

Certifying Complexity Analysis

Clément Aubert¹, Thomas Rubiano²,
Neea Rusch¹, Thomas Seiller^{2,3}

¹ Augusta University

² LIPN - Laboratoire d'Informatique de Paris-Nord

³ CNRS - Centre National de la Recherche Scientifique

CoqPL 2023 21 January 2023

$$\llbracket C \rrbracket (x_1, \dots, x_n \rightsquigarrow x'_1, \dots, x'_n)$$

$$C' \equiv \begin{array}{l} X1 := X2 + X3; \\ X1 := X1 + X1 \end{array}$$

$$C' \equiv \begin{array}{l} X1 := X2 + X3; \\ X1 := X1 + X1 \end{array}$$

$\llbracket C' \rrbracket(x_1, x_2, x_3 \rightsquigarrow x'_1, x'_2, x'_3)$ implies $x'_1 \leq 2x_2 + 2x_3$ and $x'_2 \leq x_2$ and $x'_3 \leq x_3$.

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

```

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

```

$\llbracket C'' \rrbracket(x_1, x_2 \rightsquigarrow x'_1, x'_2)$ implies $x'_1 \leq 2^{x_2}$ and $x'_2 \leq x_2$.


We present a plan to formalize a complexity analysis *technique* that guarantees imperative program input variable values have polynomial growth bounds.

mwp-Flow Analysis¹

Tracks how each variable depends on other variables.

Flows characterize dependencies:

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



weaker

stronger

Apply inference rules to program statements.

Collect analysis result in a matrix.

¹Neil 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.

Initial state

```
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	<i>m</i>	0	0
X2	0	<i>m</i>	0
X3	0	0	<i>m</i>

```
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	<i>m</i>	0	<i>p</i>
X2	0	<i>m</i>	0
X3	0	0	<i>m</i>

```
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	<i>m</i>	0	0
X2	0	<i>m</i>	<i>p</i>
X3	0	0	<i>m</i>

```
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	<i>m</i>	0	<i>p</i>
X2	0	<i>m</i>	<i>p</i>
X3	0	0	<i>m</i>

```
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	<i>m</i>	0	0
X2	<i>w</i>	<i>m</i>	0
X3	<i>w</i>	0	<i>m</i>

```

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	<i>m</i>	0	0
X2	<i>w</i>	<i>m</i>	0
X3	<i>w</i>	0	<i>m</i>

$= M^*$

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

```

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	<i>p</i>	0	<i>p</i>
X2	<i>p</i>	<i>m</i>	<i>p</i>
X3	<i>w</i>	0	<i>m</i>

= C;C

$$\frac{}{\vdash e : \{\overset{w}{i} \mid X_i \in \text{var}(e)\}} \text{ E2}$$

$$\frac{\vdash X_i : V_1 \quad \vdash X_j : V_2}{\vdash X_i \star X_j : pV_1 \oplus V_2} \text{ E3}$$

$$\frac{\vdash X_i : V_1 \quad \vdash X_j : V_2}{\vdash X_i \star X_j : V_1 \oplus pV_2} \text{ E4}$$

Alternative choice at step 4

```
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	<i>m</i>	0	0
X2	<i>m</i>	<i>m</i>	0
X3	<i>p</i>	0	<i>m</i>

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

The result of the analysis is an “*mwp*-bound”.

An *mwp*-bound is a number-theoretic expression of form $\max(\vec{x}, \text{poly}_1(\vec{y})) + \text{poly}_2(\vec{z})$ where poly_1 and poly_2 are honest polynomials.

An *honest polynomial* is built up from constants in \mathbb{N} and variables by applying operators $+$ (addition) and \times (multiplication).

TODO: \square a mechanical proof of the *mwp*-analysis technique, as defined in the original paper, in Coq.

This requires defining and proving:

1. programming language under analysis,
2. mathematical framework (matrices, vectors, *mwp*-bound, ...),
3. typing system, and
4. the lemmas and proofs from the paper.

Programming Language

The language is Imp with added loop command.

Variable $X_1 \mid X_2 \mid X_3 \mid \dots$

Expression $X \mid e + e \mid e * e$

Boolean Exp. $e = e, e < e, \text{etc.}$

Commands $\text{skip} \mid X := e \mid C;C \mid \text{loop } X \{C\} \mid$
 $\text{if } b \text{ then } C \text{ else } C \mid \text{while } b \text{ do } \{C\}$



2



3

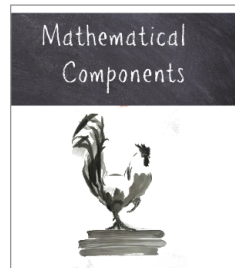
²Benjamin C. Pierce et al. “Logical Foundations”. In: *Software Foundations*. Ed. by Benjamin C. Pierce. Version 6.2. Vol. 1. 2022. URL: <http://softwarefoundations.cis.upenn.edu>.

³Benjamin C. Pierce et al. “Programming Language Foundations”. In: *Software Foundations*. Ed. by Benjamin C. Pierce. Version 6.2. Vol. 2. 2022. URL: <http://softwarefoundations.cis.upenn.edu>.

Mathematical Framework

Requires:

- Matrices, vectors, semi-ring
- Matrix operations: add, multiply, equivalence, fixpoint
- *mwp*-bound, honest polynomials



4

⁴Assia Mahboubi and Enrico Tassi. *Mathematical Components*. Zenodo, 2022. DOI: 10.5281/zenodo.7118596.

Typing System

- Connects the language and mathematical framework.
- Defined based on the inference rules.



Inference Rules

$$\frac{}{\vdash X_i : \{i^m\}} \text{ E1}$$

$$\frac{\vdash C1 : M_1 \quad \vdash C2 : M_2}{\vdash \text{if } b \text{ then } C1 \text{ else } C2 : M_1 \oplus M_2} \text{ I}$$

$$\frac{}{\vdash e : \{i^w \mid X_i \in \text{var}(e)\}} \text{ E2}$$

$$\frac{\vdash X_i : V_1 \quad \vdash X_j : V_2}{\vdash X_i \star X_j : pV_1 \oplus V_2} \text{ E3}$$

$$\frac{\vdash X_i : V_1 \quad \vdash X_j : V_2}{\vdash X_i \star X_j : V_1 \oplus pV_2} \text{ E4}$$

$$\frac{\vdash e : V}{\vdash X_j = e : 1 \stackrel{j}{\leftarrow} V} \text{ A}$$

$$\forall i, M_{ii}^* = m \quad \frac{\vdash C : M}{\vdash \text{loop } X_\ell \{C\} : M^* \oplus \{ \stackrel{p}{\ell} \rightarrow j \mid \exists i, M_{ij}^* = p \}} \text{ L}$$

$$\frac{\vdash C1 : M_1 \quad \vdash C2 : M_2}{\vdash C1; C2 : M_1 \otimes M_2} \text{ C}$$

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

Lemmas and Theorems

The soundness theorem is the main achievement of the paper.

Theorem: Soundness

$\vdash C : M$ implies $\models C : M$.

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

Multiple proofs are about the correctness of inference rules, e.g., the loop rules.

Theorem: 7.18

If $\models C : M$ and $M_{ii}^* = m$ for all i , then

$$\models \text{loop } x_{\ell}\{C\} : M^* \oplus \{\ell \rightarrow j \mid \exists i [M_{ij}^* = p]\}$$

Theorem: 7.19

If $\models C : M$ and $M_{ii}^* = m$ for all i , and $M_{ij}^* \neq p$ for all i, j , then
 $\models \text{while } b\{C\} : M^*.$

“These proofs are long, technical and occasionally highly nontrivial.”⁵

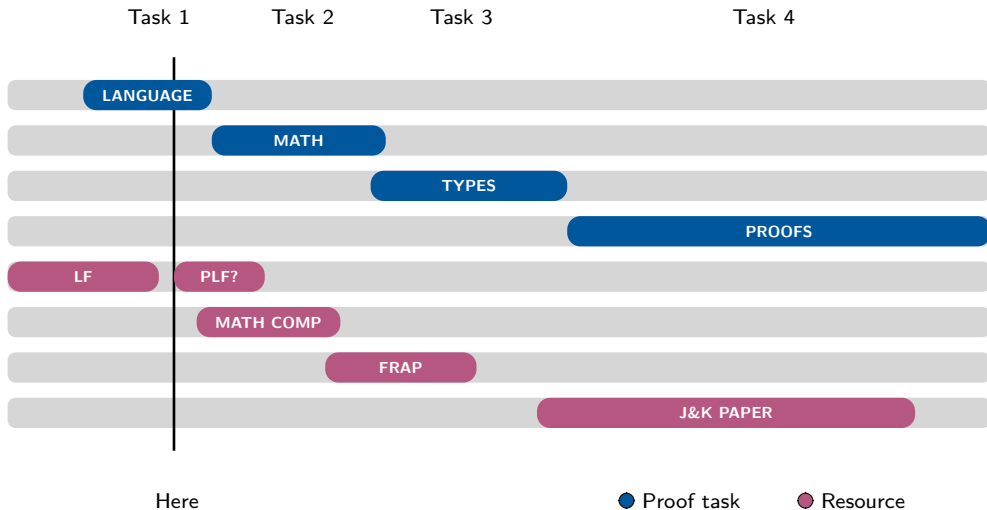
⁵Jones and Kristiansen, “A flow calculus of *mwp*-bounds for complexity analysis”, p. 2.

Main Contribution

A certified complexity analysis technique.

- Proof that the analysis technique is sound.
- Proof that a positive result obtained by analysis is correct.
- Enables obtaining a certified “growth bound” on input variable values.

Timeline and Progress



Discussion

Hot takes inspired by POPL'23 😊

A scientific contribution must be novel, useful, and challenging.

A scientific contribution must be novel, useful, and challenging.

“To me, Coq is just another programming language, and formalization requires high effort . . . Theoretically, the analysis is not very interesting, . . . , practically, it is not good for analysing realistic programs.”

A scientific contribution must be novel, useful, and challenging.

“[T]here’s some related recent work in Easycrypt since complexity is crucial for cryptography. . . It would be great if your approach could work for embedded imperative language, as it would provide functionality similar to easycrypt in Coq.”

A scientific contribution must be novel, useful, and challenging.

"Looks like a great proposal! . . . it should lead to interesting discussion."

Possibilities

- Our previous work adjusted analysis to makes it practical and fast⁶
- Proof shows the technique is correct, but not fast.
- It should be possible to combine those two results.

⁶Clément 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.

References

- Aubert, Clément, Thomas Rubiano, Neea Rusch, and Thomas Seiller. “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.
- Jones, Neil D. 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.
- Mahboubi, Assia and Enrico Tassi. *Mathematical Components*. Zenodo, 2022. DOI: 10.5281/zenodo.7118596.
- Pierce, Benjamin C., Arthur Azevedo de Amorim, Chris Casinghino, Marco Gaboardi, Michael Greenberg, Cătălin Hrițcu, Vilhelm Sjöberg, Andrew Tolmach, and Brent Yorgey. “Programming Language Foundations”. In: *Software Foundations*. Ed. by Benjamin C. Pierce. Version 6.2. Vol. 2. 2022. URL: <http://softwarefoundations.cis.upenn.edu>.
- Pierce, Benjamin C., Arthur Azevedo de Amorim, Chris Casinghino, Marco Gaboardi, Michael Greenberg, Cătălin Hrițcu, Vilhelm Sjöberg, and Brent Yorgey. “Logical Foundations”. In: *Software Foundations*. Ed. by Benjamin C. Pierce. Version 6.2. Vol. 1. 2022. URL: <http://softwarefoundations.cis.upenn.edu>.