Formally Verified Resource Bounds through Implicit Computational Complexity

Neea Rusch
Augusta University, United States

SPLASH 2022 Doctoral Symposium
6 December 2022

Significance of Resource Bounds

- Constant-time programs
- Excessive time/space usage makes programs fail

Hypothesis

From Implicit Computational Complexity (ICC) we get new approaches to automatic program analysis and can resolve certain limitations.

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`.

mwp-Flow Analysis[2]

For an imperative program:
is the growth of input variable values polynomially bounded?

Will use the *mwp*-flow analysis to determine this.

[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 *mwp*-Calculus

- Track how variable depends on other variables.
- Flows characterize dependencies:

0	- no dependency	
m	- maximal	*weaker* $\uparrow$
w	- weak polynomial	
p	- polynomial	$\downarrow$ *stronger*

- Apply inference rules to program statements.
- Analysis result is collected in a matrix.

mwp-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

mwp-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

mwp-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	$\boxed{p}$
X3	0	0	m

mwp-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	m

mwp-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	w	m	0
X3	w	0	m

mwp-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;
    }
}
```

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

mwp-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;
    }
}
```

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

mwp-Analysis Example - Final Result

```
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	p	0	p
X2	p	m	p
X3	w	0	m

Analysis Soundness

For program $\texttt{C}$ and *mwp*-matrix M[3],

- Relation $\vdash \texttt{C} : M$ holds iff there exists a derivation in the calculus.
- $\vdash \texttt{C} : M$ means the calculus *assigns* the matrix M to the command $\texttt{C}$.
- Command $\texttt{C}$ is *derivable* if the calculus assigns at least one matrix to it.

Theorem (Soundness)

$\vdash \texttt{C} : M$ implies $\vDash \texttt{C} : M$.

[3] Jones and Kristiansen, "A flow calculus of *mwp*-bounds for complexity analysis", p. 11.

Proving Programs

- Prove that some property holds with the strongest possible guarantee.
- Done using an interactive theorem prover.
- Construct rigorous logical arguments.
- Machine-checkable for correctness.

Trade-off

Mechanical proofs require specifying every detail (slow, tedious).

$\Updownarrow$

Get the strongest possible guarantee of correctness.

My Goal

☐ Prove the *mwp* analysis technique.

- As defined in the original paper.
- Using the Coq proof assistant.

Steps - 1 of 4

Define the programming language under analysis.

- Simple, memory-less imperative language.
- Syntax: variables, arithmetic and boolean exp., commands.

Steps - 2 of 4

Define the mathematical machinery.

- Need e.g., (sparse) matrices, semi-ring.
- Other related mathematical concepts e.g., honest polynomial.

Steps - 3 of 4

Implementing the typing system.

- Define the flow calculus rules[4].
- Define a typing system.

[4] There is some non-determinism in these rules

Steps - 4 of 4

Prove the paper lemmas and theorems.

- There are 8 lemmas and 7 theorems.
- The soundness theorem, $\vdash \texttt{C} : M$ implies $\vDash \texttt{C} : M$, is essential.
- "These proofs are long, technical and occasionally highly nontrivial."[5]

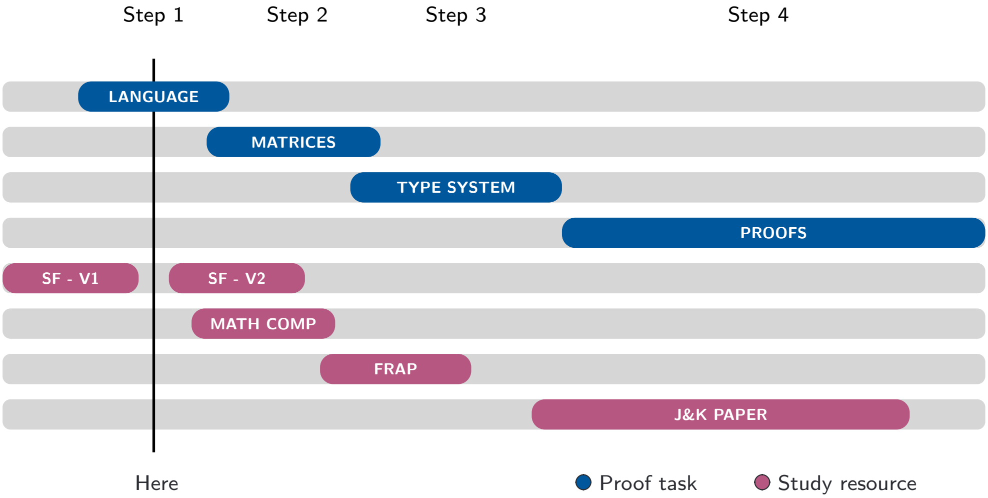
[5] Jones and Kristiansen, "A flow calculus of *mwp*-bounds for complexity analysis", p. 2.

Expected Main Result

A *certified* complexity analysis technique.

- Proves a positive result obtained by analysis is correct.
- Establishes certified "growth bound" on input variable values.

Timeline and Progress



Discussion

Many directions can follow from the correctness proof
e.g., a formally verified static analyzer.

- Our previous work: adjusting analysis makes it it practical and fast[6]
- Proof would show the original technique is correct, but not fast.
- It should be possible to combine those two results.

[6] 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.

mwp-Flow Analysis: Program Syntax

Variable	$\mathtt{X}_1 \mid \mathtt{X}_2 \mid \mathtt{X}_3 \mid \ldots$
Expression	`X` | `e + e` | `e * e`
Boolean Exp.	`e = e`, `e < e`, etc.
Commands	`skip` | `X := e` | `C;C` | `loop X {C}` | `if b then C else C` | `while b do {C}`

mwp-Flow Analysis: Rules

$$\frac{}{\vdash \mathsf{Xi} : \{ \overset{m}{\mathsf{i}} \}}\ \mathsf{E1} \qquad \frac{}{\vdash \mathsf{e} : \{ \overset{w}{\mathsf{i}} \mid \mathsf{Xi} \in \mathrm{var}(\mathsf{e}) \}}\ \mathsf{E2} \qquad \frac{\vdash \mathsf{e} : V}{\vdash \mathsf{Xj} = \mathsf{e} : 1 \overset{\mathsf{j}}{\leftarrow} V}\ \mathsf{A}$$

$$\star \in \{+, -\}\ \frac{\vdash \mathsf{Xi} : V_1 \quad \vdash \mathsf{Xj} : V_2}{\vdash \mathsf{Xi}\star\mathsf{Xj} : pV_1 \oplus V_2}\ \mathsf{E3} \qquad \star \in \{+, -\}\ \frac{\vdash \mathsf{Xi} : V_1 \quad \vdash \mathsf{Xj} : V_2}{\vdash \mathsf{Xi}\star\mathsf{Xj} : V_1 \oplus pV_2}\ \mathsf{E4}$$

$$\frac{\vdash \mathsf{C1} : M_1 \quad \vdash \mathsf{C2} : M_2}{\vdash \mathsf{C1};\ \mathsf{C2} : M_1 \otimes M_2}\ \mathsf{C} \qquad \frac{\vdash \mathsf{C1} : M_1 \quad \vdash \mathsf{C2} : M_2}{\vdash \textbf{if}\ \ \mathsf{b}\ \textbf{then}\ \mathsf{C1}\ \textbf{else}\ \ \mathsf{C2} : M_1 \oplus M_2}\ \mathsf{I}$$

$$\forall i, M^*_{ii} = m\ \ \frac{\vdash \mathsf{C} : M}{\vdash \textbf{loop}\ \ \mathsf{Xl}\ \ \{\mathsf{C}\} : M^* \oplus \{ \overset{p}{\mathsf{l}} \to j \mid \exists i, M^*_{ij} = p \}}\ \mathsf{L}$$

$$\forall i, M^*_{ii} = m \text{ and } \forall i, j, M^*_{ij} \neq p\ \ \frac{\vdash \mathsf{C} : M}{\vdash \textbf{while}\ \ \mathsf{b}\ \ \textbf{do}\ \{\mathsf{C}\} : M^*}\ \mathsf{W}$$