Cost-aware Program Repair

Roopsha Samanta

Joint work with Oswaldo Olivo and E. Allen Emerson

The University of Texas at Austin and IST, Austria

12 September, 2014
Fallibility of human programmers

- Infamous bugs
  - Medical, military, aerospace, financial services, automobiles
- Annual cost of software bugs $\approx 60$ billion dollars!

We need correct software.
Fallibility of human programmers

- Infamous bugs
  - Medical, military, aerospace, financial services, automobiles
- Annual cost of software bugs $\approx 60$ billion dollars!

We need correct software.
The road to correct programs?

- Program design + error detection + manual debugging
  - Lengthy, tedious, expensive
  - Error-prone

- Program synthesis
  - Need detailed specification
  - Computationally intensive
  - Why not just modify/reuse legacy code?
The road to correct programs?

- Program design + error detection + manual debugging
  - Need: automated debugging
- Program synthesis
  - Need detailed specification
  - Computationally intensive
  - Why not just modify/reuse legacy code?
Challenges in automated debugging

- Hard to formalize
- Hard to assimilate and automate human expertise
- Undecidable
Challenges in automated debugging

- Hard to formalize
- Hard to assimilate and automate human expertise
- Undecidable
Challenges in automated debugging

- Hard to formalize
- Hard to assimilate and automate human expertise
- Undecidable
Programming language, Correctness

- Typed program variables
- Labeled statement sequences
- Parallel assignment, conditionals, loops, skip
- (Recursive) procedures
- Assertions

\( P \) is (partially) correct iff every finite execution path satisfies all assertions.

```plaintext
\( P \):
main() { 
    int x;
    \( \ell_1 \): if (x \leq 0) 
    \( \ell_2 \): while (x < 0) {
        \( \ell_3 \): x := x + 2;
        \( \ell_4 \): skip;
    } 
    else 
    \( \ell_5 \): if (x == 1) 
    \( \ell_6 \): x := x - 1;
    \( \ell_7 \): assert (x > 1);
}
```
The program debugging/repair problem

Given:
- incorrect $P$
- set of update schemas: $U$

Compute $\hat{P}$ such that:
- $\hat{P}$ is correct
- $\hat{P}$ obtained from $P$ by applying update schemas from $U$

$U = \{id, assign \mapsto assign, if \mapsto if, while \mapsto while\}$
The **cost-aware** program repair problem

Given:
- incorrect $P$
- set of update schemas: $U$
- cost function $c: U \times L \to \mathbb{N}$
- repair budget $\delta$

- $U = \{id, assign \mapsto assign, if \mapsto if, while \mapsto while\}$
- $u \neq id \Rightarrow c((u, \ell)) = 1$
- $\delta = 2$

Compute $\hat{P}$ such that:
- $\hat{P}$ is correct
- $\hat{P}$ obtained from $P$ by applying update schemas from $U$
- total modification cost $\leq \delta$

$\hat{P}$ is a $(U, c, \delta)$-repair of $P$. 
**Predicate abstraction-based solution framework**

- **Concrete program, $P$**
  - Predicate abstraction
  - **Boolean program, $B$**
    - Model checking
      - $B$ correct?
        - Yes: Declare $P$ correct
        - No
          - Error trace in $B$ feasible in $P$?
            - Yes: Declare $P$ incorrect
              - Cost-aware repair of $B$ to get $\widehat{B}$
            - No: Theorem-prover
              - Refine $P$ with new predicates to eliminate error trace
                - Concretize $\widehat{B}$ to get $\widehat{P}$
Predicate abstraction-based solution framework

\(\mathcal{P}:\)
main() {
    int x;
    \(l_1: \text{if } (x \leq 0)\)
    \(l_2: \text{while } (x < 0) \{\)
    \(l_3: \quad x := x + 2;\)
    \(l_4: \quad \text{skip;}\)
    \}
    else
    \(l_5: \text{if } (x == 1)\)
    \(l_6: \quad x := x - 1;\)
    \(l_7: \text{assert } (x > 1);\)
}\n
- \(\mathcal{U} = \{\text{id, assign} \mapsto \text{assign,}\)
  \text{if} \mapsto \text{if, while} \mapsto \text{while}\}\)
- \(u \neq \text{id} \Rightarrow c((u, l)) = 1\)
- \(\delta = 2\)
Predicate abstraction-based solution framework

\( \mathcal{P} \):

```plaintext
main() {
    int x;
    \( l_1 \): if (x \leq 0) {
        \( l_2 \): while (x < 0) {
            \( l_3 \): x := x + 2;
            \( l_4 \): skip;
        }
    } else
    \( l_5 \): if (x == 1)
    \( l_6 \): x := x - 1;
    \( l_7 \): assert (x > 1);
}
```

\( \mathcal{B} \):

```plaintext
main() {
    /* \( \gamma(b_0) = x \leq 1 \) */
    /* \( \gamma(b_1) = x == 1 \) */
    /* \( \gamma(b_2) = x \leq 0 \) */
    bool b_0, b_1, b_2 := *, *, *;
    \( l_1 \): if (!b_2) then goto \( l_5 \);
    \( l_2 \): if (*) then goto \( l_0 \);
    \( l_3 \): b_0, b_1, b_2 := *, *, *
    \( l_4 \): goto \( l_2 \);
    \( l_0 \): goto \( l_7 \);
    \( l_5 \): if (!b_1) then goto \( l_7 \);
    \( l_6 \): b_0, b_1, b_2 := *, *, *
    \( l_7 \): assert (!b_0);
}
```
Predicate abstraction-based solution framework

\[ \mathcal{P}: \]
main() {
    int x;
    \( \ell_1: \) if \( (x \leq 0) \)
    \( \ell_2: \) while \( (x < 0) \) {
        \( \ell_3: \) x := x + 2;
        \( \ell_4: \) skip;
    } else
    \( \ell_5: \) if \( (x == 1) \)
    \( \ell_6: \) x := x - 1;
    \( \ell_7: \) assert \( (x > 1) \);
} \]

\[ \mathcal{B}: \]
main() {
    /* \( \gamma(b_0) = x \leq 1 \) */
    /* \( \gamma(b_1) = x == 1 \) */
    /* \( \gamma(b_2) = x \leq 0 \) */
    Bool \( b_0, b_1, b_2 := *, *, *; \)
    \( \ell_1: \) if \( (\neg b_0 \land \neg b_1 \land b_2) \) then goto \( \ell_5; \)
    \( \ell_2: \) if \( (\neg b_0 \land \neg b_1 \land \neg b_2) \) then goto \( \ell_0; \)
    \( \ell_3: \) \( b_0, b_1, b_2 := *, *, *; \)
    \( \ell_4: \) goto \( \ell_2; \)
    \( \ell_0: \) goto \( \ell_7; \)
    \( \ell_5: \) if \( (\neg b_1) \) then goto \( \ell_7; \)
    \( \ell_6: \) \( b_0, b_1, b_2 := *, *, *; \)
    \( \ell_7: \) assert \( (\neg b_0) \);
}
Predicate abstraction-based solution framework

Motivation
Overview
Approach
Conclusion

Predicate abstraction-based solution framework

Roopsha Samanta
Cost-aware Program Repair

8 / 20
Predicate abstraction-based solution framework

\[ P: \]
\[
\text{main()} \{ \\
i \text{nt x;} \\
l_1: \text{if (true)} \\
l_2: \quad \text{while (x \leq 1)} \{ \\
l_3: \quad x := x + 2; \\
l_4: \quad \text{skip;} \\
\} \\
\quad \text{else} \\
l_5: \quad \text{if (x == 1)} \\
l_6: \quad x := x - 1; \\
l_7: \quad \text{assert (x > 1);} \\
\} \]
Predicate abstraction-based solution framework

Concrete program, $P$

Predicate abstraction

Boolean program, $B$

Model checking

$B$ correct?

Yes

Declare $P$ correct

No

Error trace in $B$ feasible in $P$?

Yes

Declare $P$ incorrect

No

Theorem-prover

Refine $P$ with new predicates to eliminate error trace

Cost-aware repair of $B$ to get $\hat{B}$

Concretize $\hat{B}$ to get $\hat{P}$
Cost-aware repair of Boolean programs

- Adapt method of inductive assertions
- Formulate SMT query — cost-aware repairability condition
- If query is satisfiable, extract $\widehat{B}$ and proof of correctness.
Review: Method of inductive assertions

$$B: \quad main() \{$$
\begin{align*}
/ \ast & \gamma(b_0) = x \leq 1 \ast / \\
/ \ast & \gamma(b_1) = x == 1 \ast / \\
/ \ast & \gamma(b_2) = x \leq 0 \ast / \\
\text{Bool } & b_0, b_1, b_2 := \ast, \ast, \ast; \\
\ell_1 : & \text{ if } (\neg b_2) \text{ then goto } \ell_5; \\
\ell_2 : & \text{ if } (*) \text{ then goto } \ell_0; \\
\ell_3 : & b_0, b_1, b_2 := \ast, \ast, \ast; \\
\ell_4 : & \text{ goto } \ell_2; \\
\ell_0 : & \text{ goto } \ell_7; \\
\ell_5 : & \text{ if } (\neg b_1) \text{ then goto } \ell_7; \\
\ell_6 : & b_0, b_1, b_2 := \ast, \ast, \ast; \\
\ell_7 : & \text{ assert } (\neg b_0); \\
\}
\end{align*}

Transition graph:
Review: Method of inductive assertions

Choose cut-points

Motivation  Overview  Approach  Conclusion

Roopsha Samanta  Cost-aware Program Repair  11 / 20
Review: Method of inductive assertions

- Choose cut-points
- Enumerate verification paths
Review: Method of inductive assertions

- Choose cut-points
- Enumerate verification paths
- Attach inductive assertions to cut-points
Review: Method of inductive assertions

- Choose cut-points
- Enumerate verification paths (denoted $\pi$)
- Attach inductive assertions to cut-points
- For each $\pi$, formulate $VC(\pi)$
  - $VC(\pi) \sim <I_\ell > stmt(\pi) <I_{\ell'}>$
Review: Method of inductive assertions

- Choose cut-points
- Enumerate verification paths
- Attach inductive assertions to cut-points
- For each $\pi$, formulate $VC(\pi)$

$B$ is (partially) correct if:

$\exists I_{\ell_1} \ldots I_{\ell_{exit}} \forall \text{Var} : \land_{\pi} VC(\pi)$
Cost-aware repairability conditions

Choose cut-points
- Enumerate verification paths
- Attach inductive assertions to cut-points
Cost-aware repairability conditions

- Choose cut-points
- Enumerate verification paths
- Attach inductive assertions to cut-points
- Attach costs to locations
Cost-aware repairability conditions

- Choose cut-points
- Enumerate verification paths
- Attach inductive assertions to cut-points
- Attach costs to locations
- Expressions → unknown functions

\[
\begin{align*}
I_{\ell_1}, C_{\ell_1} & \quad \text{assume}(f_1) \\
I_{\ell_2}, C_{\ell_2} & \quad \text{assume}(f_3) \\
C_{\ell_3} & \quad \text{assume}(f_4) \\
I_{\ell_0}, C_{\ell_0} & \quad \text{assume}(f_5) \\
b_0, b_1, b_2 := f_7, f_8, f_9 & \quad C_{\ell_4} \\
I_{\ell_7}, C_{\ell_7} & \quad \text{assume}(f_6) \\
b_0, b_1, b_2 := f_{10}, f_{11}, f_{12} & \\
err & \quad I_{exit}, C_{exit} \\
exit & 
\end{align*}
\]
Cost-aware repairability conditions

- Choose cut-points
- Enumerate verification paths
- Attach inductive assertions to cut-points
- Attach costs to locations
- Expressions $\rightarrow$ unknown functions
- For each $\pi$, formulate $CRC(\pi)$
Cost-aware repairability conditions

- Choose cut-points
- Enumerate verification paths
- Attach inductive assertions to cut-points
- Attach costs to locations
- Expressions $\rightarrow$ unknown functions
- For each $\pi$, formulate $CRC(\pi)$

$B$ is repairable within budget $\delta$ if:
\[ \exists Unknown \ \forall Var : C_{exit} \leq \delta \land \bigwedge_{\pi} CRC(\pi) \land \text{AssumeConstraints} \]
Key result

Given:
- incorrect Boolean program $B$, annotated with assertions
- $U = \{\text{id}, \text{assign} \leftrightarrow \text{assign}, \text{assign} \leftrightarrow \text{skip}, \text{assume} \leftrightarrow \text{assume}, \text{call} \leftrightarrow \text{call}, \text{call} \leftrightarrow \text{skip}\}$.
- cost function $c$
- repair budget $\delta$

1. if there exists a $(U, c, \delta)$-repair of $B$, our method finds a $(U, c, \delta)$-repair of $B$,
2. if our method finds a $\widehat{B}$, then $\widehat{B}$ is a $(U, c, \delta)$-repair of $B$. 
Predicate abstraction-based solution framework

Concrete program, $\mathcal{P}$

Predicate abstraction

Boolean program, $\mathcal{B}$

Model checking

$\mathcal{B}$ correct?

Yes

Declare $\mathcal{P}$ correct

No

Error trace in $\mathcal{B}$ feasible in $\mathcal{P}$?

Yes

Declare $\mathcal{P}$ incorrect

Cost-aware repair of $\mathcal{B}$ to get $\hat{\mathcal{B}}$

No

Theorem-prover

Refine $\mathcal{P}$ with new predicates to eliminate error trace

Concretize $\hat{\mathcal{B}}$ to get $\hat{\mathcal{P}}$
Concretization of $\hat{B}$

- $\Gamma(\text{skip}) = \text{skip}$
- $\Gamma(\text{if}(g)) = \text{if}(\gamma(g))$
- $\Gamma(\text{while}(g)) = \text{while}(\gamma(g))$
- $\Gamma(\text{call } F_j(e_1, \ldots, e_k) = \text{call } F_j(\gamma(e_1), \ldots, \gamma(e_k))$

Concretization of assignment statements is more involved.

- $b_1, \ldots, b_r := e_1, \ldots, e_r$ is concretizable if:
  \[ \exists f_1, \ldots, f_q \ \forall v_1, \ldots, v_q : \ \land_{i=1}^r \gamma(b_i)[v_1/f_i, \ldots, v_q/f_q] = \gamma(e_i) \]
- If $\text{expr}_1, \ldots, \text{expr}_q$ are models for $f_1, \ldots, f_q$:
  \[ v_1, \ldots, v_q := \text{expr}_1, \ldots, \text{expr}_q \in \Gamma(b_1, \ldots, b_r := e_1, \ldots, e_r) \]
- Can use templates for $f_1, \ldots, f_q$
Concretization of $\widehat{B}$

- $\Gamma(\text{skip}) = \text{skip}$
- $\Gamma(\text{if}(g)) = \text{if}(\gamma(g))$
- $\Gamma(\text{while}(g)) = \text{while}(\gamma(g))$
- $\Gamma(\text{call } F_j(e_1, \ldots, e_k) = \text{call } F_j(\gamma(e_1), \ldots, \gamma(e_k))$

Concretization of assignment statements is more involved.

- $b_1, \ldots, b_r := e_1, \ldots, e_r$ is concretizable if:
  - $\exists f_1, \ldots, f_q \forall v_1, \ldots, v_q : \bigwedge_{i=1}^r \gamma(b_i)[v_1/f_1, \ldots, v_q/f_q] = \gamma(e_i)$
- If $\text{expr}_1, \ldots, \text{expr}_q$ are models for $f_1, \ldots, f_q$:
  - $v_1, \ldots, v_q := \text{expr}_1, \ldots, \text{expr}_q \in \Gamma(b_1, \ldots, b_r := e_1, \ldots, e_r)$
- Can use templates for $f_1, \ldots, f_q$
Experiments

handmade1: int main() {
    int x;
    $\ell_1$: while ($x < 0$)
    $\ell_2$: $x := x + 1$;
    $\ell_3$: assert ($x > 0$);
}

handmade2: int main() {
    int x;
    $\ell_1$: while ($x \leq 0$)
    $\ell_2$: $x := x + 1$;
    $\ell_3$: assert ($x > 0$);
}
Experiments

handmade2:
int main() {
    int x;
    l1: if (x <= 0) {
        l2: while (x < 0) {
            l3: x := x + 2;
            l4: skip;
        }
        else
        l5: if (x == 1)
        l6: x := x - 1;
        l7: assert (x > 1);
    }
    else
    l1: if (true)
    l2: while (x <= 1) {
        l3: x := x + 2;
        l4: skip;
    }
    else
    l5: if (x == 1)
    l6: x := x - 1;
    l7: assert (x > 1);
}
Experiments

```c
necex6:
int x, y;
int foo(int *ptr) {
    
    l4: if (ptr == &x) {
        l5: *ptr := 0;
        l6: if (ptr == &y) {
            l7: *ptr := 1;
            return 1;
        }
    }
}

int main() {
    l1: foo(&x);
    l2: foo(&y);
    l3: assert (x > y);
}
```

```c
necex6:
int x, y;
int foo(int *ptr) {
    
    l4: if (ptr == &x) {
        l5: *ptr := 0;
        l6: if (ptr == &y) {
            l7: *ptr := 1;
            return 1;
        }
    }
}

int main() {
    l1: foo(&x);
    l2: foo(&y);
    l3: assert (x > y);
}
```
Experiments

```c
necexl4:
int main() {
  int x, y;
  int a[10];
  l1: x := 1U;
  l2: while (x ≤ 10U) {
    l3: y := 11 − x;
    l4: assert (y ≥ 0 ∧ y < 10);
    l5: a[y] := −1;
    l6: x := x + 1;
  }
}
```
## Experiments

<table>
<thead>
<tr>
<th>Name</th>
<th>LoC((P))</th>
<th>LoC((B))</th>
<th>(V(B))</th>
<th>(B)-time</th>
<th>Que-time</th>
<th>Sol-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>handmade1</td>
<td>6</td>
<td>58</td>
<td>1</td>
<td>0.180s</td>
<td>0.009s</td>
<td>0.012s</td>
</tr>
<tr>
<td>handmade2</td>
<td>16</td>
<td>53</td>
<td>3</td>
<td>0.304s</td>
<td>0.040s</td>
<td>0.076s</td>
</tr>
<tr>
<td>necex6</td>
<td>24</td>
<td>66</td>
<td>3</td>
<td>0.288s</td>
<td>0.004s</td>
<td>0.148s</td>
</tr>
<tr>
<td>necex14</td>
<td>13</td>
<td>60</td>
<td>2</td>
<td>0.212s</td>
<td>0.004s</td>
<td>0.032s</td>
</tr>
<tr>
<td>while_inf_loop</td>
<td>5</td>
<td>33</td>
<td>1</td>
<td>0.196s</td>
<td>0.002s</td>
<td>0.008s</td>
</tr>
<tr>
<td>array</td>
<td>23</td>
<td>57</td>
<td>4</td>
<td>0.384s</td>
<td>0.004s</td>
<td>0.116s</td>
</tr>
<tr>
<td>n.c11</td>
<td>27</td>
<td>50</td>
<td>2</td>
<td>0.204s</td>
<td>0.002s</td>
<td>0.024s</td>
</tr>
<tr>
<td>terminator_03</td>
<td>22</td>
<td>38</td>
<td>2</td>
<td>0.224s</td>
<td>0.004s</td>
<td>0.036s</td>
</tr>
<tr>
<td>trex03</td>
<td>23</td>
<td>58</td>
<td>3</td>
<td>0.224s</td>
<td>0.036s</td>
<td>0.540s</td>
</tr>
<tr>
<td>trex04</td>
<td>29</td>
<td>36</td>
<td>1</td>
<td>0.200s</td>
<td>0.004s</td>
<td>0.004s</td>
</tr>
<tr>
<td>veris.c</td>
<td>30</td>
<td>144</td>
<td>23</td>
<td>3.856s</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>vogal</td>
<td>41</td>
<td>-</td>
<td>-</td>
<td>&gt;10m</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>count_up_down</td>
<td>18</td>
<td>-</td>
<td>-</td>
<td>&gt;10m</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Contributions

- Methodical problem formulation
  - Update schemas
  - Cost-awareness
  - Template-based repair
Contributions

- Predicate abstraction-based framework
  - Can repair programs that
    - are infinite-state
    - have interesting datatypes
    - have recursive procedures
    - have multiple assertions
    - need multiple statement modifications
  - Sound and complete algorithm for repairing Boolean programs
  - Concretization strategies
  - Prototype tool
Related Work

- Repair of Boolean programs [GBC06]
- Repair of concrete programs [KB11]
- Games for program repair [JGB05,SJB05]
- Partial program synthesis [SRBE05,SGF10]
- Fault localization [Shapiro82,JM11,ZH02]
Future Work

- Extend to total correctness (ranking functions)
- Combine computation of $\hat{B}$ and concretization
- Compute weakest inductive assertions, least restrictive $\hat{B}$
- Other update schemas
Thank you.