Almost-Correct Specifications:
A Modular Semantic Framework for Assigning Confidence to Warnings

Sam Blackshear
University of Colorado Boulder

Shuvendu K. Lahiri
Microsoft Research Redmond
False Positives Hurt Tool Adoption

Program with assertions ➞ Static assertion checker
False Positives Hurt Tool Adoption

Program with assertions → Static assertion checker → 
Interesting warnings
“Stupid false positives”
False Positives Hurt Tool Adoption

Program with assertions → Static assertion checker → Interesting warnings
“Stupid false positives”
False Positives Hurt Tool Adoption

Program with assertions → Static assertion checker → Interesting warnings

"Stupid false positives"
False Positives Hurt Tool Adoption

Program with assertions → Static assertion checker → Interesting warnings

"Stupid false positives"

Monday, July 8, 13
Program with assertions $\rightarrow$ Static assertion checker $\rightarrow$ Interesting warnings

“‘Stupid false positives’

‘The initial reports matter inordinately; if the first $N$ reports are false positives ($N=3$?), users tend to utter variations on ‘This tool sucks.’ Furthermore, you never want an embarrassing false positive. A stupid false positive implies the tool is stupid.’

- Bessey et al. CACM ’10 (Coverity paper)
Prioritizing Interesting Warnings Can Placate Users
Prioritizing Interesting Warnings Can Placate Users

Goal: warning control knob
Prioritizing Interesting Warnings Can Placate Users

Goal: warning control knob

High-confidence warnings

All warnings
Prioritizing Interesting Warnings Can Placate Users

Goal: warning control knob

High-confidence warnings

warning confidence metric

All warnings
Prioritizing Interesting Warnings Can Placate Users

Goal: warning control knob

Want confidence metric that is:

High-confidence warnings

All warnings

warning confidence metric
Prioritizing Interesting Warnings Can Placate Users

Goal: warning control knob

High-confidence warnings

Want confidence metric that is:

- General (not tool or warning specific)

All warnings

warning confidence metric
Prioritizing Interesting Warnings Can Placate Users

Goal: warning control knob

High-confidence warnings

warning confidence metric

All warnings

Want confidence metric that is:

- General (not tool or warning specific)

- Semantic (not statistics- or sampling-based)
Prioritizing Interesting Warnings Can Placate Users

Goal: warning control knob

High-confidence warnings

All warnings

Want confidence metric that is:

- General (not tool or warning specific)
- Semantic (not statistics- or sampling-based)
- Local/context-independent (not interprocedural)
Prioritizing Interesting Warnings Can Placate Users

Goal: warning control knob

- High-confidence warnings
- All warnings

Want confidence metric that is:

- General (not tool or warning specific)
- Semantic (not statistics- or sampling-based)
- Local/context-independent (not interprocedural)

Post-processing step for any static assertion checker (not a new static analysis)
Realizing a Warning Confidence Metric
Realizing a Warning Confidence Metric

Define confidence in warning using the specification needed to suppress the warning
Realizing a Warning Confidence Metric

Define confidence in warning using the specification needed to suppress the warning

• Many “stupid” warnings caused by missing or implicit specifications (open programs, imprecise interprocedural analysis, etc.)
Realizing a Warning Confidence Metric

Define confidence in warning using the specification needed to suppress the warning

- Many “stupid” warnings caused by missing or implicit specifications (open programs, imprecise interprocedural analysis, etc.)

- If a simple specification suppresses the warning, assume missing spec is the cause and rate as low confidence
Realizing a Warning Confidence Metric

Define confidence in warning using the specification needed to suppress the warning

• Many "stupid" warnings caused by missing or implicit specifications (open programs, imprecise interprocedural analysis, etc.)

• If a simple specification suppresses the warning, assume missing spec is the cause and rate as low confidence

• If warning can only be suppressed by complex specification, rate as higher confidence
Specification Continuum

Angelic view (e.g. weakest precondition):
try to make assertions pass
Specification Continuum

Angelic view (e.g. weakest precondition):
try to make assertions pass

Demonic view: (e.g. conservative static checker):
try to make assertions fail
Specification Continuum

Angelic view (e.g. weakest precondition):
try to make assertions pass

Key idea: Confidence in assertion failure proportional to the amount of weakening of angelic spec needed to show failure

Demonic view: (e.g. conservative static checker):
try to make assertions fail
Specification Continuum

Angelic view (e.g. weakest precondition):
try to make assertions pass

A1 A2 A3

 ✓ ✓ ✓

Key idea: Confidence in assertion failure
proportional to the *amount of weakening* of angelic spec needed to show failure

Demonic view: (e.g. conservative static checker):
try to make assertions fail
Specification Continuum

Angelic view (e.g. weakest precondition):
try to make assertions pass

A1 A2 A3
✓ ✓ ✓

Key idea: Confidence in assertion failure
proportional to the amount of weakening
of angelic spec needed to show failure

Demonic view: (e.g. conservative static checker):
try to make assertions fail

Monday, July 8, 13
Specification Continuum

Angelic view (e.g. weakest precondition):
try to make assertions pass

A1 A2 A3
✓ ✓ ✓

Key idea: Confidence in assertion failure
proportional to the amount of weakening
of angelic spec needed to show failure

Demonic view: (e.g. conservative static checker):
try to make assertions fail

A1 A2 A3
✓ X ✓

A1 A2 A3
X X ✓
Specification Continuum

Angelic view (e.g. weakest precondition):
try to make assertions pass

Demonic view: (e.g. conservative static checker):
try to make assertions fail

- **A1** **A2** **A3**
  - Key idea: Confidence in assertion failure proportional to the *amount of weakening of angelic spec* needed to show failure

- **A1** **A2** **A3**
  - More confident in failure of A2 than in failure of A1

- **A1** **A2** **A3**

  Demonic view: (e.g. conservative static checker):
  try to make assertions fail
Specification Continuum

Angelic view (e.g. weakest precondition): try to make assertions pass

A1 A2 A3
✓ ✓ ✓

Key idea: Confidence in assertion failure proportional to the amount of weakening of angelic spec needed to show failure

More confident in failure of A2 than in failure of A1

A1 A2 A3
✓ X ✓

A1 A2 A3
X X ✓

Goal: define general method for finding reasonable specifications that show interesting assertion failures.

Demonic view: (e.g. conservative static checker): try to make assertions fail
if (cmd == READ) {
    ...
    free(c);
    free(buf);
}

...
Demonic View

\text{free}(x) := \{ \textbf{assert} \ !\text{freed}[x]; \text{freed}[x] = \text{true}; \} \\

... 
\textbf{if} (\text{cmd} == \text{READ}) \{ \\
  ... \\
  \text{free}(c); \\
  \text{free}(buf); \\
\}

... 
\text{free}(c); \\
\text{free}(buf); \\
\text{return};
Demonic View

\[
\text{free}(x) := \{ \text{assert } \neg \text{freed}[x]; \text{freed}[x] = \text{true}; \} \\
\]

Asserts

...  
if (cmd == READ) {  
  ...
  \text{free}(c); \quad \text{assert } \neg \text{freed}[c]  
  \text{free}(buf); \quad \text{assert } \neg \text{freed}[buf] 
}
...
\text{free}(c); \quad \text{assert } \neg \text{freed}[c]  
\text{free}(buf); \quad \text{assert } \neg \text{freed}[buf]  
\text{return};
Demonic View

```c
free(x) := { assert !freed[x]; freed[x] = true; }
```

Asserts

```c
... if (cmd == READ) {
    ...
    free(c); assert !freed[c]
    free(buf); assert !freed[buf]
    // ERR: missing return
}
...
free(c); assert !freed[c]
free(buf); assert !freed[buf]
return;
```
free(x) := { assert !freed[x]; freed[x] = true; }

pre: true

if (cmd == READ) {
    free(c);  assert !freed[c]
    free(buf);  assert !freed[buf]
    // ERR: missing return
}

return;
free(x) := { assert !freed[x]; freed[x] = true; }

pre: true

... if (cmd == READ) {
  ...
  free(c); assert !freed[c]  \(\times\) cmd = READ ∧ freed[c]
  free(buf); assert !freed[buf]
  // ERR: missing return
}

... free(c); assert !freed[c]
free(buf); assert !freed[buf]
return;
free(x) := { assert !freed[x]; freed[x] = true; }

pre: true

... if (cmd == READ) {
... free(c); assert !freed[c] \( \times \) cmd = READ \( \land \) freed[c]
free(buf); assert !freed[buf] \( \times \) cmd = READ \( \land \) (freed[buf]
// ERR: missing return
\lor c = buf)
}

... free(c); assert !freed[c]
free(buf); assert !freed[buf]
return;
Demonic View

\[
\text{free}(x) := \{ \text{assert} \ \neg \text{freed}[x]; \ \text{freed}[x] = \text{true}; \ \}
\]

\text{pre: true}

... if (cmd == READ) {
    ...
    free(c); \quad \text{assert} \ \neg \text{freed}[c] \quad \times \quad \text{cmd} = \text{READ} \land \neg \text{freed}[c]
    free(buf); \quad \text{assert} \ \neg \text{freed}[buf] \quad \times \quad \text{cmd} = \text{READ} \land (\neg \text{freed}[buf] \lor c = buf)
    // ERR: missing \text{return}
}
...
free(c); \quad \text{assert} \ \neg \text{freed}[c] \quad \times \quad \text{freed}[c] \lor \text{cmd} = \text{READ}
free(buf); \quad \text{assert} \ \neg \text{freed}[buf]
free(x) := { assert !freed[x]; freed[x] = true; }

pre: true

if (cmd == READ) {
  free(c);
  assert !freed[c]
  free(buf);
  assert !freed[buf]
  // ERR: missing return
}

return;
free(x) := { assert !freed[x]; freed[x] = true; }

pre: true

... if (cmd == READ) {
    ...
    free(c); assert !freed[c]  
    free(buf); assert !freed[buf]  
    // ERR: missing return
}

... free(c); assert !freed[c]  
free(buf); assert !freed[buf]  
return;

cmd = READ \land freed[c] 
\lor c = \text{buf}

Demonic specifications show too many assertion failures
... if (cmd == READ) {
  ...
  free(c);       assert !freed[c]
  free(buf);     assert !freed[buf]
  // ERR: missing return
}
...
free(c);       assert !freed[c]
free(buf);     assert !freed[buf]
return;
... if (cmd == READ) {
    ...
    free(c);     assert !freed[c]
    free(buf);   assert !freed[buf]
    // ERR: missing return
}
...
free(c);     assert !freed[c]
free(buf);   assert !freed[buf]
return;
Angelic View

WP - largest set of input states that do not fail an assertion (loop-free)

pre: !freed[buf] \land !freed[c] \land cmd \neq READ \land c \neq buf

... if (cmd == READ) {
  ...
  free(c); assert !freed[c]
  free(buf); assert !freed[buf]
  // ERR: missing return
}

... free(c); assert !freed[c]
free(buf); assert !freed[buf]
return;
Angelic View

WP - largest set of input states that do not fail an assertion (loop-free)

pre: !freed[buf] \land !freed[c] \land cmd \neq READ \land c \neq buf

... if (cmd == READ) {

... free(c);  \textcolor{green}{\textbf{assert} \ \!freed[c]}  \checkmark
free(buf);  \textcolor{green}{\textbf{assert} \ \!freed[buf]}  \checkmark
// ERR: missing \textbf{return}
}

... free(c);  \textcolor{green}{\textbf{assert} \ \!freed[c]}  \checkmark
free(buf);  \textcolor{green}{\textbf{assert} \ \!freed[buf]}  \checkmark
\textbf{return};
Angellic View

WP - largest set of input states that do not fail an assertion (loop-free)

pre: !freed[buf] \land !freed[c] \land cmd \neq \text{READ} \land c \neq buf

... 
if (cmd == \text{READ}) { 
  ...
  free(c);    // ERR: missing return

  assert !freed[c] ✓
  free(buf);  ✓

  return;

Angelic specifications hide all assertion failures.
Contributions

---

Almost-correct specifications: parametric (in set of predicates $Q$) framework for weakening specifications

---

Conservative

Shows no warnings

---

Conservative

Shows all warnings
Contributions

Almost-correct specifications: parametric (in set of predicates \( Q \)) framework for weakening specifications
Contributions

Almost-correct specifications: parametric (in set of predicates $Q$) framework for weakening specifications

Assertion Failures

Semantic inconsistency bugs (SIB’s)
Contributions

Almost-correct specifications: parametric (in set of predicates Q) framework for weakening specifications

Assertion Failures

Semantic inconsistency bugs (SIB’s)

[Engler et al. SOSP ’01, Dillig et al. PLDI ’07, Hoenicke et al. FMSD ’10, Tomb et al. ISSTA ’11]
Contributions

Almost-correct specifications: parametric (in set of predicates \( Q \)) framework for weakening specifications

Assertion Failures

Semantic inconsistency bugs (SIB’s)

[Engler et al. SOSP ’01, Dillig et al. PLDI ’07, Hoenicke et al. FMSD ’10, Tomb et al. ISSTA ’11]

All assertion failures
Contributions

Almost-correct specifications: parametric (in set of predicates $Q$) framework for weakening specifications

Conservative

Atoms(WP)

Abstracting predicate set

Clause quality metrics

Semantic inconsistency bugs (SIB’s)

All assertion failures

All assertion failures
Contributions

Almost-correct specifications: parametric (in set of predicates $Q$) framework for weakening specifications

Q

Assertion Failures

Semantic inconsistency bugs (SIB’s)

Abstracting predicate set

Clause quality metrics

$\emptyset$

All assertion failures
Almost-Correct Specs Idea

WP/Angelic spec
Almost-Correct Specs Idea

true / Demonic spec

WP/Angelic spec
Almost-Correct Specs Idea

true / Demonic spec

WP/Angelic spec

Assertion failure
Almost-Correct Specs Idea

Want to weaken WP over Q enough to see failures

true / Demonic spec

Assertion failure
Almost-Correct Specs Idea

Want to weaken WP over \( \mathcal{Q} \) enough to see failures

true / Demonic spec

Assertion failure

WP/Angelic spec

Rules for weakening:
(1) Spec must allow all code in procedure to be reachable (not too strong)
Almost-Correct Specs Idea

Want to weaken WP over $Q$ enough to see failures.

true / Demonic spec

Assertion failure

WP/Angelic spec

Rules for weakening:

1. Spec must allow all code in procedure to be reachable (not too strong)
2. Spec must show as few assertion failures as possible (not too weak)
Almost-Correct Specs Idea

Want to weaken WP over $\mathcal{Q}$ enough to see failures

Rules for weakening:
(1) Spec must allow all code in procedure to be reachable (not too strong)
(2) Spec must show as few assertion failures as possible (not too weak)
Almost-Correct Specs Idea

Want to weaken WP over $Q$ enough to see failures.

Rules for weakening:
(1) Spec must allow all code in procedure to be reachable (not too strong)
(2) Spec must show as few assertion failures as possible (not too weak)

$true$ / Demonic spec

WP/Angelic spec

Almost-correct spec

Assertion failure
Almost-Correct Specs Algorithm

\[ \phi := \beta_Q(wp(p, \text{true})) \]
Almost-Correct Specs Algorithm

\[ \phi := \beta_Q(\wp(p, \text{true})) \]

Weakest underapproximation of WP w.r.t set of predicates \( Q \) (enables abstraction by \( Q \))
Almost-Correct Specs Algorithm

\[ \phi := \beta_Q(wp(p, \text{true})) \]

\( \phi \) creates dead code?
Almost-Correct Specs Algorithm

\[ \phi := \beta_Q(wp(p, \text{true})) \]

\( \phi \) creates dead code?

“Is \( \phi \) too strong?” Can be replaced with other definitions (e.g. does not allow observed concrete value)
Almost-Correct Specs Algorithm

\[ \phi := \beta_Q(wp(p, true)) \]

minimally weaken over \( Q \) to \( \phi' \)

"Is \( \phi \) too strong?" Can be replaced with other definitions (e.g. does not allow observed concrete value)

Yes

\( \phi \) creates dead code?
Almost-Correct Specs Algorithm

\[ \phi := \beta_Q(wp(p, \text{true})) \]

\[ \phi \text{ creates dead code?} \]

Will cause some assertions to fail

\[ \text{minimally weaken over } Q \text{ to } \phi' \]

Yes
Almost-Correct Specs Algorithm

\[ \phi := \beta_Q(wp(p, \text{true})) \]

- \( \phi \) creates dead code?

- Will cause some assertions to fail

Can accomplish by "dropping" clauses from maximal clause CNF representation of WP over \( Q \)

minimally weaken over \( Q \) to \( \phi' \)

Yes
Almost-Correct Specs Algorithm

\[ \phi := \beta_Q(wp(p, \text{true})) \]

\[ \phi := \phi' \]

\[ \phi \text{ creates dead code?} \]

Will cause some assertions to fail

Can accomplish by “dropping” clauses from maximal clause CNF representation of WP over \( Q \)
Almost-Correct Specs Algorithm

\[ \phi := \beta_Q(wp(p, \text{true})) \]

\[ \phi := \phi' \]

\( \phi \) creates dead code?

Yes

minimally weaken over \( Q \) to \( \phi' \)

No

Save \( \phi \) as candidate spec
Almost-Correct Specs Algorithm

\[ \phi := \beta_Q(wp(p, \text{true})) \]

\[ \phi := \phi' \]

\[ \phi \text{ creates dead code?} \]

Yes

minimally weaken over \( Q \) to \( \phi' \)

No

Save \( \phi \) as candidate spec

Output \( \phi \)'s that fail the smallest number of assertions as almost-correct specifications
Almost-Correct Specs Reveal Bugs

WP: \(!\text{freed}[\text{buf}] \land !\text{freed}[\text{c}] \land \text{cmd} \neq \text{READ} \land c \neq \text{buf}\)

... 

\textbf{if} (\text{cmd} == \text{READ}) \{ 

\dots

\text{free}(\text{c}); \quad \textbf{assert} \ !\text{freed}[\text{c}] \quad \checkmark 
\text{free}(\text{buf}); \quad \textbf{assert} \ !\text{freed}[\text{buf}] \quad \checkmark 

// ERR: missing \text{return}

\}

... 

\text{free}(\text{c}); \quad \textbf{assert} \ !\text{freed}[\text{c}] \quad \checkmark 
\text{free}(\text{buf}); \quad \textbf{assert} \ !\text{freed}[\text{buf}] \quad \checkmark 

\text{return};
Almost-Correct Specs Reveal Bugs

WP: \(!\text{freed[buf]} \land \neg \text{freed[c]} \land \text{cmd} \neq \text{READ} \land \text{c} \neq \text{buf}\)

...  
if (cmd == \text{READ}) {  
...  
  free(c);  
  assert \(!\text{freed[c]}\)  
  assert \(!\text{freed[buf]}\)  
  // ERR: missing \text{return}  
}
...  
free(c);  
assert \(!\text{freed[c]}\)  
free(buf);  
assert \(!\text{freed[buf]}\)  
return;  

Creates dead code
Almost-Correct Specs Reveal Bugs

Weaken using almost-correct spec algorithm

WP: !freed[buf] \land !freed[c] \land \text{cmd} \neq \text{READ} \land c \neq buf

... if (cmd == READ) {
...
free(c); \quad \textbf{assert} \quad !\text{freed}[c]
free(buf); \quad \textbf{assert} \quad !\text{freed}[buf]
// ERR: missing return
}

... free(c); \quad \textbf{assert} \quad !\text{freed}[c]
free(buf); \quad \textbf{assert} \quad !\text{freed}[buf]
return;

Creates dead code
Almost-Correct Specs Reveal Bugs

Weaken using almost-correct spec algorithm

WP: \(!\text{freed}[\text{buf}] \land \neg \text{freed}[\text{c}] \land \text{cmd} \neq \text{READ} \land \text{c} \neq \text{buf}\)

... if (cmd == READ) {
    ...
    free(c); \textbf{assert} \neg \text{freed}[\text{c}] \checkmark
    free(buf); \textbf{assert} \neg \text{freed}[\text{buf}] \checkmark
    // ERR: missing return
}
... free(c); \textbf{assert} \neg \text{freed}[\text{c}] \checkmark
free(buf); \textbf{assert} \neg \text{freed}[\text{buf}] \checkmark
return;
Weaken using almost-correct spec algorithm

Weakened spec allows assertion failures.
Almost-Correct Specs Reveal Bugs

Weakening WP to almost-correct spec reveals semantic inconsistency bugs (SIB’s) without adding many false alarms.
Bad News: Not All Bugs Are SIB’s
Bad News: Not All Bugs Are SIB’s

```c
x.f = 1;
y = getObj();
if (foo()) {
    y.f = 2;
} else {
    if (y != NULL) {
        y.f = 3;
    } else {
    }
}
```
Bad News: Not All Bugs Are SIB’s

```c
x.f = 1;
y = getObj();
if (foo()) {
    y.f = 2;
} else {
    if (y != NULL) {
        y.f = 3;
    } else {
```

```c
    assert x != NULL
    assert y != NULL
    assert y != NULL
}
```

```c
}
```

```c

Monday, July 8, 13
```
Bad News: Not All Bugs Are SIB’s

```c
x.f = 1;
y = getObj();
if (foo()) {
    // ERR: y may be NULL
    y.f = 2;
} else {
    if (y != NULL) {
        y.f = 3;
    } else {
    }
}
```
Bad News: Not All Bugs Are SIB’s

WP: \( x \neq \text{NULL} \land (\text{ret\_foo} \Rightarrow \text{ret\_getObj} \neq \text{NULL}) \)

\[
x.f = 1;
\]

\[
y = \text{getObj}();
\]

\[
\text{if} (\text{foo}()) \{
    // ERR: y may be NULL
    y.f = 2;
\}
\]

\[
\text{assert } x \neq \text{NULL}
\]

\[
\text{assert } y \neq \text{NULL}
\]

\[
\text{if} (y \neq \text{NULL}) \{
    y.f = 3;
\}
\]

\[
\text{assert } y \neq \text{NULL}
\]

\[
\text{else} \{
\}
\]

\[
\text{else} \{
\}
\]
Bad News: Not All Bugs Are SIB’s

WP: \( x \neq \text{NULL} \land (\text{ret\_foo} \Rightarrow \text{ret\_getObj} \neq \text{NULL}) \)

\[
x.f = 1; \\
y = \text{getObj}(); \\
\text{if (foo())} \\
\quad \text{// ERR: y may be NULL} \\
\quad y.f = 2; \\
\text{else} \\
\quad \text{if (y \neq \text{NULL})} \\
\quad \quad y.f = 3; \\
\quad \text{else} \\
\]

\[
\text{assert } x \neq \text{NULL} \checkmark \\
\text{assert } y \neq \text{NULL} \checkmark \\
\text{assert } y \neq \text{NULL} \checkmark 
\]
Bad News: Not All Bugs Are SIB’s

WP: \( x \neq \text{NULL} \land (\text{ret\_foo} \Rightarrow \text{ret\_getObj} \neq \text{NULL}) \)

```c
x.f = 1;
y = \text{getObj}();
if (\text{foo}()) {
    // ERR: y may be NULL
    y.f = 2;
} else {
    if (y \neq \text{NULL}) {
        y.f = 3;
    } else {}
}
```

WP assumes correlation between return values of \text{getObj}() and \text{foo}() to hide bug without creating dead code.

- \texttt{assert} \( x \neq \text{NULL} \checkmark \)
- \texttt{assert} \( y \neq \text{NULL} \checkmark \)
- \texttt{assert} \( y \neq \text{NULL} \checkmark \)
Contributions

Almost-correct specifications: parametric (in set of predicates \( \mathcal{Q} \)) framework for weakening specifications

\[ \begin{align*}
\text{Conservative} & \quad \text{Atoms(WP)} \\
\downarrow & \\
\text{Abstracting predicate set} & \\
\downarrow & \\
\text{Clause quality metrics} & \\
\varnothing & \\
\end{align*} \]

Assertion Failures

Semantic inconsistency bugs (SIB's)

[Engler et al. SOSP '01, Dillig et al. PLDI '07, Tomb et al. ISSTA '11, Bertolini et al. VSTTE '12]

All assertion failures
Taking angelic away power from WP

WP: \( x \neq \text{NULL} \land (\text{ret\_foo} \Rightarrow \text{ret\_getObj} \neq \text{NULL}) \)
Taking angelic away power from WP

WP: $x \neq \text{NULL} \land (\text{ret}_\text{foo} \Rightarrow \text{ret}_\text{getObj} \neq \text{NULL})$

Want to exclude clauses like $\text{ret}_\text{foo} \Rightarrow \text{ret}_\text{getObj} \neq \text{NULL}$
Taking angelic away power from WP

\[ \text{WP: } x \neq \text{NULL} \land (\text{ret\_foo} \implies \text{ret\_getObj} \neq \text{NULL}) \]

Want to exclude clauses like \( \text{ret\_foo} \implies \text{ret\_getObj} \neq \text{NULL} \)

We can accomplish this by:

- Performing predicate abstraction over set of predicates

\[ \bigwedge \]
Taking angelic away power from WP

WP: \( x \neq \text{NULL} \land (\text{ret}_\text{foo} \Rightarrow \text{ret}_\text{getObj} \neq \text{NULL}) \)

Want to exclude clauses like \( \text{ret}_\text{foo} \Rightarrow \text{ret}_\text{getObj} \neq \text{NULL} \)

We can accomplish this by:

- Performing predicate abstraction over set of predicates

  - Ex: treat conditionals as \( * \) (non-deterministic-choice)
Taking angelic away power from WP

\[ \text{WP: } x \neq \text{NULL} \land (\text{ret}_\text{foo} \Rightarrow \text{ret}_\text{getObj} \neq \text{NULL}) \]

Want to exclude clauses like \( \text{ret}_\text{foo} \Rightarrow \text{ret}_\text{getObj} \neq \text{NULL} \)

We can accomplish this by:

- Performing predicate abstraction over set of predicates

- Ex: treat conditionals as * (non-deterministic-choice)

- Ex: treat return values as *
Taking angelic away power from WP

WP: \( x \neq \text{NULL} \land (\text{ret\_foo} \Rightarrow \text{ret\_getObj} \neq \text{NULL}) \)

Want to exclude clauses like \( \text{ret\_foo} \Rightarrow \text{ret\_getObj} \neq \text{NULL} \)

We can accomplish this by:

- Performing predicate abstraction over set of predicates \( \mathcal{Q} \)
  - Ex: treat conditionals as * (non-deterministic-choice)
  - Ex: treat return values as *

- Excluding clauses based on quality metric (e.g. limit number of disjunctions), see paper
Abstracting the WP

WP: $x \neq \text{NULL} \land (\text{ret\_foo} \Rightarrow \text{ret\_getObj} \neq \text{NULL})$
Abstracting the WP

WP: $x \neq \text{NULL} \land (\text{ret}_{\text{foo}} \Rightarrow \text{ret}_{\text{getObj}} \neq \text{NULL})$

(1) Collect atomic predicates $\mathcal{Q}$ that appear in WP
Abstracting the WP

WP: \( x \neq \text{NULL} \land (\text{ret\_foo} \Rightarrow \text{ret\_getObj} \neq \text{NULL}) \)

(1) Collect atomic predicates \( \Diamond \) that appear in WP

\{ \( x \neq \text{NULL}, \text{ret\_getObj} \neq \text{NULL}, \text{ret\_foo} \) \}
Abstracting the WP

WP: \( x \neq \text{NULL} \land (\text{ret\_foo} \Rightarrow \text{ret\_getObj} \neq \text{NULL}) \)

(1) Collect atomic predicates \( Q \) that appear in WP

\[
\{ x \neq \text{NULL}, \text{ret\_getObj} \neq \text{NULL}, \text{ret\_foo} \}
\]

(2) Abstract \( Q \) (e.g., treat conditionals as *)
Abstracting the WP

WP: $x \neq \text{NULL} \land (\text{ret}\_\text{foo} \Rightarrow \text{ret}\_\text{getObj} \neq \text{NULL})$

(1) Collect atomic predicates $Q$ that appear in WP

\{ $x \neq \text{NULL}, \text{ret}\_\text{getObj} \neq \text{NULL}, \text{ret}\_\text{foo}$ \}

(2) Abstract $Q$ (e.g., treat conditionals as $*$)
Abstracting the WP

WP: \( x \neq \text{NULL} \land (\text{ret\_foo} \Rightarrow \text{ret\_getObj} \neq \text{NULL}) \)

1) Collect atomic predicates \( Q \) that appear in WP

\[ \{ x \neq \text{NULL}, \text{ret\_getObj} \neq \text{NULL}, \text{ret\_foo} \} \]

2) Abstract \( Q \) (e.g., treat conditionals as *)

3) Compute weakest underapproximation of WP w.r.t \( Q \)

\[ \beta_Q(x \neq \text{NULL} \land (\text{ret\_foo} \Rightarrow \text{ret\_getObj} \neq \text{NULL})) \]
Abstracting the WP

WP: \( x \neq \text{NULL} \land (\text{ret\_foo} \Rightarrow \text{ret\_getObj} \neq \text{NULL}) \)

(1) Collect atomic predicates \( Q \) that appear in WP

\[
\{ x \neq \text{NULL}, \text{ret\_getObj} \neq \text{NULL}, \text{ret\_foo} \}
\]

(2) Abstract \( Q \) (e.g., treat conditionals as *)

(3) Compute weakest underapproximation of WP w.r.t \( Q \)

\[
\beta_Q(x \neq \text{NULL} \land (\text{ret\_foo} \Rightarrow \text{ret\_getObj} \neq \text{NULL})) = x \neq \text{NULL} \land \text{ret\_getObj} \neq \text{NULL}
\]
Abstracting the WP

WP: \( x \neq \text{NULL} \land (\text{ret\_foo} \Rightarrow \text{ret\_getObj} \neq \text{NULL}) \)

(1) Collect atomic predicates \( Q \) that appear in WP

\[
\{ x \neq \text{NULL}, \text{ret\_getObj} \neq \text{NULL}, \text{ret\_foo} \}
\]

(2) Abstract \( Q \) (e.g., treat conditionals as * \)

(3) Compute weakest underapproximation of WP w.r.t \( Q \)

\[
\beta_Q(x \neq \text{NULL} \land (\text{ret\_foo} \Rightarrow \text{ret\_getObj} \neq \text{NULL}))
\]

\[
= x \neq \text{NULL} \land \text{ret\_getObj} \neq \text{NULL}
\]

Can now compute almost-correct spec from abstract WP
Abstract SIB’s

WP: $x \neq \text{NULL} \land \text{ret\_getObj} \neq \text{NULL}$

```c
x.f = 1;
y = getObj();
if (foo()) {
   // ERR: y may be NULL
   y.f = 2;
} else {
   if (y != \text{NULL}) {
      y.f = 3;
   } else {
   }
}
```

- `assert x != \text{NULL} \checkmark`
- `assert y != \text{NULL} \checkmark`
- `assert y != \text{NULL} \checkmark`
Abstract SIB’s

WP: \(x \neq \text{NULL} \land \text{ret} \_ \text{getObj} \neq \text{NULL}\)

\begin{align*}
x.f &= 1; \\
y &= \text{getObj}(); \\
\text{if} \ (\text{foo}()) \ {\{} \\
&\quad \text{// ERR: y may be NULL} \\
y.f &= 2; \\
\text{\{} \text{else} \ {\}} \\
&\quad \text{if} \ (y \neq \text{NULL}) \ {\{} \\
&\quad \quad y.f = 3; \\
&\quad \text{\{} \text{else} \ {\}} \\
&\quad \text{\{}} \\
\end{align*}

\text{assert} \ x \neq \text{NULL \ checkmark}

\text{assert} \ y \neq \text{NULL \ checkmark}

\text{assert} \ y \neq \text{NULL \ checkmark}

\text{Abstract WP creates dead code}
Weaken to eliminate dead code and reveal bug, as before

WP: \( x \neq \text{NULL} \land \text{ret\_getObj} \neq \text{NULL} \)

\[
x.f = 1;
\]

\[
y = \text{getObj}();
\]

\[
\text{if} \ (\text{foo}()) \ {\}
\]

// ERR: \( y \) may be \text{NULL}

\[
y.f = 2;
\]

\[
\text{else} \ {\}
\]

\[
\text{if} \ (y \neq \text{NULL}) \ {\}
\]

\[
y.f = 3;
\]

\[
\text{else} \ {\}
\]

Abstract WP creates dead code
Abstract SIB's

Weaken to eliminate dead code and reveal bug, as before

\[ WP: x \neq \text{NULL} \land \text{ret\_getObj} \neq \text{NULL} \]

\[
\begin{align*}
x.f &= 1; \\
y &= \text{getObj}(); \\
\text{if} (\text{foo}()) \{ \\
\quad \text{// ERR: y may be NULL} \\
\quad y.f &= 2; \\
\} \text{ else } \{ \\
\quad \text{if} (y \neq \text{NULL}) \{ \\
\quad \quad y.f &= 3; \\
\quad \} \text{ else } \{} \\
\} \\
\]

\[
\begin{align*}
\text{assert } x \neq \text{NULL} \checkmark \\
\text{assert } y \neq \text{NULL} \checkmark \\
\text{assert } y \neq \text{NULL} \checkmark \\
\end{align*}
\]

Abstract WP creates dead code
Abstract SIB’s

Weaken to eliminate dead code and reveal bug, as before

WP: \( x \neq \text{NULL} \land \text{ret_getObj} \neq \text{NULL} \)

\[
x.f = 1; \\
y = \text{getObj}(); \\
\text{if (foo())} \\
\quad \text{// ERR: y may be NULL} \\
\quad y.f = 2; \\
\text{else} \\
\quad \text{if (y \neq \text{NULL})} \\
\quad \quad y.f = 3; \\
\text{else} \\
\]

assert \( x \neq \text{NULL} \)

assert \( y \neq \text{NULL} \)

assert \( y \neq \text{NULL} \)
Abstract SIB’s

Weaken to eliminate dead code and reveal bug, as before

WP: $x \neq \text{NULL} \land \text{ret}_\text{getObj} \neq \text{NULL}$

```c
x.f = 1;
y = getObj();
if (foo()) {
    // ERR: y may be NULL
    y.f = 2;
} else {
    if (y != NULL) {
        y.f = 3;
    } else {
    }
}
```

- assert $x \neq \text{NULL}$ ✓
- assert $y \neq \text{NULL}$ ✗
- assert $y \neq \text{NULL}$ ✓
Abstract SIB’s

Weaken to eliminate dead code and reveal bug, as before

WP: \( x \neq \text{NULL} \land \text{ret} \_\text{getObj} \neq \text{NULL} \)

\[
x.f = 1; \\
y = \text{getObj}(); \\
\text{if} (\text{foo}()) \{
    \text{assert} \ x \neq \text{NULL} \checkmark
    \text{y.f} = 2; \\
}\text{else} \{
\text{assert} \ y \neq \text{NULL} \xmark
\}
\text{if} (y \neq \text{NULL}) \{
    \text{assert} \ y \neq \text{NULL} \checkmark
    y.f = 3;
\} \text{else} \{
\}
\}

Abstractions let us see more assertion failures, but with less confidence (had to assume return values not correlated)
Evaluation

Question: does this strategy actually prioritize interesting alarms?
Evaluation

Question: does this strategy actually prioritize interesting alarms?

(1) Define “interesting” as true alarm, use SAMATE benchmarks (all assertions are annotated as safe or unsafe)
Question: does this strategy actually prioritize interesting alarms?

(1) Define “interesting” as true alarm, use SAMATE benchmarks (all assertions are annotated as safe or unsafe)

(2) Define interesting as “requiring complex validation”, sample alarms to examine from run on 1M+ lines of Windows code (see paper)
Experimental Configuration
Experimental Configuration

Client: null pointer dereference in C
Experimental Configuration

Client: null pointer dereference in C
All loops unrolled twice
Experimental Configuration

Client: null pointer dereference in C

All loops unrolled twice

<table>
<thead>
<tr>
<th>Alias</th>
<th>Havoc returns</th>
<th>Ignore conditionals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conc</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>(A_0)</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>(A_1)</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>(A_2)</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
Client: null pointer dereference in C

All loops unrolled twice

Four configurations

<table>
<thead>
<tr>
<th>Alias</th>
<th>Havoc returns</th>
<th>Ignore conditionals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conc</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>$A_0$</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>$A_1$</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>$A_2$</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
### Experimental Configuration

**Client:** null pointer dereference in C

**All loops unrolled twice**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Havoc returns</th>
<th>Ignore conditionals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conc</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>$A_0$</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>$A_1$</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>$A_2$</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

**Four configurations**

- Same as A1

---

Monday, July 8, 13
Experimental Configuration

Client: null pointer dereference in C

All loops unrolled twice

Four configurations
Same as A1

One clause quality metric: < k disjunctions for k = 1 to 3

<table>
<thead>
<tr>
<th>Alias</th>
<th>Havoc returns</th>
<th>Ignore conditionals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conc</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>A₀</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>A₁</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>A₂</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
Experimental Configuration

Client: null pointer dereference in C

All loops unrolled twice

Four configurations
  Same as A₁

One clause quality metric: < k disjunctions for k = 1 to 3

<table>
<thead>
<tr>
<th>Alias</th>
<th>Havoc returns</th>
<th>Ignore conditionals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conc</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>A₀</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>A₁</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>A₂</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
Experimental Configuration

Client: null pointer dereference in C

All loops unrolled twice

Four configurations

Same as A1

One clause quality metric: < k disjunctions for k = 1 to 3

<table>
<thead>
<tr>
<th>Alias</th>
<th>Havoc returns</th>
<th>Ignore conditionals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conc</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>$A_0$</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>$A_1$</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>$A_2$</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

Two abstractions

WP
Boogie
## Effectiveness of Prioritization (SAMATE Benchmarks)

<table>
<thead>
<tr>
<th>Configuration</th>
<th>False Alarm %</th>
<th>True Alarm %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conc</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>A1</td>
<td>0</td>
<td>25</td>
</tr>
<tr>
<td>A2</td>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>100</td>
</tr>
</tbody>
</table>
## Effectiveness of Prioritization (SAMATE Benchmarks)

<table>
<thead>
<tr>
<th>Configuration</th>
<th>False Alarm %</th>
<th>True Alarm %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conc</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>A1</td>
<td>0</td>
<td>25</td>
</tr>
<tr>
<td>A2</td>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>100</td>
</tr>
</tbody>
</table>

What % of reported alarms are false alarms?
# Effectiveness of Prioritization (SAMATE Benchmarks)

<table>
<thead>
<tr>
<th>Configuration</th>
<th>False Alarm %</th>
<th>True Alarm %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conc</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A1</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>A2</td>
<td>1</td>
<td>40</td>
</tr>
</tbody>
</table>

**What % of reported alarms are false alarms?**

**What % of known true alarms did we find (recall)?**

Monday, July 8, 13
Effectiveness of Prioritization (SAMATE Benchmarks)

<table>
<thead>
<tr>
<th>Configuration</th>
<th>False Alarm %</th>
<th>True Alarm %</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A2</td>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td>Conc</td>
<td>0</td>
<td>25</td>
</tr>
</tbody>
</table>

Monday, July 8, 13
## Effectiveness of Prioritization (SAMATE Benchmarks)

<table>
<thead>
<tr>
<th>Configuration</th>
<th>False Alarm %</th>
<th>True Alarm %</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A2</td>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td>Conc</td>
<td>0</td>
<td>20</td>
</tr>
</tbody>
</table>
Effectiveness of Prioritization (SAMATE Benchmarks)

<table>
<thead>
<tr>
<th>Configuration</th>
<th>False Alarm %</th>
<th>True Alarm %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conc</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>Al</td>
<td>0</td>
<td>25</td>
</tr>
</tbody>
</table>
Effectiveness of Prioritization (SAMATE Benchmarks)

<table>
<thead>
<tr>
<th>Configuration</th>
<th>False Alarm %</th>
<th>True Alarm %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conc</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>A1</td>
<td>0</td>
<td>25</td>
</tr>
<tr>
<td>A2</td>
<td>1</td>
<td>40</td>
</tr>
</tbody>
</table>
Effectiveness of Prioritization (SAMATE Benchmarks)

<table>
<thead>
<tr>
<th>Configuration</th>
<th>False Alarm %</th>
<th>True Alarm %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conc</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A1</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>A2</td>
<td>1</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>100</td>
</tr>
</tbody>
</table>

Monday, July 8, 13
# Effectiveness of Prioritization
(SAMATE Benchmarks)

<table>
<thead>
<tr>
<th>Configuration</th>
<th>False Alarm %</th>
<th>True Alarm %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conc</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>A1</td>
<td>0</td>
<td>25</td>
</tr>
<tr>
<td>A2</td>
<td>1</td>
<td>40</td>
</tr>
</tbody>
</table>

Can sacrifice missed alarms for near-zero FP rate.
Future Challenges
Future Challenges

- False positives (macros, abstractions too coarse)
Future Challenges

- False positives (macros, abstractions too coarse)

Explore finer/new/more abstractions
Future Challenges

- False positives (macros, abstractions too coarse)
  Explore finer/new/more abstractions

- Missed true positives (as indicated by SAMATE)
Future Challenges

- False positives (macros, abstractions too coarse)
  Explore finer/new/more abstractions

- Missed true positives (as indicated by SAMATE)
  Use limited interprocedural analysis to debunk some specs/correlations and see more bugs
Future Challenges

- False positives (macros, abstractions too coarse)
  Explore finer/new/more abstractions

- Missed true positives (as indicated by SAMATE)
  Use limited interprocedural analysis to debunk some specs/correlations and see more bugs

- Don’t know what users will think
Future Challenges

- False positives (macros, abstractions too coarse)
  Explore finer/new/more abstractions

- Missed true positives (as indicated by SAMATE)
  Use limited interprocedural analysis to debunk some specs/correlations and see more bugs

- Don’t know what users will think
  User study to evaluate whether warnings are interesting to users, involve user in the loop to define Q
Conclusion

Semantic warning control knob
Conclusion

Semantic warning control knob

SIB’s revealed by weakening angelic spec over 🛑

All warnings
Conclusion

Semantic warning control knob

SIB’s revealed by weakening angelic spec over $\mathcal{Q}$

Abstracting $\mathcal{Q}$ to trade off less confidence for more warnings

All warnings