A tier-based typed programming language characterizing Feasible Functionals

Emmanuel Hainry  Bruce Kapron*  Jean-Yves Marion  Romain Péchoux

CNRS, Inria, Université de Lorraine - LORIA, and University of Victoria*

LICS 2020
Studies of polynomial time complexity:

- **Type-1** ($\mathbb{N} \rightarrow \mathbb{N}$): $FP_1$
  - Several tools for program analysis:
    - type systems (linear, affine, light, tiering, ...)
    - interpretations (abstract, polynomial, ...)
    - and other techniques of Implicit Computational Complexity

- **Type-2** ($(\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$) and above: $FP_2$, ...
  - No (tractable) tools
  - Programming languages with restrictions:
    - BTLP, ITLP (Irwin-Kapron-Royer [2001])

**Goal:** a static analysis technique for certifying type-2 polynomial time complexity
Introduction to type-2 complexity

Type-2 polynomial time $FP_2$ has been defined by Mehlhorn [1976].

**Theorem [Cook and Urquhart [1993]]**

$$FP_2 = \lambda(FP_1 \cup \{R\})_2$$

- $\lambda(X)_2$: type-2 restriction of the simply typed lambda closure with constants in $X$
- $\mathcal{R} : \Sigma^* \times \Sigma^* \times (\Sigma^* \rightarrow \Sigma^*) \times (\Sigma^* \rightarrow \Sigma^*) \rightarrow \Sigma^*$ is defined by:
  $$\mathcal{R}(\epsilon, a, \phi, \psi) = a$$
  $$\mathcal{R}(ix, a, \phi, \psi) = \min(\phi(ix, \mathcal{R}(x, a, \phi, \psi)), \psi(ix))$$

  min returns the operand of minimal size.
Theorem [Cook and Kapron [1990]]

The set of type-2 functionals computable by an Oracle Turing Machine (OTM) $M$ in time $P(|\phi|, |a|)$ is exactly $\text{FP}_2$.

- OTM are Turing Machines with an oracle $\phi$
- $P$ is a type-2 polynomial defined by:
  
  $$P(X_1, X_0) ::= c \in \mathbb{N} \mid X_0 \mid X_1(P) \mid P + P \mid P \times P.$$ 

- $|\phi|(n) = \max_{|x| \leq n}(|\phi(x)|)$

The class $\text{FP}_2$ is called BFF for Basic Feasible Functionals.
How to get rid of type-2 polynomials?

One option: Oracle Polynomial Time (OPT) by Cook [1992]:

**Definition**

$m_{\phi,a}^M$ is the maximum of the size of the input $a$ and of the biggest oracle’s answer in the run of $M(\phi,a)$.

**Definition**

An OTM is in OPT if it runs in time bounded by $P(m_{\phi,a}^M)$ on any input, for some type-1 polynomial $P$.

However BFF $\subset$ OPT as it contains exponential functions.
How to recover $\text{FP}_2$: finite length revision

**Definition [Finite Length Revision - Kawamura and Steinberg [2017]]**

An OTM has *Finite Length Revision* (FLR), if, for any input, the number of times the oracle answer is bigger than all of the previous oracle answers is bounded by a constant.

**Example**

```plaintext
while (x > 0) {
    y = φ(x);
    x = x - 1;
}
not (FLR) if φ ↘
```

**Example**

```plaintext
x = 0;
while (x < n && y < 8) {
    y = φ(x);
    x = x + 1;
}
(FLR) with constant 8
```
How to recover $\text{FP}_2$: finite lookahead revision

Definition [Finite LookAhead Revision - Kapron and Steinberg [2018]]

An OTM has *Finite LookAhead Revision* (FLAR), if, for any input, the number of times a query is posed whose size exceeds the size of all previous queries is bounded by a constant.

**Example**

\[
\textbf{while } (x > 0) \{
    y = \phi(x);
    x = x - 1;
\}
\]

(FLAR) with constant 0

**Example**

\[
\textbf{while } (x < n \land y < 8) \{
    y = \phi(x);
    x = x + 1;
\}
\]

not (FLAR) for $\phi = \lambda n.4$
How to recover $\text{FP}_2$?

**Definition**

- $\text{SPT} = \text{OPT} \cap \text{FLR}$
- $\text{MPT} = \text{OPT} \cap \text{FLAR}$

Both $\text{SPT} \subsetneq \text{FP}_2$ and $\text{MPT} \subsetneq \text{FP}_2$ hold.

**Theorem [Kapron and Steinberg [2018]]**

$$\text{FP}_2 = \lambda(\text{SPT})_2 = \lambda(\text{MPT})_2$$
Goal: a static analysis tool for certifying type-2 polynomial time

- tractable (no type-2 polynomial)
- automatic (polynomials are not explicitly provided)

Idea: adapt a type-1 Implicit Computational Complexity tool to type-2 and combine it with the MPT technique (FLAR ∩ OPT).

- Tool: Safe recursion and tiering
- PL: Imperative with oracles
Safe recursion and tiering

Theorem [Bellantoni and Cook [1992]]

The class of functions that contains:

- constants, projections, successor, predecessor, conditional,
- functions defined by safe composition:
  \[ f(x^1; a^0) = s(r(x^1); t(x^1; a)^0), \]
- functions defined by safe recursion:
  \[ f(\epsilon, y^1; a^0) = g(y^1; a^0) \]
  \[ f(i(x)^1, y^1; a) = h_i(x^1, y^1; f(x^1, y^1; a)^0), \quad \text{with } i \in \{0, 1\}, \]
  provided \( s, r, t, g, h_i \) are already defined in the class,

is exactly \( \text{FP}_1 \).
Tiering for imperative PL

Imperative language over binary words $\Sigma^*$

\[
E ::= x \mid \text{true} \mid \text{false} \mid \text{op}(E,\ldots,E) \\
I ::= [x:=E]; \mid I I \mid \text{while}(E)\{I\} \mid \text{if}(E)\{I\}\text{else}\{I\}
\]

Tier $\tau \in \{0, 1\}$ with $0 < 1$

Intuition:
- **0**: data may grow and cannot control the program flow.
- **1**: data cannot grow and may control the program flow.
Typing rules

\[
\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \quad (\text{Var}) \quad \frac{\Gamma \vdash e : \tau}{\Gamma \vdash op(e) : \tau} \quad (\text{Des}) \quad \frac{\Gamma \vdash e : \tau}{\Gamma \vdash op(e) : 0} \quad (\text{Cons})
\]

\[
\frac{\Gamma \vdash c : \tau}{\Gamma \vdash Cst} \quad \frac{\Gamma \vdash l : \tau, \tau \leq \tau'}{\Gamma \vdash l : \tau'} \quad (\text{Sub})
\]

\[
\frac{\Gamma \vdash l_1 : \tau, \Gamma \vdash l_2 : \tau}{\Gamma \vdash l_1 \; l_2 : \tau} \quad (\text{Seq}) \quad \frac{\Gamma \vdash e : \tau, \Gamma \vdash l_i : \tau}{\Gamma \vdash \text{if}(E)\{l_1\} \text{else}\{l_2\} : \tau} \quad (\text{If})
\]

\[
\frac{\Gamma \vdash x : \tau, \Gamma \vdash E : \tau', \tau \leq \tau'}{\Gamma \vdash x := E : \tau} \quad (A) \quad \frac{\Gamma \vdash E : 1, \Gamma \vdash l : \tau}{\Gamma \vdash \text{while}(E)\{l\} : 1} \quad (\text{Wh})
\]
Safe operators

Extension to polynomial time computable operators:

\[ op :: \tau_1 \times \ldots \times \tau_n \rightarrow \tau \]

- Neutral operators computing a predicate:
  \[ \tau \leq \min_{i \in [1,n]} \tau_i \]

- Positive operators satisfying:
  \[ \forall w, \|op\| (w_1, \ldots, w_n) \leq \max_{i \in [1,n]} |w_i| + c, \text{ for } c \geq 0 \]

\[ \tau = 0 \]
Example: addition

Example (add :: int × int → int)

add(x, y)
{
    while (x > 0)
    {
        x = x - 1;
        y = y + 1;
    }
    return y;
}

- y is necessarily of tier 0.
- x is necessarily of tier 1.
- Consequently, add is typed by 1 × 0 → 0.
Example: multiplication

```
Example (mult :: int × int → int)

mult (x, y) {
    int z = 0;
    while (x > 0) {
        x = x - 1;
        z = add(y, z); // add: 1 × 0 → 0
    }
    return z;
}
```

- The output of add is 0. Consequently, z is of tier 0.
- Both x and y are of tier 1.
- Consequently, mult is typed by 1 × 1 → 0.
Counter-example: exponential

Example (exp :: int → int)

```c
exp (x) {
    int y = 1;
    while (x > 0) {
        x = x - 1;
        z = y;
        y^0 = add (y^1, z);  // add : 1 × 0 → 0
    }
    return y;
}
```

- The tier of y cannot be defined.
- Consequently, exp do not type.
Results

Theorem [Marion [2011]]

The set of functions computable by a typable and terminating program is exactly $\text{FP}_1$.

- **Soundness:**
  - No flow from 0 to 1 (guards of tier 1)
  - At most $n^k$ configurations under termination assumption

- **Completeness:**
  - Simulation of a polynomial time TM

Theorem [Hainry, Marion and Péchoux [2013]]

Type inference can be done in polynomial time.

- Reduction to 2-SAT
Imperative language with oracles

Design a type system ensuring that programs are in MPT.

PL with oracles

\[ E ::= x \mid \text{true} \mid \text{false} \mid \text{op}(E, \ldots, E) \mid \phi(E \upharpoonright E) \]

\[ I ::= [x := E]; \mid I I \mid \text{while}(E)\{I\} \mid \text{if}(E)\{I\}\text{else}\{I\} \]

In an oracle call \( \phi(w \upharpoonright v) \):

- \( w \) is the **oracle input**.
- \( v \) is the **oracle input bound**.
- If \( |v| \geq k \) then \( w \upharpoonright v = w_1 \ldots w_{|v|} \).
Towards a type system for MPT

Observations:
1. The number of lookahead revisions can be controlled by tiers.
2. A restriction on the oracle input bound is needed.
3. Operators are in need of a more flexible treatment.

Solutions:
1. Use more than two tiers: \( \{0, 1, 2, 3, \ldots, k, \ldots\} \).
2. Keep track of the tier of the outermost while \( k_{out} \).
3. Keep track of the tier of the innermost while \( k_{in} \).

Judgments: \( \Gamma, \Delta \vdash I : (k, k_{in}, k_{out}) \)
Type system (easy)

\[
\begin{align*}
\Gamma(x) &= k & \forall i \in \{1, 2\}, & \vdash I_i : (k, k_{in}, k_{out}) \\
\Gamma, \Delta \vdash x : (k, k_{in}, k_{out}) & \quad \text{(VAR)} & \vdash I_1 \ I_2 : (k, k_{in}, k_{out}) & \quad \text{(SEQ)} \\
\vdash \_ : (0, k_{in}, k_{out}) & \quad \text{(SK)} & \vdash I : (k, k_{in}, k_{out}) & \quad \text{(SUB)} \\
\vdash E : (k, k_{in}, k_{out}) & \quad \forall i \in \{1, 2\}, & \vdash I_i : (k, k_{in}, k_{out}) & \quad \text{(IF)} \\
\vdash \text{if}(E)\{I_1\} \ \text{else} \ \{I_2\} : (k, k_{in}, k_{out}) & \ \vdash x : (k_1, k_{in}, k_{out}) & \vdash E : (k_2, k_{in}, k_{out}) & \quad k_1 \preceq k_2 \\
\vdash x := E : (k_1, k_{in}, k_{out}) & \quad \text{(ASG)}
\end{align*}
\]
Type system (hard)

\[ k_1 \rightarrow \cdots \rightarrow k_n \rightarrow k \in \Delta(op)(k_{in}) \quad \forall i, \quad \vdash E_i : (k_i, k_{in}, k_{out}) \]

\[ \Gamma, \Delta \vdash op(E_1, \ldots, E_n) : (k, k_{in}, k_{out}) \]

with \( k_1 \rightarrow \cdots \rightarrow k_n \rightarrow k \in \Delta(op)(k_{in}) \) if:

- \( k \leq \min_{i \in [1,n]} k_i \) and \( \max_{i \in [1,n]} k_i \leq k_{in} \)
- \( k < k_{in} \) for positive operators.

\[ \vdash E : (k, k_{in}, k_{out}) \quad \vdash E' : (k_{out}, k_{in}, k_{out}) \quad k < k_{in} \quad k \leq k_{out} \]

\[ \vdash \phi(E \upharpoonright E') : (k, k_{in}, k_{out}) \]

\[ \vdash E : (k, k_{in}, k_{out}) \quad \vdash I : (k, k, k_{out}) \quad 1 \leq k \leq k_{out} \]

\[ \vdash \text{while}(E)\{I\} : (k, k_{in}, k_{out}) \]
Example

The program computes the decision problem \( \exists n \leq x, \phi(n) = 0 \).

\[ y = x; \]
\[ z = false; \]
\[ while(x^1 >= 0) \{
  \quad if(\phi(y^0 \upharpoonright x^1) == 0) \{
    \quad z^0 = true;
  \quad \} else \{ ; \}
  \quad x^1 = x^1 - 1;
\}\]

\[ return z; \]

- The program is in MPT.
- The program is typable and the inner command has tier \((1, 1, 1)\).
False negative

Example

The program computes the decision problem $\exists n \leq x, \phi(n) = 0$.

```plaintext
x := \epsilon ;
z := 0 ;
while (y >= x) ^{k} {
    if (\phi(y | x) == 0) {z := 1} else {;}
    x := x + 1 ; : (k, k, k')
}
return z ;
```

- $x$ and $y$ have tier at least $k$ in the guard.
- $x$ is of tier strictly less than the inner tier $k$ as $+1$ is positive.
- But it is not in FLAR.
Let ST be the class of typable and terminating programs.

**Theorem [Soundness]**

\[ ST \subseteq \lambda(MPT)_2 \]

**Theorem [Completenesses]**

- \( ST_1 = FP_1 \)
- \( \lambda(ST)_2 = FP_2 \)

By simulating a variant of \( \mathcal{R} \).
Conclusion

We have presented:

▶ a completeness result at type-1,
▶ a completeness result at type-2 for a strict natural extension,
▶ a decidable type inference (in polynomial time).

Completeness is preserved for some decidable termination techniques (size-change principle, Lee-Jones-Ben-Amram[2001]).

Open issues

▶ How to get rid of the lambda-closure?
▶ What are the completeness preserving termination techniques?
▶ Are there sound extensions to capture more false negatives?