Realizing Implicit Computational Complexity

Clément Aubert[1], Thomas Rubiano[2],
Neea Rusch[1], Thomas Seiller[2,3]

[1] Augusta University, USA
[2] LIPN - Laboratoire d'Informatique de Paris-Nord
[3] CNRS - Centre National de la Recherche Scientifique

TYPES 2022 20 June 2022

mwp-analysis

$\downarrow$

$$\begin{bmatrix} m & p & 0 \\ 0 & m & 0 \\ 0 & 0 & m \end{bmatrix}$$

1. **Input program**: simple, imperative
2. **mwp-Calculus**: inference rules
3. **Matrix**: assigned to commands by the inference rules
4. **Typed flows**: represent variable dependencies in matrix

Neil Jones and Lars Kristiansen. *"A Flow Calculus of mwp-Bounds for Complexity Analysis"*. ACM Trans. Comput. Logic (2009).

mwp-analysis

If derivation succeeds, guarantees that the values computed by an imperative program will be bounded by polynomials in the program's inputs.

Neil Jones and Lars Kristiansen. *"A Flow Calculus of mwp-Bounds for Complexity Analysis"*. ACM Trans. Comput. Logic (2009).

mwp-Analysis: Example

Program

```
loop X3 {
   X2 = X1 + X2;
}
```

$\rightarrow$

Analysis result

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

The many properties of *mwp*-analysis

- Multi-variate result
- Language-agnostic, expressive syntax
- Compositional method
- Termination and loop conditions have no impact
- Nondeterministic inference rules
- Derivability problem is NP-complete
- Pen & paper analysis

There were several open questions

- **Powerfulness** – what is the size of the class programs that can be analyzed?
- **Richness** – can it be extended to analyze more commands?
- **Practicality** – can it be used to analyze real-world programs?
- **Utility** – what else can be done with this analysis?

The extended and improved *mwp*-analysis

We defined an *extended* and *improved mwp*-analysis and created a practical implementation.

The extended and improved *mwp*-analysis – highlights

1. Improved by defining **deterministic inference rules**: analysis always completes and can internally handle failure.
2. Extended the syntax richness with support for **function calls**, including recursion.
3. Gained efficiency by **separating computation** into 2 phases: determining if a bound exists and computing its value.
4. Our **tool implementation,** `pymwp`, supports complexity analysis on a subset of C syntax.

Realizing Implicit Computation Complexity

We know ICC offers powerful analysis tools, that can be extended in richness, and made practical.

Realizing Implicit Computation Complexity

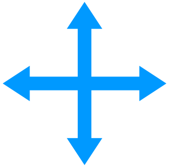
We know ICC offers powerful analysis tools, that can be extended in richness, and made practical.

. . . but wait, there is more!

Many other directions are being explored

performance optimization

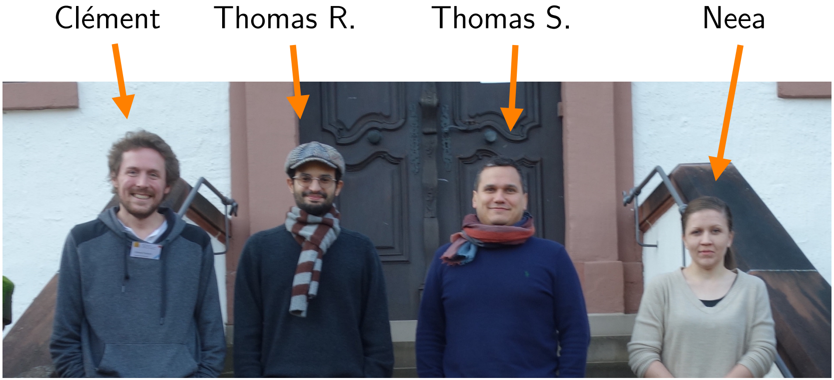
compiler integrations



formal verification

other kinds of ICC analyses

Come talk to us @ TYPES 2022!



Our source code – `pymwp` and more:

`https://github.com/statycc`