EXERCISE #7

MEMORY ATTACKS REVIEW

What is a script injection attack and why does the W⊗X defense defeat it?

COMPUTABILITY

EECS 677: Software Security Evaluation

Drew Davidson

ADMINISTRIVIA AND ANNOUNCEMENTS

TODAY’S ROADMAP

Decidability

The Halting Problem

Type I/Type II Errors

Soundness / Completeness

THE LIMITS OF COMPUTATION

DECIDABILITY

Computers! What can't they do?!

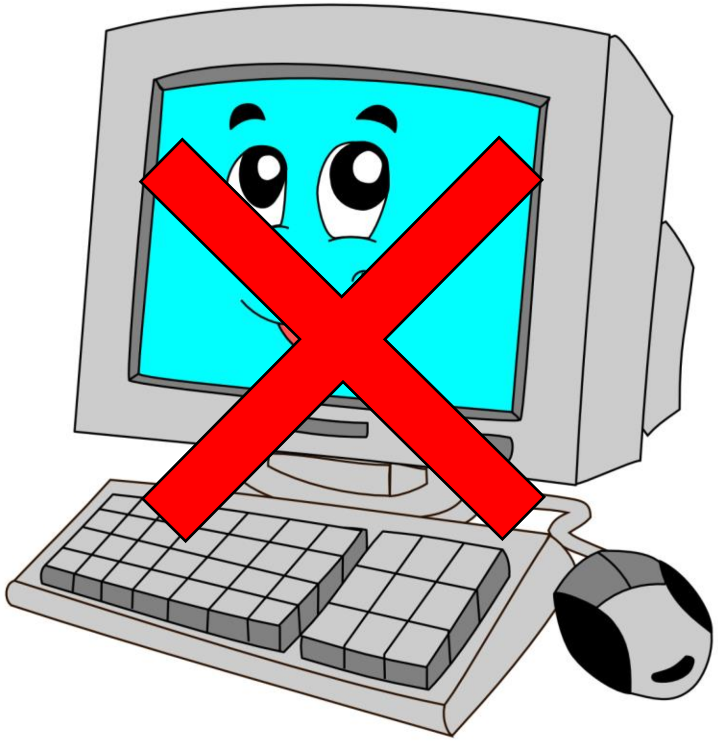
- As we begin our exploration of security evaluation, we care about this question for two reasons:
  - We need to know the capabilities of our analysis target
  - We need to know the capabilities of our analysis engine

THEORETICAL LIMITS OF COMPUTATION

DECIDABILITY

Computability theory

- The study of what is computable
- Focused on abstractions for the sake of generalizability
  - Considers theoretical hardware, for example

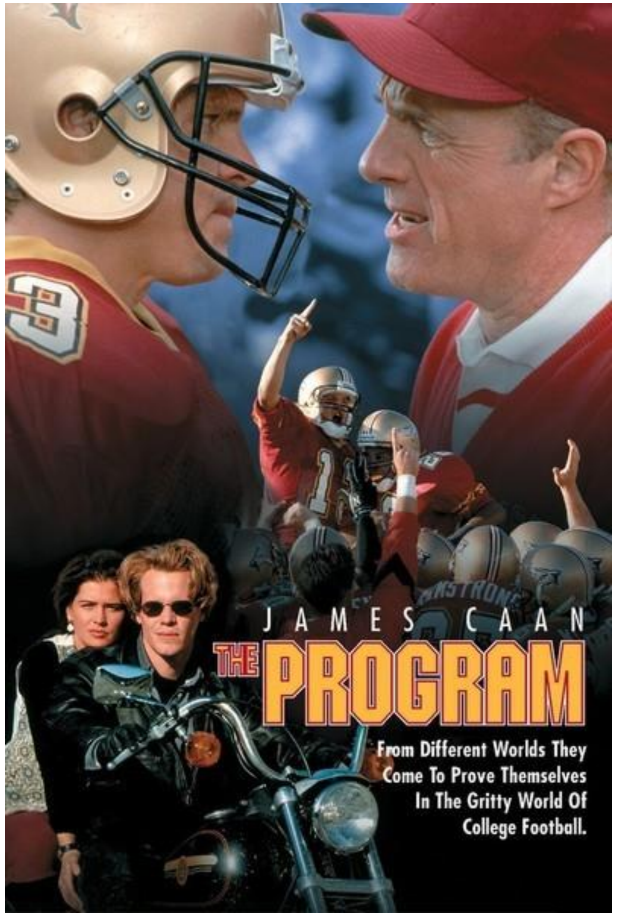


COMPUTATIONAL POWER

DECIDABILITY

What is a program?

- A set of executable instructions



COMPUTATIONAL POWER

DECIDABILITY

What is a program?

- A set of executable instructions

There are many formats for programs

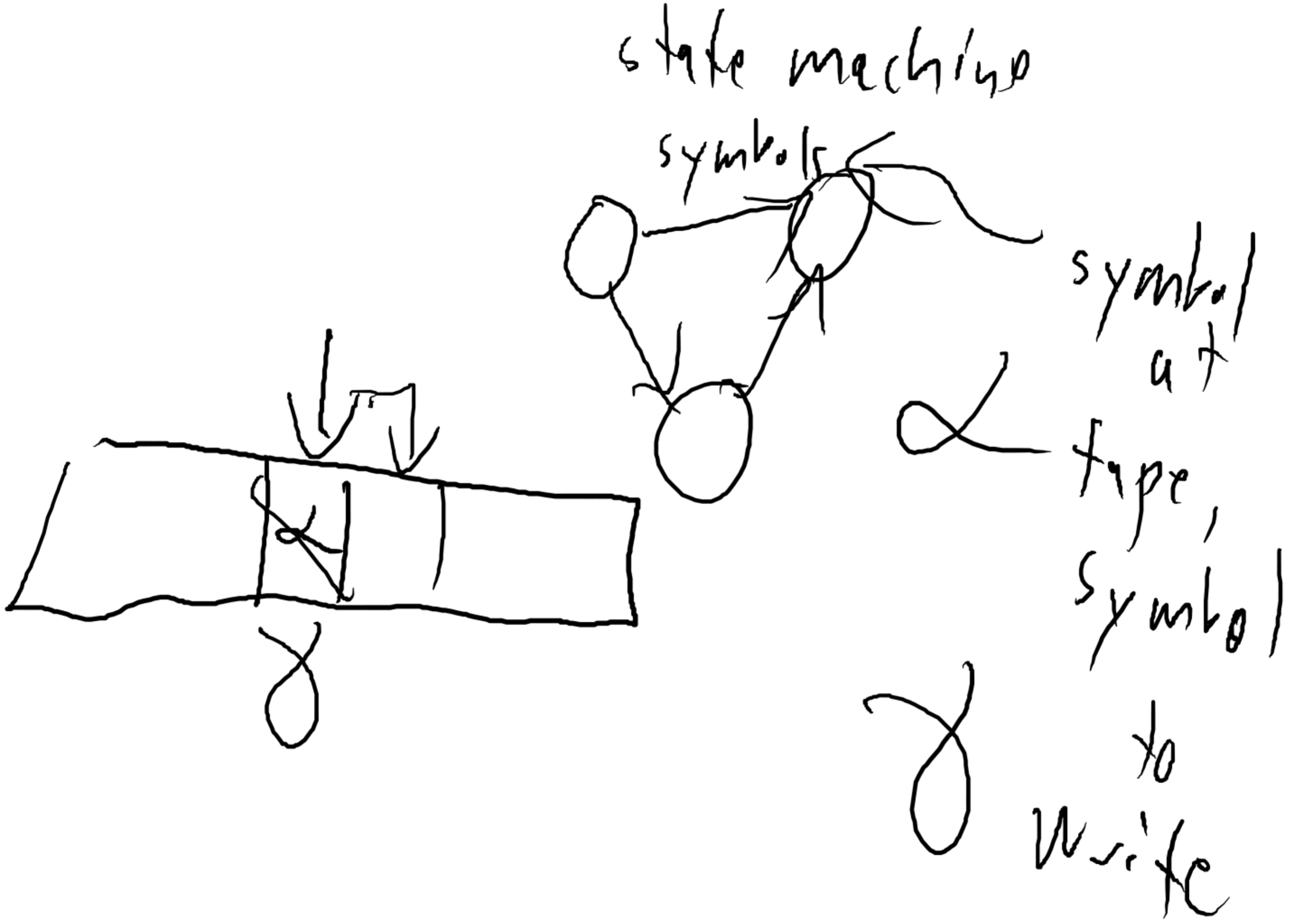
- i.e. programming languages
- It would be nice to generalize what these programs can compute (without getting bogged down in syntax)

ABSTRACTING COMPUTATION

DECIDABILITY

Computability theory considers classes of expressiveness

- Combinational logic
- Finite-state machines
- Pushdown automata
- Turing machines

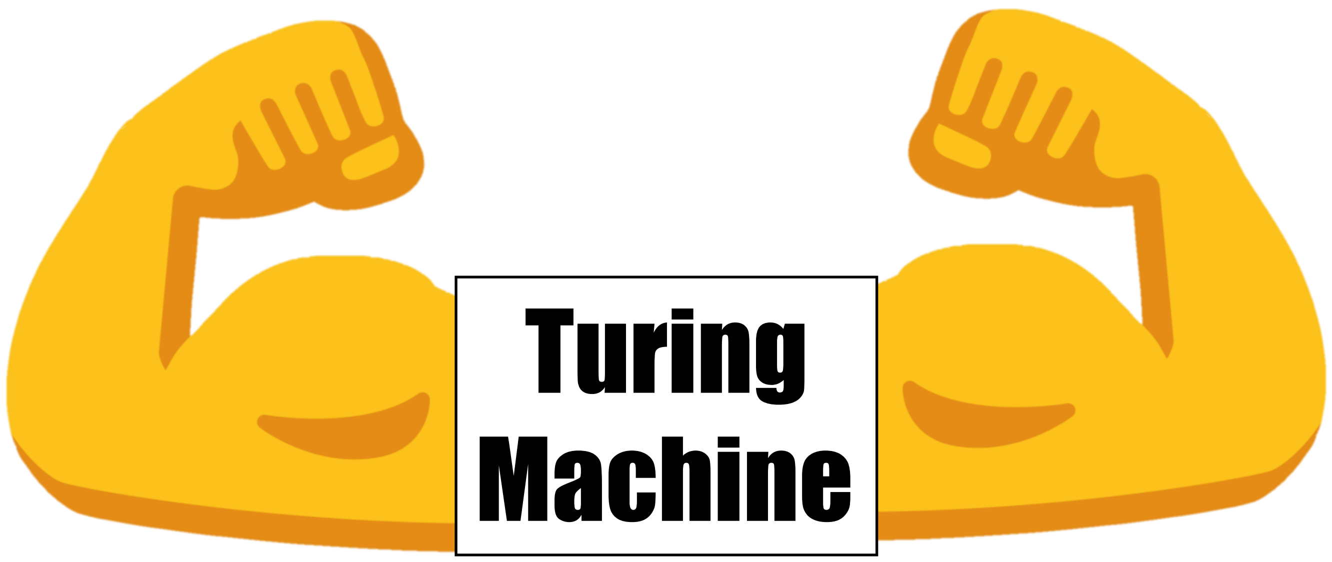


CHURCH-TURING THESIS

DECIDABILITY

Roughly: a function on the natural numbers can be calculated if and only if it is computable by a Turing machine

Practical Upshot: Turing machines are powerful!



VIBE CHECK

DECIDABILITY

Does everyone remember why we are doing this?

- We want to determine the power of our analysis target
- We want to determine the power of our analysis engine

Good news! Both are bounded by Turing computability

- Next up: abstracting analysis itself

DECISION PROCEDURES

DECIDABILITY

A little vocabulary:

A **decision problem** is a computational question that can be solved with either a yes or a no. *Frequently, we consider decision problems as detection of a property in a program*

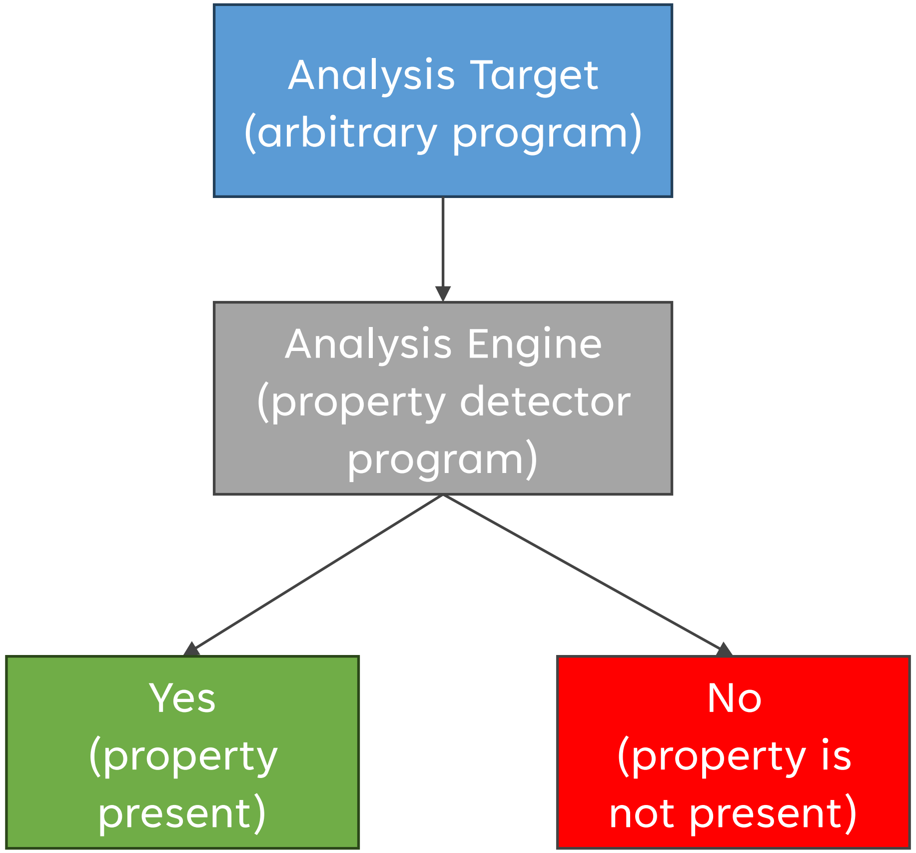
A **decision procedure** is a method for solving a decision problem that always yields the correct answer

If there is no decision procedure for a given decision problem, that decision problem is called **undecidable**

PROGRAM ANALYSIS AS DECISION PROCEDURE

DECIDABILITY

Since a program is just a list of instructions, it is valid input to a decision procedure



STRONG GUARANTEES

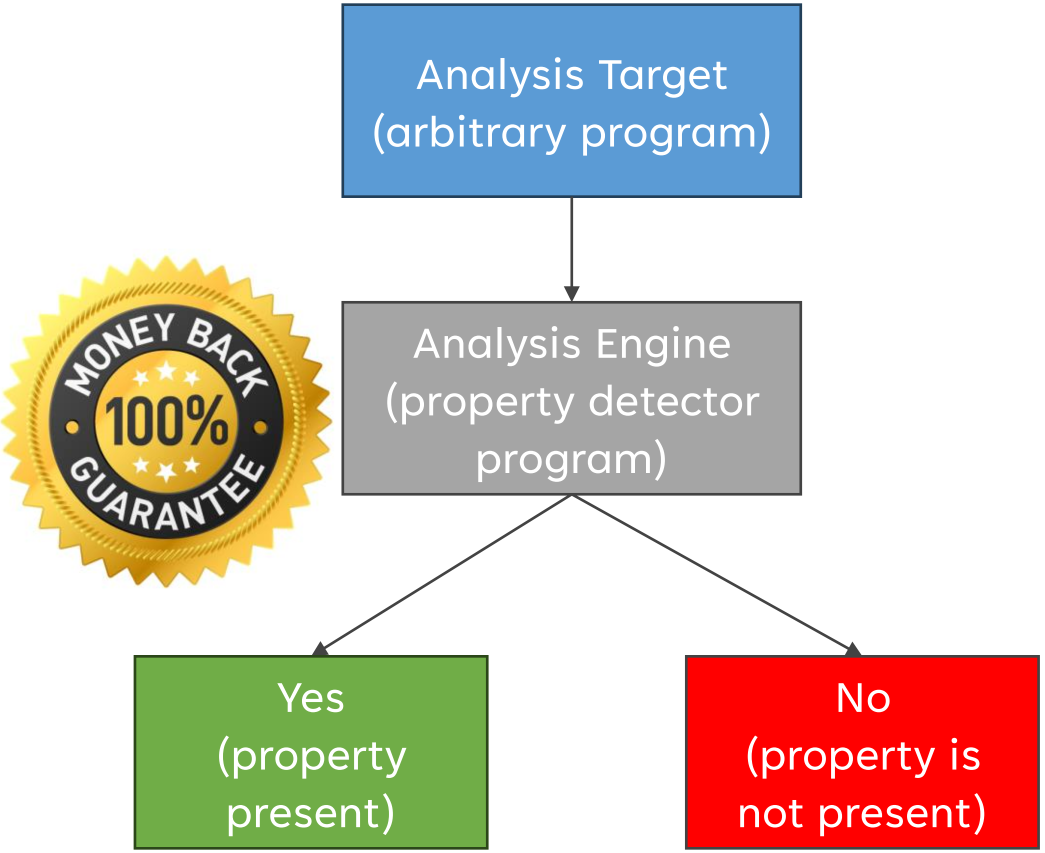
DECIDABILITY

A decision procedure is a high bar

Guarantee that:

- The analysis engine accepts every program
- The analysis engine always returns an answer
- The answer returned is always correct

Rice's Theorem



TODAY'S ROADMAP

Decidability

The Halting Problem

Type I/Type II Errors

Soundness / Completeness

STATING THE PROBLEM

THE HALTING PROBLEM

Given a description of a Turing machine and its initial input, determine whether the program, when executed on this input, ever halts (completes). The alternative is that it runs forever without halting

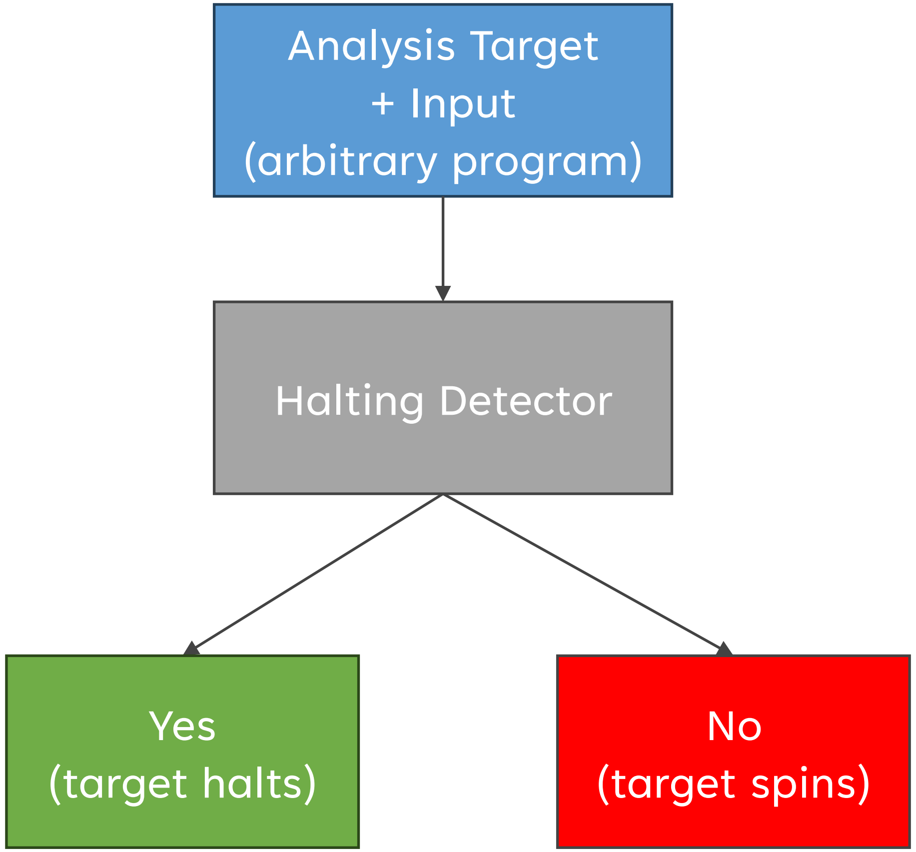
A HALTING DETECTOR

THE HALTING PROBLEM

Given a description of a Turing machine and its initial input, determine whether the program, when executed on this input, ever halts (completes). The alternative is that it runs forever without halting

Is there a decision procedure for the halting problem?

- We'll sketch the proof outline that there is NOT
- Relies on a proof by contradiction



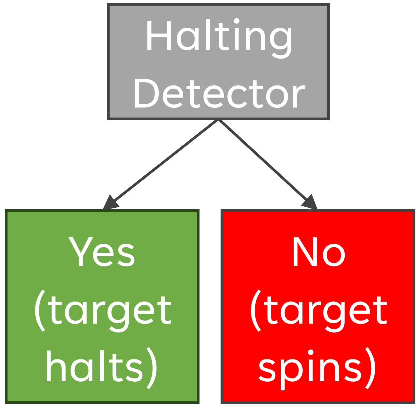
PROOF BY CONTRADICTION

THE HALTING PROBLEM

Reductio ad absurdum – Assuming the premise has obviously incorrect consequences

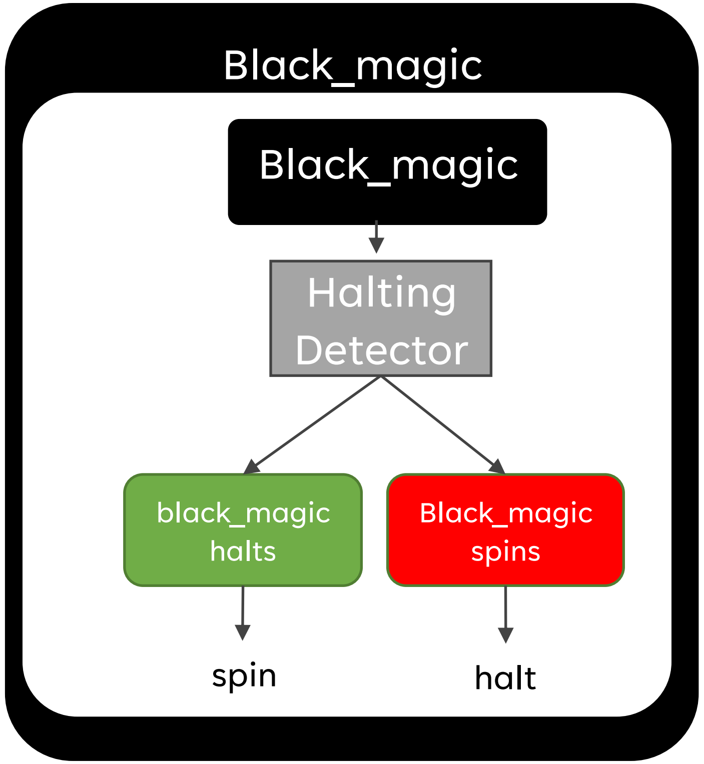
Here: assume there is a halting detector

Assumption



`halts(ANY ARBITRARY PROGRAM)`

```
black_magic(){
    if (halts(black_magic){
      while(true){} //Spin
    }
    //Halt
}
```



WHO CARES?

THE HALTING PROBLEM

No halting decision procedure means no reachability decision procedure

```
1. int main(){
2.     if ( black_magic() ){
3.         int * a = nullptr;
4.         *a = 1;
5.     }
6. }
```

This program crashes if and only if it reaches line 4, which depends on the result of a function call being true

RICE'S THEOREM

THE HALTING PROBLEM

No halting decision procedure means no reachability decision procedure

```
1. int main(){
2.     if (black_magic()) {
3.         int * a = nullptr;
4.         *a = 1;
5.     }
6. }
```

behavior you care about

Exhibits the behavior you care about

This program ~~crashes~~ if and only if it reaches line 4, which depends on the result of a function call being true

RICE'S THEOREM
THE HALTING PROBLEM

"All non-trivial semantic properties of programs are undecidable"

LIMITATIONS OF RICE'S THEOREM

THE HALTING PROBLEM

Rice's Theorem is less catastrophic than you might expect for security:

- A decision procedure is a pretty high bar
- A Turing machine is actually not a perfect approximation of the computers we use!

Despite these limitations, it is widely accepted that program analysis is **always** approximate

- We can't be right all of the time
- We can choose what types of errors we make

TODAY'S ROADMAP

Decidability

The Halting Problem

Categorizing Program Analyses

Soundness / Completeness

CLASSIFYING DETECTORS

CATEGORIZING PROGRAM ANALYSES

Abstractly: an analysis is a system to detect a phenomenon

A hand detector: when hand detected, emit soap

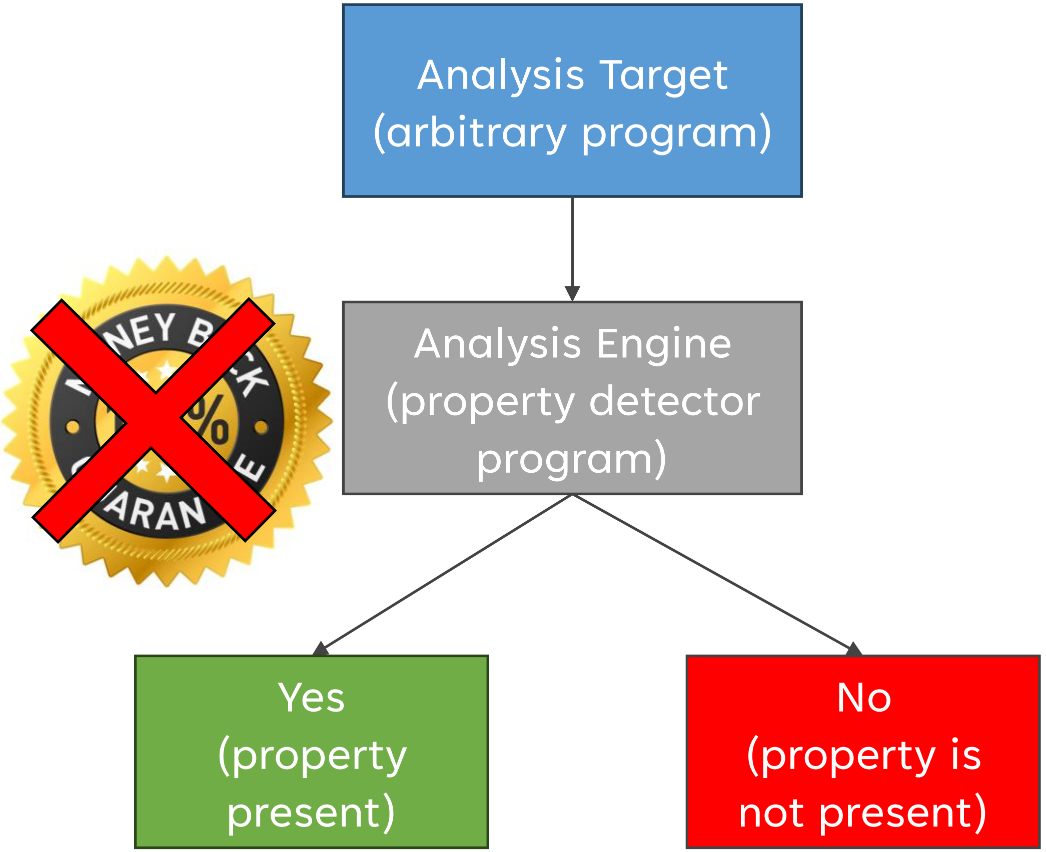
CLASSIFYING DETECTORS

CATEGORIZING PROGRAM ANALYSES

	TRUE correct	**False** wrong
POSITIVE "alert"		
NEGATIVE "Not here"		

TYPES OF ANALYSIS
CATEGORIZING PROGRAM ANALYSES

In order to determine the properties of a given program analysis, let's frame it as a detector

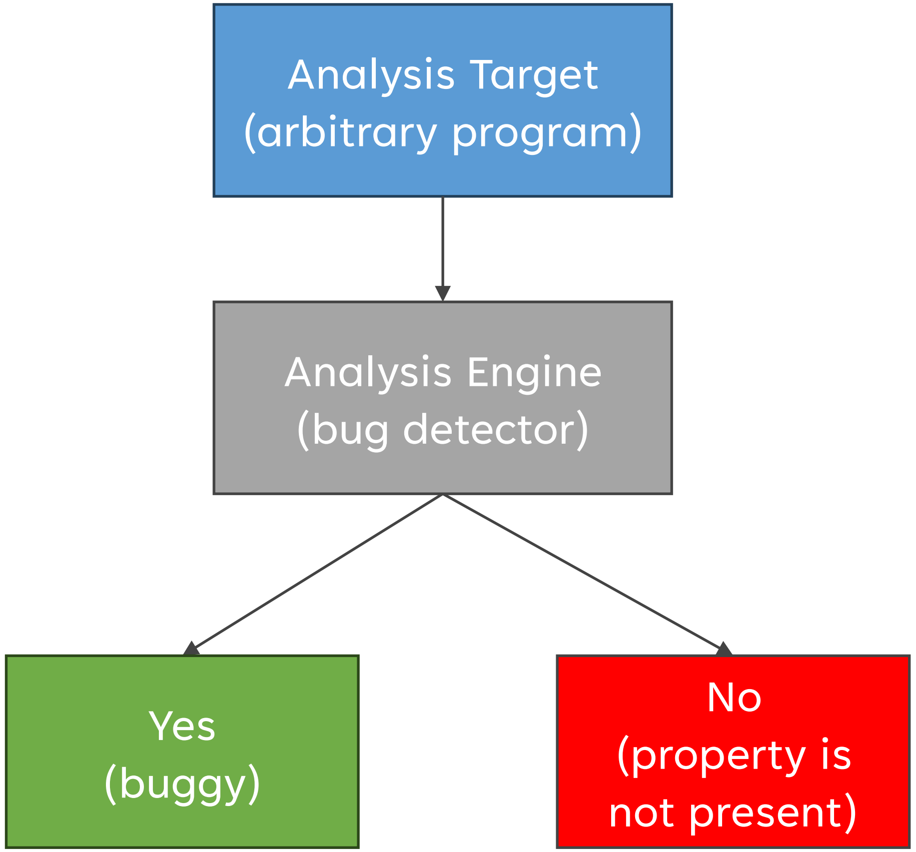


Note: we can detect bad behavior or good behavior

CLASSIFYING ERRORS

CATEGORIZING PROGRAM ANALYSES

	True Analysis is correct	False Analysis is wrong
Positive report bug	Has report Has bug Correct	Has report No bug Type I Error
Negative No bug report	No report No bug Correct	No report Has bug Type II Error



TODAY'S ROADMAP

Decidability

The Halting Problem

Categorizing Program Analyses

Soundness / Completeness

GUARANTEES OF IMPERFECT ANALYSES

SOUNDNESS / COMPLETENESS

Consistency / Reliability super important for users

We'd like to limit the kinds of errors we report

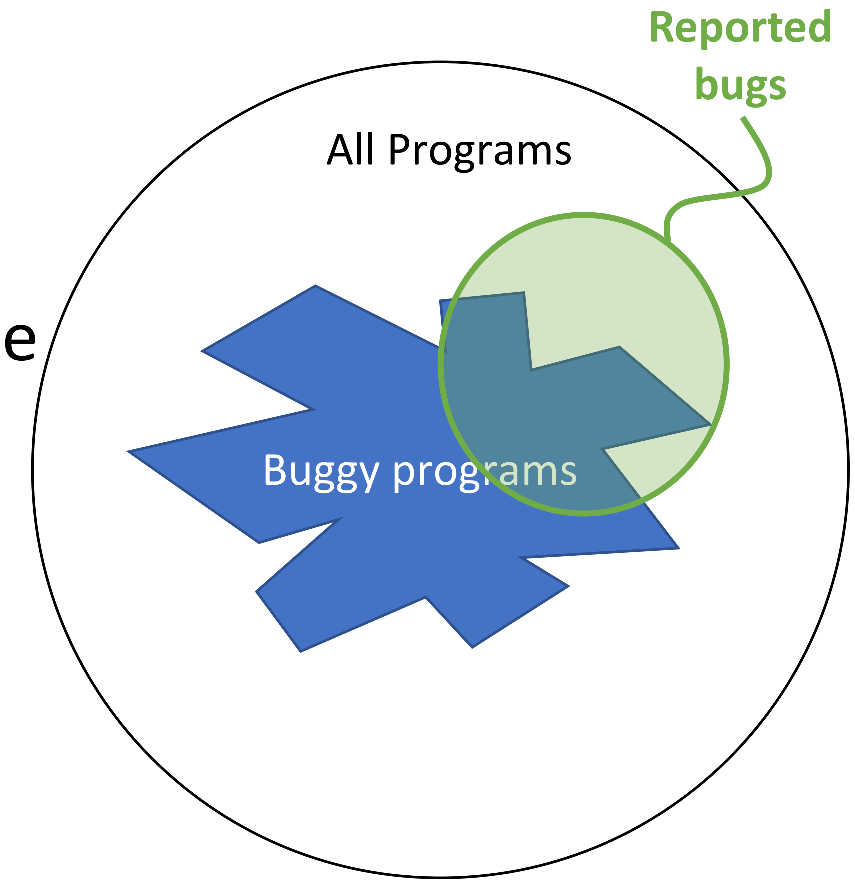
We can choose which type of bug report error to avoid

- Soundness: No false positives
- Completeness: No false negatives

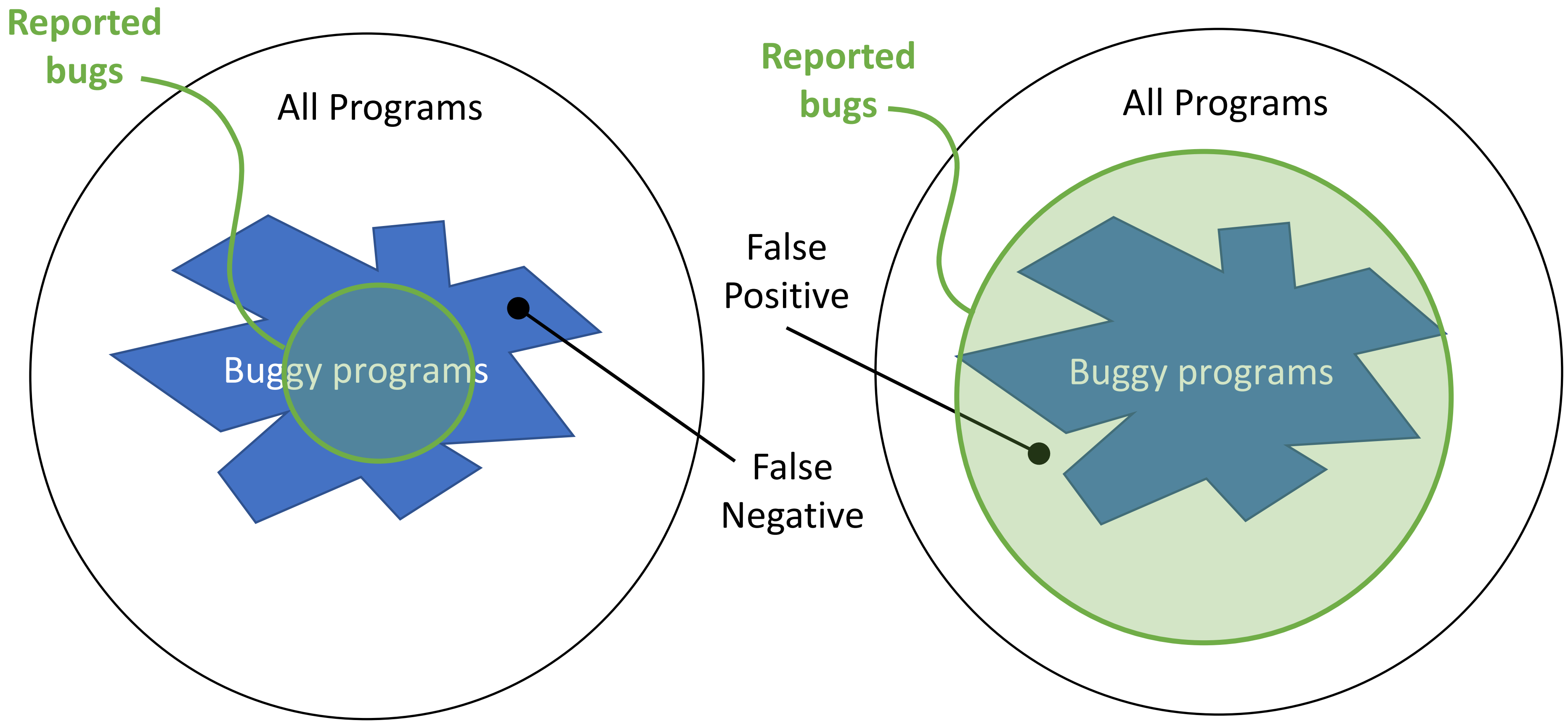
VISUAL ANALOGY
SOUNDNESS / COMPLETENESS

Imagine the universe of all programs is contained in a circle

- You can draw a circle around the programs you report as buggy
- The actual buggy programs occupy a jagged region



VISUAL ANALOGY
SOUNDNESS / COMPLETENESS



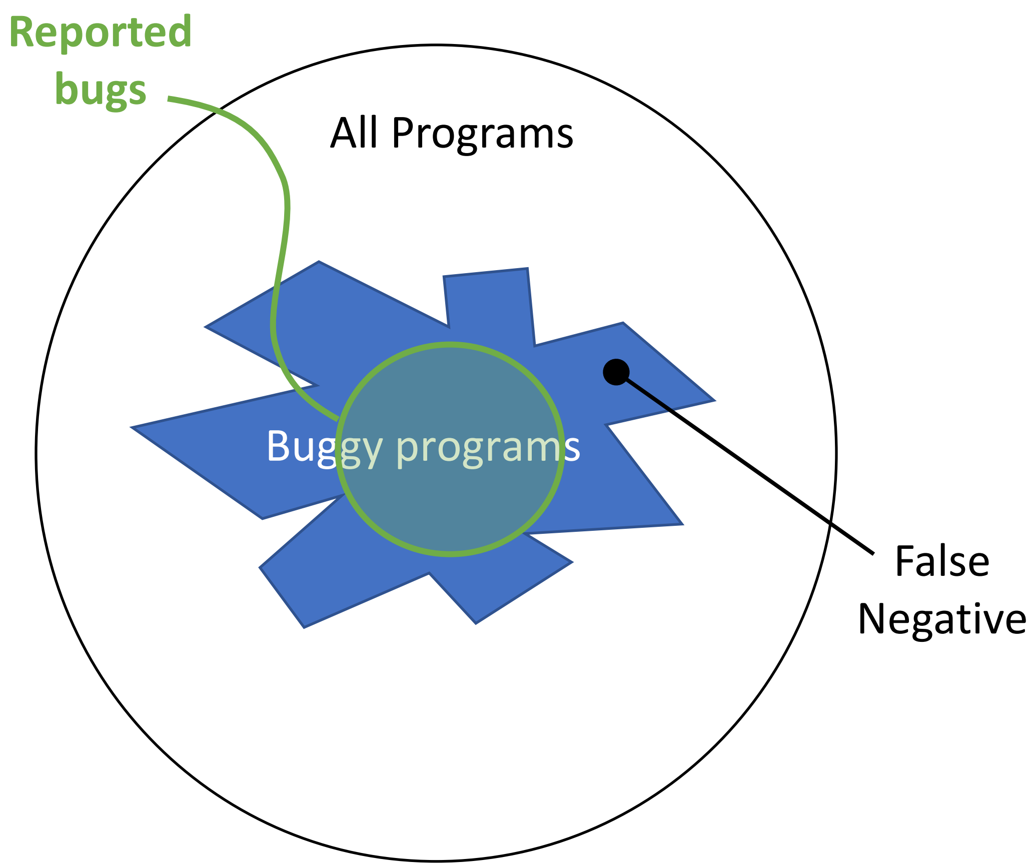
Sound bug detection
All correct programs pass through
(No false positive problem)

Some buggy programs pass through
(has false negative problem)

Complete bug detection
All buggy programs get flagged
(No false negative problem)

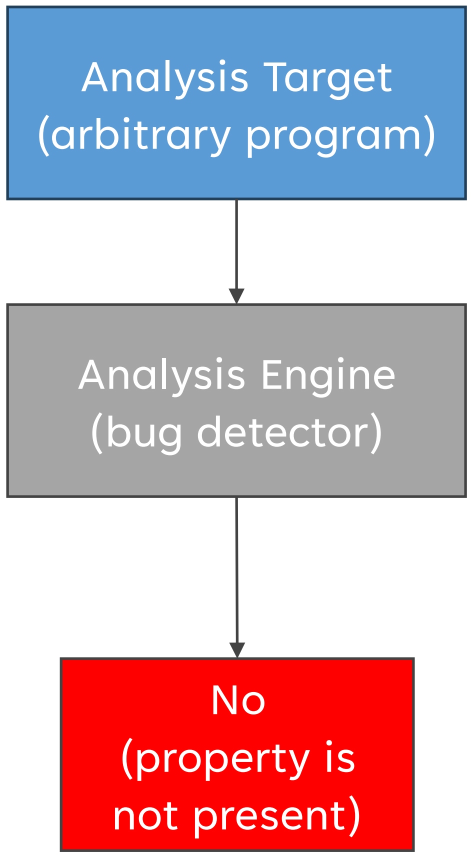
Some correct programs get flagged
(has false positive problem)

TRIVIAL SOUNDNESS
CATEGORIZING PROGRAM ANALYSES

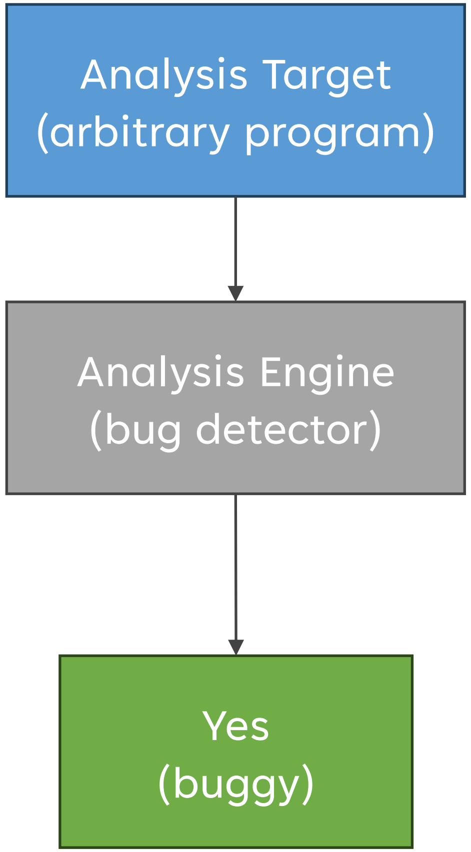


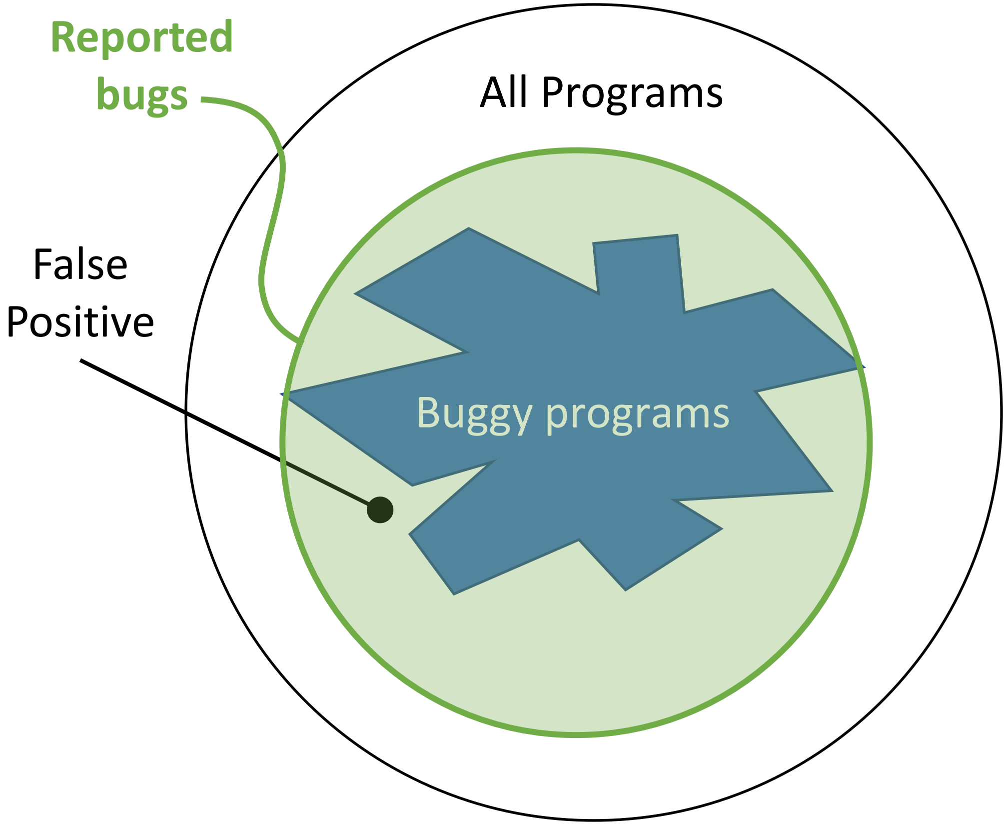
Sound bug detection
All correct programs pass through
(No false positive problem)

Some buggy programs pass through
(has false negative problem)



TRIVIAL COMPLETENESS
CATEGORIZING PROGRAM ANALYSES





Complete bug detection
All buggy programs get flagged
(No false negative problem)

Some correct programs get flagged
(has false positive problem)

BEYOND ALL-OR-NOTHING

SOUNDNESS / COMPLETENESS

As you can imagine, soundness and completeness are not the full story

- Guarantees are nice, but we want legitimately useful analyses!
- Many practical analyses are neither sound nor complete

STATIC VS DYNAMIC ANALYSIS

SOUNDNESS / COMPLETENESS

One distinction in analysis is how the analysis treats the target

- Static analysis – Operates without running the program
- Dynamic analysis - Operates with running the program

ANALYSIS METHOD VS ERRORS

SOUNDNESS / COMPLETENESS

It's natural to consider the types of compromises of each analysis method

- Static analysis
  - Often builds a model of the program, makes inferences on that model
  - Tends to make completeness easier
  - Scalability concerns for large programs
- Dynamic analysis
  - Often performs the analysis by straight up running the program, observing behavior
  - Tends to make soundness easier
  - Coverage problems

ABOUT COVERAGE

SOUNDNESS / COMPLETENESS

```
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();
}
```

Line coverage

Branch coverage

Path coverage

LECTURE END

Summary:

- Decidability
- Computational Theory
- Categorizing analysis