Regression Verification: Project Proposal

Presentation by Dennis Felsing within the Projektgruppe Formale Methoden der Softwareentwicklung

SS 2013
Introduction

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
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

Overview

Function $f$ (val $n$; ret $r$)

Function $g$ (val $x$; ret $y$)

Equivalent?

Uninterpreted Functions

Function $f$ without recursions

Function $g$ without recursions

Static Single Assignment $S_f$

Static Single Assignment $S_g$

$(n = x \land S_f \land S_g) \rightarrow r = y$

SMT Solver

Valid / Invalid
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
- Make use of similarity between programs
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
- Make use of similarity between programs

```c
int f(int n) {
    int r = 0;
    if (n <= 0) {
        r = n;
    } else {
        r = n + f(n - 1);
    }
    return r;
}

int g(int x) {
    int y = 0;
    if (x <= 1) {
        y = x;
    } else {
        y = x + g(x - 1);
    }
    return y;
}
```
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
- Make use of similarity between programs

```c
int f(int n) {
    int r = 0;
    if (n <= 0) {
        r = n;
    } else {
        r = n + f(n - 1);
    }
    return r;
}
```

```c
int g(int x) {
    int y = 0;
    if (x <= 1) {
        y = x;
    } else {
        y = x + g(x - 1);
    }
    return y;
}
```
Uninterpreted Functions

Overview

Function $f$ (val $n$; ret $r$)

Uninterpreted Functions

Function $g$ (val $x$; ret $y$)

Equivalent?

Function $f$ without recursions

Static Single Assignment $S_f$

(n = $x \land S_f \land S_g$) $\rightarrow$ $r = y$

Static Single Assignment $S_g$

SMT Solver

Valid / Invalid
Uninterpreted Functions

- Given the same inputs an **Uninterpreted Function** always returns the same outputs.
- Motivation: Proof by Induction, to prove $f(n) = g(n)$ assume $f(n - 1) = g(n - 1)$

```c
int f(int n) {
    int r = 0;
    if (n <= 0) {
        r = n;
    } else {
        r = n + U(n - 1);
    }
    return r;
}
```  

```c
int g(int x) {
    int y = 0;
    if (x <= 1) {
        y = x;
    } else {
        y = x + U(x - 1);
    }
    return y;
}
```
Static Single Assignment

Overview

Function $f$ (val $n$; ret $r$) → Equivalent?

Function $g$ (val $x$; ret $y$)

Uninterpreted Functions

Function $f$ without recursions

Function $g$ without recursions

Static Single Assignment $S_f$

Static Single Assignment $S_g$

$(n = x \land S_f \land S_g) \rightarrow r = y$

SMT Solver

Valid / Invalid
Static Single Assignment

- Translate program functions to formulas
- Recursions: Abstraction by Uninterpreted Function
- In assignments \( x = exp \) replace \( x \) with a new variable \( x_1 \)
- Represents the states of the program
Static Single Assignment

- Translate program functions to formulas
- Recursions: Abstraction by Uninterpreted Function
- In assignments $x = \text{exp}$ replace $x$ with a new variable $x_1$
- Represents the states of the program

```c
int f(int n) {
    int r = 0;
    if (n ≤ 0) {
        r = n;
    } else {
        r = n + U(n - 1);
    }
    return r;
}
```

$$S_f = \begin{pmatrix} r_0 = 0 \end{pmatrix}$$
Static Single Assignment

- Translate program functions to formulas
- Recursions: Abstraction by Uninterpreted Function
- In assignments $x = \text{exp}$ replace $x$ with a new variable $x_1$
- Represents the states of the program

```c
int f(int n) {
    int r = 0;
    if (n <= 0) {
        r = n;
    } else {
        r = n + U(n - 1);
    }
    return r;
}
```

$$S_f = \begin{cases} 
  r_0 = 0 \\
  n \leq 0 \rightarrow r_1 = n 
\end{cases}$$
Static Single Assignment

- Translate program functions to formulas
- Recursions: Abstraction by Uninterpreted Function
- In assignments $x = exp$ replace $x$ with a new variable $x_1$
- Represents the states of the program

```c
int f(int n) {
    int r = 0;
    if (n <= 0) {
        r = n;
    } else {
        r = n + U(n - 1);
    }
    return r;
}
```

$$S_f = \begin{cases} r_0 = 0 \\
    n \leq 0 \rightarrow r_1 = n \\
    n > 0 \rightarrow r_1 = n + U(n - 1) \end{cases}$$
Static Single Assignment

- Translate program functions to formulas
- Recursions: Abstraction by Uninterpreted Function
- In assignments $x = exp$ replace $x$ with a new variable $x_1$
- Represents the states of the program

```c
int f(int n) {
    int r = 0;
    if (n <= 0) {
        r = n;
    }
    else {
        r = n + U(n - 1);
    }
    return r;
}
```

$$S_f = \begin{cases} 
    r_0 = 0 \\ 
    n \leq 0 \rightarrow r_1 = n \\ 
    n > 0 \rightarrow r_1 = n + U(n - 1) \\ 
    r = r_1 
\end{cases}$$
Function $f (\text{val } n; \text{ret } r)$

Function $g (\text{val } x; \text{ret } y)$

Equivalent?

Uninterpreted Functions

Function $f$ without recursions

Function $g$ without recursions

Static Single Assignment $S_f$

Static Single Assignment $S_g$

$(n = x \land S_f \land S_g) \rightarrow r = y$

Equal inputs

Equal outputs
SMT Solver
Overview

Function $f$ (val $n$; ret $r$)

Function $g$ (val $x$; ret $y$)

Uninterpreted Functions

Function $f$ without recursions

Function $g$ without recursions

Static Single Assignment $S_f$

Static Single Assignment $S_g$

$(n = x \land S_f \land S_g) \rightarrow r = y$

SMT Solver

Valid / Invalid
Extensions

- SMT solver still complains:

\[ f(n) = \begin{cases} 
& -1 \quad \text{if } n = 0 \\
& g(n) \quad \text{otherwise}
\end{cases} \]
Extensions

- SMT solver still complains:

\[
f(n) = \begin{cases} 
-1 & \text{if } n = 0 \\
g(n) & \text{otherwise}
\end{cases}
\]

```c
int f(int n) {
    int r = 0;
    if (n <= 0) {
        r = n;
    } else {
        r = n + f(n - 1);
    }
    return r;
}
```

```c
int g(int x) {
    int y = 0;
    if (x <= 1) {
        y = x;
    } else {
        y = x + g(x - 1);
    }
    return y;
}
```
• SMT solver still complains:

\[ f(n) = \begin{cases} 
-1 & \text{if } n = 0 \\
g(n) & \text{otherwise}
\end{cases} \]

```c
int f(int n) {
    int r = 0;
    if (n \leq 0) {
        r = n;
    } else {
        r = n + f(n - 1);
    }
    return r;
}
```

```c
int g(int x) {
    int y = 0;
    if (x \leq 1) {
        y = x;
    } else {
        y = x + g(x - 1);
    }
    return y;
}
```

• But we can fix it:

\[ f(0) = 0 \]
Finding Counter Examples

Function $f$ (val $n$; ret $r$)  
Function $f$ without recursions  
Static Single Assignment $S_f$

Function $g$ (val $x$; ret $y$)  
Function $g$ without recursions  
Static Single Assignment $S_g$

$(n = x \land S_f \land S_g) \rightarrow r = y$

SMT Solver  
Valid / Invalid  
Execute  
Counter Example
Extensions
Determining Corner Cases

Function $f$ (val $n$; ret $r$)

Function $g$ (val $x$; ret $y$)

Function $f$ without recursions

Function $g$ without recursions

Static Single Assignment $S_f$

Static Single Assignment $S_g$

Uninterpreted Functions

Equivalent?

$(n = x \land S_f \land S_g \land U(0) = 0) \rightarrow r = y$

Add

SMT Solver

Valid / Invalid

Execute

Counter Example
Extensions

Functional Condition Extraction

Function $f$ (val $n$; ret $r$)

Function $g$ (val $x$; ret $y$)

Function $f$ without recursions

Function $g$ without recursions

Static Single Assignment $S_f$

Static Single Assignment $S_g$

Functional Condition

$(n = x \land S_f \land S_g \land \alpha) \rightarrow r = y$

SMT Solver

Valid / Invalid
Extensions

Relational Equivalence

Function $f$ (val $n$; ret $r$)

Function $f$ without recursions

Static Single Assignment $S_f$

Function $g$ (val $x$; ret $y$)

Function $g$ without recursions

Static Single Assignment $S_g$

$(n \geq 0 \land n = x \land S_f \land S_g) \rightarrow r \sim y$

SMT Solver

Valid / Invalid
Example Catalog

- Collect examples: Papers, Refactoring Rules, ...
- 51 program pairs so far
- Test how well approach and extensions work
Regression Verification

- Better chance of being adopted than Formal Verification
- More powerful than Regression Testing
- Extensions to cover more cases
- Example Catalog for evaluation