# Higher-order model checking meets implicit automata

(work in progress)

Lê Thành Dũng (Tito) Nguyễn — nltd@nguyentito.eu — Aix-Marseille Univ. (new!)
with Abhishek De (Birmingham), Charles Grellois (Sheffield) & Cécilia Pradic (Swansea)

Journées du GT Scalp, mardi 19 novembre 2024

# Model checking (disclaimer: I don't know anything about it)

A basic setting:

- represent possible system behaviors for time $t \to +\infty$ as a language $L \subseteq \Sigma^\omega$
  typically, $L = \mathscr{L}(\mathscr{A})$ for some automaton $\mathscr{A}$

## Model checking (disclaimer: I don't know anything about it)

A basic setting:

- represent possible system behaviors for time $t \to +\infty$ as a language $L \subseteq \Sigma^\omega$
  typically, $L = \mathscr{L}(\mathscr{A})$ for some automaton $\mathscr{A}$
- represent "nice" wanted behaviors as $\mathscr{L}(\varphi)$ for some specification $\varphi$
  typically, $\varphi$ is a formula in some temporal *logic*
  - "no bad things happen in finite time"
  - "every `open` is followed by a `close`"
  - ...

# Model checking (disclaimer: I don't know anything about it)

A basic setting:

- represent possible system behaviors for time $t \to +\infty$ as a language $L \subseteq \Sigma^\omega$
  typically, $L = \mathscr{L}(\mathscr{A})$ for some automaton $\mathscr{A}$
- represent "nice" wanted behaviors as $\mathscr{L}(\varphi)$ for some specification $\varphi$
  typically, $\varphi$ is a formula in some temporal *logic*
  - "no bad things happen in finite time"
  - "every `open` is followed by a `close`"
  - ...

**The model-checking problem: "are all possible behaviors nice?"**

$$L \overset{?}{\subseteq} \mathscr{L}(\varphi)$$

# A variant of model checking

**System behavior = just one infinite tree $T$**

Branching may be used for conditionals (if/then/else),
uninterpreted function symbols with multiple arguments, ...

# A variant of model checking

**System behavior = just one infinite tree $T$**

Branching may be used for conditionals (if/then/else),
uninterpreted function symbols with multiple arguments, ...

**Wanted behaviors = formula $\varphi$ in monadic second-order (MSO) logic**

MSO is a canonical choice from theoretical POV:

- more expressive than temporal logics
- corresponds to regular languages on finite words
- on infinite trees: corresponds to e.g. alternating parity automata

# A variant of model checking

**System behavior = just one infinite tree $T$**

Branching may be used for conditionals (if/then/else), uninterpreted function symbols with multiple arguments, ...

**Wanted behaviors = formula $\varphi$ in monadic second-order (MSO) logic**

MSO is a canonical choice from theoretical POV:

- more expressive than temporal logics
- corresponds to regular languages on finite words
- on infinite trees: corresponds to e.g. alternating parity automata

**Decision problem:** $T \in \mathscr{L}(\varphi)$ ? i.e. $T \vDash \varphi$ ?

# A variant of model checking

**System behavior = just one infinite tree $T$**

Branching may be used for conditionals (if/then/else),
uninterpreted function symbols with multiple arguments, ...

**Wanted behaviors = formula $\varphi$ in monadic second-order (MSO) logic**

MSO is a canonical choice from theoretical POV:

- more expressive than temporal logics
- corresponds to regular languages on finite words
- on infinite trees: corresponds to e.g. alternating parity automata

**Decision problem:** $T \in \mathscr{L}(\varphi)$ ? i.e. $T \vDash \varphi$ ?

$\rightsquigarrow$ system representation = finite description of tree with *decidable MSO theory*

# Which infinite trees have a decidable MSO theory?

- Some open problems, for instance automatic structures
- A big class of trees equivalently described by
  - higher-order pushdown automata (related to Muchnik iteration)
  - tree→graph transductions + graph→tree unfolding: "Caucal hierarchy" (2002)
  - *safe higher-order recursion schemes* (HORS) [Knapkik, Niwiński & Urzyczyn '02]

## Which infinite trees have a decidable MSO theory?

- Some open problems, for instance automatic structures
- A big class of trees equivalently described by
  - higher-order pushdown automata (related to Muchnik iteration)
  - tree→graph transductions + graph→tree unfolding: "Caucal hierarchy" (2002)
  - *safe higher-order recursion schemes* (HORS) [Knapkik, Niwiński & Urzyczyn '02]
- Natural extension [Ong 2006, reproved many times]:
  unsafe HORS = *simply typed $\lambda$-calculus* with *recursive* definitions

  $\rightsquigarrow$ verification of functional programs! (tested on subsets of OCaml)

# Which infinite trees have a decidable MSO theory?

- Some open problems, for instance automatic structures
- A big class of trees equivalently described by
  - higher-order pushdown automata (related to Muchnik iteration)
  - tree→graph transductions + graph→tree unfolding: "Caucal hierarchy" (2002)
  - *safe higher-order recursion schemes* (HORS) [Knapkik, Niwiński & Urzyczyn '02]
- Natural extension [Ong 2006, reproved many times]:
  unsafe HORS = *simply typed $\lambda$-calculus* with *recursive* definitions
  $\rightsquigarrow$ verification of functional programs! (tested on subsets of OCaml)

(in practice, higher-order fixpoint logic seems to work better for model-checking functional programs: cf. recent work of Kobayashi et al.)

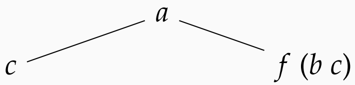
# Simply typed $\lambda$-terms with recursion generating infinite trees

$$\texttt{let rec } f = \lambda x.\, a\, x\, (f\, (b\, x))\ \texttt{in } f\, c$$

$$f\ c$$

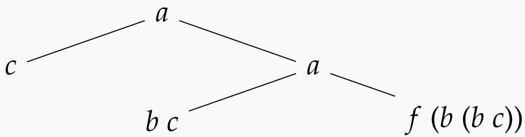
# Simply typed $\lambda$-terms with recursion generating infinite trees

$$\texttt{let rec}\ f = \lambda x.\, a\, x\, (f\, (b\, x))\ \texttt{in}\ f\, c$$



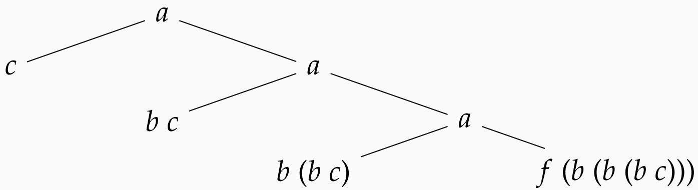
# Simply typed $\lambda$-terms with recursion generating infinite trees

$\texttt{let rec}\ f = \lambda x.\, a\, x\, (f\, (b\, x))\ \texttt{in}\ f\, c$



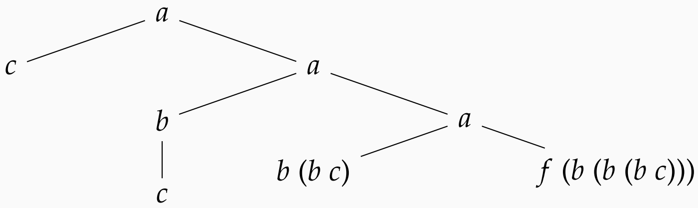
# Simply typed $\lambda$-terms with recursion generating infinite trees

$\texttt{let rec}\ f = \lambda x.\, a\, x\, (f\, (b\, x))\ \texttt{in}\ f\, c$



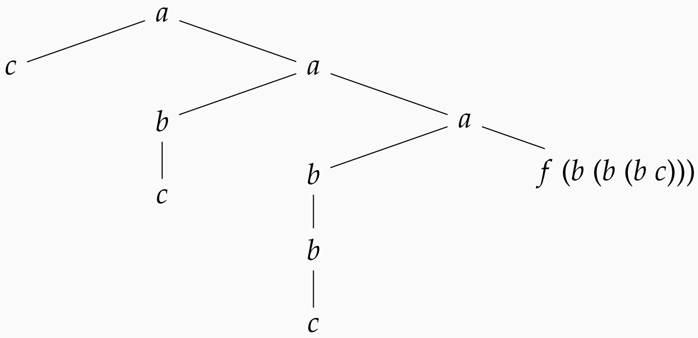
# Simply typed $\lambda$-terms with recursion generating infinite trees

$$\texttt{let rec}\ f = \lambda x.\, a\, x\, (f\, (b\, x))\ \texttt{in}\ f\, c$$



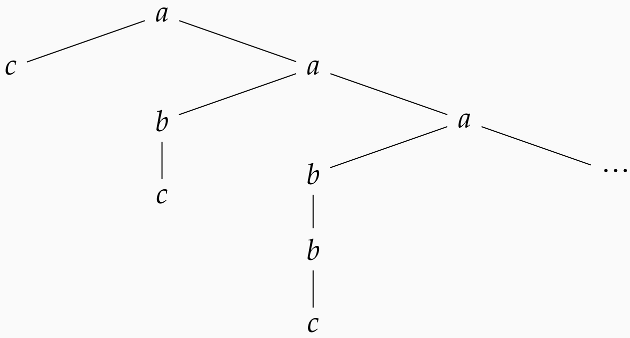
# Simply typed $\lambda$-terms with recursion generating infinite trees

let rec $f = \lambda x.\, a\, x\, (f\, (b\, x))$ in $f\, c$



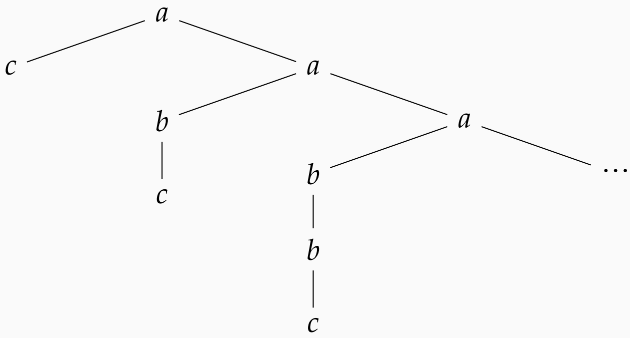
# Simply typed $\lambda$-terms with recursion generating infinite trees

$\texttt{let rec}\ f = \lambda x.\, a\, x\, (f\, (b\, x))\ \texttt{in}\ f\, c$



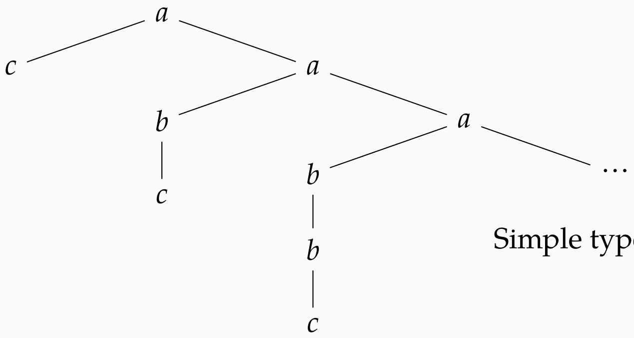
# Simply typed $\lambda$-terms with recursion generating infinite trees

$$a : o \to o \to o,\ b : o \to o,\ c : o \vdash \ \texttt{let rec}\ f = \lambda x.\, a\, x\, (f\, (b\, x))\ \texttt{in}\ f\, c\ : o$$



# Simply typed $\lambda$-terms with recursion generating infinite trees

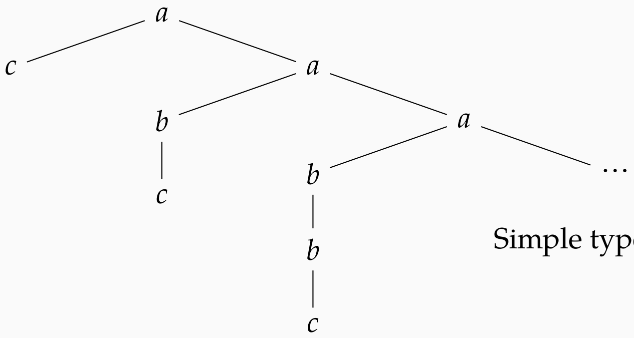
$$a : o \to o \to o,\ b : o \to o,\ c : o \vdash \ \texttt{let rec}\ f = \lambda x.\, a\, x\, (f\, (b\, x))\ \texttt{in}\ f\, c \ : o$$




Simple types: $\sigma, \tau ::= o \mid \sigma \to \tau$

# Simply typed $\lambda$-terms with recursion generating infinite trees

$$a : o \to o \to o,\ b : o \to o,\ c : o \vdash \ \texttt{let rec}\ f = \lambda x.\, a\, x\, (f\,(b\,x))\ \texttt{in}\ f\, c \ : o$$

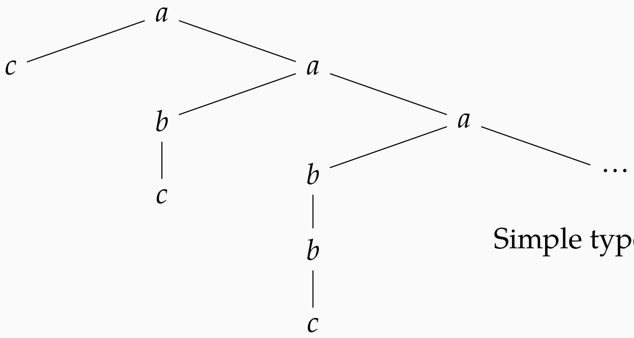



Simple types: $\sigma, \tau ::= o \mid \sigma \to \tau$

Decidability of MSO logic over such trees: flagship success of *denotational semantics*

# Simply typed $\lambda$-terms with recursion generating infinite trees

$$a : o \to o \to o,\ b : o \to o,\ c : o \vdash \texttt{let rec } f = \lambda x.\, a\, x\, (f\, (b\, x))\ \texttt{in } f\, c\ : o$$




Simple types: $\sigma, \tau ::= o \mid \sigma \to \tau$

Decidability of MSO logic over such trees: flagship success of *denotational semantics*

idea best illustrated by a theorem on finite words, w/o recursion: next slide

# Defining languages in the simply typed $\lambda$-calculus

## Church encodings of binary strings [Böhm & Berarducci 1985]

$\simeq$ `fold_right` on a list of characters (generalizable to any alphabet; $\mathsf{Nat} = \mathsf{Str}_{\{1\}}$):

$$\overline{\texttt{011}} = \lambda f_0.\, \lambda f_1.\, \lambda x.\, f_0\ (f_1\ (f_1\ x)) : \mathsf{Str}_{\{\texttt{0},\texttt{1}\}}[\tau] = (\tau \to \tau) \to (\tau \to \tau) \to \tau \to \tau$$

# Defining languages in the simply typed $\lambda$-calculus

## Church encodings of binary strings [Böhm & Berarducci 1985]

$\simeq$ `fold_right` on a list of characters (generalizable to any alphabet; $\mathsf{Nat} = \mathsf{Str}_{\{1\}}$):

$$\overline{\texttt{011}} = \lambda f_0.\ \lambda f_1.\ \lambda x.\ f_0\ (f_1\ (f_1\ x)) : \mathsf{Str}_{\{\texttt{0},\texttt{1}\}}[\tau] = (\tau \to \tau) \to (\tau \to \tau) \to \tau \to \tau$$

Simply typed $\lambda$-terms $t : \mathsf{Str}_{\{\texttt{0},\texttt{1}\}}[\tau] \to \mathsf{Bool}$ define **languages** $L \subseteq \{\texttt{0},\texttt{1}\}^*$

# Defining languages in the simply typed $\lambda$-calculus

## Church encodings of binary strings [Böhm & Berarducci 1985]

$\simeq$ `fold_right` on a list of characters (generalizable to any alphabet; $\mathsf{Nat} = \mathsf{Str}_{\{1\}}$):

$$\overline{\mathtt{011}} = \lambda f_0.\, \lambda f_1.\, \lambda x.\, f_0\ (f_1\ (f_1\ x)) : \mathsf{Str}_{\{\mathtt{0},\mathtt{1}\}}[\tau] = (\tau \to \tau) \to (\tau \to \tau) \to \tau \to \tau$$

Simply typed $\lambda$-terms $t : \mathsf{Str}_{\{\mathtt{0},\mathtt{1}\}}[\tau] \to \mathsf{Bool}$ define **languages** $L \subseteq \{\mathtt{0},\mathtt{1}\}^*$

Example: $t = \lambda s.\, s\ \mathtt{id}\ \mathtt{not}\ \mathtt{true} : \mathsf{Str}_{\{\mathtt{0},\mathtt{1}\}}[\mathsf{Bool}] \to \mathsf{Bool}$ (even number of 1s)

$$t\ \overline{\mathtt{011}} \longrightarrow_\beta \overline{\mathtt{011}}\ \mathtt{id}\ \mathtt{not}\ \mathtt{true} \longrightarrow_\beta \mathtt{id}\ (\mathtt{not}\ (\mathtt{not}\ \mathtt{true})) \longrightarrow_\beta \mathtt{true}$$

# Defining languages in the simply typed $\lambda$-calculus

## Church encodings of binary strings [Böhm & Berarducci 1985]

$\simeq$ `fold_right` on a list of characters (generalizable to any alphabet; $\mathsf{Nat} = \mathsf{Str}_{\{1\}}$):

$$\overline{\mathtt{011}} = \lambda f_0.\ \lambda f_1.\ \lambda x.\ f_0\ (f_1\ (f_1\ x)) : \mathsf{Str}_{\{0,1\}}[\tau] = (\tau \to \tau) \to (\tau \to \tau) \to \tau \to \tau$$

Simply typed $\lambda$-terms $t : \mathsf{Str}_{\{0,1\}}[\tau] \to \mathsf{Bool}$ define **languages** $L \subseteq \{\mathtt{0},\mathtt{1}\}^*$

Example: $t = \lambda s.\ s\ \mathtt{id}\ \mathtt{not}\ \mathtt{true} : \mathsf{Str}_{\{0,1\}}[\mathsf{Bool}] \to \mathsf{Bool}$ (even number of 1s)

$$t\ \overline{\mathtt{011}} \longrightarrow_\beta \overline{\mathtt{011}}\ \mathtt{id}\ \mathtt{not}\ \mathtt{true} \longrightarrow_\beta \mathtt{id}\ (\mathtt{not}\ (\mathtt{not}\ \mathtt{true})) \longrightarrow_\beta \mathtt{true}$$

## Theorem (Hillebrand & Kanellakis 1996)

*All* regular (i.e. MSO-definable) languages, *and only those, can be defined this way.*

# Proof of STLC-definable $\implies$ regular, by semantic evaluation

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

*For any type $A$ and any simply typed $\lambda$-term $t : \mathsf{Str}_\Sigma[\tau] \to \mathsf{Bool}$, the language $\mathscr{L}(t) = \{w \in \Sigma^* \mid t\,\overline{w} \to_\beta^* \mathtt{true}\}$ is regular.*

**Part 1 of proof.**

Fix a type $\tau$. Any *denotational semantics* $[\![-]\!]$ quotients words:

$$w \in \Sigma^* \rightsquigarrow \overline{w} : \mathsf{Str}[\tau] \rightsquigarrow [\![\overline{w}]\!]_{\mathsf{Str}_\Sigma[\tau]} \in [\![\mathsf{Str}_\Sigma[\tau]]\!]$$

$[\![\overline{w}]\!]_{\mathsf{Str}_\Sigma[\tau]}$ determines behavior of $w$ w.r.t. all $\mathsf{Str}_\Sigma[\tau] \to \mathsf{Bool}$ terms:

$$w \in \mathscr{L}(t) \iff t\,\overline{w} \to_\beta^* \mathtt{true} \underbrace{\iff}_{\text{assuming } [\![\mathtt{true}]\!] \neq [\![\mathtt{false}]\!]} [\![t\,\overline{w}]\!] = [\![t]\!]([\![\overline{w}]\!]) = [\![\mathtt{true}]\!]$$

Goal: to decide $\mathscr{L}(t)$, compute $w \mapsto [\![\overline{w}]\!]$ in some denotational model.

# Proof of STLC-definable $\implies$ regular, by semantic evaluation

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

*For any type $A$ and any simply typed $\lambda$-term $t : \mathsf{Str}_\Sigma[\tau] \to \mathsf{Bool}$, the language $\mathscr{L}(t) = \{w \in \Sigma^* \mid t\,\overline{w} \to_\beta^* \mathtt{true}\}$ is regular.*

**Part 2 of proof.**

We use $[\![-]\!] : \mathrm{STLC} \to \mathsf{FinSet}$ to build a DFA with states $Q = [\![\mathsf{Str}_\Sigma[\tau]]\!]$, acceptation as $[\![t]\!](-) = [\![\mathtt{true}]\!]$.

$$\longrightarrow [\![\overline{\varepsilon}]\!] \xrightarrow{\mathsf{a}} [\![\overline{\mathsf{a}}]\!] \xrightarrow{\mathsf{b}} [\![\overline{\mathsf{ab}}]\!] \xrightarrow{\mathsf{b}} [\![\overline{\mathsf{abb}}]\!] \longrightarrow \cdots$$

$$w \in \mathscr{L}(t) \iff [\![t]\!]\left([\![\overline{w}]\!]_{\mathsf{Str}_\Sigma[\tau]}\right) = [\![\mathtt{true}]\!] \iff w \text{ accepted.} \qquad \square$$

# Proof of STLC-definable $\implies$ regular, by semantic evaluation

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

*For any type $A$ and any simply typed $\lambda$-term $t : \mathsf{Str}_\Sigma[\tau] \to \mathsf{Bool}$, the language $\mathscr{L}(t) = \{w \in \Sigma^* \mid t\,\overline{w} \to_\beta^* \mathtt{true}\}$ is regular.*

**Part 2 of proof.**

We use $[\![-]\!] : \mathrm{STLC} \to \mathsf{FinSet}$ to build a DFA with states $Q = [\![\mathsf{Str}_\Sigma[\tau]]\!]$, acceptation as $[\![t]\!](-) = [\![\mathtt{true}]\!]$. $\quad$ ($|Q| < \infty$, e.g. $2^{2^{134}}$ when $\tau = \mathsf{Bool}$ & $|\,[\![o]\!]\,| = 2 = |\Sigma|$)

$$\longrightarrow [\![\overline{\varepsilon}]\!] \xrightarrow{\mathsf{a}} [\![\overline{\mathsf{a}}]\!] \xrightarrow{\mathsf{b}} [\![\overline{\mathsf{ab}}]\!] \xrightarrow{\mathsf{b}} [\![\overline{\mathsf{abb}}]\!] \longrightarrow \cdots$$

$$w \in \mathscr{L}(t) \iff [\![t]\!]\left([\![\overline{w}]\!]_{\mathsf{Str}_\Sigma[\tau]}\right) = [\![\mathtt{true}]\!] \iff w \text{ accepted.} \qquad \square$$

# Takeaways from the Hillebrand–Kanellakis theorem

## From the theorem statement

analogous to implicit complexity (e.g. “Light Linear Logic captures $\mathsf{P}$”)
$\rightsquigarrow$ *implicit automata* research programme (N. & Pradic), for instance:
star-free languages in a non-commutative linear[1] $\lambda$-calculus (ICALP’20)

[1] Affine in the paper, strictly linear in my PhD dissertation.

# Takeaways from the Hillebrand–Kanellakis theorem

## From the theorem statement

analogous to implicit complexity (e.g. "Light Linear Logic captures $\mathsf{P}$")

$\rightsquigarrow$ *implicit automata* research programme (N. & Pradic), for instance:

star-free languages in a non-commutative linear[1] $\lambda$-calculus (ICALP'20)

## From the proof: connection with finite automata via finitary semantics

$\rightsquigarrow$ for MSO on $\infty$ trees: alternating parity automata $\leftrightarrow$ bespoke semantics

$\longrightarrow$ decidability proofs for trees generated by ST$\lambda$C+recursion

[Kobayashi & Ong, Salvati & Walukiewicz, Grellois & Melliès, ...]

[1] Affine in the paper, strictly linear in my PhD dissertation.

# Takeaways from the Hillebrand–Kanellakis theorem

**From the theorem statement**

analogous to implicit complexity (e.g. "Light Linear Logic captures $\mathsf{P}$")
$\rightsquigarrow$ *implicit automata* research programme (N. & Pradic), for instance:
star-free languages in a non-commutative linear[1] $\lambda$-calculus (ICALP'20)

**From the proof: connection with finite automata via finitary semantics**

$\rightsquigarrow$ for MSO on $\infty$ trees: alternating parity automata $\leftrightarrow$ bespoke semantics
$\longrightarrow$ decidability proofs for trees generated by ST$\lambda$C+recursion
[Kobayashi & Ong, Salvati & Walukiewicz, Grellois & Melliès, ...]

Combining the two: *implicit automata over infinite words/trees*

[1] Affine in the paper, strictly linear in my PhD dissertation.

# Our main goal

**Conjecture**

*If $f : \Sigma^\omega \to \Gamma^\omega$ is computed by some* infinitary *simply typed $\lambda$-term $t : \mathsf{Str}_\Sigma[\tau] \to \mathsf{Str}_\Gamma$, then for every $\omega$-regular $L \subseteq \Gamma^\omega$, the preimage $f^{-1}(L)$ is also $\omega$-regular.*

Should also hold on infinite trees…

# Our main goal

**Conjecture**

*If $f : \Sigma^\omega \to \Gamma^\omega$ is computed by some* infinitary *simply typed $\lambda$-term $t : \mathsf{Str}_\Sigma[\tau] \to \mathsf{Str}_\Gamma$, then for every $\omega$-regular $L \subseteq \Gamma^\omega$, the preimage $f^{-1}(L)$ is also $\omega$-regular.*

Should also hold on infinite trees...

**Corollary (assuming the conjecture is true)**

*$L$ is $\omega$-regular $\iff$ $L = f^{-1}$(parity language) for such an $f$.*

$\to$ because deterministic parity automata recognize all $\omega$-regular languages

(fails for infinite trees!)

# Our main goal

**Conjecture**

*If $f : \Sigma^\omega \to \Gamma^\omega$ is computed by some* infinitary *simply typed $\lambda$-term $t : \mathsf{Str}_\Sigma[\tau] \to \mathsf{Str}_\Gamma$, then for every $\omega$-regular $L \subseteq \Gamma^\omega$, the preimage $f^{-1}(L)$ is also $\omega$-regular.*

Should also hold on infinite trees…

**Corollary (assuming the conjecture is true)**

*$L$ is $\omega$-regular $\iff$ $L = f^{-1}$(parity language) for such an $f$.*

$\to$ because deterministic parity automata recognize all $\omega$-regular languages

(fails for infinite trees!)

Proof method for conjecture?

# Our main goal

**Conjecture**

*If $f : \Sigma^\omega \to \Gamma^\omega$ is computed by some* infinitary *simply typed $\lambda$-term $t : \mathsf{Str}_\Sigma[\tau] \to \mathsf{Str}_\Gamma$, then for every $\omega$-regular $L \subseteq \Gamma^\omega$, the preimage $f^{-1}(L)$ is also $\omega$-regular.*

Should also hold on infinite trees…

**Corollary (assuming the conjecture is true)**

*$L$ is $\omega$-regular $\iff$ $L = f^{-1}$(parity language) for such an $f$.*

$\to$ because deterministic parity automata recognize all $\omega$-regular languages

(fails for infinite trees!)

Proof method for conjecture?

Finitary semantics of infinitary ST$\lambda$C with parity conditions $\rightsquigarrow$ *colors*

# A colored denotational semantics

Cartesian closed categories (CCCs): semantics of simply typed $\lambda$-calculus

**Coloring modality for $k$ colors in a CCC $\mathscr{C}$ [Melliès 2017; Walukiewicz 2019]**

$\square A = \underbrace{A \times \cdots \times A}_{k+1 \text{ times}}$ + comultiplication $\square A \to \square\square A$ taking max of indices

$\square$ is a linear exponential comonad $\implies$ $\mathscr{C}_\square$ is also a CCC

- (Scott model of linear logic)$_\square$ with clever interpretation of recursion = Grellois and Melliès's (2015) colored semantics for deciding MSO
- interpretation in Scott$_\square$ = int. in Scott $\circ$ ("coloring translation" i.e. int. in $\Lambda_\square$) over ST$\lambda$C (w/o recursion), where $\Lambda$ = syntactic (or initial) CCC

# The coloring translation, purely syntactically

On types: $\hat{o} = o$ and $\widehat{\sigma \to \tau} = \underbrace{\widehat{\sigma} \to \ldots \to \widehat{\sigma}}_{k+1 \text{ times}} \to \widehat{\tau}$.

$$\frac{}{\Gamma, x : \tau \vdash x : \tau} \rightsquigarrow \frac{}{\widehat{\Gamma}, x^{(0)} : \widehat{\tau}, \ldots, x^{(k)} : \widehat{\tau} \vdash x^{(0)} : \widehat{\tau}}$$

$$\frac{\Gamma, x : \sigma \vdash t : \tau}{\Gamma \vdash \lambda x.\, t : \sigma \to \tau} \rightsquigarrow \frac{\widehat{\Gamma}, x^{(0)} : \widehat{\sigma}, \ldots, x^{(k)} : \widehat{\sigma} \vdash \hat{t} : \widehat{\tau}}{\widehat{\Gamma} \vdash \widehat{\lambda x.\, t} = \lambda x^{(0)}.\ \ldots\ \lambda x^{(k)}.\, \hat{t} : \widehat{\sigma \to \tau}}$$

$$\frac{\Gamma \vdash t : \sigma \to \tau \quad \Gamma \vdash u : \sigma}{\Gamma \vdash t\, u : \tau} \rightsquigarrow \frac{\dfrac{\widehat{\Gamma} \vdash \hat{t} : \widehat{\sigma \to \tau} \quad \widehat{\Gamma} \vdash \uparrow_0 \widehat{u} : \widehat{\sigma}}{\vdots} \ \ldots \quad \widehat{\Gamma} \vdash \uparrow_k \widehat{u} : \widehat{\sigma}}{\widehat{\Gamma} \vdash \widehat{t\, u} = \hat{t}\, (\uparrow_0 \widehat{u})\ \ldots\ (\uparrow_k \widehat{u}) : \widehat{\tau}}$$

where $\uparrow_c \widehat{u} = \widehat{u}[x^{(i)} := x^{(\max(c,i))} \mid x \in \mathrm{dom}(\Gamma),\ i = 0, \ldots, k]$.

## Going infinitary

**Important remark**

The syntactic presentation of the coloring translation extends effortlessly to *infinitary* simply typed $\lambda$-terms, by reading the rules coinductively.

colored Scott semantics only exists for ST$\lambda$C+recursion; we would like to *define*:

$$\text{infinitary colored Scott} := \text{infinitary Scott} \circ \text{coloring translation}$$

# Going infinitary

**Important remark**

The syntactic presentation of the coloring translation extends effortlessly to *infinitary* simply typed $\lambda$-terms, by reading the rules coinductively.

colored Scott semantics only exists for ST$\lambda$C+recursion; we would like to *define*:

$$\text{infinitary colored Scott} := \text{infinitary Scott} \circ \text{coloring translation}$$

but this requires encoding the parity acceptance conditions
$\rightsquigarrow$ a subset of $\infty$ branches in the output of the translation are “accepting”

# Going infinitary

**Important remark**

The syntactic presentation of the coloring translation extends effortlessly to *infinitary* simply typed $\lambda$-terms, by reading the rules coinductively.

colored Scott semantics only exists for ST$\lambda$C+recursion; we would like to *define*:

$$\text{infinitary colored Scott} := \text{infinitary Scott} \circ \text{coloring translation}$$

but this requires encoding the parity acceptance conditions
$\rightsquigarrow$ a subset of $\infty$ branches in the output of the translation are "accepting"
  $\rightarrow$ a *boundary* in the sense of [Melliès 2017]: introduces and studies a well-behaved Scott semantics for infinitary ST$\lambda$C with boundaries

# Going infinitary

**Important remark**

The syntactic presentation of the coloring translation extends effortlessly to *infinitary* simply typed $\lambda$-terms, by reading the rules coinductively.

colored Scott semantics only exists for ST$\lambda$C+recursion; we would like to *define*:

$$\text{infinitary colored Scott} := \text{infinitary Scott} \circ \text{coloring translation}$$

but this requires encoding the parity acceptance conditions
$\rightsquigarrow$ a subset of $\infty$ branches in the output of the translation are "accepting"
$\rightarrow$ a *boundary* in the sense of [Melliès 2017]: introduces and studies a well-behaved Scott semantics for infinitary ST$\lambda$C with boundaries

**Conjecture (needed to get invariance for infinitary colored Scott)**

*The infinitary colored translation with boundary is compatible with* $\to_\beta^\infty$.

# Conclusion

**Conjecture**

*The infinitary colored translation with boundary is compatible with* $\to_\beta^\infty$.

- would give us a well-behaved colored Scott semantics of infinitary ST$\lambda$C
- as a consequence we could prove our "implicit $\omega$-automata" conjecture
- for more details (e.g. intersection types): cf. our ITRS'24 abstract

# Conclusion

## Conjecture

*The infinitary colored translation with boundary is compatible with* $\to_\beta^\infty$.

- would give us a well-behaved colored Scott semantics of infinitary ST$\lambda$C
- as a consequence we could prove our "implicit $\omega$-automata" conjecture
- for more details (e.g. intersection types): cf. our ITRS'24 abstract

## Some points to remember

- Which infinite structures have decidable MSO theory? (for verification)
  $\longrightarrow$ simply typed $\lambda$-calculus + recursion provides a large class of trees
  (*higher-order model checking* — also reprovable from above conjecture!)
- regular languages in ST$\lambda$C [Hillebrand & Kanellakis 1996] $\rightsquigarrow$ implicit automata
  + its proof: evaluation in a finitary semantics

**Thanks for your attention!**

### Conjecture

*The infinitary colored translation with boundary is compatible with* $\to_\beta^\infty$.

- would give us a well-behaved colored Scott semantics of infinitary ST$\lambda$C
- as a consequence we could prove our "implicit $\omega$-automata" conjecture
- for more details (e.g. intersection types): cf. our ITRS'24 abstract

### Some points to remember

- Which infinite structures have decidable MSO theory? (for verification)
  $\longrightarrow$ simply typed $\lambda$-calculus + recursion provides a large class of trees
  (*higher-order model checking* — also reprovable from above conjecture!)
- regular languages in ST$\lambda$C [Hillebrand & Kanellakis 1996] $\rightsquigarrow$ implicit automata
  + its proof: evaluation in a finitary semantics