EXERCISE #8

LATTICES REVIEW

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

- Recall that the set of English words, ranked by substring inclusion, forms a poset but NOT a lattice. Explain why and give an example

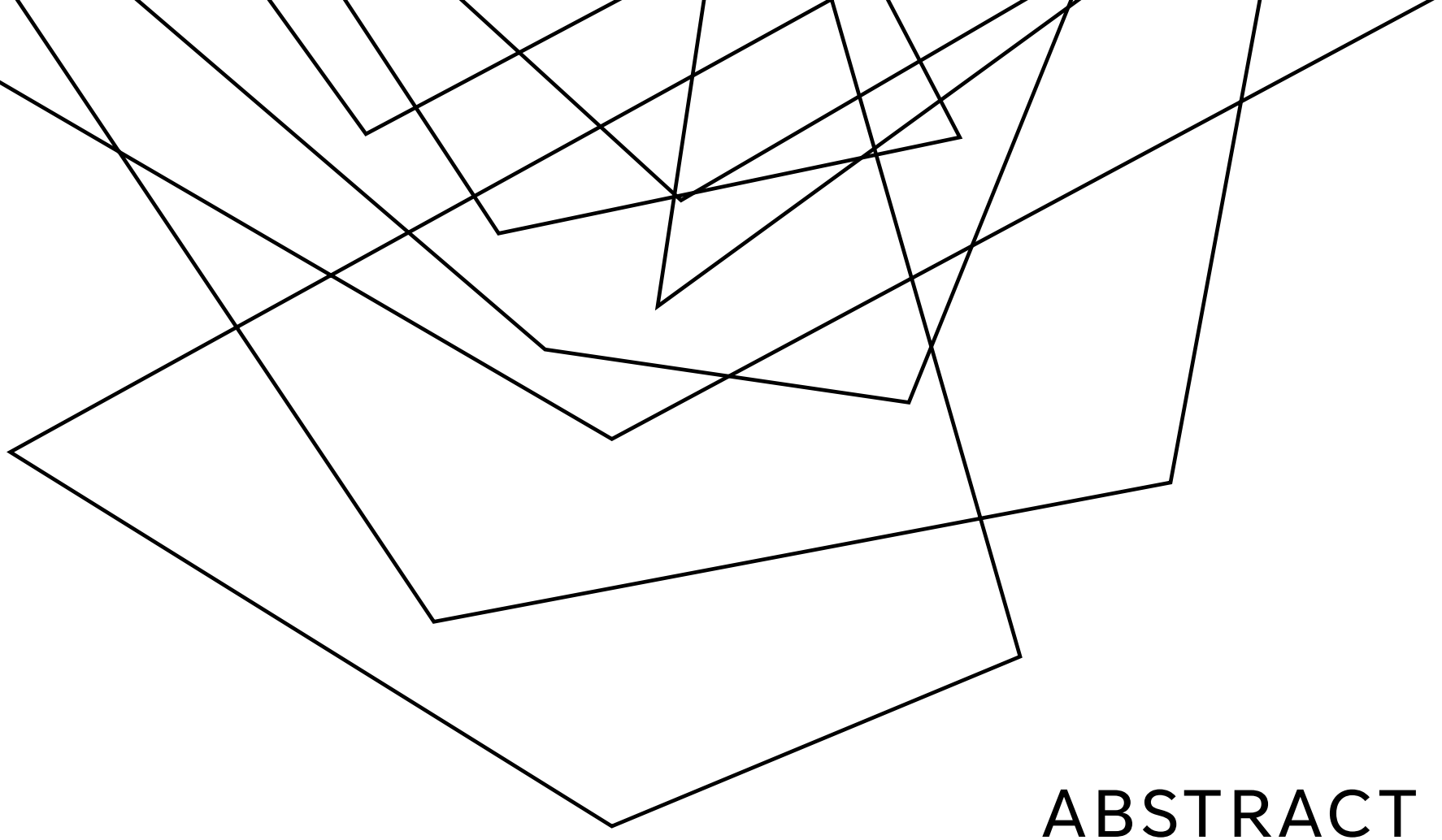
Reflexive: every word includes the substring of itself

Anti-symmetric: if two words include substrings of each other, they must be the same word

Transitive: if a is a substring b, and b is a substring of c, then a is a substring of c

Not a lattice: consider the English words a and he. They have no greatest lower bound

ADMINISTRIVIA AND ANNOUNCEMENTS



ABSTRACT INTERPRETATION

EECS 677: Software Security Evaluation

Drew Davidson

LAST TIME: LATTICES

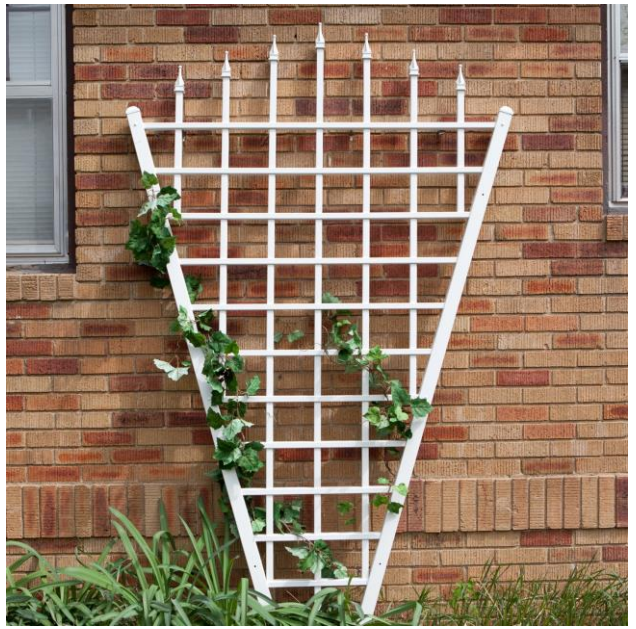
REVIEW: STATIC ANALYSIS

Described Formal Conditions to Guarantee Termination

- Dataflow facts can be ordered into a complete lattice
- Iterative application of a monotonic function can achieve a fixpoint

Some practical Limitations

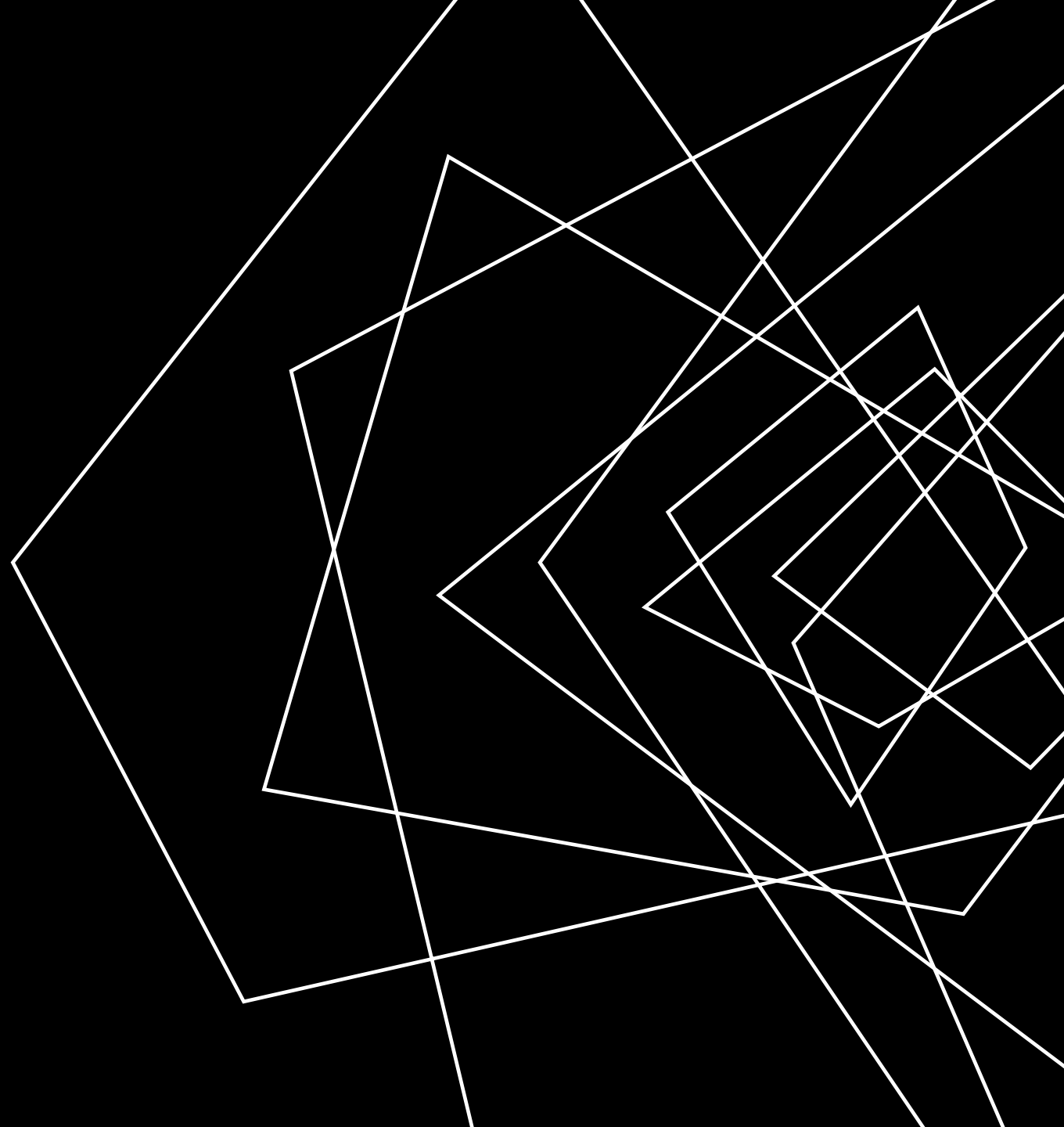
- No guarantee about how long the algorithm will run
- Sadly out of luck if we don't have a complete lattice



It's a lattice!

LECTURE OUTLINE

- Abstract Interpretation
- LLVM (time permitting)



ANALYSIS PRECISION

ABSTRACT INTERPRETATION

PRECISION / EFFICIENCY TRADEOFF

With a complete lattice we can, in theory, eventually terminate

That's not a very strong guarantee!

The shallower the lattice, the faster the fixpoint

Choose to approximate the lattice

ANALYSIS PRECISION

ABSTRACT INTERPRETATION

PRECISION / EFFICIENCY TRADEOFF

With a complete lattice we can, in theory, eventually terminate

That's not a very strong guarantee!

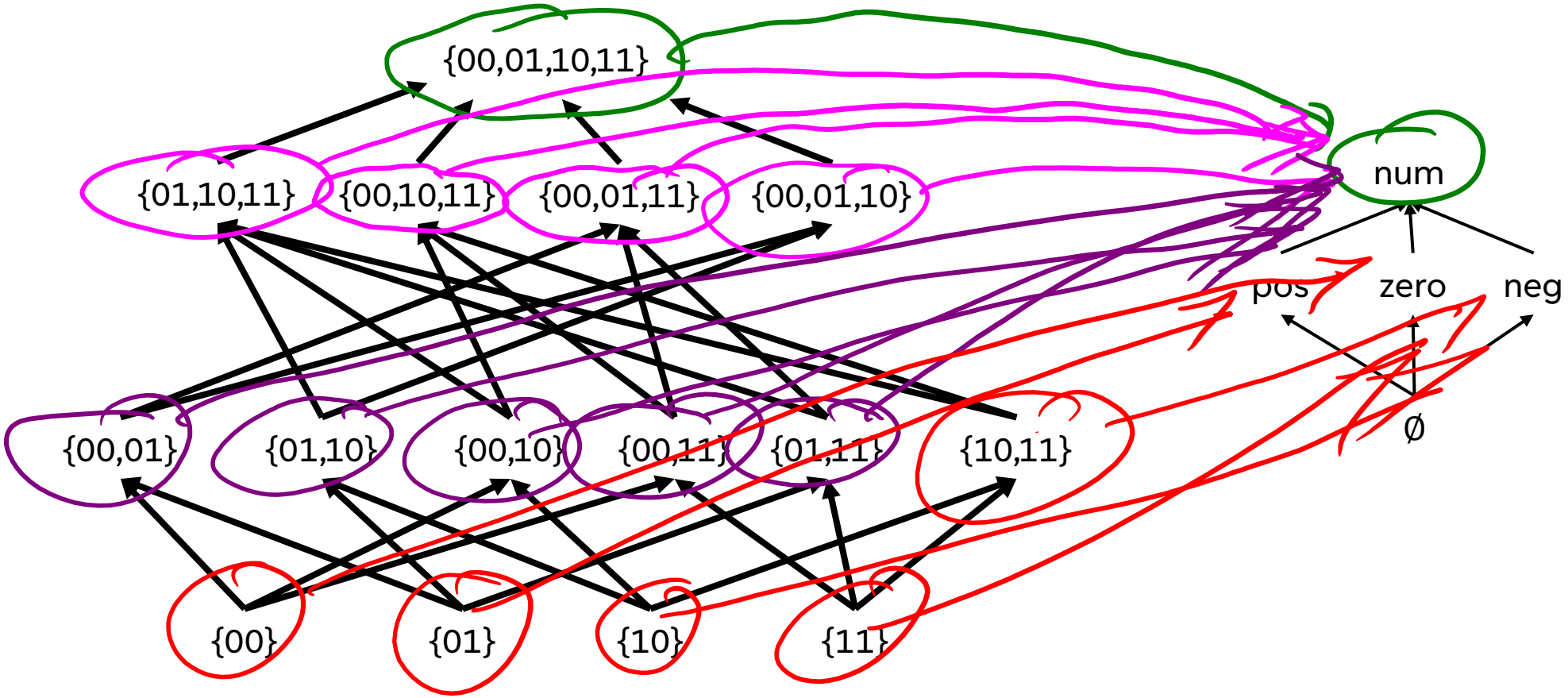
The shallower the lattice, the faster the fixpoint

Choose to approximate the lattice

LET'S CONSIDER A VERY APPROXIMATE LATTICE

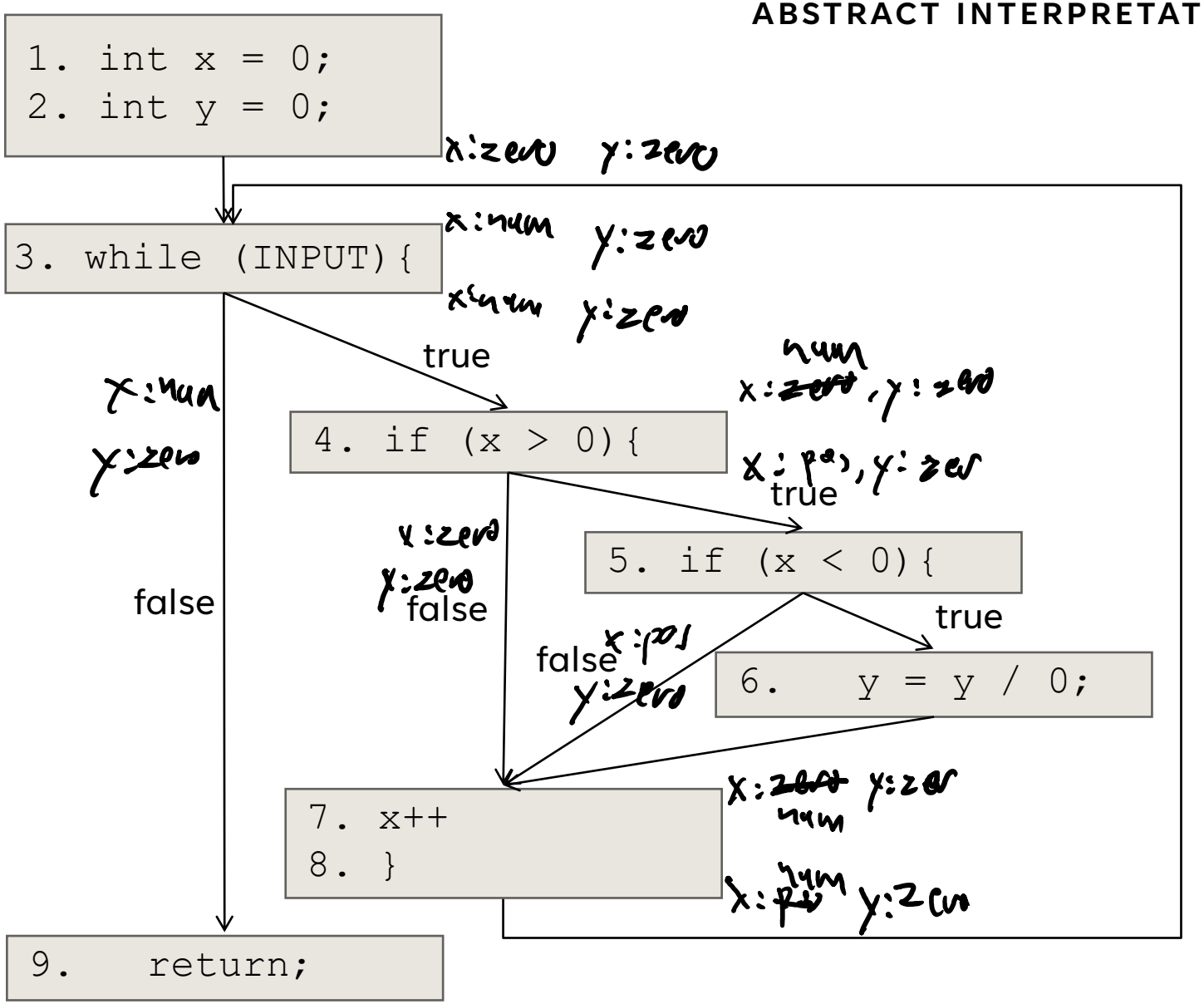
ABSTRACT INTERPRETATION

ABSTRACT DOMAIN OF SIGNS



ANALYSIS PRECISION

ABSTRACT INTERPRETATION




```
1.  int x = 0;
2.  int y = 0;
3.  while (INPUT){
4.   if (x > 0)
5.     if (x < 0)
6.       y = y / 0;
7.   x++;
8.   }
9.   return;
```

THE ABSTRACTION FUNCTION

ABSTRACT INTERPRETATION

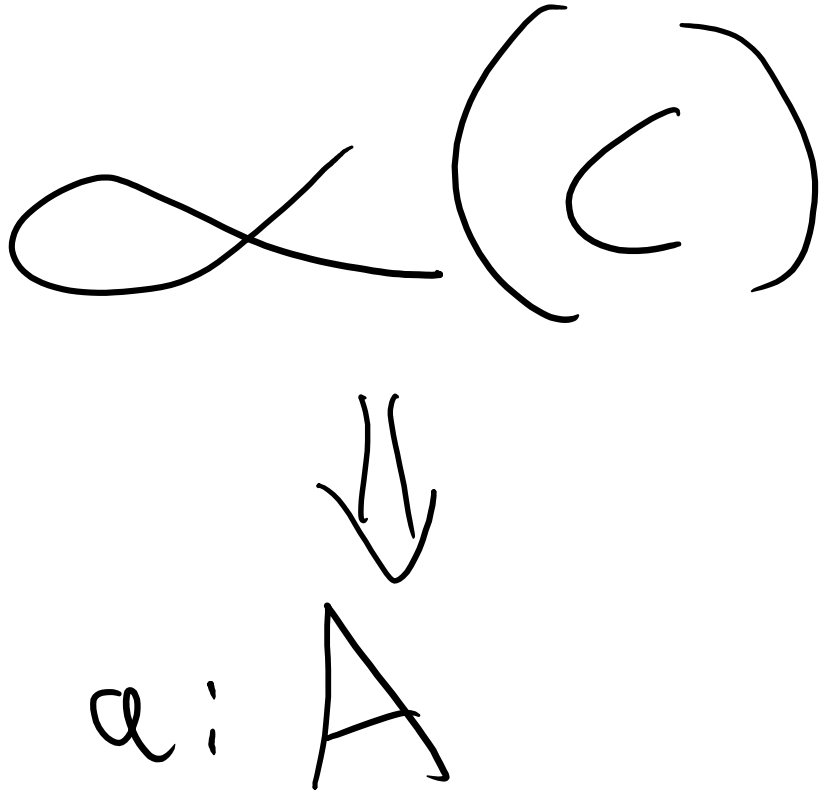
FROM THE CONCRETE DOMAIN TO THE ABSTRACT

$\alpha(\{0\})$ = zero

$\alpha(S)$ = if all values in S are greater than 0 then pos

else if all values in S are less than 0 then neg

else unk



THE CONCRETIZATION FUNCTION

ABSTRACT INTERPRETATION

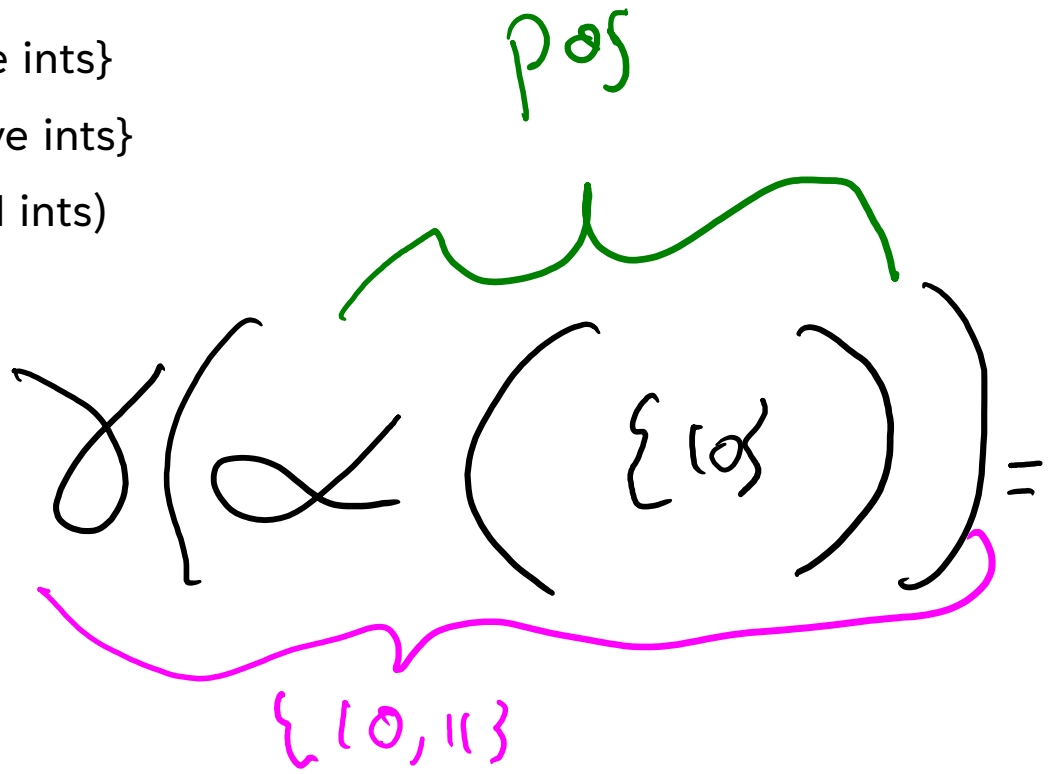
FROM THE ABSTRACT DOMAIN TO THE CONCRETE

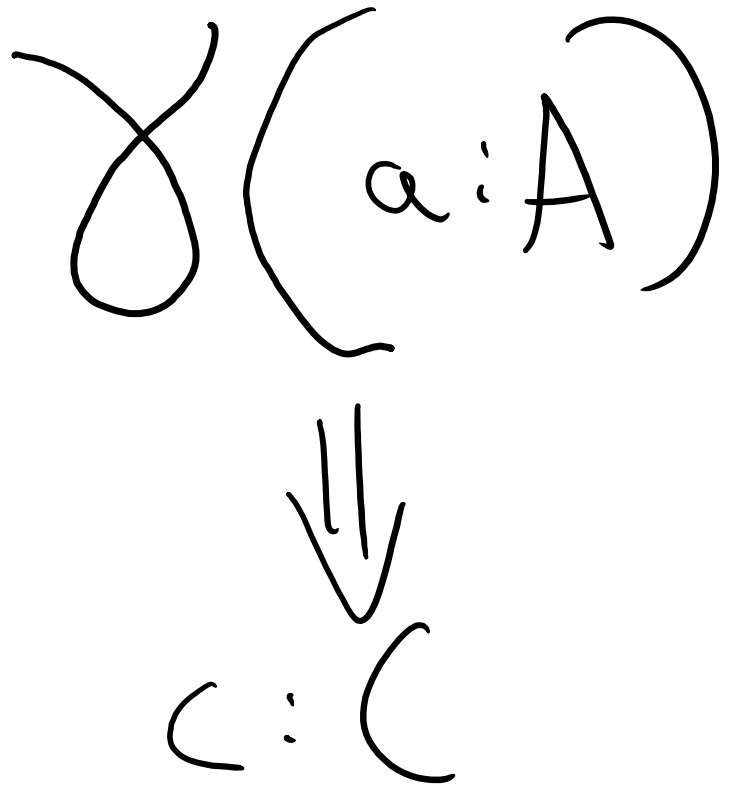
γ(zero) = {0}

γ(pos) = {all positive ints}

γ(neg) = {all negative ints}

γ(num) = Int (i.e., all ints)





A DETOUR INTO FORMALIZATION

ABSTRACT INTERPRETATION

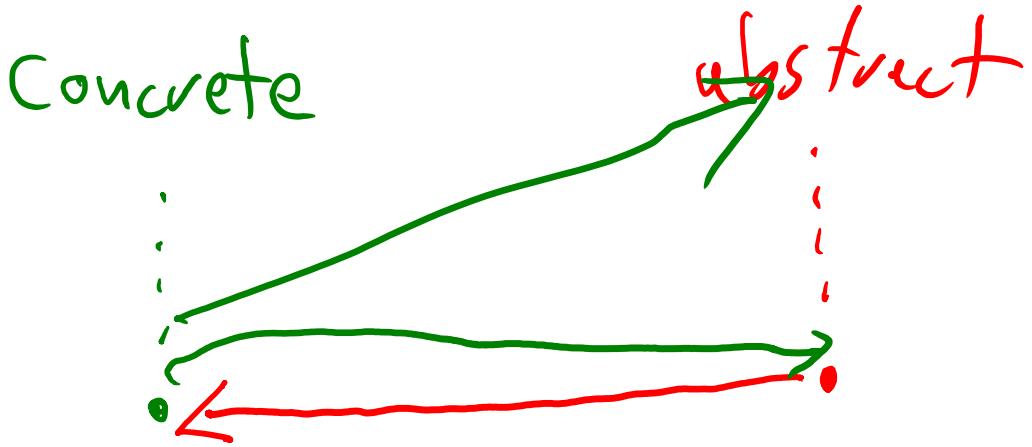
GALOIS CONNECTION

ABSTRACT INTERPRETATION

A *Galois connection* is a pair of functions, α and γ between two partially ordered sets $(C, \subseteq)$ and $(A, \leq)$, such that both of the following hold.

1. $\forall\, a \in A, c \in C: \alpha(c) \leq a$ iff $c \subseteq \gamma(a)$

2. $\forall\, a \in A: \alpha(\gamma(a)) \leq a$



END FORMALIZATION DETOUR

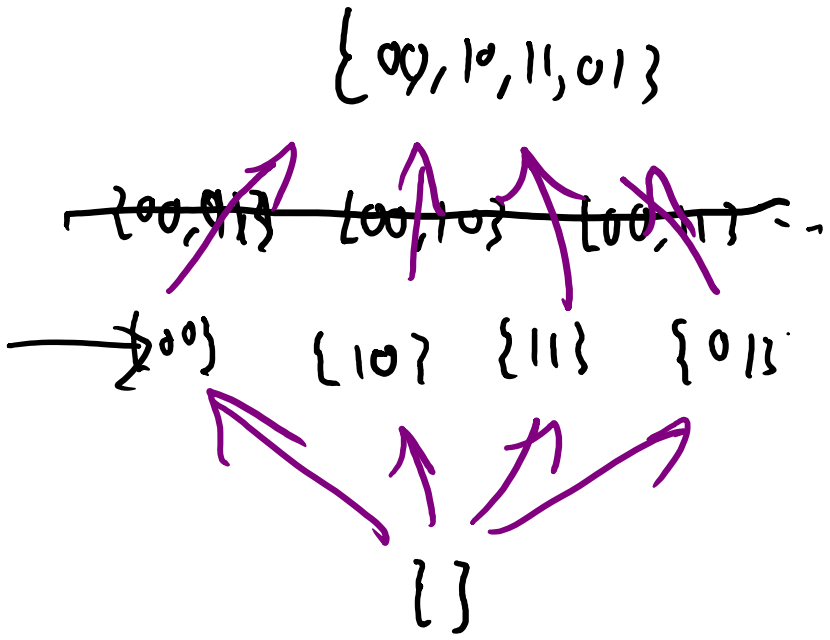
ABSTRACT INTERPRETATION

ABSTRACT DOMAINS IN PRACTICE

STATIC ANALYSIS

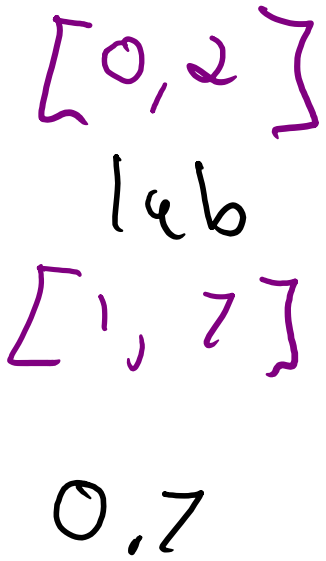
"SINGLETON INTEGER SETS"

- You know the number, or you don't



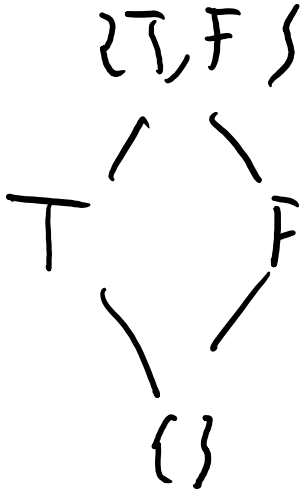
INTERVALS

- You know a concrete range



PROPERTY EXISTENCE

- A property does or does not hold



SECTION SUMMARY
STATIC ANALYSIS

STATIC ANALYSIS GIVES US AN IMPORTANT GUARANTEE

- Completeness of bug finding / Soundness of verification
- Thus far we've been using source code

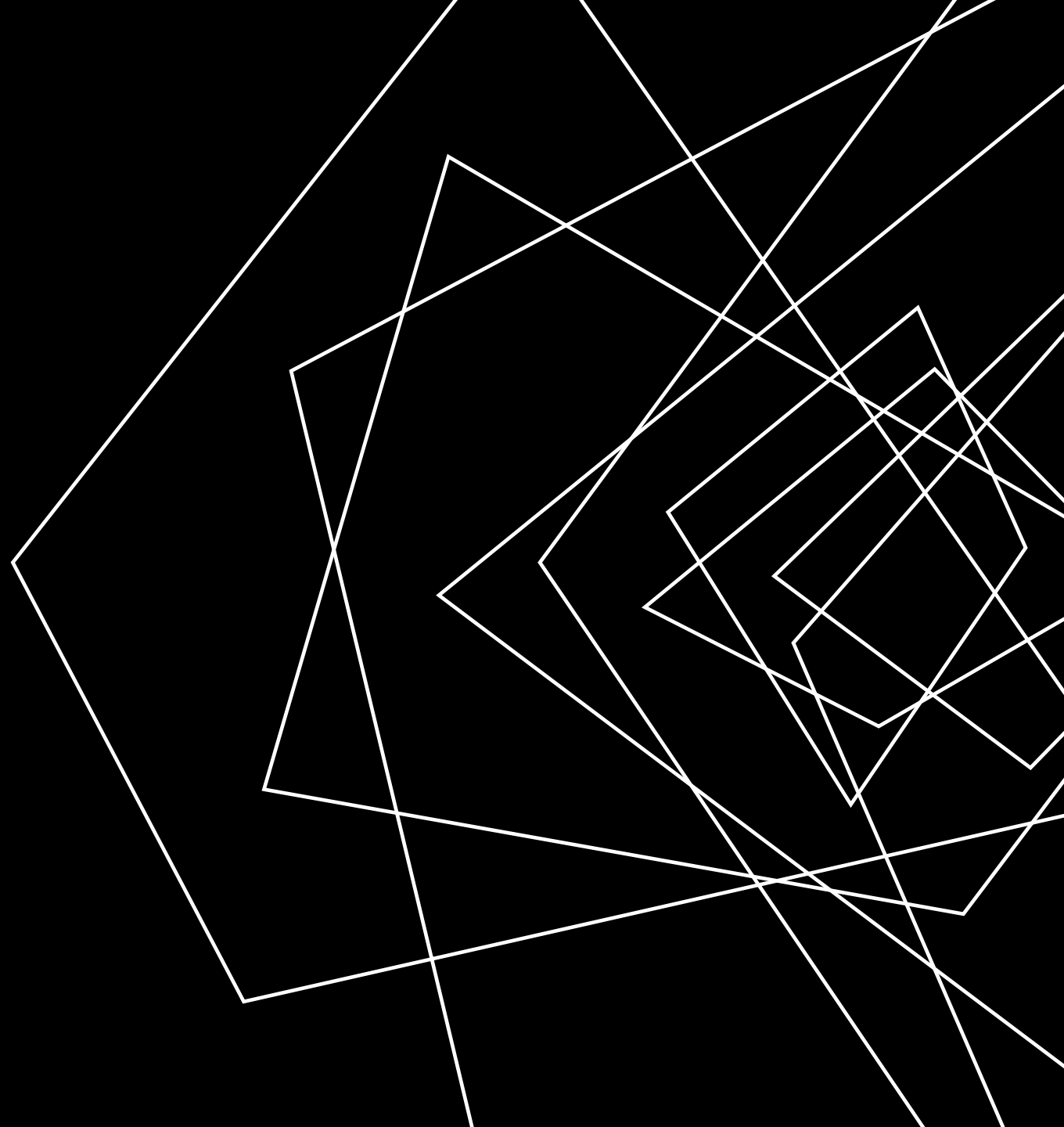
Anything that isn't crystal clear to a static analysis tool probably isn't clear to your fellow programmers, either. The classic hacker disdain for "bondage and discipline languages" is short-sighted – the needs of large, long-lived, multi-programmer projects are just different than the quick work you do for yourself.

- John Carmack's Static Code Analysis post

LECTURE OUTLINE

- Abstract Interpretation
- LLVM

Bonus stage unlocked!



APPLYING STATIC ANALYSIS

We Know the Theory of Static Analysis

- But how do we apply that theory in practice?
- Thus far we've been using source code as a target (that's limited!)

LLVM

LLVM: OVERVIEW

The LLVM Project is a collection of modular and reusable compiler and toolchain technologies. Despite its name, LLVM has little to do with traditional virtual machines. The name "LLVM" itself is not an acronym; it is the full name of the project.

Is this the LLVM logo?

LLVM

LLVM: OVERVIEW

The LLVM Project is a collection of modular and reusable compiler and toolchain technologies. Despite its name, LLVM has little to do with traditional virtual machines. The name "LLVM" itself is not an acronym; it is the full name of the project.

No, it's a Yu-gi-oh card!

LLVM

LLVM: OVERVIEW

The LLVM Project is a collection of modular and reusable compiler and toolchain technologies. Despite its name, LLVM has little to do with traditional virtual machines. The name "LLVM" itself is not an acronym; it is the full name of the project.

***This* is the actual LLVM logo**

LLVM: WHY WE USE IT

LLVM: OVERVIEW

Let's consider two goals of software security evaluation

- Discover vulnerabilities in "trusted" code
- Discover attacks in "untrusted" code

Most useful for this one

LLVM: WHY WE USE IT

LLVM: OVERVIEW

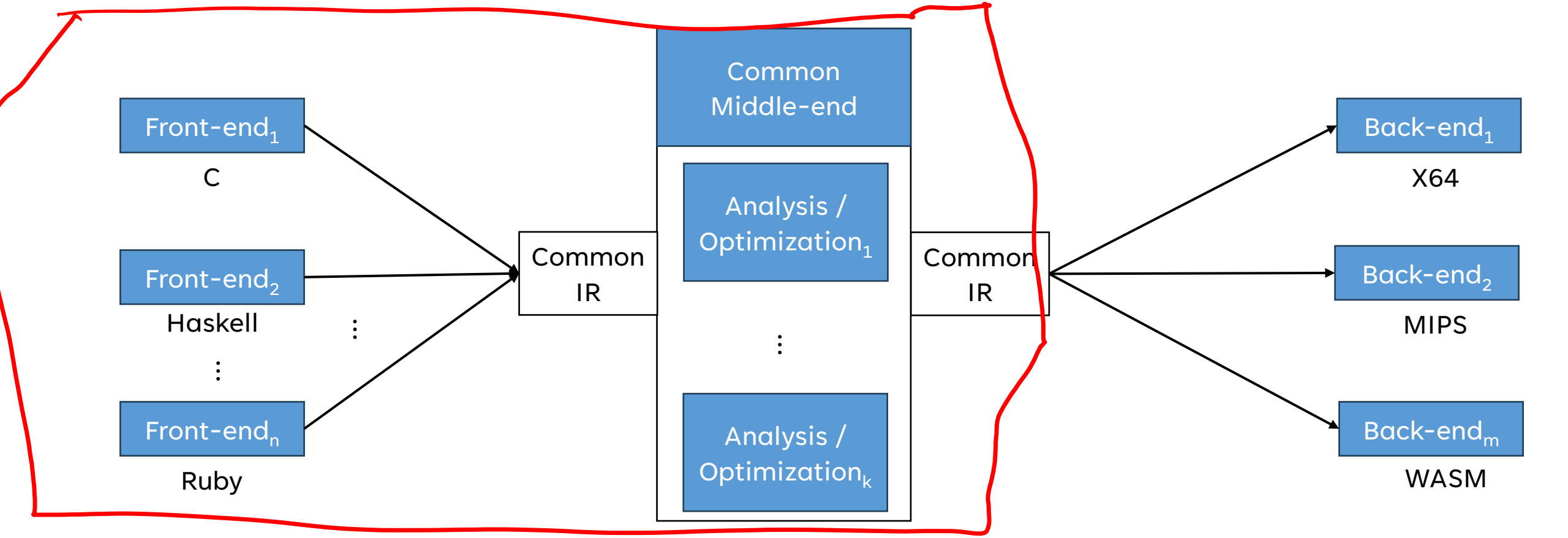
Genericity

LLVM: WHY WE USE IT

LLVM: OVERVIEW

Genericity

LLVM High-Level Architecture



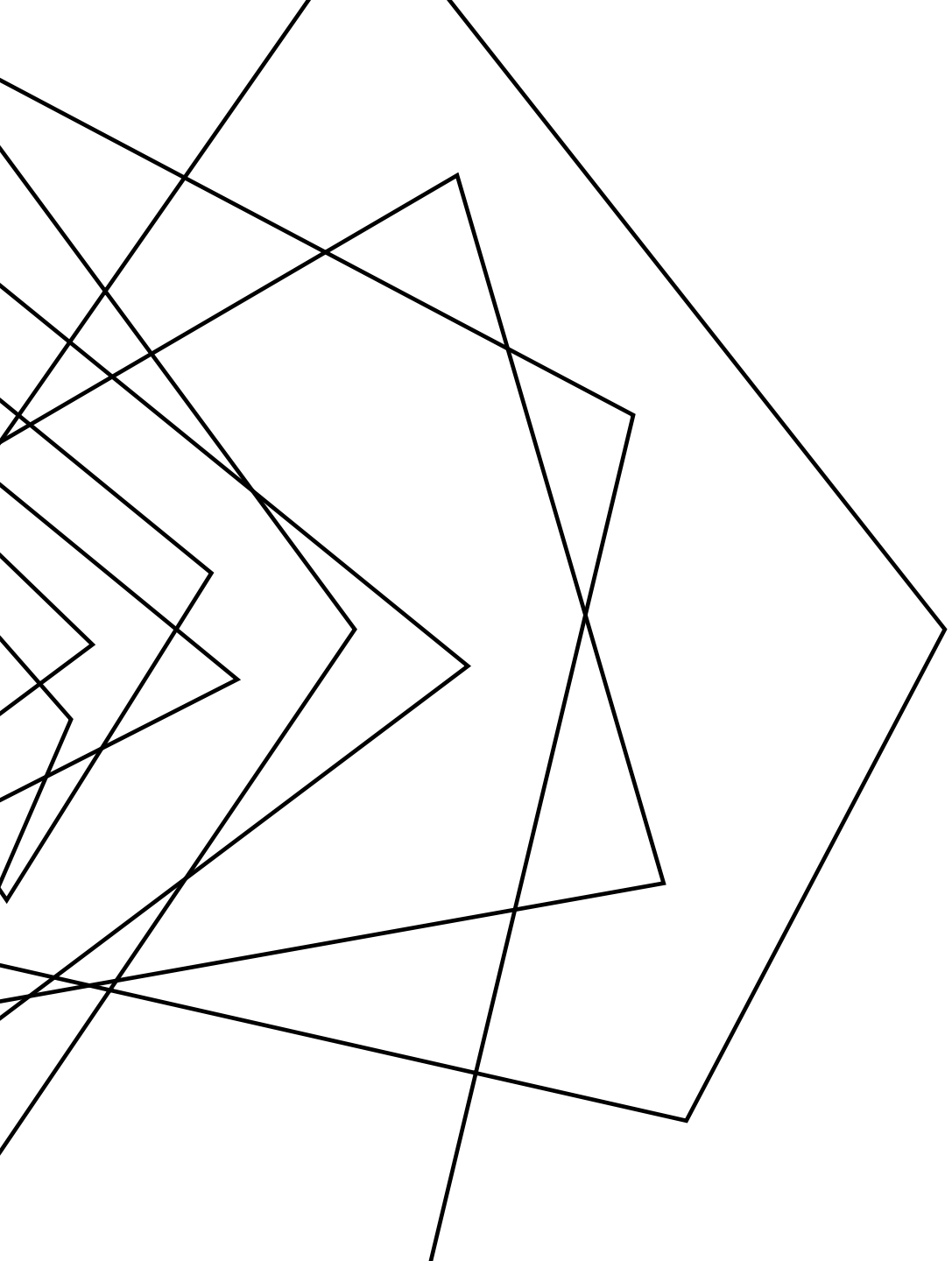
SECTION SUMMARY

LLVM

LLVM Provides a good basis for analysis

Allows us to avoid a lot of the parsing / language specific details that we view as boilerplate

Gives us a good analysis target in llvm bitcode



NEXT TIME

BEGIN LOOKING AT A PROGRAM REPRESENTATION WE CAN USE FOR TO BUILD OUR OWN ANALYSES