Tiered complexity at higher order

Emmanuel Hainry  Bruce Kapron*  Jean-Yves Marion  
Romain Péchoux

LORIA, Université de Lorraine and University of Victoria*

湘南国際村センター 2019
Introduction

Study of polynomial time complexity:

- **Type-1** \((\mathbb{N} \rightarrow \mathbb{N})\):
  - Several tools for program analysis:
    - type systems (light logics),
    - interpretations (abstract, polynomial, ...),
    - ...

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

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

Type-2 polynomial time $\text{FP}_2$ has been defined by Mehlhorn [1976].

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

$$\text{FP}_2 = \lambda(\text{FP}_1 \cup \{\mathcal{R}\})_2$$

- $\text{FP}_1$ is the class of type-1 polynomial time functions,
- $\mathcal{R} : \Sigma^* \times \Sigma^* \times (\Sigma^* \rightarrow \Sigma^*) \times (\Sigma^* \rightarrow \Sigma^*) \rightarrow \Sigma^*$ 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.
Basic Feasible Functionals

Theorem [OTM based characterization by Cook-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**

\[ m^M_{\phi,a} \text{ is the maximum of the size of the input } a \text{ and of the biggest oracle’s answer in the run of } M(\phi,a). \]

**Definition**

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

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

**Definition [Finite Length Revision]**

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**

```
while (x > 0)
{
    y = \phi(x);
    x = x - 1;
}
```

not (FLR) if $\phi \downarrow$

**Example**

```
while (x < n && y < 8)
{
    y = \phi(x);
    x = x + 1;
}
```

(FLR) with constant 8
How to recover \( \text{FP}_2 \): finite lookahead revision

**Definition [Finite LookAhead Revision]**

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**

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

(FLAR) with constant 0

**Example**

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

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 $\text{MPT} \subsetneq \text{FP}_2$.

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

$$\text{FP}_2 = \lambda(\text{SPT})_2 = \lambda(\text{MPT})_2$$
Motivations

- Find a criterion for complexity certificates.
- Provide a characterization of $\mathbb{FP}_2$ on imperative languages.
- Develop a static analysis technique with polynomial bounds:
  - of type-1 (Hilbert’s 10th pb, Tarski’s Quantifier Elimination)
  - implicit (not explicitly provided)

Objective: Adapt Implicit Computational Complexity techniques to an imperative setting with oracles.

Tool: Safe recursion and Tiering
Safe recursion

Theorem [Bellantoni-Cook[1992]]

The class of functions:

- constants, projections, successor, predecessor, conditional,
- defined by safe composition:
  \[ f(\overline{x}^1; \overline{a}^0) = s(r(\overline{x}^1); t(\overline{x}^1; \overline{a})^0) \]

- and defined by safe recursion:
  \[ f(\epsilon, \overline{y}^1; \overline{a}^0) = g(\overline{y}^1; \overline{a}^0) \]
  \[ f(i(\overline{x})^1, \overline{y}^1; \overline{a}) = h_i(\overline{x}^1, \overline{y}^1; f(\overline{x}^1, \overline{y}^1; \overline{a})^0) \quad i \in \{0, 1\}, \]

provided \( s, r, t, g, h_i \) are already defined in the class, is exactly \( \text{FP}_1 \).
Tiering

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

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

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

\[\frac{\Gamma \vdash I_1 : \tau \quad \Gamma \vdash I_2 : \tau}{\Gamma \vdash I_1 \ I_2 : \tau} \quad \text{(Seq)} \quad \frac{\Gamma \vdash e : \tau \quad \Gamma \vdash I_i : \tau}{\Gamma \vdash \text{if}(E)\{I_1\}\text{else}\{I_2\} : \tau} \quad \text{(If)}\]

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

Extension to polynomial time computable operators:

\[ \text{op} :: \tau_1 \times \ldots \times \tau_n \rightarrow \tau \]

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

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

Example \((add :: int \times int \rightarrow int)\)

```
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, \(add :: 1 \times 0 \rightarrow 0\).
Example: multiplication

Example ($\text{mult :: int} \times \text{int} \to \text{int}$)

```plaintext
\text{mult}(x, y)\
  \{\text{int } z = 0;\
   \text{while } (x > 0)\{\
     x = x - 1;\
     z = \text{add}(y, z); // add: }1 \times 0 \to 0\
   \}\text{return } z;\}
```

▶ the output of add is 0. Consequently, $z$ is of tier 0.
▶ both $x$ and $y$ are of tier 1.
▶ consequently, $\text{mult :: }1 \times 1 \to 0$. 
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!
Results

Theorem [Marion [2011]]

The set of functions computable by a typable and terminating program with safe operators 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 $MPT = OPT \cap FLAR$.

$E ::= x \mid true \mid false \mid op(E, \ldots, E) \mid \phi(E \upharpoonright E)$

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

$\phi(w \upharpoonright v)$:

- $w$ is the oracle input
- $v$ is the oracle input bound
- $w \upharpoonright v = w_1 \ldots w_{|v|}$, if $|v| \geq k$
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)

\[ \Gamma(x) = k \]
\[ \Gamma, \Delta \vdash x : (k, k_{in}, k_{out}) \]
\[ \forall i \in \{1, 2\}, \vdash l_i : (k, k_{in}, k_{out}) \]
\[ \vdash l_1 \cdot l_2 : (k, k_{in}, k_{out}) \] (SEQ)

\[ \vdash ; : (0, k_{in}, k_{out}) \] (SK)
\[ \vdash l : (k, k_{in}, k_{out}) \]
\[ \vdash l : (k+1, k_{in}, k_{out}) \] (SUB)

\[ \vdash E : (k, k_{in}, k_{out}) \]
\[ \forall i \in \{1, 2\}, \vdash l_i : (k, k_{in}, k_{out}) \]
\[ \vdash \text{if}(E)\{l_1\} \text{ else } \{l_2\} : (k, k_{in}, k_{out}) \] (IF)

\[ \vdash x : (k_1, k_{in}, k_{out}) \]
\[ \vdash E : (k_2, k_{in}, k_{out}) \]
\[ k_1 \preceq k_2 \]
\[ \vdash x := E : (k_1, k_{in}, k_{out}) \] (ASG)
Type system (hard)

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

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

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.

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

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

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

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

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

```plaintext
y = x;
z = false;
while (x^1 >= 0)
    if (\phi(y^0 | x^1) == 0)
        z^0 = true;
    else
        x^1 = x^1 - 1;
return z;
```

The program is in MPT.

The program is typable and the inner command has tier $(1,1,1)$. 
A more complex example

Example

\[ \sum_{i=0}^{\max^n_{x=0}} \phi(x) \phi(i) \] can be computed by:

\[
\begin{align*}
x &:= n ; \\
y^2 &:= x^3 ; \\
z^2 &:= 0 ; \\
\text{while}(x^3 >= 0) &\{} \\
&\quad z^2 := \max(\phi(y^2 \upharpoonright x^3)^2, z^2) ; \\
&\quad x^3 := x - 1^3 ; \\
&\{} ; \\
v^1 &:= z^2 ; \\
u^0 &:= 0 ; \\
\text{while}(z^2 >= 0) &\{} \\
&\quad w^1 := \phi(v^1 \upharpoonright z^2)^1 ; \\
&\quad \text{while}(w^1 >= 0) &\{} \\
&\quad &\quad u^0 := u + 1^0 ; \\
&\quad &\quad w^1 := w - 1^1 ; \\
&\quad &\} ; \\
&\quad z^2 := z^2 - 1 ; \\
&\} \\
\text{return u ;}
\end{align*}
\]

This program can be typed by \((3, 0, 0)\).
False negative

Example

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

```plaintext
x := e;
z := 0;
while(y >= x)^k{
    if(\phi(y \upharpoonright 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 [Completeness]**

$ST_1 = FP_1$

$\lambda(ST)_2 = FP_2$.

By simulating a variant of $R$. 
Conclusion

We have presented:

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

Drawbacks and Open questions

- Termination is assumed.
- Completeness is obtained under lambda-closure.