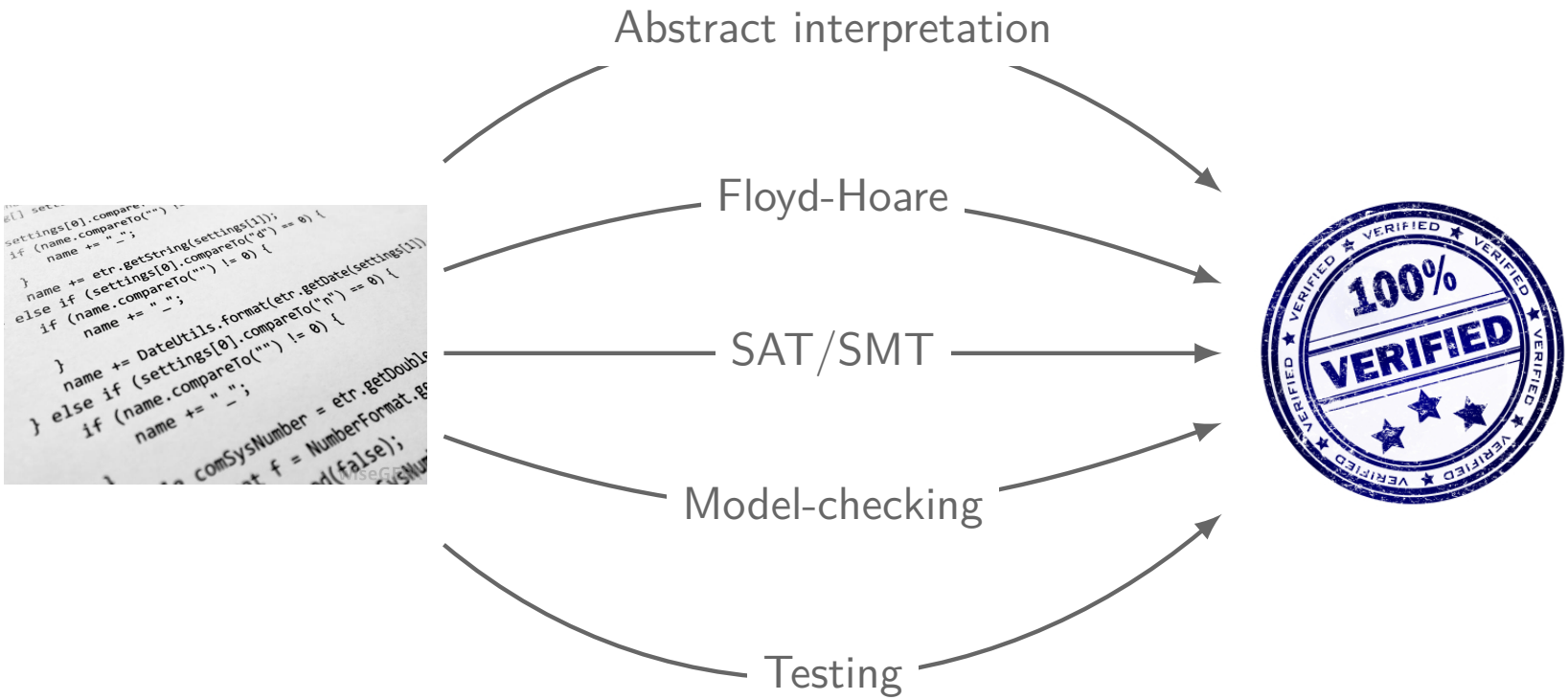
Polynomial invariants by linear algebra

| Steven de Oliveira, Saddek Bensalem, Virgile Prevosto

Invariants for automatic verification

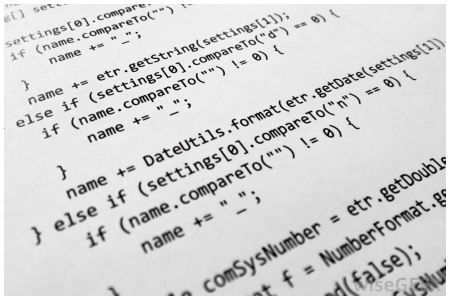
Background

Context : Automatic program verification



Background

Context : Automatic program verification



+ **loop**

Abstract interpretation

Floyd-Hoare

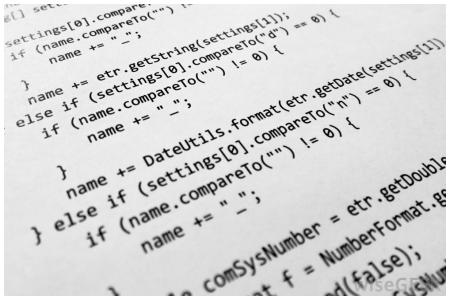
SAT/SMT

Model-checking

Testing

Background

Context : Automatic program verification



+ **loop** + **invariant**

Abstract interpretation

Floyd-Hoare

SAT/SMT

Model-checking

Testing

Polynomial assignments

We are interested in generating polynomial invariants of loops with **polynomial assignments and non-deteministic conditions**.

```
while(y<N){
x = x + y*y;
y = y + 1;
}
```

- Many SMT solvers are hard put to deal with polynomial assignments.
- Replacing the loop by the invariant $2y^3 - 3y^2 + y = 6x$ can be helpful.

Realistic example : optimized multiplication

```
int prod (int a,int b){
  int x = a,y = b;
  int z = 0;

  while( y!=0 ){
    if ( y % 2 ==1 ){
      z = z+x;
      y = y-1;
      }
    x = 2*x;
    y = y/2;
    }
  return z;
  }
```

The inductive invariant $z + x * y == a * b$ is **needed** to prove that this function multiplies a and b.

Overview

We present a **new invariant generation procedure** :

- that is sound and complete ;
- that relies on simple linear algebra algorithms ;
- implemented in PILAT which is faster than equivalent tools.

Invariants for automatic verification

Invariant generation

- Known as a hard problem
- Linearization
- Linear invariant generation

In a nutshell

Invariant generation

State of the art

Methods	Karr [Kar76]	Gröbner [Kov]	A. I. [CJJK14]	Us
Tool name	–	Aligator	Fastind	Pilat
Poly. loops	✗	✓	✓	✓
All degree	✗	✓	✗	✗
Correct	✓	✓	✓	✓
Complete	✓	✓	✗	✓
Complexity	✓	✗	✓	✓
Conditions	✓	✓	✓	✓
Nested loops	✓	✓	✓	✗

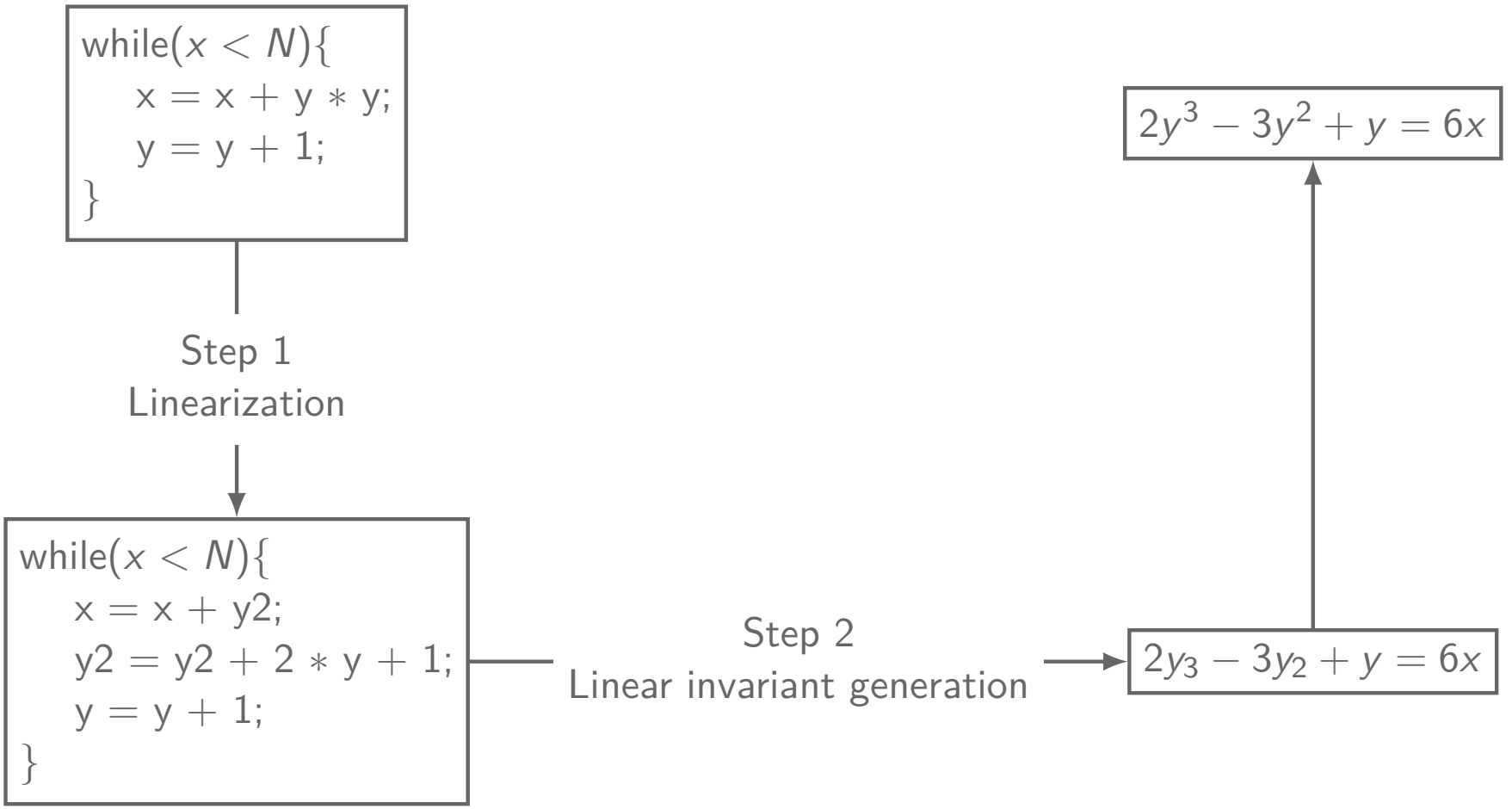
[Kar76] M. Karr. *Affine Relationships Among Variables of a Program.*

[Kov] L. Kovács. *A complete invariant generation approach for p-solvable loops.*

[CJJK14] D. Cachera. *Inference of polynomial invariants for imperative programs: A farewell to Gröbner bases.*

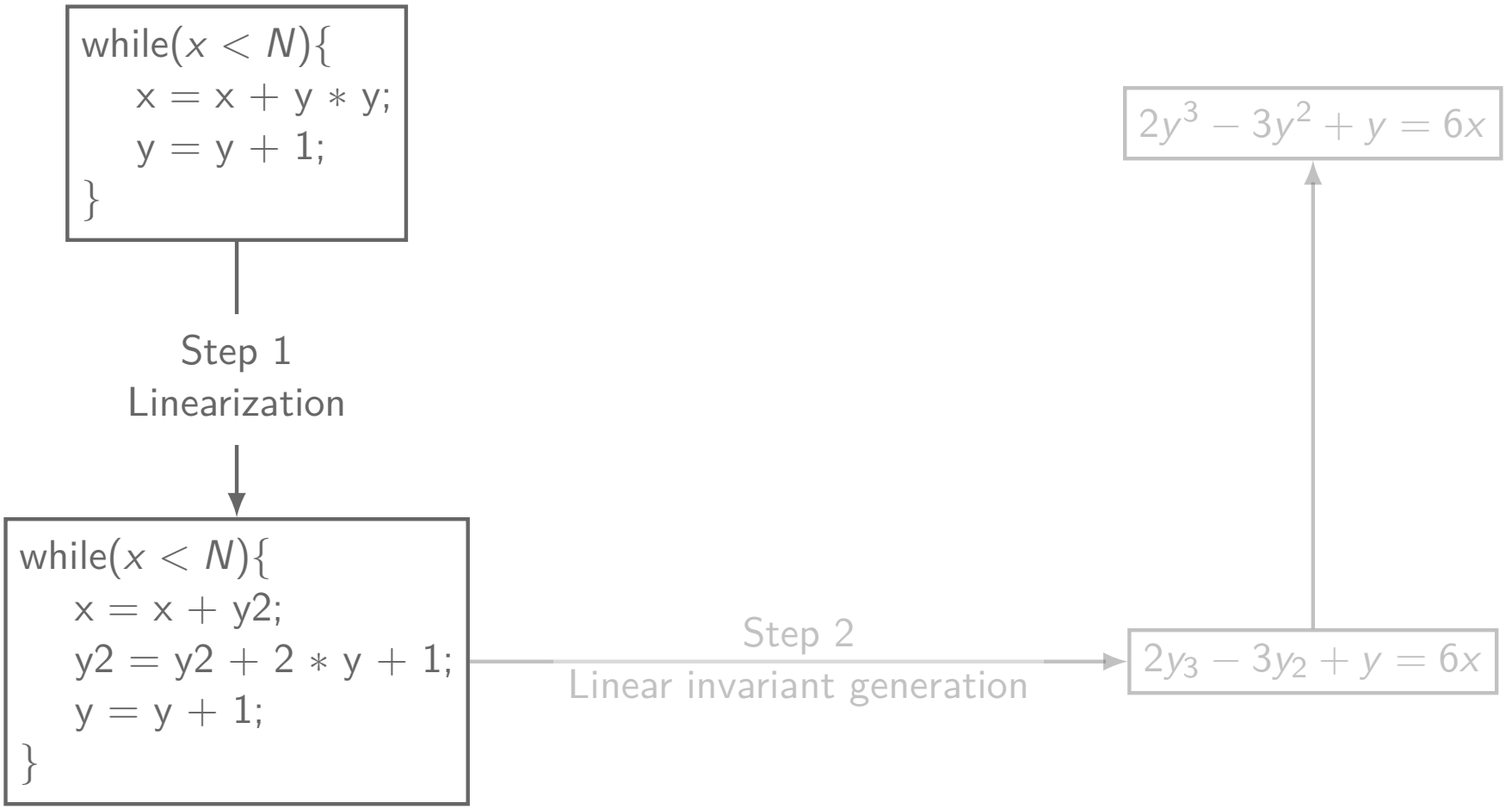
Pilat

Two **independent** steps



Pilat

Two **independent** steps



From linear to polynomial

Operator strength elevation

```
y = 0;
```

```
y = y + 1;
```

- y^2 not linear
- Idea : computing the next value of y^2 without variable multiplication

From linear to polynomial

Operator strength elevation

```
y = 0;
y2 = 0;

y2 = y2 + 2 * y + 1;
y = y + 1;
```

- y^2 not linear
- Idea : computing the next value of y^2 without variable multiplication
- $(y+1)^2 = y^2 + 2.y + 1$

Linearization

Our idea : removing monomials from assignments

```
x = 0;
y = 0;
y2 = 0;
```

```
y2 = y2 + 2 * y + 1;
y = y + 1;
x = x + y * y
```

- y^2 not linear
- Idea : computing the next value of y^2 without variable multiplication
- $(y+1)^2 = y^2 + 2.y + 1$

Linearization

Our idea : removing monomials from assignments

```
x = 0;
y = 0;
y2 = 0;
```

```
y2 = y2 + 2 * y + 1;
y = y + 1;
x = x + y2;
```

- y^2 not linear
- Idea : computing the next value of y^2 without variable multiplication
- $(y+1)^2 = y^2 + 2.y + 1$

Linearization

Our idea : removing monomials from assignments

```
x = 0;
y = 0;
y2 = 0;
one = 1;

y2 = y2 + 2 * y + one;
y = y + one;
x = x + y2;
```

- y^2 not linear
- Idea : computing the next value of y^2 without variable multiplication
- $(y+1)^2 = y^2 + 2.y + 1$
- Equivalent, but this is linear

Does this work on any assignment?

```
x = x * x;
```

- $x.x$ monomial
- $x^2.x^2 = x^4$
- $x^4.x^4 = x^8$
- ...

So far...

- If a variable assignment is affine, then any polynomial of this variable can be computed by an affine assignment.
  $(y = y + 1 \Rightarrow y^2 = y^2 + 2y + 1)$
- If an assignment for a variable x transforms it polynomially, then it is not linearizable.
  $(x = x^2)$

Thus

We need to restrict to assignments that *does not polynomially transform a variable itself*.

Thus

We need to restrict to assignments that *does not polynomially transform a variable itself*.

- Affine transformations ($y = y + 1$)

Thus

We need to restrict to assignments that *does not polynomially transform a variable itself*.

- Affine transformations ($y = y + 1$)
- Affine transformations, plus polynomial transformations of the previous variables ($x = x + y^2$)

Thus

We need to restrict to assignments that *does not polynomially transform a variable itself*.

- Affine transformations ($y = y + 1$)
- Affine transformations, plus polynomial transformations of the previous variables ($x = x + y^2$)
- etc.

Formally

Definition [RCK07]

Let $g \in \mathbb{Q}[X]^m$ a polynomial affectation. g is a solvable mapping if there exist a partition of the variables into subvectors $x = w_1 \uplus ... \uplus w_k$ such that $\forall j.\ 1 \leqslant j \leqslant k$ we have

$$g_{w_j}(x) = M_j w_j^T + P_j(w_1, ..., w_{j-1})$$

with M_j a linear transformation

Proposition

Solvable mappings are linearizable

Example

$$g(x, y) = (x + y^2, y + 1)$$

```
while( y < N ) {
  x = x + y * y;

  y = y + 1;
}
```

Example

```
while( y < N ) {
   x = x + y * y;

   y = y + 1;
}
```

$$g(x,y) = (x + y^2, y + 1) \rightarrow$$

$$\begin{array}{lllll} w_1 & = & (y) & w_2 & = & (x) \\ P_1() & = & 1 & P_2(y) & = & y^2 \end{array}$$

Example

```
while( y < N ) {

  x = x + y * y;

  y = y + 1;

}
```

$$g(x,y) = (x + y^2, y + 1) \rightarrow$$

$$\begin{array}{lllll} w_1 & = & (y) & w_2 & = & (x) \\ P_1() & = & 1 & P_2(y) & = & y^2 \end{array}$$

$$g_{w_1}(y) = y + P_1()$$
$$g_{w_2}(x) = x + P_2(y)$$

Example

```
while( y < N ) {

  x = x + y * y;

  y = y + 1 ;

}
```

$$g(x,y) = (g_{w_2}(x), g_{w_1}(y)) \rightarrow$$

$$\begin{array}{cccccc} w_1 & = & (y) & \quad w_2 & = & (x) \\ P_1() & = & 1 & \quad P_2(y) & = & y^2 \end{array}$$

$$g_{w_1}(y) = y + P_1()$$
$$g_{w_2}(x) = x + P_2(y)$$

Example

```
while( y < N ) {

  x = x + y2;

  y2 = y2 + 2.y + one;
  y = y + one;

}
```

$$g(x, y) = (g_{w_2}(x), g_{w_1}(y)) \rightarrow$$

$$\begin{array}{lllll} w_1 & = & (y) & w_2 & = & (x) \\ P_1() & = & 1 & P_2(y) & = & y^2 \end{array}$$

$$g_{w_1}(y) = y + P_1()$$
$$g_{w_2}(x) = x + P_2(y)$$

g is *linearized* by
$f(x, y, y_2, \mathbb{1}) =$
$(x + y_2, y + \mathbb{1}, y_2 + 2.y + \mathbb{1}, \mathbb{1})$

More variables

- f works on 4 variables $(x, y, y_2, \mathbb{1})$
- Our invariant generator will focus on variables used in f
- g admits one invariant of degree 3 (y^3)

If f does not work with $y_3 = y^3$, the generation will fail.
$\rightarrow$ we must provide a degree

Example

```
while( y < N ){

x = x + y2;

y2 = y2 + 2.y + one;
y = y + one;

}
```

$$(y+1)^3 = y^3 + 3y^2 + 3y + 1$$

Example

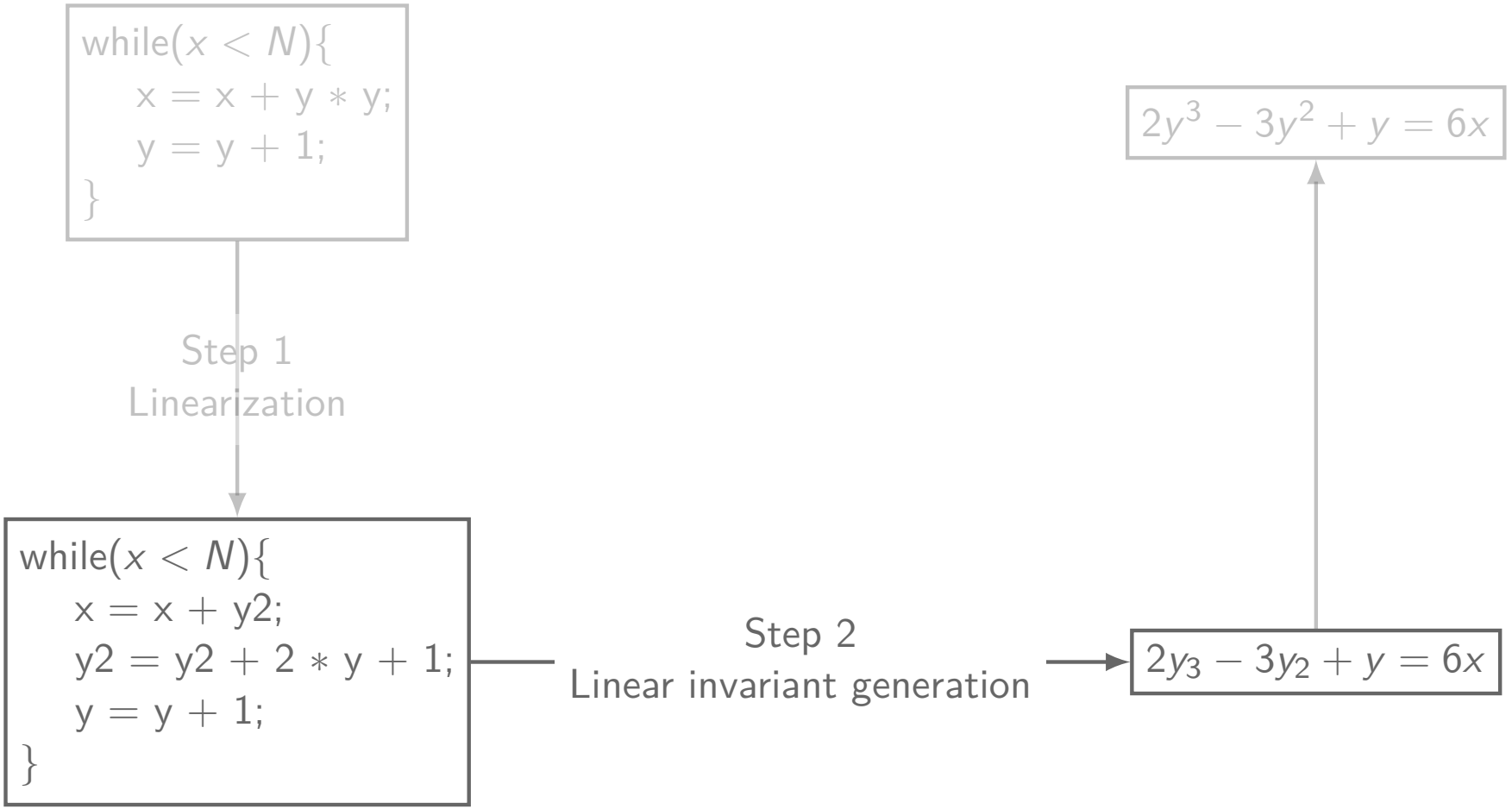
```
while( y < N ){

x = x + y2;
y3 = y3 + 3.y2 + 3.y + one;
y2 = y2 + 2.y + one;
y = y + one;

}
```

$$(y+1)^3 = y^3 + 3y^2 + 3y + 1$$

Next step


while($x < N$){
x = x + y ∗ y;
y = y + 1;
}
Step 1
Linearization
while($x < N$){
x = x + y2;
y2 = y2 + 2 ∗ y + 1;
y = y + 1;
}
Step 2
Linear invariant generation
$2y_3 - 3y_2 + y = 6x$
$2y^3 - 3y^2 + y = 6x$

What is a loop invariant ?

Logical formula φ that :

- is true at the beginning of the loop (initialisation);
- is true at every step of the loop (induction).

Invariants as vectors

Invariants : linear combination of variables always equal to 0.
Example :

$$-6x + y - 3y_2 + 2y_3 = 0$$

Vectorial representation :

$$(-6, 1, -3, 2).(x, y, y_2, y_3)^T = 0$$

$\varphi = (-6, 1, -3, 2)$ invariant.

Idea

Induction criterion : (IC)

$$\varphi(X) = 0 \Rightarrow \varphi(f(X)) = 0$$

By linear algebra :

$$\varphi(X) = 0 \Rightarrow f^*(\varphi)(X) = 0$$

with f^* the dual application of f.

Idea

If φ eigenvector of f^* (i.e. $f^*(\varphi) = \lambda.\varphi$), then

$$\varphi(X) = 0 \Rightarrow f^*(\varphi)(X) = 0$$

becomes

$$\varphi(X) = 0 \Rightarrow \lambda.\varphi(X) = 0$$

which is always true.

$\rightarrow$ Eigenvectors satisfy the IC.

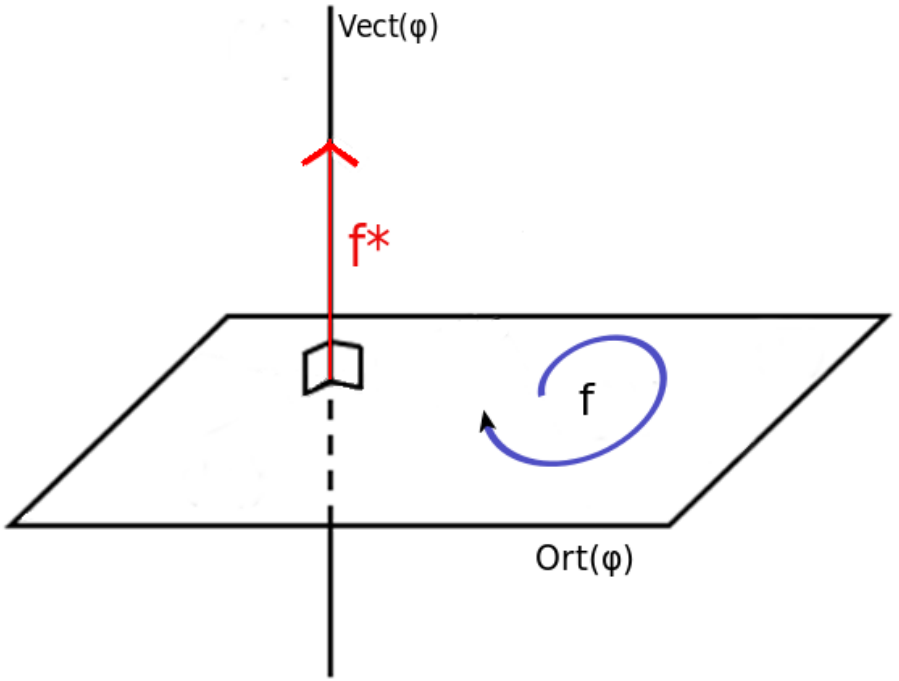
Completeness

Proposition

Only eigenvectors satisfy the IC

Idea of the proof

$$\varphi(X) = 0 \Rightarrow \varphi(f(X)) = 0$$



Example

$$f(x, y, y_2, y_3, \mathbb{1}) = (x + y_2, y + \mathbb{1}, y_2 + 2.y + \mathbb{1}, y_3 + 3.y_2 + 3.y + \mathbb{1}, \mathbb{1})$$

Matrix representation

$$A = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 2 & 1 & 0 & 1 \\ 0 & 3 & 3 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Matrix representation of f^* is A^T.

$$A^T = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 1 & 0 & 1 & 3 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example

$$f(x, y, y_2, y_3, \mathbb{1}) = (x + y_2, y + \mathbb{1}, y_2 + 2.y + \mathbb{1}, y_3 + 3.y_2 + 3.y + \mathbb{1}, \mathbb{1})$$

$$A^T = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 1 & 0 & 1 & 3 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$A^T * \begin{pmatrix} -6 \\ 1 \\ -3 \\ 2 \\ 0 \end{pmatrix} = \begin{pmatrix} -6 \\ 1 \\ -3 \\ 2 \\ 0 \end{pmatrix} \qquad A^T * \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Example

Eigenvectors of A^T

- $e_1 = (-6, 1, -3, 2, 0)$
- $e_2 = (0, 0, 0, 0, 1)$

Example

Eigenvectors of A^T

- $e_1 = (-6, 1, -3, 2, 0)$
- $e_2 = (0, 0, 0, 0, 1)$

$\Rightarrow \forall k_1, k_2$
$k_1.e_1 + k_2 e_2$ is an invariant

Example

Eigenvectors of A^T

- $e_1 = (-6, 1, -3, 2, 0)$
- $e_2 = (0, 0, 0, 0, 1)$

$\Rightarrow \forall k_1, k_2$
$k_1.e_1 + k_2 e_2$ is an invariant

With $X = (x, y, y_2, y_3, 1)^T$,

$\rightarrow (k_1.e_1 + k_2.e_2).X = 0$ satisfies the IC

$\rightarrow e_1.X = k.e_2.X$ satisfies the IC $(k = -\frac{k_2}{k_1})$

$\rightarrow -6x + 1.y - 3y_2 + 2y_3 = k$ satisfies the IC

Example

Eigenvectors of A^T

- $e_1 = (-6, 1, -3, 2, 0)$
- $e_2 = (0, 0, 0, 0, 1)$

$\Rightarrow \forall k_1, k_2$
$k_1.e_1 + k_2 e_2$ is an invariant

With $X = (x, y, y_2, y_3, 1)^T$,

$\rightarrow (k_1.e_1 + k_2.e_2).X = 0$ satisfies the IC

$\rightarrow e_1.X = k.e_2.X$ satisfies the IC $(k = -\frac{k_2}{k_1})$

$\rightarrow -6x + 1.y - 3y_2 + 2y_3 = k$ satisfies the IC

With $X_{init} = (x_{init}, y_{init}, y_{2_{init}}, y_{3_{init}}, 1)^T$
$\rightarrow -6x_{init} + y_{init} - 3y_{2_{init}} + 2y_{3_{init}} = k$
$\rightarrow -6x + y - 3y_2 + 2y_3 = k$ is an invariant

Example

Eigenvectors of A^T

- $e_1 = (-6, 1, -3, 2, 0)$
- $e_2 = (0, 0, 0, 0, 1)$

$\Rightarrow \forall k_1, k_2$
$k_1.e_1 + k_2 e_2$ is an invariant

With $X = (x, y, y_2, y_3, 1)^T$,

$\rightarrow (k_1.e_1 + k_2.e_2).X = 0$ satisfies the IC

$\rightarrow e_1.X = k.e_2.X$ satisfies the IC $(k = -\frac{k_2}{k_1})$

$\rightarrow -6x + 1.y - 3y_2 + 2y_3 = k$ satisfies the IC

With $X_{init} = (0, 0, 0, 0, 1)^T$
$\rightarrow 0 = k$
$\rightarrow -6x + y - 3y_2 + 2y_3 = 0$ is an invariant

Conditions

```
int prod (int a,int b){
  int x = a,y = b;
  int z = 0;

  while( y!=0 ){
    if ( y % 2 ==1 ){
      z = z+x;
      y = y-1;
      }
    x = 2*x;
    y = y/2;
    }
  return z;
  }
```

2 possible paths :

- $g_1(x, y, z) = (2x, y/2, z)$
- $g_2(x, y, z) = (2x, (y-1)/2, z+x)$

Each one have a different invariant base

Intersection

$g_1(x, y, z) = (2x, y/2, z)$
Invariants :

- $\langle e_1, X \rangle = xy$
- $\langle e_2, X \rangle = z$
- $\langle e_3, X \rangle = z^2$
- $\langle e_4, X \rangle = \mathbb{1}$

$g_2(x, y, z) = (2x, (y-1)/2, z + x)$
Invariants :

- $\langle e'_1, X \rangle = -x + xy$
- $\langle e'_2, X \rangle = x + z$
- $\langle e'_3, X \rangle = xz + x^2 + z^2$
- $\langle e'_4, X \rangle = \mathbb{1}$.

$$Vect(\{e_i\}_{i\in[1,4]}) \cap Vect(\{e'_i\}_{i\in[1,4]}) = Vect(\{e_1 + e_2, e_4\})$$

- $\langle e_1 + e_2, X \rangle = xy + z$
- $\langle e_4, X \rangle = 1$

$\Rightarrow xy + z = k$ is an inductive invariant.

In a nutshell

About complexity

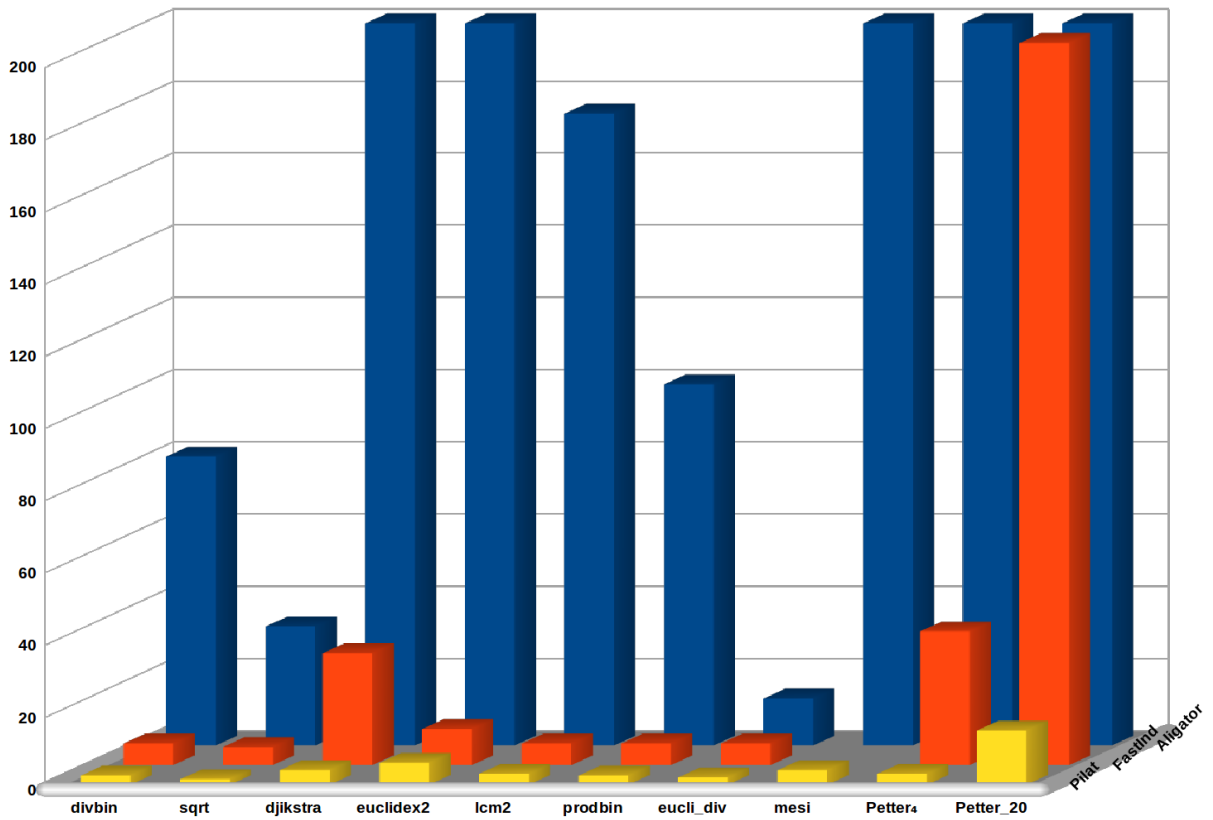
With d the input degree and n the number of initial variables

- Linearization : generates $\binom{n+d}{n} = O(d^n)$ variables
- Invariant generation :
  1. Matrix multiplication / transpose : $O(|M|^3)$ / $O(|M|^2)$
  2. Eigenvector problem : $O(|M|^3)$

$\rightarrow$ for a given number of variables n, polynomial complexity.

Pilat, a new invariant generator

- Frama-C plugin, developped in OCaml, open source (very soon)



In practice

Input			Time (in ms)		
Name	Var	Deg	Aligator	Fast Ind	Pilat
divbin	5	2	80	6	2.5
sqrt	4	2	33	5	1.5
djikstra	5	2	279	31	4
euclidex2	8	2	1759	10	6
lcm2	6	2	175	6	3
prodbin	5	2	100	6	2.5
fermat2	5	2	30	9	2
eucli_div	3	2	13	6	2
read_writ	6	2	82	–	12
mesi	4	2	620	–	4
petter_4	2	10	19000	37	3
petter_20	2	21	$>$ 10 mn	41000	15

Conclusion

So far

- Program linearization
- New fast loop invariant generation
- Pilat, Frama-C plugin generating annotations

Next steps

- What about irrational eigenvectors ?
- What about nested loops ?
- What about complex data structures ?
- What about non deterministic assignments ?

Conclusion


David Cachera, Thomas Jensen, Arnaud Jobin, and Florent Kirchner.
Inference of polynomial invariants for imperative programs: A farewell to gröbner bases.
Science of Computer Programming, 93:89–109, 2014.


Michael Karr.
Affine relationships among variables of a program.
Acta Informatica, 6(2):133–151, 1976.


Laura Kovács.
A complete invariant generation approach for p-solvable loops.
In *Perspectives of Systems Informatics*, pages 242–256.


Enric Rodríguez-Carbonell and Deepak Kapur.
Generating all polynomial invariants in simple loops.
Journal of Symbolic Computation, 42(4):443–476, 2007.