A general noninterference policy for polynomial time

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

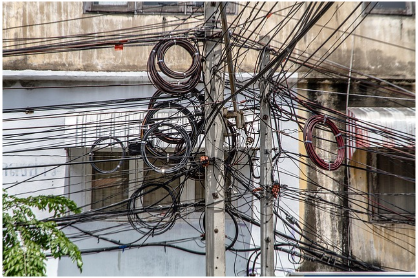
POPL23

January 19th, 2023

Program complexity analysis

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 $\mathcal{C}$:

$$\mathcal{R} \subseteq \mathcal{L} \text{ is an } \textbf{ICC criterion} \text{ if } \{[\![\mathrm{p}]\!] \mid \mathrm{p} \in \mathcal{R}\} = \mathcal{C}.$$

Examples of complexity class $\mathcal{C}$

- `P`, `FP`,
- `PSPACE`, `FPSPACE`,
- `EXP`, `2-EXP`, `...`, `ELEMENTARY`,
- `NP`,
- $\mathtt{NC}^0$, $\mathtt{NC}^1$, `...`, `NC`
- `PP`, `BPP`, `EQP`, `BQP`, `...`

Examples of programming language $\mathcal{L}$

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

Examples of techniques

- types, interpretations, ...

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 $\forall M,\ \forall N$,

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

- 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 $\forall M,\ \forall N$,

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

- 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]

$[\![\mathrm{SAFE} \cap \mathrm{SN}]\!] = \mathrm{FP}$.

Type system for safety

$\tau \in \{0, 1\}$, with $0 \leq 1$.

$$\frac{\Gamma(\mathtt{x}) = \tau}{\Gamma \vdash \mathtt{x} : \tau} \qquad \frac{\Gamma \vdash e : \tau}{\Gamma \vdash e - 1 : \tau} \qquad \frac{\Gamma \vdash e : \tau}{\Gamma \vdash e + 1 : 0}$$

$$\frac{\Gamma \vdash \mathtt{x} : \tau \qquad \Gamma \vdash \mathtt{e} : \tau' \qquad \tau \leq \tau'}{\Gamma \vdash \mathtt{x}{:=}\mathtt{e} : \tau} \qquad \frac{\Gamma \vdash \mathtt{st}_1 : \tau \qquad \Gamma \vdash \mathtt{st}_2 : \tau}{\Gamma \vdash \mathtt{st}_1\ \mathtt{st}_2 : \tau}$$

$$\frac{\Gamma \vdash \mathtt{e} : \tau \qquad \Gamma \vdash \mathtt{st}_1 : \tau \qquad \Gamma \vdash \mathtt{st}_2 : \tau}{\Gamma \vdash \mathtt{if(e)\{st_1\}else\{st_2\}} : \tau} \qquad \frac{\Gamma \vdash \mathtt{e} : 1 \qquad \Gamma \vdash \mathtt{st} : \tau}{\Gamma \vdash \mathtt{while(e)\{st\}} : 1}$$

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

Type inference is tractable.

Illustrating toy examples

Assume that add :: $1 \times 0 \to 0$

mult(int x, int y)

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

can be typed as mult :: $1 \times 1 \to 0$

exp(int x)

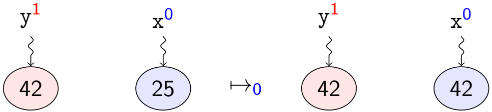
```
int y=1;
while (x>0)^1 {
  x^1 = x-1^1 ;
  y^0 = add(y^1 , y^? )^0 ;
}
return y;
```

cannot be typed...

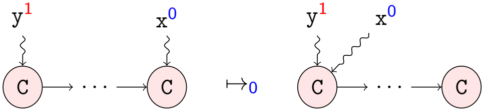
Adaptation to OO: not like taking candy off a baby (1/2)

$$x^{0} := y^{1}$$

- primitive data (pass-by-value):



- reference data (pass-by-reference):



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: $(\texttt{x}^1\texttt{.clone()})^0$.

A simple counterexample

reverse on List {int hd; List tl}

```
y = null;
while (x ≠ null)¹{
  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 Σ_0^2-complete problem.

→ But there might be some happy medium...

Stratification

- $\mathtt{Conf} \triangleq \{(\mathtt{st}, H)\} \in \mathit{Statements} \times \mathit{MemoryGraphs}$
- $\mapsto_{\mathsf{n}} \in \ \mathtt{Conf} \to \mathtt{Conf}.$
- with n the minimal level of a while loop guard encompassing the executed statement.
- $\subseteq_{\mathsf{n}}$ = subgraph relation on nodes of level $\geq$ n.
- $R(P) \subseteq \mathtt{Conf}$, the reachable configurations of P.

Definition

A program $P \in NI_{\Gamma}$ is *stratified* if for any $(\mathtt{st}, H) \in R(P)$,

$$(\mathtt{st}, H) \mapsto_{\mathsf{n}>0} (\mathtt{st}', H') \text{ implies } H' \subseteq_{\mathsf{n}} H.$$

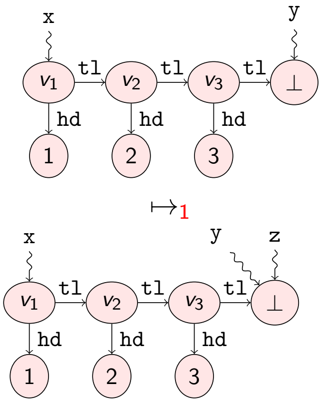
Let STR be the set of stratified programs.

Illustrating example

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

reverse $\in NI_\Gamma$

```
y = null;
while (x ≠ null)^1 {
   z^0 = y^0          ;
   y^0 = x^1; //using (subE)
   x^1 = x.tl^1;
   y.tl^0 = z^0;
}
```



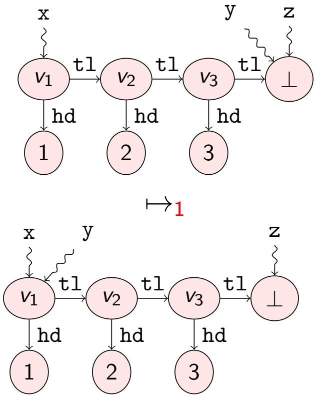
Consequently, reverse $\in$ STR.

Illustrating example

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

reverse $\in \mathrm{NI}_\Gamma$

$$
\begin{array}{l}
\texttt{y = null;}\\
\texttt{while (x} \neq \texttt{null)}^{1}\{\\
\quad \texttt{z}^{0} \texttt{ = y}^{0}\texttt{;}\\
\quad \texttt{y}^{0} \texttt{ = x}^{1}\texttt{; //using (subE)}\\
\quad \texttt{x}^{1} \texttt{ = x.tl}^{1}\texttt{;}\\
\quad \texttt{y.tl}^{0} \texttt{ = z}^{0}\texttt{;}\\
\}
\end{array}
$$



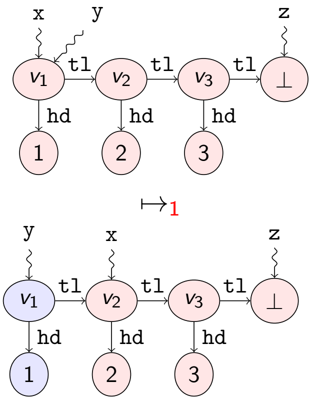
Consequently, reverse $\in \mathrm{STR}$.

Illustrating example

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

reverse $\in \mathrm{NI}_\Gamma$

```
y = null;
while (x ≠ null)¹{
  z⁰ = y⁰;
  y⁰ = x¹; //using (subE)
  x¹ = x.tl¹;
  y.tl⁰ = z⁰;
}
```



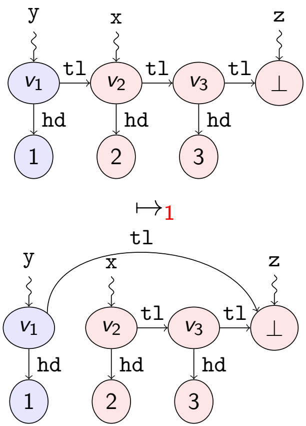
Consequently, reverse $\in \mathrm{STR}$.

Illustrating example

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

reverse $\in \mathrm{NI}_\Gamma$

```
y = null;
while (x ≠ null)^1{
  z^0 = y^0;
  y^0 = x^1; //using (subE)
  x^1 = x.tl^1;
  y.tl^0 = z^0;
}
```



Consequently, reverse $\in \mathrm{STR}$.

Characterization of Polytime

Theorem [Soundness & Completeness]

$[\![\mathrm{STR} \cap \mathrm{SN}]\!] = \mathbf{FP}.$

Theorem [A proper generalization]

$\mathrm{SAFE} \cap \mathrm{SN} \subsetneq \mathrm{STR} \cap \mathrm{SN}.$

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 Π^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]

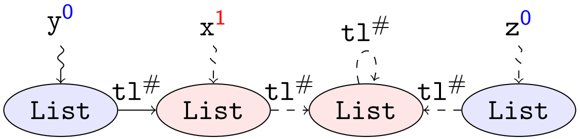
Providing an intensionally-complete characterization of $\mathtt{FP}$ is a Σ^2_0-complete problem.

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:



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

$$\frac{\Gamma \vdash^{\mathsf{m}}_{\mathrm{SA}} \mathsf{e} : \mathsf{n} \qquad \Gamma \vdash^{\mathsf{n}}_{\mathrm{SA}} \mathtt{st} : \mathsf{n} \qquad {\color{red}1} \leq \mathsf{n}}{\Gamma \vdash^{\mathsf{m}}_{\mathrm{SA}} \mathtt{while(e)\{st\}} : \mathsf{n}}$$

$$\frac{\forall i,\ \Gamma \vdash^{\mathsf{m}}_{\mathrm{SA}} \mathsf{e}_i : \mathsf{n} \qquad \mathsf{n} < \mathsf{m}}{\Gamma \vdash^{\mathsf{m}}_{\mathrm{SA}} \mathtt{new}\ \mathsf{C}(\overline{\mathsf{e}}) : \mathsf{n}} \qquad \frac{\Gamma(\mathtt{x}.\mathsf{a}) = \mathsf{n} \qquad \Gamma \vdash^{\mathsf{m}}_{\mathrm{SA}} \mathsf{e} : \mathsf{n} \qquad \mathsf{n} < \mathsf{m} \qquad \mathtt{x} \in \mathsf{n}_{\ell,\Gamma}}{\Gamma \vdash^{\mathsf{m}}_{\mathrm{SA}} \ell : \mathtt{x}.\mathsf{a} = \mathsf{e}; : \mathsf{n}}$$

where $\mathtt{x} \in \mathsf{n}_{\ell,\Gamma}$ iff $\mathtt{x}$ only points to abstract nodes of level smaller than n in the ASG of ℓ.

Theorem [Soundness & Completeness]

$\mathrm{SAFE} \cap \mathrm{SN} \subsetneq \mathrm{SA} \cap \mathrm{SN} \subsetneq \mathrm{STR} \cap \mathrm{SN}$

Theorem [Type inference]

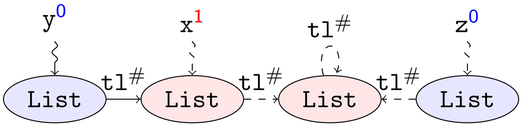
Deciding whether $P \in \mathrm{SA}$ can be done in time $2^{O(|P|)}$.

Illustrating example

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

reverse $\in$ NI$_\Gamma$

```
y = null;
while (x ≠ null)^1
   z = y;
   y^0 = x^1; //allowed
   x = x.tl;
ℓ:y.tl^0 = z^0;
}
```



$\rightarrow y \in 0_{\ell,\Gamma}$

$$\frac{\Gamma(y.tl) = 0 \qquad \Gamma \vdash^1_{SA} z : 0 \qquad 0 < 1 \qquad y \in 0_{\ell,\Gamma}}{\Gamma \vdash^1_{SA} \ell : y.tl = z; : 0}$$

$\rightarrow$ reverse $\in \mathrm{SA}$

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),
- Π^1_0-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]