1. **Finite semantics of polymorphism, complexity and the power of type fixpoints**
   j.w.w. T. Seiller, P. Pistone and L. Tortora de Falco
2. From normal functors to logarithmic space queries
   j.w.w. P. Pradic

Nguyễn Lê Thành Dũng (a.k.a. Tito) — nltd@nguyentito.eu
LIPN, Université Paris 13
DICE–FOPARA 2019, Prague

1. **Finite semantics of polymorphism, complexity and the power of type fixpoints**
   j.w.w. T. Seiller, P. Pistone and L. Tortora de Falco
2. From normal functors to logarithmic space queries
   j.w.w. P. Pradic

A factual title for the entire thing:
"Sub-polynomial classes in (2nd-order) Elementary Linear Logic via semantic evaluation"

Nguyễn Lê Thành Dũng (a.k.a. Tito) — nltd@nguyentito.eu
LIPN, Université Paris 13
DICE–FOPARA 2019, Prague

Prologue (1)

Second-order Elementary Linear Logic (ELL2):
a programming language for elementary complexity.

Theorem (Girard / Danos & Joinet)

The languages decided by programs of type $!\mathsf{Str} \multimap !^k\mathsf{Bool}$ *in* ELL2 *are exactly the elementary recursive languages.*

- $\mathsf{Bool} = 1 \oplus 1$
- Str = "stratified" Church encoding of bitstrings
- k depends on the language to be recognized

Prologue (1)

Second-order Elementary Linear Logic (ELL2):
a programming language for elementary complexity.

Theorem (Girard / Danos & Joinet)

The languages decided by programs of type $!\mathsf{Str} \multimap !^k\mathsf{Bool}$ *in* ELL2 *are exactly the elementary recursive languages.*

- $\mathsf{Bool} = 1 \oplus 1$
- Str = "stratified" Church encoding of bitstrings
- k depends on the language to be recognized

Related questions:

- what happens if we fix k (*exponential depth*)?
- capture "reasonable" classes within ELL2?

Prologue (2)

Depth $k = 2$ case, in a variant of ELL2:

Theorem (Baillot, APLAS'11)

The proofs of $!\mathsf{Str} \multimap !!\mathsf{Bool}$ *in 2nd-order Elementary Affine Logic* with recursive types *decide exactly the* polynomial time *languages.* (*More generally,* $!\mathsf{Str} \multimap !^k\mathsf{Bool}$ *captures* k-EXPTIME.)

- "Affine" instead of "Linear": only for simplicity
- Recursive types seem necessary...

Prologue (2)

Depth $k = 2$ case, in a variant of ELL2:

Theorem (Baillot, APLAS'11)

The proofs of $!\mathsf{Str} \multimap !!\mathsf{Bool}$ *in 2nd-order Elementary Affine Logic* with recursive types *decide exactly the* polynomial time *languages.* (*More generally,* $!\mathsf{Str} \multimap !^k\mathsf{Bool}$ *captures* k-EXPTIME.)

- "Affine" instead of "Linear": only for simplicity
- Recursive types seem necessary...

Theorem

The proofs of $!\mathsf{Str} \multimap !!\mathsf{Bool}$ *in* ELL2 *decide* regular *languages.*

A result very dependent on the use of *Church encodings*.
New perspectives: other input representations (next episode!).

The intervention of finite semantics

Theorem

The proofs of $!\mathsf{Str} \multimap !!\mathsf{Bool}$ *in* ELL2 *decide* regular *languages.*

Unlike most work on ELL, does not rely on combinatorics on proof nets... instead, apply *semantic evaluation*! As used in:

- Implicit complexity results of the 90's in the *simply typed λ-calculus* (STλ)
- Terui, RTA'12: STλ normalization at fixed order
- Applications to System T / PCF (cf. L. Kristiansen's work)

NB: only *monomorphic* type systems above. We will see why.

Regular languages in the simply typed λ-calculus

Our direct inspiration:

Theorem (Hillebrand & Kanellakis, LICS'96)

The languages decided by STλ*-terms of type* $\mathsf{Str}[A] \to \mathsf{Bool}$ *are exactly the* regular *languages.*

- For $w \in \{0,1\}^*$, $w : \mathsf{Str}[A]$ (meta-$\forall$)
  - $\mathsf{Str}[A] = (A \to A) \to (A \to A) \to (A \to A)$
  - $\overline{w} = \lambda f_0.\, \lambda f_1.\, \lambda x.\, f_{w[0]}\, (\ldots (f_{w[n-1]}\, x) \ldots)$
- $\mathsf{Bool} = o \to o \to o$ (o base type)
- For $t : \mathsf{Str}[A] \to \mathsf{Bool}$, $\mathcal{L}(t) = \{w \in \{0,1\}^* \mid t\, \overline{w} \to_\beta^* \mathtt{true}\}$
- Choose A depending on the language you want

(Correspondence Church encodings / finite automata;
related: higher-order model checking)

Regular languages in $\mathrm{ST}\lambda$

Theorem (Hillebrand & Kanellakis, LICS'96)

For any type A and any $\mathrm{ST}\lambda$*-term* $t : \mathsf{Str}[A] \to \mathsf{Bool}$*, the language* $\mathcal{L}(t) = \{w \in \{0,1\}^* \mid t\,\overline{w} \to_\beta^* \mathtt{true}\}$ *is* regular.

Part 1 of proof.

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

$$w \in \{0,1\}^* \rightsquigarrow \overline{w} : \mathsf{Str}[A] \rightsquigarrow [\![\overline{w}]\!]_{\mathsf{Str}[A]} \in [\![\mathsf{Str}[A]]\!]$$

When $[\![-]\!]$ *non-trivial* ($[\![\mathtt{true}]\!] \neq [\![\mathtt{false}]\!]$), $[\![\overline{w}]\!]_{\mathsf{Str}[A]}$ determines behavior of w w.r.t. all $\mathsf{Str}[A] \to \mathsf{Bool}$ terms:

$$w \in \mathcal{L}(t) \iff t\,\overline{w} \to_\beta^* \mathtt{true} \iff [\![t\,\overline{w}]\!] = [\![t]\!]([\![\overline{w}]\!]) = [\![\mathtt{true}]\!]$$

Goal: to decide $\mathcal{L}(t)$, compute $w \mapsto [\![\overline{w}]\!]$ in some model of $\mathrm{ST}\lambda$.

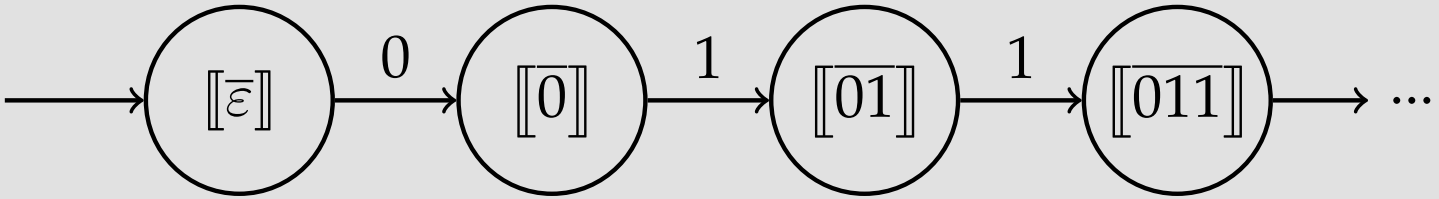
Regular languages in $\mathrm{ST}\lambda$

Theorem (Hillebrand & Kanellakis, LICS'96)

For any type A and any $\mathrm{ST}\lambda$*-term* $t : \mathsf{Str}[A] \to \mathsf{Bool}$*, the language* $\mathcal{L}(t) = \{w \in \{0,1\}^* \mid t\,\overline{w} \to_\beta^* \mathtt{true}\}$ *is* regular.

Part 2 of proof.

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



$$w \in \mathcal{L}(t) \iff [\![t]\!]\left([\![\overline{w}]\!]_{\mathsf{Str}[A]}\right) = [\![\mathtt{true}]\!] \iff w \text{ accepted}$$

□

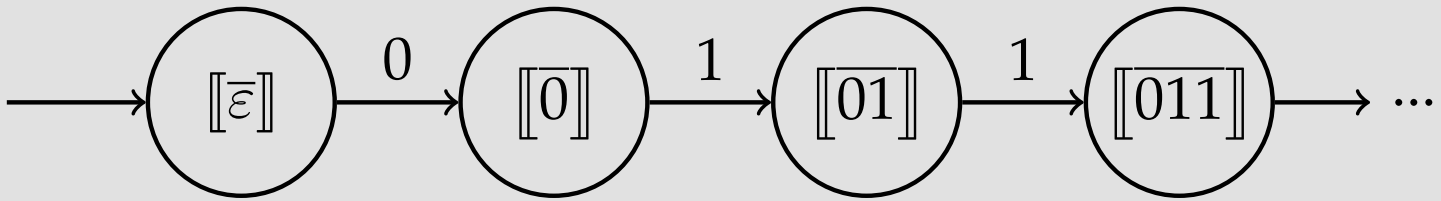
Regular languages in $\mathrm{ST}\lambda$

Theorem (Hillebrand & Kanellakis, LICS'96)

For any type A and any $\mathrm{ST}\lambda$*-term* $t : \mathsf{Str}[A] \to \mathsf{Bool}$*, the language* $\mathcal{L}(t) = \{w \in \{0,1\}^* \mid t\,\overline{w} \to_\beta^* \mathtt{true}\}$ *is* regular.

Part 2 of proof.

We use $[\![-]\!] : \mathrm{ST}\lambda \to \mathsf{FinSet}$ to build a DFA with states $Q = [\![\mathsf{Str}[A]]\!]$, acceptation as $[\![t]\!](-) = [\![\mathtt{true}]\!]$.
($|Q| < \infty$, e.g. $2^{2^{33}}$ when $A = \mathsf{Bool}$)



$$w \in \mathcal{L}(t) \iff [\![t]\!]\left([\![\overline{w}]\!]_{\mathsf{Str}[A]}\right) = [\![\mathtt{true}]\!] \iff w \text{ accepted}$$

For applications of semantics, finiteness is key. □

Stating the main theorem (1)

Theorem

The proofs of $!\mathsf{Str} \multimap !!\mathsf{Bool}$ *in* ELL2 *decide* regular *languages.*

ELL2 = MALL2 (2nd-order Multiplicative-Additive LL)
+ restricted exponential rules:

$$\frac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A} \qquad \frac{\vdash \Gamma}{\vdash \Gamma, ?A} \qquad \frac{\vdash A_1, \ldots, A_n, B}{\vdash ?A_1, \ldots, ?A_n, !B}$$

!-intro: promotion and dereliction must come together.

$\longrightarrow$ enforces *stratification* by !-depth:
the number of !-intro rules between a subproof and the root
cannot change during cut-elimination.

(Equivalently: no dereliction and no digging,
i.e. $!(-)$ lax monoidal functor but not comonad)

Stating the main theorem (2)

Theorem

The proofs of $!\mathsf{Str} \multimap !!\mathsf{Bool}$ *in* ELL2 *decide* regular *languages.*

- $\mathsf{Bool} = 1 \oplus 1$
- $\mathsf{Str} = \forall X.\, !(X \multimap X) \multimap !(X \multimap X) \multimap !(X \multimap X)$
- Note that output has the minimum sensible depth: $\mathrm{depth}(!\mathsf{Str}[A]) = 2 + \mathrm{depth}(A) \geq 2 = \mathrm{depth}(!!\mathsf{Bool})$

Extensional completeness:
instantiate Str at a type of DFA states $A = \bigoplus_{q \in Q} 1$

Soundness: next slide, using finite semantics **for MALL2**.

Semantic evaluation in ELL, in a nutshell

Let $\pi : !\mathsf{Str} \multimap !!\mathsf{Bool}$. There exists

$$\widehat{\pi} : \mathsf{Str}[A_1] \multimap \ldots \multimap \mathsf{Str}[A_n] \multimap !\mathsf{Bool}$$

such that $\forall w.\ \pi(!\overline{w}) = !\widehat{\pi}(\overline{w}[A_1], \ldots, \overline{w}[A_n])$. $\quad (\mathsf{Str} = \forall X.\, \mathsf{Str}[X])$

Thanks to stratification, w.l.o.g. $A_1, \ldots, A_n \in$ MALL2**.**
(The erasure of all depth $\geq k$ exponentials is a functor in ELL2.)

Using finite MALL2 semantics $[\![-]\!]$, $\overline{w}[A]$ induces map

$$\|w\|_A : [\![A \multimap A]\!] \times [\![A \multimap A]\!] \to [\![A \multimap A]\!]$$

such that $\overline{w}[A](!f_1, !f_2) = !g \implies \|w\|_A([\![f_1]\!], [\![f_2]\!]) = [\![g]\!]$.

- Church encoding $\longrightarrow \|w\|_A$ computable by automaton
- $(\|w\|_{A_1}, \ldots, \|w\|_{A_n})$ determine $\widehat{\pi}(\overline{w}[A_1], \ldots, \overline{w}[A_n])$ and therefore $\pi(!\overline{w})$

Finite semantics for MALL2

A finite semantics of an impredicative polymorphic language seems impossible. For instance in System F:

Proposition

Any non-trivial semantics must be injective on $\forall X.\, \mathsf{Str}[X]$.

Finite semantics for MALL2

A finite semantics of an impredicative polymorphic language seems impossible. For instance in System F:

Proposition

Any non-trivial semantics must be injective on $\forall X.\, \mathsf{Str}[X]$.

This example relies fundamentally on *non-linearity*.
No problem for a purely linear polymorphic language!

Theorem

MALL2 *admits non-trivial finite semantics.*

The rest of this first talk focuses on semantics. (Sorry!)

Intuition: such semantics must be "witness-erasing" in some way; i.e. keep only the information about abstract data types ($\exists$) to determine their reaction to the generic programs ($\forall$) using them.

A finite observational quotient

First attempt at finite semantics: quotient of the syntax.

Definition (Equivalence for propositional observations)

Let A be a MALL2 formula and $\pi, \pi' : A$. Define $\pi \sim_A \pi'$ as:

$$\forall B\,\text{MALL0},\ \forall \rho : (A \vdash B),\ \mathbf{cut}(\pi, \rho) \equiv \mathbf{cut}(\pi', \rho)$$

- MALL0 = propositional MALL
- $\equiv$ is usual proof equivalence on MALL0 (think $=_{\beta\eta}$)

Theorem

For any MALL2 *formula A, there are* finitely many *classes for* $\sim_A$.

Sadly, the quotient is a non-effective model in many ways:

- MALL2 provability is undecidable (Lafont 1996)
- this observational equivalence is undecidable (new)

Girard's historical models of linear logic

Let's come back to the origins of linear logic:

- decomposition $A \Rightarrow B \equiv !A \multimap B$ in *coherence spaces* ...

Girard's historical models of linear logic

Let's come back to the origins of linear logic:

- decomposition $A \Rightarrow B \equiv !A \multimap B$ in *coherence spaces* ...
- ...coming from "qualitative domains" for System F
  The system F of variable types, 15 years later (1986)
  represent types w/ parameters as *normal functors*

Girard's historical models of linear logic

Let's come back to the origins of linear logic:

- decomposition $A \Rightarrow B \equiv !A \multimap B$ in *coherence spaces* ...
- ...coming from "qualitative domains" for System F
  The system F of variable types, 15 years later (1986)
  represent types w/ parameters as *normal functors*
- previously: the 1st quantitative semantics of STλ
  Normal functors, power series and λ-calculus (1985)
  origin of "linear" terminology (degree 1 monomial)

Girard's historical models of linear logic

Let's come back to the origins of linear logic:

- decomposition $A \Rightarrow B \equiv !A \multimap B$ in *coherence spaces* ...
- ...coming from "qualitative domains" for System F
  The system F of variable types, 15 years later (1986)
  represent types w/ parameters as *normal functors*
- previously: the 1st quantitative semantics of STλ
  Normal functors, power series and λ-calculus (1985)
  origin of "linear" terminology (degree 1 monomial)
- previously[2]: dilators (`http://girard.perso.math.cnrs.fr/ptlc2.pdf`)

Girard's historical models of linear logic

Let's come back to the origins of linear logic:

- decomposition $A \Rightarrow B \equiv !A \multimap B$ in *coherence spaces* ...
- ...coming from "qualitative domains" for System F
  The system F of variable types, 15 years later (1986)
  represent types w/ parameters as *normal functors*
- previously: the 1st quantitative semantics of STλ
  Normal functors, power series and λ-calculus (1985)
  origin of "linear" terminology (degree 1 monomial)
- previously^2: dilators (`http://girard.perso.math.cnrs.fr/ptlc2.pdf`)

$\rightarrow$ the original semantics of 2nd-order (full) linear logic:
coherence spaces and normal functors

Claim: this semantics is finite and effective for MALL2**.**

Relational / coherent semantics: propositional case

Relational semantics:

- formula/type $\rightsquigarrow$ set
- proof/program $\rightsquigarrow$ subset
  - in case $[\![A \multimap B]\!] = [\![A]\!] \times [\![B]\!]$, subset = *relation*
- composition = relational composition

Coherence spaces refine this:

- type $\rightsquigarrow$ undirected graph (adjacency called *coherence*)
- program $\rightsquigarrow$ clique (pairwise coherent vertex subset)
  - notation: $c \sqsubset X$ means "c clique of X"
  - we still have $|\,[\![A \multimap B]\!]\,| = |\,[\![A]\!]\,| \times |\,[\![B]\!]\,|$ ($|\cdot|$: vertex set)
- composition = relational composition

Forgetful functor $\mathsf{Coh} \to \mathsf{Rel}$ preserves $\otimes$, $\parr$, $\oplus$, $\&$, $(-)^{\perp}$.

Relational / coherent semantics: some examples

In both Rel and Coh, identity is

$$\mathrm{id}_X = \{(x, x) \mid x \in X\} \subset X \times X$$

In Rel, $A \,\&\, B = A \oplus B = A + B$.

In Coh, $|A \,\&\, B| = |A \oplus B| = |A| + |B|$ but

- $a \in A$ and $b \in B$ *coherent* in $A \,\&\, B$
- $a \in A$ and $b \in B$ *incoherent* in $A \oplus B$

so that

- $\mathrm{Cliques}(A \,\&\, B) \cong \mathrm{Cliques}(A) \times \mathrm{Cliques}(B)$
- $\mathrm{Cliques}(A \oplus B) \cong \mathrm{Cliques}(A) + \mathrm{Cliques}(B)$

Variable types: a relational example

Let $\pi : X \multimap X$. For all sets S, $[\![\pi]\!]_{X \mapsto S} \subseteq S \times S$ in Rel.

This family should somehow be "uniform" in S, such that

uniform families $r_S \subset S \times S \cong$ subsets $r \subset [\![\forall X.\, X \multimap X]\!]$

to interpret the $\forall$-intro rule.

Variable types: a relational example

Let $\pi : X \multimap X$. For all sets S, $[\![\pi]\!]_{X \mapsto S} \subseteq S \times S$ in Rel.

This family should somehow be "uniform" in S, such that

uniform families $r_S \subset S \times S \cong$ subsets $r \subset [\![\forall X.\, X \multimap X]\!]$

to interpret the $\forall$-intro rule.

What makes sense in all sets: $=, \neq$.
Represent equality by "bound variables": $\langle x, y, \ldots \vdash t \rangle$.

$$\{(s,s) \mid s \in S\} \leftrightarrow \{\langle x \vdash (x,x)\rangle\}$$
$$\{(s,s') \mid s \neq s' \in S\} \leftrightarrow \{\langle x, y \vdash (x,y)\rangle\}$$
$$S \times S \leftrightarrow \{\langle x \vdash (x,x)\rangle, \langle x, y \vdash (x,y)\rangle\}$$

Variable types: a relational example

Let $\pi : X \multimap X$. For all sets S, $[\![\pi]\!]_{X \mapsto S} \subseteq S \times S$ in Rel.

This family should somehow be "uniform" in S, such that

uniform families $r_S \subset S \times S \cong$ subsets $r \subset [\![\forall X.\, X \multimap X]\!]$

to interpret the $\forall$-intro rule.

What makes sense in all sets: $=, \neq$.
"substitution of bound variables" = functoriality for injections

$$\{(s,s) \mid s \in S\} \leftrightarrow \{\langle x \vdash (x,x)\rangle\}$$
$$\{(s,s) \mid s \in S\} = \{(\iota(x),\iota(x)) \mid \iota : \{x\} \hookrightarrow S\}$$
$$\{(s,s') \mid s \neq s' \in S\} \leftrightarrow \{\langle x,y \vdash (x,y)\rangle\}$$
$$\{(s,s') \mid s \neq s' \in S\} = \{(\iota(x),\iota(y)) \mid \iota : \{x,y\} \hookrightarrow S\}$$

{expr. w/ binders} $\cong$ {uniform families (def. next slide)}

A relational counter-example...

Definition (Girard's "mutilation property")

A family $r_S \subset F(S)$ is *uniform* if for $X \subseteq Y$, $r_X = r_Y \cap F(X)$, and similarly for injections (F functorial on injections).

Uniformity *not closed under composition* in the relational model:

- $\langle x \vdash (*, x) \rangle \leftrightarrow \{*\} \times S \subseteq [\![1 \multimap X]\!]_{X \mapsto S}$
- $\langle x \vdash (x, *) \rangle \leftrightarrow S \times \{*\} \subseteq [\![X \multimap 1]\!]_{X \mapsto S}$
- composition: $\{(*, *)\}$ if $S \neq \emptyset$, $\emptyset$ if $S = \emptyset$

...hence coherence spaces!

Uniformity *not closed under composition* in the relational model. But it works in coherence spaces; miracle due to *stability*.

Definition (Girard's "mutilation property")

A family $r_S \sqsubset F(S)$ is *uniform* if for $X \subseteq Y$, $r_X = r_Y \cap |F(X)|$, and similarly for embeddings (F functorial on embeddings).

where $X \subseteq Y$ means "X induced subgraph of Y".

Composition issue part of the folkore.

- Subtlety overlooked by Girard
- Can be proved for coherence spaces via "Moggi's trick" (mentioned in *Proofs and Types* appendix for another purpose!)
- In relational model: investigated by A. Bac, T. Ehrhard and C. Tasson

Variable types in coherence spaces

So types with n parameters $\rightsquigarrow$ functors $\mathsf{Coh}^n \to \mathsf{Coh}$ where

- objects of Coh: coherence spaces
- morphisms $X \hookrightarrow Y$ in Coh: embeddings
  (i.e. graph isomorphism between X and induced subgraph of Y)

Note that $X \subseteq Y \Rightarrow X^\perp \subseteq Y^\perp$ ($X^\perp$ = complement graph of X), hence *covariance* (cf. previous example $X^\perp \parr X$).

And proofs $\rightsquigarrow$ uniform families. To interpret $\forall$, we want

$$\text{uniform families } c_X \sqsubset F(X) \;\cong\; \text{cliques } c \sqsubset \mathrm{Tr}(F)$$

$\to$ for this, Girard requires F to be a *normal functor*
($\simeq$ categorification of stable functions)

Normal functors

Definition

A functor $F : \mathsf{CohI} \to \mathsf{CohI}$ is *normal* if it preserves
(1) pullbacks and (2) filtered colimits.

Theorem (Girard's normal form theorem)

If $x \in |F(X)|$ *then* $\exists! X_0 \subseteq X$ finite minimal *s.t.* $x \in |F(X_0)|$.

(More precisely: should replace inclusion by embedding in statement; categorically, initiality in a slice cat. of a cat. of elements)

- $\exists$ minimal X_0 ensured by (1) (cf. *polynomial functors*)
- finiteness ensured by (2) (normal f. = *finitary* polynomial f.)

Set of *normal forms* (morally, $X_0 = (\mathrm{fv}(x), \text{coherence})$):

$$\mathrm{NF}(F) = \{\langle X_0 \vdash x\rangle \mid x \in F(X_0) \text{ with } X_0 \text{ minimal}\}$$

Traces of normal functors

Representation of $\forall$:

- define a non-reflexive coherence relation on $\mathrm{NF}(F)$
- def. $|\mathrm{Tr}(F)| = \{\langle X_0 \vdash x\rangle \in \mathrm{NF}(F) \mid \langle X_0 \vdash x\rangle \text{ self-coherent}\}$
- restrict coherence on $\mathrm{NF}(F)$ to $|\mathrm{Tr}(F)|$
  $\to$ makes $\mathrm{Tr}(F)$ a coherence space
- its cliques are in bijection with uniform families for F

Revisiting previous examples:

- $[\![\forall X.\, X \multimap X]\!] = \{\langle x \vdash (x, x)\rangle\} \cong [\![1]\!]$
  only 1 self-coherent point
- $[\![\forall X.\, 1 \multimap X]\!] = [\![\forall X.\, X \multimap 1]\!] = \emptyset$
  thus no problem with composition here!

Normal functors of finite degree

New: for our purposes, we consider *finite* normal functors, i.e.:

- preserving finiteness: $\mathrm{Card}(X) < \infty \implies \mathrm{Card}(F(X)) < \infty$
- of *finite degree*: $\deg(F) = \sup_{\langle X_0 \vdash x \rangle \in \mathrm{NF}(F)} \mathrm{Card}(X_0) < \infty$

(Mutatis mutandis for open types with n variables.)

Intuition: $\deg(F)$ = max number of bound variables used

$$\deg(F^\perp) = \deg(F) \quad \deg(X \mapsto \mathrm{Tr}(F(X, -))) \leq \deg(F)$$

$$\deg(F \otimes G) = \deg(F) + \deg(G) \quad \deg(F \oplus G) = \max(\deg(F), \deg(G))$$

Also, $\mathrm{Card}(\mathrm{Tr}(F(X, -))$ bounded using $\deg(F)$.

Theorem

Coherence spaces and finite normal functors provide a finite semantics of MALL2.

Summary of the first episode

- $!\mathsf{Str} \multimap !!\mathsf{Bool}$ in ELL2 captures
  - polynomial time with recursive types (Baillot 2011)
  - **regular languages** without them
- Proof by semantic evaluation, using stratification + finite semantics for MALL2
- The historical coherence space model is finite for MALL2
  - categorical formalism $\longrightarrow$ syntactic presentation

Summary of the first episode

- $!\mathsf{Str} \multimap !!\mathsf{Bool}$ in ELL2 captures
  - polynomial time with recursive types (Baillot 2011)
  - **regular languages** without them
- Proof by semantic evaluation, using stratification + finite semantics for MALL2
- The historical coherence space model is finite for MALL2
  - categorical formalism $\longrightarrow$ syntactic presentation

Next:

- Replace Str with alternative input representation
  $\longrightarrow$ get larger sub-polynomial complexity class
- Apply size bounds and effectivity of coherence spaces

1. Finite semantics of polymorphism, complexity and the power of type fixpoints
   j.w.w. T. Seiller, P. Pistone and L. Tortora de Falco
2. **From normal functors to logarithmic space queries**
   j.w.w. P. Pradic

A factual title for the entire thing:
"Sub-polynomial classes in (2nd-order) Elementary Linear Logic via semantic evaluation"

Nguyễn Lê Thành Dũng (a.k.a. Tito) — nltd@nguyentito.eu
LIPN, Université Paris 13
DICE–FOPARA 2019, Prague

Recap of the previous episode...

Previously:
methods for ICC in simply-typed λ-calculus (STλ)
transposed to second-order Elementary Linear Logic (ELL2),
thanks to finite semantics for MALL2.

Theorem (Hillebrand & Kanellakis, LICS'96)

The languages decided by STλ*-terms of type* $\mathsf{Str}[A] \to \mathsf{Bool}$ *are exactly the* regular *languages.*

Theorem

The proofs of $!\mathsf{Str} \multimap !!\mathsf{Bool}$ *in* ELL2 *decide* regular *languages.*

Both use *Church encodings*, proofs by *semantic evaluation*.

The return of Hillebrand & Kanellakis

Idea: *change input representation* to overcome expressivity limit.

Direct inspiration from 90's work on STλ.

The return of Hillebrand & Kanellakis

Idea: *change input representation* to overcome expressivity limit.

Direct inspiration from 90's work on STλ (called TLC below).

> *These classical expressibility results have cast a negative and slightly confusing light on the possible encodings of computations in TLC. Simple types have been criticized for limiting flexibility in programs, but they have also been criticized for limiting expressibility.* [...]
> *The criticism of TLC expressibility is unjustified, and a theme of this paper is to quantify how rich a framework TLC is for expressing computations,* provided that the right setting is chosen.

The return of Hillebrand & Kanellakis

Idea: *change input representation* to overcome expressivity limit.
Direct inspiration from 90's work on STλ (called TLC below).

> *These classical expressibility results have cast a negative and slightly confusing light on the possible encodings of computations in TLC. Simple types have been criticized for limiting flexibility in programs, but they have also been criticized for limiting expressibility.* [...]
> *The criticism of TLC expressibility is unjustified, and a theme of this paper is to quantify how rich a framework TLC is for expressing computations,* provided that the right setting is chosen.

in *Database Query Languages Embedded in the Typed Lambda Calculus*, Hillebrand, Kanellakis & Mairson, 1996

Database queries?

Relational database = *finite relational* (*1st-order*) *structure*

a.k.a. *finite model* (relation = set of records (tuples) = table)

$\rightarrow$ connections w/ finite model theory / descriptive complexity

Example

Relational signature for binary strings: $\langle \leq, S \rangle$.

Finite models are $(D, \leq^D, S^D)$, $|D| < \infty$. $S^D(d) =$ "d^{th} bit is 1".

Database queries?

Relational database = *finite relational* (*1st-order*) *structure*
a.k.a. *finite model* (relation = set of records (tuples) = table)
$\rightarrow$ connections w/ finite model theory / descriptive complexity

Example

Relational signature for binary strings: $\langle \leq, S \rangle$.
Finite models are $(D, \leq^D, S^D)$, $|D| < \infty$. $S^D(d) =$ "d^{th} bit is 1".

Theorem (Hillebrand, Kanellakis & Mairson, LICS'93)

With the right choice of encoding, the queries over finite models in STλ *are exactly the* elementary recursive *queries.*

(Quote in previous slide from journal version in *Information and Computation*)

$\longrightarrow$ just like ELL2, STλ is a "calculus of elementary complexity"

(Remark: both embed in Linear Logic by Levels)

What this talk is about

We define a type Inp of inputs as finite models in ELL2.

Queries computed by proofs of $\mathsf{Inp} \multimap !!\mathsf{Bool}$ should:

- go beyond regular languages
- still be below P because of geometric constraint (depth 2)

What this talk is about

We define a type Inp of inputs as finite models in ELL2.

Queries computed by proofs of $\mathsf{Inp} \multimap !!\mathsf{Bool}$ should:

- go beyond regular languages ($\subset \mathsf{NC}^1$)
- still be below P because of geometric constraint (depth 2)

What we obtain – unexpectedly – is *logarithmic space*:

- lower bound: all L queries expressible
- upper bound: NL
- encouraging partial results towards L upper bound
  - exact characterization of L with ad-hoc restriction on $\exists$-intro

What this talk is about

We define a type Inp of inputs as finite models in ELL2.

Queries computed by proofs of $\mathsf{Inp} \multimap !!\mathsf{Bool}$ should:

- go beyond regular languages ($\subset \mathsf{NC}^1$)
- still be below P because of geometric constraint (depth 2)

What we obtain – unexpectedly – is *logarithmic space*:

- lower bound: all L queries expressible
- upper bound: NL (actually L^{UL}, UL = *unambiguous* NL)
- encouraging partial results towards L upper bound
  - exact characterization of L with ad-hoc restriction on $\exists$-intro

What this talk is about

We define a type Inp of inputs as finite models in ELL2.

Queries computed by proofs of $\mathsf{Inp} \multimap !!\mathsf{Bool}$ should:

- go beyond regular languages ($\subset \mathsf{NC}^1$)
- still be below P because of geometric constraint (depth 2)

What we obtain – unexpectedly – is *logarithmic space*:

- lower bound: all L queries expressible
- upper bound: NL (actually L^{UL}, UL = *unambiguous* NL)
- encouraging partial results towards L upper bound
  - exact characterization of L with ad-hoc restriction on $\exists$-intro

Another precursor (Gurevich, FOCS'83):
L = primitive recursion over finite models

Finite models in ELL

Let's encode a finite model of domain size n.

- Type of domain elements: $D = 1 \oplus \ldots (n \text{ times}) \ldots \oplus 1$
- Domain iterator: $\mathsf{List}[D] = \forall X.\, !(D \multimap X \multimap X) \multimap !(X \multimap X)$
- k-ary relation: $\mathsf{Rel}_k[D] = D^k \multimap \mathsf{Bool}$

Finite models in ELL

Let's encode a finite model of domain size n.

- Type of domain elements: $D = 1 \oplus \ldots (n \text{ times}) \ldots \oplus 1$
- Domain iterator: $\mathsf{List}[D] = \forall X.\, !(D \multimap X \multimap X) \multimap !(X \multimap X)$
- k-ary relation: $\mathsf{Rel}_k[D] = D^k \multimap \mathsf{Bool}$
- Package this in an *existential type* (a.k.a. abstract data type)

$$\mathsf{Inp} = \exists D.\, !\mathsf{List}[D] \otimes \bigotimes_{i=1}^{m} !!\mathsf{Rel}_{k_i}[D]$$

for signature w/ relation symbols of arities $k_1, \ldots, k_m$
$\rightarrow$ type Inp independent from size n

Finite models in ELL

Let's encode a finite model of domain size n.

- Type of domain elements: $D = 1 \oplus \ldots (n \text{ times}) \ldots \oplus 1$
- Domain iterator: $\mathsf{List}[D] = \forall X.\, !(D \multimap X \multimap X) \multimap !(X \multimap X)$
- k-ary relation: $\mathsf{Rel}_k[D] = D^k \multimap \mathsf{Bool}$
- Package this in an *existential type* (a.k.a. abstract data type)

$$\mathsf{Inp} = \exists D.\, !\mathsf{List}[D] \otimes \bigotimes_{i=1}^{m} !!\mathsf{Rel}_{k_i}[D] \otimes !!\mathsf{Cont}[D] \otimes !!\mathsf{Wk}[D]$$

  for signature w/ relation symbols of arities $k_1, \ldots, k_m$
  $\rightarrow$ type Inp independent from size n
- Interface needs to allow non-linear use of D:
  $\mathsf{Cont}[D] = D \multimap D \otimes D$, $\mathsf{Wk}[D] = D \multimap 1$

Lower bound via descriptive complexity (1)

Descriptive complexity considers queries defined by formulas in extensions of 1st-order logic, to capture complexity classes.

Theorem (Immerman 1983)

L = *queries in 1st-order logic* + **deterministic transitive closure**.

$\phi, \psi, \ldots ::= \mathcal{R}_i(x_1, \ldots, x_{r_i}) \mid \neg\phi \mid \phi \vee \psi \mid \exists x.\phi \mid \mathrm{DTC}_{\vec{x},\vec{y}}(\phi)$

($\mathcal{R}_i$: relation in signature; x: variable for domain element)

Lower bound via descriptive complexity (1)

Descriptive complexity considers queries defined by formulas in extensions of 1st-order logic, to capture complexity classes.

Theorem (Immerman 1983)

L = *queries in 1st-order logic* + **deterministic transitive closure**.

$\phi, \psi, \ldots ::= \mathcal{R}_i(x_1, \ldots, x_{r_i}) \mid \neg\phi \mid \phi \vee \psi \mid \exists x.\phi \mid \mathrm{DTC}_{\vec{x},\vec{y}}(\phi)$
($\mathcal{R}_i$: relation in signature; x: variable for domain element)

For $R \subseteq D^k \times D^k$ (D: domain):

- $R_{\mathrm{d}} = \{(x, y) \in R \mid \forall z \neq y, (x, z) \notin R\}$
  - graph of *partial function* $f_R : D^k \rightharpoonup D^k$
- $\mathrm{DTC}(R) = R_{\mathrm{d}}^*$ (reflexive transitive closure of R_{d})
  - $(x, y) \in \mathrm{DTC}(R) \iff \exists n : y = (f_R)^n(x)$

Note: general TC ($R \mapsto R^*$ instead of R_{d}^*) corresponds to NL.

Lower bound via descriptive complexity (2)

How to compute transitive closure in ELL2?

$\psi_R : Q \mapsto \{(x, z) \mid x = z \vee (\exists y : (x, y) \in R \wedge (y, z) \in Q)\}$
R^* = *least fixpoint* of ψ_R, obtained by iteration (ψ_R monotone).

Problem: $\mathsf{List}[D]$ can only iterate *linear* maps, because of ELL stratification, and ψ_R is non-linear (because of $\exists y$).

Lower bound via descriptive complexity (2)

How to compute transitive closure in ELL2?

$\psi_R : Q \mapsto \{(x, z) \mid x = z \vee (\exists y : (x, y) \in R \wedge (y, z) \in Q)\}$
R^* = *least fixpoint* of ψ_R, obtained by iteration (ψ_R monotone).

Problem: $\mathsf{List}[D]$ can only iterate *linear* maps, because of ELL stratification, and ψ_R is non-linear (because of $\exists y$).

For DTC, we have: $\psi_{R_\mathrm{d}} : Q \mapsto \{(x, z) \mid x = z \vee (f_R(x), z) \in Q\}$
which can be expressed as a linear map $\mathsf{Rel}_{2k}[D] \multimap \mathsf{Rel}_{2k}[D]$.

Determinism (L vs NL) corresponds to *linearity*.

Lower bound via descriptive complexity (2)

How to compute transitive closure in ELL2?

$\psi_R : Q \mapsto \{(x,z) \mid x = z \vee (\exists y : (x,y) \in R \wedge (y,z) \in Q)\}$
$R^* =$ *least fixpoint* of ψ_R, obtained by iteration (ψ_R monotone).

Problem: $\mathsf{List}[D]$ can only iterate *linear* maps, because of ELL stratification, and ψ_R is non-linear (because of $\exists y$).

For DTC, we have: $\psi_{R_\mathrm{d}} : Q \mapsto \{(x,z) \mid x = z \vee (f_R(x),z) \in Q\}$
which can be expressed as a linear map $\mathsf{Rel}_{2k}[D] \multimap \mathsf{Rel}_{2k}[D]$.

Determinism (L vs NL) corresponds to *linearity*. In the end:

Theorem

All L *queries can be computed by an* ELL2 *proof of* $\mathsf{Inp} \multimap \mathsf{!!Bool}$.

Alternative proof for inputs as strings: two-way multihead automata

Towards the upper bound

When using Church-encoded input:

- a proof $\pi : !\mathsf{Str} \multimap !!\mathsf{Bool}$ comes from some proof $\hat{\pi} : \mathsf{Str}[A_1] \multimap \ldots \multimap \mathsf{Str}[A_k] \multimap \mathsf{Bool}$
- main computational engine: iterating over the string using $\mathsf{Str}[A_i]$; accumulator type $A_i \in$ MALL2.
- A_i is known in advance and has a finite semantics $[\![A_i]\!]$
  $\rightarrow$ iteration can be performed by a finite automaton.

Towards the upper bound

When using Church-encoded input:

- a proof $\pi : !\mathsf{Str} \multimap !!\mathsf{Bool}$ comes from some proof $\hat{\pi} : \mathsf{Str}[A_1] \multimap \ldots \multimap \mathsf{Str}[A_k] \multimap \mathsf{Bool}$
- main computational engine: iterating over the string using $\mathsf{Str}[A_i]$; accumulator type $A_i \in$ MALL2.
- A_i is known in advance and has a finite semantics $[\![A_i]\!]$ $\rightarrow$ iteration can be performed by a finite automaton.

With finite models as inputs:

- Recall that $\mathsf{Inp} = \exists D.\, !\mathsf{List}[D] \otimes !!B_1 \otimes \ldots \otimes !!B_l$, $B_i \in$ MALL2
- main computational engine: iterating over domain elements using $\mathsf{List}[D][A_i]$; accumulator type $A_i \in$ MALL2
- thanks to $\exists D$, A_i (and thus $\mathrm{Card}([\![A_i]\!])$) depends on D
  - expressivity comes from *existential* input type

Coherence spaces and normal functors: bounds and complexity

Previous episode: coherence spaces and normal functors *of finite degree* provide a finite semantics for MALL2.

Theorem (finite degree = polynomial growth)

Let $F : \mathsf{Coh} \to \mathsf{Coh}$ *be a normal functor. There exists* $d \in \mathbb{N}$ *s.t.* $\mathrm{Card}(|F(X)|) = O(\mathrm{Card}(|X|^d)$ *if and only if* F *is a finite n.f.*
In that case, $\deg F$ *is the smallest such* d*.*

Complexity properties (thanks to syntactic presentation):

- Given fixed F, $X \mapsto F(X)$ computable in L (logspace)
- Given fixed $c \sqsubset \mathrm{Tr}(F)$, $X \mapsto c_X \sqsubset F(X)$ computable in L

$\longrightarrow$ hope: perform semantic evaluation in coherence spaces

(In fact the semantics is *effective*:
$A \mapsto [\![A]\!]$ and $(\pi : A) \mapsto [\![\pi]\!] \sqsubset [\![A]\!]$ are computable.)

Upper bound by semantic evaluation

Bottleneck for evaluation in coherence spaces: iterations $[\![D \multimap A_i \multimap A_i]\!] \to [\![A_i \multimap A_i]\!]$ corresponding to $\mathsf{List}[D][A_i]$.

Thanks to $\exists D$, A_i (and thus $\mathrm{Card}([\![A_i]\!])$) depends on D.
$\mathrm{Card}([\![A_i]\!]) = \mathrm{poly}(\mathrm{Card}(D))$, and instantiation complexity is L.

Upper bound by semantic evaluation

Bottleneck for evaluation in coherence spaces: iterations $[\![D \multimap A_i \multimap A_i]\!] \to [\![A_i \multimap A_i]\!]$ corresponding to $\mathsf{List}[D][A_i]$.

Thanks to $\exists D$, A_i (and thus $\mathrm{Card}([\![A_i]\!])$) depends on D.
$\mathrm{Card}([\![A_i]\!]) = \mathrm{poly}(\mathrm{Card}(D))$, and instantiation complexity is L.

So these iterations L-reduce to the *Iterated composition problem*:
given a coherence space A and cliques $f_1, \ldots, f_n \sqsubset A \multimap A$,
compute composition $(f_1 \circ \ldots \circ f_n) \sqsubset A \multimap A$

Naive goal: solve this in L
$\longrightarrow$ evaluate $\mathsf{List}[D][A_i]$ iterations
$\longrightarrow$ get L soundness for our ELL2 queries

Note: Previous space bounds for variants of LL used *Geometry of Interaction*. Not possible here (additives/quantifiers).

Complexity of iterated composition

Iterated composition problem:
given a coherence space A and cliques $f_1, \ldots, f_n \sqsubset A \multimap A$,
compute (relational) composition $(f_1 \circ \ldots \circ f_n) \sqsubset A \multimap A$

- Iterated composition of general relations is NL-complete
  $\longrightarrow \mathsf{NL}$ bound on our ELL2 queries

Complexity of iterated composition

Iterated composition problem:
given a coherence space A and cliques $f_1, \ldots, f_n \sqsubset A \multimap A$,
compute (relational) composition $(f_1 \circ \ldots \circ f_n) \sqsubset A \multimap A$

- Iterated composition of general relations is NL-complete
  $\longrightarrow \mathsf{NL}$ bound on our ELL2 queries
- Stability property in coherence spaces:
  $\forall (x, y) \in (f_1 \circ \ldots \circ f_n), \exists! (x = x_0, x_1, \ldots, x_n = y) : (x_{i-1}, x_i) \in f_i$
  uniqueness of x_i translates to *unambiguous* nondeterminism:
  for each input, at most 1 accepting run
  $\longrightarrow$ iterated composition in UL, ELL2 queries in L^{UL}

Complexity of iterated composition

Iterated composition problem:
given a coherence space A and cliques $f_1, \ldots, f_n \sqsubset A \multimap A$,
compute (relational) composition $(f_1 \circ \ldots \circ f_n) \sqsubset A \multimap A$

- Iterated composition of general relations is NL-complete
  $\longrightarrow \mathsf{NL}$ bound on our ELL2 queries
- Stability property in coherence spaces:
  $\forall (x, y) \in (f_1 \circ \ldots \circ f_n), \exists! (x = x_0, x_1, \ldots, x_n = y) : (x_{i-1}, x_i) \in f_i$
  uniqueness of x_i translates to *unambiguous* nondeterminism:
  for each input, at most 1 accepting run
  $\longrightarrow$ iterated composition in UL, ELL2 queries in L^{UL}
- Possible in L for coherence spaces?
  Don't know, seems very hard, possibly false.
  Idea: sidestep the issue by *fixed-parameter complexity*.
  Not yet working in general case.

Some ideas from game semantics (1)

Idea: $A[D/X]$ should be a "game" with $O(1)$ depth and $\mathrm{poly}(\mathrm{Card}(D))$ branching (A fixed, D varying).
To evaluate ELL2 queries, iterated composition in L at (arbitrary) fixed depth suffices.

Proof of concept for now, carried out in coherence spaces: games of depth 3.

Some ideas from game semantics (1)

Idea: $A[D/X]$ should be a "game" with $O(1)$ depth and $\mathrm{poly}(\mathrm{Card}(D))$ branching (A fixed, D varying).
To evaluate ELL2 queries, iterated composition in L at (arbitrary) fixed depth suffices.

Proof of concept for now, carried out in coherence spaces: games of depth 3. Enough room for extensional completeness!

Theorem

The L *queries are exactly those expressible as proofs of* $\mathsf{Inp} \multimap !!\mathsf{Bool}$ *in* ELL2 *with* $\exists$*-intro restricted to types of the form* $P \otimes (Q \multimap R)$*, where* P*,* Q *and* R *are* positive quantifier-free *types.*

Typically: $\mathsf{Rel}_k[D] = 1 \otimes (D^k \multimap 1 \oplus 1)$,
becomes depth 2 game when D instantiated to $1 \oplus \ldots \oplus 1$.

Some ideas from game semantics (2)

Write $[n] = 1 \oplus \ldots \oplus 1$. Depth 2 games: $[p] \multimap [q] = \bigwith_{i=1}^{p} \bigoplus_{j=1}^{q} 1$.

Typically: $\mathsf{Rel}_k[D] = 1 \otimes (D^k \multimap 1 \oplus 1) = [n^k] \multimap [2]$ when $D = [n]$
opponent plays $\{1, \ldots, n^k\}$ / player answers true/false

Iterated composition of $f_1, \ldots, f_n : ([p] \multimap [q]) \multimap ([p] \multimap [q])$:

- first propagate $[p]$ backwards (negative polarity)
- then propagate $[q]$ forwards (positive polarity)

From our ELL2 encoding of deterministic transitive closure, recover naive procedure.

Some ideas from game semantics (2)

Write $[n] = 1 \oplus \ldots \oplus 1$. Depth 2 games: $[p] \multimap [q] = \bigwith_{i=1}^{p} \bigoplus_{j=1}^{q} 1$.

Typically: $\mathsf{Rel}_k[D] = 1 \otimes (D^k \multimap 1 \oplus 1) = [n^k] \multimap [2]$ when $D = [n]$
opponent plays $\{1, \ldots, n^k\}$ / player answers true/false

Iterated composition of $f_1, \ldots, f_n : ([p] \multimap [q]) \multimap ([p] \multimap [q])$:

- first propagate $[p]$ backwards (negative polarity)
- then propagate $[q]$ forwards (positive polarity)

From our ELL2 encoding of deterministic transitive closure, recover naive procedure.

Depth 3 ($[m] \otimes ([p] \multimap [q])$): starts getting more tricky
Depth ≥ 4: unsolved for now (but soon!)

Summary of second episode

Conjecture

Queries defined by ELL2 *proofs of* $\mathsf{Inp} \multimap !!\mathsf{Bool}$ *compute* L *queries.*

- A functional language for space-bounded database queries
- First characterization of sub-polynomial class in a language with impredicative polymorphism

Results for now:

- Extensional completeness via descriptive complexity
- NL upper bound w/ coherence spaces + normal functors
  - Novelty: space bound in LL without GoI
- Algorithm with game-semantical flavor
  - Proves L upper bound for restricted case

Towards soundness in general case: hypercoherences as games?