Dynamic Invariant Analysis

- Dynamic Discovery of Program Invariants
  - Execute Program on a set of inputs
  - Infer Invariants using obtained traces

- Useful in
  - Program Documentation
  - Refactoring
  - Debugging
  - Verification
Dynamic Invariant Analysis

- Daikon is widely used for such analysis
  - Supports only a limited subset of Linear Relations
  - No support for Nonlinear Relations
  - Limited support for Array Invariants

- Contribution from the Paper
  - Polynomial (Nonlinear) Invariants
  - Linear Array Invariants
    - Simple
    - Nested
Polynomial Invariants

- Polynomial Equalities
  - Solve using Linear Algebra

- Polynomial Inequalities
  - Use Polyhedra
  - Deduction from Loop Conditions
Terminology

- $V$ = Set of Instrumented Variables at a Location
- $D$ = Maximum Degree of Polynomial
- $T$ = Set of Terms over $V$ with Maximum degree $D$
Polynomial Equalities

```
x := a
i := 1
while (i <= n){
  // Inv: x = i * a
  x := x + a
  i := i + 1
}
```

- $V = \{x, i, a\}$
- $D = 2$
- $T = \{1, x, i, a, xi, xa, ia, x^2, i^2, a^2\}$
- Linear Equation Template:
  \[c_1 + c_2x + c_3i + c_4a + \ldots + c_{10}a^2 = 0\]
  instantiated per Trace
- Complexity of solving this Linear System is $O(|T|^3)$
Polynomial Inequalities

- $V = \{x, n, a\}$
- $D = 2$
- $T = \{1, x, n, a, xn, xa, na, x^2, n^2, a^2\}$
- Construct $|T|$ dimensional points from traces and build Bounded Convex Polyhedron that covers all trace points
- Boundary of Polyhedron satisfies $c_1 + c_2x + c_3n + c_4a + \ldots + c_{10}a^2 \geq 0$
- The Complexity of building Polyhedron with $k$ points in $n$ dimensions has upper bound $O(k^{\lfloor n/2 \rfloor})$
Deduction From Loop Conditions

```
x := a
i := 1
while (i <= n) {
  // Inv: x <= n * a
  x := x + a
  i := i + 1
}
```

- Combine inequalities at loop head with found equalites
- $x \leq n * a$ can be deduced from $i \leq n$ and $x = i * a$
- $O(|T|^3)$ Complexity but can only deduce Inequalities derivable from Loop Conditions and Found Equalities
Linear Array Invariants

- Simple Array Relations
  - $D = 1$
  - Relations Among Array Elements
  - Relations Among Array Indices

- Nested Array Relations
  - Reachability Analysis
  - Satisfiability Problem Formulation
  - Functions
Simple Array Relations

- Expand set $V$ of array variables to $V'$ representing elements of arrays in $V$
- Find Set of Linear Equalities $R$ between variables in $V'$ from traces of the form

\[
\begin{align*}
A_0 + b_0 B_{j_0} + c_0 &= 0 \\
A_1 + b_1 B_{j_1} + c_1 &= 0 \\
A_2 + b_2 B_{j_2} + c_2 &= 0 \\
&\quad \vdots \\
A_m + b_m B_{j_m} + c_m &= 0
\end{align*}
\]

A is pivot as $c_i, b_i$ and $j_i$ are expressed in terms of indices of $A$
Simple Array Relations

- \( b_i, c_i \) and \( j_i \) are linear relations ranging over indices of \( A \)
  \[
  A[i] = (p_0i + q_0)B[p_1i + q_1] + (p_2i + q_2)
  \]

- The Coefficients are determined using information from \( R \)

- The Complexity of the procedure is \( O(|V'|^3) \)
Nested Array Relations

- Reachability Analysis
- Satisfiability Problem Formulation
- Functions
Elements of $A$ reach elements of $C$ through elements of $B$

$$A[i] = B[C[pi + q]]$$

- Elements of $A$ are subset of elements of $B$
- Indices of $B$ are subset of elements of $C$
- The Time Complexity of Reachability Analysis is Exponential in Nesting Depth
Satisfiability Problem Formulation

- Encode finding Nested Array Relations into a CNF formula $f$
- We can pose

$$A[i] = B[C[pi + q]]$$

as a CNF formula $f$:

$$(1 = q) \land (2 = p + q \lor 3 = p + q) \land (5 = 2p + q)$$

- Use SMT Solver to find Solution of $f$
- Same Worst Case Complexity
- Improves Performance of Reachability Analysis compared to Original method
Functions

- Consider user defined functions
  \[ A[i] = f(C[i], g(D[i])) \]

- Consider a function \( f \) with \( n \) arguments as an \( n \) dimensional array \( F \)
  \[ F[i_1] \ldots [i_n] = f(i_1 \ldots i_n) \]

- Enforce finite depth in Nested Array Relations by disallowing a function to appear in scope of one of its arguments
Prototype tool *invgen* in python uses *Sage* mathematical environment and *Z3* as SMT solver

Available at https://code.google.com/p/invgen/

Evaluation on Nonlinear Arithmetic (NLA) test suite containing simple algorithms and an implementation of AES

Can find all the documented invariants for NLA test suite and 57% of the documented invariants for AES
NLA

- Cohencu
- Cohendiv
- Dijkstra
- Euclidex
- Fermat
- Freire
- LCM
- MannaDiv
- Sqrt
- Wensley

More Info about functions can be found [here](#)
AES

- AddRoundKey
- RotWord
- ShiftRows
- Block2State
- KeySetupEnc
- SubBytes
- Mul
- Xor
- SubBytes
- SubWord
Conclusion

Pros

- Extends current Dynamic Analysis Techniques
- Loop Invariant Inference
- Can be applied in Verification of Complex Numeric Algorithms

Cons

- May not scale well for large programs
- Effectiveness depends on quality of traces
- Does not consider certain forms of Array Invariants like

\[ A[i] = 2B[100C[\ldots]] \]