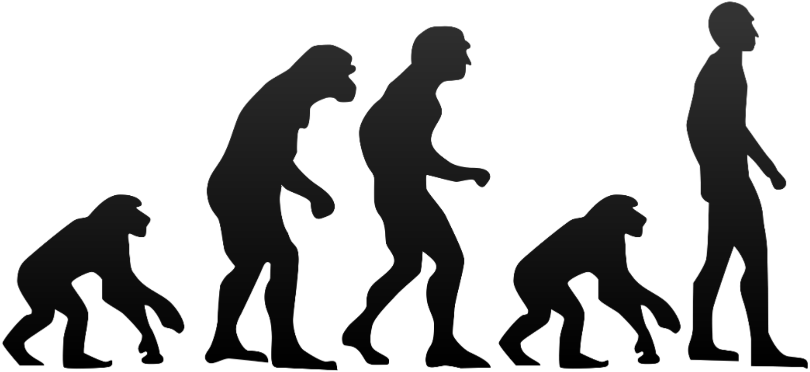
Automating Regression Verification

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

2014-07-23

Karlsruhe Institute of Technology

How to prevent regressions in software development?



How to prevent regressions in software development?

Formal Verification

Formally prove correctness of software
$\Rightarrow$ Requires formal specification

Regression Testing

Discover new bugs by testing for them
$\Rightarrow$ Requires test cases

How to prevent regressions in software development?

Formal Verification

Formally prove correctness of software
$\Rightarrow$ Requires formal specification

Regression Testing

Discover new bugs by testing for them
$\Rightarrow$ 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

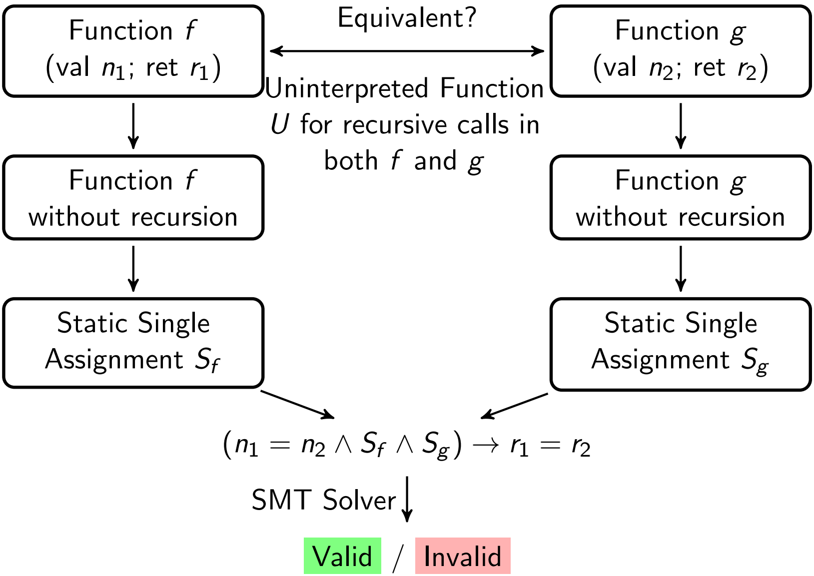
1. Overapproximation using Uninterpreted Functions

2. Approximation using Uninterpreted Predicates

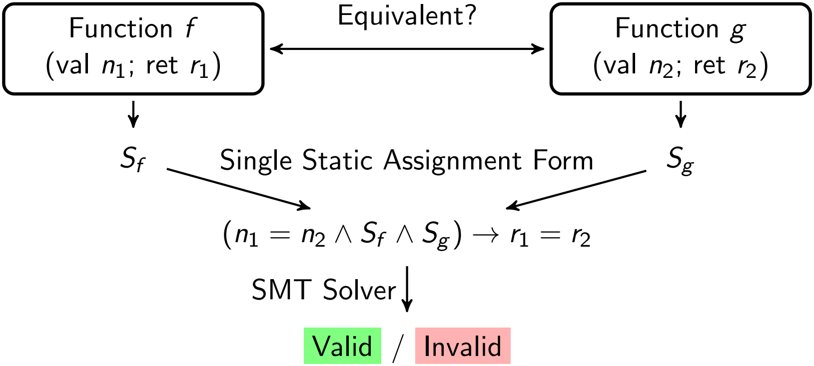
3. Results and Future Work

Function Equivalence

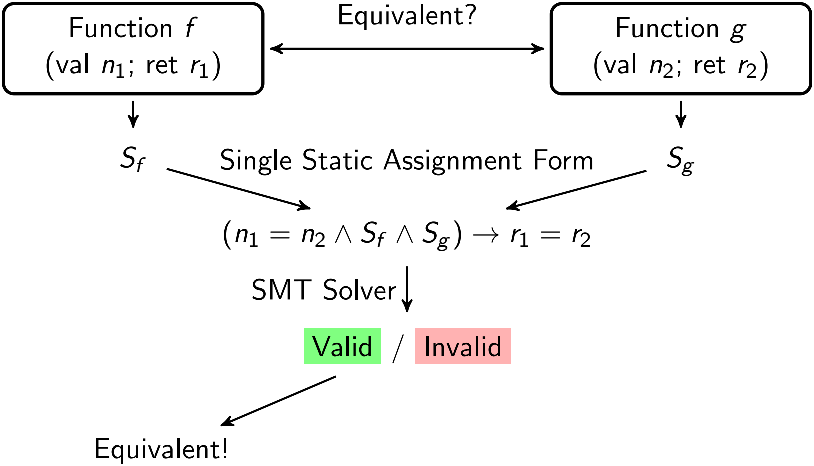
Existing approach by Strichman & Godlin



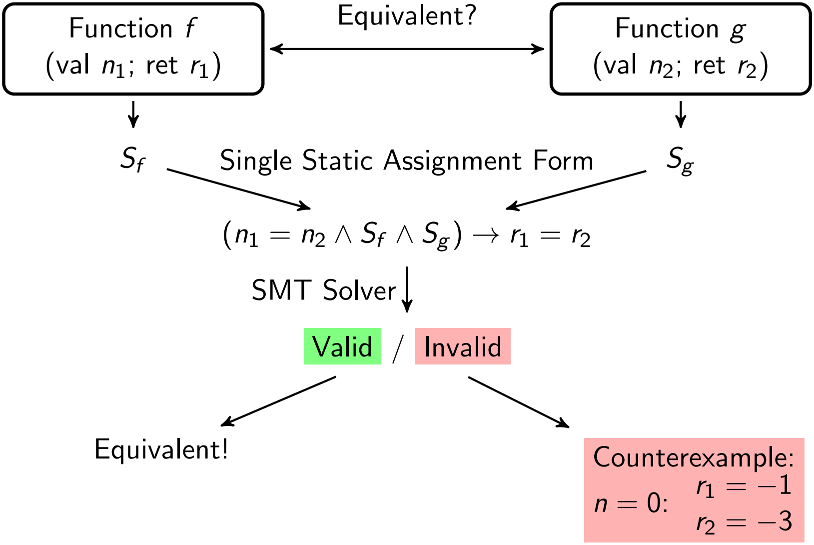
Our Contribution: Extensions



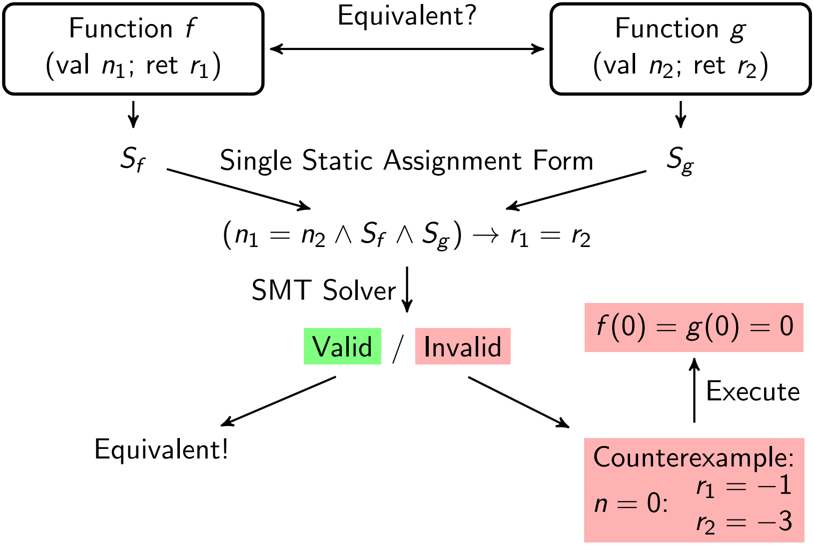
Our Contribution: Extensions



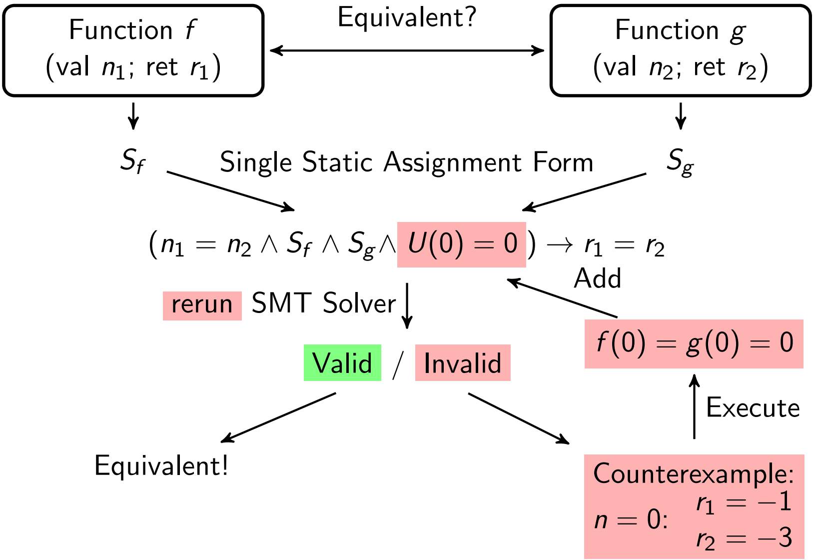
Our Contribution: Extensions



Our Contribution: Extensions



Our Contribution: Extensions



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:

$$\forall U.constraints(U) \land S_f \land S_g \land \ldots \rightarrow r_1 = r_2$$

New approach

- Infer a predicate C which couples recursive calls:

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

- 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

Automatic Invariant Inference

```
int f1(int n) {
  int r = 0;
  if (n == 0) return 1;
  while (n > 0) {
    n /= 10; r++;



  }
  return r;

}
```

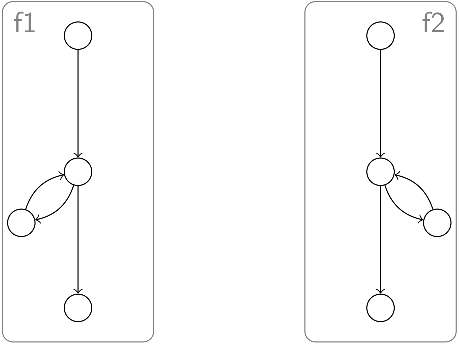
Automatic Invariant Inference

```
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;
  }
}
```

Automatic Invariant Inference

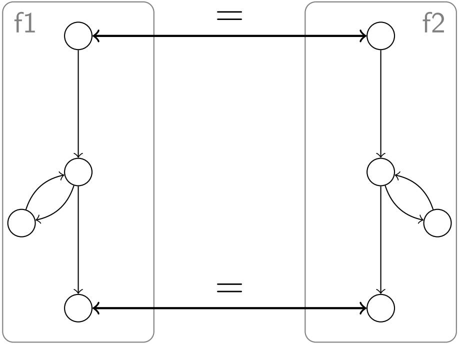
Loop synchronisation



- **To show:** Equal input gives equal output

Automatic Invariant Inference

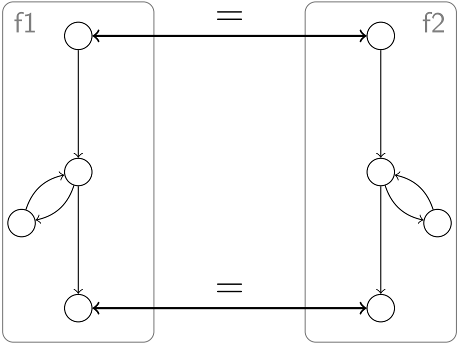
Loop synchronisation



- **To show:** Equal input gives equal output

Automatic Invariant Inference

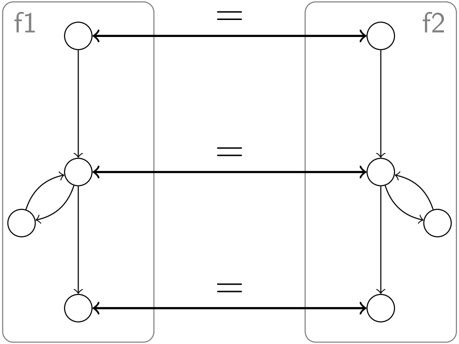
Loop synchronisation



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

Automatic Invariant Inference

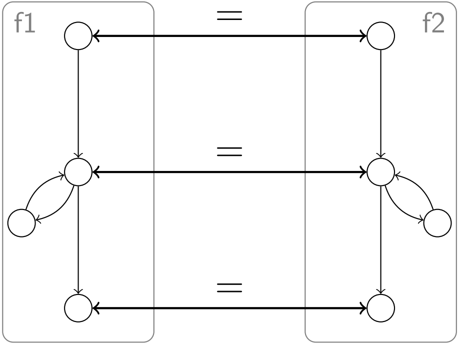
Loop synchronisation



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

Automatic Invariant Inference

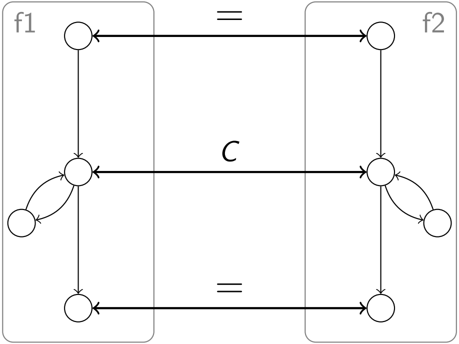
Loop synchronisation



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

Automatic Invariant Inference

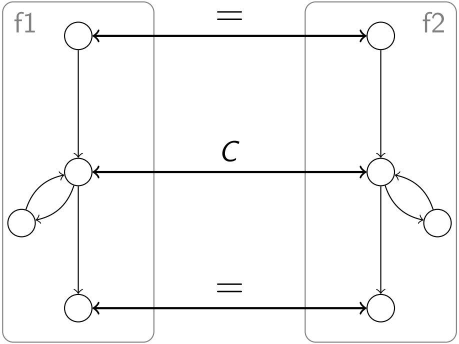
Loop synchronisation



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

Automatic Invariant Inference

Loop synchronisation



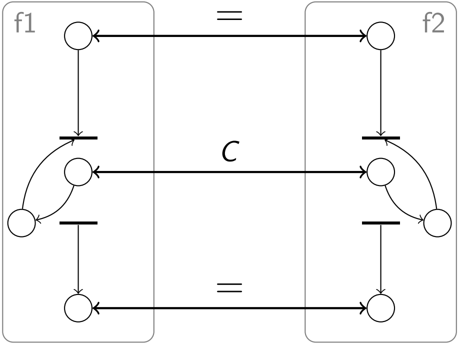
- **To show:** Equal input gives equal output
- Loops are **synchronised**
- ... at least loosely synchronised

$\Rightarrow$ Use C as **loop invariant** for **both** programs.

($\rightarrow$*coupling invariant*)

Automatic Invariant Inference

Loop synchronisation



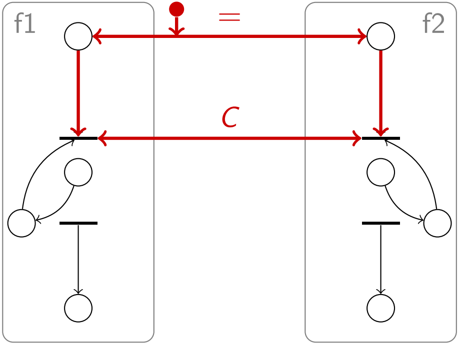
- **To show:** Equal input gives equal output
- Loops are **synchronised**
- ... at least loosely synchronised

$\Rightarrow$ Use C as **loop invariant** for **both** programs.

($\rightarrow$*coupling invariant*)

Automatic Invariant Inference

Loop synchronisation



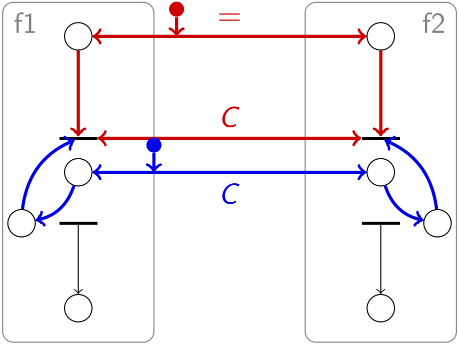
- **To show:** Equal input gives equal output
- Loops are **synchronised**
- ... at least loosely synchronised

$\Rightarrow$ Use C as **loop invariant** for **both** programs.

(→*coupling invariant*)

Automatic Invariant Inference

Loop synchronisation



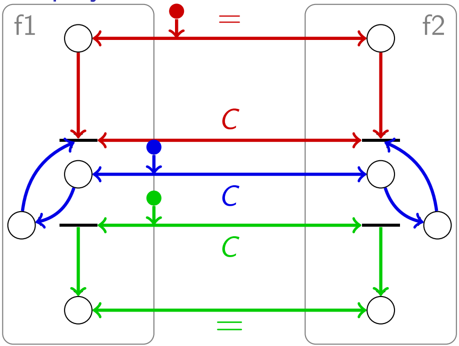
- **To show:** Equal input gives equal output
- Loops are **synchronised**
- ... at least loosely synchronised

$\Rightarrow$ Use C as **loop invariant** for **both** programs.

($\rightarrow$*coupling invariant*)

Automatic Invariant Inference

Loop synchronisation



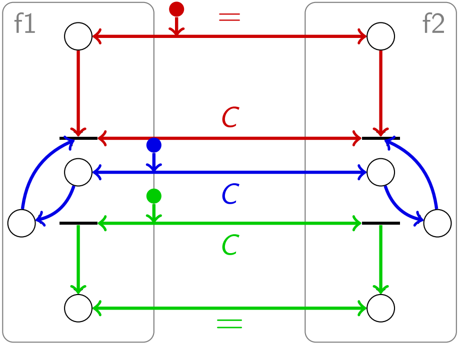
- **To show:** Equal input gives equal output
- Loops are **synchronised**
- ... at least loosely synchronised

$\Rightarrow$ Use C as **loop invariant** for **both** programs.

(→*coupling invariant*)

Automatic Invariant Inference

Loop synchronisation



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

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

(→*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)

$$
\begin{aligned}
&(n_1 > 0 \rightarrow (n_1 = n_2 \wedge r_1 + 1 = r_2)) \\
\wedge &(n_2 \leq 0 \rightarrow \mathit{return}_2 = r_1) \\
\wedge &n_1 \geq n_2
\end{aligned}
$$

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)

$$\begin{aligned} &(n_1 > 0 \rightarrow (n_1 = n_2 \wedge r_1 + 1 = r_2)) \\ \wedge &(n_2 \leq 0 \rightarrow \mathit{return}_2 = r_1) \\ \wedge &n_1 \geq n_2 \end{aligned}$$

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

Overview

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