Higher order interpretations for higher order complexity

Emmanuel Hainry, Romain Péchoux

Higher order interpretations for higher order complexity: extended abstract

Emmanuel Hainry[1,2] and Romain Péchoux[1,3]

[1] INRIA project Carte, LORIA and Université de Lorraine
[2] hainry@loria.fr
[3] pechoux@loria.fr

Abstract

We design an interpretation-based theory of higher order functions that is well-suited for the complexity analysis of a standard higher order functional language à la ML. We manage to express the interpretation of a given program in terms of a least fixpoint and we show that when restricted to functions bounded by higher order polynomials, they characterize exactly classes of tractable functions known as Basic Feasible Functions at any order.

1 Functional language

Syntax. The language considered in this paper consists in a lambda calculus with constructors, primitive operators, a `case` construct for pattern matching and a `letRec` instruction for (recursive) function definitions. It can be seen as an extension of PCF to inductive data types. A term of our language is defined by the following syntax:

$$\mathtt{M},\mathtt{N} ::= \mathtt{x} \mid \mathtt{c} \mid \mathtt{op} \mid \mathtt{M}\,\mathtt{N} \mid \lambda \mathtt{x}.\mathtt{M} \mid \mathtt{letRec}\ \mathtt{f} = \mathtt{M} \mid \mathtt{case}\ \mathtt{M}\ \mathtt{of}\ \mathtt{c}_1(\overrightarrow{\mathtt{x}_1}) \to \mathtt{M}_1 | ... | \mathtt{c}_n(\overrightarrow{\mathtt{x}_n}) \to \mathtt{M}_n$$

In the above syntax, $\mathtt{c}, \mathtt{c}_1, \cdots, \mathtt{c}_n$ are constructor symbols of fixed arity and `op` is an operator of fixed arity. Given a constructor or operator symbol b, we write $ar(b) = n$ whenever b is of arity n. $\mathtt{x}, \mathtt{f}$ are variables in $\mathcal{X}$ and $\overrightarrow{\mathtt{x}_i}$ is a sequence of $ar(\mathtt{c}_i)$ variables. The free variables $FV(\mathtt{M})$ of a term $\mathtt{M}$ are defined as usual.

Finally, a closed term will consist in a term $\mathtt{M}$ such that $FV(\mathtt{M}) = \emptyset$ (i.e. with no free variables).
A substitution $\{\mathtt{N}_1/\mathtt{x}_1, \cdots, \mathtt{N}_n/\mathtt{x}_n\}$ is a partial function mapping variables $\mathtt{x}_1, \cdots, \mathtt{x}_n$ to terms $\mathtt{N}_1, \cdots, \mathtt{N}_n$.

Semantics. The semantics of our language is described by a relation $\to_{\mathcal{S}}$ between two terms defined by $\to_{\mathcal{S}} = \to_\beta \cup \to_{\mathtt{case}} \cup \to_{\mathtt{op}} \cup \to_{\mathtt{letRec}}$

- $\to_\beta$ is the standard β-reduction defined by $\lambda \mathtt{x}.\mathtt{M}\,\mathtt{N} \to_\beta \mathtt{M}\{\mathtt{N}/\mathtt{x}\}$,
- $\to_{\mathtt{case}}$ corresponds to pattern matching and is defined by:

$$\mathtt{case}\ \mathtt{c}_j(\overrightarrow{\mathtt{N}_j})\ \mathtt{of}\ \ldots \mid \mathtt{c}_j(\overrightarrow{\mathtt{x}_j}) \to \mathtt{M}_j \mid \ldots \to_{\mathtt{case}} \mathtt{M}_j\{\overrightarrow{\mathtt{N}_j}/\overrightarrow{\mathtt{x}_j}\},$$

- $\to_{\mathtt{letRec}}$ is a fixpoint evaluation defined by $\mathtt{letRec}\ \mathtt{f} = \mathtt{M} \to_{\mathtt{letRec}} \mathtt{M}\{\mathtt{letRec}\ \mathtt{f} = \mathtt{M}/\mathtt{f}\}$,
- $\to_{\mathtt{op}}$ is an operator evaluation defined by $\mathtt{op}\,\mathtt{M}_1 \ldots \mathtt{M}_n \to_{\mathtt{op}} \mathtt{M}$, provided that `op` is a primitive operator of arity n whose semantics $[\![\mathtt{op}]\!]$, fixed by the language implementation, is a total function that satisfies $[\![\mathtt{op}]\!](\mathtt{M}_1, \ldots, \mathtt{M}_n) = \mathtt{M}$.

In the particular case, where the pattern matching on the `case` first argument fails (i.e. $\neg \exists j, \mathtt{M} = \mathtt{c}_j(\overrightarrow{\mathtt{N}_j})$), we extend the relation $\to_\alpha$, $\alpha \in \{\beta, \mathtt{case}, \mathtt{letRec}, \mathtt{op}\}$, by if $\mathtt{M} \to_\alpha \mathtt{N}$ then $\mathtt{case}\ \mathtt{M}\ \mathtt{of}\ \ldots \to_\alpha \mathtt{case}\ \mathtt{N}\ \mathtt{of}\ \ldots$. In what follows, we will fix a left-most outermost evaluation strategy with respect to, $\to_{\{\beta,\mathtt{case},\mathtt{letRec},\mathtt{op}\}}$, noted $\Rightarrow$. Let $\Rightarrow^k$ be the k-fold self composition of the relation $\Rightarrow$ wrt such a strategy.

$$\frac{\Gamma(\mathtt{x}) = \mathtt{T}}{\Gamma;\Delta \vdash \mathtt{x} :: \mathtt{T}}\ (\mathtt{Var}) \qquad \frac{\Delta(\mathtt{c}) = \mathtt{T}}{\Gamma;\Delta \vdash \mathtt{c} :: \mathtt{T}}\ (\mathtt{Cons}) \qquad \frac{\Delta(\mathtt{op}) = \mathtt{T}}{\Gamma;\Delta \vdash \mathtt{op} :: \mathtt{T}}\ (\mathtt{Op})$$

$$\frac{\Gamma;\Delta \vdash \mathtt{M} :: \mathtt{T}_1 \longrightarrow \mathtt{T}_2 \quad \Gamma;\Delta \vdash \mathtt{N} :: \mathtt{T}_1}{\Gamma;\Delta \vdash \mathtt{M}\,\mathtt{N} :: \mathtt{T}_2}\ (\mathtt{App})$$

$$\frac{\Gamma, \mathtt{x} :: \mathtt{T}_1;\Delta \vdash \mathtt{M} :: \mathtt{T}_2}{\Gamma;\Delta \vdash \lambda \mathtt{x}.\mathtt{M} :: \mathtt{T}_1 \to \mathtt{T}_2}\ (\mathtt{Abs}) \qquad \frac{\Gamma, \mathtt{f} :: \mathtt{T};\Delta \vdash \mathtt{M} :: \mathtt{T}}{\Gamma;\Delta \vdash \mathtt{letRec}\ \mathtt{f} = \mathtt{M} :: \mathtt{T}}\ (\mathtt{Let})$$

$$\frac{\Gamma;\Delta \vdash \mathtt{M} :: \mathtt{b} \qquad \Gamma;\Delta \vdash \mathtt{c}_i :: \overrightarrow{\mathtt{b}_i} \longrightarrow \mathtt{b} \quad \Gamma, \overrightarrow{\mathtt{x}_i} :: \overrightarrow{\mathtt{b}_i};\Delta \vdash \mathtt{M}_i :: \mathtt{T}\ (1 \leq i \leq n)}{\Gamma;\Delta \vdash \mathtt{case}\ \mathtt{M}\ \mathtt{of}\ \mathtt{c}_1(\overrightarrow{\mathtt{x}_1}) \to \mathtt{M}_1 | ... | \mathtt{c}_n(\overrightarrow{\mathtt{x}_n}) \to \mathtt{M}_n :: \mathtt{T}}\ (\mathtt{Case})$$

Figure 1: Type system

Moreover, let $|\mathtt{M} \Rightarrow^k \mathtt{N}|$ be the number of reductions distinct from $\to_{\mathtt{op}}$ in a given a derivation $\mathtt{M} \Rightarrow^k \mathtt{N}$. $|\mathtt{M} \Rightarrow^k \mathtt{N}| \leq k$ always holds. $[\![\mathtt{M}]\!]$ is a notation for the term computed by $\mathtt{M}$ (if it exists), i.e. $\exists k, \mathtt{M} \Rightarrow^k [\![\mathtt{M}]\!]$ and $\nexists \mathtt{N}, [\![\mathtt{M}]\!] \Rightarrow \mathtt{N}$. A (first order) value $\mathtt{v}$ is defined inductively by either $\mathtt{v} = \mathtt{c}$, if $ar(\mathtt{c}) = 0$, or $\mathtt{v} = \mathtt{c}\ \vec{\mathtt{v}}$, for $ar(\mathtt{c}) > 0$ values $\vec{\mathtt{v}}$, otherwise.

Type system. We fix a set **B** of basic inductive types $\mathtt{b}$ described by their constructor set $C_{\mathtt{b}}$. For example, the type of natural numbers $\mathtt{Nat}$ is described by $C_{\mathtt{Nat}} = \{0, +1\}$. The set of simple types is defined by $\mathtt{T} ::= \mathtt{b} \mid \mathtt{T} \longrightarrow \mathtt{T}$, with $\mathtt{b} \in \mathbf{B}$. As usual $\longrightarrow$ associates to the right.

In what follows, we will consider only well-typed terms. The type system assigns types to all the syntactic constructions of the language and ensures that a program does not go wrong. Notice that the typing discipline does not prevent a program from diverging. The type system is described in Figure 1 and proves judgments of the shape $\Gamma;\Delta \vdash \mathtt{M} :: \mathtt{T}$ meaning that the term $\mathtt{M}$ has type $\mathtt{T}$ under the variable and constructor (and operator) symbol contexts Γ and Δ respectively ; a variable (a constructor or operator symbol) context being a partial function that assigns types to variables (respectively constructors or operators). As usual, the input type and output type of constructors and operators of arity n will be restricted to basic types. Consequently, their types are of the shape $\mathtt{b}_1 \longrightarrow \ldots \longrightarrow \mathtt{b}_n \longrightarrow \mathtt{b}$. A well-typed term will consist in a term $\mathtt{M}$ such that $\emptyset;\Delta \vdash \mathtt{M} :: \mathtt{T}$ (Consequently, it is mandatory for a term to be closed in order to be well-typed).

Given a term $\mathtt{M}$ of type $\mathtt{T}$, i.e. $\emptyset;\Delta \vdash \mathtt{M} :: \mathtt{T}$, the order of $\mathtt{M}$, noted $\mathtt{ord}(\mathtt{M})$, is equal to the order of $\mathtt{T}$, noted $\mathtt{ord}(\mathtt{T})$ and defined inductively by:

$\mathtt{ord}(\mathtt{b}) = 0$, if $\mathtt{b} \in \mathbf{B}$, and $\mathtt{ord}(\mathtt{T} \longrightarrow \mathtt{T}') = \max(\mathtt{ord}(\mathtt{T}) + 1, \mathtt{ord}(\mathtt{T}'))$ otherwise..

Example 1. *Consider the following term* $\mathtt{M}$ *that maps a function to a list given as inputs:*

$$\begin{aligned}\mathtt{letRec}\ \mathtt{f} = \lambda g.\lambda \mathtt{x}.\mathtt{case}\ \mathtt{x}\ \mathtt{of}\ \mathtt{c}(\mathtt{y},\mathtt{z}) &\to \mathtt{c}(g\ \mathtt{y})(\mathtt{f}\ g\ \mathtt{z})\\ \mid \mathtt{nil} &\to \mathtt{nil}\end{aligned}$$

Suppose that $[\mathtt{Nat}]$ *is the base type for lists of natural numbers of constructor and set* $C_{[\mathtt{Nat}]} = \{\mathtt{nil}, \mathtt{c}\}$. *The term* $\mathtt{M}$ *can be typed by* $(\mathtt{Nat} \longrightarrow \mathtt{Nat}) \longrightarrow [\mathtt{Nat}] \longrightarrow [\mathtt{Nat}]$. *Consequently,* $\mathtt{ord}(\mathtt{M}) = 2$.

2 Interpretations

In this section, we define the interpretation tools that will allow us to control the complexity of closed terms of our language.

Types. We briefly recall some basic definitions that are very close from the notions used in denotational semantics (See [Win93]) since, as we shall see later, our notion of interpretation also allows us to obtain fixpoints. Let $(\mathbb{N}, \leq, \sqcup, \sqcap)$ be the set of natural numbers equipped with the usual ordering $\leq$, a max operator $\sqcup$ and min operator $\sqcap$ and let $\overline{\mathbb{N}}$ be $\mathbb{N} \cup \{\top\}$, where $\forall n \in \mathbb{N},\ n \leq \top$, $n \sqcup \top = \top \sqcup n = \top$ and $n \sqcap \top = \top \sqcap n = n$. The strict order relation over natural numbers $<$ will also be used in the sequel and is extended in a somewhat unusual manner, by $\top < \top$. The *interpretation* of a type is defined by:

$$\begin{aligned} (\!|\mathtt{b}|\!) &= \overline{\mathbb{N}}, && \text{if } \mathtt{b} \text{ is a basic type,} \\ (\!|\mathtt{T} \longrightarrow \mathtt{T}'|\!) &= (\!|\mathtt{T}|\!) \longrightarrow^{\uparrow} (\!|\mathtt{T}'|\!), && \text{otherwise,} \end{aligned}$$

where $(\!|\mathtt{T}|\!) \longrightarrow^{\uparrow} (\!|\mathtt{T}'|\!)$ denotes the set of total strictly monotonic functions from $(\!|\mathtt{T}|\!)$ to $(\!|\mathtt{T}'|\!)$. A function F from the set A to the set B being strictly monotonic if for each $X, Y \in A$, $X <_A Y$ implies $F(X) <_B F(Y)$, where $<_A$ is the usual pointwise ordering induced by $<$ and defined by:

$$n <_{\overline{\mathbb{N}}} m \text{ iff } n < m \text{ and } F <_{A \longrightarrow^{\uparrow} B} G \qquad \text{iff } \forall X \in A,\ F(X) <_B G(X)$$

Example 2. *The type* $\mathtt{T} = (\mathtt{Nat} \longrightarrow \mathtt{Nat}) \longrightarrow [\mathtt{Nat}] \longrightarrow [\mathtt{Nat}]$ *of the term* `letRec f = M` *in Example 1 is interpreted by:*

$$(\!|\mathtt{T}|\!) = (\overline{\mathbb{N}} \longrightarrow^{\uparrow} \overline{\mathbb{N}}) \longrightarrow^{\uparrow} (\overline{\mathbb{N}} \longrightarrow^{\uparrow} \overline{\mathbb{N}}).$$

In what follows, given a sequence $\overrightarrow{R}$ of m terms in the interpretation domain and a sequence $\overrightarrow{\mathtt{T}}$ of k types, the notation $\overrightarrow{R} \in \overrightarrow{(\!|\mathtt{T}|\!)}$ means that both $k = m$ and $\forall i \in [1, m],\ R_i \in (\!|\mathtt{T}_i|\!)$.

Terms. Each closed term of type $\mathtt{T}$ will be interpreted by a functional in $(\!|\mathtt{T}|\!)$. The application is denoted as usual whereas we use the notation Λ for abstraction on this function space in order to avoid confusion between terms of our calculus and objects of the interpretation domain. Variables of the interpretation domain will be denoted using upper case letters. Moreover, we will sometimes use Church typing discipline in order to highlight the type of the bound variable in a lambda abstraction.

An important distinction between the terms of our language and the objects of the interpretation domain lies in the fact that beta-reduction is considered as an equivalence relation on (closed terms of) the interpretation domain, i.e. $\Lambda X.F\ G = F\{G/X\}$ underlying that $\Lambda X.F\ G$ and $F\{G/X\}$ are distinct notations that represent the same higher order function. The same property holds for η-reduction, i.e. $\Lambda X.F\ X$ and F denote the same function.

Since we are interested in complete lattices, we need to complete each type $(\!|\mathtt{T}|\!)$ by a lower bound $\bot_{(\!|\mathtt{T}|\!)}$ and an upper bound $\top_{(\!|\mathtt{T}|\!)}$ as follows:

$$\begin{aligned} \bot_{\overline{\mathbb{N}}} &= 0 & \top_{\overline{\mathbb{N}}} &= \top \\ \bot_{(\!|\mathtt{T} \longrightarrow \mathtt{T}'|\!)} &= \Lambda X^{(\!|\mathtt{T}|\!)}.\bot_{(\!|\mathtt{T}'|\!)} & \top_{(\!|\mathtt{T} \longrightarrow \mathtt{T}'|\!)} &= \Lambda X^{(\!|\mathtt{T}|\!)}.\top_{(\!|\mathtt{T}'|\!)} \end{aligned}$$

We can show by an easy structural induction on types that for each $F \in (\!|\mathtt{T}|\!)$, $\bot_{(\!|\mathtt{T}|\!)} \leq_{(\!|\mathtt{T}|\!)} F \leq_{(\!|\mathtt{T}|\!)} \top_{(\!|\mathtt{T}|\!)}$. Notice that for each type $\mathtt{T}$ it also holds that $\top_{(\!|\mathtt{T}|\!)} <_{(\!|\mathtt{T}|\!)} \top_{(\!|\mathtt{T}|\!)}$, by an easy induction.

In the same spirit, we extend inductively the max and min operators $\sqcup$ (and $\sqcap$) over $\overline{\mathbb{N}}$ to arbitrary higher order functions F, G of type $(\!|\mathtt{T}|\!) \longrightarrow^{\uparrow} (\!|\mathtt{T}'|\!)$ by:

$$\sqcup^{(\!|\mathtt{T}|\!)\longrightarrow^{\uparrow}(\!|\mathtt{T}'|\!)}(F,G) = \Lambda X^{(\!|\mathtt{T}|\!)}.\, \sqcup^{(\!|\mathtt{T}'|\!)}(F(X), G(X))$$
$$\sqcap^{(\!|\mathtt{T}|\!)\longrightarrow^{\uparrow}(\!|\mathtt{T}'|\!)}(F,G) = \Lambda X^{(\!|\mathtt{T}|\!)}.\, \sqcap^{(\!|\mathtt{T}'|\!)}(F(X), G(X))$$

In the following, we use the notations $\bot$, $\top$, $\leq$, $<$, $\sqcup$ and $\sqcap$ instead of $\bot_{(\!|\mathtt{T}|\!)}$, $\top_{(\!|\mathtt{T}|\!)}$, $\leq_{(\!|\mathtt{T}|\!)}$, $<_{(\!|\mathtt{T}|\!)}$, $\sqcup^{(\!|\mathtt{T}|\!)}$ and $\sqcap^{(\!|\mathtt{T}|\!)}$, respectively, when $(\!|\mathtt{T}|\!)$ is clear from the typing context.

Lemma 1. *For each type* $\mathtt{T}$, $((\!|\mathtt{T}|\!), \leq, \sqcup, \sqcap, \top, \bot)$ *is a complete lattice.*

Now we need to define a unit (or constant) cost function for any interpretation of type $\mathtt{T}$ in order to take the cost of recursive calls into account. For that purpose, let $+$ denote natural number addition extended to $\overline{\mathbb{N}}$ by $\forall n,\ \top + n = n + \top = \top$. For each type $(\!|\mathtt{T}|\!)$, we define a dyadic sum function $\oplus_{(\!|\mathtt{T}|\!)}$ by:

$$X^{\overline{\mathbb{N}}} \oplus_{\overline{\mathbb{N}}} Y^{\overline{\mathbb{N}}} = X + Y \qquad\qquad F \oplus_{(\!|\mathtt{T}\longrightarrow\mathtt{T}'|\!)} G = \Lambda X^{(\!|\mathtt{T}|\!)}.(F(X) \oplus_{(\!|\mathtt{T}'|\!)} G(X))$$

Let us also define the constant function $n_{(\!|\mathtt{T}|\!)}$, for each type $\mathtt{T}$ and each integer $n \geq 1$, by $n_{\overline{\mathbb{N}}} = n$ and $n_{(\!|\mathtt{T}\longrightarrow\mathtt{T}'|\!)} = \Lambda X^{(\!|\mathtt{T}|\!)}.n_{(\!|\mathtt{T}'|\!)}$ Once again, we will omit the type when it is unambiguous using the notation $n\oplus$ to denote the function $n_{(\!|\mathtt{T}|\!)}\oplus_{(\!|\mathtt{T}|\!)}$ when $(\!|\mathtt{T}|\!)$ is clear from the typing context.

Now we are ready to define the notions of variable assignment and interpretation of a term $\mathtt{M}$:

Definition 1 (Interpretation)**.**

- *A* variable assignment*, denoted ρ, is a map associating to each $\mathtt{f} \in \mathcal{X}$ of type $\mathtt{T}$ a variable F of type $(\!|\mathtt{T}|\!)$.*
- *Given a variable assignment ρ, an* interpretation *is the extension of ρ to well-typed terms, mapping each term of type $\mathtt{T}$ to an object in $(\!|\mathtt{T}|\!)$ and defined in Figure 2, where $(\!|\mathtt{op}|\!)_\rho$ is a sup-interpretation, i.e. a total function such that:*

$$\forall \mathtt{M}_1, \ldots, \forall \mathtt{M}_n,\ (\!|\mathtt{op}\ \mathtt{M}_1\ \ldots\ \mathtt{M}_n|\!)_\rho \geq (\!|[\![\mathtt{op}]\!](\mathtt{M}_1, \ldots, \mathtt{M}_n)|\!)_\rho.$$

Notice that, contrarily to other program constructs, operators may have non monotonic interpretations. This is the reason why we have fixed a left-most outermost strategy (we never reduce the operator operands) and operators are supposed to be total functions.

Example 3. *Consider the following term* $\mathtt{M} :: \mathtt{Nat} \longrightarrow \mathtt{Nat}$ *computing the double of a unary number given as input* $\mathtt{letRec\ f} = \lambda\mathtt{x.case\ x\ of\ +1(y)} \to \mathtt{+1(+1(f\ y))} \mid 0 \to 0$*. We can see in Figure 3 how the interpretation rules of Figure 2 are applied on such a term. At the end we search for the minimal strictly monotonic function F greater than $\Lambda X.(5 \oplus (F\ (X-1))) \sqcup 4$, for $X > 1$. That is $\Lambda X.5X \oplus 4$.*

The interpretation of a term is always defined. Indeed, in Definition 1, $(\!|\mathtt{letRec\ f} = \mathtt{M}|\!)_\rho$ is defined in terms of the least fixpoint of the function $\Lambda X^{(\!|\mathtt{T}|\!)}.1 \oplus_{(\!|\mathtt{T}|\!)} (\Lambda (\!|\mathtt{f}|\!)_\rho.(\!|\mathtt{M}|\!)_\rho\ X)$ and, consequently, we obtain the following result as a direct consequence of Knaster-Tarski [Tar55, KS01] Fixpoint Theorem:

Proposition 1. *Each term* $\mathtt{M}$ *of type* $\mathtt{T}$ *has an interpretation.*

We can also show that the length of a derivation is bounded by the interpretation of the initial term:

Lemma 2. *For all terms,* $\mathtt{M}, \vec{\mathtt{N}}$*, such that* $\emptyset; \Delta \vdash \mathtt{M}\ \vec{\mathtt{N}} :: \mathtt{T}$*, if* $\mathtt{M}\ \vec{\mathtt{N}} \Rightarrow^k \mathtt{M}'$ *then* $(\!|\mathtt{M}|\!)_\rho (\!|\vec{\mathtt{N}}|\!)_\rho \geq |\mathtt{M}\ \vec{\mathtt{N}} \Rightarrow^k \mathtt{M}'| \oplus (\!|\mathtt{M}'|\!)_\rho$.

- $(\!|\mathtt{f}|\!)_\rho = \rho(\mathtt{f})$, if $\mathtt{f} \in \mathcal{X}$,
- $(\!|\mathtt{c}|\!)_\rho = 1 \oplus (\Lambda X_1. \ldots . \Lambda X_n. \sum_{i=1}^{n} X_i)$, if $ar(\mathtt{c}) = n$,
- $(\!|\mathtt{MN}|\!)_\rho = (\!|\mathtt{M}|\!)_\rho (\!|\mathtt{N}|\!)_\rho$,
- $(\!|\lambda \mathtt{x}.\mathtt{M}|\!)_\rho = 1 \oplus (\Lambda (\!|\mathtt{x}|\!)_\rho . (\!|\mathtt{M}|\!)_\rho)$,
- $(\!|\mathtt{case\ M\ of\ c}_1(\overrightarrow{\mathtt{x}_1}) \to \mathtt{M}_1 | ... | \mathtt{c}_n(\overrightarrow{\mathtt{x}_n}) \to \mathtt{M}_n|\!)_\rho = 1 \oplus \sqcup_{1 \leq i \leq m} \{ (\!|\mathtt{M}_i|\!)_\rho \{ \overrightarrow{R_i} / \overrightarrow{(\!|\mathtt{x}_i|\!)_\rho} \} \mid \forall \overrightarrow{R_i} \text{ s.t. } (\!|\mathtt{M}|\!)_\rho \geq (\!|\mathtt{c}_i|\!)_\rho \overrightarrow{R_i} \}$,
- $(\!|\mathtt{letRec\ f = M}|\!)_\rho = \sqcap \{ F \in (\!|\mathtt{T}|\!) \mid F \geq 1 \oplus \Lambda (\!|\mathtt{f}|\!)_\rho . (\!|\mathtt{M}|\!)_\rho F \}$.

Figure 2: Interpretation of a term of type T

$$
\begin{aligned}
&(\!|\mathtt{M}|\!)_\rho \\
&= \sqcap \{ F \in (\!|\mathtt{T}|\!) \mid F \geq 1 \oplus \Lambda (\!|\mathtt{f}|\!)_\rho . (\!|\lambda \mathtt{x.case\ x\ of\ } +1(\mathtt{y}) \to +1(+1(\mathtt{f\ y}))|\ 0 \to 0|\!)_\rho F \} \\
&= \sqcap \{ F \in (\!|\mathtt{T}|\!) \mid F \geq 2 \oplus (\Lambda (\!|\mathtt{f}|\!)_\rho . \Lambda (\!|\mathtt{x}|\!)_\rho . (\!|\mathtt{case\ x\ of\ } +1(\mathtt{y}) \to +1(+1(\mathtt{f\ y}))|\ 0 \to 0|\!)_\rho F) \} \\
&= \sqcap \{ F \in (\!|\mathtt{T}|\!) \mid F \geq 3 \oplus (\Lambda (\!|\mathtt{f}|\!)_\rho . \Lambda (\!|\mathtt{x}|\!)_\rho . (\sqcup_{(\!|\mathtt{x}|\!)_\rho \geq (\!|+1(\mathtt{y})|\!)_\rho} (\!|+1(+1(\mathtt{f\ y}))|\!)_\rho) \sqcup (\sqcup_{(\!|\mathtt{x}|\!)_\rho \geq (\!|0|\!)_\rho} (\!|0|\!)_\rho) F) \} \\
&= \sqcap \{ F \in (\!|\mathtt{T}|\!) \mid F \geq 3 \oplus (\Lambda (\!|\mathtt{f}|\!)_\rho . \Lambda (\!|\mathtt{x}|\!)_\rho . (\sqcup_{(\!|\mathtt{x}|\!)_\rho \geq 1 \oplus (\!|\mathtt{y}|\!)_\rho} 2 \oplus ((\!|\mathtt{f}|\!)_\rho\ (\!|\mathtt{y}|\!)_\rho)) \sqcup (\sqcup_{(\!|\mathtt{x}|\!)_\rho \geq 1} 1) F) \} \\
&= \sqcap \{ F \in (\!|\mathtt{T}|\!) \mid F \geq 3 \oplus (\Lambda (\!|\mathtt{x}|\!)_\rho . (\sqcup_{(\!|\mathtt{x}|\!)_\rho \geq 1 \oplus (\!|\mathtt{y}|\!)_\rho} 2 \oplus (F\ (\!|\mathtt{y}|\!)_\rho)) \sqcup (1)) \} \\
&= \sqcap \{ F \in (\!|\mathtt{T}|\!) \mid F \geq 3 \oplus (\Lambda (\!|\mathtt{x}|\!)_\rho . (2 \oplus (F\ ((\!|\mathtt{x}|\!)_\rho - 1))) \sqcup (1)) \}, \quad (\!|\mathtt{x}|\!)_\rho - 1 \geq 0 \\
&= \sqcap \{ F \in (\!|\mathtt{T}|\!) \mid F \geq \Lambda X . (5 \oplus (F\ (X - 1))) \sqcup (4) \} \\
&= \Lambda X . 5X + 4
\end{aligned}
$$

Figure 3: Example of interpretation

2.1 Higher Order Polynomial Interpretations

At the present time, the interpretation of a term of type T can be any total functional over $(\!|\mathtt{T}|\!)$. In the next section, we will concentrate our efforts to study polynomial time at higher order. Consequently, we need to restrict the shape of the admissible interpretations. For that purpose, we introduce higher order polynomials which are the higher order counterpart to polynomials in this theory of complexity.

Definition 2. *Let P_i denote a polynomial of order i and let X_i denote an order i variable. Higher order (order 1 and order i+1) polynomials can be defined by the following grammar ($c \in \mathbb{N}$):*

$$
\begin{aligned}
P_1 &::= c | X_0 | P_1 + P_1 | P_1 \times P_1 \\
P_{i+1} &::= P_i | P_{i+1} + P_{i+1} | P_{i+1} \times P_{i+1} | X_i(P_{i+1})
\end{aligned}
$$

Clearly, the set of order i polynomials is strictly included in the set of order $i+1$ polynomials by the above definition. Moreover, by definition, a higher order polynomial P_{i+1} has variables of order at most

i. If $\overrightarrow{X}$ is the sequence of such variables ordered by decreasing order, we will treat the polynomial P_{i+1} as total functions $\Lambda\overrightarrow{X}.P_{i+1}(\overrightarrow{X})$.

We are now ready to define the class of functions computed by terms admitting an interpretation that is (higher order) polynomially bounded:

Definition 3. *Let* FP_i, $i > 0$, *be the class of polynomial functionals at order* i *that consist in functionals computed by terms* $\mathtt{M}$ *over the basic type* $\mathcal{C}_{\mathtt{Nat}}$ *and such that* $\mathtt{ord}(\mathtt{M}) = i$ *and* $(\!|\mathtt{M}|\!)_\rho$ *is bounded by an order* i *polynomial (i.e.* $\exists P_i,\ (\!|\mathtt{M}|\!)_\rho \leq P_i$*).*

3 A characterization of Safe Feasible Functionals of any order

BTLP is a higher order language defined in [IKR02] based on loops of the shape **Loop** $v_0^{\mathtt{D}}$ **with** $v_1^{\mathtt{D}}$ **do** $\{I^*\}$. The operational semantics of BTLP procedures is standard: parameters are passed by call-by-value and each loop statement is evaluated by iterating $|v_0|$-many times the loop body instruction under the following restriction: if an assignment $v := E$ is to be executed within the loop body, we check if the value obtained by evaluating E is of size smaller than the size of the loop bound $|v_1|$. If not then the result of evaluating this assignment is to assign 0 to v.

Definition 4 (BFF_i)**.** *For any* $i \geq 1$, BFF_i *is the class of order* i *functionals computable by a BTLP procedure (see [IKR02])*[1].

It is straightforward that $\mathrm{BFF}_1 = \mathrm{FPTIME}$ and BFF_2=BFF. Now we restrict the domain of BFF_i classes to inputs in BFF_k for $k < i$, the obtained classes are named SFF for Safe Feasible Functionals.

Definition 5 (SFF_i)**.** SFF_1 *is defined to be the class of order* 1 *functionals computable by BTLP a procedure and, for any* $i \geq 1$, SFF_{i+1} *is the class of order* $i+1$ *functionals computable by BTLP a procedure on the input domain* SFF_i*. In other words,*

$$\mathrm{SFF}_1 = \mathrm{BFF}_1,$$
$$\forall i \geq 1,\ \mathrm{SFF}_{i+1} = \mathrm{BFF}_{i+1\restriction \mathrm{SFF}_i}$$

Theorem 1. *For any order* $i \geq 1$*, the class of functions in* FP_i *over* FP_k, $k < i$, *is exactly the class of functionals in* SFF_i*. In other words,* $\mathrm{SFF}_i \equiv \mathrm{FP}_{i\restriction(\cup_{k\leq i}\mathrm{FP}_k)}$, *for all* $i \geq 1$.

Proof. Soundness relies on Lemma 2. Completeness is shown by simulating a BTLP procedure by a functional program admitting an interpretation. □

References

[IKR02] Robert J. Irwin, Bruce M. Kapron, and James S. Royer. On characterizations of the basic feasible functionals (part II). Technical report, Syracuse University, 2002.

[KS01] William A. Kirk and Brailey Sims. *Handbook of metric fixed point theory*. Springer, 2001.

[Tar55] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific journal of Mathematics*, 5(2):285–309, 1955.

[Win93] Glynn Winskel. *The formal semantics of programming languages: an introduction*. MIT press, 1993.

[1] As demonstrated in [IKR02], all types in the procedure can be restricted to be of order at most i without any distinction.