EXERCISE #8

COMPUTABILITY REVIEW

Write your name and answer the following on a piece of paper

Consider a simple bug-finding analysis that looks for null pointer deferences in C programs. The analysis raises an alert on any program that has ANY pointer operation, and does not raise an alert on any other program.
Is this analysis sound, complete, neither, or both? Justify your answer.

STATIC ANALYSIS

EECS 677: Software Security Evaluation

Drew Davidson

ADMINISTRIVIA AND ANNOUNCEMENTS

LAST TIME: ANALYSIS DEFINITIONS

REVIEW: COMPUTABILITY

Analysis Target – The system being analyzed

- For us this will usually be a software program

Analysis Engine – The system doing analysis

- For us this will usually be a software program

Analysis Goal – The phenomenon we are detecting

- The existence of a certain (program) behavior?
- The absence of a certain (program) behavior?

LAST TIME: ANALYSIS LIMITS

REVIEW: COMPUTABILITY

The limits of computability

- The Halting Problem: No decision procedure for halting
- Rice’s Theorem: The Halting Problem implies no decision procedure for any reachability problem

Analysis without decision procedures

- Approximation
- How do we approximate? Soundness / Completeness

LAST TIME: ANALYSIS GUARANTEES

REVIEW: COMPUTABILITY

NO analysis can be both sound and complete

Building an analysis that is either sound or complete is trivial

- Complete – Always report positive, no false negatives
- Sound – Always report negative, no false positives

LECTURE OUTLINE

- The Big Idea
- Program Guarantees
- Analysis Specificity
- Dataflow analysis

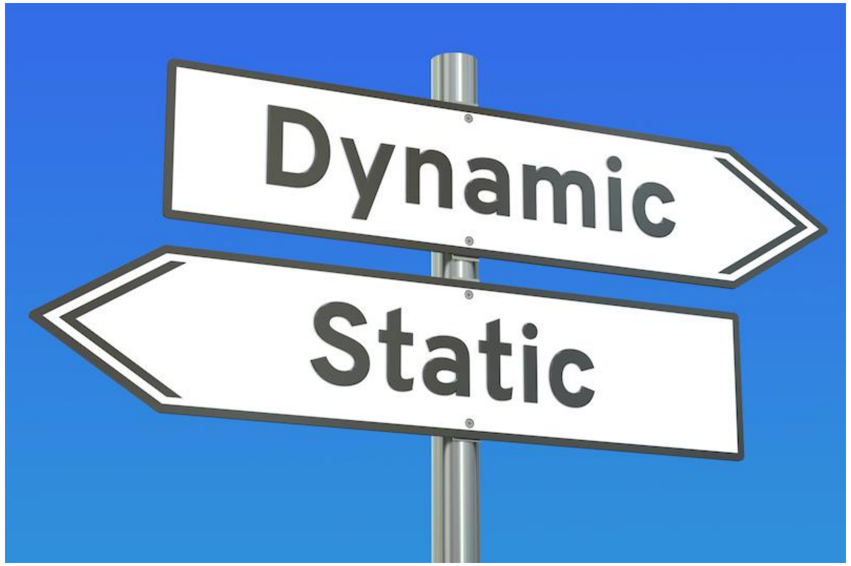
HIGH-LEVEL DEFINITION

STATIC ANALYSIS – THE BIG IDEA

Static analysis – analysis that is done without running the program

ANALYSIS IN CONTRAST

STATIC ANALYSIS PHILOSOPHY



Static analysis – analysis that is done without running the program

Dynamic analysis – analysis that is done with running the program

Simplest example - testing

Bug finding - testing
is
Sound

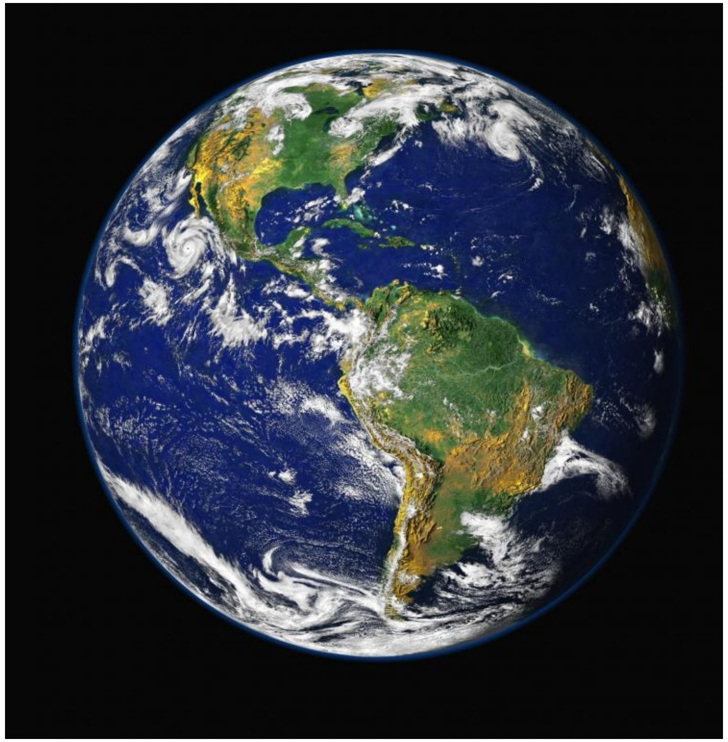
is
not complete

STATIC ANALYSIS – OPPORTUNITIES

STATIC ANALYSIS PHILOSOPHY

Global view of every instruction in the program

- Provide result about what a program MIGHT do



A global view (of the program)

STATIC ANALYSIS FOR QUALITY ASSURANCE

STATIC ANALYSIS PHILOSOPHY

Filter out “trivial” code issues

strcpy strncpy

Provide insights to aid manual analysis

“TRIVIAL” SYNTAX ANALYSIS

OVERVIEW: STATIC ANALYSIS

Some troubling behavior of a program may be discoverable via simply observing syntactic structure

```
int main(int argc, const char * argv[]){
  const char * password = argv[1];
  if (password == “supersecret”){
     authenticate();
  }
}
```

INSIGHTS

OVERVIEW: STATIC ANALYSIS

Software engineering “code smells” / stats

- Use of the forbidden / arcane constructs
- Cyclomatic complexity
- Long functions

goto considered harmful

LECTURE OUTLINE

- The Big Idea
- Program Guarantees
- Analysis Specificity
- Dataflow analysis

STATIC ANALYSIS – OPPORTUNITIES

STATIC ANALYSIS PHILOSOPHY

The true power of static analysis: one-sided error

Provide assurances about what a program will NEVER or ALWAYS do

- Static analysis might report EVERY program that (possibly) has a null-pointer dereference
- Static analysis might certify EVERY program that (definitely) is null-pointer deference free

“Hey! Those are the same thing!”

STATIC ANALYSIS – OPPORTUNITIES

STATIC ANALYSIS PHILOSOPHY

Provide assurances about what a program will NEVER or ALWAYS do

- Static analysis might report EVERY program that (possibly) has a null-pointer dereference
- Static analysis might certify EVERY program that (definitely) is null-pointer deference free

"Hey! Those are the same thing!"

Program verifier (detect "good" programs)

Complete (no FNs) – all good programs are reported
Sound (no FPs) – all bad programs are unreported

Bug finder (detect "bad" programs)

Complete (no FNs) – all bad programs are reported
Sound (no FPs) – all good programs are unreported

STATIC ANALYSIS – OPPORTUNITIES

STATIC ANALYSIS PHILOSOPHY

For security analysis, we want to lock out “bad” programs (even at the cost of locking out some “good” programs)

```
analysis( program) {
  if( program has bug) {   }
}
```

Program verifier (detect “good” programs)

Complete (no FNs) – all good programs are reported

Sound (no FPs) – all bad programs are unreported

Bug finder (detect “bad” programs)

Complete (no FNs) – all bad programs are reported

Sound (no FPs) – all good programs are unreported

LECTURE OUTLINE

- The Big Idea
- Program Guarantees
- Analysis Specificity
- Dataflow analysis

CONSIDER PROGRAM CONFIGURATION

ANALYSIS SPECIFICITY

The good news about static analysis: You can see beyond the instructions that are executed in an individual trace

The bad news about static analysis: You need to construct the conditions/circumstances/context in which those instructions are executed

```
*p = 2
```

You exist in the context of all in which you live and what came before you

WRASSLIN' WITH STATE SPACE

ANALYSIS SPECIFICITY

State space: the set of all possible configurations of the analysis target

Naïve state space representation: enumerate all configurations of a program

HISTORY: HARDWARE MODEL CHECKING

STATIC ANALYSIS – ANALYSIS SPECIFICITY

Extract a (finite) state system that approximates the analysis target

Example:

- States: configuration of the system
- Edges: transitions within the system

Check if the system can violate some correctness property

Each state indicates the value of a memory bit

HISTORY: HARDWARE MODEL CHECKING

STATIC ANALYSIS – ANALYSIS SPECIFICITY

State space
(artist’s rendition)

State space explosion!

n

2^n

WRASSLIN' WITH STATE SPACE

ANALYSIS SPECIFICITY

State space: the set of all possible configurations of the analysis target

Naïve state space representation: enumerate all configurations of a program

Never gonna work for large analysis targets

Practical state space representation: Summarize sets of configurations of a program

ASIDE: SOFTWARE MODEL CHECKING

ANALYSIS SPECIFICITY

Extract a (finite) state system that approximates the analysis target

- States: configurations of the system
- Edges: transitions within the system

Each state indicates a set of values or the truth of some abstract predicate

Check if the system can violate some correctness property

ASIDE: CEGAR

ANALYSIS SPECIFICITY

Counter**e**xample-**g**uided **a**bstraction **r**efinement

- Begin with a coarse, over-approximate abstraction of the system
- Check system correctness
- If a violation is reported, verify it!
  - If its a true positive – report it
  - If it's a false positive – refine the model to exclude it and check the new model

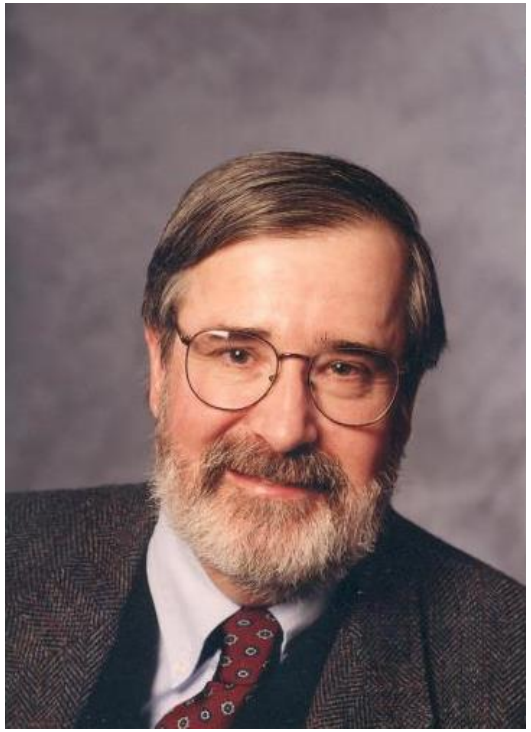
ASIDE: MODEL CHECKING IS GREAT!

ANALYSIS SPECIFICITY

Super-interesting approach to program analysis

Some scalability issues

Not the focus of our course



Edmund Clarke: Turing Award co-winner for model checking

STATE SPACE SUMMARIZATION

STATIC ANALYSIS: ANALYSIS SPECIFICITY

Lesson learned: The way we choose to summarize state space makes or breaks our analysis

- Too much summarization leads to approximation
- Too little summarization leads to state space explosion

LECTURE OUTLINE

- The Big Idea
- Program Guarantees
- Analysis Specificity
- Dataflow analysis

DATAFLOW INTUITION

STATIC ANALYSIS: DATAFLOW ANALYSIS

Capture the effect of each statement on the program’s data

- Treat each instruction as a data transformer
- Compose the effect of multiple data transformers to elicit composite effects

COMPOSITE EFFECTS

STATIC ANALYSIS: DATAFLOW

Dataflow analysis comes in a variety of configurations that stake out different precision/efficiency tradeoffs

- Maintain sound verification / complete bugfinding
- Consider sets of values that may not actually co-exist

FLOW-INSENSITIVE DATAFLOW ANALYSIS

STATIC ANALYSIS: DATAFLOW

Consider the effect of each statement without respecting the order of execution

FLOW-INSENSITIVE DATAFLOW ANALYSIS

STATIC ANALYSIS: DATAFLOW

Is a function FOO called from within a program?

```
*a();
 a = foo;

 b = a;
```

PATH-SENSITIVE DATAFLOW ANALYSIS

STATIC ANALYSIS: DATAFLOW

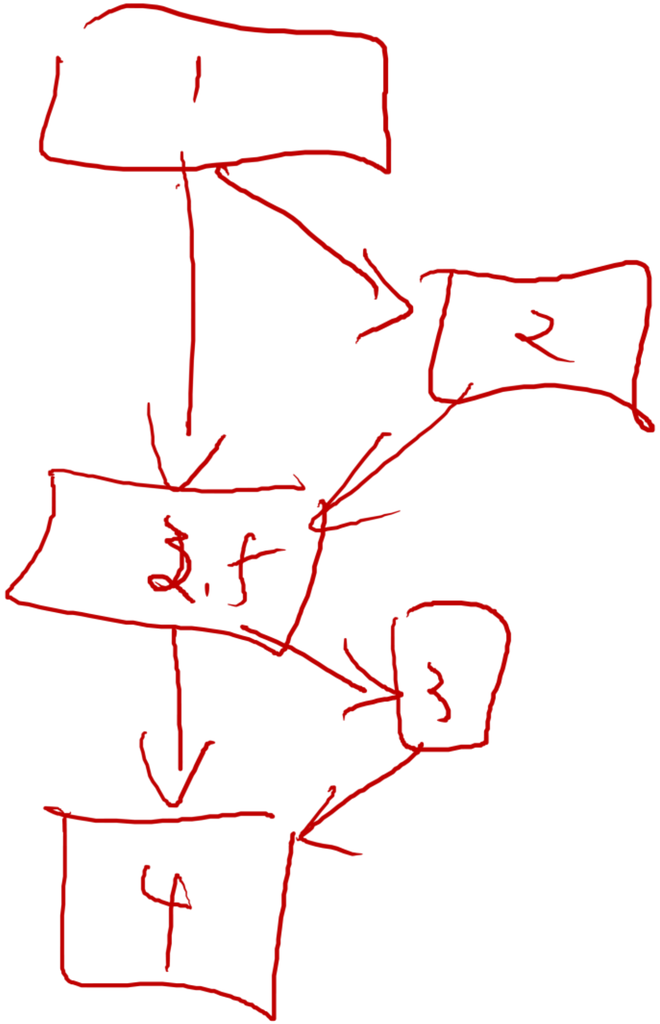
Consider the effect of each statement with respect to a unique program path

PATH-SENSITIVE DATAFLOW ANALYSIS

STATIC ANALYSIS: DATAFLOW

1,2.5,4
1,2,2.5,4
1,2.5,3,4
1,2.5,4

```
     int f(bool b) {
         Obj * o = null;
 1       int v = 2;
         if (b) {
             o = new Obj ();
             v = rand_int();
  2
2.5 ->   }
  3      if (v == 2){
             o->setInvalid()
         }
 4       return o->property();
     }
```



FLOW-SENSITIVE DATAFLOW ANALYSIS

STATIC ANALYSIS: DATAFLOW

Consider the effect of each statement with respect to order in the Control-Flow Graph

FLOW-SENSITIVE DATAFLOW ANALYSIS

STATIC ANALYSIS: DATAFLOW

```
int f(bool b) {
    Obj * o = null;
    int v = 2;
    if (b) {
        o = new Obj ();
        v = rand_int();
    }
    if (v == 2){
        o->setInvalid()
    }
    return o->property();
}
```

ABSTRACT INTERPRETATION

CATEGORIZING ANALYSES

(Over)approximate the state of the program
(Over)approximate the domain of values

LECTURE END!

- The Big Idea
- Program Guarantees
- Analysis Specificity
- Dataflow analysis

NEXT TIME

SYSTEMATIZING FLOW-SENSITIVE ANALYSES