A general noninterference policy for polynomial time

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

POPL23

January 19th, 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 $\mathcal{C}$:

$\mathcal{R} \subseteq \mathcal{L}$ is an ICC criterion if $\{\llbracket p \rrbracket \mid p \in \mathcal{R}\} = \mathcal{C}$.

**Examples of complexity class $\mathcal{C}$**

- $\mathcal{P}$, $\mathcal{FP}$,
- $\mathcal{PSPACE}$, $\mathcal{FPSPACE}$,
- $\mathcal{EXP}$, $2\text{-EXP}$, $\ldots$, $\text{ELEMENTARY}$,
- $\mathcal{NP}$,
- $\mathcal{NC}^0$, $\mathcal{NC}^1$, $\ldots$, $\mathcal{NC}$

**Examples of programming language $\mathcal{L}$**

- lambda-calculi, process calculi, $\ldots$
- imperative and OO programs,
- probabilistic and quantum programs.

**Examples of techniques**

- types, interpretations, $\ldots$
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: NI + 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 for safety

\( \tau \in \{0, 1\} \), with \( 0 \leq 1 \).

\[
\begin{align*}
\Gamma(x) &= \tau & \Gamma \vdash e : \tau & \Gamma \vdash e - 1 : \tau & \Gamma \vdash e + 1 : 0 \\
\Gamma \vdash x : \tau & \Gamma \vdash e : \tau' & \tau \leq \tau' & \Gamma \vdash x := e : \tau \\
\Gamma \vdash e : \tau & \Gamma \vdash st_1 : \tau & \Gamma \vdash st_2 : \tau & \Gamma \vdash st_1 \; st_2 : \tau \\
\Gamma \vdash if(e)\{st_1\} else\{st_2\} : \tau & \Gamma \vdash while(e)\{st\} : 1
\end{align*}
\]

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

Type inference is tractable.
Assume that $\text{add} :: 1 \times 0 \rightarrow 0$

\[ \text{mult}(\text{int } x, \text{int } y) \]

\[
\begin{array}{l}
\text{int } z=0; \\
\textbf{while} \ (x>0) \{
  x^1 = x-1; \\
  z^0 = \text{add}(y^1, z^0) \\
\}
\text{return } z;
\end{array}
\]

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

\[ \text{exp}(\text{int } x) \]

\[
\begin{array}{l}
\text{int } y=1; \\
\textbf{while} \ (x>0) \{
  x^1 = x-1; \\
  y^0 = \text{add}(y^1, y^0) \\
\}
\text{return } y;
\end{array}
\]

cannot be typed...
Adaptation to OO: not like taking candy off a baby (1/2)

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

▶ primitive data (pass-by-value):

\[ y^1 \]
\[ \downarrow \]
\[ 42 \]
\[ x^0 \]
\[ \rightarrow_0 \]
\[ 25 \]
\[ y^1 \]
\[ \downarrow \]
\[ 42 \]
\[ x^0 \]
\[ \rightarrow_0 \]
\[ 42 \]

▶ reference data (pass-by-reference):

\[ y^1 \]
\[ \downarrow \]
\[ C \]
\[ \cdots \]
\[ C \]
\[ x^0 \]
\[ \rightarrow_0 \]
\[ C \]
\[ \cdots \]
\[ C \]
Adaptation to OO: not like taking candy off a baby (2/2)

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: \((x^1.clone())^0\).
A simple counterexample

reverse on List \{\text{int} \text{ hd}; \text{List} \text{ tl}\}

\text{y} = \text{null};
\text{while} (x \neq \text{null}) \{ 
  \text{z} = \text{y};
  \text{y}^0 = x^1; \text{//Prohibited: needs explicit cloning}
  \text{x} = \text{x}.tl;
  \text{y}.tl^0 = \text{z};
\}

\text{Theorem [Hájek, TCS 79]}

Providing an intensionally-complete characterization of FP is a \Sigma_2^0\text{-complete problem.}

\rightarrow \text{But there might be some happy medium...}
**Stratification**

- \( \text{Conf} \triangleq \{(\text{st}, H)\} \in \text{Statements} \times \text{MemoryGraphs} \)
- \( \rightarrow_n \in \text{Conf} \rightarrow \text{Conf} \).
- with \( n \) the minimal level of a while loop guard encompassing the executed statement.
- \( \subseteq_n \) = subgraph relation on nodes of level \( \geq n \).
- \( R(P) \subseteq \text{Conf} \), the reachable configurations of \( P \).

**Definition**

A program \( P \in NI \) is *stratified* if for any \( (\text{st}, H) \in R(P) \),

\[
(\text{st}, H) \rightarrow_{n>0} (\text{st}', H') \text{ implies } H' \subseteq_n H.
\]

Let \( \text{STR} \) be the set of stratified programs.
Illustrating example

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

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

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

Consequently, \( \text{reverse} \in \text{STR} \).
Illustrating example

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

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

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

Consequently, \(\text{reverse} \in \text{STR}.\)
Illustrating example

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

\[ reverse \in NI \]

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

Consequently, \( reverse \in \text{STR} \).
Illustrating example

Γ(x) = 1, Γ(y) = Γ(z) = 0.

reverse ∈ NIΓ

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

Consequently, reverse ∈ 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 $\text{FP}$ 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 😞
A decidable instance based on shape-analysis

We abstract graphs using standard Shape-Analysis techniques:

\[ y^0 \xrightarrow{tl} \text{List} \xrightarrow{tl} \text{List} \xrightarrow{tl} \text{List} \xrightarrow{tl} \text{List} \xrightarrow{z^0} \]

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: $\text{SA}$

\[
\Gamma \vdash^m_{\text{SA}} e : n \quad \Gamma \vdash^n_{\text{SA}} \text{st} : n \quad 1 \leq n \\
\Gamma \vdash^m_{\text{SA}} \text{while}(e)\{\text{st}\} : n
\]

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

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

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 $2^{O(|P|)}$. 
Motivations  |  NI-based techniques  |  A general NI criterion  |  Shape-analysis-based instance  |  Conclusion
---|---|---|---|---
Illustrating example

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

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

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

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

\[ \begin{array}{c}
\Gamma(y.\text{tl}) = 0 \\
\Gamma \vdash_{\text{SA}}^1 z : 0 \\
0 < 1 \\
y \in 0_{\ell,\Gamma}
\end{array} \]

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

\[ \rightarrow \text{reverse} \in \text{SA} \]
We have designed a new NI-based technique for FP:

- separating clearly NI and Complexity requirements,
- generalizing previous NI-based techniques (SAFE),
- $\Pi_0^1$-complete
- and with decidable instances based on SA
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]