From Tests To Proofs

Heejung Kim
hjkim@ropas.snu.ac.kr

Aug 31, 2009

Reference

- "From Tests To Proofs"

  Ashutosh Gupta, Rupak Majumdar, and Andrey Rybalchenko

Contents

- **Motivation**
- **Key Definitions**
- **Constraint-based Invariant Generation**
- **Constraint Simplification**
  - Simplification from tests
  - Simplification from symbolic execution
- **Conclusion**

Motivation

- **What is the key to proving a program correctness?**
  - ➔ Program invariants
- **They often require explicit and expensive programmer annotations.**
  - ➔ Automatic inference of program invariants
- **This method generates a set of constraints from the program text.**
- **Its solution provides an inductive invariant proof of program correctness.**

Motivation

- **Approach**
  - Abstract interpretation based
  - Counterexample-guided abstraction refinement
  - Constraint-based
- **Each technique by itself often fails to verify programs.**
- **This paper uses the combination of these techniques.**

Comparison of invariant-based verification tools

File	State-of-the-art techniques				This paper
	INTERPROC	BLAST	INVGEN	INVGEN+Z3	
Seq	×	diverge	23s	1s	**0.5s**
Seq-z3	×	diverge	23s	9s	**0.5s**
Seq-len	×	diverge	T/O	T/O	**2.8s**
nested	×	**1.2s**	T/O	T/O	2.3s
svd(light)	×	50s	T/O	T/O	**14.2s**
heapsort	×	**3.4s**	T/O	T/O	13.3s
mergesort	×	**18s**	T/O	52s	170s
SpamAssassin-loop*	✓	22s	T/O	5s	**0.4s**
apache-get-tag*	×	5s	**0.4s**	10s	0.7s
sendmail-fromqp*	×	diverge	0.3s	5s	**0.3s**

Main idea

- **To scale the invariant generation engine by using static and dynamic information**
- **Step 1 (static information)**
  - Obtain invariant template map by techniques based on abstract interpretation
- **Step 2 (static information)**
  - The output of step 1 is used as an initial to support constraint based invariant generation
- **Step 3 (dynamic information)**
  - Collect dynamic information by executing the program

Main idea

- **Two approaches for dynamic information**
  - Direct approach
    : use program states to compute additional constraints
  - Symbolic approach
    : use symbolic execution to collect sets of states

Key definitions

Transition system

- $P = (X, \mathcal{L}, l_I, \mathcal{T}, l_\varepsilon)$
- X : a set of variables
- $\mathcal{L}$: a set of control loctions
- l_I : initial location, $l_I \in \mathcal{L}$
- l_ε : error location, $l_\varepsilon \in \mathcal{L}$
- $\mathcal{T}$: a set of transitions
- $\tau : (l, \rho, l')$, $\tau \in \mathcal{T}$, $l, l' \in \mathcal{L}$
- ρ : transition relation

Key definitions

Computation of the program *P*

- a sequence of pair $<\mathrm{l}_0, s_0>$, $<\mathrm{l}_1, s_1>$, ...
- $\mathrm{l}_0 = \mathrm{l}_I$, $\mathrm{l}_i \in \mathcal{L}$
- s_i : a valuation of the variables X, also called a state

A state s is reachable

- if $<\mathrm{l}, s>$ appears in some computation.

The program is safe

- if the error location l_ε does not appear in any computation

Key definitions

¤ Path of the program *P*

- a sequence of transitions
- $\pi = (l_0, \rho_0, l_1), (l_1, \rho_1, l_2), \ldots$
- $l_0 = l_I, l_i \in \mathcal{L}$
- ρ_i : transition relation

¤ Error path (or Counterexample path)

- A path that ends at the error location.

Remind main idea

- **Step 1 (static information)**
  - Obtain invariant template map by techniques based on abstract interpretation
- **Step 2 (static information)**
  - Step 1's output is used as an initial to support constraint based invariant generation
- **Step 3 (dynamic information)**
  - Collect dynamic information by executing the program

Constraint-based Invariant Generation

Basic algorithm

```
input
  P : program;  η :  invariant template map with parameters P
vars
  Ψ : static constraint
begin
  Ψ := InvGenSystem(P,η)

  /* algorithm for constraint simplification in here*/

  if  P* := Solve(Ψ) succeeds then
      return  "inductive invariant map η[P*/P]"
  else
      return  "no invariant map for given template"
end
```

Constraint-based Invariant Generation

A function InvGenSystem

- $\tau : (1, \rho, 1')$
- $\rho = (x \leq y \;\wedge\; x' = x + 1 \;\wedge\; y' = y)$
- $\phi = (\alpha + \alpha_x x + \alpha_y y \leq 0 \;\wedge\; \beta + \beta_x x + \beta_y y \leq 0)$ at location 1
- $\psi = (\gamma + \gamma_x x + \gamma_y y \leq 0)$ at location $1'$
- starting point : $\phi \;\wedge\; \rho \rightarrow \psi'$
- eliminate the prime : $\phi \;\wedge\; x \leq y \rightarrow \psi[x + 1 / x]$
- rewrite in the matrix form :

$$\begin{pmatrix} \alpha_x & \alpha_y \\ \beta_x & \beta_y \\ 1 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \leq \begin{pmatrix} -\alpha \\ -\beta \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} \gamma_x + 1 & \gamma_y \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \leq -\gamma$$

Constraint-based Invariant Generation

A function InvGenSystem

- obtain the constraint :

$$\exists \lambda \geq 0. \lambda \begin{pmatrix} \alpha_x & \alpha_y \\ \beta_x & \beta_y \\ 1 & -1 \end{pmatrix} = \left(\gamma_{x+1} \gamma_y \right) \wedge \lambda \begin{pmatrix} -\alpha \\ -\beta \\ 0 \end{pmatrix} \leq -\gamma$$

Constraint Simplification

- **Use additional dynamic information to restrict the search space.**

- **INVGEN + TEST : Simplification from tests**
  - Create additional constraints by using program executions.

- **INVGEN + SYMB : Simplification from symbolic execution**
  - Create additional constraints by performing symbolic execution along a collection of program paths.

INVGEN + TEST : Simplification from tests

```
input
  P : program;  η :  invariant template map with parameters P
vars
  Ψ : static constraint; Φ : dynamic constraint
begin
  Ψ := InvGenSystem(P,η)
  Φ := true
  repeat
    s_1, …., s_n := GenerateAndRunTest(P)
    if s_n (pc) = ⊥_ε then
      return "counterexample s_1, …., s_n"
    else
      Φ := Φ □ Λ_{i=1}^{n} (η.s_i(pc))[s_i / X]
  until no more tests
  if P* := Solve(Ψ, Φ) succeeds then
    return "inductive invariant map η[P*/P]"
  else
    return "no invariant map for given template"
end
```

INVGEN + TEST : Simplification from tests

- **An example about dynamic constraint**
  - $t(x,y) : \alpha x + \beta y + \gamma \leq 0$ at location 1
  - concrete state : $x = 35$, $y = -9$
  - obtain the constraint : $35\alpha - 9\beta + \gamma \leq 0$

INVGEN + SYMB : Simplification from symbolic execution

```
repeat
    π := GeneratePath(P)
    (* π_i = (ℓ_i, ρ_i, ℓ_{i+1}) for 1 ≤ i ≤ n *)
  if ℓ_{n+1} = ℓ_ε and π is feasible then
      return "counterexample π"
  else
      φ := (∃X. ρ_1 ∘ ... ∘ ρ_n)[X/X′]
      Φ := Φ □ Encode(φ → η.ℓ_{n+1})
until no more paths
```

INVGEN + SYMB : Simplification from symbolic execution

An example about dynamic constraint

- $t(x,y,z) : \alpha + \alpha_x x + \alpha_y y + \alpha_z z \leq 0$ □ $\beta + \beta_x x + \beta_y y + \beta_z z \leq 0$
- a set of states : $\phi = (-x \leq 0$ □ $-y \leq 0$ □ $x + y - z \leq 0\,)$
- the encoding of the implication $\phi \rightarrow t$ obtains the constraint :

$$\exists \Lambda \geq 0.\Lambda \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 1 & 1 & -1 \end{pmatrix} = \begin{pmatrix} \alpha_x & \alpha_y & \alpha_z \\ \beta_x & \beta_y & \beta_z \end{pmatrix} \wedge \Lambda \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \leq \begin{pmatrix} -\alpha \\ -\beta \end{pmatrix}$$

Conclusion

- **If Algorithm INVGEN+TEST or INVGEN+SYMB on input program *P* and invariant template map η returns**
  - (a) "counterexample s_1, … s_n"
    - there is an execution of the program that reaches the error location.
  - (b) "inductive invariant map η^*"
    - η^* is an invariant map for program P, and the program P is safe.
  - (c) "no invariants with template η"
    - *there* is no invariant map for program P with the given invariant template map η.

Conclusion

- **Relation between this paper and our corpus project**
  - What is the method that can use the dynamic information like this paper's approach?

Q&A

Thank you!