Finding bugs from failed verification attempts

Christoph Gladisch

University of Koblenz

May 20th, 2009
Related Work from the KeY group

- Verification-based Testing (Engel, Hähnle)
  \[(\text{pathCond}_0 \implies \{ U_0 \} \langle p_0 \rangle \text{post}) \ldots (\text{pathCond}_n \implies \{ U_n \} \langle p_n \rangle \text{post})\]

- White-box testing through black-box testing + specification mining (Beckert, Gladisch)

- Improving specifications for testing (Bender, Gladisch)

- Disproving/Disverification (Rümmer)
  \[\exists S. \{ S \} \neg (\text{pre} \implies \langle p \rangle \text{post})\]

- Symbolic Debugger (Baum, Hähnle, Rothe)

- Tool chain KeYGenU for regression unit testing (Gladisch, Tyshberowicz, Beckert, Yehudai)
  \[PUT \rightarrow \text{KeY} \rightarrow U \rightarrow \text{GenUTest} \rightarrow U'\]

- Full feasible branch coverage
  \[\text{CBP}: \text{pre} \land \{ M := M_{sk} \} (\text{post} \land \phi)\]
  \[\text{Strength cond.}: \text{pre} \land \{ M := M_{sk} \} \text{post} \implies \langle p \rangle M = \forall M_{sk}\]
Related Work from the KeY group

- Verification-based Testing (Engel, Hähnle)
  \((\text{pathCond}_0 \implies \{U_0\}\langle p_0\rangle\text{post}) \ldots (\text{pathCond}_n \implies \{U_n\}\langle p_n\rangle\text{post})\)

- White-box testing through black-box testing + specification mining (Beckert, Gladisch)

- Improving specifications for testing (Bender, Gladisch)

- Disproving/Disverification (Rümmer)
  \(\exists S.\{S\}\neg (\text{pre} \implies \langle p\rangle\text{post})\)

- Symbolic Debugger (Baum, Hähnle, Rothe)

- Tool chain KeYGenU for regression unit testing (Gladisch, Tyshberowicz, Beckert, Yehudai)
  \(\text{PUT} \rightarrow \text{KeY} \rightarrow U \rightarrow \text{GenUTest} \rightarrow U'\)

- Full feasible branch coverage
  \(\text{CBP: } \text{pre} \land \{M := M_{sk}\}(\text{post} \land \phi)\)
  \(\text{Strength cond.: } \text{pre} \land \{M := M_{sk}\}\text{post} \rightarrow \langle p\rangle M = \forall M_{sk}\)
Related Work from the KeY group

- Verification-based Testing (Engel, Hähnle)
  
  \[
  (\text{pathCond}_0 \implies \{U_0\}(p_0)\text{post}) \ldots (\text{pathCond}_n \implies \{U_n\}(p_n)\text{post})
  \]

- White-box testing through black-box testing + specification mining
  (Beckert, Gladisch)

- Improving specifications for testing (Bender, Gladisch)

- Disproving/Disverification (Rümmer)
  \[
  \exists S. \{S\} \neg (\text{pre} \implies \langle p \rangle \text{post})
  \]

- Symbolic Debugger (Baum, Hähnle, Rothe)

- Tool chain KeYGenU for regression unit testing (Gladisch, Tyshberowicz, Beckert, Yehudai)
  \[
  PUT \rightarrow \text{KeY} \rightarrow U \rightarrow \text{GenUTest} \rightarrow U'
  \]

- Full feasible branch coverage
  \[
  \text{CBP: } \text{pre} \land \{M := M_{sk}\}(\text{post} \land \phi)
  \]
  \[
  \text{Strength cond.: } \text{pre} \land \{M := M_{sk}\}\text{post} \rightarrow \langle p \rangle M = \forall M_{sk}
  \]

Finding bugs from failed verification attempts: May 20th, 2009
Related Work from the KeY group

- Verification-based Testing (Engel, Hähnle)
  \[ \text{pathCond}.0 \Rightarrow \{U_0\}\langle p_0\rangle\text{post} \ldots \text{pathCond}.n \Rightarrow \{U_n\}\langle p_n\rangle\text{post} \]
- White-box testing through black-box testing + specification mining (Beckert, Gladisch)
- Improving specifications for testing (Bender, Gladisch)
- Disproving/Disverification (Rümmer)
  \[ \exists S.\{S\}\neg (\text{pre} \rightarrow \langle p \rangle\text{post}) \]
- Symbolic Debugger (Baum, Hähnle, Rothe)
- Tool chain KeYGenU for regression unit testing (Gladisch, Tyshberowicz, Beckert, Yehudai)
  \[ \text{PUT} \rightarrow \text{KeY} \rightarrow U \rightarrow \text{GenUTest} \rightarrow U' \]
- Full feasible branch coverage
  \[ \text{CBP}: \text{pre} \land \{M := M_{sk}\}(\text{post} \land \phi) \]
  Strength cond.: \[ \text{pre} \land \{M := M_{sk}\}\text{post} \rightarrow \langle p \rangle M = \forall M_{sk} \]

Finding bugs from failed verification attempts: May 20th, 2009
Related Work from the KeY group

- Verification-based Testing (Engel, Hähnle)
  \[(\text{pathCond}_0 \Rightarrow \{U_0\}\langle p_0\rangle post) \ldots (\text{pathCond}_n \Rightarrow \{U_n\}\langle p_n\rangle post)\]

- White-box testing through black-box testing + specification mining
  (Beckert, Gladisch)

- Improving specifications for testing (Bender, Gladisch)

- Disproving/Disverification (Rümmer)
  \[\exists S.\{S\}\neg (\text{pre} \rightarrow \langle p \rangle \text{post})\]

- Symbolic Debugger (Baum, Hähnle, Rothe)

- Tool chain KeYGenU for regression unit testing (Gladisch, Tyshberowicz, Beckert, Yehudai)
  \[\text{PUT} \rightarrow \text{KeY} \rightarrow U \rightarrow \text{GenUTest} \rightarrow U'\]

- Full feasible branch coverage
  CBP: \[\text{pre} \land \{M := M_{sk}\}(\text{post} \land \phi)\]
  Strength cond.: \[\text{pre} \land \{M := M_{sk}\}\text{post} \rightarrow \langle p \rangle M = \forall M_{sk}\]

Finding bugs from failed verification attempts: May 20th, 2009
Related Work from the KeY group

- Verification-based Testing (Engel, Hähnle)
  \[ (\text{pathCond}.0 \Rightarrow \{U_0\}\langle p_0 \rangle\text{post}) \ldots (\text{pathCond}.n \Rightarrow \{U_n\}\langle p_n \rangle\text{post}) \]

- White-box testing through black-box testing + specification mining (Beckert, Gladisch)

- Improving specifications for testing (Bender, Gladisch)

- Disproving/Disveriﬁcation (Rümmer)
  \[ \exists S. \{S\} \neg (\text{pre} \rightarrow \langle p \rangle\text{post}) \]

- Symbolic Debugger (Baum, Hähnle, Rothe)

- Tool chain KeYGenU for regression unit testing (Gladisch, Tyshberowicz, Beckert, Yehudai)
  \[ PUT \rightarrow \text{KeY} \rightarrow U \rightarrow \text{GenUTest} \rightarrow U' \]

- Full feasible branch coverage
  \[ \text{CBP: } \text{pre} \land \{M := M_{sk}\}(\text{post} \land \phi) \]
  \[ \text{Strength cond.: } \text{pre} \land \{M := M_{sk}\}\text{post} \rightarrow \langle p \rangle M = \forall M_{sk} \]
Related Work from the KeY group

- Verification-based Testing (Engel, Hähnle)
  \[(\text{pathCond}.0 \Rightarrow \{U_0\}⟨p0⟩post) \ldots (\text{pathCond}.n \Rightarrow \{U_n\}⟨pn⟩post)\]

- White-box testing through black-box testing + specification mining (Beckert, Gladisch)

- Improving specifications for testing (Bender, Gladisch)

- Disproving/Disverification (Rümmer)
  \[\exists S.\{S\}\neg(pre \rightarrow ⟨p⟩post)\]

- Symbolic Debugger (Baum, Hähnle, Rothe)

- Tool chain KeYGenU for regression unit testing (Gladisch, Tyshberowicz, Beckert, Yehudai)
  \(PUT \rightarrow KeY \rightarrow U \rightarrow \text{GenUTest} \rightarrow U'\)

- Full feasible branch coverage
  \[\text{CBP: } pre \land \{M := M_{sk}\}(post \land \phi)\]
  \[\text{Strength cond.: } pre \land \{M := M_{sk}\}post \rightarrow ⟨p⟩M = \forall M_{sk}\]
Introduction

Goal
Combine verification and bug detection in a very unified way

Core ideas
- Purely symbolic
- Reuse a maximum of information from a failed proof attempt
- Search bugs on open proof branches

Problem/Challenge
- Falsifiable branches do not imply program error
- Contract rules (more general: abstraction techniques)
Introduction

Goal
Combine verification and bug detection in a very unified way

Core ideas
- Purely symbolic
- Reuse a maximum of information from a failed proof attempt
- Search bugs on open proof branches

Problem/Challenge
- Falsifiable branches do not imply program error
- Contract rules (more general: abstraction techniques)
## Introduction

### Goal
Combine verification and bug detection in a very unified way

### Core ideas
- Purely symbolic
- Reuse a maximum of information from a failed proof attempt
- Search bugs on open proof branches

### Problem/Challenge
- Falsifiable branches do not imply program error
- Contract rules (more general: abstraction techniques)
Method Contract Rule \[(pre_m, post_m, M, [])\] \(M = mod(p)\)

1: \(\Gamma \Rightarrow \{U\}pre_m, \Delta\)
2: \(\Gamma \Rightarrow \{U\}(pre_m \rightarrow [m()]post_m), \Delta\)
3: \(\Gamma \Rightarrow \{U\}{M}(post_m \rightarrow post), \Delta\)
\(\Gamma \Rightarrow \{U\}[m()]post, \Delta\)

Loop Invariant Rule \[(I, I \land \neg c, M, [])\] \(M = mod(b)\)

1: \(\Gamma \Rightarrow \{U\}I, \Delta\)
2: \(\Gamma \Rightarrow \{U\}{M}(I \land c \rightarrow [b]I), \Delta\)
3: \(\Gamma \Rightarrow \{U\}{M}((I \land \neg c) \rightarrow post), \Delta\)
\(\Gamma \Rightarrow \{U\}[while(c){b;}]post, \Delta\)
Approach

1. Try to verify program
2. Improve contract until 1st and 2nd branch of contract rule is proved
3. Check falsifiability of 3rd branch
4. prove SFP, “some Strange and Funny Property”

$$(((\{M^1 := M^2\}S_n) \land \{U\}{M^2}\text{post}) \rightarrow S_n$$
Approach

1. Try to verify program
2. Improve contract until 1st and 2nd branch of contract rule is proved
3. Check falsifiability of 3rd branch
4. Prove SFP, “some Strange and Funny Property”

\[(\{M^1 := M^2\} S_n) \land \{U\}\{M^2\} post) \rightarrow S_n\]
Approach

1. Try to verify program
2. Improve contract until 1st and 2nd branch of contract rule is proved
3. Check falsifiability of 3rd branch
4. prove SFP, “some Strange and Funny Property”

\[((\{M^1 := M^2\} S_n) \land \{U\}{M^2}\text{post}) \rightarrow S_n\]
Approach

1. Try to verify program
2. Improve contract until 1st and 2nd branch of contract rule is proved
3. Check falsifiability of 3rd branch
4. prove **SFP**, “*some Strange and Funny Property*”

\[
((\{ M^1 := M^2 \} S_n) \land \{ U \} \{ M^2 \} post) \to S_n
\]
Example 1

```java
/*@ public normal_behavior
 requires x>=0;
 ensures \result*\result<=x && (\result+1)*(\result+1)>x;
 diverges true;
@*/

public int sqrtA(int x){
    int i=0;
    //@ loop_invariant i*i<=x+1; modifies i;@*/
    while(i*i<=x){i++;}
    return i;
}
```

Finding bugs from failed verification attempts: May 20th, 2009
Falsifiability Preservation Analysis

1. Try to verify program
2. Improve contract until 1st and 2nd branch of contract rule is proved
3. Check falsifiability of 3rd branch
4. prove SFP, "some Strange and Funny Property"

\[
((\{M^1 := M^2\} S_n) \land \{U\}\{M^2\} post) \rightarrow S_n
\]
Falsifiability Preservation Analysis

1. Try to verify program
2. Improve contract until 1st and 2nd branch of contract rule is proved
3. Check falsifiability of 3rd branch
4. prove \textbf{SFP}, \textbf{S}pecial \textbf{F}alsifiability \textbf{P}reservation

\[ (\{M^1 := M^2\} S_n) \land \{U\}{\{M^2\} post} \rightarrow S_n \]
Falsifiability Preservation Analysis

Falsifiability Preservation of a Rule

\[
\frac{\Gamma_0 \Rightarrow \Delta_0 \ldots \Gamma_n \Rightarrow \Delta_n}{\Gamma \Rightarrow \Delta}
\]

Soundness: \( \Gamma_0 \Rightarrow \Delta_0 \ldots \Gamma_n \Rightarrow \Delta_n \vdash \Gamma \Rightarrow \Delta \)

Falsifiability preservation: Soundness: \( \Gamma \Rightarrow \Delta \vdash \Gamma_0 \Rightarrow \Delta_0 \ldots \Gamma_n \Rightarrow \Delta_n \)

Strength condition: \( \text{pre} \land \{M := M_{sk}\} \text{post} \rightarrow \langle p \rangle M =_{\forall} M_{sk} \)

Finding bugs from failed verification attempts: May 20th, 2009
Falsifiability Preservation Analysis

Falsifiability Preservation of a Rule

\[ \Gamma_0 \Rightarrow \Delta_0 \ldots \Gamma_n \Rightarrow \Delta_n \]
\[ \Gamma \Rightarrow \Delta \]

Soundness: \( \Gamma_0 \Rightarrow \Delta_0 \ldots \Gamma_n \Rightarrow \Delta_n \models \Gamma \Rightarrow \Delta \)

Falsifiability preservation: Soundness: \( \Gamma \Rightarrow \Delta \models \Gamma_0 \Rightarrow \Delta_0 \ldots \Gamma_n \Rightarrow \Delta_n \)

Strength condition: \( \text{pre} \land \{ M := M_{sk} \} \text{post} \rightarrow \langle p \rangle M = \forall M_{sk} \)

Finding bugs from failed verification attempts: May 20th, 2009
Falsifiability Preservation Analysis

Falsifiability Preservation of a Rule

\[
\frac{\Gamma_0 \Rightarrow \Delta_0 \ldots \Gamma_n \Rightarrow \Delta_n}{\Gamma \Rightarrow \Delta}
\]

Soundness: \( \Gamma_0 \Rightarrow \Delta_0 \ldots \Gamma_n \Rightarrow \Delta_n \models \Gamma \Rightarrow \Delta \)

Falsifiability preservation: Soundness: \( \Gamma \Rightarrow \Delta \models \Gamma_0 \Rightarrow \Delta_0 \ldots \Gamma_n \Rightarrow \Delta_n \)

Strength condition: \( pre \land \{ M := M_{sk} \} post \rightarrow \langle p \rangle M = \forall M_{sk} \)

Finding bugs from failed verification attempts: May 20th, 2009
Falsifiability Preservation Analysis

Falsifiability Preservation of a Branch

- “path- and postcondition-focussed” strength condition.
- For a branch $S_0 \ldots S_n$ the $F_P^{S_n}_{S_0}$ condition is $\models \neg S_n \rightarrow \neg S_0$
- “falsifiable branch” $\rightarrow \neg (\text{pre} \rightarrow \langle p \rangle \text{post})$

Properties of the approach

- Simple to understand and to implement
- Sound for bug finding, wrt. to soundness of constraint/sat solving
- Properties similar to Phillips approach (symbolic, complete, etc.)
- Search space for bugs is pruned drastically: Correct/ infeasible paths
- Starts with what people want: to verify a program
- Can disambiguate verification failure
- Allows a “performance boost” by specialization

Finding bugs from failed verification attempts:

May 20th, 2009
Falsifiability Preservation Analysis

Falsifiability Preservation of a Branch

- “path- and postcondition-focussed” strength condition.
- For a branch $S_0 \ldots S_n$ the $FP_{S_0}^{S_n}$ condition is $\models \neg S_n \rightarrow \neg S_0$
- “falsifiable branch” $\rightarrow \neg (pre \rightarrow \langle p \rangle post)$

Properties of the approach

- Simple to understand and to implement
- Sound for bug finding, wrt. to soundness of constraint/sat solving
- Properties similar to Phillips approach (symbolic, complete, etc.)
- Search space for bugs is pruned drastically: Correct/ infeasible paths
- Starts with what people want: to verify a program
- Can disambiguate verification failure
- Allows a “performance boost” by specialization

Finding bugs from failed verification attempts: May 20th, 2009
Falsifiability Preservation Analysis

Falsifiability Preservation of a Branch

- “path- and postcondition-focussed” strength condition.
- For a branch $S_0 \ldots S_n$ the $FP_{S_0}^{S_n}$ condition is $\models \neg S_n \rightarrow \neg S_0$
- “falsifiable branch” $\rightarrow \neg (pre \rightarrow \langle p \rangle post)$

Properties of the approach

- Simple to understand and to implement
- Sound for bug finding, wrt. to soundness of constraint/sat solving
  - Properties similar to Phillips approach (symbolic, complete, etc.)
  - Search space for bugs is pruned drastically: Correct/ infeasible paths
- Starts with what people want: to verify a program
- Can disambiguate verification failure
- Allows a “performance boost” by specialization

Finding bugs from failed verification attempts: May 20th, 2009
Falsifiability Preservation Analysis

**Falsifiability Preservation of a Branch**

- “path- and postcondition-focussed” strength condition.
- For a branch $S_0 \ldots S_n$ the $FP_{S_0}^{S_n}$ condition is $\models \neg S_n \rightarrow \neg S_0$
- “falsifiable branch” $\rightarrow \neg (pre \rightarrow \langle p \rangle post)$

**Properties of the approach**

- Simple to understand and to implement
- Sound for bug finding, wrt. to soundness of constraint/sat solving
- Properties similar to Phillips approach (symbolic, complete, etc.)
- Search space for bugs is pruned drastically: Correct/ infeasible paths
- Starts with what people want: to verify a program
- Can disambiguate verification failure
- Allows a “performance boost” by specialization

Finding bugs from failed verification attempts: May 20th, 2009
Falsifiability Preservation Analysis

**Falsifiability Preservation of a Branch**

- “path- and postcondition-focussed” strength condition.
- For a branch $S_0 \ldots S_n$ the $FP_{S_0}^{S_n}$ condition is $\models \neg S_n \rightarrow \neg S_0$
- “falsifiable branch” $\rightarrow \neg (pre \rightarrow \langle p \rangle post)$

**Properties of the approach**

- Simple to understand and to implement
- Sound for bug finding, wrt. to soundness of constraint/sat solving
- Properties similar to Phillips approach (symbolic, complete, etc.)
- Search space for bugs is pruned drastically: Correct/ infeasible paths
- Starts with what people want: to verify a program
- Can disambiguate verification failure
- Allows a “performance boost” by specialization
Falsifiability Preservation Analysis

Falsifiability Preservation of a Branch

- “path- and postcondition-focussed” strength condition.
- For a branch $S_0 \ldots S_n$ the $FP_{S_0}^{S_n}$ condition is $\models \neg S_n \rightarrow \neg S_0$
- “falsifiable branch” $\rightarrow \neg (\text{pre} \rightarrow \langle p \rangle \text{post})$

Properties of the approach

- Simple to understand and to implement
- Sound for bug finding, wrt. to soundness of constraint/sat solving
- Properties similar to Phillips approach (symbolic, complete, etc.)
- Search space for bugs is pruned drastically: Correct/ infeasible paths
- Starts with what people want: to verify a program
- Can disambiguate verification failure
- Allows a “performance boost” by specialization
Falsifiability Preservation Analysis

Falsifiability Preservation of a Branch

- “path- and postcondition-focussed” strength condition.
- For a branch $S_0 \ldots S_n$ the $FP^S_{S_0}$ condition is $\models \neg S_n \rightarrow \neg S_0$
- “falsifiable branch” $\rightarrow \neg (pre \rightarrow \langle p \rangle post)$

Properties of the approach

- Simple to understand and to implement
- Sound for bug finding, wrt. to soundness of constraint/sat solving
- Properties similar to Phillips approach (symbolic, complete, etc.)
- Search space for bugs is pruned drastically: Correct/ infeasible paths
- Starts with what people want: to verify a program
- Can disambiguate verification failure
- Allows a “performance boost” by specialization

Finding bugs from failed verification attempts: May 20th, 2009
Falsifiability Preservation Analysis

**Falsifiability Preservation of a Branch**

- “path- and postcondition-focussed” strength condition.
- For a branch $S_0 \ldots S_n$ the $FP_{S_0}^{S_n}$ condition is $\models \neg S_n \rightarrow \neg S_0$
- “falsifiable branch” $\rightarrow \neg (pre \rightarrow \langle p \rangle post)$

**Properties of the approach**

- Simple to understand and to implement
- Sound for bug finding, wrt. to soundness of constraint/sat solving
- Properties similar to Phillips approach (symbolic, complete, etc.)
- Search space for bugs is pruned drastically: Correct/ infeasible paths
- Starts with what people want: to verify a program
- Can disambiguate verification failure
- Allows a “performance boost” by specialization

Finding bugs from failed verification attempts:

May 20th, 2009
Special Falsifiability Preservation Analysis

**SFP: Special Falsification Preservation condition**

The special falsifiability preservation condition $SFP_{S_i}^{S_n}$ is the conjunction of

$$
((\{M^1 := M^2\} S_n) \land \{U\}\{M^2\} post) \rightarrow S_n
$$

(1)

$$(\neg S_n \land \Gamma_i \land \neg \Delta_i) \rightarrow \{U\} \psi
$$

(2)

$$
\neg S_n \rightarrow (\Gamma_i \land \neg \Delta_i)
$$

(3)

where $\psi = \langle p \rangle true$ “or” $\psi = true$, and some side conditions ...

**Theorem**

Under some side conditions (that are ensured by the iterative algorithm)

$$
SFP_{S_i}^{S_n} \rightarrow FP_{S_i}^{S_n}
$$

Finding bugs from failed verification attempts: May 20th, 2009
### Proof confluence vs. FOL-confluence

**Loop Invariant Rule**

\[ (I, I \land \neg c, M, []) \quad M = \text{mod}(b) \]

1. \( \Gamma \Rightarrow \{U\}I, \Delta \)
2. \( \Gamma \Rightarrow \{U\}\{M\}(I \land c \rightarrow [b]I), \Delta \)
3. \( \Gamma \Rightarrow \{U\}\{M\}((I \land \neg c) \rightarrow \text{post}), \Delta \)

\[ \Gamma \Rightarrow \{U\}[\text{while}(c)\{b;\}]\text{post}, \Delta \]
**Conclusion**

- **Contribution 1**: Falsifiability Preservation Analysis
- **Contribution 2**: Special Falsifiability Preservation Analysis
- It’s **simple**, **fast**, **sound**, **complete***, and **symbolic**
- Hopefully pretty complete in practice when automated
- **Extension** for symbolic execution- or wp-based verification techniques adding **sound bug detection** and **disambiguation** of the reason for verification failure.

Finding bugs from failed verification attempts: May 20th, 2009


**Conclusion**

- **Contribution 1**: Falsifiability Preservation Analysis
- **Contribution 2**: Special Falsifiability Preservation Analysis

- It's simple, fast, sound, complete*, and symbolic
- Hopefully pretty complete in practice when automated
- **Extension** for symbolic execution- or wp-based verification techniques adding sound bug detection and disambiguation of the reason for verification failure.

---

Finding bugs from failed verification attempts: May 20th, 2009
Conclusion

- **Contribution 1**: Falsifiability Preservation Analysis
- **Contribution 2**: Special Falsifiability Preservation Analysis Analysis
- It’s simple, fast, sound, complete*, and symbolic
- Hopefully pretty complete in practice when automated
- **Extension** for symbolic execution- or wp-based verification techniques adding **sound bug detection** and **disambiguation** of the reason for verification failure.
Conclusion

- **Contribution 1**: Falsifiability Preservation Analysis
- **Contribution 2**: Special Falsifiability Preservation Analysis
- It’s *simple, fast, sound, complete*, and *symbolic*
- Hopefully pretty complete in practice when automated
- **Extension** for symbolic execution- or wp-based verification techniques adding *sound bug detection* and *disambiguation* of the reason for verification failure.

Finding bugs from failed verification attempts:

May 20th, 2009
Conclusion

- **Contribution 1**: Falsifiability Preservation Analysis
- **Contribution 2**: Special Falsifiability Preservation Analysis
- It’s **simple, fast, sound, complete**, and **symbolic**
- Hopefully pretty complete in practice when automated
- **Extension** for symbolic execution- or wp-based verification techniques adding **sound bug detection** and **disambiguation** of the reason for verification failure.