Implicit complexity and finite models in the simply typed λ-calculus

Nguyễn Lê Thành Dũng (a.k.a. Tito) — `nltd@nguyentito.eu`
Laboratoire d'informatique de Paris Nord (LIPN), Université Paris 13
GT Scalp inaugural meeting, Orsay, November 27th, 2018

Implicit complexity with proofs-as-programs

Curry-Howard approach to implicit complexity:

1. Define logic / programming language
2. Bound evaluation complexity (*soundness*)
3. Show language expressivity (*extensional completeness*)
4. Result: expressible functions = some complexity class

Finding a logic (e.g. Girard's Light Linear Logic) for a given complexity class (e.g. P): usually non-trivial.

This talk: instead, ask (2)–(4) for the well-known *simply typed λ-calculus* (STλ).
Old results from the 90's which deserve to be better known.

If time permits: adaptation of these old methods to Elementary Linear Logic (my own work, joint with Thomas Seiller).

Church integers in STλ

For all simple types A, Church integers can be typed as

$$\overline{n} = \lambda f.\lambda x. f(\ldots(f\,x)) : \mathsf{Nat}[A] = (A \to A) \to (A \to A)$$

Take $\mathsf{mult} = \lambda n.\lambda m.\lambda f.\, n\,(m\,f) : \mathsf{Nat}[A] \to \mathsf{Nat}[A] \to \mathsf{Nat}[A]$.
$\mathsf{mult}\,\overline{2} : \mathsf{Nat}[A] \to \mathsf{Nat}[A]$ can be iterated by a $\mathsf{Nat}[\mathsf{Nat}[A]]$...

$$\longrightarrow \mathsf{exp2} = \lambda n.\, n\,(\mathsf{mult}\,\overline{2})\,\overline{1} : \mathsf{Nat}[\mathsf{Nat}[A]] \to \mathsf{Nat}[A]$$

which cannot be iterated!

Church integers in STλ

For all simple types A, Church integers can be typed as

$$\overline{n} = \lambda f.\lambda x. f(\dots(f\,x)) : \mathsf{Nat}[A] = (A \to A) \to (A \to A)$$

Take $\mathsf{mult} = \lambda n.\lambda m.\lambda f.\, n\,(m\,f) : \mathsf{Nat}[A] \to \mathsf{Nat}[A] \to \mathsf{Nat}[A]$.
$\mathsf{mult}\,\overline{2} : \mathsf{Nat}[A] \to \mathsf{Nat}[A]$ **can be iterated by a** $\mathsf{Nat}[\mathsf{Nat}[A]]$...

$$\longrightarrow \mathsf{exp2} = \lambda n.\, n\,(\mathsf{mult}\,\overline{2})\,\overline{1} : \mathsf{Nat}[\mathsf{Nat}[A]] \to \mathsf{Nat}[A]$$

which cannot be iterated! Smaller types, still heterogenous:

$$\mathsf{exp2} = \lambda n.\, n\,\overline{2} : \mathsf{Nat}[A \to A] \to \mathsf{Nat}[A]$$

Church integers in STλ

For all simple types A, Church integers can be typed as

$$\overline{n} = \lambda f.\lambda x. f(\ldots(f x)) : \mathsf{Nat}[A] = (A \to A) \to (A \to A)$$

Take $\mathsf{mult} = \lambda n.\lambda m.\lambda f. n\,(m f) : \mathsf{Nat}[A] \to \mathsf{Nat}[A] \to \mathsf{Nat}[A]$.
$\mathsf{mult}\,\overline{2} : \mathsf{Nat}[A] \to \mathsf{Nat}[A]$ can be iterated by a $\mathsf{Nat}[\mathsf{Nat}[A]]$...

$$\longrightarrow \mathsf{exp2} = \lambda n. n\,(\mathsf{mult}\,\overline{2})\,\overline{1} : \mathsf{Nat}[\mathsf{Nat}[A]] \to \mathsf{Nat}[A]$$

which cannot be iterated! Smaller types, still heterogenous:

$$\mathsf{exp2} = \lambda n. n\,\overline{2} : \mathsf{Nat}[A \to A] \to \mathsf{Nat}[A]$$

Towers of exponentials of fixed height $\mathsf{Nat}[T[A]] \to \mathsf{Nat}[A]$,
but non-elementary functions seem out of reach from STλ.

The computational power of STλ (1)

Recall k-$\mathsf{EXPTIME} = \mathsf{DTIME}$(tower of exponentials of height k),
$\mathsf{ELEMENTARY} = \bigcup_{k \in \mathbb{N}} k\text{-}\mathsf{EXPTIME}$.

Let's simulate an $\mathsf{EXPTIME}$ Turing machine in STλ.
Code its transition function as $t : S \to S$ (S type of states).
From $\overline{n} : \mathsf{Nat}[S \to S]$, obtain $\mathsf{exp2}\ \overline{n} : \mathsf{Nat}[S]$.
Use this to iterate t 2^n times, starting from coding of initial state.

Similarly, using $\overline{n} : \mathsf{Nat}[T[S]]$ for big enough T, one can show that β-reduction in STλ is k-$\mathsf{EXPTIME}$-hard for all k.

The computational power of STλ (2)

β-reduction in STλ is k-EXPTIME-hard for all k.
From the time hierarchy theorem follows:

Theorem (Statman 1982)

β-equivalence of STλ *terms is not in* ELEMENTARY.

The proof we presented is due to Mairson (1992).[1]

[1]"A simple proof of a theorem of Statman", *Theoretical Computer Science*, 1992.

The computational power of STλ (2)

β-reduction in STλ is k-EXPTIME-hard for all k.
From the time hierarchy theorem follows:

Theorem (Statman 1982)

β-equivalence of STλ *terms is not in* ELEMENTARY*.*

The proof we presented is due to Mairson (1992).[1]

So, STλ can somehow express all ELEMENTARY computations.
And this kind of encoding shouldn't work beyond ELEMENTARY.

$\rightarrow$ implicit complexity characterization of ELEMENTARY by STλ?

[1] "A simple proof of a theorem of Statman", *Theoretical Computer Science*, 1992.

STλ predicates on Church-encoded strings (1)

Let L be any ELEMENTARY language. We would like a STλ term t_L with the right type deciding L. That is,

$$\forall w \in \{0,1\}^*,\ t_L\,\overline{w} \rightarrow_\beta^* \texttt{true} \iff w \in L$$

Need to define encoding of inputs $\overline{w}$.
Natural solution: use Church encoding of bitstrings
$\mathsf{Str}[A] = (A \to A) \to (A \to A) \to (A \to A)$.

STλ predicates on Church-encoded strings (1)

Let L be any ELEMENTARY language. We would like a STλ term t_L with the right type deciding L. That is,

$$\forall w \in \{0,1\}^*,\ t_L\,\overline{w} \to_\beta^* \texttt{true} \iff w \in L$$

Need to define encoding of inputs $\overline{w}$.
Natural solution: use Church encoding of bitstrings
$\mathsf{Str}[A] = (A \to A) \to (A \to A) \to (A \to A)$.

However this naive attempt fails spectacularly.

Theorem (Hillebrand & Kanellakis, LICS'96)

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

(Note: A can be chosen depending on which regular language we want to decide.)

STλ predicates on Church-encoded strings (2)

Theorem (Hillebrand & Kanellakis, LICS'96)

For any type A and any STλ*-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.

Idea: use a *finite semantics*, e.g. $[\![-]\!] : \mathrm{ST}\lambda \to \mathsf{FinSet}$; one can build a finite automaton with states $[\![\mathsf{Str}[A]]\!]$ recognizing $\mathcal{L}(t)$.

STλ predicates on Church-encoded strings (2)

Theorem (Hillebrand & Kanellakis, LICS'96)

For any type A and any STλ*-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.

Idea: use a *finite semantics*, e.g. $[\![-]\!] : \mathsf{ST}\lambda \to \mathsf{FinSet}$; one can build a finite automaton with states $[\![\mathsf{Str}[A]]\!]$ recognizing $\mathcal{L}(t)$.

$\longrightarrow$ *semantic evaluation* technique,
often useful for studying complexity in STλ
(Terui: "better semantics, faster computation")

STλ predicates on Church-encoded strings (2)

Theorem (Hillebrand & Kanellakis, LICS'96)

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

Idea: use a *finite semantics*, e.g. $[\![-]\!] : \mathrm{ST}\lambda \to \mathsf{FinSet}$; one can build a finite automaton with states $[\![\mathsf{Str}[A]]\!]$ recognizing $\mathcal{L}(t)$.

$\longrightarrow$ *semantic evaluation* technique,
often useful for studying complexity in STλ
(Terui: "better semantics, faster computation")

$\longrightarrow$ correspondence Church encoding / finite automata,
extended to automata over infinite trees through finite semantics for λY (i.e. STλ + fixpoints):
semantic approach to *higher-order model checking*
(Aehlig, Salvati–Walukiewicz, Grellois–Melliès...)

Towards extensional completeness

To express all ELEMENTARY predicates in STλ we need an alternative input representation.

Such an alternative is studied in Hillebrand's PhD thesis, *Finite Model Theory in the Simply Typed Lambda Calculus* (1994), supervised by Kanellakis.

Finite model theory $\neq$ finite semantics of programs!
It refers to *finite first-order structures*, as used

- in descriptive complexity,
- in the theory of relational databases
  (Kanellakis came from the database community).

A bit of descriptive complexity

Data represented as (totally ordered) finite structures over a first-order signature made of relation symbols.

Example

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".

Descriptive complexity: characterize a complexity class $\mathcal{C}$ as set of *queries* written in some logic $L_{\mathcal{C}}$, i.e. "is this $L_{\mathcal{C}}$ formula true in this finite model?". For instance:

Theorem (Fagin 1974)

Queries in existential second-order logic $= \mathsf{NP}$.

Finite models in STλ and extensional completeness (1)

Goal: represent finite models for signature $\langle \mathcal{R}_1, \ldots, \mathcal{R}_p \rangle$ in STλ.

Idea: if $\mathcal{R}_i$ is k_i-ary, list of k_i-tuples,

$$\mathsf{Rel}_k[d, A] = (d^k \to A \to A) \to A \to A$$

(in the spirit of database theory: relation = set of records)

Now, what is is type d?

Finite models in STλ and extensional completeness (1)

Goal: represent finite models for signature $\langle \mathcal{R}_1, \ldots, \mathcal{R}_p \rangle$ in STλ.

Idea: if $\mathcal{R}_i$ is k_i-ary, list of k_i-tuples,

$$\mathsf{Rel}_k[d, A] = (d^k \to A \to A) \to A \to A$$

(in the spirit of database theory: relation = set of records)

Now, what is is type d?

$\to$ A *free type variable* in the type of the program.

Query terms $t : \mathsf{Rel}_{k_1}[d, A_1] \to \ldots \to \mathsf{Bool}$, with meta-level $\forall d$.

Morally equivalent to $t : (\exists d.\ \mathsf{Rel}_{k_1} \times \ldots) \to \mathsf{Bool}$.

Finite models in STλ and extensional completeness (1)

Goal: represent finite models for signature $\langle \mathcal{R}_1, \ldots, \mathcal{R}_p \rangle$ in STλ.

Idea: if $\mathcal{R}_i$ is k_i-ary, list of k_i-tuples,

$$\mathsf{Rel}_k[d, A] = (d^k \to A \to A) \to A \to A$$

(in the spirit of database theory: relation = set of records)

Now, what is is type d?

$\to$ A *free type variable* in the type of the program.
Query terms $t : \mathsf{Rel}_{k_1}[d, A_1] \to \ldots \to \mathsf{Bool}$, with meta-level $\forall d$.
Morally equivalent to $t : (\exists d.\ \mathsf{Rel}_{k_1} \times \ldots) \to \mathsf{Bool}$.

We also need to give an equality predicate ($\mathtt{Eq} : d \to d \to \mathsf{Bool}$), and a list of domain elements ($\mathsf{List}[d, A] = \mathsf{Rel}_1[d, A]$).
Define *query terms* as terms of type

$$\mathsf{Rel}_{k_1}[d, A_1] \to \ldots \to \mathsf{List}[d, A] \to (d \to d \to \mathsf{Bool}) \to \mathsf{Bool}$$

Finite models in STλ and extensional completeness (2)

To feed input to

$$t : \mathsf{Rel}_{k_1}[d, A_1] \to \ldots \to \mathsf{List}[d, A] \to (d \to d \to \mathsf{Bool}) \to \mathsf{Bool}$$

instantiate $d = o^n \to o$ (n = domain size).

The semantics of the input now has *size depending on n*
$\longrightarrow$ breaks finite automaton argument.

Finite models in STλ and extensional completeness (2)

To feed input to

$$t : \mathsf{Rel}_{k_1}[d, A_1] \to \ldots \to \mathsf{List}[d, A] \to (d \to d \to \mathsf{Bool}) \to \mathsf{Bool}$$

instantiate $d = o^n \to o$ (n = domain size).

The semantics of the input now has *size depending on n*
$\longrightarrow$ breaks finite automaton argument.

Which is enough to get all the expressivity we want!

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

Query terms in STλ *compute exactly* ELEMENTARY *queries over finite models.*

Proof.

ompleteness: encode Turing machines (as before).
Soundness: next slide. □

Functionality order and complexity (1)

Parameter controlling complexity: *functionality order*

$$\mathrm{ord}(\alpha \to \beta) = \max(\mathrm{ord}(\alpha) + 1, \mathrm{ord}(\beta))$$

Proposition

$\forall k \in \mathbb{N}\ \exists f(k) \in \mathbb{N}$ s.t. normalization of λ-terms with order $\leq k$ subterms is in $f(k)$-EXPTIME.

$\longrightarrow$ Soundness: each query term represents a f(max order in subterm)-EXPTIME query.

Functionality order and complexity (1)

Parameter controlling complexity: *functionality order*

$$\mathrm{ord}(\alpha \to \beta) = \max(\mathrm{ord}(\alpha) + 1, \mathrm{ord}(\beta))$$

Proposition

$\forall k \in \mathbb{N}\ \exists f(k) \in \mathbb{N}$ s.t. normalization of λ-terms with order $\leq k$ subterms is in $f(k)$-EXPTIME.

$\longrightarrow$ Soundness: each query term represents a f(max order in subterm)-EXPTIME query.

Suggests looking at *fixed order* terms to characterize k-EXPTIME classes for some k.

Functionality order and complexity (2)

Theorem (Hillebrand & Kanellakis)

The STλ *query terms*

$$t : \mathsf{Rel}_{k_1}[d, A_1] \to \ldots \to \mathsf{List}[d, A] \to (d \to d \to \mathsf{Bool}) \to \mathsf{Bool}$$

with $\mathrm{ord}(A_i) \leq 2k+1$ (*resp.* $2k+2$) *compute exactly the* k-EXPTIME (*resp.* k-EXPSPACE) *queries.*

- order 1 is P
- order 2 is PSPACE
- order 3 is EXPTIME
- order 4 is EXPSPACE

And so on. Unsatisfying point: $\mathrm{ord}(d)$ is counted as 0, while it should morally be 1 since eventually $d = o^n \to o$.

Functionality order and complexity (3)

Theorem (Hillebrand & Kanellakis)

The STλ *query terms*

$$t : \mathsf{Rel}_{k_1}[d, A_1] \to \ldots \to \mathsf{List}[d, A] \to (d \to d \to \mathsf{Bool}) \to \mathsf{Bool}$$

with $\mathrm{ord}(A_i) \leq 2k + 1$ (*resp.* $2k + 2$) *compute exactly the* k-EXPTIME (*resp.* k-EXPSPACE) *queries.*

So exponential height is roughly *half* the order, and we have time-space alternation. Same phenomenon:

Theorem (Terui, RTA'12)

Normalizing an STλ*-term of type* Bool *w/ order* $\leq r$ *subterms is*

- k-EXPTIME-*complete for* $r = 2k + 2$ (P-*complete for* $r = 2$)
- k-EXPSPACE-*complete for* $r = 2k + 3$ (PSPACE-*c. for* $r = 3$)

Functionality order and complexity (3)

Why half the order?
To simulate a k-EXPTIME TM, we use $\overline{n} : \mathsf{Nat}[T[S]]$.

- S = type of TM configurations, adding an exponential to the *space* used increments $\mathrm{ord}(S)$
- adding an exponential to the *number of iterations* increments $\mathrm{ord}(\mathsf{Nat}[T[S]])$ through the T part

Functionality order and complexity (3)

Why half the order?
To simulate a k-$\mathsf{EXPTIME}$ TM, we use $\overline{n} : \mathsf{Nat}[T[S]]$.

- S = type of TM configurations, adding an exponential to the *space* used increments $\mathrm{ord}(S)$
- adding an exponential to the *number of iterations* increments $\mathrm{ord}(\mathsf{Nat}[T[S]])$ through the T part

Syntactic normalization takes around $(\mathrm{order} + O(1))$-$\mathsf{EXPTIME}$, whereas would like $(\mathrm{order}/2 + O(1))$-$\mathsf{EXPTIME}$.
Instead, both theorems are proven by a mix of β-reduction and *semantic evaluation*.
(H&K: finite sets; Terui: Scott model of linear logic)

From STλ to Elementary Linear Logic (1)

To sum up:

- *Church encodings* of inputs restrict expressivity
- *Semantic evaluation* can prove this (and lots of other stuff)
- To overcome this, one can represent inputs as *finite models*

We will now see that these phenomena also occur in Elementary Linear Logic.

From STλ to Elementary Linear Logic (2)

Using a suitable type Str of Church-encoded bitstrings:

Theorem (Baillot, APLAS'11)

The proofs of $!\mathsf{Str} \multimap !!\mathsf{Bool}$ *in 2nd order elementary affine logic* with recursive types *decide exactly the languages in* P.

Recursive types are crucial for the above, as we show:

Theorem

The proofs of $!\mathsf{Str} \multimap !!\mathsf{Bool}$ *in 2nd order* ELL *decide exactly the* regular *languages.*

Proof idea: again, semantic evaluation, in a *finite semantics for 2nd order MALL* (whose existence is a new result!).

From $\mathsf{ST}\lambda$ to Elementary Linear Logic (3)

What do we get if we replace $!\mathsf{Str}$ by a encoding Inp of finite relational structures?

Proposition

All logarithmic space *queries can be computed by proofs of* $\mathsf{Inp} \multimap !!\mathsf{Bool}$.

Proved using descriptive complexity.

Conjecture

Proofs of $\mathsf{Inp} \multimap !!\mathsf{Bool}$ *decide* exactly *logarithmic space queries*.

Currently working on this!