Mixing Type Checking and Symbolic Execution

Khoo Yit Phang (UMD College Park)
Bor-Yuh Evan Chang (CU Boulder)
Jeffrey S. Foster (UMD College Park)

FRACTAL - December 5, 2009
An all too common scenario ...

Oh Verifier, help me prove my program has no bugs.

On line 142, there may be a bug.

Isn’t it obvious this can’t happen!??!

Static verifiers must over-approximate and thus raise false alarms.
False alarm example:
The need for path sensitivity

if (multithreaded) fork();
... statements₁ ...
if (multithreaded) lock();
... statements₂ ...
if (multithreaded) unlock();

This abstraction is too coarse. Standard practice is to re-design it to be precise enough for this example.
Re-design with path sensitivity

\[
\begin{align*}
&\text{if (multithreaded) fork();} \\
&\text{... statements}_1 ... \\
&\text{if (multithreaded) lock();} \\
&\text{... statements}_2 ... \\
&\text{if (multithreaded) unlock();}
\end{align*}
\]

**Bad:** Too much precision leads to **slow**, inefficient analysis

**Bad:** Ad-hoc addition of precision leads to **brittle** analyzers
Observation: Just need precision in select places

```c
if (multithreaded) fork();
... statements_1 ...
if (multithreaded) lock();
... statements_2 ...
if (multithreaded) unlock();
```

```
assume(multithreaded);
```

```
assume(!multithreaded);
```
Approach: Split the program between analyses

if (multithreaded) fork();

... statements_1 ...

if (multithreaded) lock();  coarse

... statements_2 ...

if (multithreaded) unlock();  coarse

precise

coarse

Switch to precise analysis only where needed
Mix is ...

A tunable program analysis that alternates between type inference and symbolic execution

- Standard, off-the-shelf type inference
- Standard, off-the-shelf symbolic execution
- Mixing rules to translate information at block boundaries
Why type inference and symbolic execution?

• Case study of extremes

- Simple, well-understood algorithms
- Hard to imagine how to combine in more intricate ways (e.g., in contrast to combining abstract interpreters)
Outline

- Mixing rules
- Examples and idioms for switching blocks
- Preliminary experience with MIXY, a mixed type qualifier inference engine for C
Type checking

**Typing context**

\[ x : \text{int}, b : \text{bool}, y : \text{int} \]

\[ x + (\text{if } b \text{ then } y \text{ else } 3) \]

\[ : \text{int} \]

Type of the expression

Symbolic execution

**Symbolic context**

\[ \delta ; x = \alpha : \text{int}, b = \beta : \text{bool}, y = \gamma : \text{int} \]

\[ x + (\text{if } b \text{ then } y \text{ else } 3) \]

\[ = (\delta \land \beta ; \alpha + \gamma : \text{int}) \]

\[ = (\delta \land \neg \beta ; \alpha + 3 : \text{int}) \]

Path condition

Symbolic result along the path

Khoo Yit Phang, Bor-Yuh Evan Chang, Jeffrey S. Foster - Mixing Type Checking and Symbolic Execution
Mixing rules: Conservatively translate states

Formalized and proven sound for an ML-like language with references

Mixing rules are not particularly surprising

What may be surprising is that such simple rules with off-the-shelf algorithms yield increased precision in many ways
Outline

• Mixing rules

• Examples and idioms for switching blocks

• Preliminary experience with MIXY, a mixed type qualifier inference engine for C
Flow, path, and context sensitivity

x := 1; ... ; x := “hello”; ... ;

let pred n = if n = 0 then “err” else n - 1
in ... + (pred 3)

Static type checking for dynamically-typed code
Local refinement

```java
if (x > 0) {
    x: posint ... type
}
else if (x == 0) {
    x: zero ... type
}
else {
    x: negint ... type
}
```
Abstraction during symbolic execution

```
let x = unknown_function() in
let y = recursive_function() in
let z = ... operation not supported by solver in
...
```
Outline

• Mixing rules

• Examples and idioms for switching blocks

• Preliminary experience with MIXY, a mixed type qualifier inference engine for C
Preliminary experience

- **MIXY**, a prototype mixed type qualifier inference engine for C
- Applied to check that a *free* function is called only with a *non-null pointer* (usingnonnull type qualifier)
  - On vsftpd 2.0.7
  - Eliminated 2 false warnings
  - A combination of flow, path, and context-sensitivity was required
Conclusion

• New approach for trading off precision and efficiency in static program analysis

• Key: Nestable switching blocks to alternate between different off-the-shelf analyses

• Studied the mixing of type checking and symbolic evaluation
  - Proven soundness of symbolic execution/mix