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

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

$\mathcal{P}$:

```
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);
}
```

The program debugging/repair problem

Given:

- incorrect $\mathcal{P}$
- set of update schemas: $\mathcal{U}$

- $\mathcal{U} = \{id, \texttt{assign} \mapsto \texttt{assign}, \texttt{if} \mapsto \texttt{if}, \texttt{while} \mapsto \texttt{while}\}$

Compute $\widehat{\mathcal{P}}$ such that:

- $\widehat{\mathcal{P}}$ is correct
- $\widehat{\mathcal{P}}$ obtained from $\mathcal{P}$ by applying update schemas from $\mathcal{U}$

The **cost-aware** program repair problem

Given:

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

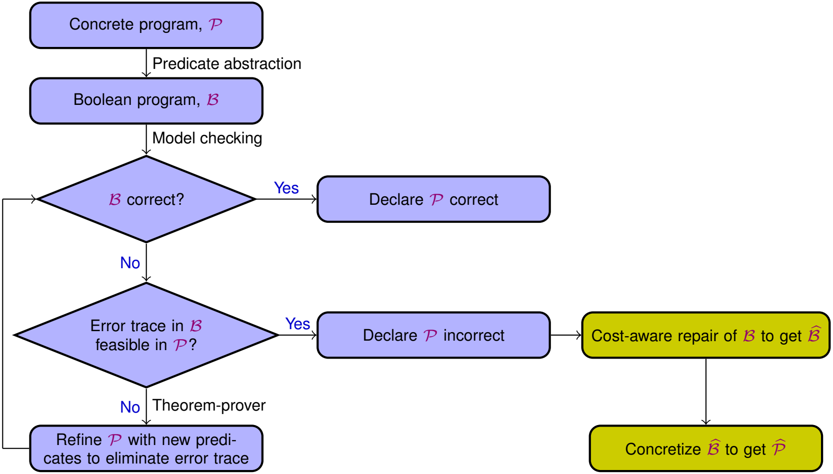
- $\mathcal{U} = \{id, \texttt{assign} \mapsto \texttt{assign}, \texttt{if} \mapsto \texttt{if}, \texttt{while} \mapsto \texttt{while}\}$
- $u \neq id \;\Rightarrow\; c((u, \ell)) = 1$
- $\delta = 2$

Compute $\widehat{\mathcal{P}}$ such that:

- $\widehat{\mathcal{P}}$ is correct
- $\widehat{\mathcal{P}}$ obtained from $\mathcal{P}$ by applying update schemas from $\mathcal{U}$
- total modification cost $\leq \delta$

$\widehat{\mathcal{P}}$ is a $(\mathcal{U}, c, \delta)$-repair of $\mathcal{P}$.

Predicate abstraction-based solution framework



Predicate abstraction-based solution framework

$\mathcal{P}$:
```
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);
}
```

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

Predicate abstraction-based solution framework

$\mathcal{P}$:

```
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);
}
```

$\mathcal{B}$:

```
main() {
   /* γ(b0) = x ≤ 1 */
   /* γ(b1) = x == 1 */
   /* γ(b2) = x ≤ 0 */
   Bool b0, b1, b2 := *, *, *;
   ℓ1 : if (¬b2) then goto ℓ5;
   ℓ2 : if (*) then goto ℓ0;
   ℓ3 : b0, b1, b2 := *, *, *;
   ℓ4 : goto ℓ2;
   ℓ0 : goto ℓ7;
   ℓ5 : if (¬b1) then goto ℓ7;
   ℓ6 : b0, b1, b2 := *, *, *;
   ℓ7 : assert (¬b0);
}
```

Predicate abstraction-based solution framework

$\mathcal{P}$:

```
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);
}
```

$\widehat{\mathcal{B}}$:

```
main() {
   /* γ(b0) = x ≤ 1 */
   /* γ(b1) = x == 1 */
   /* γ(b2) = x ≤ 0 */
   Bool b0, b1, b2 := *, *, *;
   ℓ1 : if (¬b0 ∧ ¬b1 ∧ b2) then goto ℓ5;
   ℓ2 : if (¬b0 ∧ ¬b1 ∧ ¬b2) then goto ℓ0;
   ℓ3 : b0, b1, b2 := *, *, *;
   ℓ4 : goto ℓ2;
   ℓ0 : goto ℓ7;
   ℓ5 : if (¬b1) then goto ℓ7;
   ℓ6 : b0, b1, b2 := *, *, *;
   ℓ7 : assert (¬b0);
}
```

Predicate abstraction-based solution framework

$\widehat{\mathcal{P}}$:

```
main() {
  int x;
  ℓ1 : if (x ≤ 1 ∨ x == 1 ∨ x > 0)
  ℓ2 :    while (x ≤ 1 ∨ x == 1 ∨ x ≤ 0) {
  ℓ3 :      x := x + 2;
  ℓ4 :      skip;
          }
       else
  ℓ5 : if (x == 1)
  ℓ6 :    x := x − 1;
  ℓ7 : assert (x > 1);
}
```

$\widehat{\mathcal{B}}$:

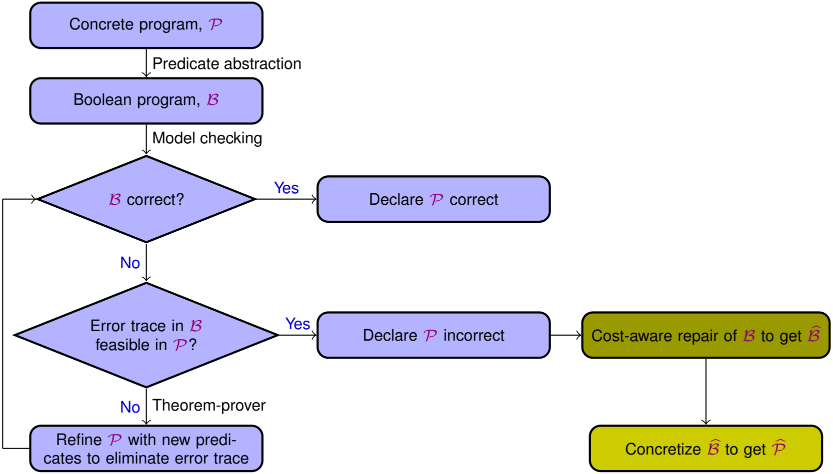
```
main() {
   /* γ(b0) = x ≤ 1 */
   /* γ(b1) = x == 1 */
   /* γ(b2) = x ≤ 0 */
   Bool b0, b1, b2 := *, *, *;
   ℓ1 : if (¬b0 ∧ ¬b1 ∧ b2)
        then goto ℓ5;
   ℓ2 : if (¬b0 ∧ ¬b1 ∧ ¬b2)
        then goto ℓ0;
   ℓ3 : b0, b1, b2 := *, *, *;
   ℓ4 : goto ℓ2;
   ℓ0 : goto ℓ7;
   ℓ5 : if (¬b1) then goto ℓ7;
   ℓ6 : b0, b1, b2 := *, *, *;
   ℓ7 : assert (¬b0);
}
```

Predicate abstraction-based solution framework

$\widehat{\mathcal{P}}$:

```
main() {
   int x;
   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);
}
```

Predicate abstraction-based solution framework



Cost-aware repair of Boolean programs

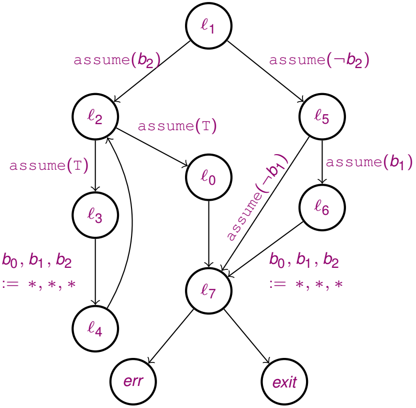
- Adapt method of inductive assertions
- Formulate SMT query — cost-aware repairability condition
- If query is satisfiable, extract $\widehat{\mathcal{B}}$ and proof of correctness.

Review: Method of inductive assertions

$\mathcal{B}$:

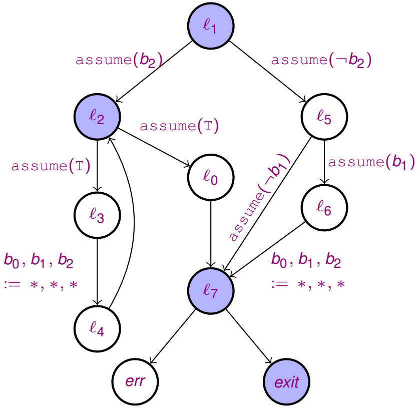
```
main() {
   /* γ(b0) = x ≤ 1 */
   /* γ(b1) = x == 1 */
   /* γ(b2) = x ≤ 0 */
   Bool b0, b1, b2 := *, *, *;
   ℓ1 : if (¬b2) then goto ℓ5;
   ℓ2 : if (*) then goto ℓ0;
   ℓ3 : b0, b1, b2 := *, *, *;
   ℓ4 : goto ℓ2;
   ℓ0 : goto ℓ7;
   ℓ5 : if (¬b1) then goto ℓ7;
   ℓ6 : b0, b1, b2 := *, *, *;
   ℓ7 : assert (¬b0);
}
```

Transition graph:



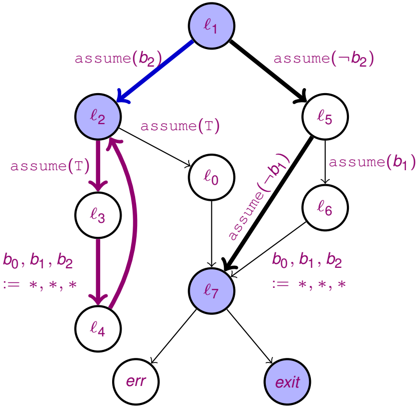
Review: Method of inductive assertions

- Choose cut-points



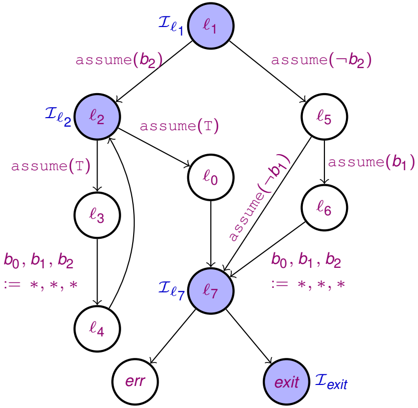
Review: Method of inductive assertions

- Choose cut-points
- Enumerate verification paths



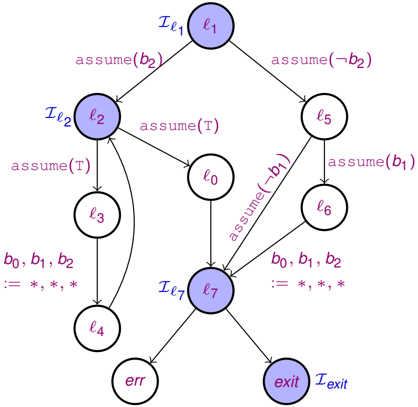
Review: Method of inductive assertions

- Choose cut-points
- Enumerate verification paths
- Attach inductive assertions to cut-points



Review: Method of inductive assertions

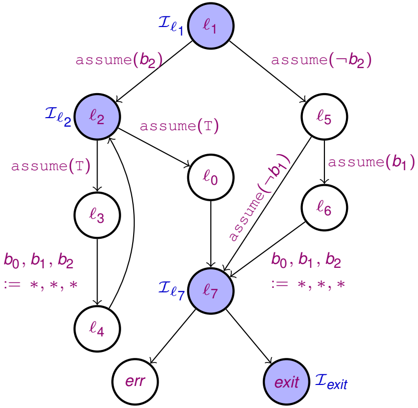
- Choose cut-points
- Enumerate verification paths (denoted π)
- Attach inductive assertions to cut-points
- For each π, formulate $VC(\pi)$
  - $VC(\pi) \sim \langle \mathcal{I}_{\ell} \rangle \; stmt(\pi) \; \langle \mathcal{I}_{\ell'} \rangle$



Review: Method of inductive assertions

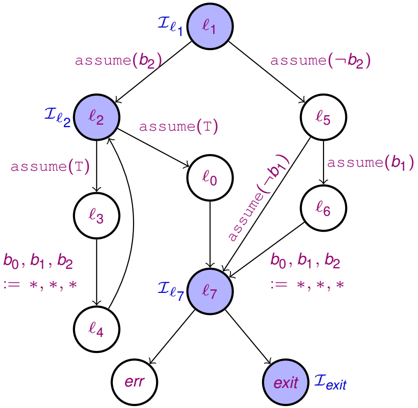
- Choose cut-points
- Enumerate verification paths
- Attach inductive assertions to cut-points
- For each π, formulate $VC(\pi)$

$\mathcal{B}$ is (partially) correct if:
$\exists \mathcal{I}_{\ell_1} \ldots \mathcal{I}_{exit} \, \forall Var : \bigwedge_{\pi} VC(\pi)$



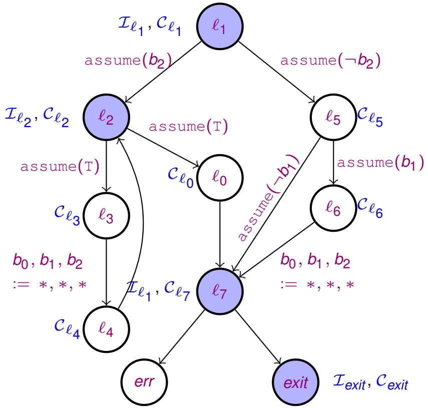
Cost-aware repairability conditions

- Choose cut-points
- Enumerate verification paths
- Attach inductive assertions to cut-points



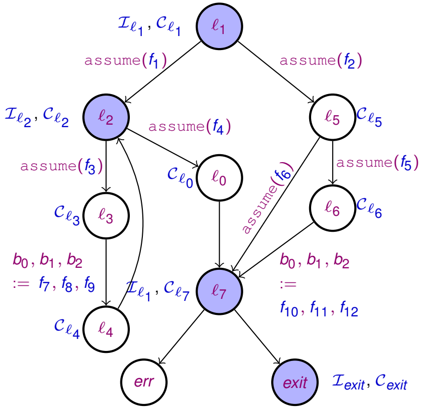
Cost-aware repairability conditions

- Choose cut-points
- Enumerate verification paths
- Attach inductive assertions to cut-points
- Attach costs to locations



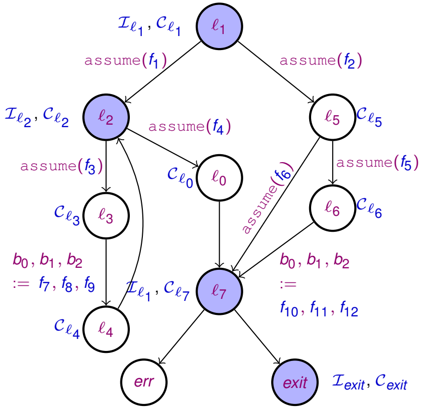
Cost-aware repairability conditions

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



Cost-aware repairability conditions

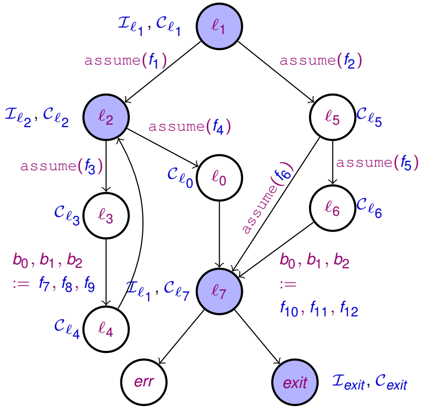
- Choose cut-points
- Enumerate verification paths
- Attach inductive assertions to cut-points
- Attach costs to locations
- Expressions → unknown functions
- For each π, formulate $CRC(\pi)$



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 π, formulate $CRC(\pi)$

$\mathcal{B}$ is repairable within budget δ if:
$\exists Unknown\ \forall Var : \mathcal{C}_{exit} \leq \delta \wedge \bigwedge_{\pi} CRC(\pi) \wedge AssumeConstraints$



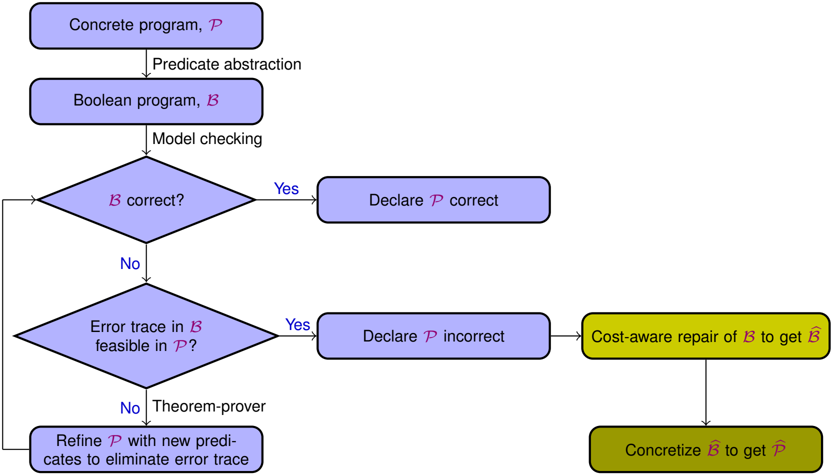
Key result

Given:

- incorrect Boolean program $\mathcal{B}$, annotated with assertions
- $\mathcal{U} = \{id, \texttt{assign} \mapsto \texttt{assign}, \texttt{assign} \mapsto \texttt{skip}, \texttt{assume} \mapsto \texttt{assume}, \texttt{call} \mapsto \texttt{call}, \texttt{call} \mapsto \texttt{skip}\}$.
- cost function c
- repair budget δ

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

Predicate abstraction-based solution framework



Concretization of $\widehat{\mathcal{B}}$

- $\Gamma(\texttt{skip}) = \texttt{skip}$
- $\Gamma(\texttt{if}\,(g)) = \texttt{if}\,(\gamma(g))$
- $\Gamma(\texttt{while}\,(g)) = \texttt{while}\,(\gamma(g))$
- $\Gamma(\texttt{call}\ F_j(e_1, \ldots, e_k) = \texttt{call}\ F_j(\gamma(e_1), \ldots, \gamma(e_k))$
- 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 $expr_1, \ldots, expr_q$ are models for $f_1, \ldots, f_q$: $v_1, \ldots, v_q := expr_1, \ldots, expr_q \in \Gamma(b_1, \ldots, b_r := e_1, \ldots, e_r)$
  - Can use templates for $f_1, \ldots, f_q$

Concretization of $\widehat{\mathcal{B}}$

- $\Gamma(\texttt{skip}) = \texttt{skip}$
- $\Gamma(\texttt{if}\,(g)) = \texttt{if}\,(\gamma(g))$
- $\Gamma(\texttt{while}\,(g)) = \texttt{while}\,(\gamma(g))$
- $\Gamma(\texttt{call}\ F_j(e_1, \ldots, e_k) = \texttt{call}\ F_j(\gamma(e_1), \ldots, \gamma(e_k))$
- 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 $expr_1, \ldots, expr_q$ are models for $f_1, \ldots, f_q$: $v_1, \ldots, v_q := expr_1, \ldots, expr_q \ \in \ \Gamma(b_1, \ldots, b_r := e_1, \ldots, e_r)$
  - Can use templates for $f_1, \ldots, f_q$

Experiments

```
handmade1 :
int main() {
    int x;
    ℓ1 : while (x < 0)
    ℓ2 :     x := x + 1;
    ℓ3 : assert (x > 0);
}
```

```
handmade1^ :
int main() {
    int x;
    ℓ1 : while (x ≤ 0)
    ℓ2 :     x := x + 1;
    ℓ3 : 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);
}
```

```
handmade2 (with hat) :
int main() {
    int x;
    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

```
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);
}
```

```
necex6 (hat) :
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

```
necex14 :
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;
         }
}
```

```
necex14 (repaired):
int main() {
    int x, y;
    int a[10];
    ℓ1 : x := 1U;
    ℓ2 : while (x ≤ 10U) {
    ℓ3 :     y := 10 − x;
    ℓ4 :     assert (y ≥ 0 ∧ y < 10);
    ℓ5 :     a[y] := − 1;
    ℓ6 :     x := x + 1;
         }
}
```

Experiments

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

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 $\widehat{\mathcal{B}}$ and concretization
- Compute weakest inductive assertions, least restrictive $\widehat{\mathcal{B}}$
- Other update schemas

Thank you.