Implicit automata in typed λ-calculi: regular functions

Cécilia Pradic (Swansea University)
j.w.w. Lê Thành Dũng (a.k.a. Tito) Nguyễn (ÉNS Lyon)

June 28th 2023

The untyped λ-calculus

Syntax

Defined inductively, with x ranging over variables

$$t, u ::= x \mid \lambda x.t \mid t\ u$$

The untyped λ-calculus

Syntax

Defined inductively, with x ranging over variables

$$t, u ::= x \mid \lambda x.t \mid t\ u$$

- Introduced by Alonzo Church in the 30s

The untyped λ-calculus

Syntax

Defined inductively, with x ranging over variables

$$t, u ::= x \mid \lambda x.t \mid t\ u$$

- Introduced by Alonzo Church in the 30s
- An algebra for anonymous *functions*

$$\lambda x.t \simeq x \mapsto t$$

- The core functional programming language

Real-world examples of extensions: Scheme, ML, Haskell...

The untyped λ-calculus

Syntax

Defined inductively, with x ranging over variables

$$t, u ::= x \mid \lambda x.t \mid t\ u$$

- Introduced by Alonzo Church in the 30s
- An algebra for anonymous *functions*

$$\lambda x.t \ \simeq \ x \mapsto t$$

- The core functional programming language

Real-world examples of extensions: Scheme, ML, Haskell…

- Turing-complete

Some λ-terms

Examples

- The identity function $\lambda x.\, x$
- Composition $\lambda f\, g\, x.\, f\,(g\, x)$
- Church numeral $\underline{2} = \lambda s\, z.\, s\,(s\, z)$
- Stranger things... $\lambda x.\, x\, x$

Some technical details:

- Equality up to renaming of bound variables — α-conversion
- Notations: $\lambda x\, y.t \;=\; \lambda x.\lambda y.t$ and $t\, u\, v \;=\; (t\, u)\, v$
- Capture-avoiding subsitution $t[u/x]$

$$x[t/x] \;=\; t \qquad (t\, u)[v/x] \;=\; t[v/x]\; u[v/x] \qquad (\lambda y.t)[u/x] \;=\; \lambda y.t[u/x] \quad (x \neq y, y \notin FV(u))$$

Computing with the pure λ-calculus

One-step β-reduction

$\to_\beta$ is the closure under congruence of

$$(\lambda x.t)\ u \quad \to_\beta \quad t[u/x]$$

Computing with the pure λ-calculus

One-step β-reduction

$\to_\beta$ is the closure under congruence of

$$(\lambda x.t)\ u \quad \to_\beta \quad t[u/x]$$

- $\to_\beta$ non-deterministic

Computing with the pure λ-calculus

One-step β-reduction

$\to_\beta$ is the closure under congruence of

$$(\lambda x.t)\ u \quad \to_\beta \quad t[u/x]$$

- $\to_\beta$ non-deterministic
- Call $\to^*$ its reflexive transitive closure
- Call a λ-term t **normal** if $t \not\to_\beta$

Computing with the pure λ-calculus

One-step β-reduction

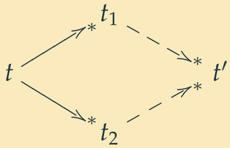
$\to_\beta$ is the closure under congruence of

$$(\lambda x.t)\ u \quad \to_\beta \quad t[u/x]$$

- $\to_\beta$ non-deterministic
- Call $\to^*$ its reflexive transitive closure
- Call a λ-term t **normal** if $t \not\to_\beta$

Theorem

$\to^*$ is confluent



Computing with the pure λ-calculus

One-step β-reduction

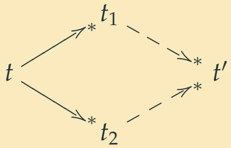
$\to_\beta$ is the closure under congruence of

$$(\lambda x.t)\ u \quad \to_\beta \quad t[u/x]$$

- $\to_\beta$ non-deterministic
- Call $\to^*$ its reflexive transitive closure
- Call a λ-term t **normal** if $t \not\to_\beta$

Theorem

$\to^*$ is confluent



$\leadsto$ Well-behaved notion of computation

- Each term reduce to ≤ 1 normal form
- Independent of the evaluation strategy
- Example:

$$\begin{aligned} \lambda s.\underline{1}\ s\ (\underline{1}\ s) &\to_\beta \lambda s.(\lambda z.s\ z)\ (\underline{1}\ s) \\ &\to_\beta \lambda s.(\lambda z.s\ z\ (\underline{1}\ s) \\ &\to_\beta \lambda s.(\lambda z.s\ z)(\lambda z.s\ z) \\ &\to_\beta \lambda s.s\ (s\ z) \end{aligned}$$

Type systems

Rationale

Classify well-behaved sets of programs

- Practical motivations — Crash-free programs
- Proof-theoretical motivations — Curry-Howard correspondence

Type systems

Rationale

Classify well-behaved sets of programs

- Practical motivations — Crash-free programs
- Proof-theoretical motivations — Curry-Howard correspondence

All type systems for λ-calculus **here**after will satisfy the following

Subject reduction (SR)

If $t \to_\beta u$ and t has type A $(t : A)$, then so does u

"Types are invariant under computation"

Type systems

Rationale

Classify well-behaved sets of programs

- Practical motivations — Crash-free programs
- Proof-theoretical motivations — Curry-Howard correspondence

All type systems for λ-calculus **here**after will satisfy the following

Subject reduction (SR)

If $t \to_\beta u$ and t has type A ($t : A$), then so does u

"Types are invariant under computation"

Strong normalization (SN)

If $t : A$, then t will always reduce to a normal form.

"All typed programs terminate (no matter what is the evaluation strategy)"

Simply-typed λ-calculus

Simple types

$$A, B \ ::= \ o \ | \ A \to B$$

o is a fixed ground type

Typing rules

- Variable

$$\frac{}{\Gamma, x : A, \Gamma' \vdash x : A}$$

- λ-abstraction

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A}$$

- Application

$$\frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u : A}$$

System F

Simple types

$$A, B ::= X \mid \forall X.A \mid A \to B$$

X is a type variable

Typing rules

STLC rules with

- $\forall$-intro (X free in Γ)

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall X.A}$$

- $\forall$-elim

$$\frac{\Gamma \vdash t : \forall X.A}{\Gamma \vdash t : A[B/X]}$$

- SN much harder to prove

Requires impredicativity

- Convenient for programming

Church encoding of strings

Impredicative encodings

- The type of booleans Bool

$$\forall X. X \to X \to X$$

$$\underline{\text{true}} = \lambda x\, y. x \qquad \underline{\text{false}} = \lambda x\, y. y$$

- The type of natural numbers $\mathbb{N}$

$$\forall X. (X \to X) \to X \to X$$

- The type of binary strings Str

$$\forall X. (X \to X) \to (X \to X) \to X \to X$$

Church encoding $w \mapsto \underline{w}$ of strings $\{a, b\}^*$ into Str

$$abba \qquad \mapsto \qquad \lambda a\; b\; e.\; a\; (b\; (b\; (a\; e)))$$

Church encoding of strings

Impredicative encodings

- The type of booleans Bool
$$\forall X. X \to X \to X$$
$$\underline{\text{true}} = \lambda x\, y. x \qquad \underline{\text{false}} = \lambda x\, y. y$$
- The type of natural numbers $\mathbb{N}$
$$\forall X. (X \to X) \to X \to X$$
- The type of binary strings Str
$$\forall X. (X \to X) \to (X \to X) \to X \to X$$

Church encoding $w \mapsto \underline{w}$ of strings $\{a, b\}^*$ into Str

$$abba \qquad \mapsto \qquad \lambda a\; b\; e.\; a\; (b\; (b\; (a\; e)))$$

Consequence of SN

For every t : Str, there is $w \in \{a, b\}^*$ s.t. $t \to^* \underline{w}$

Affine/Linear λ-calculus

Resource-aware decomposition of STLC/System F.

Affine/Linear types

$$A, B \;::=\; !A \;\mid\; A \otimes B \;\mid\; A \multimap B \;\mid\; \dots$$

- Terms of type $A \multimap B$ use their arguments at most/exactly once

$$\frac{}{x : A \vdash x : A} \qquad \frac{\Gamma \vdash t : A \multimap B \qquad \Delta \vdash u : A}{\Gamma, \Delta \vdash t\ u : B}$$

Affine/Linear λ-calculus

Resource-aware decomposition of STLC/System F.

Affine/Linear types

$$A, B \ ::= \ !A \ | \ A \otimes B \ | \ A \multimap B \ | \ \ldots$$

- Terms of type $A \multimap B$ use their arguments at most/exactly once

$$\frac{}{x : A \vdash x : A} \qquad \frac{\Gamma \vdash t : A \multimap B \qquad \Delta \vdash u : A}{\Gamma, \Delta \vdash t\ u : B}$$

- ! allows duplication and discarding

$$!A \multimap\ !A \otimes\ !A \qquad\qquad !A \otimes B \multimap B$$

$\rightsquigarrow$ Encode $A \to B$ as $!A \multimap B$

Affine/Linear λ-calculus

Resource-aware decomposition of STLC/System F.

Affine/Linear types

$$A, B \ ::= \ !A \ \mid \ A \otimes B \ \mid \ A \multimap B \ \mid \ \ldots$$

- Terms of type $A \multimap B$ use their arguments at most/exactly once

$$\frac{}{x : A \vdash x : A} \qquad \frac{\Gamma \vdash t : A \multimap B \qquad \Delta \vdash u : A}{\Gamma, \Delta \vdash t\ u : B}$$

- ! allows duplication and discarding

$$!A \multimap !A \otimes !A \qquad\qquad !A \otimes B \multimap B$$

$\rightsquigarrow$ Encode $A \to B$ as $!A \multimap B$

Example

Str is isomorphic to

$$\mathsf{Str}^L \quad = \quad \forall X.!(X \multimap X) \multimap !(X \multimap X) \multimap X \multimap X$$

but certainly not to $\forall X.(X \multimap X) \multimap (X \multimap X) \multimap X \multimap X$

Implicit computational complexity

Problems

Fix a programming language and a type $\mathsf{Str} \to \mathsf{Bool}$

$\rightsquigarrow$ class of functions implemented by terms $t : \mathsf{Str} \to \mathsf{Bool}$?

- Landmark paper: safe recursion

  PTIME [Bellantoni-Cook, 1992]

- A few characterizations based on linear λ-calculi

  LLL/μELL2 for **PTIME** for instance [Girard 1996, Baillot 2005]

Implicit computational complexity

Problems

Fix a programming language and a type $\mathsf{Str} \to \mathsf{Bool}$

$\rightsquigarrow$ class of functions implemented by terms $t : \mathsf{Str} \to \mathsf{Bool}$?

- Landmark paper: safe recursion

  PTIME [Bellantoni-Cook, 1992]

- A few characterizations based on linear λ-calculi

  LLL/μELL2 for **PTIME** for instance [Girard 1996, Baillot 2005]

Remark

Un(i)typed λ-calculus	$\simeq$	recursive functions
System F	$\simeq$	PA2-definable recursive functions

Implicit computational complexity

Problems

Fix a programming language and a type $\mathsf{Str} \to \mathsf{Bool}$

$\leadsto$ class of functions implemented by terms $t : \mathsf{Str} \to \mathsf{Bool}$?

- Landmark paper: safe recursion

  PTIME [Bellantoni-Cook, 1992]

- A few characterizations based on linear λ-calculi

  LLL/μELL2 for **PTIME** for instance [Girard 1996, Baillot 2005]

Remark

Un(i)typed λ-calculus	$\simeq$	recursive functions
System F	$\simeq$	PA2-definable recursive functions

$\leadsto$ What about STLC?

Impredicative encodings in STLC

A slight wrinkle: quantification unavailable

- Define $\mathsf{Str}[A] = (A \to A) \to (A \to A) \to A \to A$
  $\mathsf{Bool}[A] = A \to A \to A$

Definition

We call a language $L \subseteq \{a, b\}^*$ definable in STLC iff there exists

- a simple type A
- a simply-typed λ-term $t : \mathsf{Str}[A] \to \mathsf{Bool}[o]$

such that for every $w \in \{a, b\}^*$,

$$t\ \underline{w} \quad \to^* \underline{\text{true}} \qquad \text{iff} \qquad w \in L$$

- Note that if $t : \mathsf{Str}[A] \to \mathsf{Bool}[o]$, then $t : \mathsf{Str} \to \mathsf{Bool}$ in System F

$\rightsquigarrow$ SN guarantees $t\ \underline{w} \to^* \underline{\text{true}}$ or $t\ \underline{w} \to^* \underline{\text{false}}$ for every $w \in \{a, b\}^*$

Hillebrand and Kanellakis' result

Theorem [Hillebrand and Kanellakis, 1996]

The STLC definable languages are the regular languages.

Hillebrand and Kanellakis' result

Theorem [Hillebrand and Kanellakis, 1996]

The STLC definable languages are the regular languages.

($\Leftarrow$) The encoding of DFAs in System F goes through

- I.e., impredicative quantification are spurious

Hillebrand and Kanellakis' result

Theorem [Hillebrand and Kanellakis, 1996]

The STLC definable languages are the regular languages.

($\Leftarrow$) The encoding of DFAs in System F goes through

- I.e., impredicative quantification are spurious

($\Rightarrow$) Semantic evaluation of $t : \mathsf{Str}[A] \to \mathsf{Bool}[o]$

Hillebrand and Kanellakis' result

Theorem [Hillebrand and Kanellakis, 1996]

The STLC definable languages are the regular languages.

($\Leftarrow$) The encoding of DFAs in System F goes through

- I.e., impredicative quantification are spurious

($\Rightarrow$) Semantic evaluation of $t : \mathsf{Str}[A] \to \mathsf{Bool}[o]$

- Interpret types as finite sets with $[\![o]\!] = \{\text{true}, \text{false}\}$

Set inductively $[\![A \to B]\!] = [\![B]\!]^{[\![A]\!]}$

- At the level of λ-terms, $t : A$ yields $[\![t]\!] \in [\![A]\!]$

Hillebrand and Kanellakis' result

Theorem [Hillebrand and Kanellakis, 1996]

The STLC definable languages are the regular languages.

($\Leftarrow$) The encoding of DFAs in System F goes through

- I.e., impredicative quantification are spurious

($\Rightarrow$) Semantic evaluation of $t : \mathsf{Str}[A] \to \mathsf{Bool}[o]$

- Interpret types as finite sets with $[\![o]\!] = \{\text{true}, \text{false}\}$

Set inductively $[\![A \to B]\!] = [\![B]\!]^{[\![A]\!]}$

- At the level of λ-terms, $t : A$ yields $[\![t]\!] \in [\![A]\!]$
- Note that $w \mapsto wa$ is definable by a term $c_a : \mathsf{Str}[A] \to \mathsf{Str}[A]$

Hillebrand and Kanellakis' result

Theorem [Hillebrand and Kanellakis, 1996]

The STLC definable languages are the regular languages.

($\Leftarrow$) The encoding of DFAs in System F goes through

- I.e., impredicative quantification are spurious

($\Rightarrow$) Semantic evaluation of $t : \mathsf{Str}[A] \to \mathsf{Bool}[o]$

- Interpret types as finite sets with $[\![o]\!] = \{\text{true}, \text{false}\}$

Set inductively $[\![A \to B]\!] = [\![B]\!]^{[\![A]\!]}$

- At the level of λ-terms, $t : A$ yields $[\![t]\!] \in [\![A]\!]$
- Note that $w \mapsto wa$ is definable by a term $c_a : \mathsf{Str}[A] \to \mathsf{Str}[A]$
- Build a DFA $(Q, [\![\epsilon]\!], \delta, F)$

$$Q = [\![\mathsf{Str}[A]]\!] \qquad \delta(q, a) = [\![c_a]\!](q) \qquad q \in F \Leftrightarrow [\![t]\!](q) \;=\; [\![\underline{\text{true}}]\!]$$

What about $\mathsf{Str} \to \mathsf{Str}$?

Definition

We call a function $f : \{a,b\}^* \to \{a,b\}^*$ **definable in STLC** iff there is

- a simple type A
- a simply-typed λ-term $t : \mathsf{Str}[A] \to \mathsf{Str}[o]$

such that for every $w \in \{a,b\}^*$,

$$t\ \underline{u} \quad \to^* \underline{v} \qquad \text{iff} \qquad f(u) = v$$

What about $\mathsf{Str} \to \mathsf{Str}$?

Definition

We call a function $f : \{a, b\}^* \to \{a, b\}^*$ **definable in STLC** iff there is

- a simple type A
- a simply-typed λ-term $t : \mathsf{Str}[A] \to \mathsf{Str}[o]$

such that for every $w \in \{a, b\}^*$,

$$t\ \underline{u} \quad \to^* \underline{v} \qquad \text{iff} \qquad f(u) = v$$

- Closed under composition

What about $\mathsf{Str} \to \mathsf{Str}$?

Definition

We call a function $f : \{a, b\}^* \to \{a, b\}^*$ **definable in STLC** iff there is

- a simple type A
- a simply-typed λ-term $t : \mathsf{Str}[A] \to \mathsf{Str}[o]$

such that for every $w \in \{a, b\}^*$,

$$t\ \underline{u} \quad \to^* \underline{v} \qquad \text{iff} \qquad f(u) = v$$

- Closed under composition
- By Hillebrand and Kanellakis' theorem

$$f \text{ STLC-definable} \quad \wedge \quad L \text{ regular} \qquad \Rightarrow \qquad f^{-1}(L) \text{ regular}$$

What about $\mathsf{Str} \to \mathsf{Str}$?

Definition

We call a function $f : \{a,b\}^* \to \{a,b\}^*$ **definable in STLC** iff there is

- a simple type A
- a simply-typed λ-term $t : \mathsf{Str}[A] \to \mathsf{Str}[o]$

such that for every $w \in \{a,b\}^*$,

$$t\ \underline{u} \quad \to^* \ \underline{v} \qquad \text{iff} \qquad f(u) = v$$

- Closed under composition
- By Hillebrand and Kanellakis' theorem

$$f \text{ STLC-definable} \quad \wedge \quad L \text{ regular} \qquad \Rightarrow \qquad f^{-1}(L) \text{ regular}$$

- Contains HDT0L-transduction

$\equiv$ copyful streaming string transducers, a kind of register automata

What about $\mathsf{Str} \to \mathsf{Str}$?

Definition

We call a function $f : \{a, b\}^* \to \{a, b\}^*$ **definable in STLC** iff there is

- a simple type A
- a simply-typed λ-term $t : \mathsf{Str}[A] \to \mathsf{Str}[o]$

such that for every $w \in \{a, b\}^*$,

$$t\ \underline{u} \quad \to^* \underline{v} \qquad \text{iff} \qquad f(u) = v$$

- Closed under composition
- By Hillebrand and Kanellakis' theorem

$$f \text{ STLC-definable} \quad \wedge \quad L \text{ regular} \qquad \Rightarrow \qquad f^{-1}(L) \text{ regular}$$

- Contains HDT0L-transduction

  $\equiv$ copyful streaming string transducers, a kind of register automata

- But we do not know more at the moment

Simplification: the affine case

We now turn to affine λ-calculus and set

$$\mathsf{Str}^{L}[A] = !(A \multimap A) \multimap !(A \multimap A) \multimap A \multimap A$$

Simplification: the affine case

We now turn to affine λ-calculus and set

$$\mathsf{Str}^L[A] = !(A \multimap A) \multimap !(A \multimap A) \multimap A \multimap A$$

Definition

We call $f : \{a,b\}^* \to \{a,b\}^*$ definable in affine STLC iff there is

- a **!-free** linear type A (i.e., $!o$ or $\mathsf{Str}^L[o]$ unavailable)
- a simply-typed **affine** λ-term $t : \mathsf{Str}^L[A] \to \mathsf{Str}^L[o]$

such that for every $u, v \in \{a,b\}^*$,

$$t\ \underline{u} \quad \to^* \underline{v} \qquad \text{iff} \qquad f(u) = v$$

Simplification: the affine case

We now turn to affine λ-calculus and set

$$\mathsf{Str}^L[A] = !(A \multimap A) \multimap !(A \multimap A) \multimap A \multimap A$$

Definition

We call $f : \{a,b\}^* \to \{a,b\}^*$ definable in affine STLC iff there is

- a **!-free** linear type A (i.e., $!o$ or $\mathsf{Str}^L[o]$ unavailable)
- a simply-typed **affine** λ-term $t : \mathsf{Str}^L[A] \to \mathsf{Str}^L[o]$

such that for every $u, v \in \{a,b\}^*$,

$$t\ \underline{u} \quad \to^* \underline{v} \qquad \text{iff} \qquad f(u) = v$$

- Same niceness properties as for STLC-definable

Regular functions: main theorem for string

Theorem [Nguyễn & P.]

f is regular a *regular function*

$\iff$

f is definable in the affine STLC

$\iff$

f is definable in the linear STLC with additives $\oplus, \&$

Regular functions: main theorem for string

Theorem [Nguyễn & P.]

f is regular a *regular function*

$\iff$

f is definable in the affine STLC

$\iff$

f is definable in the linear STLC with additives $\oplus$, $\&$

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

Single-state streaming string transducers

Definition

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

Single-state streaming string transducers

Definition

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

Execution over *abaa*: start with

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

Single-state streaming string transducers

Definition

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

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

Execution over *abaa*:

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

Single-state streaming string transducers

Definition

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

Execution over *abaa*:

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

Single-state streaming string transducers

Definition

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

Execution over *abaa*:

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

Single-state streaming string transducers

Definition

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

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

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

Single-state streaming string transducers

Definition

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

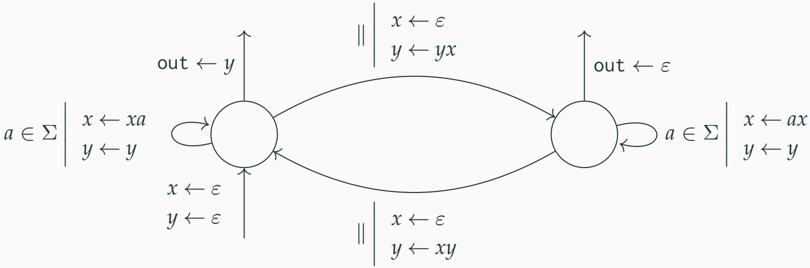
Execution over *abaa*: $f(abaa) = abaaaaba$

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

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

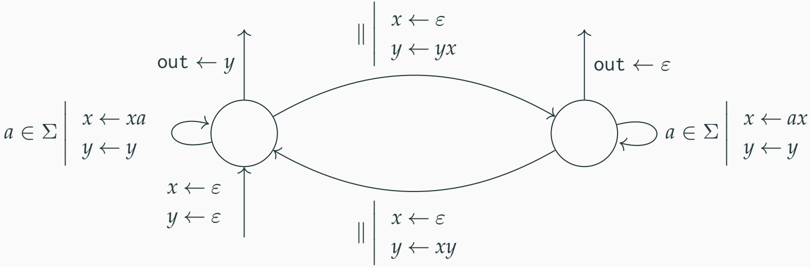
Stateful streaming string transducers

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 $\leftarrow$

(for each fixed input letter, at most once among all the associated $\leftarrow$)

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

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

Categorical automata

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

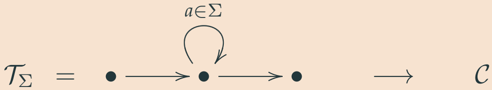
- internal memory = object of a *category* $\mathcal{C}$
- transitions = morphisms (and [letter $\mapsto$ transition] = functor $\mathcal{T}_\Sigma \to \mathcal{C}$)

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

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



Formally

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

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

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

Categorical automata: simple examples

Formally

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

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

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

Some examples in the wild:

- DFAs: $\mathcal{C} = \mathrm{Finset}$, $\top\!\!\top = 1$, $\perp\!\!\perp = 2$
- $\mathrm{Hom}_{\mathcal{C}}(n, k) = \mathbb{Q}[X_1, \ldots, X_n]^k$ = polynomial automata
- Exercise: $\mathcal{C}$ = the Lawvere theory of the string, $\top\!\!\top = 1$, $\perp\!\!\perp = t$

SSTs as categorical automata

The register category with output alphabet Σ

- **Objects:** finite sets R, S — think register variables
- **Morphisms:** $\mathrm{Hom}_{\mathcal{R}}(R, S) =$ maps $S \to (R + \Sigma)^*$ corresponding to copyless register affectations

  $\sum_{s \in S} |f(s)|_r \leq 1$

- Monoidal with $\otimes = +$
- Free affine monoidal category over an object $\Sigma^* = \{\bullet\}$, morphisms $\varepsilon, a : \mathbf{I} \to \Sigma^*$ for $a \in \Sigma$ and $cat : \Sigma^* \otimes \Sigma^* \to \Sigma^*$
- For the streaming setting, take $\top = \mathbf{I} = 0$ and $\perp = \Sigma^* = \{\bullet\}$

SSTs as categorical automata

The register category with output alphabet Σ

- **Objects:** finite sets R, S — think register variables
- **Morphisms:** $\text{Hom}_{\mathcal{R}}(R, S) =$ maps $S \to (R + \Sigma)^*$ corresponding to copyless register affectations

$\sum_{s \in S} |f(s)|_r \leq 1$

- Monoidal with $\otimes = +$
- Free affine monoidal category over an object $\Sigma^* = \{\bullet\}$, morphisms $\varepsilon, a : \mathbf{I} \to \Sigma^*$ for $a \in \Sigma$ and $cat : \Sigma^* \otimes \Sigma^* \to \Sigma^*$
- For the streaming setting, take $\top = \mathbf{I} = 0$ and $\bot = \Sigma^* = \{\bullet\}$

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

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

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

Compiling into higher-order transducers

Transductions definable in linear λ-calculus can be turned into automata over a category $\mathcal{L}$ of purely linear λ-terms (w/ const $f_c : o \multimap o$ for $c \in \Sigma$)

Claim

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

Proof: syntactic analysis of normal forms

Compiling into higher-order transducers

Transductions definable in linear λ-calculus can be turned into automata over a category $\mathcal{L}$ of purely linear λ-terms (w/ const $f_c : o \multimap o$ for $c \in \Sigma$)

Claim

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

Proof: syntactic analysis of normal forms

Compiling into higher-order transducers

Transductions definable in linear λ-calculus can be turned into automata over a category $\mathcal{L}$ of purely linear λ-terms (w/ const $f_c : o \multimap o$ for $c \in \Sigma$)

Claim

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

Proof: syntactic analysis of normal forms

Proof strategy for linear λ-definable $\implies$ regular function

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

Useful fact: there is a canonical functor from $\mathcal{L}$ to any *symmetric monoidal closed category* with (co)products

Unfortunately $\mathcal{R}_\oplus$ is **not** monoidal closed...

Toward a monoidal closed category

So far, we encountered:

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

Now consider:

- the free finite *product* completion: $\mathcal{C} \mapsto \mathcal{C}_\& = ((\mathcal{C}^{\mathrm{op}})_\oplus)^{\mathrm{op}}$

  Objects: formal products $\&_x C_x$

- the composite completion $\mathcal{C} \mapsto \mathcal{C}_\& \mapsto (\mathcal{C}_\&)_\oplus$

  Objects: formal sums of products $\bigoplus_u \&_x C_{u,x}$

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

Goals toward our main theorem

- Structure: $(\mathcal{R}_\&)_\oplus$ has finite products and is monoidal closed
- Conservativity: $(\mathcal{R}_\&)_\oplus$-automata and $\mathcal{R}_\oplus$-automata are equivalent

Structure (1): generic remarks $(\mathcal{C}_{\&})_{\oplus}$

Tensorial products can be lifted to the completions

- The new tensorial products satisfy the additional laws

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

- In particular, $(\mathcal{C}_{\&})_{\oplus}$ has distributive cartesian products

$$A \mathbin{\&} (B \oplus C) \equiv (A \mathbin{\&} B) \oplus (A \mathbin{\&} C)$$

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

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

- In particular, $(\mathcal{C}_{\&})_{\oplus}$ has distributive cartesian products

$$A \mathbin{\&} (B \oplus C) \equiv (A \mathbin{\&} B) \oplus (A \mathbin{\&} C)$$

When embedded in (co)presheafs $\cong$ Day convolution

Lemma

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.

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

Structure (2): combinatorics on strings

Lemma

$\mathcal{R}_\oplus$ has the internal homs $A \multimap B$ for all $A, B \in \mathcal{R}$.

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

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

copyless SST $\implies$ finitely many shapes: use as states; registers for parameters

Structure (2): combinatorics on strings

Lemma

$\mathcal{R}_\oplus$ has the internal homs $A \multimap B$ for all $A, B \in \mathcal{R}$.

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

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

copyless SST $\implies$ finitely many shapes: use as states; registers for parameters

Conclusion

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

Conservativity

Lemma

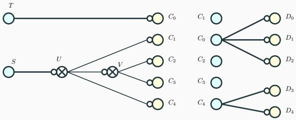
$(\mathcal{C}_\&)_\oplus$ automata are equivalent to non-deterministic $\mathcal{C}_\oplus$ automata.

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

Conservativity

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

- Uniformization already known [Alur & Deshmuk 2011]
- Argument implicitly based on monoidal closure!



Theorem

For any monoidal category $\mathcal{C}$, if $\mathcal{C}_\oplus$ has all the internal homsets $A \multimap B$ for $A, B \in \mathcal{C}$, then $(\mathcal{C}_\&)_\oplus$-automata and $\mathcal{C}_\oplus$-automata are equivalent.

equivalently: ND $\mathcal{C}_\oplus$-automata can be uniformized

Characterization of string-to-string regular functions

Just discussed:

Today's main theorem [Nguyễn & P.]

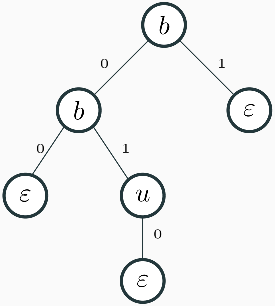
regular string function $\iff$ definable by some $t : \mathsf{Str}_\Gamma[A] \multimap \mathsf{Str}_\Sigma$ in ILL with A purely linear

Some thoughts:

- Non-trivial technical arguments, for good reasons — λ-terms compose easily
- More conceptual POV on the uniformization argument? Is $\multimap$ overkill?
- The category of register: an **affine clone** of the PROP of the string
  $\to$ linear clone + $\oplus\&$ $\Rightarrow$ smcc

Regular tree-to-tree functions

Over ranked alphabets such as e.g. $\Sigma = \{b : 2, u : 1, \varepsilon : 0\}$



$$b(b(\varepsilon, u(\varepsilon)), \varepsilon)$$

$$\lambda b\, u\, \varepsilon.\, b\, (b\, \varepsilon\, (u\, \varepsilon))\, \varepsilon : \mathrm{Tree}_\Sigma$$

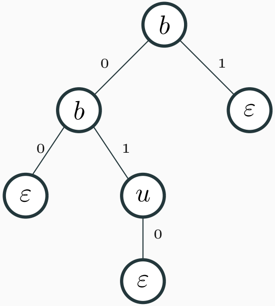
$$\mathrm{Tree}_\Sigma = (o \multimap o \multimap o) \to (o \multimap o) \to o \to o$$

Main theorem for trees [Nguyễn & P.]

regular *tree* function $\iff$ definable by some $t : \mathsf{Tree}_\Gamma[A] \multimap \mathsf{Tree}_\Sigma$ in ILL with A purely linear

Regular tree-to-tree functions

Over ranked alphabets such as e.g. $\Sigma = \{b : 2, u : 1, \varepsilon : 0\}$



$$b(b(\varepsilon, u(\varepsilon)), \varepsilon)$$

$$\lambda b\, u\, \varepsilon.\, b\, (b\, \varepsilon\, (u\, \varepsilon))\, \varepsilon : \mathrm{Tree}_\Sigma$$

$$\mathrm{Tree}_\Sigma = (o \multimap o \multimap o) \to (o \multimap o) \to o \to o$$

Main theorem for trees [Nguyễn & P.]

regular *tree* function $\iff$ definable by some $t : \mathrm{Tree}_\Gamma[A] \multimap \mathrm{Tree}_\Sigma$ in ILL with A purely linear

Important: additive connectives need to be included!

Why are we including additive connectives?

Additives are required for trees

Copyless streaming *tree* transducers $\subset$ regular *tree* functions; conjectured to be *strict*.

To recover an equality: ad-hoc relaxation called “single use restriction”.

Why are we including additive connectives?

Additives are required for trees

Copyless streaming *tree* transducers $\subset$ regular *tree* functions; conjectured to be *strict*.
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!

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

Why are we including additive connectives?

Additives are required for trees

Copyless streaming *tree* transducers $\subset$ regular *tree* functions; conjectured to be *strict*.
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!

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

Categorical tree transducers

- Streaming settings: now **symmetric monoidal** categories, no $\top$, bottom-up processing
- $\mathcal{R}$ is built using the same conceptual recipe as for strings:

Affine clone of the free PROP generated by the output alphabet

The register category with output alphabet Σ: details

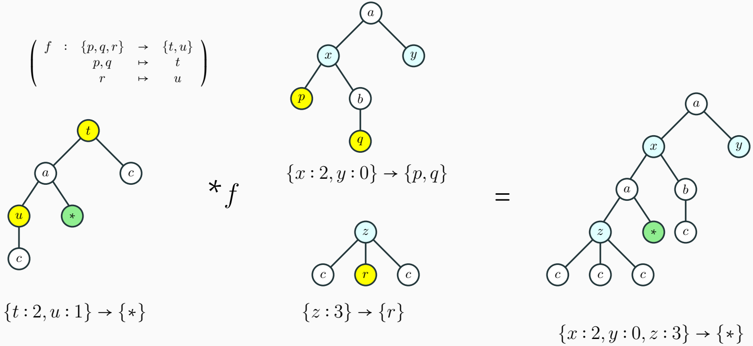
- **Objects:** finite ranked alphabet R, S
- **Morphisms:** $\mathrm{Hom}_{\mathcal{R}}(R, S) =$ copyless transitions, registers contain trees with leaf holes, constructors of Σ usable

Example: a morphism $\{r : 0, s : 2\}$ to $\{r' : 1\}$ with $\Sigma = \{b : 2, \varepsilon : 0\}$

- a λ-term of type $\underbrace{(o \multimap o \multimap o)}_{b} \to \underbrace{o}_{\varepsilon} \to \underbrace{o}_{r} \multimap \underbrace{(o \multimap o \multimap o)}_{s} \multimap \underbrace{o \multimap o}_{r'}$
- A tree with order 2 holes over the ranked alphabet r, s, r', b, ε subject to affineness constraints on r, s and the input of r'.

Composition of the corresponding multicategory in action

- Easier to describe as a multicategory, i.e. a variant of categories with multiple inputs and a single output



- Moving to a category $\mathcal{R}$ = freely complete by saying that each morphisms partition the output according to the output

To bottom-up ranked tree transducers

Easy observation

$\mathcal{R}_{\oplus\&} \simeq$ macro tree transducers $\Rightarrow$ characterizes regular functions

Mostly bureaucratic details: multi-hole registers vs single-hole, single-use restriction vs $\&$.

To bottom-up ranked tree transducers

Easy observation

$\mathcal{R}_{\oplus\&} \simeq$ macro tree transducers $\Rightarrow$ characterizes regular functions

Mostly bureaucratic details: multi-hole registers vs single-hole, single-use restriction vs $\&$.

What's left: how to interpret $\multimap$?

To bottom-up ranked tree transducers

Easy observation

$\mathcal{R}_{\oplus\&} \simeq$ macro tree transducers $\Rightarrow$ characterizes regular functions

Mostly bureaucratic details: multi-hole registers vs single-hole, single-use restriction vs $\&$.

What's left: how to interpret $\multimap$? $\rightsquigarrow$ same reasoning as for strings:

Lemma

$\mathcal{R}_{\oplus}$ has the internal homs $A \multimap B$ for all $A, B \in \mathcal{R}$.

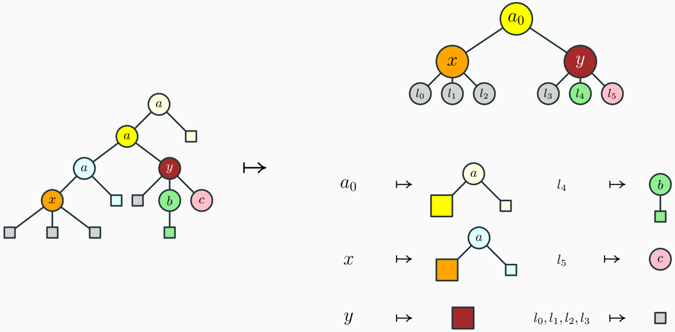
Lemma

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.

Internal homs in $\mathcal{R}_{\oplus}$ in pictures

Lemma

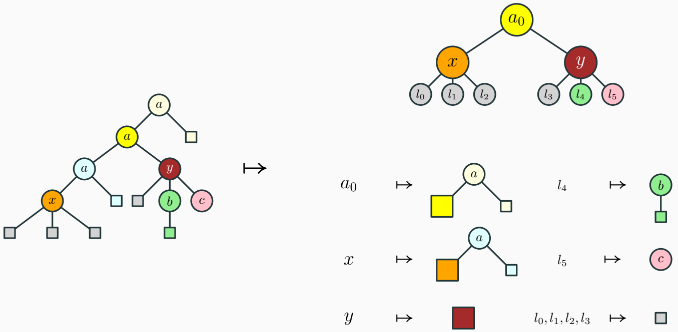
$\mathcal{R}_{\oplus}$ has the internal homs $A \multimap B$ for all $A, B \in \mathcal{R}$.



Internal homs in $\mathcal{R}_\oplus$ in pictures

Lemma

$\mathcal{R}_\oplus$ has the internal homs $A \multimap B$ for all $A, B \in \mathcal{R}$.



Curiosity question: does it appear already in the literature on operads as a special case of something?

Conclusion

What happened here:

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

Some take-aways:

- Important ingredient in uniformization: monoidal closure

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

Conclusion

What happened here:

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

Some take-aways:

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

Conclusion

What happened here:

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

Some take-aways:

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

Conclusion

Broader picture

$\mathsf{Str}_\Sigma[A] \multimap \mathsf{Bool}$ with A linear (adapted as needed):

λ-calculus	languages	status
simply typed	regular	✓ [Hillebrand & Kanellakis 1996]
linear or affine	regular	✓
non-commutative linear or affine	star-free	✓

$\mathsf{Str}_\Gamma[A] \multimap \mathsf{Str}_\Sigma$ with A affine (adapted as needed):

λ-calculus	transducers	status
linear (without additives)	weird (?)	✓ (?)
affine	regular functions	✓
non-commutative affine	first-order regular fn.	✓?
linear/affine with additives	regular functions	✓
parsimonious	polyregular	??
simply typed	variant of CPDA???	???

Conclusion

Broader picture

$\mathsf{Str}_\Sigma[A] \multimap \mathsf{Bool}$ with A linear (adapted as needed):

λ-calculus	languages	status
simply typed	regular	✓ [Hillebrand & Kanellakis 1996]
linear or affine	regular	✓
non-commutative linear or affine	star-free	✓

$\mathsf{Str}_\Gamma[A] \multimap \mathsf{Str}_\Sigma$ with A affine (adapted as needed):

λ-calculus	transducers	status
linear (without additives)	weird (?)	✓ (?)
affine	regular functions	✓
non-commutative affine	first-order regular fn.	✓ ?
linear/affine with additives	regular functions	✓
parsimonious	polyregular	??
simply typed	variant of CPDA???	???

+ a characterization of $\mathsf{Str}[A] \to \mathsf{Str}$ as *comparison-free* polyregular functions

Conclusion

Broader picture

$\mathsf{Str}_\Sigma[A] \multimap \mathsf{Bool}$ with A linear (adapted as needed):

λ-calculus	languages	status
simply typed	regular	✓ [Hillebrand & Kanellakis 1996]
linear or affine	regular	✓
non-commutative linear or affine	star-free	✓

$\mathsf{Str}_\Gamma[A] \multimap \mathsf{Str}_\Sigma$ with A affine (adapted as needed):

λ-calculus	transducers	status
linear (without additives)	weird (?)	✓ (?)
affine	regular functions	✓
non-commutative affine	first-order regular fn.	✓?
linear/affine with additives	regular functions	✓
parsimonious	polyregular	??
simply typed	variant of CPDA???	???

+ a characterization of $\mathsf{Str}[A] \to \mathsf{Str}$ as *comparison-free* polyregular functions

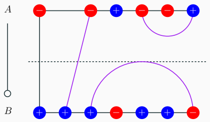
Thanks for listening! Questions?

Dropping the additives for string-to-string functions

String functions without additive

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

⇝ Interpretation as bidirectional automata (w/o registers)



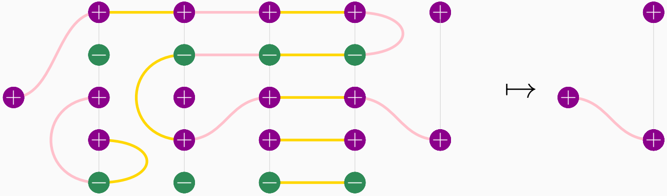
Planar diagrams
⇝
FO fragments

Dropping the additives

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

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

Define regular languages



Consequence (not interesting)

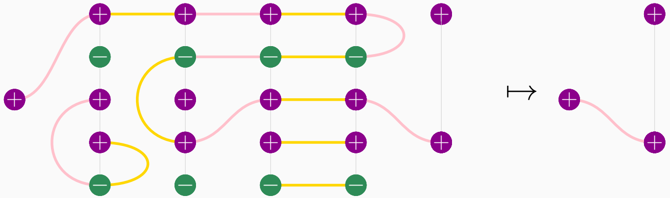
Every linear term $t : \mathsf{Str}_\Sigma[A] \multimap \mathsf{Bool}$ with A $\to$-free defines a regular language.

Dropping the additives and commutativity

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

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

Define **star-free** languages



Consequence [Nguyễn, P. 2020]

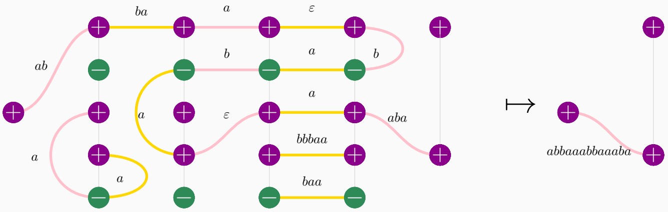
Every **planar** linear term $t : \mathsf{Str}_\Sigma[A] \multimap \mathsf{Str}$ with A $\to$-free defines a star-free language.

Dropping the additives and commutativity

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

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

Define **first-order** regular functions



Consequence

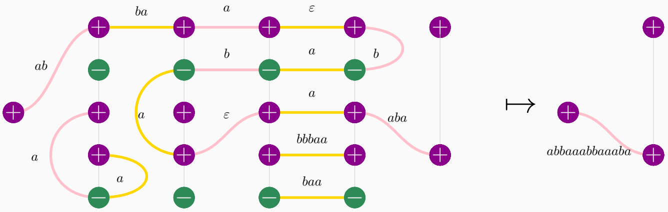
Every **planar** linear term $t : \mathsf{Str}_\Sigma[A] \multimap \mathsf{Str}$ with A $\to$-free defines a FO-transduction.

Dropping the additives and commutativity

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

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

Define **first-order** regular functions



Consequence

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

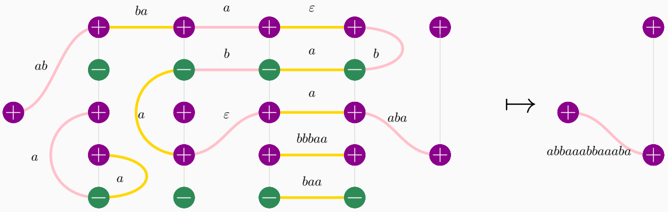
Alas, planar linear terms are much weaker than FO-transductions (preserve Parikh images)

Dropping the additives and commutativity

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

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

Define **first-order** regular functions



Conjecture

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

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

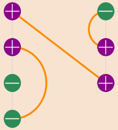
A category of planar diagrams

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

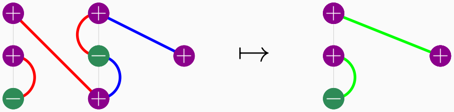
(well-known model of linear logic)

A compact closed category of planar diagrams

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



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

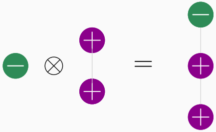


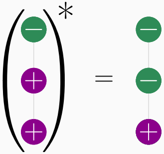
Compact-closure and interpretation of the λ-calculus

Structure to interpret the linear λ-calculus

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

(injective interpretation of booleans)

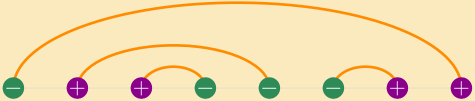




Examples

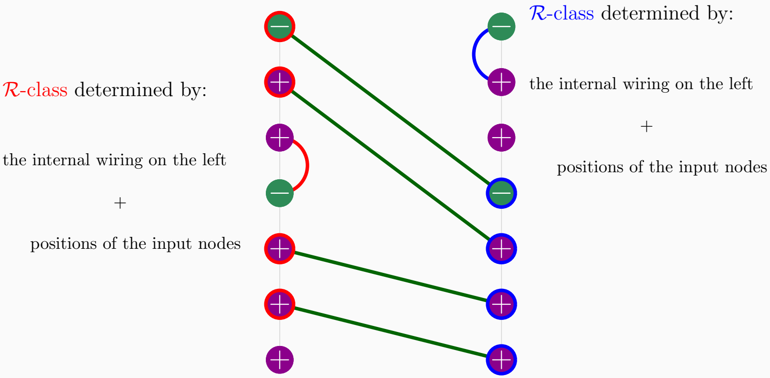
$$[\![((o \multimap o) \multimap o \multimap o) \multimap ((o \multimap o) \multimap o) \multimap o]\!] \;=\; - + + - - - + +$$

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



Aperiodicity

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

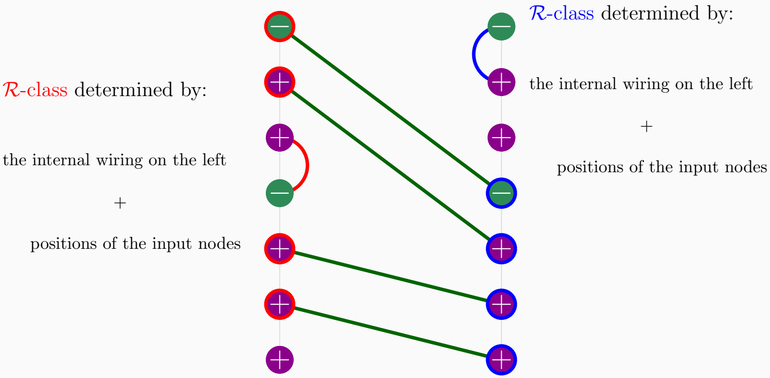


Therefore: planar $\implies$ $\mathcal{H}$-trivial

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

Aperiodicity

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

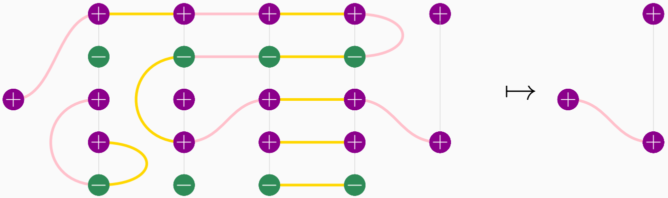


Therefore: planar $\implies$ $\mathcal{H}$-trivial

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

Diagrams and two-way automata

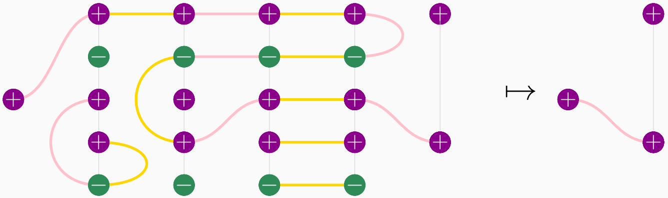
Non-planar diagrams (with crossings): reminiscent of runs in 2DFAs!



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

Diagrams and two-way automata

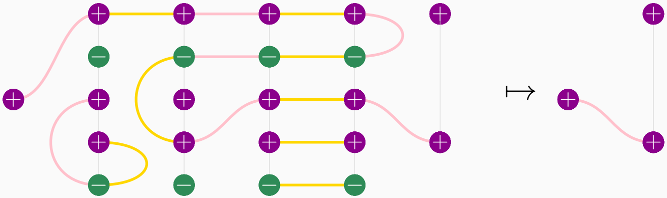
Non-planar diagrams (with crossings): reminiscent of runs in 2DFAs!



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

Diagrams and two-way automata

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



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

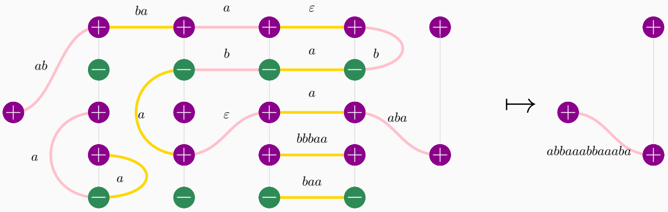
Theorem

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

More generally: first-order transductions

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

(labels of compositions given by concatenation)



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

Theorem

First-order transduction (FO regular functions) = reversible planar 2DFTs.

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

(hence reversible planar 2DFTs ⊆ FO-transductions)

- FO transduction ⊆ reversible planar 2DFTs: compose + Krohn—Rhodes