma[A] \rightarrow \text{Bool}$ with $A$ linear (adapted as needed):</th>
<th>\text{\lambda-calculus}</th>
<th>\text{languages}</th>
<th>\text{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>$\text{Str}<em>\Gamma[A] \rightarrow \text{Str}</em>\Sigma$ with $A$ affine (adapted as needed):</th>
<th>\text{\lambda-calculus}</th>
<th>\text{transducers}</th>
<th>\text{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 $\text{Str}[A] \rightarrow \text{Str}$ as comparison-free polyregular functions

Thanks for listening!