Automatic Generation of Local Repairs for Boolean Programs

Roopsha Samanta, 
Jyotirmoy V. Deshmukh and E. Allen Emerson

The University of Texas at Austin

April 5, 2012
Outline

- Motivation
- Solution Framework
- The Algorithm
- Conclusions
The road to correct programs . . .

- **Program synthesis**
  - Correct by construction
  - Detailed specification
  - Hard
  - Also, legacy code?

- **Program verification**
  - Program design + verification + fault localization + repair
  - Lengthy, iterative cycle
  - Long, unreadable error traces
  - Essentially manual debugging
The road to correct programs . . .

- Program *synthesis*
  - Correct by construction
  - Detailed specification
  - Hard
  - Also, legacy code?

- Program *verification*
  - Program design + *verification* + *fault localization* + repair
    - Lengthy, iterative cycle
    - Long, unreadable error traces
    - Essentially manual debugging
The road to correct programs . . .

- **Program synthesis**
  - Correct by construction
  - Detailed specification
  - Hard
  - Also, legacy code?

- **Program verification**
  - Program design + verification + fault localization + repair
    - Lengthy, iterative cycle
    - Long, unreadable error traces
    - Essentially manual debugging
The road to correct programs . . .

- Program synthesis
  - Correct by construction
  - Detailed specification
  - Hard
  - Also, legacy code?

- Program verification

- Program design + verification + fault localization + repair
  - Lengthy, iterative cycle
  - Long, unreadable error traces
  - Essentially manual debugging
The road to correct programs . . .

- **Program synthesis**
  - Correct by construction
  - Detailed specification
  - Hard
  - Also, legacy code?

- **Program verification**

- **Program design + verification + fault localization + repair**
  - Lengthy, iterative cycle
  - Long, unreadable error traces
  - Essentially manual *debugging*
The road to correct programs . . .

- **Program synthesis**
  - Correct by construction
  - Detailed specification
  - Hard
  - Also, legacy code?

- **Program verification**

- **Program design + verification + fault localization + repair**
  - Lengthy, iterative cycle
  - Long, unreadable error traces
  - Essentially manual **debugging**
The road to correct programs . . .

- Program *synthesis*
  - Correct by construction
  - Detailed specification
  - Hard
  - Also, legacy code?

- Program *verification*
  - Program design + *verification* + *fault localization* + *repair*
    - Lengthy, iterative cycle
    - Long, unreadable error traces
    - Essentially manual *debugging*
The repair problem

Given a program $P$ and a specification $\Phi$ such that $P \not\models \Phi$, transform $P$ to $P'$ such that $P' \models \Phi$
A specialization...

- Program model: sequential Boolean programs
- Specifications: Hoare-style pre-conditions, post-conditions
- Permissible faults/repairs: incorrect Boolean expressions
Iterative (predicate) abstraction-refinement

\[ P_C \models \Phi \]  Correct!

\[ P_C \not\models \Phi \]  Bug!

\[ P_A \models \Phi \]  Yes

Feasible Error Trace?

\[ P_A \not\models \Phi \]  No

Refine \( P_A \)

Theorem Prover

Model Checking

Predicate Abstraction

\( P_C \)  \( P_A \)
Iterative (predicate) abstraction-refinement

![Diagram](attachment:image.png)
What are Boolean programs?

- Abstractions of concrete programs
- Boolean variables
- Similar control flow
  - Conditionals, loops, procedures
- Nondeterminism
  - Some expressions may evaluate to either true or false
Example C program and Boolean program

```
while (x>0){
    x := x-1;
}
```

```
while (p){
    p := nd(0,1);
}
```
Why Boolean programs?

- Used as program abstractions for software verification
  - e.g., SLAM, BLAST, etc.
Repair of software programs
Why Boolean programs?

- Used as program abstractions for software verification
  - e.g., SLAM, BLAST, etc.
- Could be used to model some Boolean circuits
Program Syntax

- **Prog** \( P = (V, \text{main}, \mathcal{F}) \)
  - \( V = \{v_1, v_2, \ldots, v_t\} \): Boolean vars
  - main = \((S, V), S: s_1; s_2; \ldots; s_n: \text{stmts} \)
  - \( \mathcal{F} \): functions, \( f = (S_f, V_f, l) \)

- **Expr** \( E \): Boolean expr + \( \text{nd}(0, 1) \)
  - e.g., \( v_2 \land \text{nd}(0, 1) \)

- **Prog** stmt \( s_i \): function call or return or,
  - assignment: \( v_j := E \)
  - conditional: if (G) \( S_{\text{if}} \) else \( S_{\text{else}} \)
  - loop: while (G) \( S_{\text{body}} \)
Program Syntax

- **Prog** $\mathcal{P} = (\mathcal{V}, \text{main}, \mathcal{F})$
  - $\mathcal{V} = \{v_1, v_2, \ldots, v_t\}$: Boolean vars
  - main $= (S, \mathcal{V})$, $S$: $s_1; s_2; \ldots; s_n$: stmts
  - $\mathcal{F}$: functions, $f = (S_f, \mathcal{V}_{f,l})$

- **Expr** $E$: Boolean expr + $\text{nd}(0, 1)$
  - e.g., $v_2 \land \text{nd}(0, 1)$

- Prog stmt $s_i$: function call or return or,
  - assignment: $v_j := E$
  - conditional: if $(G)$ $S_{\text{if}}$ else $S_{\text{else}}$
  - loop: while $(G)$ $S_{\text{body}}$
Program Syntax

- **Prog** $\mathcal{P} = (\mathcal{V}, \text{main}, \mathcal{F})$
  - $\mathcal{V} = \{v_1, v_2, \ldots, v_t\}$: Boolean vars
  - main = $(S, \mathcal{V})$, $S: s_1; s_2; \ldots; s_n$: stmts
  - $\mathcal{F}$: functions, $f = (S_f, \mathcal{V}_{f,l})$

- **Expr** $E$: Boolean expr + $nd(0, 1)$
  - e.g., $v_2 \land nd(0, 1)$

- **Prog stmt** $s_i$: function call or return or,
  - assignment: $v_j := E$
  - conditional: if $(G)$ $S_{if}$ else $S_{else}$
  - loop: while $(G)$ $S_{body}$
Example Boolean program and its state diagram

```plaintext
swap(x, y) {
    x := x ⊕ y;
    y := x ∧ y;
    x := x ⊕ y;
}
```
Specification

**Total correctness:** \( \langle \varphi \rangle P \langle \psi \rangle \)

- Pre-condition \( \varphi \): init states of \( P \)
- Post-condition \( \psi \): desired final states

\( P \) is correct iff execution of \( P \), begun in any state in \( \varphi \), terminates in a state in \( \psi \), for all choices that \( P \) might make.
Specification

**Total correctness:** $\langle \varphi \rangle P \langle \psi \rangle$

- Pre-condition $\varphi$: init states of $P$
- Post-condition $\psi$: desired final states

$P$ is correct *iff* execution of $P$, begun in any state in $\varphi$, terminates in a state in $\psi$, for *all* choices that $P$ might make.
Example Boolean program with its specification

\(\varphi : \text{true}\)

\[
\begin{align*}
x & := x \oplus y; \\
y & := x \land y; \\
x & := x \oplus y;
\end{align*}
\]

\(\psi : (y_f \equiv x(0) \land (x(f) \equiv y(0)))\)
Fault/repair model

- Extra statement (needs deletion)
- Assignment: faulty LHS or RHS
- Conditional: faulty $G$ or faulty statement in $S_{if}$ or $S_{else}$
- Loop: faulty $G$ or faulty statement in $S_{body}$

Our algorithm seeks to repair only the above kinds of faults.
Fault/repair model

- Extra statement (needs deletion)
- Assignment: faulty LHS or RHS
- Conditional: faulty $G$ or faulty statement in $S_{if}$ or $S_{else}$
- Loop: faulty $G$ or faulty statement in $S_{body}$

Our algorithm seeks to repair only the above kinds of faults.
Algorithm sketch

- **Annotation:**
  - Propagate $\varphi$ and $\psi$ through statements

- **Repair:**
  - Use annotations to inspect statements for *repairability*
  - Generate repair if possible
Program annotation

\( \varphi_0 : \text{true} \)

**Incorrect Program**

\[
\begin{align*}
S_0: & \quad x' := x(0) \oplus y(0) \\
S_1: & \quad y' := x \land y \\
S_2: & \quad x(f) := x \oplus y \\
\end{align*}
\]

\( \psi_3 : x(f) \equiv y(0) \land y(f) \equiv x(0) \)
Program annotation

\[ \varphi_0 : \text{true} \]

**Incorrect Program**

\[ \begin{align*}
S_0 &: x' := x(0) \oplus y(0) \\
S_1 &: y' := x \land y \\
S_2 &: x(f) := x \oplus y
\end{align*} \]

\[ \psi_3 : x(f) \equiv y(0) \land y(f) \equiv x(0) \]

*Post-condition propagation*
Program annotation

$\varphi_0 : \text{true}$

**Incorrect Program**

$S_0 : x' := x(0) \oplus y(0)$

$S_1 : y' := x \land y$

$S_2 : x(f) := x \oplus y$

$\psi_2$

$\psi_3 : x(f) \equiv y(0) \land y(f) \equiv x(0)$

Post-condition propagation
Program annotation

**Motivation**

**Solution Framework**

**The Algorithm**

**Conclusions**

---

**Incorrect Program**

\[\varphi_0 : true\]

\[S_0 : x' := x(0) \oplus y(0);\]

\[S_1 : y' := x \land y;\]

\[S_2 : x(f) := x \oplus y;\]

\[\psi_1\]

\[\psi_2\]

\[\psi_3 : x(f) \equiv y(0) \land y(f) \equiv x(0)\]

*Post-condition propagation*
Program annotation

\( \varphi_0 : true \)

\( S_0: x' := x(0) \oplus y(0); \)

\( S_1: y' := x \land y; \)

\( S_2: x(f) := x \oplus y; \)

\( \psi_0 \)

\( \psi_1 \)

\( \psi_2 \)

\( \psi_3 : x(f) \equiv y(0) \land y(f) \equiv x(0) \)

Post-condition propagation
Program annotation

Pre-condition propagation

$\varphi_0 : true$

Incorrect Program

$S_0: x' := x(0) \oplus y(0)$;

$S_1: y' := x \land y$;

$S_2: x(f) := x \oplus y$;

$\psi_0$

$\psi_1$

$\psi_2$

$\psi_3 : x(f) \equiv y(0) \land y(f) \equiv x(0)$

Post-condition propagation
Program annotation

Pre-condition propagation

\( \varphi_0 : \text{true} \)
\( \varphi_1 \)
\( \varphi_2 \)
\( \varphi_3 \)

Incorrect Program

\( S_0: x' := x(0) \oplus y(0); \)
\( S_1: y' := x \land y; \)
\( S_2: x(f) := x \oplus y; \)

Post-condition propagation

\( \psi_0 \)
\( \psi_1 \)
\( \psi_2 \)
\( \psi_3 : x(f) \equiv y(0) \land y(f) \equiv x(0) \)
Backward propagation of $\psi_i$ through $s_i$

Weakest pre-condition $wp(s_i, \psi_i)$:
Set of all input states from which $s_i$ is guaranteed to terminate in $\psi_i$ for all choices made by $s_i$.

To propagate $\psi_i$ back through $s_i$, compute $wp(s_i, \psi_i)$. 
Assignments: \( v_j := E; \)
\[ \psi_{i-1} = \psi_i[v_j' \rightarrow E, \text{for each } m \neq j, v'_m \rightarrow v_m] \]

Rule for sequential composition:
\[ wp((s_{i-1}; s_i), \psi_i) = wp(s_{i-1}, wp(s_i, \psi_i)) \]

Conditionals: \( \text{if } (G) \ S_\text{if} \ \text{else } S_\text{else}; \)
\[ \psi_{i-1} = (G \Rightarrow wp(S_\text{if}, \psi_i)) \land (\neg G \Rightarrow wp(S_\text{else}, \psi_i)) \]

Loops: \( \text{while } (G) \ S_\text{body}; \)
\[ \psi_{i-1} = (\psi_i \land \neg G) \lor \bigvee_{l=1}^{L} wp(S_\text{body}, Y_{l-1} \land \neg G) \]
where, \( Y_0 = \psi_i, Y_k = wp(S_\text{body}, Y_{k-1} \land \neg G) \)
Forward propagation of $\varphi_{i-1}$ through $s_i$

Strongest post-condition $sp(s_i, \varphi_{i-1})$:
Smallest set of output states in which $s_i$ is guaranteed to terminate, starting in $\varphi_{i-1}$, for all choices that $s_i$ might make.

To propagate $\varphi_{i-1}$ forward through $s_i$, compute $sp(s_i, \varphi_{i-1})$. 
Example program annotation

Pre-condition propagation

\( \varphi_0: \text{true} \)

\( \varphi_1: x' \equiv (x(0) \oplus y(0)) \land y' \equiv y(0) \)

\( \varphi_2: x' \equiv (x(0) \oplus y(0)) \land y' \equiv (\neg x(0) \land y(0)) \)

\( \varphi_3: x' \equiv (x(0) \land \neg y(0)) \land y' \equiv (\neg x(0) \land y(0)) \)

Incorrect Program

\[
\begin{align*}
x' & := x(0) \oplus y(0); \\
y' & := x \land y;
\end{align*}
\]

Post-condition propagation

\( \psi_0: y(0) \equiv (x(0) \land \neg y(0)) \land x(0) \equiv (\neg x(0) \land y(0)) \)

\( \psi_1: y(0) \equiv (x \land \neg y) \land x(0) \equiv (x \land y) \)

\( \psi_2: y(0) \equiv x \oplus y \land x(0) \equiv y \)

\( \psi_3: x(f) \equiv y(0) \land y(f) \equiv x(0) \)
Local Hoare triples

\[ \begin{align*}
\varphi_0 & : x' := x(0) \oplus y(0); \\
\varphi_1 & : y' := x \land y; \\
\varphi_2 & : x(f) := x \oplus y; \\
\varphi_3 & : \\
\psi_0 & : \\
\psi_1 & : \\
\psi_2 & : \\
\psi_3 & : 
\end{align*} \]
Local Hoare triples

Local Hoare triple: $\langle \phi_0 \rangle s_0 \langle \psi_1 \rangle$

$\phi_0$

$s_0$: $x' := x(0) \oplus y(0)$

$\phi_1$

$s_1$: $y' := x \land y$

$\phi_2$

$s_2$: $x(f) := x \oplus y$

$\phi_3$

$s_3$

$\psi_0$

$\psi_1$

$\psi_2$

$\psi_3$
Local Hoare triples

\[
\begin{align*}
\varphi_0 & : x' := x(0) \oplus y(0) \\
\varphi_1 & : y' := x \land y \\
\varphi_2 & : x(f) := x \oplus y \\
\varphi_3 & : \\
\end{align*}
\]

Local Hoare triple: \( \langle \varphi_0 \rangle_{S_0} \langle \psi_1 \rangle \)

\[
\begin{align*}
S_0 & : x' := x(0) \oplus y(0) \\
S_1 & : y' := x \land y \\
S_2 & : x(f) := x \oplus y \\
\end{align*}
\]

Local Hoare triple: \( \langle \varphi_2 \rangle_{S_2} \langle \psi_3 \rangle \)
A key lemma

\[ \langle \varphi \rangle P \langle \psi \rangle \text{ false} \iff \text{all local Hoare triples false.} \]

All local Hoare triples \textit{false} \iff some local Hoare triple \textit{false}. 
What does this lemma mean for us?

If for some $i$, $s_i$ can be fixed to make $\langle \varphi_{i-1} \rangle s_i \langle \psi_i \rangle$ true, then we have found $P'$ such that $\langle \varphi \rangle P' \langle \psi \rangle$!

This is the basis for our repair algorithm.
What does this lemma mean for us?

If for some $i$, $s_i$ can be fixed to make $\langle \varphi_{i-1} \rangle s_i \langle \psi_i \rangle$ true, then we have found $P'$ such that $\langle \varphi \rangle P' \langle \psi \rangle$!

This is the basis for our repair algorithm.
Sketch of repair algorithm

• Choose promising order
  • Query stmts in turn for repairability
    • If yes, Repair stmt, return modified program
    • If not, move to next stmt
  • If Query fails for all stmts, report failure
Sketch of repair algorithm

- Choose promising order
- *Query* stmts in turn for repairability
  - If yes, *Repair* stmt, return modified program
  - If not, move to next stmt
- If *Query* fails for all stmts, report failure
Sketch of repair algorithm

- Choose promising order
- **Query** stmts in turn for repairability
  - If yes, **Repair** stmt, return modified program
  - If not, move to next stmt
- If **Query** fails for all stmts, report failure
Sketch of repair algorithm

- Choose promising order
- **Query** stmts in turn for repairability
  - If yes, **Repair** stmt, return modified program
  - If not, move to next stmt
- If **Query** fails for all stmts, report failure
Sketch of repair algorithm

- Choose promising order
- **Query** stmts in turn for repairability
  - If yes, **Repair** stmt, return modified program
  - If not, move to next stmt
- If **Query** fails for all stmts, report failure
Query for assignment statement

- Let $\hat{s}_j \colon v_j := \text{expr}$ be potential repair for $s_i$
- Use variable $z$ to denote $\text{expr}$ to enable formulation of Quantified Boolean Formula (QBF)

Query returns yes iff following QBF is true for some $j$:
$$\forall v_1(0) \forall v_2(0) \ldots \forall v_t(0) \exists z \varphi_{i-1} \Rightarrow \hat{\psi}_{i-1,j}$$
Query for assignment statement

- Let $\hat{s}_j: v_j := \text{expr}$ be potential repair for $s_j$
- Use variable $z$ to denote $\text{expr}$ to enable formulation of Quantified Boolean Formula (QBF)

Query returns yes iff following QBF is true for some $j$:
$$\forall v_1(0) \forall v_2(0) \ldots \forall v_t(0) \exists z \quad \varphi_{i-1} \Rightarrow \hat{\psi}_{i-1,j}$$
Repair for assignment statement

Let $m^{th}$ QBF be true
Thus, $\hat{s}_i$: $v_m := z$

How do we obtain $z$ in terms of variables in $\forall$?

$\forall v_1(0) \forall v_2(0) \ldots \forall v_t(0) \exists z \quad \varphi_{i-1} \Rightarrow \hat{\psi}_{i-1,m}$

$z = T|_z$ is a witness to QBF validity
Repair for assignment statement

- Let $m^{th}$ QBF be true
- Thus, $\hat{s}_i: \forall v_m := z$

How do we obtain $z$ in terms of variables in $\mathcal{V}$?

$$\forall v_1(0) \forall v_2(0) \ldots \forall v_t(0) \exists z \varphi_{i-1} \Rightarrow \hat{\psi}_{i-1,m}$$

$z = T|_z$ is a witness to QBF validity
Repair for assignment statement

- Let $m^{th}$ QBF be true
- Thus, $\hat{s}_i : v_m := z$
- How do we obtain $z$ in terms of variables in $V$?

\[
\forall v_1(0) \forall v_2(0) \ldots \forall v_t(0) \exists z \quad \varphi_{i-1} \Rightarrow \hat{\psi}_{i-1,m} \\
z = T|_z \text{ is a witness to QBF validity}
\]
Example

**Pre-condition propagation**

\( \varphi_0: \text{true} \)

\( \varphi_1: x' \equiv (x(0) \oplus y(0)) \land y' \equiv y(0) \)

\( \varphi_2: x' \equiv (x(0) \oplus y(0)) \land y' \equiv (\neg x(0) \land y(0)) \)

\( \varphi_3: x' \equiv (x(0) \land \neg y(0)) \land y' \equiv (\neg x(0) \land y(0)) \)

**Incorrect Program**

\[ x' := x(0) \oplus y(0); \]

\[ y' := x \land y; \]

\[ x(f) := x \oplus y; \]

**Post-condition propagation**

\( \psi_0: y(0) \equiv (x(0) \land \neg y(0)) \land x(0) \equiv (\neg x(0) \land y(0)) \)

\( \psi_1: y(0) \equiv (x \land \neg y) \land x(0) \equiv (x \land y) \)

\( \psi_2: y(0) \equiv x \oplus y \land x(0) \equiv y \)

\( \psi_3: x(f) \equiv y(0) \land y(f) \equiv x(0) \)

**QBF for** \( \hat{s}_2: \forall x(0) \forall y(0) \exists z \; \varphi_1 \Rightarrow \hat{\psi}_{1,y} = \text{true} \)

**Synthesized repair:** \( y' := x \oplus y; \)
Complexity

Worst-case complexity exponential in \( \# \) Boolean predicates

In practice, most computations are efficient using BDDs

- Symbolic storage
- Efficient manipulation of pre-/post-conditions
- Efficient computation of fix-points
- Easy QBF validity checking
- Easy cofactor computation
Complexity

Worst-case complexity exponential in \# Boolean predicates

In practice, most computations are efficient using BDDs

- Symbolic storage
- Efficient manipulation of pre-/post-conditions
- Efficient computation of fix-points
- Easy QBF validity checking
- Easy cofactor computation
Extant work

- Error localization based on analyzing error traces: [Ze02], [RenRei03], [BaNaRa03], [ShQiLi04], [Gro05]
- Repair of Boolean programs: [GrBlCoo06]
- Sketching: [S-LTaBoSeSa06]
- Repair of circuits using QBFs: [StBl07]
- Dynamic repair of data structures: [DeRi03], [ElGaSuKh07]
Contributions

- Novel application of Hoare logic
- Identification of program model, fault model and specification logic for tractable repair algorithm
- Framework for repair without prior fault localization
- Exponentially lower complexity than existing algorithm ([Griesmayer et al. 2006]) for our fragment
Contributions

- Novel application of Hoare logic
- Identification of program model, fault model and specification logic for tractable repair algorithm
- Framework for repair without prior fault localization
- Exponentially lower complexity than existing algorithm ([Griesmayer et al. 2006]) for our fragment
Contributions

- Novel application of Hoare logic
- Identification of program model, fault model and specification logic for tractable repair algorithm
- Framework for repair without prior fault localization
- Exponentially lower complexity than existing algorithm ([Griesmayer et al. 2006]) for our fragment
Contributions

- Novel application of Hoare logic
- Identification of program model, fault model and specification logic for tractable repair algorithm
- Framework for repair without prior fault localization
- Exponentially lower complexity than existing algorithm ([Griesmayer et al. 2006]) for our fragment
The road ahead . . .

- More general fault models
  - e.g., swapped statements, multiple incorrect expressions
- Boolean programs with arbitrary recursion
- Bit-vector programs
  - VHDL or Verilog programs
  - Software programs with small integer domains