Angelic Verification: Precise Verification Modulo Unknowns

Ankush Das, **Shuvendu Lahiri**, Akash Lal,

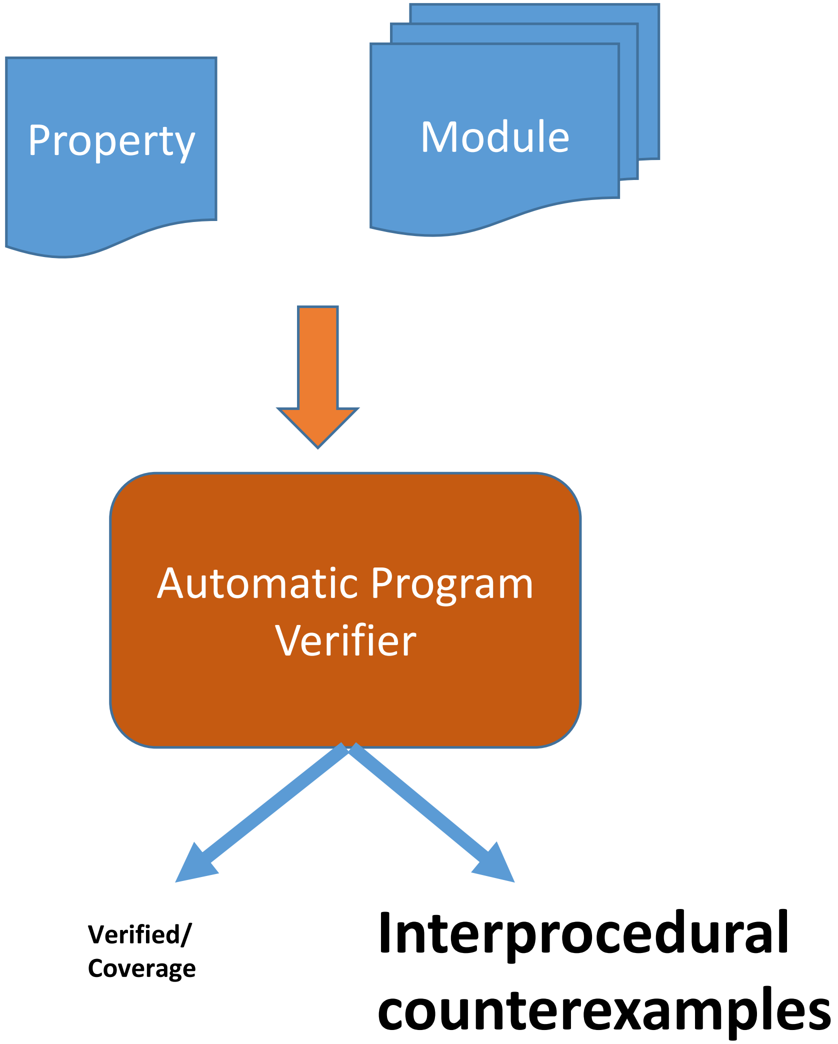
(Microsoft Research)

Yi Li

(University of Toronto)

Automatic whole-program verifiers

- Automatic whole program verifiers
  - SLAM, BLAST, IMPACT, JPF, FSOFT, CORRAL, ...
- Several success stories
  - Numerous bugs found and fixed



Open programs and program verifiers

- Most verification tasks require analyzing **open programs** interacting with their **environment**
  - Under-constrained inputs (parameters, globals)
  - Under-constrained library calls (no definition)
- Results in numerous “dumb alarms” when applied directly to a problem
  - “Stupid false positives” [Coverity paper, CACM’10]

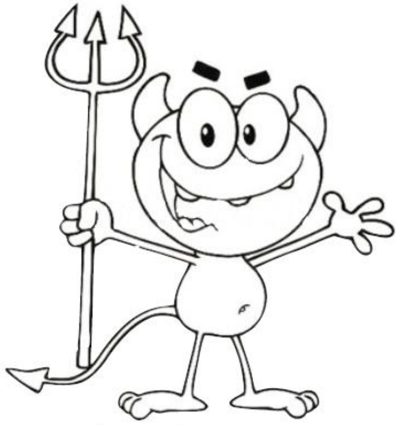
Dumb alarms

Often due to ***demonic*** assumptions about **environment** by the verifier

- Ignoring imprecision in analysis in this work

```
void foo(int *x, int *y) {
    free(x);  ✗   ← Possible double-free
    *y = 2;   ✗   ← Possible use-after-free
    free(x);  ✗   ← Possible double-free
}
```

Overly pessimistic



Check use-after-free

Open programs and program verifiers

- Most verification tasks require analyzing **open programs** interacting with its **environment**
  - Under-constrained inputs (parameters, globals)
  - Under-constrained library calls (no definition)
- Results in numerous “dumb alarms”
- Prescribed methodology
  - Modeling of **environment** (preconditions, models of external APIs)
    - SDV [Ball et al., CACM’11]
    - Significant “upfront” overhead, several man years work
- (In practice) Ad-hoc heuristics baked inside static analyzer
    - Specific to properties, statistical methods [Kremenek et al. SAS’03],

Problem: Hinders adoption of verifiers

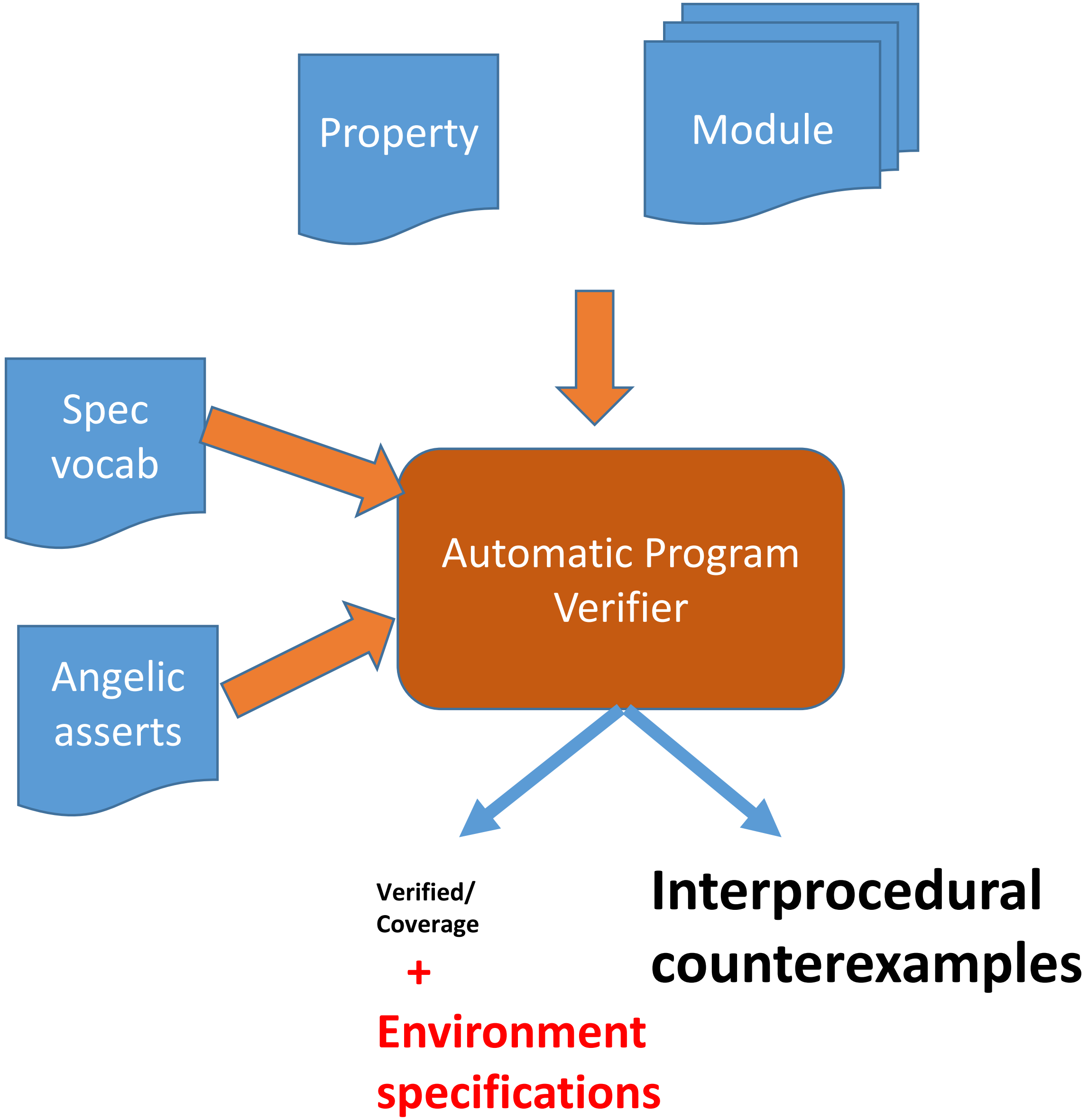
- No “out-of-the-box” experience
  - Find a few “interesting” alarms without a lot of effort
  - More effort (modeling) ➔ more “interesting” alarms

- Hard for a user to control/configure the tool
  - Adding manual pre/post conditions too low-level and cumbersome

- Expose more **knobs** to a user to control quality of alarms

Angelic verification

- Two knobs
  - *Vocabulary* of acceptable environment specifications
  - *Angelic assertions*


Property
Module
Spec vocab
Angelic asserts
Automatic Program Verifier
Verified/ Coverage
+
Environment specifications
Interprocedural counterexamples

Acceptable env specifications (example)

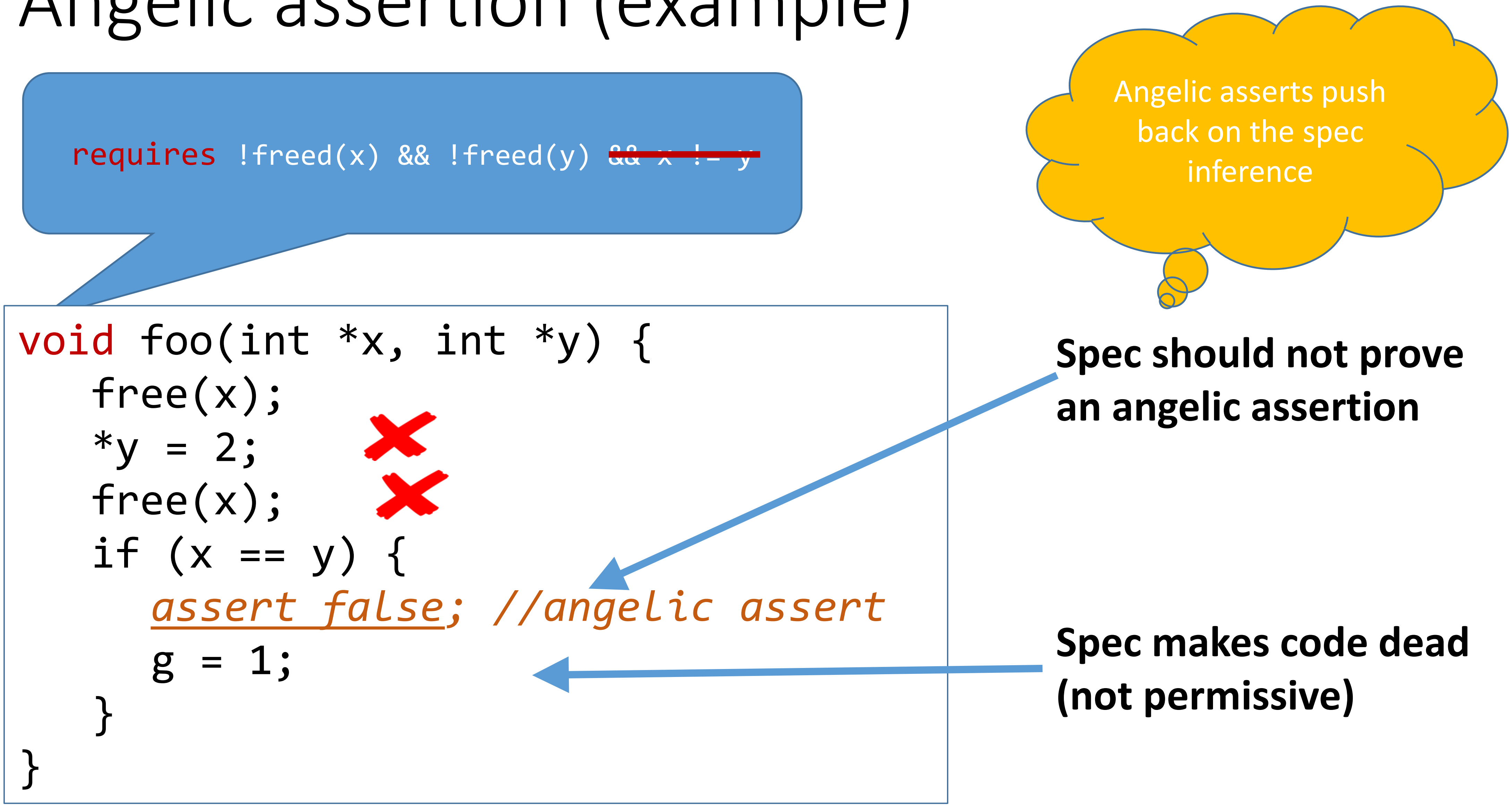
```
requires !freed(x) && !freed(y) && x != y
```

```
void foo(int *x, int *y) {
   free(x);
   *y = 2;
   free(x); ✗
}
```

Check use-after-free

Is there any acceptable specification over aliasing and property type-states

Angelic assertion (example)



requires !freed(x) && !freed(y) ~~&& x != y~~

```
void foo(int *x, int *y) {
    free(x);
    *y = 2;        ✗
    free(x);       ✗
    if (x == y) {
      assert false; //angelic assert
      g = 1;
    }
}
```

Angelic asserts push back on the spec inference

Spec should not prove an angelic assertion

Spec makes code dead (not permissive)

Angelic verification: problem statement

- Given a program **P** and a set of assertions **A** and
  1. A vocabulary of environment specifications **S**
  2. A set of angelic assertions **B**

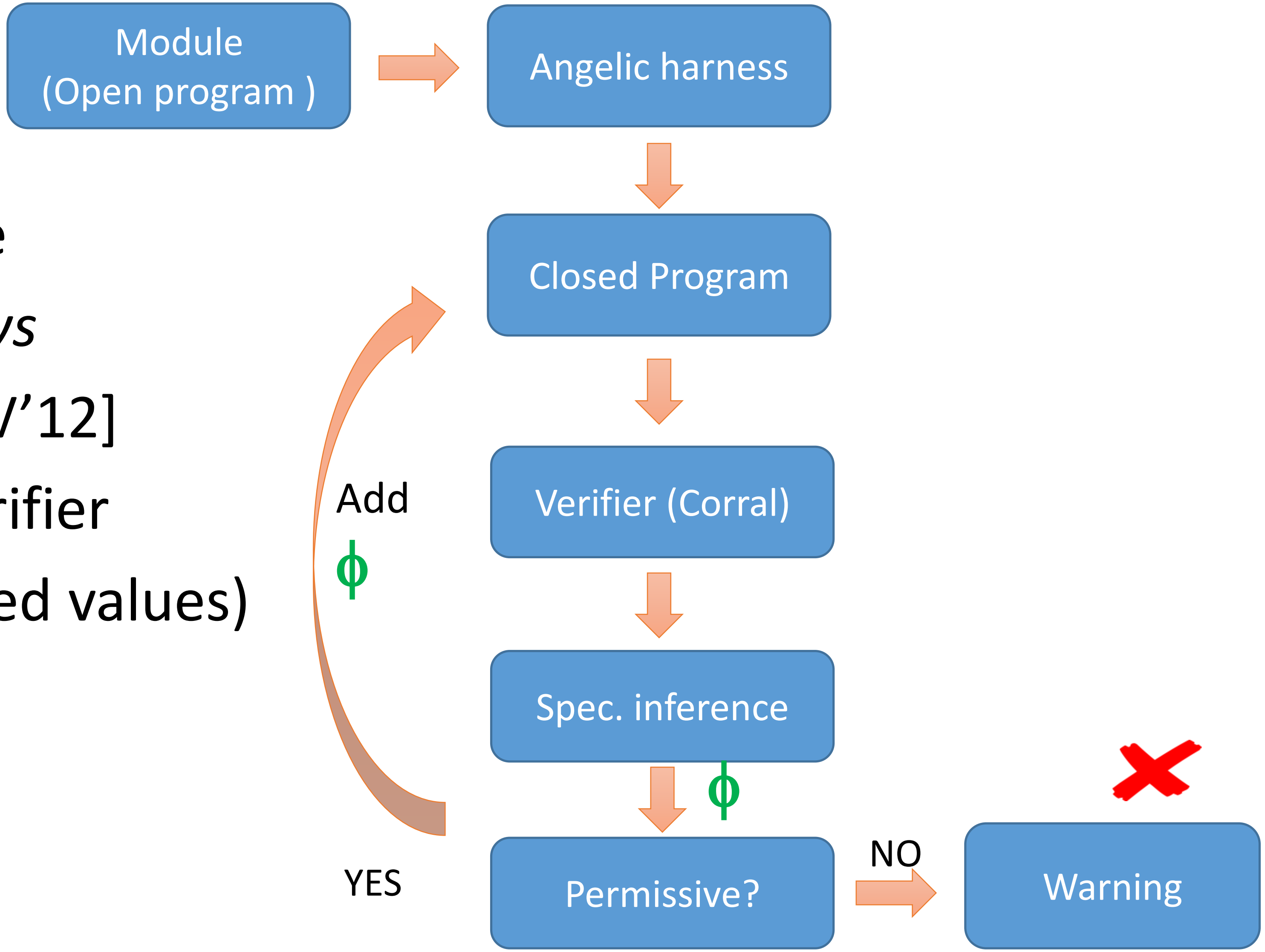
P is *angelically correct under (**S**,**B**),* if *there exists a specification **s** in **S** such that*

1. *For each **a** in **A**, **P** |= **s** ➔ **a***
2. *For each **b** in **B**, **P** |= **s** ➔ **b** only if **P** |= **b***

Rest of the talk

- Design of a *specific* angelic verifier (AV)
  - Angelic harness: closing an open program
  - Family of specifications provided by a **template of predicates**
  - Angelic assertions model **absence of dead code**
- Instantiate the AV for two case studies against existing tools
  - PREfix for null dereference
  - SDV for API usage properties

Architecture



- Programs compiled to Boogie
  - Heap modeled using *arrays*
- Corral [Lal, Qadeer, Lahiri CAV'12]
  - SMT-Based (bounded) Verifier
  - Demonic (for unconstrained values)
  - Whole-program
  - Optimized for bug-finding

Angelic harness: external calls

- External calls
    - Specs (at entry to Foo) cannot express constraints over callee returns
- Add explicit “triggers” as assumes

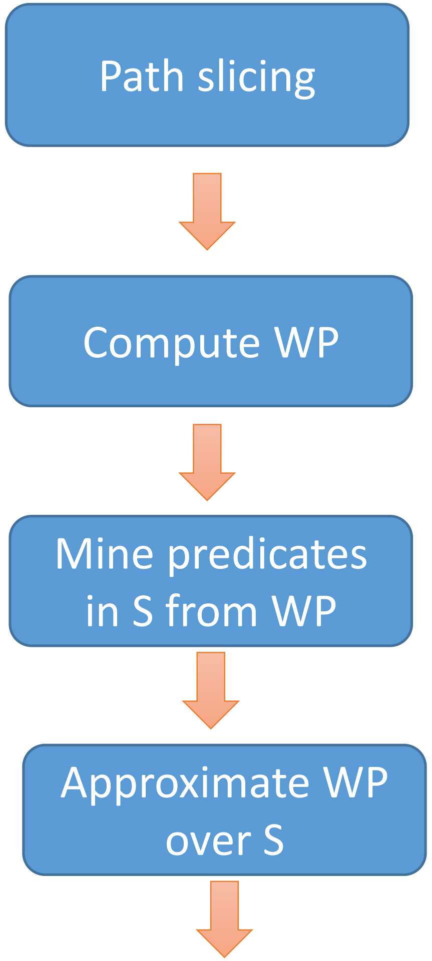
```
requires forall u: u != 0  //WP, too strong
requires forall u: {unknown_L(u)} :: unknown_L(u) ➔ u != 0
procedure Foo(…) {
   while(…) {
 L:    ~~call x := External(y);~~  //multiple dyn call sites
       x := *;
       assume unknown_L(x);
       assert x != 0;
    }
}
```

Spec inference (ExplainError)

- Given
  - A failure trace T
  - A family of predicates S
  - Boolean structure
- Output
  - A (weak) specification s in S that can rule out the trace

Boolean structure

- [Fast] Clause (c1 || c2 || c3)
- [Slow] CNF (c1 || c2 || c3)(c1' || c2' || c3')…



Path slicing

Compute WP

Mine predicates in S from WP

Approximate WP over S

Evaluation

- Research question
  - Can we instantiate AV to be comparable with existing mature solvers?
- Two case studies
  - PREfix for null dereference
  - Static Driver Verifier (SDV) for API usage

PREfix

- Large code bases
  - 10 modules: 400 KLOC, 18K procedures, 84K non-null asserts (before pruning)

- Compared against PREfix [Sielaff et al. ‘00]
  - PREfix is a production tool, used by Windows
  - Bottom up summarization
  - Has models for many OS APIs

- Alias analysis for pruning
  - Several hundred asserts per module, after pruning

AV configuration

- Predicates
  - Aliasing (**e1 != e2**), non-null (**e1 != NULL**)
- Boolean combination
  - Find single clause, if none then CNF
- Angelic asserts
  - Instrument conditionals of the form **e <> NULL**

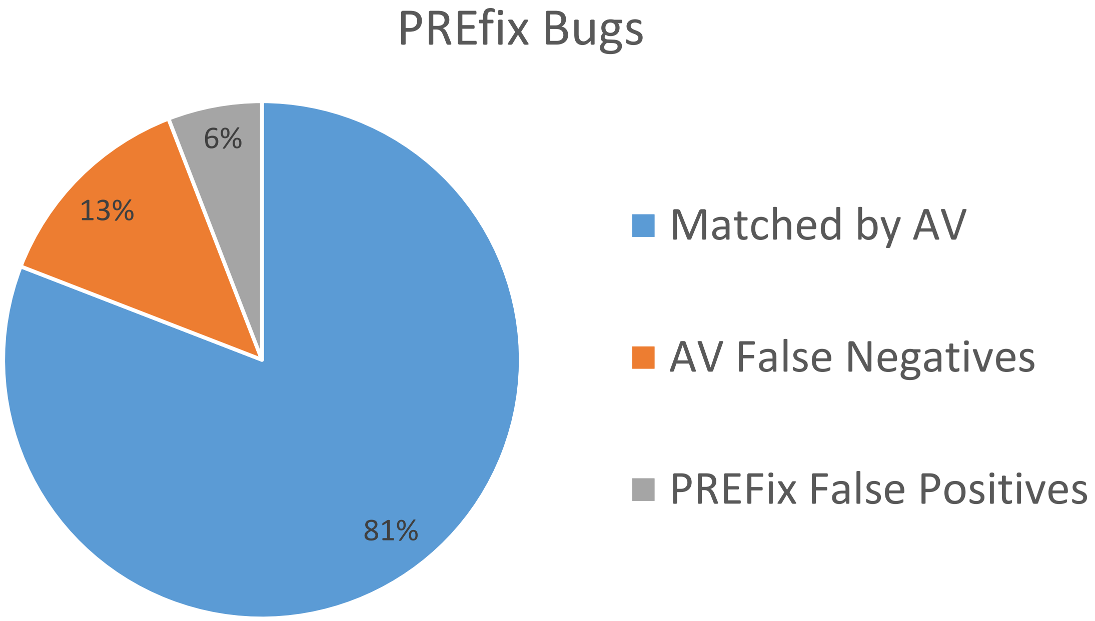
Results - PREfix

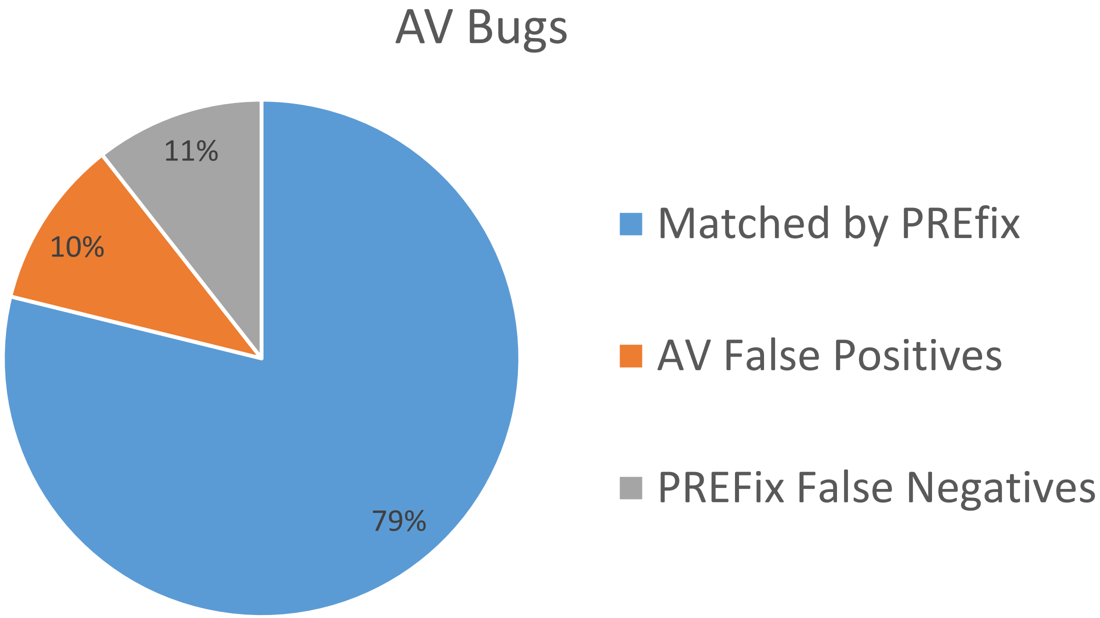
- PREfix reports 68 warnings
  - Unknown time (runs on a dedicated cluster behind a web interface)
- AV reports 104 warnings in 11 hours
  - More verbose

AV: Two warnings
PREfix: One warning

```
x = null;
if(…) { *x = … }
else  { *x = … }
```

Results - PREfix





Angelic assertions

- 6 true positives (missed by PREfix)

False positives

- Missing models (5), C->Boogie (6)

False negatives

- Missing models (1), timeouts (4), C->Boogie (5)

Without AV

- Corral reports almost 400 warnings, mostly false alarms
- Masks true bugs

Comparing against SDV

- Checking API usage properties
  - Lock usage, double completion of Interrupt Request (IRP) packets, …
- SDV modeling
  - Harness construction
  - Models of external APIs
- For AV
  - Remove the harness/initialization, models of external APIs

Results on SDV Benchmarks

Tool	Time (Ksec)	Bugs	False Positives	False Negatives
SDV (Buggy)	1.7	13	0	0
(Correct)	1.1	0	0	0
SDV, No Models	**.47**	**12**	**12**	**0**
	.28	21	13	5
AVN, No Models	**3.19**	**9**	**0**	**4**
	9.97	0	0	0
AVN, Some Modeling	**3.5**	**13**	**0**	**0**
	16.8	0	0	0

Related work

- Almost-correct specs [Blackshear & Lahiri, PLDI’13]
    - Expensive, can only be applied to procedure level
- Abductive inference [Dillig et al., PLDI’12]
    - Quantifier elimination after minsat, requires user in the loop for each alarm
- Bi-abduction in separation logic [Calcagno et al., POPL’09]
    - Similar to most bottom up analysis, no whole program counterexamples, user cannot control

Summary

- Need more knobs for automatic whole-program verifiers
- Angelic verification
  - Spec vocabulary
  - Angelic assertions
- Can be configured to match existing checkers without upfront modeling
  - More modeling ==> more interesting alarms!
- Current work (http://corral.codeplex.com/)
  - More properties (lifetime properties of pointers)
  - Completeness of predicate generation
  - Quantifier elimination for arithmetic properties
  - Automating inferring the right set of acceptable specifications

Questions