Tiered complexity at higher order

Emmanuel Hainry, Bruce Kapron, Jean-Yves Marion, Romain Péchoux

Tiered complexity at higher order (Extended abstract)

Emmanuel Hainry[1], Bruce Kapron[2], Jean-Yves Marion[1] and Romain Péchoux[1]

[1] Université de Lorraine, LORIA
[2] Victoria University
{emmanuel.hainry,jean-yves.marion,romain.pechoux}@loria.fr,
bmkapron@uvic.ca

Abstract. A characterization of the class of Basic Feasible Functionals (BFF) is provided in terms of typable and terminating imperative programs with oracles. The type system is a tier-based type system and type inference can be done in polynomial time.

1 Motivations

In [2], Kapron and Steinberg introduce restrictions on Oracle Turing Machines to characterize the class of basic feasible functionals (BFF).

The Oracle Turing Machines they consider M_ϕ are Turing Machines with one query tape for oracle calls. If a query is written on such a tape and the machine enters a query-state, then the machine outputs the oracle's answer on the query tape in one step.

Definition 1. *Given an OTM M_ϕ and an input $\boldsymbol{a}$, let $m_{\boldsymbol{a}}^{M_\phi}$ be the maximum of the size of the input $\boldsymbol{a}$ and of the biggest oracle's answer in the run of machine on input $\boldsymbol{a}$ with oracle ϕ. A machine M_ϕ has:*

- *a polynomial step count if there is a polynomial P such that for any input $\boldsymbol{a}$ and oracle ϕ, M runs in time bounded by $P(m_{\boldsymbol{a}}^{M_\phi})$.*
- *a finite length revision if there exists a natural number n such that for any oracle and any input, in the run of the machine, the number of times it happens that an oracle answer is bigger than the input and all of the previous oracle answers is at most n.*
- *a finite lookahead revision if there exists a natural number n such that for any oracle and any input, in the run of the machine, it happens at most n times that a query is posed whose size exceeds the size of all previous queries.*

Definition 2 (Strong and Moderate Poly-Time).

- SPT *is the class of second order functions computable by an OTM with a polynomial step count and finite length revision.*
- MPT *is the class of second order functions computable by an OTM with a polynomial step count and finite lookahead revision.*

For a given class of functionals X, let $\lambda(X)$ be the set of simply typed lambda-terms where a constant symbol is available for each element of X. Let $\lambda(X)_2$ be the set of type two functionals represented by type two terms of $\lambda(X)$.

They obtain the following characterization of BFF:

Theorem 1. $\lambda(\mathrm{MPT})_2 = \lambda(\mathrm{SPT})_2 = BFF$

The main interest of this characterization is that it does not require the use of polynomial at order 2 and the semantics restrictions on the OTM are very natural.

We are now interested in finding a static analysis criterion allowing to ensure that a program computes a function of BFF. Our target is an implicit computational complexity characterization of BFF based on Kapron and Steinberg's restrictions.

2 Imperative programs with Oracles

For that purpose, we consider a simple imperative programming language over binary words W with basic operators and oracles:

$$
\begin{array}{lll}
\texttt{Expressions} & \mathtt{e}, \mathtt{e}_1, \ldots, \mathtt{e}_n & ::= \mathtt{x} \mid \mathtt{op}(\mathtt{e}_1, \ldots, \mathtt{e}_{ar(\mathtt{op})}) \mid \phi(\mathtt{e}_1 \upharpoonright \mathtt{e}_2) \\
\texttt{Commands} & \mathtt{c}, \mathtt{c}_1, \mathtt{c}_2 & ::= \mathtt{skip} \mid \mathtt{x} := \mathtt{e} \mid \mathtt{c}_1 \ ; \ \mathtt{c}_2 \\
 & & \quad \mid \mathtt{if(e)\{c_1\}\ else\ \{c_2\}} \mid \mathtt{while(e)\{c\}} \\
\texttt{Programs} & \mathtt{p}_\phi & ::= \mathtt{c\ return\ x}
\end{array}
$$

Program semantics is standard. Oracle calls $\phi(\mathtt{e}_1 \upharpoonright \mathtt{e}_2)$ first evaluate their arguments $\mathtt{e}_1$ and $\mathtt{e}_2$ to values (binary words) v and w, respectively. Then the truncation $v \upharpoonright w$ is computed. It consists in the first $|w|$ symbols of v, if $|w| \leq |v|$, where $|-|$ is the usual size function over binary words. For a fixed operator $\mathtt{op}$ of arity $ar(\mathtt{op})$, let $[\![\mathtt{op}]\!] : W^{ar(\mathtt{op})} \to W$ be the function computed by the operator.

3 A tier-based type system

We introduce a type system with $\mathbf{k}$ tiers (a tier can be viewed as a natural number) inspired by the type system of [3] that prevents data flows from lower tiers to higher tiers. Types are tuples of 3 tiers.

Atomic types are elements of the set $(\mathbf{N}, \preceq, \mathbf{0}, \vee, \wedge)$ where $\mathbf{N} = \{\mathbf{0}, \mathbf{1}, \mathbf{2}, \ldots\}$ is the set of integers, called *tiers*, $\preceq$ is the usual ordering on integers and $\vee$ and $\wedge$ are the max and min operators over integers. Let $\prec$ be defined by $\prec := \preceq \cap \neq$. We use the symbols $\mathbf{k}, \mathbf{k}', \ldots, \mathbf{k}_1, \mathbf{k}_2, \ldots, \mathbf{k}_{in}, \mathbf{k}_{out}$ to denote tier variables.

A variable typing environment Γ is a finite mapping from $\mathbb{V}$ to $\mathbf{N}$, which assigns a single tier to each variable. An operator typing environment Δ is a mapping that associates to each operator $\mathtt{op}$ and each tier $\mathbf{k}$ a set of operator types $\Delta(\mathtt{op})(\mathbf{k})$, where the operator types corresponding to the operator $\mathtt{op}$ are of the shape $\mathbf{k}_1 \to \ldots \mathbf{k}_{ar(\mathtt{op})} \to \mathbf{k}'$, with $\mathbf{k}_i, \mathbf{k}' \in \mathbf{N}$. Let $dom(\Gamma)$ (resp. $dom(\Delta)$) denote the set of variables typed by Γ (resp. the set of operators typed by Δ).

Typing judgments are either *command (expression) typing judgments* of the shape $\Gamma, \Delta \vdash \mathtt{c} : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out})$ or $\Gamma, \Delta \vdash \mathtt{e} : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out})$, respectively. Typing rules are provided in Figure 1.

$$\frac{\mathbf{k}_1 \to \cdots \to \mathbf{k}_{ar(\mathtt{op})} \to \mathbf{k} \in \Delta(\mathtt{op})(\mathbf{k}_{in}) \qquad \forall i \leq ar(\mathtt{op}),\ \Gamma, \Delta \vdash \mathtt{e}_i : (\mathbf{k}_i, \mathbf{k}_{in}, \mathbf{k}_{out})}{\Gamma, \Delta \vdash \mathtt{op}(\mathtt{e}_1, \ldots, \mathtt{e}_{ar(\mathtt{op})}) : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out})} \text{ (OP)}$$

$$\frac{\Gamma, \Delta \vdash \mathtt{e}_1 : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out}) \qquad \Gamma, \Delta \vdash \mathtt{e}_2 : (\mathbf{k}_{out}, \mathbf{k}_{in}, \mathbf{k}_{out}) \qquad \mathbf{k} \prec \mathbf{k}_{in} \wedge \mathbf{k} \preceq \mathbf{k}_{out}}{\Gamma, \Delta \vdash \phi(\mathtt{e}_1 \restriction \mathtt{e}_2) : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out})} \text{ (OR)}$$

$$\frac{\Gamma(\mathtt{x}) = \mathbf{k}}{\Gamma, \Delta \vdash \mathtt{x} : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out})} \text{ (V)} \quad \frac{\Gamma, \Delta \vdash \mathtt{c}_1 : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out}) \qquad \Gamma, \Delta \vdash \mathtt{c}_2 : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out})}{\Gamma, \Delta \vdash \mathtt{c}_1 \ ; \ \mathtt{c}_2 \ : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out})} \text{ (S)}$$

$$\frac{}{\Gamma, \Delta \vdash \mathtt{skip} \ : (\mathbf{0}, \mathbf{k}_{in}, \mathbf{k}_{out})} \text{ (SK)} \qquad \frac{\Gamma, \Delta \vdash \mathtt{c} \ : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out})}{\Gamma, \Delta \vdash \mathtt{c} \ : (\mathbf{k}{+}\mathbf{1}, \mathbf{k}_{in}, \mathbf{k}_{out})} \text{ (SUB)}$$

$$\frac{\Gamma, \Delta \vdash \mathtt{e} : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out}) \qquad \Gamma, \Delta \vdash \mathtt{c}_1 : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out}) \qquad \Gamma, \Delta \vdash \mathtt{c}_0 : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out})}{\Gamma, \Delta \vdash \mathtt{if(e)\{c_1\}\ else\ \{c_0\}} \ : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out})} \text{ (C)}$$

$$\frac{\Gamma, \Delta \vdash \mathtt{x} : (\mathbf{k}_1, \mathbf{k}_{in}, \mathbf{k}_{out}) \qquad \Gamma, \Delta \vdash \mathtt{e} : (\mathbf{k}_2, \mathbf{k}_{in}, \mathbf{k}_{out}) \qquad \mathbf{k}_1 \preceq \mathbf{k}_2}{\Gamma, \Delta \vdash \mathtt{x} := \mathtt{e} \ : (\mathbf{k}_1, \mathbf{k}_{in}, \mathbf{k}_{out})} \text{ (A)}$$

$$\frac{\Gamma, \Delta \vdash \mathtt{e} : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out}) \qquad \Gamma, \Delta \vdash \mathtt{c} : (\mathbf{k}, \mathbf{k}, \mathbf{k}_{out}) \qquad \mathbf{1} \preceq \mathbf{k} \preceq \mathbf{k}_{out}}{\Gamma, \Delta \vdash \mathtt{while(e)\{c\}} \ : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out})} \text{ (W)}$$

$$\frac{\Gamma, \Delta \vdash \mathtt{e} : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}) \qquad \Gamma, \Delta \vdash \mathtt{c} : (\mathbf{k}, \mathbf{k}, \mathbf{k}) \qquad \mathbf{1} \preceq \mathbf{k}}{\Gamma, \Delta \vdash \mathtt{while(e)\{c\}} \ : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{0})} \text{ (W_0)}$$

Fig. 1. Tiered based type System

As in [3], we define two classes of operators called neutral and positive. Let $\trianglelefteq$ be the subword relation.

- An operator $\mathtt{op}$ is *neutral* if:
  1. either $[\![\mathtt{op}]\!] : \mathbb{W}^{ar(\mathtt{op})} \to \{0, 1\}$ is a predicate;
  2. or $\forall w_1, \ldots, w_{ar(\mathtt{op})} \in \mathbb{W},\ \exists i \in \{1, \ldots, ar(\mathtt{op})\}, [\![\mathtt{op}]\!](w_1, \ldots, w_{ar(\mathtt{op})}) \trianglelefteq w_i$.
- An operator $\mathtt{op}$ is *positive* if there is a constant $c_{\mathtt{op}}$ such that:

$$\forall w_1, \ldots, w_{ar(\mathtt{op})} \in \mathbb{W},\ |[\![\mathtt{op}]\!](w_1, \ldots, w_{ar(\mathtt{op})})| \leq \max_i |w_i| + c_{\mathtt{op}}$$

A neutral operator is always a positive operator but the converse is not true. In the remainder, we name positive operators those operators that are positive but not neutral.

An operator typing environment Δ is *safe* if $\forall \mathtt{op} \in dom(\Delta)$, $\forall \mathbf{k} \in \mathbf{N}$, $\forall \mathbf{k}_1 \to \ldots \mathbf{k}_{ar(\mathtt{op})} \to \mathbf{k}' \in \Delta(\mathtt{op})(\mathbf{k})$, we have:

- $\mathbf{k}' \preceq \wedge_{i=1,\ldots,ar(\mathtt{op})}\mathbf{k}_i \preceq \vee_{i=1,\ldots,ar(\mathtt{op})}\mathbf{k}_i \preceq \mathbf{k}$,
- if the operator $\mathtt{op}$ is positive but not neutral, then $\mathbf{k}' \prec \mathbf{k}$.

Given a program $\mathtt{p}_\phi = \mathtt{c}$ `return` $\mathtt{x}$ we sometimes write $\Gamma, \Delta \vdash \mathtt{p}_\phi : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out})$ as an abuse of notation for $\Gamma, \Delta \vdash \mathtt{c} : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out})$.

Definition 3 (Safe program). *Given Γ a variable typing environment and Δ a operator typing environment, the program $\mathtt{p}_\phi = \mathtt{c}$ `return` $\mathtt{x}$ is a* safe program *if there are $\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out}$ such that $\Gamma, \Delta \vdash \mathtt{c} : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out})$ and Δ is safe.*

Example 1 (Oracles). Consider the following program with oracle ϕ returning for a given input $\mathtt{x}$ whether there exists a unary integer n of size smaller than $|\mathtt{x}|$ such that $\phi(n) = 0$.

```
y := x ;
z := 0 ;
while(x^1 >= 0){
   {if(φ(y ↾ x) == 0){z := 1} else {skip} ;
   x^1 := x − 1^1} : (1, 1, 1)(S)
}
return z
```

This program can be typed by $(\mathbf{1}, \mathbf{0}, \mathbf{0})$. The operators $==$ and $>= 0$ compute a predicate and, consequently, are a neutral operator. -1 computes a subword on unary words and is also neutral. The while loop will be typed using rule (W_0). Consequently, the inner command is typed by $(\mathbf{1}, \mathbf{1}, \mathbf{1})$. It is easy to verify that the command $\mathtt{z} := 1$, $\mathtt{skip}$ and $\mathtt{x} := \mathtt{x} - 1$ can be typed by $(\mathbf{1}, \mathbf{1}, \mathbf{1})$ using typing rules $(OP), (SK), (A)$ and (SUB). The conditional can be given the same type provided that $\Gamma, \Delta \vdash \phi(\mathtt{y} \upharpoonright \mathtt{x}) == 0 : (\mathbf{1}, \mathbf{1}, \mathbf{1})$ can be derived:

$$\dfrac{\mathbf{1} \to \mathbf{1} \in \Delta(==)(\mathbf{1}) \qquad \dfrac{\dfrac{\Gamma(\mathtt{y}) = \mathbf{0}}{\vdash \mathtt{y} : (\mathbf{0}, \mathbf{1}, \mathbf{1})}\ (\mathrm{V}) \quad \dfrac{\Gamma(\mathtt{x}) = \mathbf{1}}{\vdash \mathtt{x} : (\mathbf{1}, \mathbf{1}, \mathbf{1})}\ (\mathrm{V})}{\vdash \phi(\mathtt{y} \upharpoonright \mathtt{x}) : (\mathbf{1}, \mathbf{1}, \mathbf{1})}\ (\mathrm{OR})}{\vdash \phi(\mathtt{y} \upharpoonright \mathtt{x}) == 0 : (\mathbf{1}, \mathbf{1}, \mathbf{1})}\ (\mathrm{OP})$$

Definition 4. *Let $[\![\mathrm{ST}]\!]$ be the set of functions computed by terminating and safe programs.*

The main properties ensured by the type system are:

- a standard non-interference property ensuring that computations on higher tiers do not depend on lower tiers.
- a polynomial step count.

– a finite lookahead revision property.

and consequently, we have a soundness property:

Proposition 1 ($[\![\mathrm{ST}]\!] \subseteq \mathrm{MPT}$). *If* $\mathtt{p}_\phi \in \mathrm{ST}$*, then it computes a second order function over words in* MPT*.*

We also have a completeness result for the lambda-closure:

Proposition 2. $BFF \subseteq \lambda([\![\mathrm{ST}]\!])_2$

Completeness can be shown by simulating a variant of Cook-Urquhart recursor [1]. As a consequence,

Corollary 1. $\lambda([\![\mathrm{ST}]\!])_2 = BFF$*.*

By a reduction to 2-SAT, we also have that:

Proposition 3. *Type inference can be performed in polynomial time.*

References

1. Stephen Cook and Alasdair Urquhart. Functional interpretations of feasibly constructive arithmetic. *Annals of Pure and Applied Logic*, 63(2):103–200, 1993.
2. Bruce M. Kapron and Florian Steinberg. Type-two polynomial-time and restricted lookahead. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 579–588, 2018.
3. Jean-Yves Marion. A type system for complexity flow analysis. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada*, pages 123–132, 2011.