A general noninterference policy for polynomial time

Emmanuel Hainry and Romain Péchoux
Inria team Mocqua - CNRS, Inria, Université de Lorraine - LORIA

SCOT Seminar

January 13th, 2023
Implicit Computational Complexity (ICC):
- analyzes resource usage
  - time/space, communications, energy, ...
- provides complexity classes characterizations:
  - machine-independent
  - implicit (no prior knowledge)

**Tractability** gives an automatic Static Analyzer.

State of the art:
- 30 years of intensive research,
- hundreds of publications,
- some academic tools (Costa, SPEED, TcT, ...).
The ICC approach

ICC criterion

Take your favourite Programming Language $\mathcal{L}$ and your favorite complexity class $\mathcal{C}$:

$$\mathcal{R} \subseteq \mathcal{L} \text{ is an ICC criterion if } \{[p] \mid p \in \mathcal{R}\} = \mathcal{C}.$$
A bunch of techniques

Non-exhaustively:

- **function algebra**: [Cobham65], [Bellantoni-Cook92], ...

- **linear logic** based approaches
  - light logics: LLL [Girard87], ILAL [Asperti-Roversi02], DLAL [Baillot-Terui04],
  - soft logics: SLL [Lafont04], STA [Gaboardi-Ronchi Della Rocca07],
  - non size-increasing [Hofmann99].

- **potential** based methods
  - interpretations: [Bonfante-Marion-Moyen11],[Marion-P.09]
  - amortized resource analysis: [Jost et al.10], [Hoffmann-Hofmann10],
  - sized-types: [Vasconcelos08], [Avanzini-Dal Lago17],
  - cost semantics: [Danner et al.15].

- ...
What about Noninterference (NI)?

Noninterference [Smith, AIS 08]

- $M$ is a memory configuration; $M_L/M_H$ being its projections on low/high parts.
- A program $P$ is **noninterfering** if $\forall M$, $\forall N$,

$$ (M_L = N_L \land \langle P, M \rangle \rightarrow^* M' \land \langle P, N \rangle \rightarrow^* N') \implies M'_L = N'_L. $$

- The underlying security order is $L \subseteq H$.

NI for complexity [Marion, LICS 11]

- $M$ is a memory configuration; $M_0/M_1$ being its projections on low and high levels.
- A program $P$ is **noninterfering** if $\forall M$, $\forall N$,

$$ (M_1 = N_1 \land \langle P, M \rangle \rightarrow^* M' \land \langle P, N \rangle \rightarrow^* N') \implies M'_1 = N'_1. $$

- The underlying complexity order is $0 \leq 1$. 
Polynomial time: \( \text{NI} + \text{complexity restrictions} \)

SAFE program

- **1** is the level of data that:
  - can drive iteration/recursion
  - cannot increase
  - lies in a space of size polynomial in the input size

- **0** is the level of data that:
  - cannot drive iteration/recursion
  - can increase (by at most a constant)

- There is no flow from 0 to 1.

Theorem [Polytime Soundness & Completeness]

\[ [\text{SAFE} \cap \text{SN}] = \text{FP}. \]
Type system

SAFETY is ensured by typing: $0 \leq 1$, $\tau \in \{0, 1\}$.

$$
\begin{align*}
\Gamma(x) &= \tau \\
\Gamma \vdash x : \tau
\end{align*}
$$

$$
\begin{align*}
\Gamma \vdash e : \tau \\
\Gamma \vdash e - 1 : \tau
\end{align*}
$$

$$
\begin{align*}
\Gamma \vdash e : \tau \\
\Gamma \vdash e + 1 : 0
\end{align*}
$$

$$
\begin{align*}
\Gamma \vdash x : \tau \\
\Gamma \vdash e : \tau' \\
\tau \leq \tau'
\end{align*}
$$

$$
\begin{align*}
\Gamma \vdash x := e : \tau
\end{align*}
$$

$$
\begin{align*}
\Gamma \vdash st_1 : \tau \\
\Gamma \vdash st_2 : \tau
\end{align*}
$$

$$
\begin{align*}
\Gamma \vdash st_1 \cdot st_2 : \tau
\end{align*}
$$

$$
\begin{align*}
\Gamma \vdash e : \tau \\
\Gamma \vdash st_1 : \tau \\
\Gamma \vdash st_2 : \tau
\end{align*}
$$

$$
\begin{align*}
\Gamma \vdash st_1 \cdot st_2 : \tau
\end{align*}
$$

$$
\begin{align*}
\Gamma \vdash e : 1 \\
\Gamma \vdash st : \tau
\end{align*}
$$

$$
\begin{align*}
\Gamma \vdash \text{while}(e)\{st\} : 1
\end{align*}
$$

Theorem [Hainry, Marion, P., FoSSaCS 23]

Type inference is tractable.
Illustrating toy examples

Assume that \( \text{add} :: 1 \times 0 \rightarrow 0 \)

```c
int mult(int x, int y)
{
    int z = 0;
    while (x > 0) {
        x^1 = x - 1^1;
        z^0 = add(y^1, z^0);
    }
    return z;
}
```

can be typed as \( \text{mult} :: 1 \times 1 \rightarrow 0 \)

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

cannot be typed...
A fruitful technique

This technique can be adapted to finitely many levels: 0, 1, 2, ... This technique has been used to characterize:

- **FPSPACE** on fork processes [Hainry-Marion-P., FoSSaCS 13]
- **FP** on multi-threads [Marion-P., TAMC 14]
- **BFF** on imperative programs [Hainry-Kapron-Marion-P. LICS 20, FoSSaCS 22]

This technique has been extended to:

- programs on **Graphs** [Leivant-Marion, ICALP 13]
- **Object-Oriented** programs [Hainry-P., APLAS 15]
- **Java** programs: **COMPLEXITYPARSER** [Hainry-Jeandel-P.-Zeyen, ICTAC 21]
Adaptation to OO: not like taking candy off a baby (1/2)

\[ x^0 := y^1 \]

- primitive data (pass-by-value):

- reference data (pass-by-reference):
Major problems on complex data structures (graphs or objects)

- NI can be broken by side effect (using a pass-by-reference strategy).
- The space of level 1 configurations is no longer polynomial in the size of the inputs.

Two solutions

- A syntactical restriction in [Leivant-Marion, ICALP 13]:
  - the number (up to isomorphism) of digraphs of outdegree 1 with $n$ vertices and a generator of size $k$, is at most $n^{2k^2}$.
- A restricted flow in [Hainry-P., I&C 18]:
  - only cloned data can flow from 1 to 0: $(\mathit{x}^1.\mathit{clone()})^0$. 
A simple counterexample

```plaintext
reverse on List {int hd; List tl}

y = null;
while (x ≠ null) {
  z = y;
  y₀ = x¹; //Prohibited: needs explicit cloning
  x = x.tl;
  y.tl₀ = z;
}
```

Theorem [Hájek, TCS 79]

Providing an intensionally-complete characterization of FP is a $\Sigma^2_0$-complete problem.

→ But there might be some happy medium...
Main idea: separate NI from complexity related issues

\[ \Gamma(s) = n \quad \text{(Id)} \]

\[ \Gamma \vdash_{NI} s : n \]

\[ \forall i \leq |\bar{e}|, \quad \Gamma \vdash_{NI} e_i : n \quad \text{(Op)} \]

\[ \Gamma \vdash_{NI} \text{null} : n \quad \text{(Null)} \]

\[ \forall i \leq |\bar{e}|, \quad \Gamma \vdash_{NI} e_i : n \quad \text{(New)} \]

\[ \Gamma \vdash_{NI} \text{new } C(\bar{e}) : n \quad \text{(SubE)} \]

\[ \Gamma \vdash_{NI} \text{null} : n \quad \text{(Asg)} \]

\[ \Gamma \vdash_{NI} e : n \]

\[ \Gamma(s) = n \quad \Gamma \vdash_{NI} e : m \quad \text{(Sub)} \]

\[ \Gamma \vdash_{NI} \text{while}(e)\{s\} : n \quad \text{(Iter)} \]

\[ \Gamma \vdash_{NI} \text{while}(e)\{s\} : n \quad \Gamma \vdash_{NI} st : m \quad \text{(Sub)} \]

\[ \rightarrow \text{NI}_\Gamma \text{ is the set of programs that types wrt } \Gamma. \]
Memory representation as digraphs

- $H = (V, A)$ is a standard digraph representing the OO memory:
  - $V = \{v \in References\} \cup \{x \in Var\}$,
  - $A = \{(x, v), (v, v')\}$

- Given $\Gamma$, vertices in $H$ can be annotated by levels $f \in V \rightarrow \mathbb{N}$
  - $\forall (v, v') \in H, f(v) \leq f(v')$
  - $\forall (x, v) \in H, \Gamma(x) \leq f(v)$

- $H_{\leq n}$ is the subgraph of $H$ obtained by removing vertices of level $< n$.

- $H_1 \subseteq_n H_2$ holds if $H_1^{\leq n}$ is a subgraph of $H_2^{\leq n}$. 
Stratification

- Conf $\triangleq \{(st, H)\}$
- $\leftarrow_n \in \text{Conf} \rightarrow \text{Conf}$.
- $n$ is the minimal level of a while loop guard encompassing the executed statement.
- $R(P) \subseteq \text{Conf}$, the reachable configurations of $P$.

**Definition**

A program $P \in NI\Gamma$ is *stratified* if for any $(st, H) \in R(P)$,

$$(st, H) \leftarrow_{n>0} (st', H') \text{ implies } H' \subseteq_n H.$$

Let STR be the set of stratified programs.
Illustrating example

\[ \Gamma(x) = 1, \Gamma(y) = \Gamma(z) = 0. \]

\text{reverse} \in \text{NI}_\Gamma

\[ y = \text{null}; \]
\[ \text{while} \ (x \neq \text{null}) \{
\begin{align*}
z^0 &= y^0; \\
y^0 &= x^1; \quad //\text{using (subE)} \\
x^1 &= x.tl^1; \\
y.tl^0 &= z^0;
\end{align*}
\}

Consequently, \text{reverse} \in \text{STR}.
Motivations NI-based techniques A general NI criterion Shape-analysis-based instance Conclusion

Illustrating example

\[ \Gamma(x) = 1, \Gamma(y) = \Gamma(z) = 0. \]

\[ \text{reverse} \in \text{NI}_\Gamma \]

\[ y = \text{null}; \]
\[ \text{while} \ (x \neq \text{null}) \{ \]
\[ z^0 = y^0; \]
\[ y^0 = x^1; \quad /\text{//using (subE)} \]
\[ x^1 = x.\text{tl}^1; \]
\[ y.\text{tl}^0 = z^0; \]
\[ \} \]

Consequently, \( \text{reverse} \in \text{STR} \).
Γ(x) = 1, Γ(y) = Γ(z) = 0.

**reverse ∈ NI**

\[
\begin{align*}
y &= \text{null}; \\
\text{while } (x \neq \text{null})^1 \{ \\
& \quad z^0 = y^0; \\
& \quad y^0 = x^1; \quad \text{//using (subE)} \\
& \quad x^1 = x.tl^1; \\
& \quad y.tl^0 = z^0;
\}
\end{align*}
\]

Consequently, reverse ∈ STR.
Illustrating example

\[ \Gamma(x) = 1, \Gamma(y) = \Gamma(z) = 0. \]

\[
\begin{align*}
\text{reverse} & \in \text{NI}_\Gamma \\
y & = \text{null} \\
\text{while} \ (x \neq \text{null}) \{ \\
& z^0 = y^0; \\
& y^0 = x^1; \quad \text{//using (subE)} \\
& x^1 = x.tl^1; \\
& y.tl^0 = z^0;
\}
\end{align*}
\]

Consequently, reverse \(\in\) STR.
Characterization of Polytime

Theorem [Soundness & Completeness]

\[ \text{STR} \cap \text{SN} = \text{FP}. \]

Theorem [A proper generalization]

\[ \text{SAFE} \cap \text{SN} \subsetneq \text{STR} \cap \text{SN}. \]

We capture reverse but also many algorithmic patterns, e.g.,

- on inductive data,
- algorithms with destructive updating,
- in-place algorithms.
In the arithmetical hierarchy

Theorem [Arithmetical hierarchy]

STR is $\Pi^1_0$-complete.

$\rightarrow$ using a reduction of the blank tape non-halting problem [Endrullis et al. 2011].

Reminder on Hájek Theorem [Hájek, TCS 79]

Providing an intensionally-complete characterization of FP is a $\Sigma^2_0$-complete problem.

Comparison with Hájek’s Theorem:

- STR is incomplete (there are false negative): 😞, but expected
- STR is one level below Hájek in the arithmetical hierarchy: 😊
- STR (undecidable) vs SAFE (tractable): 😊 and 😞
We abstract graphs using standard Shape-Analysis techniques:

```
\begin{align*}
\text{List} & \rightarrow \text{List} \rightarrow \text{List} \rightarrow \text{List} \\
\end{align*}
```

On difficulty to face for complexity analysis is that we quantify over each input:

- we use a separability hypothesis on inputs.
- The abstract graphs preserve stratification.
A new type system: SA

\[
\begin{align*}
\Gamma & \vdash^m_{SA} e : n & \Gamma & \vdash^n_{SA} st : n & 1 \leq n \\
\Gamma & \vdash_{SA} \text{while}(e)\{st\} : n
\end{align*}
\]

\[
\forall i, \quad \Gamma \vdash^m_{SA} e_i : n \quad n < m \\
\Gamma \vdash_{SA} \text{new } C(e) : n
\]

\[
\Gamma(x.a) = n \quad \Gamma \vdash^m_{SA} e : n \quad n < m \quad x \in n_{\ell, \Gamma} \\
\Gamma \vdash_{SA} \ell : x.a = e; : n
\]

where \( x \in n_{\ell, \Gamma} \) iff \( x \) only points to abstract nodes of level smaller than \( n \) in the ASG of \( \ell \).

**Theorem [Soundness & Completeness]**

\[
\text{SAFE} \cap \text{SN} \subsetneq \text{SA} \cap \text{SN} \subsetneq \text{STR} \cap \text{SN}
\]

**Theorem [Type inference]**

Deciding whether \( P \in \text{SA} \) can be done in time \( 2^{O(|P|)} \).
Illustrating example

\[ \Gamma(x) = 1, \Gamma(y) = \Gamma(z) = 0. \]

\[
\text{reverse} \in \text{NI}_\Gamma
\]

\[
y = \text{null}; \quad \text{while} \ (x \neq \text{null}) \quad z = y; \\
y^0 = x^1; \quad \text{//allowed} \\
x = x.tl; \\
\ell : y.tl^0 = z^0;
\]

\[
\rightarrow y \in 0_{\ell, \Gamma}
\]

\[
\Gamma(y.tl) = 0 \quad \Gamma \vdash^1_{SA} z : 0 \quad 0 < 1 \quad y \in 0_{\ell, \Gamma}
\]

\[
\Gamma \vdash^1_{SA} \ell : y.tl = z; : 0
\]

\[
\rightarrow \text{reverse} \in \text{SA}
\]
Conclusion

A summary

We have designed a new NI-based technique for FP:

- separating clearly NI and Complexity requirements,
- generalizing previous NI-based techniques (SAFE),
- $\Sigma^0_0$-complete
- and with decidable instances based on SA