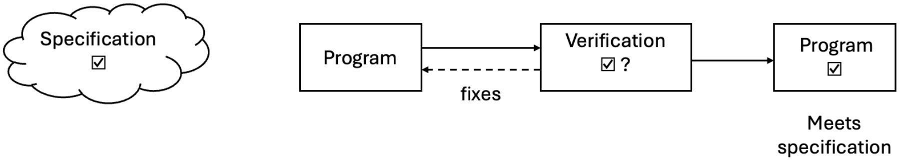
Implicit Computational Complexity: From Theory to Practice

Neea Rusch

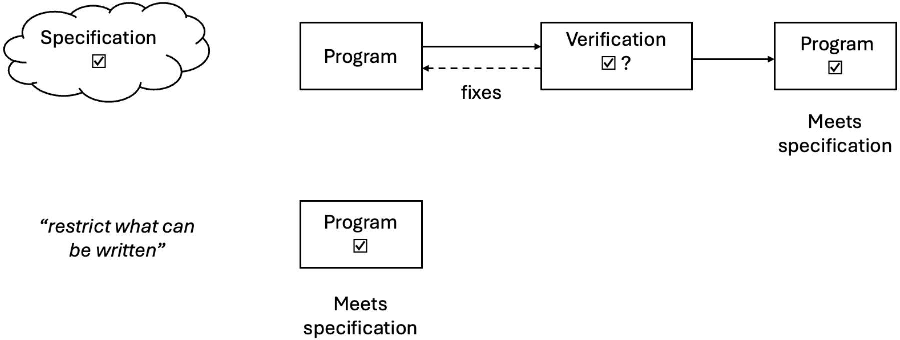
Augusta University, United States

Theory Seminar @ Aalto, 20 August 2024

Verification challenge



Verification challenge



What are the practical limitations of a non-turing complete language?

Asked 14 years ago Modified 4 months ago Viewed 12k times

74

As there are non-Turing complete languages out there, and given I didn't study Comp Sci at university, could someone explain something that a Turing-incomplete language (like Coq) cannot do?

Practical non-Turing-complete languages?

Asked 15 years, 8 months ago Modified 10 years, 2 months ago Viewed 22k times

55

Nearly all programming languages used are Turing Complete, and while this affords the language to represent any computable algorithm, it also comes with its own set of problems. Seeing as all the algorithms I write are intended to halt, I would like to be able to represent them in a language that guarantees they will halt.

---

https://stackoverflow.com/q/315340 and https://stackoverflow.com/q/3492188

What are the practical limitations of a non-turing complete language?

Asked 14 years ago Modified 4 months ago Viewed 12k times

74

As there are non-Turing complete languages out there, and given I didn't study Comp Sci at university, could someone explain something that a Turing-incomplete language (like Coq) cannot do?

Practical non-Turing-complete languages?

Asked 15 years, 8 months ago Modified 10 years, 2 months ago Viewed 22k times

55

Nearly all programming languages used are Turing Complete, and while this affords the language to represent any computable algorithm, it also comes with its own set of problems. Seeing as all the algorithms I write are intended to halt, I would like to be able to represent them in a language that guarantees they will halt.

Don't listen to the naysayers.
There are very good reasons . . . if you want to guarantee termination, or simplify code, for example by removing possible runtime errors.

`https://stackoverflow.com/q/315340` and `https://stackoverflow.com/q/3492188`

Languages with ~~restrictions~~ guarantees

(safe) Rust
no memory errors, no data races, controlled aliasing

Total functional programming
programs are provably terminating

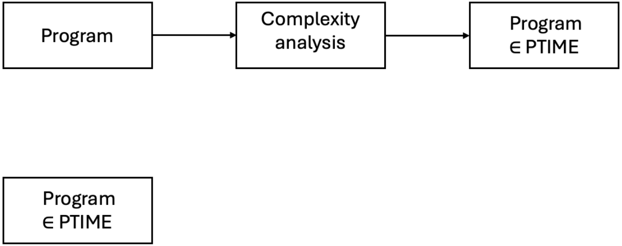
Theorem-proving languages
require termination, but enable constructing formal proofs

Synchronous languages
for real-time reactive systems with response-time and memory usage restrictions

Complexity analysis challenge

“write any program”

“restrict what can be written”



Implicit Computational Complexity (ICC)

Let L be a programming language, C a complexity class, and $[\![p]\!]$ the function computed by program p.

Find a restriction $R \subseteq L$, such that the following equality holds:

$$\{[\![p]\!] \mid p \in R\} = C$$

The variables L, C, and R are the parameters that vary greatly between different ICC systems[1].

[1] Romain Péchoux. *Complexité implicite : bilan et perspectives*. Habilitation à Diriger des Recherches (HDR). 2020. URL: `https://hal.univ-lorraine.fr/tel-02978986`.

How are Implicit Computational Complexity techniques useful in practice?

In short: applicable to domains *beyond* complexity theory, to track other semantic properties and to obtain information about runtime behavior.

TL;DR new analysis techniques to help developers write better software.

Implicit Computational Complexity: from theory to practice

- ☐ Automatic static complexity analysis
- ☐ Program transformations during compilation
- ☐ Ongoing and future explorations

mwp analysis[2]

(Theoretical) method for certifying that values computed by a deterministic imperative program will be bounded by polynomials in the program's inputs.

Can it be used for static program analysis?

[2] Neil D. Jones and Lars Kristiansen. "A flow calculus of *mwp*-bounds for complexity analysis". In: *ACM Trans. Comput. Log.* 10.4 (Aug. 2009), 28:1–28:41. DOI: 10.1145/1555746.1555752.

The goal is to discover a polynomially bounded data-flow relation between command `C`, initial values x_i, and final values x_i': $[\![\mathtt{C}]\!](x_i \rightsquigarrow x_i')$.

```
C′ ≡ X1 := X2 + X3;
     X1 := X1 + X1
```

$[\![\mathtt{C}']\!](x_1, x_2, x_3 \rightsquigarrow x_1', x_2', x_3')$

$x_1' \leq 2x_2 + 2x_3$

$x_2' \leq x_2$

$x_3' \leq x_3$

```
C″ ≡ X1 := 1;
     loop X2 {X1 := X1 + X1}
```

$[\![\mathtt{C}'']\!](x_1, x_2 \rightsquigarrow x_1', x_2')$

$x_1' \leq 2^{x_2}$

$x_2' \leq x_2$

Language

(var) $\mathtt{X}_1 \mid \mathtt{X}_2 \mid \mathtt{X}_3 \mid \dots$ (aexp) `e + e` | `e * e` (bexp) `e = e` | `e < e` | ...

(com) `skip` | `X := e` | `C;C` | `if b then C else C` | `loop X {C}` | `while b do {C}`

Inference rules

$$\frac{}{\vdash_{\text{JK}} \mathtt{Xi} : \{^{m}_{\mathtt{i}}\}}\ \text{E1} \qquad \frac{\vdash_{\text{JK}} \mathtt{Xi} : V_1 \quad \vdash_{\text{JK}} \mathtt{Xj} : V_2}{\vdash_{\text{JK}} \mathtt{Xi}\star\mathtt{Xj} : pV_1 \oplus V_2}\ \text{E3} \qquad \frac{\vdash \mathtt{e} : V}{\vdash \mathtt{Xj} \ \mathtt{=}\ \mathtt{e} : 1 \xleftarrow{\mathtt{j}} V}\ \text{A} \quad \dots$$

Dependencies (“flows”) $\xrightarrow{\text{weaker} \dots \text{stronger}}$

0 : no dependency m : maximal w : weak polynomial p : polynomial

mwp-bound $\max(\vec{x}, poly_1(\vec{y})) + poly_2(\vec{z})$

Analysis example

```
void main(int X1, int X2, int X3){
    if (X1 < X2) {
        X3 = X1 + X1;
    }
    else {
        X3 = X3 + X2;
    }
    while (X1 < 0){
        X1 = X2 + X3;
    }
}
```

	X1	X2	X3
X1	m	0	0
X2	0	m	0
X3	0	0	m

Analysis example

```
void main(int X1, int X2, int X3){
    if (X1 < X2) {
        X3 = X1 + X1;
    }
    else {
        X3 = X3 + X2;
    }
    while (X1 < 0){
        X1 = X2 + X3;
    }
}
```

	X1	X2	X3
X1	m	0	$\boxed{p}$
X2	0	m	0
X3	0	0	m

Analysis example

```
void main(int X1, int X2, int X3){
    if (X1 < X2) {
        X3 = X1 + X1;
    }
    else {
        X3 = X3 + X2;
    }
    while (X1 < 0){
        X1 = X2 + X3;
    }
}
```

	X1	X2	X3
X1	m	0	0
X2	0	m	p
X3	0	0	m

Analysis example

```
void main(int X1, int X2, int X3){
    if (X1 < X2) {
        X3 = X1 + X1;
    }
    else {
        X3 = X3 + X2;
    }
    while (X1 < 0){
        X1 = X2 + X3;
    }
}
```

	X1	X2	X3
X1	m	0	$\boxed{p}$
X2	0	m	$\boxed{p}$
X3	0	0	$\boxed{m}$

Analysis example

```
void main(int X1, int X2, int X3){
    if (X1 < X2) {
        X3 = X1 + X1;
    }
    else {
        X3 = X3 + X2;
    }
    while (X1 < 0){
        X1 = X2 + X3;
    }
}
```

	X1	X2	X3
X1	m	0	0
X2	$\boxed{w}$	m	0
X3	$\boxed{w}$	0	m

Analysis example

```
void main(int X1, int X2, int X3){
    if (X1 < X2) {
        X3 = X1 + X1;
    }
    else {
        X3 = X3 + X2;
    }
    while (X1 < 0){
        X1 = X2 + X3;
    }
}
```

$$\begin{array}{c|ccc} & \mathtt{X1} & \mathtt{X2} & \mathtt{X3} \\ \hline \mathtt{X1} & m & 0 & 0 \\ \mathtt{X2} & w & m & 0 \\ \mathtt{X3} & w & 0 & m \end{array}$$

$$= M^*$$

Side condition: $\forall i, M^*_{ii} = m$ and $\forall i, j, M^*_{ij} \neq p$

Analysis example

```
void main(int X1, int X2, int X3){
    if (X1 < X2) {
        X3 = X1 + X1;
    }
    else {
        X3 = X3 + X2;
    }
    while (X1 < 0){
        X1 = X2 + X3;
    }
}
```

$$
\begin{array}{c|ccc}
 & \mathtt{X1} & \mathtt{X2} & \mathtt{X3} \\
\hline
\mathtt{X1} & p & 0 & p \\
\mathtt{X2} & p & m & p \\
\mathtt{X3} & w & 0 & m
\end{array}
$$

$= \mathtt{C;C}$

Derivation success

$$\mathtt{C}' \equiv \mathtt{X1\ :=\ X2\ +\ X3;}$$
$$\mathtt{X1\ :=\ X1\ +\ X1}$$

$$\dfrac{\dfrac{\dfrac{}{\vdash_{\mathrm{JK}} \mathtt{X2} : \begin{pmatrix} 0 \\ m \\ 0 \end{pmatrix}}\ \text{E1} \qquad \dfrac{}{\vdash_{\mathrm{JK}} \mathtt{X3} : \begin{pmatrix} 0 \\ 0 \\ m \end{pmatrix}}\ \text{E1}}{\vdash_{\mathrm{JK}} \mathtt{X2+X3} : \begin{pmatrix} 0 \\ p \\ m \end{pmatrix}}\ \text{E3}}{\vdash_{\mathrm{JK}} \mathtt{X1:=X2+X3} : \begin{pmatrix} 0 & 0 & 0 \\ p & m & 0 \\ m & 0 & m \end{pmatrix}}\ \text{A}$$

$$\vdots$$

$$\dfrac{}{\vdash_{\mathrm{JK}} \mathtt{X1:=X1+X1} : \begin{pmatrix} p & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & m \end{pmatrix}}\ \text{A}$$

$$\vdots$$

$$\dfrac{}{\vdash_{\mathrm{JK}} \mathtt{X1:=X2+X3;X1:=X1+X1} : \begin{pmatrix} 0 & 0 & 0 \\ p & m & 0 \\ p & 0 & m \end{pmatrix}}\ \text{C}$$

$$x_1' \leq W_1(;;x_2, x_3) \ \wedge\ x_2' \leq W_2(x_2) \ \wedge\ x_3' \leq W_3(x_3)$$

Derivation failure

$$\texttt{C}'' \equiv \texttt{X1 := 1; loop X2 \{X1 := X1 + X1\}}$$

$$\dfrac{\dfrac{}{\vdash_{\text{JK}} \texttt{X1:=1} : \begin{pmatrix} m \\ 0 \end{pmatrix}}\text{E1} \qquad \vdots \qquad \dfrac{}{\vdash_{\text{JK}} \texttt{X1:=X1+X1} : \begin{pmatrix} p & 0 \\ 0 & m \end{pmatrix}}\text{A} \qquad \vdots}{\times}$$

$$\forall i, M^*_{ii} = m \quad \dfrac{\vdash_{\text{JK}} \texttt{C} : M}{\vdash_{\text{JK}} \texttt{loop X}_\ell\ \texttt{\{C\}} : M^* \oplus \{{}^p_\ell\!\rightarrow j \mid \exists i, M^*_{ij} = p\}}\ \text{L}$$

Nondeterminism

$$\frac{}{\vdash_{\mathrm{JK}} \texttt{Xi} : \{^{m}_{\texttt{i}}\}} \text{ E1}$$

$$\frac{}{\vdash_{\mathrm{JK}} \texttt{e} : \{^{w}_{\texttt{i}} \mid \texttt{Xi} \in \mathrm{var}(\texttt{e})\}} \text{ E2}$$

$$\frac{\vdash_{\mathrm{JK}} \texttt{Xi} : V_1 \quad \vdash_{\mathrm{JK}} \texttt{Xj} : V_2}{\vdash_{\mathrm{JK}} \texttt{Xi}\star\texttt{Xj} : pV_1 \oplus V_2} \text{ E3}$$

$$\frac{\vdash_{\mathrm{JK}} \texttt{Xi} : V_1 \quad \vdash_{\mathrm{JK}} \texttt{Xj} : V_2}{\vdash_{\mathrm{JK}} \texttt{Xi}\star\texttt{Xj} : V_1 \oplus pV_2} \text{ E4}$$

`X2 + X3` has 3 derivations:

by (E2) $\begin{pmatrix} 0 \\ w \\ w \end{pmatrix}$

by (E1) and (E3) $\begin{pmatrix} 0 \\ p \\ m \end{pmatrix}$

by (E1) and (E4) $\begin{pmatrix} 0 \\ m \\ p \end{pmatrix}$

In general n choices yields 3^n derivations.

Improvement

Idea: internalize the choices as functions from choices to coefficients.

If a coefficient depends on a choice, represent as 3 elements (think $\{0,1,2\}^n$)

If independent, represented as a single element.

We define basic functions $\delta(i,j)$ where i is a value, and j is index of the domain.
If j^{th} input is equal to i, then (i,j) is equal to the unit of the mwp semi-ring, else 0.

$$\star \in \{+,-\} \ \frac{}{\vdash \texttt{Xi}\star\texttt{Xj} : (0 \mapsto \{{}^{m}_{\texttt{i}}, {}^{p}_{\texttt{j}}\}) \oplus (1 \mapsto \{{}^{p}_{\texttt{i}}, {}^{m}_{\texttt{j}}\}) \oplus (2 \mapsto \{{}^{w}_{\texttt{i}}, {}^{w}_{\texttt{j}}\})} \ \mathsf{E}^{\mathrm{A}}$$

The failure problem

$\mathtt{C} \equiv$ `while(b){X1:=X2+X2}`

Derivation of `X1:=X2+X2` yields two matrices: $\begin{pmatrix} 0 & 0 \\ p & m \end{pmatrix}$ and $\begin{pmatrix} 0 & 0 \\ w & m \end{pmatrix}$

$$\forall i, M^*_{ii} = m \text{ and } \forall i,j, M^*_{ij} \neq p \; \frac{\vdash_{\text{JK}} \mathtt{C} : M}{\vdash_{\text{JK}} \texttt{while b do \{C\}} : M^*} \, \mathrm{W}$$

$\Rightarrow$ derivation $\begin{pmatrix} 0 & 0 \\ p & m \end{pmatrix}$ fails but derivation $\begin{pmatrix} 0 & 0 \\ w & m \end{pmatrix}$ succeeds.

Representing failure

Idea: We introduce ∞ flow to represent non-polynomial dependencies.

$$\{0, m, w, p, \infty\}$$

Every derivation can be completed without restarts.

Captures localized information about where failure occurs.

Once failure is introduced, it cannot be erased i.e., $\infty \times^{\infty} 0 = \infty$.

$\texttt{C} \equiv$ **`while`**`(b){X1:=X2+X2}`

$$\begin{pmatrix} m{+}\infty\delta(0{,}0){+}\infty\delta(1{,}0) & 0 \\ \infty\delta(0{,}0){+}\infty\delta(1{,}0){+}w\delta(2{,}0) & m \end{pmatrix}$$

Implementation: pymwp[3]

A prototype static analyzer for a subset of C99 programs.

Source code and demo: statycc.github.io/pymwp/demo

Install: **pip install pymwp**

Usage

```
pymwp /path/to/file.c [ARGS]
```

[3] Clément Aubert et al. "pymwp: A Static Analyzer Determining Polynomial Growth Bounds". In: *Automated Technology for Verification and Analysis*. Ed. by Étienne André and Jun Sun. Cham: Springer Nature Switzerland, 2023, pp. 263–275. ISBN: 978-3-031-45332-8.

Apart from ∞ coefficients, the original and adjusted mwp systems agree.

The latter provides a tractable technique: better proof-search strategy, fine-grained feedback, etc.

Can it be used for static program analysis? $\Rightarrow$ Yes, after adjustment.

mwp analysis improvement and implementation[4]

$$\text{mwp-analysis} \longrightarrow \text{mwp-analysis}' \xrightarrow{*} \text{static program analysis}$$

Main result
Lightweight, fast, practical data-size analysis focused on input value growth.

Key adjustments and enhancements
Adjusted mathematical framework: deterministic rules, internalized failure; concrete implementation.

Key insights
Learning how to communicate results to a different community.

[4] Clément Aubert et al. "mwp-Analysis Improvement and Implementation: Realizing Implicit Computational Complexity". In: *FSCD 2022*. Vol. 228. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 26:1–26:23. DOI: 10.4230/LIPIcs.FSCD.2022.26.

Implicit Computational Complexity: from theory to practice

- ✓ Automatic static complexity analysis
- ☐ Program transformations during compilation
- ☐ Ongoing and future explorations

Loop transformations

```
loop (0...n) {
    task_x
}
loop (0...n) {
    task_y
}
```

Fission or distribution

$\Leftrightarrow$

```
loop (0...n) {
    task_x
    task_y
}
```

Fusion or combination

$\Leftrightarrow$

```
loop (0...n/2) {
    task_x
    task_y
}
loop (n/2...n) {
    task_x
    task_y
}
```

Splitting

. . . and many more strategies.

Main idea

A loop optimization algorithm based on **loop fission** transformation, to introduce **parallelization potential** in previously uncovered cases.

Conceptually:

Distribute loops $\Rightarrow$ parallelize $\Rightarrow$ speedup in execution time

Technique overview

Input is a sequential imperative program.

1. Perform dependency analysis using data flow graphs (DFG).
2. Build a dependency graph.
3. Compute condensation graph and its covering.
4. Create loop for each statement in covering.
5. Parallelize distributed loops.

Variables in command $\texttt{C}$

We identify variables modified by (Out), used by (In), and occurring (Occ) in $\texttt{C}$.

For example, $\texttt{C} ::= \texttt{t[e}_1\texttt{]} = \texttt{e}_2$,

$$\begin{aligned}
\mathrm{Out}(\texttt{C}) &= \texttt{t} \\
\mathrm{In}(\texttt{C}) &= \mathrm{Occ}(\texttt{e}_1) \cup \mathrm{Occ}(\texttt{e}_2) \\
\mathrm{Occ}(\texttt{C}) &= \texttt{t} \cup \mathrm{Occ}(\texttt{e}_1) \cup \mathrm{Occ}(\texttt{e}_2)
\end{aligned}$$

We represent and analyze these dependencies using Data Flow Graphs (DFGs).

Data flow graph

- A DFG is a matrix over a fixed semi-ring.
- Represents a weighted relation on set of variables involved in command `C`.
- 3 types of dependencies:

∞	dependence	$\mathtt{x} \xrightarrow{\text{dependence}} \mathtt{x}$
1	propagation	$\mathtt{y} \overset{\text{propagation}}{\dashrightarrow} \mathtt{y}$
0	reinitialization	$\mathtt{z} \qquad\qquad \mathtt{z}$

Representing DFGs

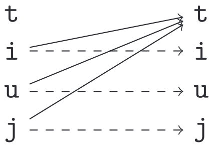
$$\texttt{C} ::= \texttt{t[i]} = \texttt{u} + \texttt{j}$$

$$\mathrm{Out}(\texttt{C}) = \{\texttt{t}\}$$
$$\mathrm{In}(\texttt{C}) = \{\texttt{i}, \texttt{u}, \texttt{j}\}$$
$$\mathrm{Occ}(\texttt{C}) = \{\texttt{t}, \texttt{i}, \texttt{u}, \texttt{j}\}$$

$\mathbb{M}(\texttt{C})$

$$\begin{array}{c c} & \begin{array}{cccc} \texttt{t} & \texttt{i} & \texttt{u} & \texttt{j} \end{array} \\ \begin{array}{c} \texttt{t} \\ \texttt{i} \\ \texttt{u} \\ \texttt{j} \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 \\ \infty & 1 & 0 & 0 \\ \infty & 0 & 1 & 0 \\ \infty & 0 & 0 & 1 \end{bmatrix} \end{array}$$

$\mathbb{M}(\texttt{C})$ as a graph



Correction

All body variables of conditional and loop statements depend on its control expression. We apply loop correction to account for this dependency.

For $\mathtt{e}$ an expression and $\mathtt{C}$ a command, $\mathrm{Corr}(\mathtt{e})_{\mathtt{C}}$, is $E^t \times O$.

- E^t – column vector with ∞ for variables in $\mathrm{Occ}(\mathtt{e})$ and 0 for other variables.
- O – row vector with ∞ for variables in $\mathrm{Out}(\mathtt{C})$ and 0 for other variables.

Algorithm

1. Pick a loop at top level.
2. Construct a *dependence graph*, which uses the DFG.
3. Compute its *condensation graph* from dependence graph.
4. Compute a *covering* of the condensation graph.
5. Create a loop per element of the covering.

Example

Identify In and Out variables

```
while (j < m) {
    x = r[i] * A[i][j];    // C1
    y = A[i][j] * p[j];    // C2
    s[j] = s[j] + x;       // C3
    q[i] = q[i] + y;       // C4
    j++;                   // C5
}
```

$$\mathrm{Out}(\mathtt{C}_1) = \{\mathtt{x}\}$$
$$\mathrm{In}(\mathtt{C}_1) = \{\mathtt{A}, \mathtt{i}, \mathtt{j}, \mathtt{r}\}$$
$$\vdots$$
$$\mathrm{Out}(\mathtt{C}_3) = \{\mathtt{s}\}$$
$$\mathrm{In}(\mathtt{C}_3) = \{\mathtt{s}, \mathtt{j}, \mathtt{x}\}$$
$$\vdots$$
$$\mathrm{Out}(\mathtt{C}_5) = \{\mathtt{j}\}$$
$$\mathrm{In}(\mathtt{C}_5) = \{\mathtt{j}\}$$

Example

Construct DFGs for each command

```
while (j < m) {
    x = r[i] * A[i][j];     // C1
    y = A[i][j] * p[j];     // C2
    s[j] = s[j] + x;        // C3
    q[i] = q[i] + y;        // C4
    j++;                    // C5
}
```

$$\mathbb{M}(C_1) = \begin{array}{c c} & \begin{array}{cccccccccc} i & j & m & x & y & A & r & s & p & q \end{array} \\ \begin{array}{c} i \\ j \\ m \\ x \\ y \\ A \\ r \\ s \\ p \\ q \end{array} & \begin{bmatrix} 1 & \cdot & \cdot & \infty & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \infty & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \infty & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \infty & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \end{bmatrix} \end{array}$$

Example

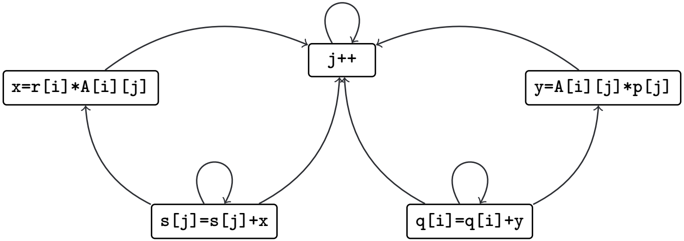
Compose DFGs of commands $\mathbb{M}(\mathtt{C}_1;\ldots;\mathtt{C}_\mathtt{n})$ and apply loop correction $E^t \times O$

$$\mathbb{M}(\mathtt{C}) = \begin{array}{cc} & \begin{array}{cccccccccc} \mathtt{i} & \mathtt{j} & \mathtt{m} & \mathtt{x} & \mathtt{y} & \mathtt{A} & \mathtt{r} & \mathtt{s} & \mathtt{p} & \mathtt{q} \end{array} \\ \begin{array}{c} \mathtt{i} \\ \mathtt{j} \\ \mathtt{m} \\ \mathtt{x} \\ \mathtt{y} \\ \mathtt{A} \\ \mathtt{r} \\ \mathtt{s} \\ \mathtt{p} \\ \mathtt{q} \end{array} & \begin{bmatrix} 1 & \cdot & \cdot & \infty & \infty & \cdot & \cdot & \cdot & \cdot & \infty \\ \cdot & \infty & \cdot & \infty & \infty & \cdot & \cdot & \infty & \cdot & \infty \\ \cdot & \infty & 1 & \infty & \infty & \cdot & \cdot & \infty & \cdot & \infty \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \infty & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \infty \\ \cdot & \cdot & \cdot & \infty & \infty & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \infty & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \infty & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \infty & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \infty \end{bmatrix} \end{array}$$

$\mathbb{M}(\mathtt{C}) = \mathbb{M}(\mathtt{C}_5) \times \cdots \times \mathbb{M}(\mathtt{C}_1) + \mathrm{Corr}(\mathtt{e})_\mathtt{C}$

Example

Construct a dependence graph. Vertices are the set of commands $\{\mathtt{C}_1; \cdots ; \mathtt{C}_\mathtt{n}\}$. Add directed edge from $\mathtt{C}_i$ to $\mathtt{C}_j$ iff $\exists\, x, y$, where $\mathbf{x} \in \mathrm{Out}(\mathtt{C}_j)$ and $\mathbf{y} \in \mathrm{In}(\mathtt{C}_i)$ and $\mathbb{M}(\mathtt{W})(\mathbf{y}, \mathbf{x}) = \infty$.



Example

Construct a condensation graph and proper saturated covering.

Example

Distribute loops and parallelize.

$$\tilde{\mathtt{W}} := \quad \mathbf{parallel} \left\{ \begin{array}{l} \texttt{while (j < m) \{} \\ \quad\quad \texttt{x = r[i] * A[i][j];} \\ \quad\quad \texttt{s[j] = s[j] + x;} \\ \quad\quad \texttt{j++;} \\ \texttt{\}} \end{array} \right\} \left\{ \begin{array}{l} \texttt{while (j < m) \{} \\ \quad\quad \texttt{y = A[i][j] * p[j];} \\ \quad\quad \texttt{q[i] = q[i] + y;} \\ \quad\quad \texttt{j++;} \\ \texttt{\}} \end{array} \right\}$$

Distributing and parallelizing non-canonical loops[5,6]

complexity analysis ⟶ command independence ⟶ program optimization

Main result

Automatable loop optimization for increasing parallelization potential.

Key insights

The internals of the analysis were easier to handle; adapted technique from one domain to another and to track a different semantic property; experimental evaluation was relevant.

---

[5] Clément Aubert et al. "Distributing and Parallelizing Non-canonical Loops". In: *Verification, Model Checking, and Abstract Interpretation*. Ed. by Cezara Dragoi, Michael Emmi, and Jingbo Wang. Vol. 13881. LNCS. Springer, 2023, pp. 1–24. DOI: 10.1007/978-3-031-24950-1_1.

[6] Clément Aubert et al. *Distributing and Parallelizing Non-canonical Loops – Artifact*. Version 1.0. Sept. 2022. DOI: 10.5281/zenodo.7080145. URL: https://github.com/statycc/loop-fission.

Implicit Computational Complexity: from theory to practice

- ✓ Automatic static complexity analysis
- ✓ Program transformations during compilation
- ☐ Ongoing and future explorations

Ongoing projects

Formally verified complexity analysis
Formalize the mwp-analysis using Coq proof assistant[7].

Noninterference analysis
The mathematical framework used in the loop transformation technique can be further adjusted to track secure data flow[8].

[7] Clément Aubert et al. "Certifying Complexity Analysis". At the Ninth International Workshop on Coq for Programming Languages. 2023.

[8] Clément Aubert and Neea Rusch. "An Information Flow Calculus for Non-Interference". At The 19th Workshop on Programming Languages and Analysis for Security. 2024.

Final remarks

Restricting programming languages is useful
Don't listen to the naysayers – we've seen many examples.

Communication is key when crossing domains
Impacts presentation style, evaluation strategy, and outcomes.

Want to collaborate or get in touch: `nrusch@augusta.edu`

Bibliography

Aubert, Clément, Thomas Rubiano, Neea Rusch, and Thomas Seiller. "Certifying Complexity Analysis". At the Ninth International Workshop on Coq for Programming Languages. 2023.

— ."Distributing and Parallelizing Non-canonical Loops". In: *Verification, Model Checking, and Abstract Interpretation*. Ed. by Cezara Dragoi, Michael Emmi, and Jingbo Wang. Vol. 13881. LNCS. Springer, 2023, pp. 1–24. DOI: 10.1007/978-3-031-24950-1_1.

— ."mwp-Analysis Improvement and Implementation: Realizing Implicit Computational Complexity". In: *FSCD 2022*. Vol. 228. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 26:1–26:23. DOI: 10.4230/LIPIcs.FSCD.2022.26.

— ."pymwp: A Static Analyzer Determining Polynomial Growth Bounds". In: *Automated Technology for Verification and Analysis*. Ed. by Étienne André and Jun Sun. Cham: Springer Nature Switzerland, 2023, pp. 263–275. ISBN: 978-3-031-45332-8.

Aubert, Clément and Neea Rusch. "An Information Flow Calculus for Non-Interference". At The 19th Workshop on Programming Languages and Analysis for Security. 2024.

Rusch, Neea. "Formally Verified Resource Bounds Through Implicit Computational Complexity". In: *Companion Proceedings of the 2022 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity*. SPLASH Companion 2022. Association for Computing Machinery, 2022. DOI: 10.1145/3563768.3565545.