Higher-order complexity in analysis

Hugo Férée, Mathieu Hoyrup

Higher-order complexity in analysis

Hugo Férée and Mathieu Hoyrup

June 13, 2013

Abstract

We present ongoing work on the development of complexity theory in analysis.

Kawamura and Cook recently showed how to carry out complexity theory on the space $\mathscr{C}[0,1]$ of continuous real functions on the unit interval. It is done, as in computable analysis, by representing objects by first-order functions (from finite words to finite words, say) and by measuring the complexity of a second-order functional in terms of second-order polynomials.

We prove that this framework cannot be directly applied to spaces that are not σ-compact. However, representing objects by higher-order functions (over finite words, say) makes it possible to carry out complexity theory on such spaces: for this purpose we develop the complexity of higher-order functionals. At orders above 3, our class of polynomial-time computable functionals strictly contains the class BFF of Buss, Cook and Urquhart.

1 Introduction

Computability theory was initiated with the definition of a computable function from $\mathbb{N}$ to $\mathbb{N}$. The theory has been extended in several directions, among which we mention higher-order computability theory and computable analysis. The first one is concerned with the definition and study of computable functionals of higher-order types (taking functions as inputs). The second one is concerned with the definition and study of computable objects that are encountered in mathematical analysis (real numbers, open sets, linear operators, etc.).

Extending complexity theory to higher-order types or in analysis are much more difficult tasks that have been only partially carried out. The class BFF_2 of *Basic Feasible Functionals* from $\mathbb{N} \to \mathbb{N}$ to $\mathbb{N} \to \mathbb{N}$, introduced in [17] and characterized in [9] is considered to be the natural class of polynomial-time computable type-two functionals. Basic Feasible Functionals above order 2 have been defined and characterized [2, 5, 3, 21, 7] but there is less evidence that they should be considered as the class of polynomial-time functionals. In analysis, Ko and Friedman's work, mostly gathered in [15], is a reference for complexity theory on the real numbers. Abstract approaches, applicable to any space in

some classes of topological spaces, have been developed by Weihrauch [**?**] and Schröder [20] and very recently by Kawamura and Cook [12]. These approaches differ in the following way. In the first approach, the polynomials measuring the execution time are ordinary (i.e., first-order) polynomials: while the input objects are infinite, their size has to be measured by a single natural number. This technical restriction imposes the space under consideration to be compact or σ-compact. The novelty of Kawamura and Cook's approach is to adopt the ideas of higher-order complexity, allowing for second-order polynomials: the size of an infinite object is a function from $\mathbb{N}$ to $\mathbb{N}$ and the execution time is a second-order function of the size function of the input. Their approach allows to have a sound complexity theory on larger spaces that are not σ-compact, as the space $\mathscr{C}[0,1]$ of continuous real functions over the unit interval.

In [11] it is asked whether complexity theory can be carried out to other spaces, in particular functions spaces. In this ongoing work, we provide both a negative and a positive result. The first one is that Kawamura and Cook's framework cannot be applied to larger spaces. Namely, if X is a Polish space that is not σ-compact then there is no representation of $\mathscr{C}(X,\mathbb{R})$ that is acceptable from a complexity perspective (Theorem 3.1). The positive result is that it is possible to overcome this limitation, at the price of increasing the order type of the functions used to represent objects. To summarize, our results tend to show that, while first-order functions admissibly represent objects from many spaces from a topological and computability-theoretic point of view, allowing to use higher-order functions is necessary and sufficient to admissibly represent objects from a complexity perspective.

In Section 2 we present the spaces on which we work: the QCB-spaces, on which computability and topology are closely related. In Section 3 we recall Kawamura and Cook's recent framework, which applies second-order complexity theory to analysis, and show the limitations (Theorem 3.1) of restricting to order 2, which suggests to develop complexity theory at higher orders. We present an outline of this development in Section 4. In Section 5 we apply this framework to analysis.

2 Computable analysis

Different directions have been followed to develop computability theory on other spaces. The most famous ones are domains [1] and represented spaces [22]. We will use the second approach. While originally the theory was developed for countably-based topological spaces only, it has been later extended to quotients of topological spaces, which happen to be exactly the spaces admitting admissible representations [19].

Definition 2.1. A ***QCB-space*** is the topological quotient of a countably-based topological space.

Every QCB-space is sequential, i.e. a function from a QCB-space to another topological space is continuous if and only if it is sequentially continuous

(i.e., maps converging sequences to converging sequences and commutes with the limit operation). The category **QCB** of QCB-spaces with continuous functions as morphisms enjoys remarkable properties. We will only be concerned with the fact that this category is cartesian closed. Precisely, the product and exponentiation are performed as follows:

- if X, Y are QCB-spaces then the cartesian product $X \times Y$ is a QCB-space with the topology induced by the following notion of convergence: (x_i, y_i) converge to (x, y) if x_i converge to x and y_i converge to y. The topology on $X \times Y$ is stronger than the product topology;
- if X, Y are QCB-spaces then the set $\mathscr{C}(X, Y)$ of continuous from X to Y is a QCB-space with the topology induced by the following notion of convergence: f_i converge to f if for every sequence x_i converging to x, $f_i(x_i)$ converge to $f(x)$.

QCB-spaces happen to be exactly the spaces on which computability theory can be extended. $\mathbb{B}$ denotes the Baire space of functions from $\mathbb{N}$ to $\mathbb{N}$ with the product topology.

Definition 2.2 ([22, 19])**.** A ***representation*** of a set X is a partial surjective map $\delta_X \subseteq: \mathbb{B} \to X$.

A representation δ_X of a topological space (X, τ) is ***admissible*** if:

- δ_X is continuous,
- δ_X is projective in the sense that for every partial continuous map $f : \mathbb{B} \to X$ there exists a partial continuous map $F : \mathbb{B} \to \mathbb{B}$ such that $f = \delta \circ F$.

Theorem 2.1. *[19] The QCB-spaces are exactly the topological spaces that admit an admissible representation.*

On QCB-spaces X, Y endowed with admissible representations δ_X, δ_Y, the continuous functions are exactly the functions $f : X \to Y$ that have a continuous realizer, i.e. a partial continuous function $F : \mathbb{B} \to \mathbb{B}$ such that $f \circ \delta_X = \delta_Y \circ F$. Hence the computable functions are automatically continuous and the continuous functions are exactly the functions that are computable relative to some oracle.

2.1 Higher-order functionals

Here we present an important class of QCB-spaces.

We define the finite types as follows:

$$\tau = \mathbb{N} \mid \tau_1 \times \ldots \times \tau_n \to \mathbb{N}$$

We interpret each type as a QCB-space, inductively on the structure of the type: $\mathbb{N}$ is interpreted as the set of natural numbers with the discrete topology and the QCB-space associated to $\tau_1 \times \ldots \times \tau_n \to \mathbb{N}$ is obtained by applying the product and exponentiation in the category **QCB**.

The elements of these QCB-spaces are exactly the Kleene-Kreisel continuous functionals (see [18]).

3 Complexity in analysis

The class BFF_2 of Basic Feasible Functionals defined by Mehlhorn [17] is considered as the natural class of polynomial-time computable type-two functional from $\Sigma^* \to \Sigma^*$ to $\Sigma^* \to \Sigma^*$, as it enjoys many desirable properties and admits several characterizations [9, 10, 8]. Kawamura and Cook [12] made use of this class to define the notion of polynomial-time computable function between represented spaces.

For technical reasons, they use as representation space the subspace Reg of length-monotonic functions from Σ^* to Σ^* (where Σ is a finite alphabet). For the purpose of the current section, it makes no difference. For us a representation starts from $\mathbb{B}$. The size of $n \in \mathbb{N}$ is the length of its binary expansion and the size of $f : \mathbb{N} \to \mathbb{N}$ is $|f|(n) = \max_{|p| \leq |n|} |f(p)|$.

Definition 3.1. Let (X, δ_X) and (Y, δ_Y) be represented spaces. A function $f : X \to Y$ is ***polynomial-time computable*** if there is a polynomial-time computable functional $F : \mathbb{B} \to \mathbb{B}$ such that $f \circ \delta_X = \delta_Y \circ F$.

Working on compact or σ-compact spaces allows for the development of first-order complexity theory, i.e. where the complexity of functions is expressed in terms of ordinary, first-order polynomials [20, 16]. The novelty of [12] was to develop complexity theory on spaces that are not σ-compact, like $\mathscr{C}([0,1], \mathbb{R})$. Kawamura [11] asks to which spaces X, Y (in place of $[0,1]$ and $\mathbb{R}$ respectively) their approach can be extended. More precisely, for which spaces X and Y is there an admissible representation of $\mathscr{C}(X, Y)$ which makes $\text{Eval}_{X,Y} : \mathscr{C}(X, Y) \times X \to Y$ polynomial-time computable and that "polynomially reduces" to any such representation. Here we show that even a weakening of the first condition cannot be satisfied when X is a Polish space that is not σ-compact: for any representation of $\mathscr{C}(X, Y)$, the time complexity of $\text{Eval}_{X,Y}$ is not even well-defined!

Definition 3.2. Let X, Y be QCB-spaces together with admissible representations. A computable function $f : X \to Y$ has ***well-defined time complexity*** if there is an oracle Turing machine M and a total continuous function $T : \mathbb{B} \times \mathbb{N} \to \mathbb{N}$ such that for each representation s of some $x \in X$, M^s computes a representation of $f(x)$, and $M^s(u)$ runs in time bounded by $T(|s|, |u|)$.

Observe that if f is polynomial-time computable then its time-complexity is well-defined: T is a second-order polynomial. In a similar way, one could define exponential-time computable functions: any such function would have well-defined time complexity.

Theorem 3.1. *Let X be a Polish space that is not σ-compact and $Y = \mathbb{R}$. There is no representation of $\mathscr{C}(X, \mathbb{R})$ making the time complexity of $Eval_{X,\mathbb{R}} : \mathscr{C}(X, \mathbb{R}) \times X \to \mathbb{R}$ well-defined.*

Proof. Let $\delta \subseteq : \mathbb{B} \to \mathscr{C}(X, \mathbb{R})$ be such a representation. The composition of Eval with δ gives partial continuous function form $\mathbb{B} \to \mathscr{C}(X, \mathbb{R})$ which is bounded by

a total continuous function $T : \mathbb{B} \to \mathscr{C}(X, \mathbb{R})$. We define a continuous function $F : X \to \mathbb{R}$ which is not in the image of δ. By a theorem of Hurewicz ([13], Theorem 7.10, p. 39) and the assumption on X, $\mathbb{B}$ is homeomorphic to a closed subset C of X, let $\Phi : \mathbb{B} \to C$ be such an homeomorphism. We first define F on C by $F(\Phi(s)) = T(s)(\Phi(s)) + 1$ and extend it to X by the Tietze extension theorem. If $F = \delta(s)$ for some s then $F(\Phi(s)) \leq T(s)(\Phi(s))$ which contradicts the definition of F. □

Particular examples of such spaces X are the Baire space $\mathbb{B}$ and the space $\mathscr{C}([0,1], \mathbb{R})$. Even if the complexity of continuous functions from X to $\mathbb{R}$ can be defined, it does not seem possible to carry out a complexity theory where these objects are first-class citizens, i.e. are objects of some represented space $\mathscr{C}(X, \mathbb{R})$.

Intuitively, the Baire space is not large enough to represent objects of larger spaces in a way that respects the size of the objects. An object contains so much information that when encoding it into a sequence of natural numbers, one has to postpone the information very far in the sequence so the time needed to reach a piece of information is arbitrarily larger than the size of the representation, and cannot be bounded by a continuous function of the size of the representation. Observe that replacing $\mathbb{B}$ with $\Sigma^* \to \Sigma^*$ or Reg does not solve the problem, as 3.1 can be adapted to those cases.

We now show that allowing larger representation spaces, namely higher-order functions spaces over $\mathbb{N}$, one can carry out complexity theory for larger spaces.

4 Higher-order complexity

As mentioned in the introduction, there is already a notion of polynomial-time computable functional at any finite type, namely the Basic Feasible Functionals (BFF_i). However this class is unsatisfactory in the sense that it misses functionals that are intuitively feasible, which is illustrated by the following example taken from [7] (Appendix A).

Example 1. Let f_∞ be the zero function of type $\mathbb{N} \to \mathbb{N}$, and for all $x \in \mathbb{N}$, f_x the function of same type such that $f_x(2^x) = 1$ and $f_x(y) = 0$ otherwise. Now we can define two functionals of order 3:

$$\Phi(F, x) = \begin{cases} 0 & \text{if } F(f_x) = F(f_\infty) \\ 1 & \text{otherwise.} \end{cases} \qquad \Psi(F, x) = \begin{cases} 0 & \text{if } F(f_x) = F(f_\infty) \\ 2^x & \text{otherwise.} \end{cases}$$

First, note that f_x is polynomial time computable, even relatively to x. So evaluating these two functionals is just a matter of evaluating F on two polynomial time computable functions and comparing the results, so this comparison is feasible. Indeed, Φ is a basic feasible functional. On the other hand, Ψ is not in BFF_3 since the size of the answer is exponential in the size of $|x|$ and the computation time of Ψ cannot be bounded by a polynomial in the size of x and the size of F (as a function). Nevertheless, Ψ is intuitively feasible (as pointed

out by the authors of [7]). The idea is that if F distinguishes $f(x)$ and f_∞, then during these evaluations, F had to ask for the value of its argument at input 2^x (since it is the only point where they differ). In this case, the time taken for this computation is at least 2^x, so writing 2^x as an output at most doubles the computation time between Φ and Ψ. The reason why Ψ is not in BFF_3 (and other previous notions of higher order feasibility) is that the evaluation of $|F|$ is seen as a black box where only the size of the input and the output matters whereas the modulus of continuity is not taken into account.

We will show in example 7 that the Ψ functional is indeed polynomial time computable in our sense.

This behavior also appears in computable analysis and the following example describes a real functional which is also intuitively feasible for similar reasons.

Example 2.

$$\Gamma(F, n) = \prod_{0 \leq i \leq 2^n} (1 + |F(h_{i,n})|)$$

where $h_{i,n}$ is the peak function of height 1 and width 2^{-n}.

This functional might not seem feasible since it requires to evaluate F on an exponential number of points. This idea is reinforced by the fact that Γ cannot be realized by a BFF_3 functional (which will be made formal in Section 5). Indeed, if we set F to be the infinite norm $||.||_\infty$, the size of $\Gamma(||.||_\infty, n) = 2^{2^n}$ is not bounded by a polynomial in $||.||_\infty$ and n.

However, there is a feasible way of computing Γ. Here are a sketch of this procedure and partial justifications for its feasibility. Compute F on the zero function (say, with precision 2^{-2n}). During this computation, F only makes a finite number of queries to its argument. Then, if a function h is small at these points, then $F(h)$ is small too. This way, we only need to compute F on the h_i which are close to these points to compute Γ. The evaluation of F on the zero function and on each h_i can be made in polynomial time. The only reason for this procedure not to be feasible is the number of h_i to evaluate, but this number already appears during the computation of F on the zero function. Then, roughly speaking, the time to compute $\Gamma(F)$ is about the time to compute $F(\mathbf{0})$ multiplied by the time to compute F on one of the h_i, which is polynomial in the computation time of F. In particular, if F is polynomial time computable, then the number of h_i to evaluate is bounded by a polynomial in n.

4.1 Notations & definitions

We define the level of finite types as follow: $\mathbb{N}$ has level 0, and $\tau_1 \times \cdots \times \tau_n \to \mathbb{N}$ has level $1 + \max_i \text{level}(\tau_i)$. In the following we may write, by abuse of language, that τ has type n instead of level n.

$|.|$ will denote the size of the binary encoding of integers or finite words.

We will further need to denote each occurrence of $\mathbb{N}$ in $\tau_1 \times \cdots \times \tau_n \to \mathbb{N}$: The only occurrence in $\mathbb{N}$ is denoted by the empty word ε. If a is a code for an occurrence in τ_i, then this occurrence is denoted by $i.a$ in $\tau_1 \times \cdots \times \tau_n \to \mathbb{N}$.

Definition 4.1. The rightmost occurrence of $\mathbb{N}$ in $\tau_1 \times \cdots \times \tau_n \to \mathbb{N}$ is positive, and any positive (resp. negative) occurrence of $\mathbb{N}$ in τ_i is negative (resp. positive) in $\tau_1 \times \cdots \times \tau_n \to \mathbb{N}$.

Note that it is equivalent to say that an occurrence is positive if and only if its code is a list of even length.

Example 3. In the type $(\mathbb{N} \times \mathbb{N} \to \mathbb{N}) \to \mathbb{N}$, the occurrences are: $\varepsilon, 1, 1.1$ and 1.2.

In the following we will use **indexed occurrences**: if $i_0 \dots i_k$ is an occurrence in τ, and $n_0, \dots n_k$ are integers, then $i_0^{n_0} \dots i_k^{n_k}$ is an indexed occurrence. For simplicity, we might omit the indexes when they are equal to 0.

4.2 Higher order strategies

A type 1 machine (i.e., a classical Turing machine) takes integers as input, so a type 2 machine must be able to evaluate its type 1 argument by providing it with integers. This is precisely the idea behind the oracle Turing machine model defined in [10] for type 2 functionals. It can be seen as a mechanism where the machine asks questions and waits for the device computing its argument to answer.

Then, naturally, a type 3 machine should be able to evaluate its type 2 argument the same way. Thus this argument should be, in turn, able to ask questions and wait for the machine to answer them. This way, we can see the interaction between a machine of arbitrary type with its arguments as a dialogue where both can ask and answer questions to each other.

We describe the dialogue between a function of type $\tau_1 \times \cdots \times \tau_n \to \mathbb{N}$ and its arguments of type $\tau_1, \dots, \tau_n$ as a game, inspired by the Game of Higher Types, used by Hyland and Ong [6] to provide a full abstraction for PCF.

Definition 4.2. A **move** in the game τ is either a question $?_a$ or an answer $!_a(v)$, where a is an indexed occurrence in τ and v an integer.

A **play** is a list of moves, with a few restrictions:

- The moves at even (resp. odd) position are called the opponent's (resp. player's) moves.
- The first move of a game is always $?_\varepsilon$.
- Each player's (resp. opponent's) move (except the first one) must be justified by a previous opponent's (resp. player's) question this way:
  - a question $?_{a.i^n}$ is justified by a previous open question $?_a$ and n must be minimal such that $?_{a.i^n}$ does not appear previously in the play
  - an answer $!_a(v)$ is justified by a question $?_a$ and we then say that this question is closed
- We say that the game is over if the last move answers the initial $?_\varepsilon$ question (it is then of the form $!_\varepsilon(v)$).

Note that the player (resp. opponent) only asks questions at negative (resp. positive) occurrences and answers to positive (resp. negative) questions.

Definition 4.3. A **strategy** s is a partial function which, given a play p of odd length, outputs a valid move $s(p)$, i.e. such that $(p, s(p))$ is still a valid play. We write $s(p) = \bot$ when the strategy is not defined.

Definition 4.4. Within type $\tau_1 \times \cdots \times \tau_n \to \mathbb{N}$, if $1 \leq i \leq n$ and $k \in \mathbb{N}$, view_i^k is defined this way on plays:

- $\text{view}_i^k(p, ?_{i^k.a}) = \text{view}_i^k(p), ?_a$
- $\text{view}_i^k(p, !_{i^k.a}(v)) = \text{view}_i^k(p), !_a(v)$
- $\text{view}_i^k(p, m) = \text{view}_i^k(p)$ otherwise.

In other words, we take the moves in p concerning an indexed occurrence beginning with i^k and remove i^k from this numbered occurrence.

More generally, we define $view(h)$ as $view_i^k(h)$ if h ends with a move concerning i^k (i.e. $?_{i^k.a}$ or $!_{i^k.a}$).

Definition 4.5. A **game** is the interaction between several strategies. If s is a strategy for $\tau_1 \times \cdots \times \tau_n \to \mathbb{N}$ and for all i, s_i is a strategy for τ_i, then the game between s and $s_1, \ldots, s_n$ is defined this way:

The initial play is $p = ?_\varepsilon$. While s has not answered the initial question (i.e the last element of p is of the form $!_\varepsilon(v)$), look at the next move $s(p)$ of s and add it to the play (p becomes $p, s(p)$). If $s(p)$ concerns i^k, then look at the next move of the i^{th} argument (namely $s_i(\text{view}(p, s(p)))$). If it is a question $?_a$, add $?_{i^k.a}$ to the play and if it is an answer $!_a(v)$, add $!_{i^k.a}(v)$ to the play.

When an answer of the form $!_\varepsilon(v)$ appears, the game terminates and v is called the **result of the game** and is denoted by $s[s_1, \ldots s_n]$. If at some point $s(p) = \bot$ or $\text{view}(p, s(p)) = \bot$, the game is interrupted and the result is undefined. $s[]$ is then a partial function mapping strategies to integers.

The final play is called the **history of the game** and is denoted by $H(s, s_1, \ldots, s_n)$.

In this definition, we use view to build the argument of a strategy. Thus we have to prove that in this case, the provided argument is a valid play.

Proposition 4.1. *At each step of the game, the current play p is valid for the game in τ, and for every (i, k), $view_i^k(p)$ is a valid play for the game in τ_i.*

Remark 4.1. There are two ways for a game not to terminate: one of the strategies cannot play (is not defined) or there are an infinite number of moves.

We define the representability of a function with a strategy by induction on the order of the function.

Definition 4.6. A partial function f of type $\tau_1 \times \cdots \times \tau_n \to \mathbb{N}$ is represented by a strategy s if for $(f_1, \ldots, f_n)$ in the domain of f and for every strategies $s_1, \ldots, s_n$ representing $f_1, \ldots f_n$ respectively, $s[s_1, \ldots s_n] = f(f_1, \ldots f_n)$.

This implies that an integer k, as a function of level 0, is represented by the only strategy s verifying $s(?_\varepsilon) = !_\varepsilon(k)$ and undefined otherwise.

Remark 4.2. A function $f : \mathbb{N} \to \mathbb{N}$ can be represented by the following strategy:

- $s(?_\varepsilon) = ?_1$
- $s(?_\varepsilon, ?_1, !_1(v)) = !_\varepsilon(f(v))$

Theorem 4.1. *A function $f : \tau_1 \times \ldots \times \tau_n \to \mathbb{N}$ is continuous if and only if it is represented by a strategy.*

Proof idea. Strategies easily correspond to associates as defined by Kleene [14]. The continuous functionals are exactly the functionals that admit an associate. □

Here is another way to present strategies:

Definition 4.7 (Execution tree)**.** Given a strategy s, we define the associated execution tree as an infinite tree with labeled nodes and edges. The nodes are labeled with the strategy moves, and the edges with its possible opponent's moves:

- The root node has no label and leads to an edge labeled with $?_\varepsilon$
- If the path from the root to an edge is labeled with p, then the node at the end of the edge is labeled with $s(p)$.
- If the path from the root to a node is labeled with p and the last move of p concerns i^k, then the outputting edges from this node are labeled with all the possible moves of an opponent s_i on $view_i^k(p)$.

We can also define the execution tree corresponding to a subset of opponent's strategies.

Remark 4.3. The interaction between s and its input strategies is always finite if and only if the corresponding execution tree has only finite branches.

4.3 Complexity

For type 1 functions, complexity is defined as the run time of a machine with respect to the size of the inputs. The inputs are now strategies, and we need to define their size before defining complexity.

Let us assume that we have a reasonable encoding *enc* of moves and plays as finite words. Then, a strategy can be seen as function from binary words to binary words, so we can say that a strategy is computable if is computable as a type 1 function, and that a functional is computable if it is represented by a computable strategy.

Definition 4.8 (Size of a strategy)**.** We define inductively the size of a strategy and the set of strategies of bounded type. The size of a strategy is a function of the same type. If s is a strategy over $\tau_1 \times \cdots \times \tau_n \to \mathbb{N}$, then its size is:

$$S_s(b_1, \dots, b_n) = \max_{(s_1, \dots s_n) \in K_{b_1} \times \cdots \times K_{b_n}} |enc(H(s, s_1, \dots s_n))|$$

Where K_b denotes the set of strategies s over type $\tau_1 \times \cdots \times \tau_n \to \mathbb{N}$ whose size is bounded by b with respect to the pointwise ordering verifying: whenever $s(p)$ is defined, then there exists strategies $(s_1, \dots s_n) \in K^{\tau_1} \times \cdots \times K^{\tau_n}$ such that p is a prefix of $H(s, s_1, \dots s_n)$ (where K^τ is the union K_b with b of type τ). Finally, call K the union of the K^τ, for every finite type τ.

Remark 4.4.
- In particular, the size of a strategy of type $\mathbb{N}$ is (roughly) the size of the encoding of the integer it computes.
- For every strategy s in $K^{\tau_1 \times \cdots \times \tau_n \to \mathbb{N}}$, $s[]$ is total on $K^{\tau_1} \times \cdots \times K^{\tau_n}$ and s is entirely described by its behavior on $K^{\tau_1} \times \cdots \times K^{\tau_n}$. In other words, the set of paths in the execution tree of s is exactly

$$\{H(s, s_1, \dots s_n) \mid (s_1, \dots s_n) \in K^{\tau_1} \times \cdots \times K^{\tau_n}\}$$

Theorem 4.2. *$S_s(b_1, \dots, b_n)$ is defined if and only if the execution tree of s restricted to strategies in $K_{b_1} \times \cdots \times K_{b_n}$ is finite.*

Remark 4.5. Note that the size of a strategy of type 0 representing an integer n is $Size(?_\varepsilon, !_\varepsilon(n)) = \mathcal{O}(|n|)$. So the size of a strategy of type 0 is close to the size of the integer it computes.

Simlarly, the size of a strategy computing a function of type 1 as described in remark 4.2 is roughly bounded by: $n \mapsto n + \max_{|k| \leq n} |f(k)|$, which is almost the definition of the size used in BFF (defined in [4] and characterized in [3]).

The size of strategies of types 0 and 1 match in some sense the size of the functions they represent.

Example 4. Let $\Phi : ((\mathbb{N} \to \mathbb{N}) \to \mathbb{N}) \times \mathbb{N} \to \mathbb{N}$ be defined by:

$$\Phi(F, n) = F(\lambda x.n)$$

A strategy to compute Φ is to query n, and then query F by playing the role of a strategy computing $\lambda x.n$. An evaluation of this strategy could be $?_\varepsilon, ?_2, !_2(n), ?_1, !_1(v), !_\varepsilon(v)$ if the strategy for F immediately answers v (thus F is constant), or of the form $?_\varepsilon, ?_2, !_2(n), (?_1, ?_{1.1}, !_{1.1}(n), ?_{1.1^2}, !_{1.1^2}(n), \dots, !_1(v)), !_\varepsilon(v)$, where the part between parenthesis corresponds to the evaluation of the strategy of F on the strategy of $\lambda x.n$. The size of the latter is bounded by $\lambda y.|n| + C$ (for a constant C), so the size of this evaluation is bounded by $|F|(\lambda y.|n| + C)$ by definition. This also bound the size of the last answer $(!_\varepsilon(v))$, so finally, the size of the strategy for Φ is bounded by $\lambda G.\lambda x.2 * G(\lambda y.x + C) + C'$ for some constant C'.

Now that we have defined the size of our inputs, we need a notion of computation time. This can be achieved by defining a model of machines simulating strategies, generalizing the oracle Turing machine model (for type 2 functionals). For sake of simplicity, we only sketch this definition here. A higher order

Turing machine over the finite type t is an oracle Turing machine with one special state for each occurrence of $\mathbb{N}$ in t and special tapes for indexes and answers. The machine can write to these tapes and enter the special states to simulate questions and answers for its arguments. Then, the corresponding argument strategy (or machine) can reply the same way. The cost of such an answer is the size of the simulated move. Finally, the complexity of such a machine is a bound on its computation time given a bound on the size of the input strategies.

4.4 Polynomial time complexity

To define a class of higher type feasible functionals we need a notion of polynomial bound for higher type time bounds. There is already such a bound to characterize the basic feasible functionals of type 2 with oracle Turing machines.

$$P := c \mid X \mid P + P \mid P \times P \mid Y(P)$$

where X is a type 0 variable and Y is a type 1 one.

This notion has been extended to all finite types in [7].

Definition 4.9 (Higher type polynomials)**.** The higher order polynomials are the terms of HTP, the simply-typed lambda calculus over $\mathbb{N}$ with special terms $+$ and $*$ of type $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$.

Definition 4.10. We say that a higher order functional is computable in polynomial time if it is computed by a machine whose running time is bounded by a higher order polynomial.

Example 5. The evaluation operator $\text{Eval} : (\tau_1 \times \ldots \times \tau_n \to \mathbb{N}) \times \tau_1 \times \ldots \times \tau_n \to \mathbb{N}$ is polynomial-time computable. Indeed, it can be computed by simulating the progress of the game between its arguments as follows: query the first argument on $?_\varepsilon$ and while the game is not over, read the provided move, translate and copy it to the corresponding argument strategy, wait for its answer, and translate it back to the main strategy (as it is done in definition 4.5). The computation time is then about the size of the main strategy applied to the size of its arguments, that is to say $T(t, t_1, \ldots, t_n) = t(t_1, t_n)$. As a consequence, the class of polynomial time computable functionals is stable with respect to composition.

Example 6. This definition of higher type polynomials coincides with the usual polynomials at type 1 and with the second order polynomials (used in [9] to characterize BFF_2). Additionally , the induced complexity class matches the standard classes at types 1 and 2 (that is $\text{FP}{=}\text{BFF}_1$ and BFF_2). More generally, every functional belonging to BFF is polynomial-time computable, but as the next example shows, the other direction does not hold.

Example 7. Now that we have the formalism of strategies we can show that the Ψ functional of example 1 is indeed computable in polynomial time. Indeed, if a strategy computes some F which distinguishes f_∞ and f_x, then the play against the strategy representing f_x necessarily contains the question $?_{2^x}$, which entails

that $|F|(|f_x|) > 2^x$. Then, in any case, the computation time of the machine computing Ψ (whose bahavior was shortly described in example 1) is about $|F|(|0|) \times |F|(|f_x|)$, which is a third order polynomial in $|F|$ and $|x|$, since $|f_x|$ is a polynomial.

5 Application to computable analysis

Now our goal is to use higher-order functionals to represent objects and to extend complexity notions from higher-order functionals to QCB-spaces.

Remind the types inductively defined by $\tau = \mathbb{N} \mid \tau_1 \times \ldots \times \tau_n \to \tau$. Remind that a type is interpreted as a QCB-space of hereditarily total continuous functions. However when representing objects one usually considers *partial* functions, which can be seen as total on their domains. Formally, wee define a particular class of QCB-spaces, the τ-spaces, which will serve as representation spaces.

Definition 5.1. Let τ be a finite type. A ***τ-space*** is defined inductively by:

- an $\mathbb{N}$-space is a subset of $\mathbb{N}$ endowed with the discrete topology,
- given τ_i-spaces A_i and a τ-space A, a $(\tau_1 \times \ldots \times \tau_n \to \tau)$-space is a subspace of the space $\mathscr{C}(A_1 \times \ldots \times A_n, A)$ of total continuous functions, with the induced topology.

Definition 5.2. A ***τ-representation*** of X is a partial surjective continuous function $\delta \subseteq\colon A \to X$ where A is a τ-space.

Observe that usual representations can be viewed as $(\mathbb{N} \to \mathbb{N})$-representations. In terms of topology and computability, τ-representations can be used in place of $(\mathbb{N} \to \mathbb{N})$-representations, as for any τ-space can be embedded in a τ'-space, for any τ, τ' such that τ' has depth at least 1 hence any admissible τ-representation induces an admissible τ'-representation.

The results presented in Section 3 can be formulated as follows: there is no admissible $(\mathbb{N} \to \mathbb{N})$-representation of $\mathscr{C}(X, Y)$ which makes the time complexity of $\mathrm{Eval} : \mathscr{C}(X, Y) \times X \to Y$ well-defined, whenever $(X, Y) = (\mathscr{C}(\mathbb{N}, \mathbb{N}), \mathbb{N})$ or $(X, Y) = (\mathscr{C}([0, 1], \mathbb{R}), \mathbb{R})$.

One has to use τ-representations with arbitrary τ to carry out complexity theory on larger functions spaces.

Definition 5.3. Let δ_X be an admissible τ_1-representation and δ_Y an admissible τ_2-representation. We define a $(\tau_1 \to \tau_2)$-representation $\delta_{X \to Y}$ of $\mathscr{C}(X, Y)$ as: if $f : X \to Y$ is continuous and $F : \tau_1 \to \tau_2$ is continuous and $f \circ \delta_X = \delta_Y \circ F$ then $\delta_{\mathscr{C}(X,Y)}(F) = f$.

Observe that as δ_X and δ_Y are admissible, $\delta_{\mathscr{C}(X,Y)}$ is a representation: it is onto and maps continuous functions to continuous functions.

Proposition 5.1. *The representation $\delta_{\mathscr{C}(X,Y)}$ is admissible. Moreover, Eval : $\mathscr{C}(X, Y) \times X \to Y$ is polynomial-time computable.*

Proof. The first point is proved by the same arguments as in [22, 19], adapted to τ-spaces.

Eval on X and Y is computed by Eval on τ_1 and τ_2, which is polynomial-time computable (see Example 5). □

We do not answer the question whether there exists a minimal representation of $\mathscr{C}(X, Y)$, i.e. a representation that polynomially reduces to any representation that makes Eval polynomial-time computable. We leave that problem for future investigations. Another interesting problem is to study the categorical properties of spaces admitting a representation admissible form a complexity-theoretic perspective. The results presented here suggest that the underlying category may be cartesian closed.

References

[1] Samson Abramsky and Achim Jung. Domain theory. In S. Abramsky, D. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science Volume 3*, pages 1–168. Oxford University Press, 1994.

[2] Samuel R. Buss. The polynomial hierarchy and intuitionistic bounded arithmetic. In Alan L. Selman, editor, *Structure in Complexity Theory, Proceedings of the Conference hold at the University of California, Berkeley, California, June 2-5, 1986*, volume 223 of *Lecture Notes in Computer Science*, pages 77–103. Springer, 1986.

[3] S.A. Cook and B.M. Kapron. Characterizations of the basic feasible functionals of finite type. *Foundations of Computer Science, IEEE Annual Symposium on*, 0:154–159, 1989.

[4] Stephen Cook and Alasdair Urquhart. Functional interpretations of feasibly constructive arithmetic. *Annals of Pure and Applied Logic*, 63(2):103 – 200, 1993.

[5] Stephen A. Cook and Alasdair Urquhart. Functional interpretations of feasibly constructive arithmetic (extended abstract). In David S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC), May 14-17, 1989, Seattle, Washigton, USA*, pages 107–112. ACM, 1989.

[6] J. M. E. Hyland and C. H. Luke Ong. On Full Abstraction for PCF: I, II, and III. *Inf. Comput.*, 163(2):285–408, 2000.

[7] Robert Irwin, Bruce Kapron, and Jim Royer. On characterizations of the basic feasible functionals, part ii. Unpublished, 2002.

[8] Robert J. Irwin, James S. Royer, and Bruce M. Kapron. On characterizations of the basic feasible functionals (part i). *J. Funct. Program.*, 11(1):117–153, 2001.

[9] Bruce M. Kapron and Stephen A. Cook. A new characterization of mehlhorn's polynomial time functionals (extended abstract). In *A New Characterization of Mehlhorn's Polynomial Time Functionals (Extended Abstract)*, pages 342–347. IEEE Computer Society, 1991.

[10] Bruce M. Kapron and Stephen A. Cook. A new characterization of type-2 feasibility. *SIAM J. Comput.*, 25(1):117–132, 1996.

[11] Akitoshi Kawamura. On function spaces and polynomial-time computability, October 2011. Presented at Dagstuhl Seminar 11411.

[12] Akitoshi Kawamura and Stephen A. Cook. Complexity theory for operators in analysis. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 495–502, 2010.

[13] Alexander S. Kechris. *Classical Descriptive Set Theory*. Springer, January 1995.

[14] SC Kleene. Countable functionals. *Constructivity in Mathematics*, pages 81–100, 1959.

[15] Ker-I Ko. *Complexity Theory of Real Functions*. Birkhauser Boston Inc., Cambridge, MA, USA, 1991.

[16] Daren Kunkle and Matthias Schröder. Some examples of non-metrizable spaces allowing a simple type-2 complexity theory. *Electr. Notes Theor. Comput. Sci.*, 120:111–123, 2005.

[17] Kurt Mehlhorn. Polynomial and abstract subrecursive classes. *J. Comput. Syst. Sci.*, 12(2):147–178, April 1976.

[18] Dag Normann. Chapter 8 the continuous functionals. In Edward R. Griffor, editor, *Handbook of Computability Theory*, volume 140 of *Studies in Logic and the Foundations of Mathematics*, pages 251 – 275. Elsevier, 1999.

[19] Matthias Schröder. Extended admissibility. *Theor. Comput. Sci.*, 284(2):519–538, 2002.

[20] Matthias Schröder. Spaces allowing type-2 complexity theory revisited. *Math. Log. Q.*, 50(4-5):443–459, 2004.

[21] A. Seth. Turing machine characterizations of feasible functionals of all finite types. *Feasible Mathematics II*, pages 407–428, 1995.

[22] Klaus Weihrauch. *Computable Analysis*. Springer, Berlin, 2000.

[23] Klaus Weihrauch. Computational complexity on computable metric spaces. *Mathematical Logic Quaterly*, 49(1):3–21, 2003.