Inferring Polynomial Invariants with Polyinvar

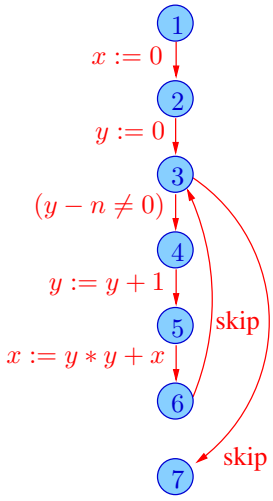
Helmut Seidl and Michael Petter

Technische Universität München, Boltzmannstrasse 3, 85748 Garching, Germany,
{seidl, petter}@cs.tum.edu,
WWW home page: http://www2.cs.tum.edu/~{seidl,petter}

Abstract. Polyinvar is a tool to infer valid polynomial relations at program points in the control flow graph of Java Code. This analysis is achieved by an incremental fixpoint iteration over a set of polynomial modules, representing the weakest precondition for a generic relation of degree n at a given program point.

1 Introduction

Deducing polynomial program invariants is a task with many applications in program verification and optimization [1]. Consider the example program from figure 1 and its control flow graph. We would like to know, how the return value of the program is related to the input parameter.




```
public class powSum{
  public int powSum (int n) {
    int x = 0;
    for(int y = 0; y != n; y = y + 1)
      x = y * y + x;
    return x;
  }
}
```

Fig. 1. Example code and resulting control flow graph

In the given example, we calculate the power sum of degree 2, therefore the relation $2y^3 + 3y^2 + y - 6x = 0$ holds at program point 7. The goal of Polyinvar is to automate the detection of such polynomial program invariants. The termination of the

algorithm proposed in [1] relies on Hilbert's basis theorem for polynomial ideals over a field – which states that these ideals can be represented finitely. Accordingly, no upper bound for the complexity is known. Polyinvar was created to survey, whether a program analysis based on the calculation of a generator set for polynomial ideals nonetheless is feasible for at least some non-trivial programs.

2 Core improvements of the analysis

```
Set fixpointiteration
          (Node u_t, Vector v_t,Set Vars, Set Edges,Set Nodes) {
  Set [] G ⇐ new Set[|Nodes|];
  forall (u ∈ Nodes) G[u] ⇐ ∅;
  Set W ⇐ {(v_t, u_t)};
  while (W ≠ ∅) {
    (v, t) ⇐ extract(W);
    Vector v' ⇐ reduce(v, G[t]);
    if (v' ≠ 0) {
      if (v' < v) G[t] ⇐ G[t] ∪ {v'};
      else G[t] ⇐ G[t] ∪ {v};
      forall ( (s, "skip;", t) ∈ Edges)
        W ⇐ W ∪ {(v, s)};
      forall ( (s, "x_j := p", t) ∈ Edges)
        W ⇐ W ∪ {(v[p/x_j], s)};
      forall ( (s, "(p ≠ 0)", t) ∈ Edges)
        W ⇐ W ∪ {(p · v, s)};
      forall ( (s, "x_j :=?", t) ∈ Edges)
        let l = max({i|ax^i ∈ monoms(v)})
          in let v ⇒ (p_{0_0} x_j^0 + ... + p_{0_l} x_j^l, ..., p_{k_0} x_j^0 + ... + p_{k_l} x_j^l)
            in let v_i ⇐ (p_{0_i}, p_{1_i}, ..., p_{k_i})
              in W ⇐ W ∪ {(v_0, u), ..., (v_l, u)};
    }
  }
  return ⟨G[u_start]⟩;
}
```

Fig. 2. Fixpoint algorithm

The analysis described in [1] is based upon a set of relations between program points and a naive fixpoint iteration, solving this relation system. Polyinvar improves on this by using the fixpoint iteration, sketched in figure 2. Basically, it is a semi-naive fixpoint iteration scheme, where only newly deduced relations are propagated through the control flow graph instead of all so far known valid relations at a program point. This leads to a significant improvement in calculation time, since each edge effect can be calculated on lesser amounts of polynomials.

As this analysis only verifies the validity of a given relation at a program point, we validate in Polyinvar a generic polynomial, whose coefficients are kept variable. Afterwards we can evaluate the precondition of the analysis to infer the sought relation.

An early version of Polyinvar was implemented based on polynomials and ideals literally following [1]. First tests showed however, that the complexity of the reductions on ideals indicated to be the bottleneck for the efficient inference of polynomial relations of degree higher than 1. Since this complexity crucially depends on the number of involved variables, we replaced the use of polynomials and ideals with vectors and modules. As indicated in [3], reductions on modules can be handled more efficiently than on ideals, so that the actual version of Polyinvar, using vectors and modules, is able to deduce even polynomial relations of degrees up to 4 in reasonable time.

Another idea to speed up the analysis, implemented in Polyinvar, was to check, how the reduced vector v' compares to the original vector v and then add the vector, that is more convenient to the module $G[t]$ instead of simply adding v. The measure, which vector would be more convenient which is actually used is based on the maximal degree of the polynomial components of the vector, as well the as its remaining non-null components. Tests in [2] have indicated, that neither simply adding the reduced vectors nor the original vectors provide the optimal behaviour. With our treatment we could achieve run-time improvements up to 50%.

3 Related work

Several related papers have considered the problem of inferring polynomial invariants.

In [4], Sankaranarayanan et al. propose an analysis of polynomial invariants based on templates for every program point. The templates represent parametric polynomial properties depending on unknown coefficients which then altogether are determined by means of Gröbner bases. The method allows to detect certain inductive invariants but is not complete. It is not clear, whether their method has been implemented so far.

In [5], Rodríguez-Carbonell and Kapur suggest an analysis of non-linear loop invariants by means of forward propagation and claim to have at least a partial implementation of their method. This analysis, however, seems to be applicable to restricted forms of assignments only. Also, opposed to backward propagation which we use, a fixpoint iteration based on forward propagation will not terminate in general for polynomial ideals without widening.

References

1. M. Müller-Olm, H. Seidl: Computing Polynomial Program Invariants, Information Processing Letters (IPL) **91** (2004) 233–244
2. M. Petter: Berechnung von polynomiellen Invarianten, Dipl. thesis, TU-München (2004)
3. T. Becker, V. Weispfenning: Gröbner Bases, Springer-Verlag (1993) 511 – 514
4. S. Sankaranarayanan, H. B. Sipma, Z. Manna: Non-linear Loop Invariant Generation using Gröbner Bases, ACM SIGPLAN Principles of Programming Languages (POPL 2004)
5. E. Rodríguez-Carbonell and D. Kapur: Program Verification Using Automatic Generation of Invariants, 1st International Colloquium on Theoretical Aspects of Computing 2004 (ICTAC'04), September 2004, Guiyang (China)