Inference of polynomial invariants for imperative programs

[Cachera+, SAS12]

Kohei Asano

What I did in the undergraduate.

- ☀ Create a poster that summarize Cacheras' work[SAS12]
- ☀ Implement it in Rust from almost scratch

→ In this presentation, I presents summary of Cacheras' work[SAS12], and introduce few related work.

Summary of Cachera et al.,[SAS12]

- Presents a **sound** method of computing **polynomial equality invariants** for imperative programs
  - Syntax is restricted to polynomial (dis)equality conditions and assignments.
- Presents a **fast polynomial invariant inference method**

Input:
Imperative program

```
x = x0; v = v0; t = t0;
while (t - t1) != 0 do {
    x = x + v * dt;
    v = v - g * dt;
    t = t + dt;
}
```

Output:
Polynomial equality invariants
at the end of a program(post condition)

$$tg - t_0 g + v - v_0 = 0$$
$$x_0 tg - x_0 t_0 g + x_0 v - x_0 v_0 = 0$$

Table of Contents

Table of Contents

What are invariants good for?

- Safety Verification, Termination Analysis

```
1  x = x0; v = v0; t = t0;
2  while (t - t1) != 0 do {
3      x = x + v * dt;
4      v = v - g * dt;
5      t = t + dt;
6  }
7  assert( t * g - t_0 * g + v - v_0 == 0 );
```

→ **This assertion never fails,** because $tg - t_0 g + v - v_0 = 0$ is invariant at location l_7(line 7).

-

What are invariants good for?

☀ Safety Verification, Termination Analysis

```
1  x = x0; v = v0; t = t0;
2  while (t - t1) != 0 do {
3      x = x + v * dt;
4      v = v - g * dt;
5      t = t + dt;
6  }
7  assert( t * g - t_0 * g + v - v_0 == 0 );
```

→ **This assertion never fails,**

☀ (inequality invariants) Inferring Complexity bounds [Breck+ POPL20]

```
int subsetSumAux(int * A, int i, int n, int sum) {
    nTicks++;
    if (i >= n) {
        if (sum == 0) { found = true; }
        return 0;
    }
    int size = subsetSumAux(A, i + 1, n, sum + A[i]);
    if (found) { return size + 1; }
    size = subsetSumAux(A, i + 1, n, sum);
    return size;
}
```

auxiliary variable

$$\mathtt{nTicks}' \leq \mathtt{nTicks} + 2^h - 1 \ \wedge \ \mathtt{return}' \leq h - 1 \wedge$$
$$h \leq \max(1, 1 + \mathtt{n} - \mathtt{i})$$

→ **This function call takes exponential time in n in the worst case.**

Definition of Polynomial invariant

In this slides $\mathbb{R}^m$ represents all of program states.

☀ (First-order) Assertion φ, which doesn't contain any quantifiers, is **Invariant at location** l

: $\Longleftrightarrow$ φ is true whatever program state reaching at l.

☀

```
1  x = x0; v = v0; t = t0;
2  while (t - t1) != 0 do {
3      x = x + v * dt;
4      v = v - g * dt;
5      t = t + dt;
6  }
7  assert( t * g - t_0 * g + v - v_0 == 0 );
```

$tg - t_0 g + v - v_0 = 0$ is an invariant at location l_7(line 7).

Definition of Polynomial invariant

In this slides $\mathbb{R}^m$ represents all of program states.

☀ (First-order) Assertion φ, which doesn't contain any quantifiers, is **Invariant at location** l

: $\Longleftrightarrow$ φ is true whatever program state reaching at l.

```
1  x = x0; v = v0; t = t0;
2  while (t - t1) != 0 do {
3      x = x + v * dt;
4      v = v - g * dt;
5      t = t + dt;
6  }
7  assert( t * g - t_0 * g + v - v_0 == 0 );
```

☀ φ is **polynomial (equality) invariant at** l

: $\Longleftrightarrow$ Invariant φ at location l can be written as

$$\varphi(x_1, \ldots, x_m) = \bigwedge_i p_i(x_1, .., x_m) = 0 \quad p_i \in \mathbb{R}[x_1, \ldots x_m], \quad x_1, \ldots x_m \text{ are program variables}$$

$tg - t_0 g + v - v_0 = 0$ is an invariant at location l_7(line 7).

Invariants can be seen as any **over-approximations of reachable sets**,

polynomial invariants can be seen as any **over-approximations of reachable sets by ideals**.

≈ Equation systems

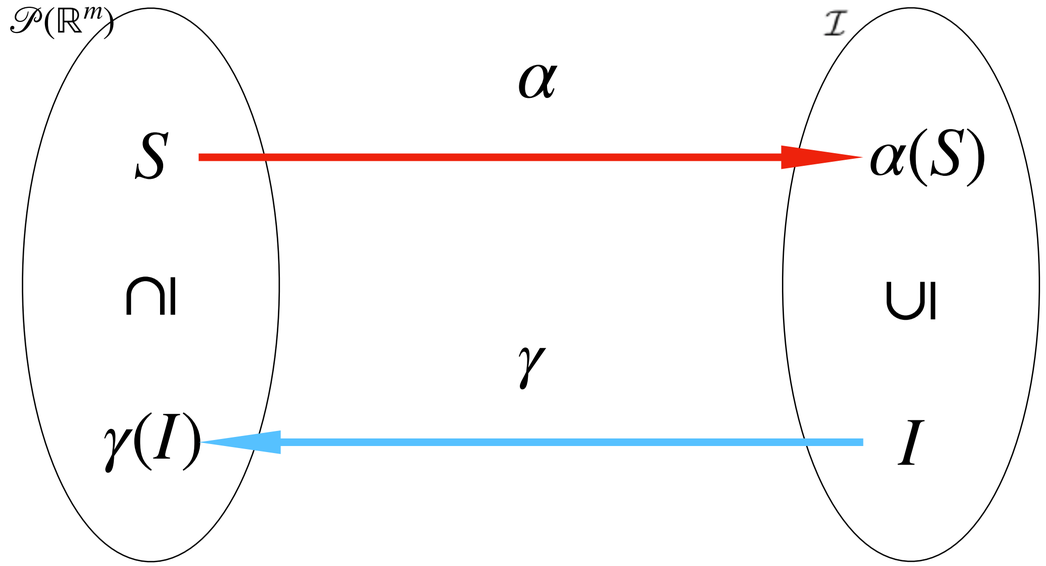
Approximation by Ideal Domain

This work uses **Abstract Interpretation on Ideal Domain** based on following **Galois Connection** to compute polynomial invariant.

Galois Connection

$$\gamma(I) \supseteq S \iff I \subseteq \alpha(S)$$

$$\begin{aligned}
\alpha: \quad \mathcal{P}(\mathbb{R}^m) &\to \mathcal{I} \\
X &\mapsto \{u \in \mathbb{R}[\mathtt{x_1}, \ldots, \mathtt{x_m}] \mid \forall x \in X,\ u(x) = 0\} \\
\gamma: \quad \mathcal{I} &\to \mathcal{P}(\mathbb{R}^m) \\
I &\mapsto \{x \in \mathbb{R}^m \mid \forall u \in I,\ u(x) = 0\}
\end{aligned}$$



Concrete Domain
(States' power sets)

Abstract Domain
(Real Ideals)

Overview of polynomial invariant computation in this work

Define **Concrete Weakest Precondition Predicate Transformer**(WPPT)$B^{\nu}[\![c]\!]$ and **Abstract WPPT** $[\![c]\!]^{\sharp}$
to ensure $\gamma\left([\![c]\!]^{\sharp}\langle g\rangle\right) \subseteq B^{\nu}[\![c]\!]\gamma\langle g\rangle$

1. Generate fixed degree **Generic Template** $\langle g\rangle$ and compute WP $[\![c]\!]^{\sharp}\langle g\rangle$ in Abstract Domain
2. Solve a constraint $[\![c]\!]^{\sharp}\langle g\rangle \equiv \langle 0\rangle$ and $\gamma\left([\![c]\!]^{\sharp}\langle g\rangle\right) \subseteq B^{\nu}[\![c]\!]\gamma\langle g\rangle$ ensure $g = 0$ is a polynomial invariant.

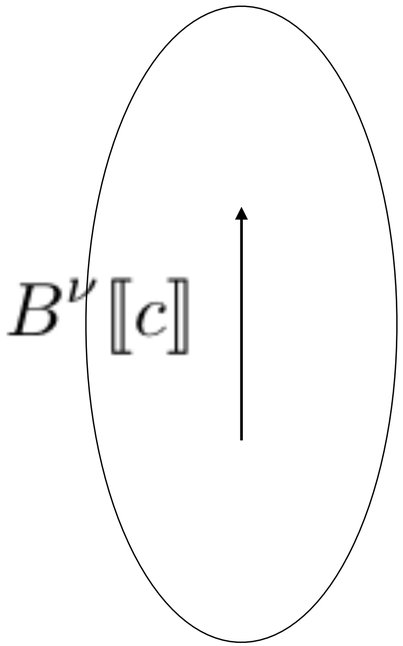
Overview of polynomial invariant computation in this work

Define **Concrete** and **Abstract Weakest Precondition Predicate Transformer**(WPPT) to ensure $\gamma(\llbracket c \rrbracket^\sharp \langle g \rangle) \subseteq B^\nu \llbracket c \rrbracket \gamma \langle g \rangle$

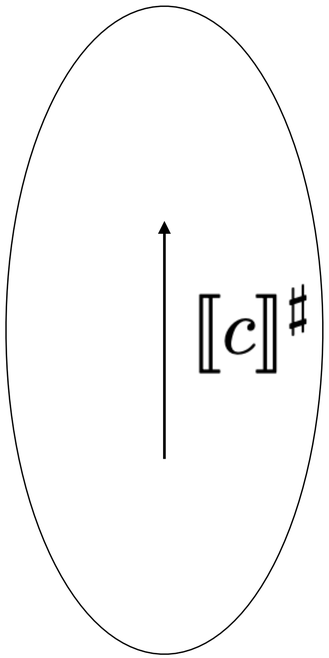
$$\begin{array}{rccl} \gamma : & \mathcal{I} & \to & \mathcal{P}(\mathbb{R}^m) \\ & I & \mapsto & \{x \in \mathbb{R}^m \mid \forall u \in I,\ u(x) = 0\} \end{array}$$

$B^\nu \llbracket c \rrbracket$: **Concrete WPPT**

$\llbracket c \rrbracket^\sharp$: **Abstract WPPT**



Concrete Domain



Abstract Domain: $\mathcal{I}$

Overview of polynomial invariant computation in this work

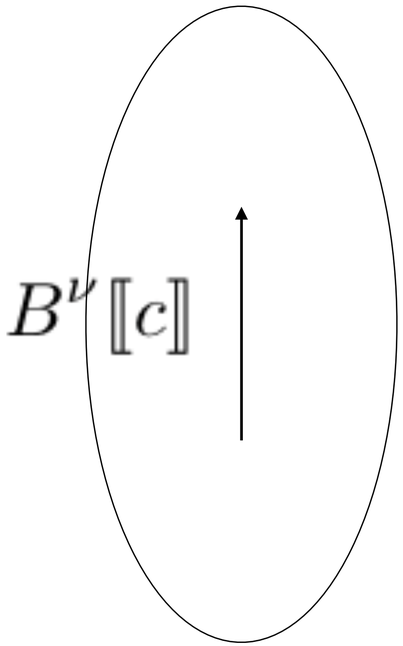
1. Generate fixed degree **Generic Template** $\langle g \rangle$ and compute WP$[\![c]\!]^\sharp\langle g \rangle$ in Abstract Domain

$$\begin{array}{rccl} \gamma: & \mathcal{I} & \to & \mathcal{P}(\mathbb{R}^m) \\ & I & \mapsto & \{x \in \mathbb{R}^m \mid \forall u \in I,\ u(x) = 0\} \end{array}$$

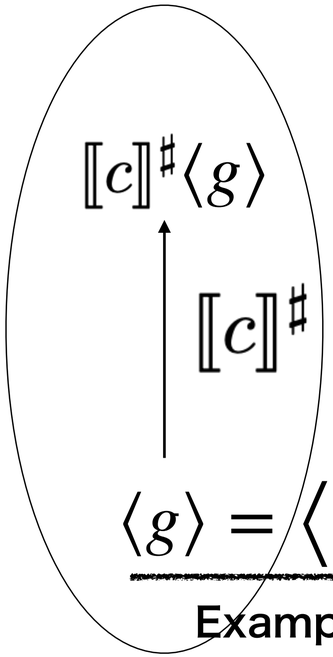
$B^\nu[\![c]\!]$: **Concrete WPPT**

$[\![c]\!]^\sharp$: **Abstract WPPT**

$$\gamma\left([\![c]\!]^\sharp\langle g\rangle\right) \subseteq B^\nu[\![c]\!]\gamma\langle g\rangle$$



Concrete Domain



Abstract Domain: $\mathcal{I}$

$$\langle g \rangle = \left\langle a_5x^2 + a_4xy + a_3y^2 + a_2x + a_1y + a_0 \right\rangle$$

Example of degree 2 most Generic Template with x, y

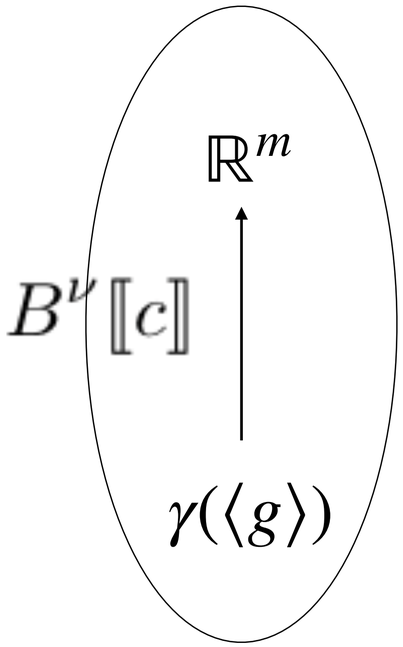
Overview of polynomial invariant computation in this work

2. Solve a constraint $[\![c]\!]^\sharp\langle g\rangle \equiv \langle 0\rangle$ and $\gamma\left([\![c]\!]^\sharp\langle g\rangle\right) \subseteq B^\nu[\![c]\!]\gamma\langle g\rangle$ ensure $g = 0$ is a polynomial invariant.

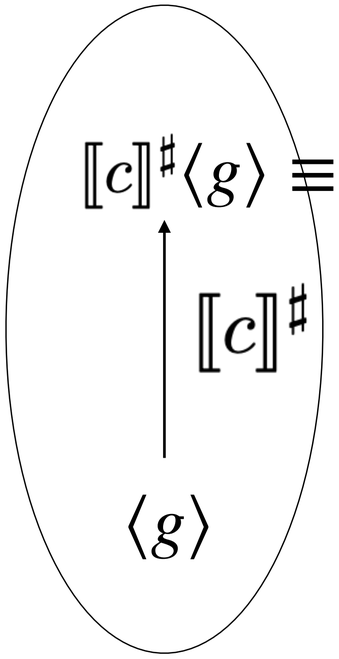
$$\begin{array}{rccl} \gamma: & \mathcal{I} & \to & \mathcal{P}(\mathbb{R}^m) \\ & I & \mapsto & \{x \in \mathbb{R}^m \mid \forall u \in I,\ u(x) = 0\} \end{array}$$

$B^\nu[\![c]\!]$: **Concrete WPPT**

$[\![c]\!]^\sharp$: **Abstract WPPT**



Concrete Domain



Abstract Domain: $\mathcal{I}$

$[\![c]\!]^\sharp\langle g\rangle = \langle 0\rangle \implies \gamma(\langle 0\rangle) = \mathbb{R}^m \subseteq B^\nu[\![c]\!]\gamma(\langle g\rangle) = \mathbb{R}^m$ and

$B^\nu[\![c]\!]\gamma(\langle g\rangle) = \mathbb{R}^m \implies g = 0$ is polynomial invariant

Table of Contents

Syntax

- The imperative language syntax of which is restricted to polynomial guards and assignments
  - A variant of Language IMP[Winskel, 1993]
  - No functions

$$
\begin{array}{lcll}
p \in \mathbb{R}[\mathtt{x}_1, \ldots, \mathtt{x}_m] & & & \textit{polynomials} \\
\mathbb{V} \ni \mathit{var} & ::= & \mathtt{x}_1 \mid \ldots \mid \mathtt{x}_m & \textit{program variables} \\
\mathbb{T} \ni \mathit{test} & ::= & p \bowtie 0 & \textit{polynomial guards} \\
\mathbb{P} \ni c & ::= & \mathit{var} := p & \textit{polynomial assignments} \\
 & & \mid c \,;\, c & \textit{sequence} \\
 & & \mid \textbf{\textit{if}}\ \mathit{test}\ \textbf{\textit{then}}\ c\ \textbf{\textit{else}}\ c & \textit{conditional structure} \\
 & & \mid \textbf{\textit{while}}\ \mathit{test}\ \textbf{\textit{do}}\ c & \textit{loop structure} \\
 & & \mid \textbf{\textit{skip}} & \textit{skip assertion}
\end{array}
$$

where $\bowtie$ *stands for* $=$ *or* $\neq$. *We will also use* $\not\bowtie$ *for the negation of* $\bowtie$.

Operational Semantics

☀ Standard Operational Semantics

$$\frac{}{\langle c_1; c_2, \sigma\rangle \sim \langle c_2, \langle c_1, \sigma\rangle\rangle}\ \textbf{\textit{seq}} \qquad \frac{\sigma \to \sigma'}{\langle c, \sigma\rangle \to \langle c, \sigma'\rangle}\ \textbf{\textit{congruence}}$$

$$\frac{}{\langle \textbf{skip}, \sigma\rangle \to \sigma}\ \textbf{\textit{skip}} \qquad \frac{p(\sigma) = v}{\langle \mathtt{x_j} := p, \sigma\rangle \to \sigma[v]_\mathtt{j}}\ \textbf{\textit{assign}}$$

$$\frac{p(\sigma) \bowtie 0 \equiv \mathit{true}}{\langle \textbf{if } p \bowtie 0 \textbf{ then } c_1 \textbf{ else } c_2, \sigma\rangle \to \langle c_1, \sigma\rangle}\ \textbf{\textit{if}} \quad \textit{where} \quad \left.\begin{array}{l} 0 = 0 \\ v \neq 0 \end{array}\right\} \equiv \mathit{true}$$

$$\frac{p(\sigma) \bowtie 0 \equiv \mathit{false}}{\langle \textbf{if } p \bowtie 0 \textbf{ then } c_1 \textbf{ else } c_2, \sigma\rangle \to \langle c_2, \sigma\rangle}\ \textbf{\textit{if}} \quad \textit{where} \quad \left.\begin{array}{l} 0 \neq 0 \\ v = 0 \end{array}\right\} \equiv \mathit{false}$$

$$\frac{}{\langle \textbf{while } b \textbf{ do } c, \sigma\rangle \sim \langle \textbf{if } b \textbf{ then } (c; \textbf{while } b \textbf{ do } c) \textbf{ else skip}, \sigma\rangle}\ \textbf{\textit{while}}$$

Notation: given $\sigma = (\sigma_1, \ldots, \sigma_m)$, we note $\sigma[v]_\mathtt{j}$ the updated state $(\sigma_1, \ldots, \sigma_{j-1}, v, \sigma_{j+1}, \ldots, \sigma_m)$.

Concrete WPPT(Weakest Precondition Predicate Transformer)

☀ Compute **Weakest (liberal) precondition** from program c and post-condition S.

$$B^{\nu}[\![c]\!] : \mathcal{P}(\mathbb{R}^m) \to \mathcal{P}(\mathbb{R}^m)$$

$$B^{\nu}[\![\mathtt{x_j} := p]\!]\ S = \{x \in \mathbb{R}^m \mid x[p(x)]_{\mathtt{j}} \in S\}$$
where $x[p(x)]_{\mathtt{j}}$ is the element $(x_1, \ldots, x_{j-1}, p(x), x_{j+1}, \ldots, x_m)$

$$B^{\nu}[\![\textbf{\textit{skip}}]\!]\ S = S$$

$$B^{\nu}[\![c_1; c_2]\!]\ S = B^{\nu}[\![c_1]\!]\ (B^{\nu}[\![c_2]\!]\ S)$$

$$B^{\nu}[\![\textbf{\textit{if}}\ p \bowtie 0\ \textbf{\textit{then}}\ c_1\ \textbf{\textit{else}}\ c_2]\!]\ S = (B^{\nu}[\![c_1]\!]\ S \bigcap [\![p \bowtie 0]\!]) \bigcup (B^{\nu}[\![c_2]\!]\ S \bigcap [\![p \not\bowtie 0]\!])$$
where $[\![p \bowtie 0]\!] = \{x \in \mathbb{R}^m \mid p(x) \bowtie 0\}$

$$B^{\nu}[\![\textbf{\textit{while}}\ p \bowtie 0\ \textbf{\textit{do}}\ c]\!]\ S = \underline{\nu F_{c,p,S}}$$ Greatest fix point operator induce partial correctness
where $F_{c,p,S} = \lambda X.([\![p \not\bowtie 0]\!] \bigcap S) \bigcup ([\![p \bowtie 0]\!] \bigcap B^{\nu}[\![c]\!]\ X)$

Example of Concrete WPPT Computation

$c :=$ `while x - 1 != 0 do x := x + 1;`

$B^{\nu}[\![c]\!]\{1\}$

$= \nu F \qquad \textbf{where}\, F = \lambda X.([\![x - 1 = 0]\!] \cap \{1\}) \cup ([\![x - 1 \neq 0]\!] \cap B^{\nu}[\![x := x + 1]\!]X)$

$= \mathbb{R}$ $\rightarrow$**partial correctness** $\{x \in \mathbb{R}\}c\{x = 1\}$

Example of Concrete WPPT Computation

$c :=$ `while x - 1 != 0 do x := x + 1;`

$B^{\nu}[\![c]\!]\{1\}$

$= \nu F \qquad \textbf{where}\, F = \lambda X.([\![x-1=0]\!] \cap \{1\}) \cup ([\![x-1 \neq 0]\!] \cap B^{\nu}[\![x := x+1]\!]X)$

$= \mathbb{R}$ $\rightarrow$**partial correctness** $\{x \in \mathbb{R}\}c\{x = 1\}$

Generally, $\sigma \in B^{\nu}[\![c]\!]\ S \Rightarrow \begin{cases} \exists \sigma_f \in S, & \langle c, \sigma \rangle \rightarrow^{+} \sigma_f \\ \text{or} & \langle c, \sigma \rangle \rightarrow^{\infty} \end{cases}$ holds.

Example of Concrete WPPT Computation

$c :=$ `while x - 1 != 0 do x := x + 1;`

$B^{\nu}[\![c]\!]\{1\}$

$= \nu F \qquad \textbf{where}\, F = \lambda X.([\![x-1=0]\!] \cap \{1\}) \cup ([\![x-1 \neq 0]\!] \cap B^{\nu}[\![x := x+1]\!]X)$

$= \mathbb{R}$ $\rightarrow$**partial correctness** $\{x \in \mathbb{R}\}c\{x = 1\}$

Generally, $\sigma \in B^{\nu}[\![c]\!]\ S \Rightarrow \left\{ \begin{array}{rl} \exists \sigma_f \in S, & \langle c, \sigma \rangle \rightarrow^{+} \sigma_f \\ \text{or} & \langle c, \sigma \rangle \rightarrow^{\infty} \end{array} \right.$ holds.

$B^{\mu}[\![c]\!]\{1\}$

$= \mu F$

$= \{1, 0, -1, -2, \cdots\}$ $\rightarrow$**total correctness** $[x \in \mathbb{Z} \wedge x \leq 1]c[x = 1]$

Abstract WPPT

☀ Compute **Polynomial Weakest precondition** from program c and polynomial post-condition I.

$$[\![c]\!]^\sharp : \mathcal{I} \to \mathcal{I}$$

$$[\![x_j := p]\!]^\sharp I = \langle \{q[x_j \mapsto p], q \in I\} \rangle$$

$$[\![\textbf{skip}]\!]^\sharp I = I$$

$$[\![s_1; s_2]\!]^\sharp I = [\![s_1]\!]^\sharp([\![s_2]\!]^\sharp I)$$

$$[\![\textbf{if } p \neq 0 \textbf{ then } c_1 \textbf{ else } c_2]\!]^\sharp I = p \cdot ([\![c_1]\!]^\sharp I) \sqcap^\sharp \mathrm{Rem}([\![c_2]\!]^\sharp I, p)$$

$$[\![\textbf{if } p = 0 \textbf{ then } c_1 \textbf{ else } c_2]\!]^\sharp I = p \cdot ([\![c_2]\!]^\sharp I) \sqcap^\sharp \mathrm{Rem}([\![c_1]\!]^\sharp I, p)$$

$$[\![\textbf{while } p \neq 0 \textbf{ do } c]\!]^\sharp I = \nu(F^\sharp_{c,p,I})$$

where $F^\sharp_{c,p,I} = \lambda J.\ p \cdot ([\![c]\!]^\sharp J) \sqcap^\sharp \mathrm{Rem}(I, p)$

$$[\![\textbf{while } p = 0 \textbf{ do } c]\!]^\sharp I = \nu(\overline{F}^\sharp_{c,p,I})$$

where $\overline{F}^\sharp_{c,p,I} = \lambda J.\ p \cdot I \sqcap^\sharp \mathrm{Rem}([\![c]\!]^\sharp J, p)$

$$\gamma([\![c]\!]^\sharp \langle g \rangle) \subseteq B^\nu [\![c]\!] \gamma(\langle g \rangle)$$

$I \sqcap^\sharp J$: an ideal generated by sum of I and J's generator

$\mathrm{Rem}(I, p)$: an ideal generated by **remainders** when I's generators are divided by p.

Example of Computing Abstract WPPT

☀ Interpretation of $P :=$ `if x - y = 0 then x := x + 1; else y := x + 1;`

Abstract Domain

$$[\![P]\!]^\sharp \langle x-1, y-2\rangle$$

$$= \langle x-y\rangle \cdot \langle x, y-2\rangle \sqcap^\sharp \mathrm{Rem}(\langle (x-1, y-1\rangle, x-y)$$

$\mathrm{Rem}(f, p)$: Graded reverse lexicographical order

$$= \left\langle x^2 - xy, xy - y^2 - 2x + 2y \right\rangle \sqcap^\sharp \langle y-1\rangle$$

$$= \left\langle x^2 - xy, xy - y^2 - 2x + 2y, y-1 \right\rangle$$

Example of Computing Abstract WPPT

☀ Interpretation of $P :=$ `if x - y = 0 then x := x + 1; else y := x + 1;`

Abstract Domain

$[\![P]\!]^\sharp \langle x-1, y-2\rangle$ → $\gamma(\langle x-1, y-2\rangle) = \{(1,2)\}$

$= \langle x-y\rangle \cdot \langle x, y-2\rangle \sqcap^\sharp \mathrm{Rem}(\langle (x-1, y-1\rangle, x-y)$

$\mathrm{Rem}(f,p)$: Graded reverse lexicographical order

$= \langle x^2 - xy, xy - y^2 - 2x + 2y\rangle \sqcap^\sharp \langle y-1\rangle$

$= \langle x^2 - xy, xy - y^2 - 2x + 2y, y-1\rangle$ → $\gamma(\langle x^2 - xy, xy - y^2 - 2x + 2y, y-1\rangle) = \{(1,1)\}$

Example of Computing Abstract WPPT

☀ Interpretation of $P :=$ `if x - y = 0 then x := x + 1; else y := x + 1;`

Abstract Domain

$[\![P]\!]^\sharp \langle x-1, y-2\rangle$ ⟶ $\gamma(\langle x-1, y-2\rangle) = \{(1,2)\}$

$= \langle x-y\rangle \cdot \langle x, y-2\rangle \sqcap^\sharp \mathrm{Rem}(\langle (x-1, y-1\rangle, x-y)$

$\mathrm{Rem}(f,p)$: Graded reverse lexicographical order

$= \langle x^2 - xy, xy - y^2 - 2x + 2y\rangle \sqcap^\sharp \langle y-1\rangle$

$= \langle x^2 - xy, xy - y^2 - 2x + 2y, y-1\rangle$ ⟶ $\gamma(\langle x^2 - xy, xy - y^2 - 2x + 2y, y-1\rangle) = \{(1,1)\}$

Concrete Domain

$B^\nu[\![P]\!]\{(1,2)\}$

$= (B^\nu[\![\texttt{x := x + 1;}]\!]\{(1,2)\} \cap [\![x-y=0]\!]) \cup (B^\nu[\![\texttt{y := x + 1;}]\!]\{(1,2)\} \cap [\![x-y\neq 0]\!])$

$= \{(1,1)\}$

$\gamma([\![c]\!]^\sharp \langle g\rangle) \subseteq B^\nu[\![c]\!]\gamma(\langle g\rangle)$

Correctness(Soundness)

Theorem 3 *(Correctness). Let g be a polynomial in $\mathbb{R}[x_1, \ldots, x_m]$ and c be a polynomial program. Then:*

$$\gamma\left(\llbracket c \rrbracket^\sharp \langle g \rangle\right) \subseteq B^\nu \llbracket c \rrbracket \gamma \langle g \rangle$$

$$\gamma : \quad \begin{array}{ccc} \mathcal{I} & \to & \mathcal{P}(\mathbb{R}^m) \\ I & \mapsto & \{x \in \mathbb{R}^m \mid \forall u \in I,\ u(x) = 0\} \end{array}$$

proved by structural induction with properties of ideals, and transfer lemma.

Lemma 2 *(Transfer lemma). Let $(\mathcal{A}, \subseteq)$, $(\mathcal{A}^\sharp, \subseteq^\sharp)$ be two complete lattices and $\gamma : \mathcal{A}^\sharp \to \mathcal{A}$ a function. Let $f : \mathcal{A} \to \mathcal{A}$ and $f^\sharp : \mathcal{A}^\sharp \to \mathcal{A}^\sharp$ be two monotonic functions such that:*

$$\gamma \circ f^\sharp \dot{\subseteq} f \circ \gamma$$

Then we have:

$$\gamma\left(\nu f^\sharp\right) \subseteq \nu f$$

Correctness(Soundness)

Theorem 3 *(Correctness). Let g be a polynomial in $\mathbb{R}[x_1, \ldots, x_m]$ and c be a polynomial program. Then:*

$$\gamma\left(\llbracket c \rrbracket^\sharp \langle g \rangle\right) \subseteq B^\nu \llbracket c \rrbracket \gamma \langle g \rangle$$

$$\gamma : \begin{array}{ccc} \mathcal{I} & \to & \mathcal{P}(\mathbb{R}^m) \\ I & \mapsto & \{x \in \mathbb{R}^m \mid \forall u \in I,\ u(x) = 0\} \end{array}$$

proved by structural induction with properties of ideals, and transfer lemma.

$\llbracket c \rrbracket^\sharp \langle g \rangle = \langle 0 \rangle \implies \gamma(\langle 0 \rangle) = \mathbb{R}^m \subseteq B^\nu \llbracket c \rrbracket \gamma(\langle g \rangle) = \mathbb{R}^m$ and

$B^\nu \llbracket c \rrbracket \gamma(\langle g \rangle) = \mathbb{R}^m \implies g = 0$ is polynomial invariant

ensure the Correctness of this work's method.

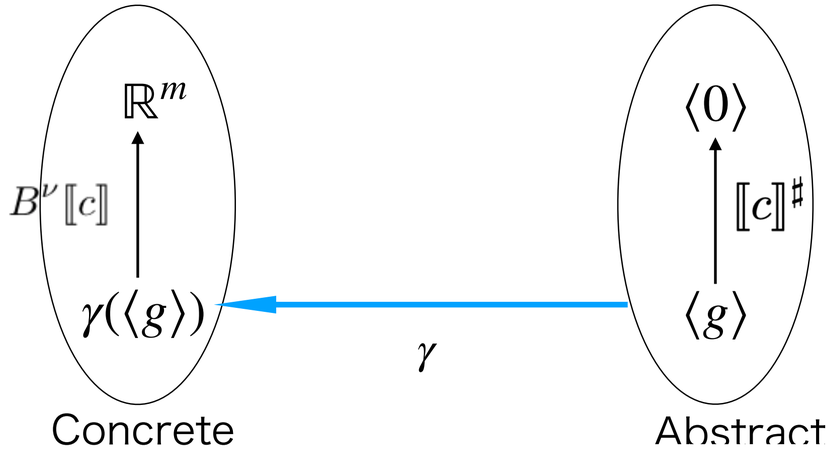


Table of Contents

Complexity problem of ideal fixed point iteration

$$[\![\textbf{while } p = 0 \textbf{ do } c]\!]^\sharp I = \nu(\overline{F}^\sharp_{c,p,I})$$

where $\overline{F}^\sharp_{c,p,I} = \lambda J.\ p \cdot I \sqcap^\sharp \mathrm{Rem}([\![c]\!]^\sharp J, p)$

$$\nu(\overline{F}^\sharp_{c,p,I}) = \bigcup_n (\overline{F}^\sharp_{c,p,I})^n(\langle 0 \rangle)$$

Computing in Ideal Domain ensure

the termination of ascending chain $\langle 0 \rangle \subseteq \overline{F}^\sharp_{c,p,I}(\langle 0 \rangle) \subseteq (\overline{F}^\sharp_{c,p,I})^2(\langle 0 \rangle) \subseteq \cdots$ (by ordinary inclusion order)

but **its iteration number upper bound is unknown**.

Complexity problem of ideal fixed point iteration

$$[\![\textbf{while } p = 0 \textbf{ do } c]\!]^\sharp I = \nu(\overline{F}^\sharp_{c,p,I})$$

where $\overline{F}^\sharp_{c,p,I} = \lambda J.\ p \cdot I \sqcap^\sharp \mathrm{Rem}([\![c]\!]^\sharp J, p)$

$$\nu(\overline{F}^\sharp_{c,p,I}) = \bigcup_n (\overline{F}^\sharp_{c,p,I})^n(\langle 0 \rangle)$$

Computing in Ideal Domain ensure

the termination of ascending chain $\langle 0 \rangle \subseteq \overline{F}^\sharp_{c,p,I}(\langle 0 \rangle) \subseteq (\overline{F}^\sharp_{c,p,I})^2(\langle 0 \rangle) \subseteq \cdots$

but **its iteration number upper bound is unknown**.

Moreover, Gröbner bases computation, which is necessary to decide the termination of Kleene Iteration, is **EXP-SPACE complete**[W. Mayr, 1996]

$$\overline{F}^\sharp_{c,p,I}(\langle 0 \rangle)^n \supseteq (\overline{F}^\sharp_{c,p,I})^{n+1}(\langle 0 \rangle)$$

Complexity problem of ideal fixed point iteration

$$[\![\textbf{while } p = 0 \textbf{ do } c]\!]^\sharp I = \nu(\overline{F}^\sharp_{c,p,I})$$
$$\text{where } \overline{F}^\sharp_{c,p,I} = \lambda J.\ p \cdot I \sqcap^\sharp \mathrm{Rem}([\![c]\!]^\sharp J, p)$$

$$\nu(\overline{F}^\sharp_{c,p,I}) = \bigcup_n (\overline{F}^\sharp_{c,p,I})^n(\langle 0 \rangle)$$

Computing in Ideal Domain ensure

the termination of ascending chain $\langle 0 \rangle \subseteq \overline{F}^\sharp_{c,p,I}(\langle 0 \rangle) \subseteq (\overline{F}^\sharp_{c,p,I})^2(\langle 0 \rangle) \subseteq \cdots$

but **its iteration number upper bound is unknown**.

Moreover, Gröbner bases computation, which is necessary to decide the termination of Kleene Iteration, is **EXP-SPACE complete**[W. Mayr, 1996]

$$\overline{F}^\sharp_{c,p,I}(\langle 0 \rangle)^n \supseteq (\overline{F}^\sharp_{c,p,I})^{n+1}(\langle 0 \rangle)$$

→ This work try to **inference** a part of **polynomial invariants**

≈ narrow the search space and shorten the computation

Avoiding fixed point iteration

$$[\![\textbf{while } p \neq 0 \textbf{ do } c]\!]^\sharp I = \nu(F^\sharp_{c,p,I})$$
$$\text{where } F^\sharp_{c,p,I} = \lambda J.\ p \cdot ([\![c]\!]^\sharp J) \sqcap^\sharp \mathtt{Rem}(I, p)$$

$$[\![\textbf{while } p = 0 \textbf{ do } c]\!]^\sharp I = \nu(\overline{F}^\sharp_{c,p,I})$$
$$\text{where } \overline{F}^\sharp_{c,p,I} = \lambda J.\ p \cdot I \sqcap^\sharp \mathtt{Rem}([\![c]\!]^\sharp J, p)$$

Idea: Focus on loop invariants by a constraint $[\![c]\!]^\sharp I \equiv I$ **equality of $\mathbb{R}^m$ zeros of polynomials**

→ Compute **constraints for loop invariants** together with loop semantics

Avoiding fixed point iteration

$$[\![\textbf{while } p \neq 0 \textbf{ do } c]\!]^\sharp I = \nu(F^\sharp_{c,p,I})$$
$$\text{where } F^\sharp_{c,p,I} = \lambda J.\ p \cdot ([\![c]\!]^\sharp J) \sqcap^\sharp \mathtt{Rem}(I, p)$$

$$[\![\textbf{while } p = 0 \textbf{ do } c]\!]^\sharp I = \nu(\overline{F}^\sharp_{c,p,I})$$
$$\text{where } \overline{F}^\sharp_{c,p,I} = \lambda J.\ p \cdot I \sqcap^\sharp \mathtt{Rem}([\![c]\!]^\sharp J, p)$$

Idea: Focus on loop invariants by a constraint $[\![c]\!]^\sharp I \equiv I$

→ Compute **constraints for loop invariants** together with loop semantics

☀Correctness for dis-equality guards is given as follows

Theorem 5. *Let $I \in \mathcal{I}$ and $\mathrm{w} \equiv$ **while** $p \neq 0$ **do** c be a polynomial program. Suppose that $[\![c]\!]^\sharp I = I$. Then $[\![\mathrm{w}]\!]^\sharp I = I$.*

!! Correctness for equality guards is not given in this work. !!

Refined Semantics for fast inferece

- Refined Semantics without fixed point computation

$$[\![c]\!]^{\sharp c} : \mathcal{I}_{par} \times \mathcal{C} \to \mathcal{I}_{par} \times \mathcal{C}$$

Parametrized Ideals and
Constraints w.r.t parameters

$$[\![x_j := p]\!]^{\sharp c}(I, C) = (\langle \{q[x_j \mapsto p], q \in I\} \rangle, C)$$

$$[\![\textbf{skip}]\!]^{\sharp c}(I, C) = (I, C)$$

$$[\![s_1; s_2]\!]^{\sharp c}(I, C) = ([\![s_1]\!]^{\sharp c}([\![s_2]\!]^{\sharp c}(I, C)))$$

$$[\![\textbf{if } p \neq 0 \textbf{ then } c_1 \textbf{ else } c_2]\!]^{\sharp c}(I, C) = (p \cdot I_1 \sqcap^{\sharp} \mathrm{Rem}_{par}(I_2, p), C_1 \cup C_2)$$
$$[\![\textbf{if } p = 0 \textbf{ then } c_1 \textbf{ else } c_2]\!]^{\sharp c}(I, C) = (p \cdot I_2 \sqcap^{\sharp} \mathrm{Rem}_{par}(I_1, p), C_1 \cup C_2)$$
$$\textit{where } [\![c_1]\!]^{\sharp c}(I, C) = (I_1, C_1)$$
$$\textit{and } [\![c_2]\!]^{\sharp c}(I, C) = (I_2, C_2)$$

$$[\![\textbf{while } p \bowtie 0 \textbf{ do } c]\!]^{\sharp c}(I, C) = \left(I, C' \cup C_w\right)$$
$$\textit{where } [\![c_1]\!]^{\sharp c}(I, C) = (I', C')$$
$$\textit{and } C_w = \{I \equiv I'\}$$

Refined Semantics for fast inferece

☀ Refined Semantics without fixed point computation

$$[\![c]\!]^{\sharp c} : \mathcal{I}_{par} \times \mathcal{C} \to \mathcal{I}_{par} \times \mathcal{C}$$

Parametrized Ideals and Constraints w.r.t parameters

$$[\![x_j := p]\!]^{\sharp c}(I, C) = (\langle \{q[x_j \mapsto p], q \in I\} \rangle, C)$$

$$[\![\textbf{skip}]\!]^{\sharp c}(I, C) = (I, C)$$

$$[\![s_1; s_2]\!]^{\sharp c}(I, C) = ([\![s_1]\!]^{\sharp c}([\![s_2]\!]^{\sharp c}(I, C)))$$

$$[\![\textbf{if } p \neq 0 \textbf{ then } c_1 \textbf{ else } c_2]\!]^{\sharp c}(I, C) = (p \cdot I_1 \sqcap^{\sharp} \mathrm{Rem}_{par}(I_2, p), C_1 \cup C_2)$$
$$[\![\textbf{if } p = 0 \textbf{ then } c_1 \textbf{ else } c_2]\!]^{\sharp c}(I, C) = (p \cdot I_2 \sqcap^{\sharp} \mathrm{Rem}_{par}(I_1, p), C_1 \cup C_2)$$
$$\textit{where } [\![c_1]\!]^{\sharp c}(I, C) = (I_1, C_1)$$
$$\textit{and } [\![c_2]\!]^{\sharp c}(I, C) = (I_2, C_2)$$

$$[\![\textbf{while } p \bowtie 0 \textbf{ do } c]\!]^{\sharp c}(I, C) = \left(I, C' \cup C_w\right)$$
$$\textit{where } [\![c_1]\!]^{\sharp c}(I, C) = (I', C')$$
$$\textit{and } C_w = \{I \equiv I'\}$$

Ignore while guards,
and **impose constraints** which requires
equality of pre and post loop ideals, $[\![c]\!]^{\sharp} I \equiv I$

Algorithm

Input: program c, degree d,

Output: a set of polynomials $\mathcal{G}$,

begin

$g :=$ the most generic a_i-polynomial of degree d;

computing abstract semantics $(I, C) = [\![c]\!]^{\sharp_C} \langle g \rangle$;

generating $\mathscr{C}_{g,c}$, the constraint $C \cup (I \equiv \langle 0 \rangle)$;

computing $\mathscr{S}_{g,c}$, set of solutions of $\mathscr{C}_{g,c}$;

$\mathcal{G} :=$

set of polynomials obtained by a_i-instantiating g by elements of $\mathscr{S}_{g,c}$

end

Algorithm

Input: program c, degree d,

Output: a set of polynomials $\mathcal{G}$,

Example of most generic template of degree 3, with program variables x, y, z

$$a_{17}x^3 + a_{16}x^2y + a_{15}xy^2 + a_{14}y^3 + a_{13}x^2z + a_{12}y^2z + a_{11}xz^2 + a_{10}yz^2 + a_9z^3 + a_8x^2 + a_7xy + a_6y^2 + a_5xz + a_4z^2 + a_3x + a_2y + a_1z + a_0$$

begin

$g :=$ the most generic a_i-polynomial of degree d;

computing abstract semantics $(I, C) = [\![c]\!]^{\sharp_C}\langle g \rangle$;

generating $\mathscr{C}_{g,c}$, the constraint $C \cup (I \equiv \langle 0 \rangle)$;

computing $\mathscr{S}_{g,c}$, set of solutions of $\mathscr{C}_{g,c}$;

$\mathcal{G} :=$

set of polynomials obtained by a_i-instantiating g by elements of $\mathscr{S}_{g,c}$

end

Monomial numbers are ${}_n\mathrm{H}_d$

where n is number of program variables

Example

$\mathrm{c} :=$ `x = 0;i = 0;while( i != n ){x = x + i;i = i + 1;}`

$x = \frac{1}{2}n^2 - n, x = \frac{1}{2}i^2 - i$ are post conditions, former is not loop invariant.

Example

$\mathrm{c} :=$ `x = 0;i = 0;while( i != n ){x = x + i;i = i + 1;}`

$\mathrm{g} := \mathrm{a}_9 x^2 + a_8 xi + a_7 i^2 + a_6 xn + a_5 in + a_4 n^2 + a_3 x + a_2 i + a_1 n + a_0$

$[\![c]\!]^{\sharp c}(\langle g \rangle, \emptyset)$

$= [\![\texttt{x = 0;i = 0;}]\!]^{\sharp c}(\langle g \rangle, \{g \equiv h\})$

$= (\langle a_4 n^2 + a_1 n + a_0 \rangle, \{\langle g \rangle \equiv \langle h \rangle\})$

$h := a_9 x^2 + (a_8 + 2a_9) xi + (a_7 + a_8 + a_9) i^2 + a_6 xn + (a_5 + a_6) in + a_4 n^2 + (a_3 + a_8) x + (a_2 + a_3 + 2a_7 + a_8) i + (a_1 + a_5) n + a_0 + a_2 + a_7$

If we assume correspondence of all coefficients (sufficient but not necessary).

$\{\langle g \rangle \equiv \langle h \rangle, \langle a_4 n^2 + a_1 n + a_0 \rangle \equiv \langle 0 \rangle\} \iff$

$$\begin{gathered} a_0 = 0 \\ a_1 = 0 \\ a_4 = 0 \\ a_5 = 0 \\ a_6 = 0 \\ a_8 = 0 \\ a_9 = 0 \\ a_2 + a_7 = 0 \\ a_3 + 2a_7 = 0 \end{gathered}$$

$\Longrightarrow$

$$a_2 = t, a_3 = 2t, a_7 = -t$$

$\forall t \in \mathbb{R} \;\; g = 2tx + ti - ti^2 = 0$ is invariant.

Table of Contents

Related work

- Generalized Homogeneous Polynomials for Efficient Template-Based Nonlinear Invariant Synthesis[Kojima+. SAS16]
  - Reduce number of monomials in generic templates by focusing on generalized homogeneous polynomials.
  - Determine the Generalized degree of polynomials in programs by using Dimensional Type Inference.

Related work

☀ Templates and Recurrences: Better Together [Breck+. POPL20]

- Compute non-linear inequality invariants for Non-linearly recursive programs.

Input:
C programs

```
int subsetSumAux(int * A, int i, int n, int sum) {
    nTicks++;
    if (i >= n) {
        if (sum == 0) { found = true; }
        return 0;
    }
    int size = subsetSumAux(A, i + 1, n, sum + A[i]);
    if (found) { return size + 1; }
    size = subsetSumAux(A, i + 1, n, sum);
    return size;
}
```

Output:
Non-linear inequality invariants.

$$\texttt{nTicks}' \leq \texttt{nTicks} + 2^h - 1 \ \wedge \ \texttt{return}' \leq h - 1 \wedge$$
$$h \leq \max(1, 1 + \texttt{n} - \texttt{i})$$

Benchmark	Actual	CHORA	ICRA	Other Tools
fibonacci	$O(\varphi^n)$	$O(2^n)$	n.b.	[2]:$O(2^n)$
hanoi	$O(2^n)$	$O(2^n)$	n.b.	[2]:$O(2^n)$
subset_sum	$O(2^n)$	$O(2^n)$	n.b.	[20]:$O(2^n)$
bst_copy	$O(2^n)$	$O(2^n)$	n.b.	[2]:$O(2^n)$
ball_bins3	$O(3^n)$	$O(3^n)$	n.b.	[20]:$O(3^n)$
karatsuba	$O(n^{\log_2(3)})$	$O(n^{\log_2(3)})$	n.b.	[9]:$O(n^{1.6})$
mergesort	$O(n\log(n))$	$O(n\log(n))$	n.b.	[2]:$O(n\log(n))$
strassen	$O(n^{\log_2(7)})$	$O(n^{\log_2(7)})$	n.b.	[9]:$O(n^{2.9})$
qsort_calls	$O(n)$	$O(2^n)$	$O(n)$	[8]:$O(n)$
qsort_steps	$O(n^2)$	$O(n2^n)$	n.b.	[9]:$O(n^2)$
closest_pair	$O(n\log(n))$	n.b.	n.b.	[9]:$O(n\log(n))$
ackermann	$Ack(n)$	n.b.	n.b.	[2]:n.b.

Related work

- Polynomial Invariant Generation for Non-deterministic Recursive Programs[Chatterjee+. PLDI20]
  - Presents semi-complete polynomial inequality invariants generation method for non-linearly recursive C programs.
  - Sub-exponential time complexity.

References

- Müller-Olm and Seidl(2002). Polynomial Constants Are decidable
- Sankaranarayanan, Sriram and Sipma, Henny B. and Manna, Zohar (2004). Non-linear loop invariant generation using Gröbner bases.
- Bruno Blanchet(2002). Introduction to Abstract Interpretation
- Deepak Kapur(2004). Automatically Generating Loop Invariants Using Quantifier Elimination –Preliminary Report–.
- Glynn Winskel(1993). The Formal Semantics of Programming Languages: An Introduction