Generating Loop Invariants via Polynomial Interpolation

Marc Moreno Maza and Rong Xiao

University of Western Ontario
Ontario Research Center for Computer Algebra

The problem

A loop invariant is a condition of the loop variables that is always true at the entry point of the loop. We are interested in *computing polynomial equations which are loop invariants* for the following type of loops in programs.

while C_0 **do**
  if C_1
  then
    $X := A_1(X);$
  elif C_2
  then
    $X := A_2(X);$
  $\ldots$
  elif C_m
  then
    $X := A_m(X);$
  end if
end while

Notations and assumptions:

- loop variables $X = x_1, \ldots, x_s$ are real values (in practice *rational value*) scalar variables;
- all conditions consist of polynomial constrains in X; given a condition C in X, denote by $Z(C)$ all points in $\mathbb{R}^s$ satisfying C;
- denote by I_0 the initial condition;
- for $i = 1, \ldots, m$, we denote by A_i a polynomial in $\mathbb{Q}[X]$ d and ny M_i the corresponding map induced by A_i;
- $C_1, C_2, \ldots, C_m$ are pair-wise exclusive; each implies C_0.

Given a loop $\mathcal{L}$, it is easy to deduce that

$$\{p \in \mathbb{Q}[X] \mid p = 0 \text{ is an invariant of } \mathcal{L}\}$$

forms a *polynomial ideal* in $\mathbb{Q}[X]$, which is called the **invariant ideal** of $\mathcal{L}$.

The proposed method

The proposed approach aims at computing all polynomial invariants up to a given **total degree** d. It is based on *polynomial interpolation* and it is rather straight forward:

step 1: sample a list of (typically $\binom{s+d}{d}$) points S from the trajectory of the given loop;

step 2: compute all the polynomials P up to degree d which have S as zeros;

step 3: return P if *one can verify* that $\wedge_{p \in P}\, p = 0$ is an invariant of the loop; otherwise, the method fails.

Example 1. *Consider the following simple infinite loop:*

$x := 1$; $y := 1$; **while true do** $x := x + 1$; $y := y + x$; **end do; ...**

It is easy to deduce that the trajectory of the loop variables (x, y) *are* $(i, \frac{i\,(i+1)}{2})$ $i = 1 \cdots \infty$ *and* $y = \frac{x\,(x+1)}{2}$ *is an equational invariant of the loop.*

Using our method, to compute polynomial equation invariants of degree ≤ 2*, one would:*

step 1: sample the following points from the loop trajectory

$$S := \{(1,1), (2,3), (6,10), (8,36), (11,66), (15,120), (20,420)\}.$$

step 2: compute all polynomials of x, y *up to degree 2 vanishing on* S*, which turn out to be multiples of* $2\,y - x\,(x+1)$*;*

step 3: one can verify that $2\,y - x\,(x+1) = 0$ *is invariant since* $2\,y - x\,(x+1) = 0$ *implies* $2\,y + x + 1 - (x+1)\,(x+1+1) = 0$*. Therefore, one can conclude that* **all** *polynomial equation invariants of degree* ≤ 2 *are equivalent to* $y = \frac{x\,(x+1)}{2}$.

The method is simple, but 3 non-trivial issues have to handled:

A. a reasonable degree must be supplied. In the next section, we shall estimate the degree bound as well the dimension of the invariant ideals for certain loops.

B. a general criterion to check whether or not a condition is invariant must be developed. A criterion is proposed at the end of this part.

C. the size of sample points might grow dramatically, direct implementation may not be efficient in practice. In out implementation, modular techniques are used to compute the interpolated polynomials.

Proposition. *Given a condition* inv *consisting of polynomial constraints, if*

$$Z(I_0) \subseteq Z(\mathrm{inv})$$

holds and if for each branch, the relation

$$M_i(Z(\mathrm{inv} \wedge C_i)) \subseteq Z(\mathrm{inv})$$

holds, then inv *is a loop invariant.*

The criterion in the above proposition can be easily implemented by set-theoretical operation of semi-algebraic sets, see `SemiAlgebraicSetTools` of `RegularChains` package in Maple 16.

Degree and dimension of invariant ideals

In this part, we study the degrees and dimensions of invariant ideals of certain type of loops: the loops have only one branch and corresponding recurrence equation $X(n+1) = A(X(n))$ induced by the assignment is P-solvable.

Definition (P-solvable recurrence)**.** *An* s*-variable recurrence* R *is called* P-solvable *over* $\mathbb{Q}$ *if it is defined by a relation of the following form:*

$$X(n+1) = M \times X(n) + \left(\mathbf{f}_{1_{n_1 \times 1}}, \mathbf{f}_{2_{n_2 \times 1}}, \cdots, \mathbf{f}_{k_{n_k \times 1}}\right)^T,$$

where M *is an* $s \times s$ *block-diagonal matrix over* $\mathbb{Q}$ *with the following shape:*

$$M := \begin{pmatrix} \mathbf{M}_{n_1 \times n_1} & \mathbf{0}_{n_1 \times n_2} & \cdots & \mathbf{0}_{n_1 \times n_k} \\ \mathbf{0}_{n_2 \times n_1} & \mathbf{M}_{n_2 \times n_2} & \cdots & \mathbf{0}_{n_2 \times n_k} \\ \cdots & \cdots & \cdots & \cdots \\ \mathbf{0}_{n_k \times n_1} & \mathbf{0}_{n_k \times n_2} & \cdots & \mathbf{M}_{n_k \times n_k} \end{pmatrix},$$

$\mathbf{f}_1 \in \mathbb{Q}^{n_1}$*, and* $\mathbf{f}_i$ $(i = 2, \ldots, k)$ *is in* $\mathbb{Q}[x_1, \ldots, x_{n_1 + \cdots + n_{i-1}}]^{n_i}$*. The matrix* M *is called the* coefficient matrix *of* R*.*

Example 2. *All linear recurrences are* P*-solvable.*

Proposition. *For the above defined* P*-solvable recurrence, one can compute poly-geometrical expressions (see definition below) in* n *w.r.t. the eigenvalues of* M *as closed form solutions. Moreover, the degree of each variable in such expressions can be estimated without computing explicitly those forms.*

Definition. *Let* $\alpha_1, \ldots, \alpha_k$ *be* k *elements of* $\overline{\mathbb{Q}}^*$*. Let* n *be a variable taking non-negative integer values. Regard* $n, \alpha_1^n, \ldots, \alpha_s^n$ *as independent variables. Then each polynomial in* $\overline{\mathbb{Q}}[n, \alpha_1^n, \ldots, \alpha_s^n]$ *is called a* poly-geometrical expression *in* n over $\overline{\mathbb{Q}}$ *w.r.t.* $\alpha_1, \ldots, \alpha_k$*.*

Let $A := (\alpha_1, \ldots, \alpha_s) \subset \overline{\mathbb{Q}}^*$. Associate each α_i with a variable y_i. The *multiplicative relation ideal* (MRI) of A is the ideal in $\mathbb{Q}[y_1, \ldots, y_s]$ generated by

$$\{\prod_{j \in \{1,\ldots,s\},\, v_j > 0} y_j^{v_j} - \prod_{i \in \{1,\ldots,s\},\, v_i < 0} y_i^{-v_i} \mid \prod_{i=1}^{s} \alpha_1^{v_1} \cdots \alpha_s^{v_s} = 1\}.$$

Example 3. *The MRI of* $A = (1/2, 1/3, -1/6)$ *associated with* y_1, y_2, y_3 *is* $\langle y_1^2 y_2^2 - y_3^2 \rangle$.

Notations: let R be a P-solvable recurrence relation defining s sequences in $\mathbb{Q}^s$, with recurrence variables $(x_1, x_2, \ldots, x_s)$. Let $\mathcal{I} \subset \mathbb{Q}[x_1, x_2, \ldots, x_s]$ be the invariant ideal of R. Let $\mathcal{M}$ be the MRI of non-zero eigenvalues of R. Denoted by $d_{\mathcal{I}}$ the degree of $\mathcal{I}$.

Theorem. *Suppose* $\mathcal{M}$ *has dimension* r *and degree* $d_{\mathcal{M}}$*. Assume the total degrees of poly-geometrical expression solutions of* R *are estimated to be no more than* d*. Then we have*

$$d_{\mathcal{I}} \leq d_{\mathcal{M}}\, d^{r+1}.$$

Moreover, if the degrees of the variable n *are* 0*, then we have*

$$d_{\mathcal{I}} \leq d_{\mathcal{M}}\, d^{r}.$$

A set of s non-zero non-one numbers is said to be *weakly multiplicatively independent* if they are can be arranged in an order $a_1, a_2, \ldots, a_s$ such that for each $i \in 2 \cdots s$ and $\forall (e_1, e_2, \ldots, e_{i-1}) \in (\mathbb{N} \cup \{0\})^{i-1}$, we have $a_i \neq \prod_{j=1}^{i-1} a_j^{e_j}$ holds.

Theorem. *The dimension of* $\mathcal{I}$ *is* at most $r + 1$*. Moreover, for generic initial values:*

1. *the dimension of* $\mathcal{I}$ *is* at least r*;*
2. *if the eigenvalues of* R *are weakly multiplicatively independent (thus not containing* 0*), then* $\mathcal{I}$ *has dimension* r*.*

Implementation and benchmarks

The proposed method has been implemented in Maple, which is the command EquationalLoopInvariants of our developing `ProgramAnalysis` package.

The `ProgramAnalysis` package will contain functionality to: automatically generate invariants, verify specifications and verify termination for **while** loops; optimize **for** loops for better data locality, ...

```
> with(ProgramAnalysis);
[CreateLoop, DisplayLoop, EquationalLoopInvariants, InequationalLoopInvariants, IsLoopInvariant,
  MultivariateInterpolation, SetPostcondition, SetPreconditon, VerifyLoop]                        (2)
> mplfile := "/home/rxiao/mapleP4/lib/ProgramAnalysis/tst/programs/fermat.mpl":
  loop := CreateLoop(mplfile);  # parse loop from a mpl file
                                  loop := loop_sextuple                                           (3)
> EquationalLoopInvariants(loop);
                              [u^2 - v^2 - 4 r - 2 u + 2 v - 4 N]                                 (4)
> vars := [a,b,d,y,Q]; initialization := [0,Q/2,1,0]; guard := [[E<=d]];
  transitions := [[[P<a+b],[a,b/2,d/2,y,Q]],[[P>=a+b],[a+b,b/2,d/2,y+b/2,Q]]];
  precond := [[Q>P, P>=0, E>0]]; postcond := [[P >= Q*y ,  Q*y > P - Q*E  ]];
                                  vars := [a, b, d, y, Q]
                              initialization := [0, 1/2 Q, 1, 0]
                                   guard := [[E <= d]]
   transitions := [[[P < a + b], [a, 1/2 b, 1/2 d, y, Q]], [[a + b <= P], [a + b, 1/2 b, 1/2 d, y + 1/2 b, Q]]]
                              precond := [[P < Q, 0 <= P, 0 < E]]
                             postcond := [[Q y <= P, P - Q E < Q y]]                              (5)
> loop := CreateLoop(vars, initialization,  transitions,  guard, precond, postcond):
  eq_invs := EquationalLoopInvariants(loop);
                               eq_invs := [a - 2 y, d Q - 2 b]                                    (6)
> eq_invs:= map(t->t=0, eq_invs):
  ineq_invs := [ P - Q*d < Q*y, Q*y <= P,y>=0]:
  all_invariants:=[op(ineq_invs), op(eq_invs)]:
  VerifyLoop(loop, 'user_invariants'=all_invariants); # verify if a loop satisfy its
  specification
                                           true                                                   (7)
```

Timings of 4 polynomial equation loop invariants generating methods: the proposed method (PI); a method based on abstract interpolation (AI) and a method based on fix point method (FP), both are developed by D. Kapur and E.R odríguez-Carbonell; the method based on solving recurrences explicitly (SE), developed by L. Kavocs.

prog.	# vars	deg	PI	AI	FP	SE
cohencu	4	3	0.6	0.93	0.28	0.13
cohencu	4	2	0.06	0.76	0.28	0.13
fermat	5	4	3.74	0.79	0.37	0.1
prodbin	5	3	1.4	0.74	0.36	0.13
rk07	6	3	3.1	2.23	NA	0.35
kov08	3	3	0.2	0.57	0.22	0.01
sum5	4	5	3.5	1.60	2.25	0.16
wensley2	3	3	0.4	0.84	0.39	0.21
int-factor	6	3	10.3	1.28	160.7	0.9
fib(coupled)	4	4	2.4	0.71	NA	NA
fib(decoupled)	6	4	4.3	1.28	160.7	FAIL
non-inv2*	4	3	1.2	3.83	NA	FAIL
coupled-5-1*	4	4	1.1	9.58	NA	NA
coupled-5-2*	5	4	5.38	15.8	NA	NA
mannadiv	3	3	0.1	0.83	NA	0.04

Conclusion

Though not complete, the proposed method is quite efficient in practice, and applies to broader situations than some other methods (e.g. FP and SE). Our degree and dimension estimates can be used to justify the completeness of our output as well as to supply a reasonable degree bound in other methods which also need a degree bound (e.g. PI).