Certifying Complexity Analysis

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

In This Presentation

We present a plan to formally verify a **complexity analysis technique**, mwp-flow analysis, in Coq.

- This is a technique from *Implicit Computational Complexity* (ICC).
- It guarantees program input variable values have polynomial growth bounds.

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`.

Analyzing Variable Value Growth

For a deterministic imperative program,
is the growth of input variable values polynomially bounded?

Example

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

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

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 }
```

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

implies $x_1' \leq 2^{x_2}$ and $x_2' \leq x_2$.

mwp-Flow Analysis[2]

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

0	– no dependency	
m	– maximal	*weaker*
w	– weak polynomial	$\updownarrow$
p	– polynomial	*stronger*

- Apply inference rules to program statements.
- Collect analysis result in a matrix.

[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.

mwp-Flow Analysis Inference Rules

$$\frac{}{\vdash \mathsf{Xi} : \{{}^{m}_{\mathsf{i}}\}}\ \mathsf{E1} \qquad \frac{\vdash \mathsf{C1} : M_1 \quad \vdash \mathsf{C2} : M_2}{\vdash \textbf{if}\ \ \mathsf{b}\ \textbf{then}\ \mathsf{C1}\ \textbf{else}\ \ \mathsf{C2} : M_1 \oplus M_2}\ \mathsf{I}$$

$$\frac{}{\vdash \mathsf{e} : \{{}^{w}_{\mathsf{i}} \mid \mathsf{Xi} \in \mathrm{var}(\mathsf{e})\}}\ \mathsf{E2} \qquad \frac{\vdash \mathsf{Xi} : V_1 \quad \vdash \mathsf{Xj} : V_2}{\vdash \mathsf{Xi} \star \mathsf{Xj} : pV_1 \oplus V_2}\ \mathsf{E3} \qquad \frac{\vdash \mathsf{Xi} : V_1 \quad \vdash \mathsf{Xj} : V_2}{\vdash \mathsf{Xi} \star \mathsf{Xj} : V_1 \oplus pV_2}\ \mathsf{E4}$$

$$\frac{\vdash \mathsf{e} : V}{\vdash \mathsf{Xj} = \mathsf{e} : 1 \overset{\mathsf{j}}{\leftarrow} V}\ \mathsf{A} \qquad \forall i, M^*_{ii} = m\ \ \frac{\vdash \mathsf{C} : M}{\vdash \textbf{loop}\ \ \mathsf{X}_\ell\{\mathsf{C}\} : M^* \oplus \{{}^{p}_{\ell} \to j \mid \exists i, M^*_{ij} = p\}}\ \mathsf{L}$$

$$\frac{\vdash \mathsf{C1} : M_1 \quad \vdash \mathsf{C2} : M_2}{\vdash \mathsf{C1};\ \mathsf{C2} : M_1 \otimes M_2}\ \mathsf{C} \qquad \forall i, M^*_{ii} = m \text{ and } \forall i, j, M^*_{ij} \neq p\ \ \frac{\vdash \mathsf{C} : M}{\vdash \textbf{while}\ \ \mathsf{b}\ \textbf{do}\ \{\mathsf{C}\} : M^*}\ \mathsf{W}$$

mwp-Analysis Example

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	m	0	0
X2	0	m	0
X3	0	0	m

mwp-Analysis Example

Step 1 of 6

```
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

Step 2 of 6

```
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

mwp-Analysis Example

Step 3 of 6

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

mwp-Analysis Example

Step 4 of 6

```
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

Step 5 of 6

```
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 $\forall i, j, M^*_{ij} \neq p$

mwp-Analysis Example

Step 6 of 6

```
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

The derivation can fail – 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	m	0	0
X2	m	m	0
X3	p	0	m

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

mwp-Bounds on Value Growth

The result of the analysis is captured as an "mwp-bound":

- An mwp-bound us a number-theoretic expression of form $\max(\vec{x}, \mathsf{poly}_1(\vec{y})) + \mathsf{poly}_2(\vec{z})$ where *poly*$_1$ and *poly*$_2$ are honest polynomials.
- An *honest polynomial* is a polynomial build up from constants in $\mathbb{N}$ and variables by applying operators $+$ (addition) and $\times$ (multiplication).

Analysis Proofs

The soundness theorem is the main achievement of the paper.

Theorem 1: Soundness

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

Analysis Proofs

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

Theorem 2: 7.18

If $\vDash \texttt{C} : M$ and $M^*_{ii} = m$ for all i, then

$$\vDash \texttt{loop X\_}\ell\{\texttt{C}\} : M^* \oplus \{^p_\ell \to j \mid \exists\, i\, [M^*_{ij} = p]\}$$

Theorem 3: 7.19

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

Analysis Proofs

The loop rules require this lemma.

Lemma 1: 7.17

Let $\texttt{C}^0 \equiv \texttt{skip}$ and $\texttt{C}^{t+1} \equiv \texttt{C}^t\texttt{;C}$. Assume that $\vDash \texttt{C} : M$ and that $M^*_{ii} = m$ for all i. Then for any $j \in \{1, \ldots, n\}$ there exists a fixed number k and honest polynomials p, q such that for any t we have

$$[\![C]\!](x_1, ..., x_n \rightsquigarrow x_1^{'}, ..., x_n^{'}) \Rightarrow x_j^{'} \leq \mathsf{max}(\vec{u}, q(\vec{y})) + (t+2)^k p(\vec{z}) \qquad (*_j)$$

where $\vec{u} = \{x_i \mid M^*_{ij} = m\}$ and $\vec{y} = \{x_i \mid M^*_{ij} = w\}$ and $\vec{z} = \{x_i \mid M^*_{ij} = p\}$. Moreover, neither the polynomial p nor the polynomial q depends on k or t; and if the list $\vec{z}$ is empty, then $p(\vec{z}) = 0$.

The Goal of This Work

`TODO:` ☐ a mechanical proof of the mwp-analysis technique, as defined in the original paper, in Coq.

This will require defining and proving:

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

Defining the Programming Language

Task 1 of 4

It is `Imp` language + `loop` command.

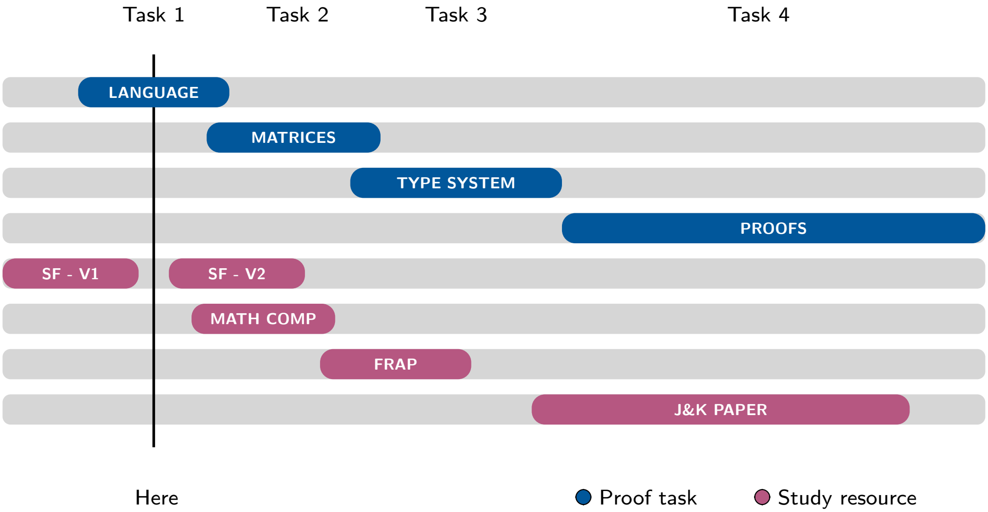
Variable	$\mathtt{X}_1 \mid \mathtt{X}_2 \mid \mathtt{X}_3 \mid \ldots$
Expression	`X` $\mid$ `e + e` $\mid$ `e * e`
Boolean Exp.	`e = e`, `e < e`, etc.
Commands	`skip` $\mid$ `X := e` $\mid$ `C;C` $\mid$ `loop X {C}` $\mid$ `if b then C else C` $\mid$ `while b do {C}`

Defining the Programming Language

Task 1 of 4

```
Inductive ceval : com → state → state → Prop :=
| E_Skip : forall st,
  st =[ skip ]⇒ st
| E_Asgn  : forall st a n x,
  aeval st a = n →
  st =[ x := a ]⇒ (x !→ n ; st)
| E_Seq : forall c1 c2 st st' st'',
  st  =[ c1 ]⇒ st'  →
  st' =[ c2 ]⇒ st'' →
  st  =[ c1 ; c2 ]⇒ st''
| E_IfTrue : forall st st' b c1 c2,
  beval st b = true →
  st =[ c1 ]⇒ st' →
  st =[ if b then c1 else c2 end]⇒ st'
| E_IfFalse : forall st st' b c1 c2,
  beval st b = false →
  st =[ c2 ]⇒ st' →
  st =[ if b then c1 else c2 end]⇒ st'
| E_WhileFalse : forall b st c,
  beval st b = false →
  st =[ while b do c end ]⇒ st
| E_WhileTrue : forall st st' st'' b c,
  beval st b = true →
  st  =[ c ]⇒ st' →
  st' =[ while b do c end ]⇒ st'' →
  st  =[ while b do c end ]⇒ st''
| E_LoopFalse : forall n st c,
  (n =? 0) = true →
  st =[ for n do c end ]⇒ st
| E_Loop : forall n st st' st'' c,
  (n =? 0) = false →
  st  =[ c ]⇒ st' →
  st' =[ while ((n-1) > 0) do c end ]⇒ st'' →
  st  =[ for n do c end ]⇒ st''
where "st =[ c ]⇒ st'" := (ceval c st st').
```

Timeline and Progress



Defining the Mathematical Framework

Task 2 of 4

Requires:

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

The plan is to use mathcomp[3] for most of these.

[3] `https://math-comp.github.io/mcb/`

Defining a Typing System

Task 3 of 4

- This connects the analyzed language with the mathematical framework.
- The system is defined based on the inference rules of the calculus.

The plan is to study and use the FRAP book[4] as a guide.

[4] `http://adam.chlipala.net/frap`

Prove Lemmas and Theorems from Paper

"These proofs are long, technical and occasionally highly nontrivial."[5]

- There are 8 lemmas and 7 theorems in total.
- The soundness theorem, $\vdash \texttt{C} : M$ implies $\vDash \texttt{C} : M$, is essential.
- Proving the lemma needed for loop rules is "hard".

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

Expected Main Result

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.

Conclusion

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

- Our previous work[6] showed adjusting analysis makes it practical and fast.
- 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.