Automating Regression Verification

Dennis Felsing, Sarah Grebing, Vladimir Klebanov, Mattias Ulbrich, Philipp Rümmer

2014-07-23

Karlsruhe Institute of Technology
Introduction

How to prevent regressions in software development?
How to prevent regressions in software development?

Formal Verification

Formally prove correctness of software
⇒ Requires formal specification

Regression Testing

Discover new bugs by testing for them
⇒ Requires test cases
How to prevent regressions in software development?

**Formal Verification**

Formally prove correctness of software
⇒ Requires formal specification

**Regression Testing**

Discover new bugs by testing for them
⇒ Requires test cases

**Regression Verification**

Formally prove there are no new bugs
Regression Verification

Formally prove there are no new bugs

- Goal: Proving the equivalence of two closely related programs
- No formal specification or test cases required
- Instead use old program version as reference
- Tools for proving function equivalence in a simple programming language using SMT solvers
Overview

1. Overapproximation using Uninterpreted Functions

2. Approximation using Uninterpreted Predicates

3. Results and Future Work
Overview

1. Overapproximation using Uninterpreted Functions

2. Approximation using Uninterpreted Predicates

3. Results and Future Work
Function Equivalence

Existing approach by Strichman & Godlin

Function $f$
(val $n_1$; ret $r_1$)

Function $g$
(val $n_2$; ret $r_2$)

Uninterpreted Function $U$ for recursive calls in both $f$ and $g$

Function $f$ without recursion

Function $g$ without recursion

Static Single Assignment $S_f$

Static Single Assignment $S_g$

$(n_1 = n_2 \land S_f \land S_g) \rightarrow r_1 = r_2$

SMT Solver

Valid / Invalid
Our Contribution: Extensions

Function $f$
(val $n_1$; ret $r_1$)

Function $g$
(val $n_2$; ret $r_2$)

$S_f$

$S_g$

$(n_1 = n_2 \land S_f \land S_g) \rightarrow r_1 = r_2$

SMT Solver

Valid / Invalid
Our Contribution: Extensions

Function \( f \) (val \( n_1 \); ret \( r_1 \))

\[ S_f \]

Function \( g \) (val \( n_2 \); ret \( r_2 \))

\[ S_g \]

\[ (n_1 = n_2 \land S_f \land S_g) \rightarrow r_1 = r_2 \]

SMT Solver

Valid / Invalid

Equivalent!
Our Contribution: Extensions

Function $f$ (val $n_1$; ret $r_1$)

Function $g$ (val $n_2$; ret $r_2$)

$S_f$ Single Static Assignment Form $S_g$

$(n_1 = n_2 \land S_f \land S_g) \rightarrow r_1 = r_2$

SMT Solver

Valid / Invalid

Equivalent!

Counterexample:

$n = 0: \begin{align*}
    r_1 &= -1 \\
    r_2 &= -3
\end{align*}$
Our Contribution: Extensions

Function $f$ (val $n_1$; ret $r_1$)

Equivalent?

Function $g$ (val $n_2$; ret $r_2$)

$S_f\quad $Single Static Assignment Form\quad S_g$

$(n_1 = n_2 \land S_f \land S_g) \rightarrow r_1 = r_2$

SMT Solver

Valid / Invalid

Equivalent!

$S_f$

Execute

Counterexample:
$n = 0: \begin{cases} r_1 = -1 \\ r_2 = -3 \end{cases}$

$S_g$

Valid

Invalid

Equivalent!

$f(0) = g(0) = 0$
Our Contribution: Extensions

Function $f$ (val $n_1$; ret $r_1$) → Equivalent? → Function $g$ (val $n_2$; ret $r_2$)

$S_f$ → Single Static Assignment Form → $S_g$

$(n_1 = n_2 \land S_f \land S_g \land U(0) = 0) \rightarrow r_1 = r_2$

Valid / Invalid

rereun SMT Solver

Execute

f(0) = g(0) = 0

Counterexample: $n = 0$: $r_1 = -1$, $r_2 = -3$

Equivalent!
Overapproximation using uninterpreted functions

Approach

- Run the programs with input gathered from counterexamples
- Detect whether CE is spurious or not
- If spurious: Add additional constraints to the uninterpreted function

$\Rightarrow$ Is a simple form of Counter Example Guided Abstraction Refinement (CEGAR)

Successful when

- Finite number of constraints on the uninterpreted function imply equivalence
- These are often the “base cases” of recursive implementations
Overview

1. Overapproximation using Uninterpreted Functions

2. Approximation using Uninterpreted Predicates

3. Results and Future Work
Approximation using Uninterpreted Predicates

First approach (just shown)

- Overapproximate recursion by uninterpreted Function \( U \):

\[
\forall U. \text{constraints}(U) \land S_f \land S_g \land \ldots \rightarrow r_1 = r_2
\]

New approach

- Infer a predicate \( C \) which couples recursive calls:

\[
\exists C. (C(...) \land \ldots \rightarrow r_1 = r_2) \land \text{“C couples f and g”}
\]

- Use state-of-the-art SMT solvers (Eldarica, Z3) to automatically find such a \( C \) or prove that is does not exist

\( \Rightarrow \) Example will show loops with coupling loop invariants
int f1(int n) {
    int r = 0;
    if (n == 0) return 1;
    while (n > 0) {
        n /= 10; r++;
    }
    return r;
}
```c
int f1(int n) {
    int r = 0;
    if (n == 0) return 1;
    while (n > 0) {
        n /= 10; r++;
    }
    return r;
}

int f2(int n) {
    int r = 1;
    while (true) {
        if(n < 10) return r;
        if(n < 100) return r+1;
        if(n < 1000) return r+2;
        if(n < 10000) return r+3;
        n /= 10000;
        r += 4;
    }
}
```
Loop synchronisation

- To show: Equal input gives equal output
To show: Equal input gives equal output
Automatic Invariant Inference

Loop synchronisation

\[ f_1 = f_2 \]

- **To show:** Equal input gives equal output
- **Loops are synchronised**
Automatic Invariant Inference

Loop synchronisation

- To show: Equal input gives equal output
- Loops are synchronised
Loop synchronisation

- To show: Equal input gives equal output
- Loops are synchronised
- ... at least loosely synchronised
Automatic Invariant Inference

Loop synchronisation

\[
f_1 = C = f_2
\]

- **To show:** Equal input gives equal output
- Loops are **synchronised**
- ... at least loosely synchronised
Automatic Invariant Inference

Loop synchronisation

\[ f_1 = C \]

\[ f_2 = \]

To show: Equal input gives equal output

Loops are synchronised

... at least loosely synchronised

⇒ Use \( C \) as loop invariant for both programs.

(\( \rightarrow \)coupling invariant)
Automatic Invariant Inference

Loop synchronisation

\[ f_1 \equiv f_2 \]

- **To show**: Equal input gives equal output
- Loops are *synchronised*
- ... at least loosely synchronised

\[ C \]

\[ \Rightarrow \text{Use } C \text{ as loop invariant for both programs.} \]

\[ \rightarrow \text{coupling invariant} \]
Automatic Invariant Inference

Loop synchronisation

Let \( f_1 \) and \( f_2 \) be two programs.

- **To show:** Equal input gives equal output
- Loops are *synchronised*
- ... at least loosely synchronised

\[ \Rightarrow \text{Use } C \text{ as loop invariant for both programs.} \]

(\( \rightarrow \text{coupling invariant} \))
To show: Equal input gives equal output

Loops are synchronised

... at least loosely synchronised

⇒ Use $C$ as loop invariant for both programs.

($\rightarrow$ coupling invariant)
To show: Equal input gives equal output
• Loops are synchronised
• ... at least loosely synchronised
⇒ Use $C$ as loop invariant for both programs.
($\rightarrow$ coupling invariant)
Automatic Invariant Inference

Loop synchronisation

\[ f_1 = \cdots = f_2 \]

- To show: Equal input gives equal output
- Loops are synchronised
- ... at least loosely synchronised

\[ \Rightarrow \text{Use } C \text{ as loop invariant for both programs.} \]

(\( \rightarrow \text{coupling invariant} \))

Automatic Regression Verification:

Do not specify \( C \) but infer it automatically.
Automatic Invariant Inference

Three cases to consider:

1. Initially coupling loop invariant $C$ holds
2. After both loop steps (or one if other finished), $C$ holds
3. After both loops finished, $C$ implies equality of results
Automatic Invariant Inference

Three cases to consider:

1. Initially coupling loop invariant $C$ holds
2. After both loop steps (or one if other finished), $C$ holds
3. After both loops finished, $C$ implies equality of results

Automatically inferred coupling loop invariant:
(Using Eldarica)

$$(n_1 > 0 \rightarrow (n_1 = n_2 \land r_1 + 1 = r_2))$$
\& ($$n_2 \leq 0 \rightarrow return_2 = r_1$$)
\& $n_1 \geq n_2$$
Automatic Invariant Inference

Three cases to consider:

1. Initially coupling loop invariant $C$ holds
2. After both loop steps (or one if other finished), $C$ holds
3. After both loops finished, $C$ implies equality of results

Automatically inferred coupling loop invariant: 
(Using Eldarica)

\[
(n_1 > 0 \rightarrow (n_1 = n_2 \land r_1 + 1 = r_2)) \\
\land (n_2 \leq 0 \rightarrow return_2 = r_1) \\
\land n_1 \geq n_2
\]

- Compare to loop invariant: \( n = \frac{n_0}{10^r} \)
- Coupling invariant is not trivial, but linear and inferable!
Overview

1. Overapproximation using Uninterpreted Functions

2. Approximation using Uninterpreted Predicates

3. Results and Future Work
Evaluation and Results

Approaches implemented for a subset of C: simplRV, Rêve
Usable with webinterface: http://formal.iti.kit.edu/improve/deduktionstreffen2014/

Rêve evaluation (uninterpreted predicates)

- 32 short benchmarks of integer programs (10-50 lines)
- Collected from literature
- Good performance on most equivalent programs
- Finds counterexample for non-equivalent programs as well
Conclusion

Regression Verification

- Initial approach limited to strongly coupled recursions or user feedback
- Automatic Invariant Inference: More powerful, using recent techniques in SMT solvers like Eldarica and Z3

Future Work

- More examples (larger)
- Support arrays, heaps, objects