Contributing to Higher Order Complexity: Outcomes and Likely Applications

Hugo Férée
September 26th, 2019
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 $\delta : R \twoheadrightarrow X$, where $R$ is a space over which we already know how to compute.

\[
\begin{align*}
X \xrightarrow{f} X' \\
\uparrow \delta \quad \uparrow \delta' \\
R \xrightarrow{g} R
\end{align*}
\]
Type-two Theory of Effectivity

To compute over a space $X$ we equip it with a surjection $\delta : R \twoheadrightarrow X$, 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 $\mathbb{R}, C[0, 1]$, etc. "correctly".
In order to compute over \( \Sigma^* \rightarrow \Sigma^* \), we use **Oracle Turing Machines**:

\[
M_f(w) \rightarrow f(w)
\]

**Definition**

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

\[
|f|(n) = \max_{|x| \leq n} |f(x)|.
\]
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 (FP\textsc{TIME}_2 )**

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

Actually, we can define many complexity classes: \( \text{NP}_2, \#P_2, \ldots \)

and the corresponding classes in analysis:

\( \text{NP}_R, \#P_R, \text{NP}_C[0,1], \#P_C[0,1], \ldots \)
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.
Once again, $R = \Sigma^* \rightarrow \Sigma^*$ may not always be the right representation space:

**Theorem (F.-Hoyrup 2013)**

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

**Example**

TTE cannot express a meaningful notion of complexity for $C[C[0, 1], \mathbb{R}]$. 
Towards a "Higher-order Theory of Effectivity"?

\[
\begin{align*}
X & \xrightarrow{f} X' \\
\uparrow \delta & \quad \quad \quad \uparrow \delta' \\
\Sigma^* & \xrightarrow{g} \Sigma^*
\end{align*}
\]
Towards a "Higher-order Theory of Effectivity"?

\[ X \xrightarrow{\delta} X' \]

\[ (\Sigma^* \rightarrow \Sigma^*) \xrightarrow{\delta'} (\Sigma^* \rightarrow \Sigma^*) \]

\[ \text{Definition (Higher-order types)} \]

\[ \tau, \sigma := N \mid \sigma \rightarrow \tau \mid \sigma \times \tau \]
Towards a "Higher-order Theory of Effectivity"?

\[
\begin{align*}
X & \xrightarrow{\delta} X' \\
(\Sigma^* \to \Sigma^*) & \to \Sigma^* \\
\Sigma^* & \to \Sigma^*
\end{align*}
\]
Towards a "Higher-order Theory of Effectivity"?

\[
\begin{align*}
X & \xrightarrow{f} X' \\
\uparrow\delta & \quad \quad \uparrow\delta' \\
\sigma & \xrightarrow{g} \tau
\end{align*}
\]

**Definition (Higher-order types)**

\[
\tau, \sigma := \mathbb{N} \mid \sigma \rightarrow \tau \mid \sigma \times \tau
\]
Higher-order Computability?

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

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

\[
BFF = \lambda + \text{FPTIME} + \mathcal{R}, \text{ with:}
\]

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

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

Theorem (Kapron & Cook 1996)

\( BFF_2 \) is the class of functions computed by an oracle Turing machine in second-order polynomial time.
Example (Irwin, Kapron, Royer)

\[ f_x(y) = 1 \iff y = 2^x \]

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

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

\[ \Psi(F, x) = \begin{cases} 
0 & \text{if } F(f_x) = F(\lambda y.0) \\
2^x & \text{otherwise.} 
\end{cases} \quad \Psi \not\in \text{BFF}_3 \]

but \( \Psi \) is "as feasible as" \( \Phi \).
Computation as a Dialogue and How It Helps with Complexity
Computation as a Dialogue (First-order functions)

@machine, what is your value?

Machine is computing.
On which input?
Input is computing.
On input 10!
Machine is computing.
I'm worth 47 on that input!
@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…
@machine, what is your value?

On which input?

On input 10!

I’m worth 47 on that input!
@machine, what is your value?

Machine is computing...

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

Input is computing...

What do you want to know about f?

Machine is computing...

What is $f(1)$?

Input is computing...

It's 2. Anything else?

Machine is computing...

What is $f(4)$?

Input is computing...

It's 7. Anything else?

...

Machine is computing...

I know enough about f, I'm worth 74 on it!
Computation as a Dialogue (Second-order functions)

@machine, what is your value?

Machine is computing…

@machine is computing…
@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...
@machine, what is your value?

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

What do you want to know about f?
@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…
@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…
@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 \textit{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…
@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!
@machine, what is your value?

Machine is computing.

Input is computing.

Machine is computing.

Input is computing.

Machine is computing.

OK, I know enough about F, I'm worth 53 on it!
@machine, what is your value?

Machine is computing…

Machine is computing. . .

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

Input is computing. . .

What do you want to know about it?

Machine is computing. . .

What is the value of $F$?

Input is computing. . .

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

Machine is computing. . .

What is the value of $F$?

Input is computing. . .

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

Machine is computing. . .

OK, I know enough about $F$, I’m worth 53 on it!
@machine, what is your value?

On which second-order input (let’s call it $F$)?
@machine, what is your value?

On which second-order input (let’s call it $F$)?
@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.

Input is computing.

Machine is computing.

Input is computing.

Machine is computing.

OK, I know enough about $F$, I’m worth 53 on 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…

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

Machine is computing…

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

Machine is computing…

OK, I know enough about $F$, I’m worth 53 on it!
@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 $F$?

...
@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...
@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...

@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...
@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!
@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…
@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!
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
An arena is defined by a set of moves:

- own by either P and O
- which are either 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 $A_{\sigma \times \tau}$ built from $A_{\tau}$ and $A_{\sigma}$
Figure 1: Arena $A_{\sigma \rightarrow \tau}$ built from $A_{\tau}$ and $A_{\sigma}$
Arenas for finite types

Figure 1: Arena for type $\mathbb{N} \rightarrow \mathbb{N}$
Figure 1: Arena for type \((\mathbb{N} \to \mathbb{N}) \to \mathbb{N}\)
## Plays & Rules

<table>
<thead>
<tr>
<th>Definition (Play)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A play is a list of named moves, i.e. $m[\alpha]$ ($m \in A, \alpha \in \mathbb{N}$).</td>
</tr>
</tbody>
</table>

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

- $p$ starts with the initial question of $\mathcal{A}_{\tau \rightarrow \mathbb{N}}$
- 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 $H(s, s')$).
Figure 2: Confrontation of $s$ (top) against $s'$ (bottom)
Figure 2: Confrontation of $s$ (top) against $s'$ (bottom)
Confrontation

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

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

Figure 2: Confrontation of $s$ (top) against $s'$ (bottom)
Figure 2: Confrontation of $s$ (top) against $s'$ (bottom)
Figure 2: Confrontation of $s$ (top) against $s'$ (bottom)
Figure 2: Confrontation of $s$ (top) against $s'$ (bottom)
Figure 2: Confrontation of \( s \) (top) against \( s' \) (bottom)
Given a finite type \( \tau \), the corresponding game \( \mathcal{G}_\tau \) is defined by innocent strategies playing justified, alternating, well-opened, strictly-nested, … plays in the arena \( \mathcal{A}_\tau \).

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Base case:</strong> If ( s(q) = a_k ), then ( s ) represents ( k \in \mathbb{N} ).</td>
</tr>
<tr>
<td><strong>Recursive case:</strong> A strategy ( s ) in represents ( F : \tau_1 \times \cdots \times \rightarrow \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) ).</td>
</tr>
</tbody>
</table>
Our presentation of game semantics allows to define an explicit encoding of moves and names: for every game on a finite type $\tau$,

- 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 $\bar{s} : \Sigma^* \rightarrow \Sigma^*$.
Computability and complexity

Definition

A strategy is \( s \) is \textbf{computable} if \( \bar{s} \) is computable.
### Computability and complexity

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>A strategy is $s$ is <strong>computable</strong> if $\bar{s}$ is computable.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Definition attempt</th>
</tr>
</thead>
<tbody>
<tr>
<td>A function is <strong>computable in time</strong> $t$, if it is represented by a strategy $s$ such that $\bar{s}$ is computable in time $t$.</td>
</tr>
</tbody>
</table>
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. If \( s \) can compute \( f(k) \) in time \( n \),
\[
s(q', q, a_k, (q, a_k)^n) = a_{f(k)}
\]
otherwise,
\[
s(q', q, a_k, (q, a_k)^n) = q
\]
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.
\[
s(q', q, a_k, (q, a_k)^n) = a_{f(k)} \text{ if } s \text{ can compute } f(k) \text{ in time } n
\]
\[
s(q', q, a_k, (q, a_k)^n) = q \text{ otherwise.}
\]
**Size of a strategy**

**Definition (Size of a play)**

\[= \text{size of its binary encoding.}\]

**Definition (Size of a strategy)**

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

\[
S_s(b) = \sup \{|H(s, s')| : s' \in \mathcal{G}_\tau \land S_{s'} \preceq_\tau b\}
\]

Additionally, for all \( F, B : \tau \rightarrow \mathbb{N} \), \( F \preceq_\tau B \) if:

\[
\forall s' b, (S_{s'} \preceq_\tau b) \implies F(S_{s'}) \leq B(b)
\]
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: $S_s(b) \geq \max_{|f| \leq b} |F(f)|$
  and on its modulus of continuity: $S_s(b) \geq n$
  whenever there are $f, g \leq b$ such that
  $(\forall |y| \leq n, f(y) = g(y))$ and $F(f) \neq F(g)$. 
Game machines

Definition (Game machine)

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

\[ \text{size} \preceq \text{complexity} \]

### Theorem

\[ \text{size} \simeq \text{smallest relativised complexity} \]

\[ \forall s, \exists M, \mathcal{O}, M^\mathcal{O} \text{ computes } s. \]

### Definition

\( f \in \text{PCF} \) 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 \( \tau \).*
Definition (Higher type polynomials)

\[ \text{HTP} = \text{simply-typed } \lambda\text{-calculus, with } + \text{ and } \times. \]

Remark

- *Order 1* \( \text{HTP} = \text{usual polynomials.} \)
- *Order 2* \( \text{HTP} = \text{second order polynomials.} \)

Definition (POLY)

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

**Proposition**

*For every finite type* $\tau$, *the complexity of the identity function of type* $\tau \rightarrow \tau$ *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 $B : \sigma \rightarrow \tau$ *bound the complexity (resp. size) of* $f : \sigma$ *and* $F : \sigma \rightarrow \tau$, *then* $B(b)$ *bounds the complexity (resp. size) of* $F(f)$. 
## 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:

\[
\lambda n_0 \lambda G \lambda B \lambda n. \quad n \cdot G(n, B(n) + n_0) + n_0.
\]
As it was already the case for first-order functions, the size functional is not computable in polynomial time.

**Proposition**

*For any \( \tau \) of order 1 or more, no polynomial-time computable function \( F : \tau \rightarrow \tau \) satisfies:*

\[
\forall f, |f| \preceq F(f)
\]
## Results

### Theorem

<table>
<thead>
<tr>
<th>Theorem</th>
</tr>
</thead>
<tbody>
<tr>
<td>$FPTIME = BFF_1 = POLY_1$</td>
</tr>
<tr>
<td>$FPTIME_2 = BFF_2 = POLY_2$</td>
</tr>
<tr>
<td>$BFF \subseteq POLY$</td>
</tr>
<tr>
<td>$BFF_3 \subsetneq POLY_3$</td>
</tr>
<tr>
<td>$POLY$ is stable by composition</td>
</tr>
</tbody>
</table>

$\implies$ 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 like 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
As initially motivated, we can use higher-order functions as names:

\[ X \xrightarrow{f} X' \]
\[ \uparrow \delta \hspace{1cm} \uparrow \delta' \]
\[ \sigma \xrightarrow{g} \tau \]
\[ \uparrow \hspace{1cm} \uparrow \]
\[ G_\sigma \xrightarrow{sg} G_\tau \]

**Remark**

- What is the *minimal order* to represent a given set \( X \)?
- If \( \sigma \) and \( \tau \) are minimal representation spaces for \( X \) and \( Y \), \( \sigma \rightarrow \tau \) might not be the minimal one for \( C[X, Y] \).