Contributing to Higher Order Complexity: Outcomes and Likely Applications

Hugo Férée

September 26th, 2019

INSTITUT DE RECHERCHE EN INFORMATIQUE FONDAMENTALE

Université de Paris

Summary

Motivation, or How Did I Get Into Higher-order Complexity?

Computation as a Dialogue and How It Helps with Complexity

And now what?

Motivation, or How Did I Get Into Higher-order Complexity?

Type-two Theory of Effectivity

To compute over a space X we equip it with a surjection $\delta : R \hookrightarrow X$, where R is a space over which we already know how to compute.

$$\begin{array}{ccc} X & \xrightarrow{f} & X' \\ \uparrow_{\delta} & & \uparrow_{\delta'} \\ R & \xrightarrow{g} & R \end{array}$$

Type-two Theory of Effectivity

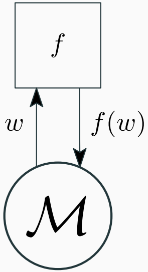
To compute over a space X we equip it with a surjection $\delta : R \hookrightarrow X$, where R is a space over which we already know how to compute.

For example:

- $R = \Sigma^*$ allows to represent discrete domains (integers, lists, graphs, etc.) but not uncountable ones
- $R = \Sigma^{\to}\Sigma*$ is enough to represent $\mathbb{R}, \mathcal{C}[0, 1]$, *etc*. "correctly".

Second-order Computations

In order to compute over $\Sigma^* \to \Sigma^*$, we use Oracle Turing Machines:



Definition

$F : (\Sigma^* \to \Sigma^*) \to \Sigma^*$ is computed by an oracle Turing machine $\mathcal{M}$ if for any oracle $f : \Sigma^* \to \Sigma^*$, $\mathcal{M}^f$ computes $F(f)$.

Second-Order Complexity

Definition (Time complexity)

The complexity of a machine is an upper bound on its computation time w.r.t the size of its input.

- ✓ size of a finite word
- ? size of an order 1 function

Second-Order Complexity

Definition (Time complexity)

The complexity of a machine is an upper bound on its computation time w.r.t the size of its input.

✓ size of a finite word

? size of an order 1 function

Definition (Size of a function)

The size of $f : \Sigma^* \to \Sigma^*$ is $|f| : \mathbb{N} \to \mathbb{N}$:

$$|f|(n) = \max_{|x| \leq n} |f(x)|.$$

Second-Order Polynomial Time

Definition (Second order polynomials)

$$P := c \mid X \mid Y\langle P\rangle \mid P + P \mid P \times P$$

Example

$$P(X, Y) = (Y\langle X \times Y\langle X + 1\rangle\rangle)^2$$

Definition ($\textsc{fptime}_2$)

Second order polynomial time computable function = computable by an OTM in second order polynomial time.

Actually, we can define many complexity classes: $\mathrm{NP}_2, \#\mathrm{P}_2, \ldots$

and the corresponding classes in analysis:

$\mathrm{NP}_{\mathbb{R}}, \#\mathrm{P}_{\mathbb{R}}, \mathrm{NP}_{\mathcal{C}[0,1]}, \#\mathrm{P}_{\mathcal{C}[0,1]}, \ldots$

Application: Complexity for Functions over Streams

Simple coinductive datatypes can be seen as first-order functions (watch out for details).

Theorem (F., Hainry, Hoyrup, Péchoux 2010)

The Implicit Computation Complexity technique called polynomial interpretations can be applied to lazy first-order rewriting systems with streams to characterise (a relevant notion of) polynomial time complexity.

Limits of First-order Representations

Once again, $R = \Sigma^* \to \Sigma^*$ may not always be the right representation space:

Theorem (F.-Hoyrup 2013)

If X is a non-σ-compact polish space with an admissible representation, then no representation $\delta : (\Sigma^* \to \Sigma^*) \hookrightarrow \mathcal{C}[X, \mathbb{R}]$ makes the complexity of the application function $Ap : \mathcal{C}[X, \mathbb{R}] \times X \to \mathbb{R}$ well-defined.

Example

TTE cannot express a meaningful notion of complexity for $\mathcal{C}[\mathcal{C}[0, 1], \mathbb{R}]$.

Towards a "Higher-order Theory of Effectivity"?

$$
\begin{array}{ccc}
X & \xrightarrow{f} & X' \\
\big\uparrow{\scriptstyle \delta} & & \big\uparrow{\scriptstyle \delta'} \\
\Sigma^* & \xrightarrow{g} & \Sigma^*
\end{array}
$$

Towards a "Higher-order Theory of Effectivity"?

$$\begin{array}{ccc} X & \xrightarrow{f} & X' \\ \uparrow_{\delta} & & \uparrow_{\delta'} \\ (\Sigma^* \to \Sigma^*) & \xrightarrow{g} & (\Sigma^* \to \Sigma^*) \end{array}$$

Towards a "Higher-order Theory of Effectivity"?

$$\begin{array}{ccc} X & \xrightarrow{f} & X' \\ \uparrow_{\delta} & & \uparrow_{\delta'} \\ (\Sigma^* \to \Sigma^*) \to \Sigma^* & \xrightarrow{g} & (\Sigma^* \to \Sigma^*) \to \Sigma^* \end{array}$$

Towards a "Higher-order Theory of Effectivity"?

$$\begin{array}{ccc} X & \xrightarrow{f} & X' \\ \uparrow_{\delta} & & \uparrow_{\delta'} \\ \sigma & \xrightarrow{g} & \tau \end{array}$$

Definition (Higher-order types)

$$\tau, \sigma := \mathbb{N} \mid \sigma \hookrightarrow \tau \mid \sigma \times \tau$$

Higher-order Computability?

- Kleene schemata
- Kleene associates
- Berry-Curien sequential algorithms
- …
- PCF (Scott, Plotkin)
  λ-calculus over $\mathbb{N}$ + fixpoint combinator.
  ✗ No simple underlying complexity notion.
- BFF (Cook, Urquhart)
  λ-calculus + FPTIME + $\mathcal{R}$ (2^{nd}-order bounded recursion)
  ✗ Defines only one complexity class (no EXPTIME, etc.)
  ✗ Misses some intuitively feasible functionals.

Basic Feasible Functionals

Definition (Cook & Urquhart (93), Mehlhorn (76))

$\textsc{bff} = \lambda + \textsc{fptime} + \mathcal{R}$, with:

$$\mathcal{R}(x_0, F, B, x). \begin{cases} x_0 \text{ if } x = 0 \\ t \text{ if } |t| \leq B(x) \\ B(x) \text{ otherwise.} \end{cases}$$

$$\text{where } t = F(x, \mathcal{R}(x_0, F, B, \lfloor \frac{x}{2} \rfloor)).$$

Theorem (Kapron & Cook 1996)

$\textsc{bff}_2$ is the class of functions computed by an oracle Turing machine in second-order polynomial time.

Feasible $\not\Longrightarrow$ BFF

Example (Irwin, Kapron, Royer)

$$f_x(y) = 1 \iff y = 2^x$$

$$\Phi, \Psi : ((\mathbb{N} \to \mathbb{N}) \to \mathbb{N}) \times \mathbb{N} \to \mathbb{N}$$

$$\Phi(F, x) = \begin{cases} 0 & \text{if } F(f_x) = F(\lambda y.0) \\ 1 & \text{otherwise.} \end{cases} \qquad \Phi \in \mathrm{BFF}_3$$

$$\Psi(F, x) = \begin{cases} 0 & \text{if } F(f_x) = F(\lambda y.0) \\ 2^x & \text{otherwise.} \end{cases} \qquad \Psi \notin \mathrm{BFF}_3$$

but Ψ is "as feasible as" Φ.

Computation as a Dialogue and How It Helps with Complexity

Computation as a Dialogue (First-order functions)

@machine, what is your value?

Computation as a Dialogue (First-order functions)

@machine, what is your value?

Machine is computing…

Computation as a Dialogue (First-order functions)

@machine, what is your value?

On which input?

Computation as a Dialogue (First-order functions)

@machine, what is your value?

On which input?

Input is computing…

Computation as a Dialogue (First-order functions)

@machine, what is your value?

On which input?

On input 10!

Computation as a Dialogue (First-order functions)

@machine, what is your value?

On which input?

On input 10!

Machine is computing…

Computation as a Dialogue (First-order functions)

@machine, what is your value?

On which input?

On input 10!

I'm worth 47 on that input!

Computation as a Dialogue (Second-order functions)

@machine, what is your value?

Computation as a Dialogue (Second-order functions)

@machine, what is your value?

Machine is computing…

Computation as a Dialogue (Second-order functions)

@machine, what is your value?

On which first-order input (let’s call it "f")?

Computation as a Dialogue (Second-order functions)

@machine, what is your value?

On which first-order input (let's call it "f")?

Input is computing…

Computation as a Dialogue (Second-order functions)

@machine, what is your value?

On which first-order input (let's call it "f")?

What do you want to know about f?

Computation as a Dialogue (Second-order functions)

@machine, what is your value?

On which first-order input (let's call it "f")?

What do you want to know about f?

Machine is computing…

Computation as a Dialogue (Second-order functions)

@machine, what is your value?

On which first-order input (let's call it "f")?

What do you want to know about f?

What is $f(1)$?

Computation as a Dialogue (Second-order functions)

@machine, what is your value?

On which first-order input (let's call it "f")?

What do you want to know about f?

What is $f(1)$?

Input is computing...

Computation as a Dialogue (Second-order functions)

@machine, what is your value?

On which first-order input (let's call it "f")?

What do you want to know about f?

What is $f(1)$?

It's 2. Anything else?

Computation as a Dialogue (Second-order functions)

@machine, what is your value?

On which first-order input (let's call it "f")?

What do you want to know about f?

What is $f(1)$?

It's 2. Anything else?

Machine is computing...

Computation as a Dialogue (Second-order functions)

@machine, what is your value?

On which first-order input (let's call it "f")?

What do you want to know about f?

What is $f(1)$?

It's 2. Anything else?

What is $f(4)$?

Computation as a Dialogue (Second-order functions)

@machine, what is your value?

On which first-order input (let's call it "f")?

What do you want to know about f?

What is $f(1)$?

It's 2. Anything else?

What is $f(4)$?

Input is computing...

Computation as a Dialogue (Second-order functions)

@machine, what is your value?

On which first-order input (let’s call it "f")?

What do you want to know about f?

What is $f(1)$?

It’s 2. Anything else?

What is $f(4)$?

It’s 7. Anything else?

Computation as a Dialogue (Second-order functions)

@machine, what is your value?

On which first-order input (let's call it "f")?

What do you want to know about f?

What is $f(1)$?

It's 2. Anything else?

What is $f(4)$?

It's 7. Anything else?

⋮

Computation as a Dialogue (Second-order functions)

@machine, what is your value?

On which first-order input (let's call it "f")?

What do you want to know about f?

What is $f(1)$?

It's 2. Anything else?

What is $f(4)$?

It's 7. Anything else?

⋮

Machine is computing…

Computation as a Dialogue (Second-order functions)

@machine, what is your value?

On which first-order input (let's call it "f")?

What do you want to know about f?

What is $f(1)$?

It's 2. Anything else?

What is $f(4)$?

It's 7. Anything else?

⋮

I know enough about f, I'm worth 74 on it!

Computation as a Dialogue (Third-order functions)

@machine, what is your value?

Computation as a Dialogue (Third-order functions)

@machine, what is your value?

Machine is computing…

Computation as a Dialogue (Third-order functions)

@machine, what is your value?

On which second-order input (let's call it F)?

Computation as a Dialogue (Third-order functions)

@machine, what is your value?

On which second-order input (let's call it F)?

Input is computing…

Computation as a Dialogue (Third-order functions)

@machine, what is your value?

On which second-order input (let's call it F)?

What do you want to know about it?

Computation as a Dialogue (Third-order functions)

@machine, what is your value?

On which second-order input (let's call it F)?

What do you want to know about it?

Machine is computing…

Computation as a Dialogue (Third-order functions)

@machine, what is your value?

On which second-order input (let's call it F)?

What do you want to know about it?

What is the value of F?

⋮

Computation as a Dialogue (Third-order functions)

@machine, what is your value?

On which second-order input (let's call it F)?

What do you want to know about it?

What is the value of F?

⋮

Input is computing...

Computation as a Dialogue (Third-order functions)

@machine, what is your value?

On which second-order input (let's call it F)?

What do you want to know about it?

What is the value of F?

⋮

F is equal to 74 on the input you just described!

Computation as a Dialogue (Third-order functions)

@machine, what is your value?

On which second-order input (let's call it F)?

What do you want to know about it?

What is the value of F?

⋮

F is equal to 74 on the input you just described!

Machine is computing…

Computation as a Dialogue (Third-order functions)

@machine, what is your value?

On which second-order input (let's call it F)?

What do you want to know about it?

What is the value of F?

⋮

F is equal to 74 on the input you just described!

What is the value of F?

⋮

Computation as a Dialogue (Third-order functions)

@machine, what is your value?

On which second-order input (let's call it F)?

What do you want to know about it?

What is the value of F?

⋮

F is equal to 74 on the input you just described!

What is the value of F?

⋮

Input is computing…

Computation as a Dialogue (Third-order functions)

@machine, what is your value?

On which second-order input (let's call it F)?

What do you want to know about it?

What is the value of F?

⋮

F is equal to 74 on the input you just described!

What is the value of F?

⋮

F is equal to 63 on the input you just described!

Computation as a Dialogue (Third-order functions)

@machine, what is your value?

On which second-order input (let's call it F)?

What do you want to know about it?

What is the value of F?

⋮

F is equal to 74 on the input you just described!

What is the value of F?

⋮

F is equal to 63 on the input you just described!

⋮

Computation as a Dialogue (Third-order functions)

@machine, what is your value?

On which second-order input (let's call it F)?

What do you want to know about it?

What is the value of F?

⋮

F is equal to 74 on the input you just described!

What is the value of F?

⋮

F is equal to 63 on the input you just described!

⋮

Machine is computing…

Computation as a Dialogue (Third-order functions)

@machine, what is your value?

On which second-order input (let's call it F)?

What do you want to know about it?

What is the value of F?

⋮

F is equal to 74 on the input you just described!

What is the value of F?

⋮

F is equal to 63 on the input you just described!

⋮

OK, I know enough about F, I'm worth 53 on it!

Game Semantics

It has (initially) nothing to do with complexity, but with programming language semantics.

Origin: provide a fully abstract semantics for PCF

Solution: (Hyland & Ong, Nickau, Abramsky):

- functions $\leftrightarrow$ strategies
- function application $\leftrightarrow$ confrontation of strategies

Arena

An arena is defined by as set of moves:

- own by either P and O
- which are either questions questions or answers
- some are initial questions
- they are connected by an enabling relation.

Arenas for finite types

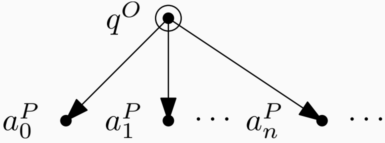


Figure 1: Arena for the base type $\mathbb{N}$

Arenas for finite types

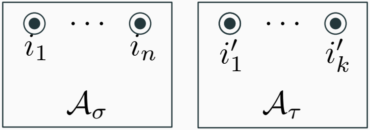


Figure 1: Arena $\mathcal{A}_{\sigma\times\tau}$ built from $\mathcal{A}_\tau$ and $\mathcal{A}_\sigma$

Arenas for finite types

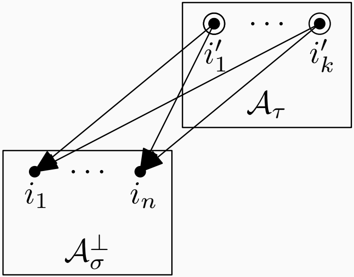


Figure 1: Arena $\mathcal{A}_{\sigma\to\tau}$ built from $\mathcal{A}_\tau$ and $\mathcal{A}_\sigma$

Arenas for finite types

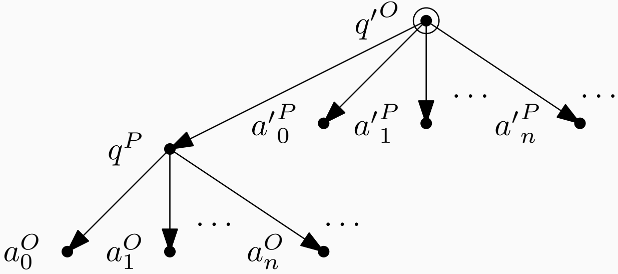


Figure 1: Arena for type $\mathbb{N} \to \mathbb{N}$

Arenas for finite types

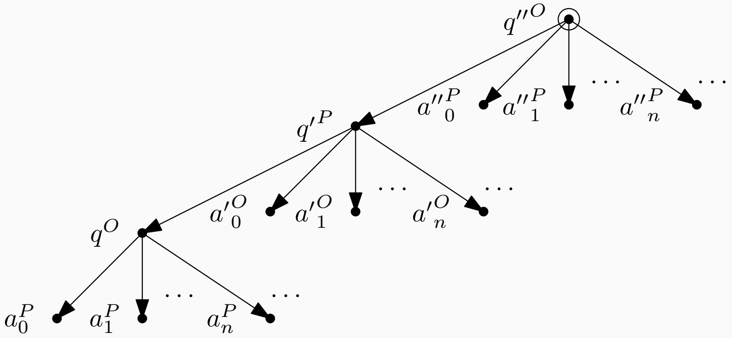


Figure 1: Arena for type $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$

Plays & Rules

Definition (Play)

A play is a list of named moves, i.e. $m[\alpha]$ ($m \in \mathcal{A}, \alpha \in \mathbb{N}$).

A play p is said to be:

- justified: every non initial move is justified by a previous move in p ;
- well-opened: there is only one initial move, at the beginning of p ;
- alternating: two consecutive moves belong to different protagonists ;
- strictly scoped: answering a question prevents further moves to be justified by this question ;
- strictly nested: Q/A pairs form a valid bracketing.

Innocent Strategies

Definition (Strategy)

A strategy is a partial function from plays to moves.

$$s(m_1, \ldots, m_k) = m_{k+1}$$

Definition (Innocent strategy)

A strategy is innocent if its output only depends on its current view of the play.

Confrontation

The confrontation of s (in $\mathcal{A}_{\tau\to\mathbb{N}}$) against s' (in $\mathcal{A}_\tau$) is:

- p starts with the initial question of $\mathcal{A}_{\tau\to\mathbb{N}}$
- we stop if s plays a final answer
- the play is successively extended this way:
  - p is extended with $s(p)$ (if defined)
  - p "contains" a sub-play p' in $\mathcal{A}_\tau$;
    p is extended with $s'(p')$ (+renaming)
- if reached, the final answer defines $s[s']$.

We also call the whole play the history of the confrontation (noted $H(s, s')$).

Confrontation

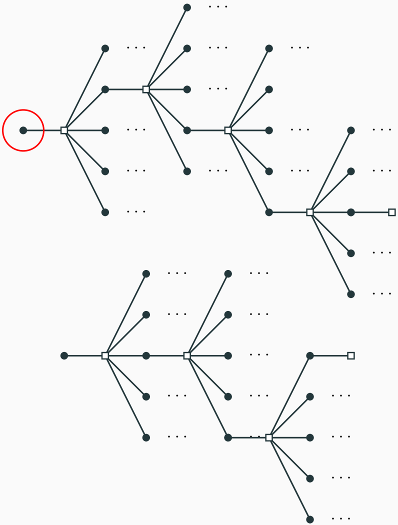


Figure 2: Confrontation of s (top) against s' (bottom)

Confrontation

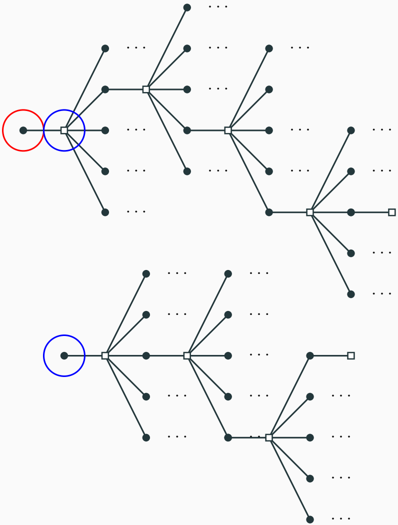


Figure 2: Confrontation of s (top) against s' (bottom)

Confrontation

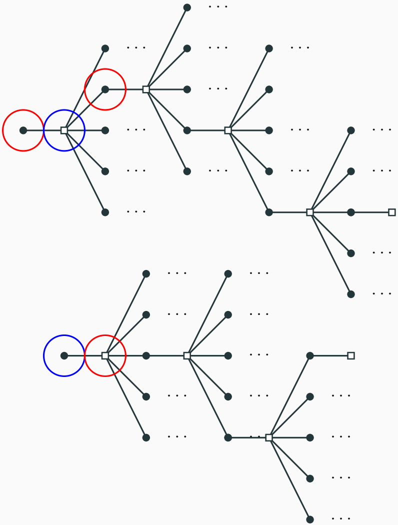


Figure 2: Confrontation of s (top) against s' (bottom)

Confrontation

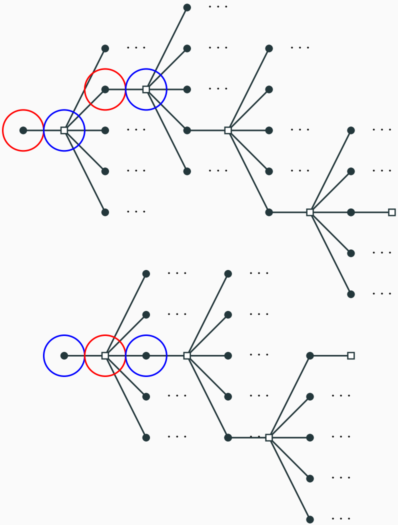


Figure 2: Confrontation of s (top) against s' (bottom)

Confrontation

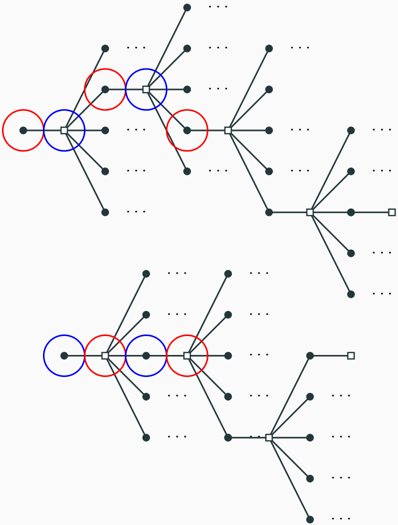


Figure 2: Confrontation of s (top) against s' (bottom)

Confrontation

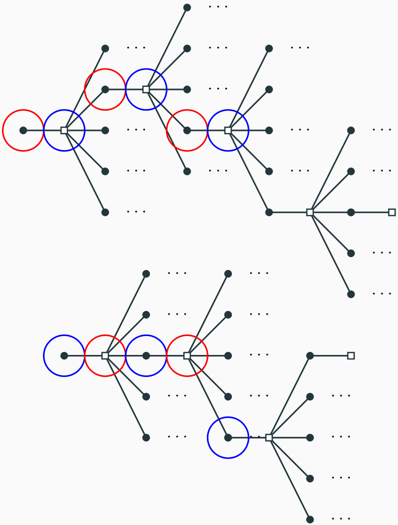


Figure 2: Confrontation of s (top) against s' (bottom)

Confrontation

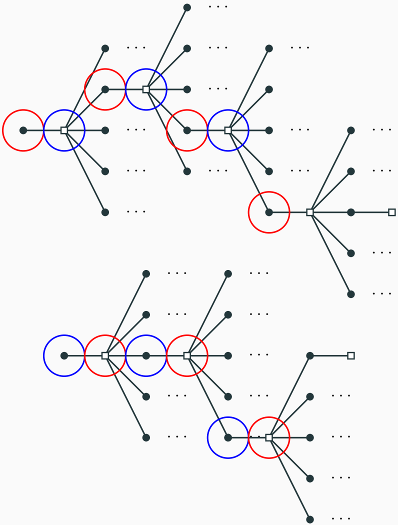


Figure 2: Confrontation of s (top) against s' (bottom)

Confrontation

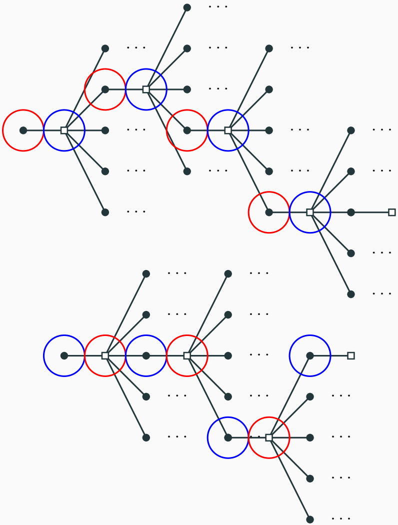


Figure 2: Confrontation of s (top) against s' (bottom)

Confrontation

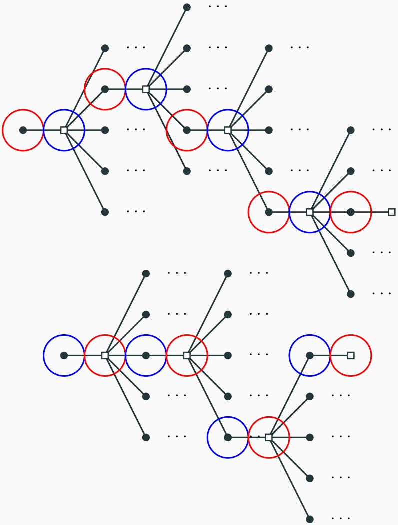


Figure 2: Confrontation of s (top) against s' (bottom)

Confrontation

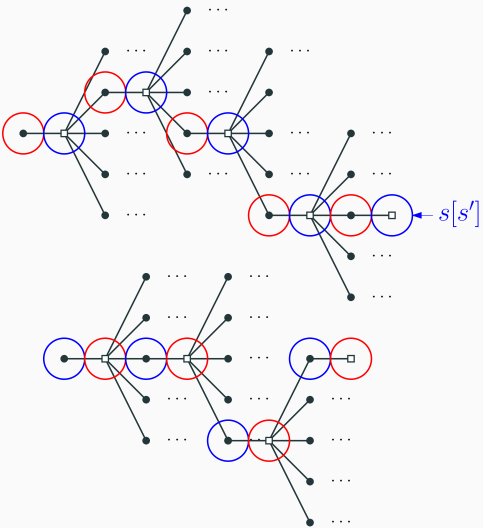


Figure 2: Confrontation of s (top) against s' (bottom)

Confrontation

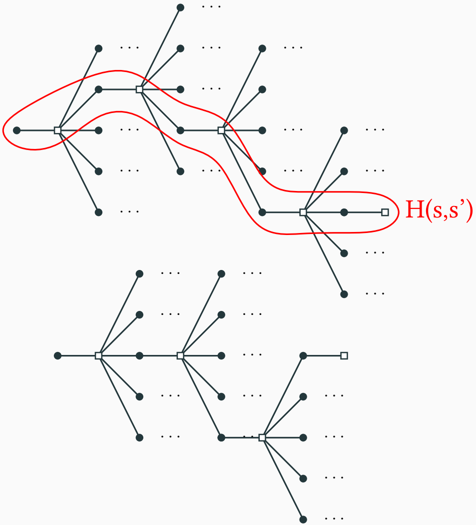


Figure 2: Confrontation of s (top) against s' (bottom)

Given a finite type τ, the corresponding game $\mathcal{G}_\tau$ is defined by innocent strategies playing justified, alternating, well-opened, strictly-nested, … plays in the arena $\mathcal{A}_\tau$.

Definition

Base case: If $s(q) = a_k$, then s represents $k \in \mathbb{N}$.

Recursive case: A strategy s in represents $F : \tau_1 \times \cdots \times \to \mathbb{N}$ if whenever $s_1, \ldots, s_n$ represent $f_1 : \tau_1, \ldots, f_n : \tau_n$, then

$s[s_1, \ldots, s_n]$ represents $F(f_1, \ldots, f_n)$

Effectivity

Our presentation of game semantics allows to define an explicit encoding of moves and names: for every game on a finite type τ,

- questions can be encoded by words of bounded size ;
- an answer representing $n \in \mathbb{N}$ (e.g. a_n) can be encoded by a binary word of size $\mathcal{O}(\log_2(n))$;
- names are integers $\rightarrow$ simple binary encoding ;
- this encoding can be extended to plays ;
- a strategy s can be represented by a partial function $\overline{s} : \Sigma^* \rightarrow \Sigma^*$

Computability and complexity

Definition

A strategy is s is computable if $\bar{s}$ is computable.

Computability and complexity

Definition

A strategy is s is computable if $\overline{s}$ is computable.

Definition attempt

A function is computable in time t, if it is represented by a strategy s such that $\overline{s}$ is computable in time t.

Computability and complexity

Definition

A strategy is s is computable if $\bar{s}$ is computable.

Definition attempt

A function is computable in time t, if it is represented by a strategy s such that $\bar{s}$ is computable in time t.

Theorem

Every computable function has a polynomial strategy.

Proof.

s can gain time by asking many useless questions.
$s(q', q, a_k, (q, a_k)^n) = a_{f(k)}$ if s can compute $f(k)$ in time n
$s(q', q, a_k, (q, a_k)^n) = q$ otherwise. □

Computability ~~and complexity~~

Definition

A strategy is s is computable if $\overline{s}$ is computable.

Definition attempt

A function is computable in time t, if it is represented by a strategy s such that $\overline{s}$ is computable in time t.

Theorem

Every computable function has a polynomial strategy.

Proof.

s can gain time by asking many useless questions.

$s(q', q, a_k, (q, a_k)^n) = a_{f(k)}$ if s can compute $f(k)$ in time n

$s(q', q, a_k, (q, a_k)^n) = q$ otherwise. □

Size of a strategy

Definition (Size of a play)

$=$ size of its binary encoding.

Definition (Size of a strategy)

The size S_s of s in $\tau \to \mathbb{N}$ is a bound on the size of the play H produced by the confrontation of s versus argument strategies:

$$S_s(b) = \sup\{|H(s, s')| \ : s' \in \mathcal{G}_\tau \wedge \ S_{s'} \preccurlyeq_\tau b\}$$

Additionally, for all $F, B : \tau \to \mathbb{N}$, $F \preccurlyeq_\tau B$ if:

$$\forall s' b, (S_{s'} \preccurlyeq_\tau b) \implies F(S_{s'}) \leq B(b)$$

Examples

Example

- $k \in \mathbb{N}$ has a strategy of size about $\log_2(k)$
  (plays are of the form: q, a_k)
- $g : \mathbb{N} \to \mathbb{N}$ has a strategy of size about
  $|g|(n) = \max_{|x| \leq n} |g(x)| + n$
  (plays are of the form: $q, q', a'_x, a_{g(x)}$)
- $F : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ has a strategy s whose size depends on
  its values: $S_s(b) \geq \max_{\{f\,:\,|f| \preccurlyeq b\}} |F(f)|$
  and on its modulus of continuity: $S_s(b) \geq n$
  whenever there are $f, g \preccurlyeq b$ such that
  $(\forall |y| \leq n, f(y) = g(y))$ and $F(f) \neq F(g)$.

Game machines

Definition (Game machine)

OTM which simulates a strategy:

- initial state $\leftrightarrow$ initial question
- oracle call $\leftrightarrow$ (encoded) player move
- oracle answer $\leftrightarrow$ (encoded) opponent move
- final state + tape's content $\leftrightarrow$ final answer

Proposition

s is simulated by a game machine $\iff$ s is computable.

Complexity

We can define the complexity of a strategy, and in particular:

$$\text{size} \preccurlyeq \text{complexity}$$

Theorem

size $\simeq$ smallest relativised complexity

$\forall s, \exists \mathcal{M}, \mathcal{O}, \mathcal{M}^{\mathcal{O}}$ *computes* s.

Definition

$f \in$ PCF is computable in time T if there is a game machine simulating an innocent strategy for f in time T.

Remark

If s represents a PCF *function $f : \tau$, then the size and complexity functions for s have type τ.*

Higher order polynomial time complexity

Definition (Higher type polynomials)

HTP $=$ simply-typed λ-calculus, with $+$ and $\times$.

Remark

- *Order 1* HTP *= usual polynomials.*
- *Order 2* HTP *= second order polynomials.*

Definition (POLY)

$f \in$ PCF is polynomial time computable ($f \in$ POLY), if it has a strategy computed by a (higher order) polynomial time machine.

Results

Proposition

For every finite type τ, the complexity of the identity function of type $\tau \to \tau$ is about $\lambda b.2 \cdot b$.

Similarly, composition, projections and expansion also have polynomial time complexity.

Proposition

Closure by composition If $b : \sigma$ and $B : \sigma \to \tau$ bound the complexity (resp. size) of $f : \sigma$ and $F : \sigma \to \tau$, then $B(b)$ bounds the complexity (resp. size) of $F(f)$.

Results

Proposition

Bounded recursion on notation is polynomial-time computable.

Proof.

It can be computed by $|x|$ iteration of F applied to x an input bounded by the size of B on x. Its complexity is bounded by:

$$\lambda n_0 \lambda G \lambda B \lambda n. \quad n \cdot G(n, B(n) + n_0) + n_0.$$

□

Size and Complexity

As it was already the case for first-order functions, the size functional is not computable in polynomial time.

Proposition

For any τ of order 1 or more, no polynomial-time computable function $F : \tau \to \tau$ satisfies:

$$\forall f, |f| \preccurlyeq F(f)$$

Results

Theorem

- $FPTIME = \mathrm{BFF}_1 = POLY_1$
- $FPTIME_2 = \mathrm{BFF}_2 = POLY_2$
- $\mathrm{BFF} \subseteq POLY$
- $\mathrm{BFF}_3 \subsetneq POLY_3$
- *POLY is stable by composition*

$\implies$ this complexity class is a good candidate for a generalisation of FPTIME at all finite types.

And now what?

Apply the Theory

We have a general notion of complexity for PCF, as well as a polynomial time complexity class for it.

- **Define** and study new complexity classes/hierarchies.
- **Obtain** new insight on first-order complexity classes
- **Apply** to other relevant sequential games
  The current framework **does not** require rules likes innocence or well bracketing
  (!) Complexity bounds for the same program in different settings need not be comparable!

Broaden the Theory

- We cannot currently deal with non-sequential games.
  Mainly, can we extend this to handle complexity for parallel computations (hard!)
  (I've heard that **Alexis Ghyselen** already took care of it!)
- Deal with sub-linear complexity classes
  There are several ways to implement names, which might affect this

Higher-Order Implicit Complexity

- Most existing Implicity complexity techniques only apply to first-order computations ;
- if not, they reduce down to first-order techniques ;
- and they can only express the complexity of first-order terms

We can now directly express the complexity of a higher-order function and so of any term/program that computes it.
So can we:

- **Develop/adapt** first-order ICC techniques to languages with higher-order features and characterise POLY?
  (rewriting systems, linear types, function algebras)
- **Derive and implement** actual complexity analysis tools for higher-order languages

Higher-order Representations

As initially motivated, we can use higher-order functions as names:

$$\begin{array}{ccc} X & \xrightarrow{f} & X' \\ \uparrow_{\delta} & & \uparrow_{\delta'} \\ \sigma & \xrightarrow{g} & \tau \\ \uparrow & & \uparrow \\ \mathcal{G}_\sigma & \overset{s_g}{\leadsto} & \mathcal{G}_\tau \end{array}$$

Remark

- *What is the minimal order to represent a given set X?*
- *If σ and τ are minimal representation spaces for X and Y, $\sigma \to \tau$ might not be the minimal one for $\mathcal{C}[X, Y]$.*