Implicit characterization of the class of Basic Feasible Functionals

Romain Péchoux

CNRS, Inria, Université de Lorraine - LORIA,
* joint work with Emmanuel Hainry, Bruce Kapron and Jean-Yves Marion

Complexity Days 2023
Motivations

We aim at providing characterizations of complexity classes:

- **machine-independent**,  
- with **no prior knowledge** on the complexity of analyzed codes.

If the characterization is **tractable** then we obtain an **automated complexity analysis** for a high-level programming language.

State of the art:

- 30 years of intensive research,  
- hundreds of publications,  
- some tools: Costa, SPEED, TcT, ...
**Motivations**

Type-2 complexity

Tier-based typing

Tractable characterizations of BFF

Conclusion

---

**The Implicit Computational Complexity approach**

**Methodology**

Consider your favorite programming language $\mathcal{L}$ and your favorite complexity class $\mathcal{C}$.

Find a tractable restriction $\mathcal{R} \subseteq \mathcal{L}$ such that $\mathcal{J}[\mathcal{R}] = \mathcal{C}$,

where $\mathcal{J}[\mathcal{X}]$ is the set of functions computed by programs in $\mathcal{X}$.

---

**Examples of (type-1) complexity class $\mathcal{C}$**

- P, FP,
- PSPACE, FPSPACE,
- NP,
- PP, BPP, EQP, BQP,
- ...

**Examples of restriction $\mathcal{R}$**

- type systems:
  - linear logic, sized types, ...
- interpretation methods,
- amortized resource analysis,
- ...

---

Romain Péchoux  
CNRS, Inria, UL, LORIA  
Implicit characterization of BFF
What about type-2 complexity classes?

**Type-2** objects are functions in \((\mathbb{N} \rightarrow \mathbb{N}) \times \mathbb{N} \rightarrow \mathbb{N}\).

**Type-2** polynomial time is taken to be the class of Basic Feasible Functionals (BFF).

**Goal (Open problem for more than 20 years)**

Find a **tractable** technique for certifying **type-2 polynomial time** complexity.

Rephrasing: Find a tractable restriction \(\mathcal{R}\) such that \([\mathcal{R}] = \text{BFF}\).

N.B.: The problem was solved for **type-1** polytime FP by Bellantoni and Cook in 1992.
A reminder on type-2 polynomial time

BFF was introduced by Melhorn in 1976.

**Theorem [Cook and Urquhart [1989]]**

\[ \text{BFF} = \lambda (\text{FP} \cup \{ \mathcal{I} \})_2 \]

\( \mathcal{I} \) is a type-2 bounded iterator:

\[ \mathcal{I}^f.g(\epsilon, a) = a \]
\[ \mathcal{I}^f.g(ix, a) = \min(f(ix, \mathcal{I}^f.g(x, a)), g(ix)) \]

\( \lambda(X)_2 \): type-2 restriction of the simply-typed lambda-closure using constants in \( X \).

**Theorem [Kapron and Cook [FOCS1991]]**

The set of functionals computable by an Oracle TM (OTM) in time \( P(|\phi|, |a|) \) is exactly BFF.

Type-2 polynomials and size function are defined by:

- \( P(X_1, X_0) ::= c \in \mathbb{N} \mid X_0 \mid P + P \mid P \times P \mid X_1(P) \)
- \( |\phi|(n) = \max_{|x| \leq n} |\phi(x)| \)
How to get rid of type-2 polynomials?

→ Type-2 polynomials are not tractable.

**Definition [Oracle Polynomial Time (OPT) – Cook [1992]]**

Let $n^\phi, A$ be the biggest size of $a$ and of an oracle’s answer in the run of $M(\phi, a)$. The OTM $M$ is in OPT if its runtime is bounded by $P(n^\phi, a)$, for a type-1 polynomial $P$.

BFF $\not\subseteq$ OPT, as OPT contains exponential functions.

**Theorem [Kapron and Steinberg [LICS2018]]**

$\text{BFF} = \lambda([\text{OPT} \cap \text{FLAR}])_2$

→ FLAR = Finite LookAhead Revision
Finite LookAhead Revision

Definition [Finite LookAhead Revision - Kapron and Steinberg [LICS2018]]

An OTM is in 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

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

in FLAR.
The constant bound is 0.

Example

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

not in FLAR for \(\phi = \lambda x.4\)
but is in FLR (I will briefly mention this class in the conclusion)
How to get rid of (Oracle Turing) machines?

→ Design a typed PL ensuring that computed functions are in $\text{OPT} \cap \text{FLAR}$.

Imperative PL on words with oracles

Expressions $\ni E ::= x \mid \text{true} \mid \text{false} \mid \text{op}(E, \ldots, E) \mid \phi(E \uparrow E)$

Commands $\ni C ::= x := E \mid C \cdot C \mid \text{if}(E)\{C\} \text{else}\{C\} \mid \text{while}(E)\{C\}$

In an oracle call $\phi(w \uparrow v)$:

- $\phi$ computes a type-1 function on words, i.e. $\phi \in \mathbb{W} \rightarrow \mathbb{W}$.
- $w$ is the oracle input.
- $v$ is the input bound: $w \uparrow v = w_1 \ldots w_{|v|}$. 
Tier-based type discipline

Tiers $k, k', ...$ are security levels (in $\mathbb{N}$) assigned to Expressions and Commands.

The type system ensures some non-interference properties.

In a tier $k$ command:
- the program flow cannot be controlled by expressions of a lower tier $k^- < k$,
- data of upper tier $k^+ \geq k$ cannot increase (in size).

Judgments: $\Gamma, \Delta \vdash C : (k, k_{in}, k_{out})$ with $(k, k_{in}, k_{out}) \in \mathbb{N}^3$

1. The tier $k$ implements the non-interference policy.
2. The innermost tier $k_{in}$ is used for declassification.
3. The outermost tier $k_{out}$ is used to ensure FLAR on oracle calls.
Tier-based type system: an overview

Typing rules

\[ \vdash x : (k_1, k_{in}, k_{out}) \quad \vdash E : (k_2, k_{in}, k_{out}) \quad k_1 \leq k_2 \]  
(Asg)

\[ \vdash x := E : (k_1, k_{in}, k_{out}) \]

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

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

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

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

\[ \vdots \]
Illustrating example

Program computing 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 typable and the while body has tier $(1, 1, 1)$.
- The computed function is in $OPT \cap FLAR$. 
A tier-based characterization of BFF

Let SAFE be the set of typable programs.
Let SN be the set of strongly normalizing programs.
Let \([X]\) be the set of functions computed by programs in X.

**Theorem [Hainry-Kapron-Marion-Péchoux [LICS2020]]**

\[
\text{BFF} = \lambda([\text{SAFE} \cap \text{SN}])_2
\]

**Main drawbacks:**

- Lambda closure (for completeness)
- Termination assumption (for soundness)
How to get rid of the lambda-closure?

Naïve idea: internalize lambda-abstraction and application into the language. 
→ cannot be done straightforwardly as it breaks soundness.

Extended language (\(e_i\): \(e\) is a type-i object)

*(Expressions)*  
\[ E ::= x_0 \mid \text{op}(E, \ldots, E) \mid x_1(E \upharpoonright E) \]

*(Statements)*  
\[ C ::= [x_0 := E] ; \mid C \ C \mid \text{if}(E)\{C\}\{C\} \mid \text{while}(E)\{C\} \]

*(Procedures)*  
\[ P ::= P(x_1, x_0)\{C \ \text{return} \ x_0\} \]

*(Terms)*  
\[ t ::= x \mid \lambda x.t \mid t @ t \mid \text{call} \ P(\{x_0 \rightarrow t_0\}, t_0) \]

*(Programs)*  
\[ \text{prog} ::= t_0 \mid \text{declare} \ P \ \text{in prog} \]

Solution: type-1 arguments in a procedure call are restricted to closures \(\{x_0 \rightarrow t_0\}\).
The extended type system just consists of two layers:

- SAFE procedures (using the HKMP[LICS2020] paper),
- Simply-typed terms on words $\mathbb{W}$.

**Definitions**

A program is a **type-i** program if all its $\lambda$-abstractions are of order $\leq i$.

- $\text{SAFE}_i$ is the set of type-i typable programs.
  - Remark: $\text{SAFE}_0$ is the set of typable programs without lambda-abstraction.
- $\text{SN}$ is still the set of strongly normalizing programs.
Example

\[ \text{prog}(\phi, w) \triangleq \text{declare} \text{KS}(Y, v) \{ \]
\[ u := 10; \]
\[ z := \varepsilon; \]
\[ \text{while } (v^1 \neq 0) \{ \quad // \quad k_{in} = k_{out} = 1 \]
\[ v^1 := v - 1; \]
\[ z^0 := Y(z^0 \upharpoonright u^1) \]
\[ \} \]
\[ \text{return } z \]
\[ \} \]
\[ \text{in call} \text{KS}({x \rightarrow \phi @ (\phi @ x)}), w) \]

\begin{itemize}
  \item \([\text{prog}] \in (W \rightarrow W) \rightarrow W \rightarrow W\)
  \item \([\text{prog}](\phi^W \rightarrow W, w^W) = F_{|w|}(\phi) \) with \( \begin{cases} 
  F_0(\phi) = \varepsilon \\
  F_{n+1}(\phi) = (\phi \circ \phi)(F_n(\phi) \leq |10|) 
\end{cases} \)
  \item \text{prog} \in \text{SAFE}_0 \cap \text{SN} \text{ whereas } [\text{prog}] \not\in \text{OPT} \cap \text{FLAR}. \)
\end{itemize}
First characterizations of BFF

Characterizations without external lambda-closure:

Theorem [Hainry-Kapron-Marion-Péchoux [FoSSaCS22]]

∀ i ≥ 0, \( \text{SAFE}_i \cap \text{SN} \) = BFF

Surprisingly, the internal lambda-abstraction is not required for completeness.

→ Can we weaken the SN requirement?
How to get rid of Strong Normalization?

We consider Size Change Termination (SCT).

General idea

Program:

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

Size change graph abstraction:

```plaintext
\begin{pmatrix}
  x & \rightarrow & x \\
  y & & y
\end{pmatrix}^\omega
```

Theorem [Lee, Jones, and Ben Amram [POPL2001]]

“If every infinite computation would give rise to an infinitely decreasing value sequence in the size-change graph, then no infinite computation is possible.”

→ SCT is not “tractable”: PSPACE-complete.
Tractable characterizations of BFF

Completeness is preserved for SCT and for an instance SCP (Ben Amram-Lee [2007]).

**Theorem**

\[ \forall i \geq 0, \ [\text{SAFE}_i \cap \text{SCP}_S] = \text{BFF} \]

SCP\(_S\) can be decided in time quadratic in the program size.

**Theorem [Type inference]**

\[ \text{prog} \in \bigcup_i \text{SAFE}_i \cap \text{SCP}_S \text{ is Ptime-complete (using Mairson[2004]).} \]
\[ \text{prog} \in \text{SAFE}_0 \cap \text{SCP}_S \text{ is in time cubic in } |\text{prog}| \text{ (using HKMP[2022]).} \]
Conclusion

We have obtained **sound** and **complete** characterizations of type-2 polynomial time:

- **machine-independent** (a typed programming language with procedure calls)
- **implicit** (no prior knowledge on the bound is required),
- **tractable** (decidable type inference in polytime) \(\Rightarrow\) it can be automated.

Open issues

- **expressive power** (capture more false negatives),
- **extension to Finite Length Revision** (harder, WIP using declassification).
Thank you for your attention!