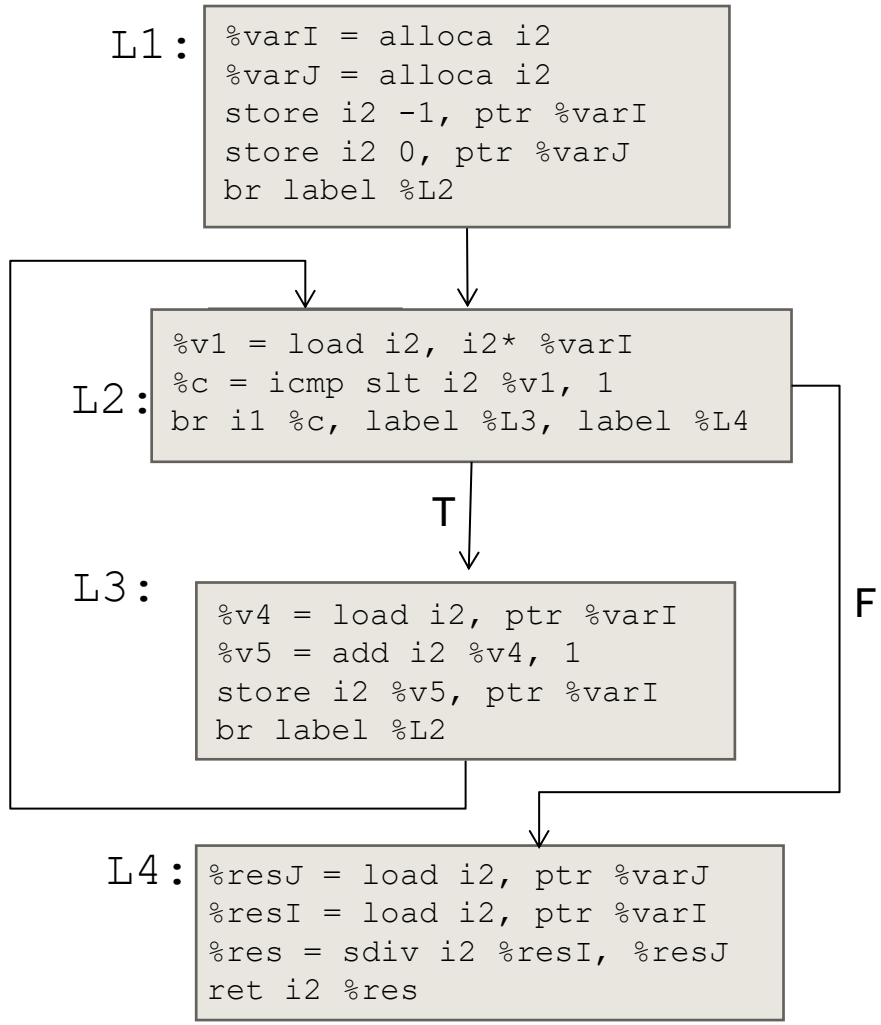
EXERCISE

Consider the CFG to the right, roughly equivalent to the C code below.
Indicate the value-set computed for `*varJ` at the entrance to L4 using the flow-sensitive saturation algorithm described in the last lecture

```
int2 i = -1;
int2 j = 0;
while (i < 1){
  i++;
  j = 1;
}
return i/j;
```

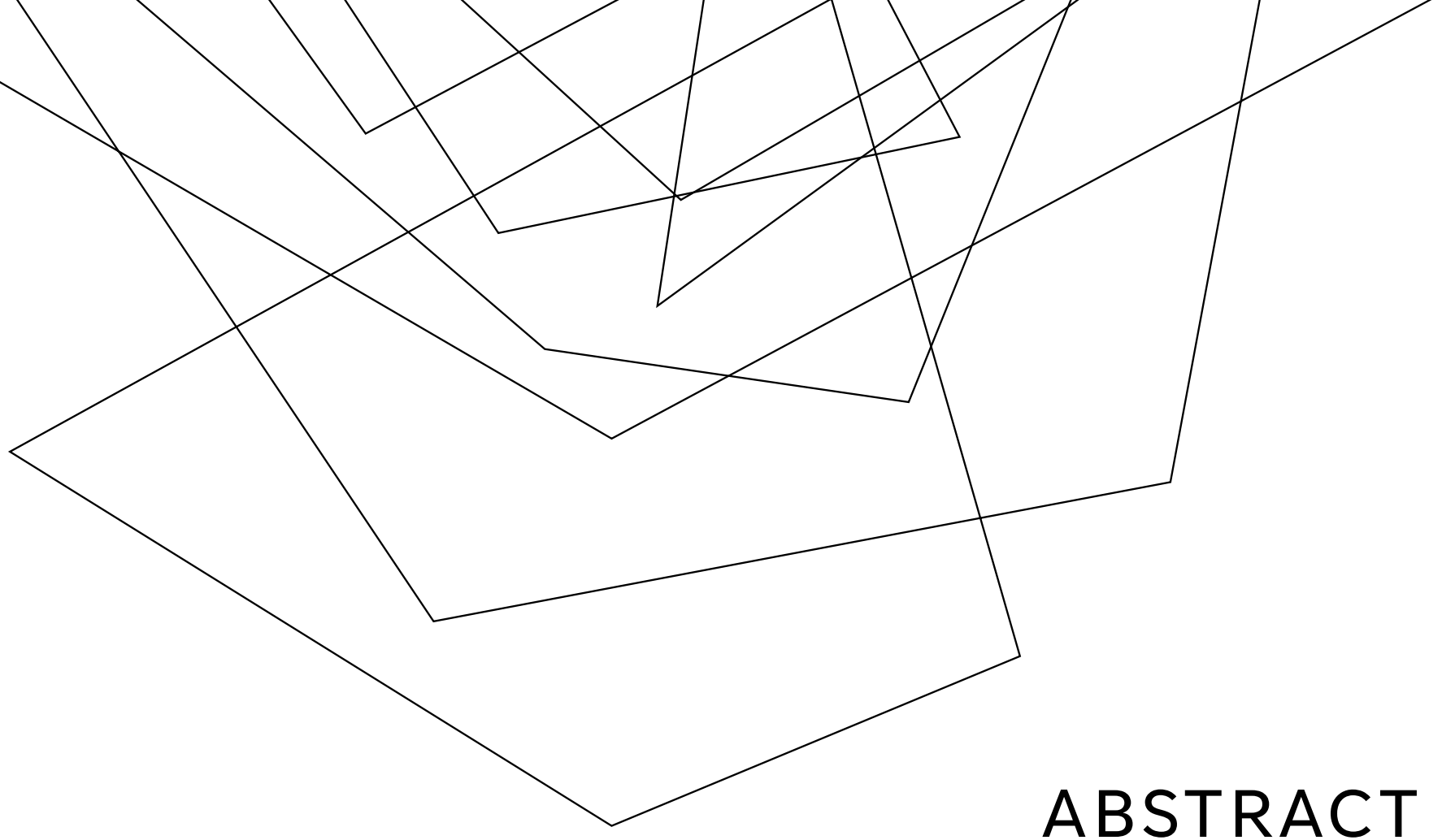


EXERCISE: SOLUTION

ADMINISTRIVIA AND ANNOUNCEMENTS

Test 1 is Friday!

Review Session tomorrow, 7 – 9, LEEP2 2420



ABSTRACT INTERPRETATION

EECS 677: Software Security Evaluation

Drew Davidson

LAST TIME: HANDLING LOOPS

REVIEW: STATIC ANALYSIS

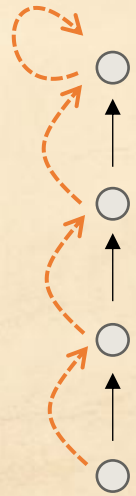
EXTENDING OUR BASIC DATAFLOW TO LOOPS

Problems

- Circular data dependence
  Introduce a TBD Value - ??? or ¯_(ツ)_/¯
- No obvious end-point (will the analysis halt?)
  Ran analysis until fact-set saturation

Guaranteeing Saturation

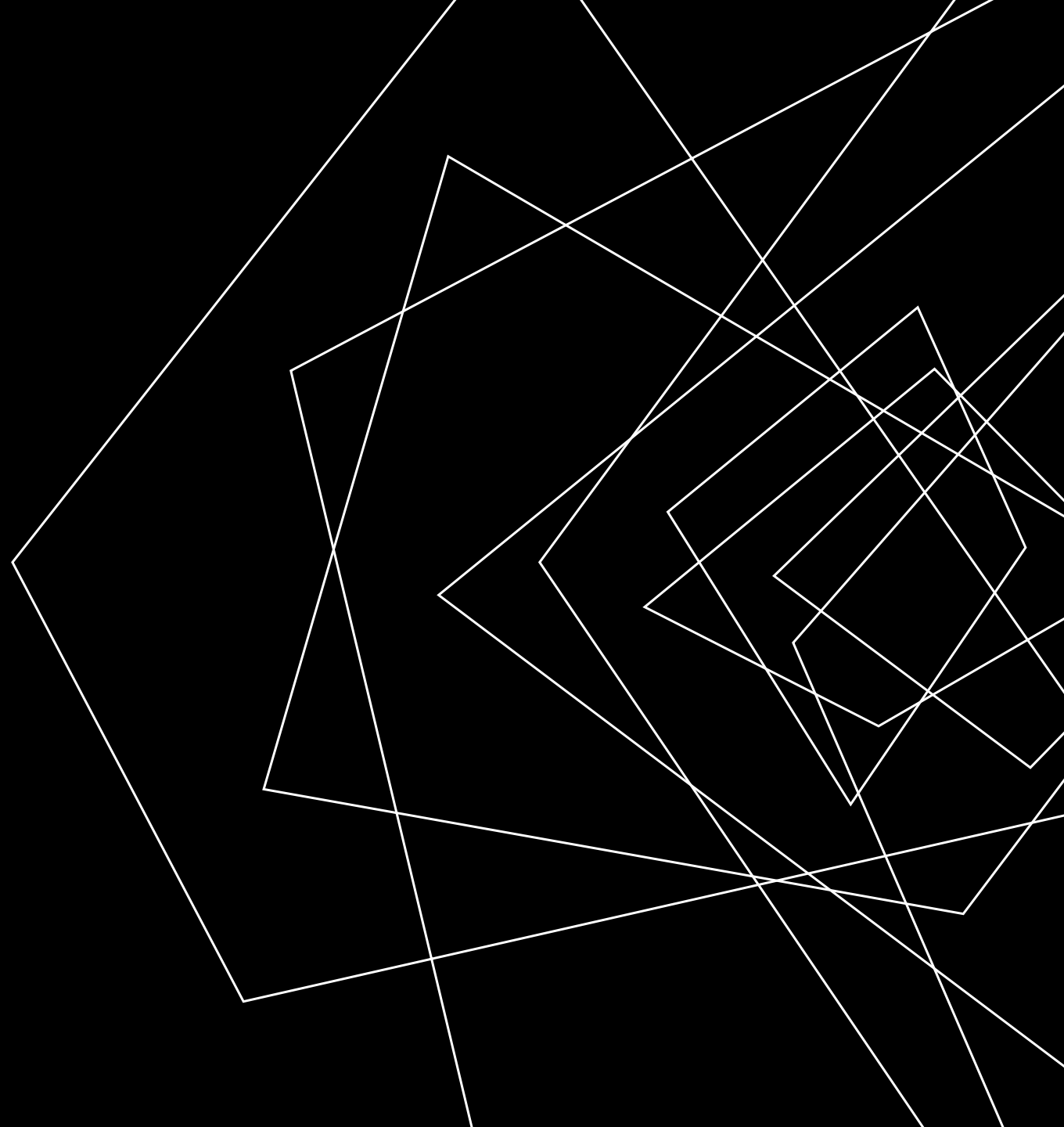
- Applying a transform is constant-time
- The fact sets are finite
- The fact sets never shrink



Perform an operation until it stops making progress

LECTURE OUTLINE

- Formalizing Saturation
- Exploring Larger Programs
- Abstract Domains



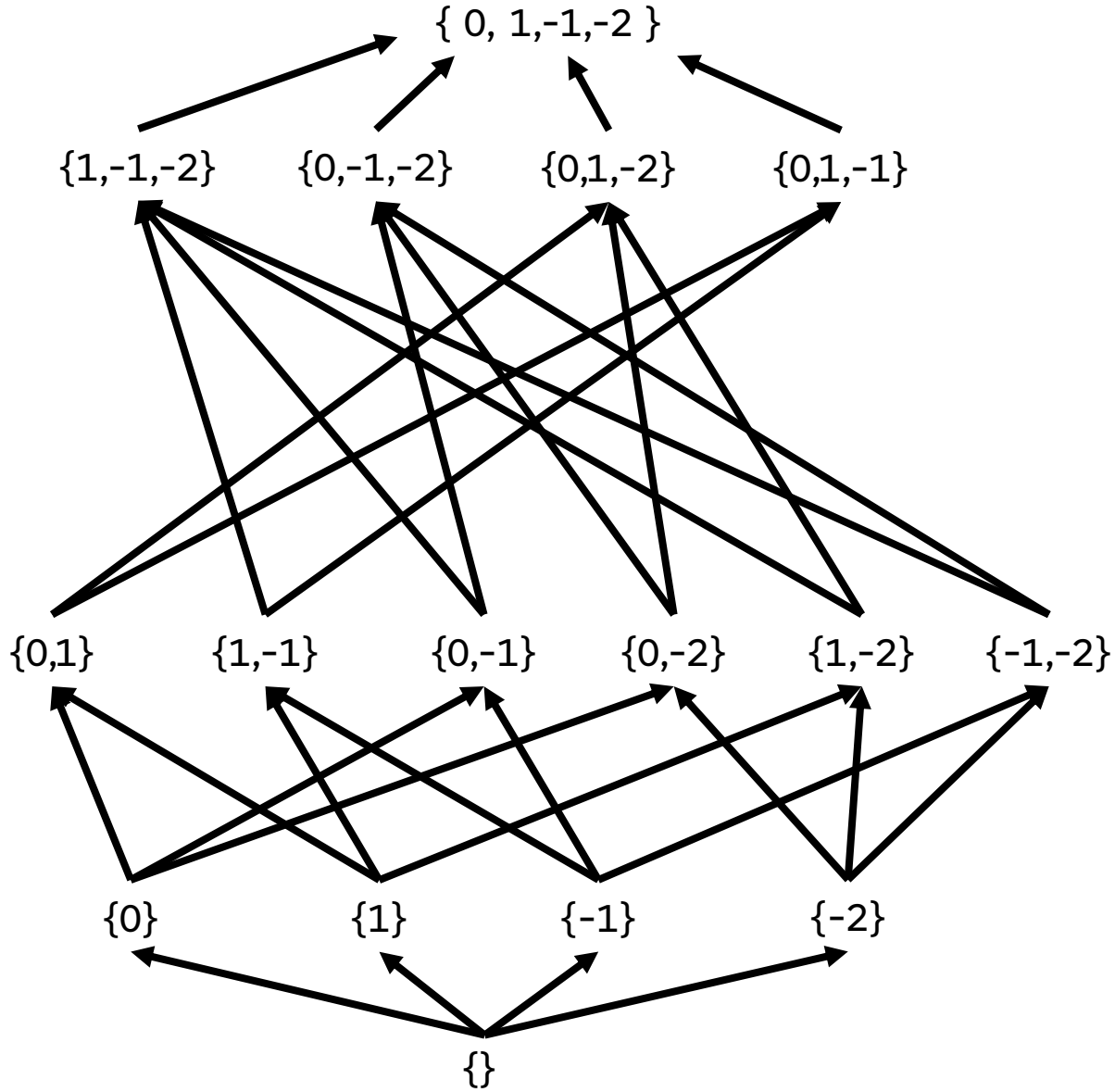
FACT SET MONOTONICITY

DATAFLOW FRAMEWORKS

WHAT DOES IT MEAN THAT A FACT SET "NEVER SHRINKS"?

- Implicitly, partial ordering by set inclusion...
- Merge operation (here, union) never moves down the partial ordering

Connecting to formal properties provides the guarantees we crave



DOMAIN NEEDS

DATAFLOW FRAMEWORKS

Some Basic Definitions

A **partially-ordered set** (poset) is a set S and a partial ordering ⊆, such that the ordering ⊆ is:

- Reflexive
- Anti-symmetric
- Transitive

A **lattice** is a poset in which each pair of elements has

- A least upper bound (the *join*)

  for x and y, the join z is defined such that:

  - x ⊆ z and
  - y ⊆ z and
  - for all w such that x ⊆ w and y ⊆ w, w ⊇ z

No upper bound lower than z

z is actually an upper bound

- A greatest lower bound (the *meet*)

  basically the same deal, but reversed

A **complete lattice** is a lattice in which all subsets have a meet and join

Example 1: S: English words, ⊆ substring

Poset: ✓ Lattice: ✗

Example 2: S: English words, ⊆ shorter or equal in length

Poset: ✗ Lattice: ✗

Example 3: S: integers, ⊆ as lte

Poset: ✓ Lattice: ✓

Example 4: S: integers, ⊆ as lt

Poset: ✗ Lattice: ✗

Example 5: S: set of all sets of letters, ⊆ is subset

Poset: ✓ Lattice: ✓

FACT DATATYPE NEEDS

DATAFLOW FRAMEWORKS

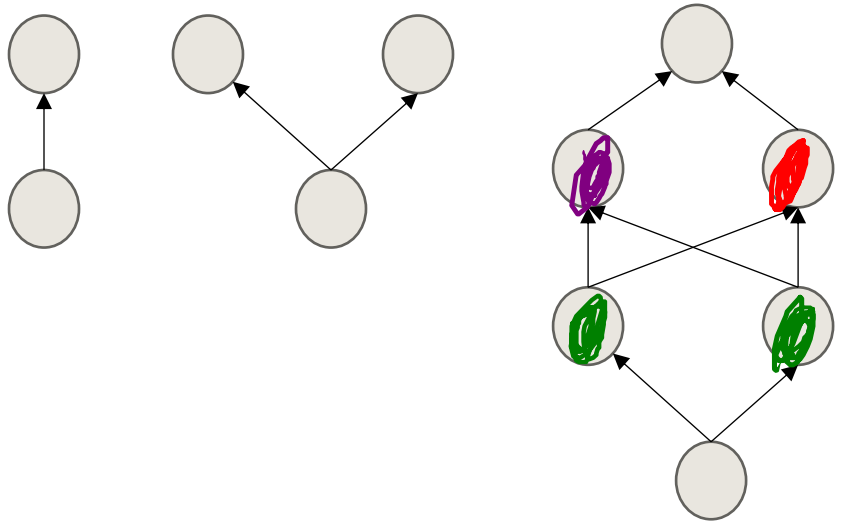
FORMAL(ISH) REQUIREMENTS

- A fact datatype (ideally of unbounded size)
- An ordering that indicates progress
- Unique solution
- A guarantee of a finite number of steps to hit the maximum value
- An update step that never loses progress

TRY-OUTS

- poset
- Lattice
- Complete lattice

POSET



FACT DATATYPE NEEDS

DATAFLOW FRAMEWORKS

FORMAL(ISH) REQUIREMENTS

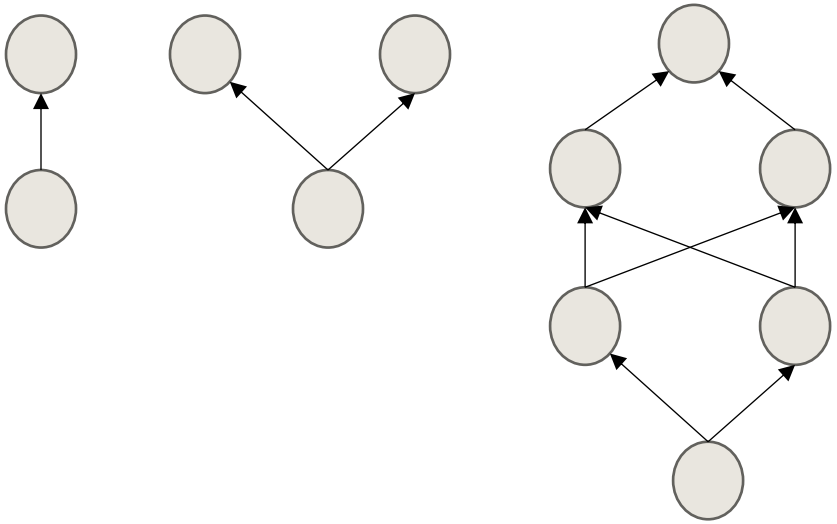
- A fact datatype (ideally of unbounded size)
- An ordering that indicates progress
- Unique solution
- A guarantee of a finite number of steps to hit the maximum value
- An update step that never loses progress

TRY-OUTS

- poset
- Lattice
- Complete lattice

LATTICE

Poset with a least upper bound and a greatest lower bound



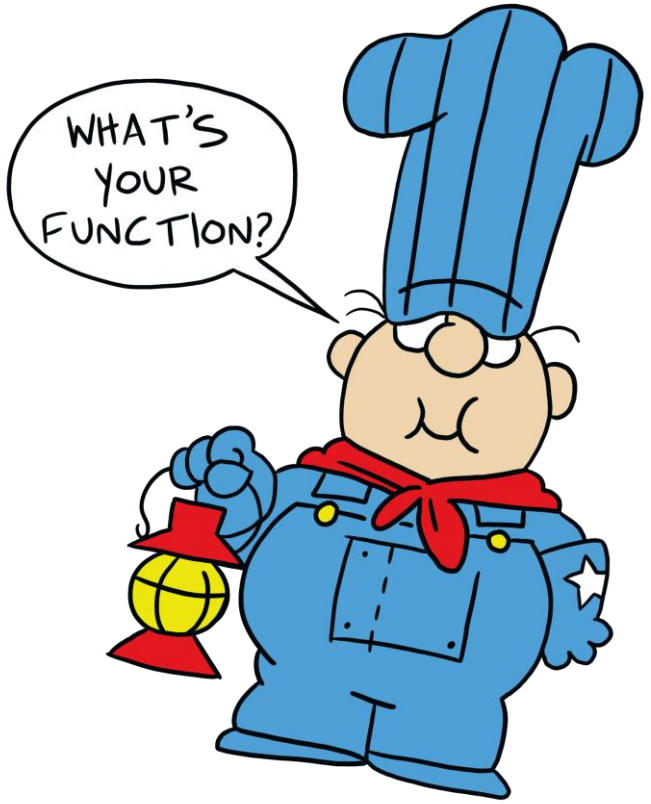
FUNCTION NEEDS

DATAFLOW FRAMEWORKS

SOME BASIC DEFINITIONS

A function f is a **monotonic function** if
$x \subseteq y$ implies $f(x) \subseteq f(y)$

An element z is a **fixpoint** of f iff $z = f(z)$



FUNCTION NEEDS

DATAFLOW FRAMEWORKS

SOME BASIC DEFINITIONS

A function f is a **monotonic function** if
x ⊆ y implies f(x) ⊆ f(y)

An element z is a **fixpoint** of f iff z = f(z)

Example

$f(x) = x \cup \{ b \}$

$f(\{b\}) = \{ b \}$ ← $\{b\}$ is a fixpoint of f

$f(\{a\}) = \{ a, b \}$ ← $\{a\}$ is **not** a fixpoint of f

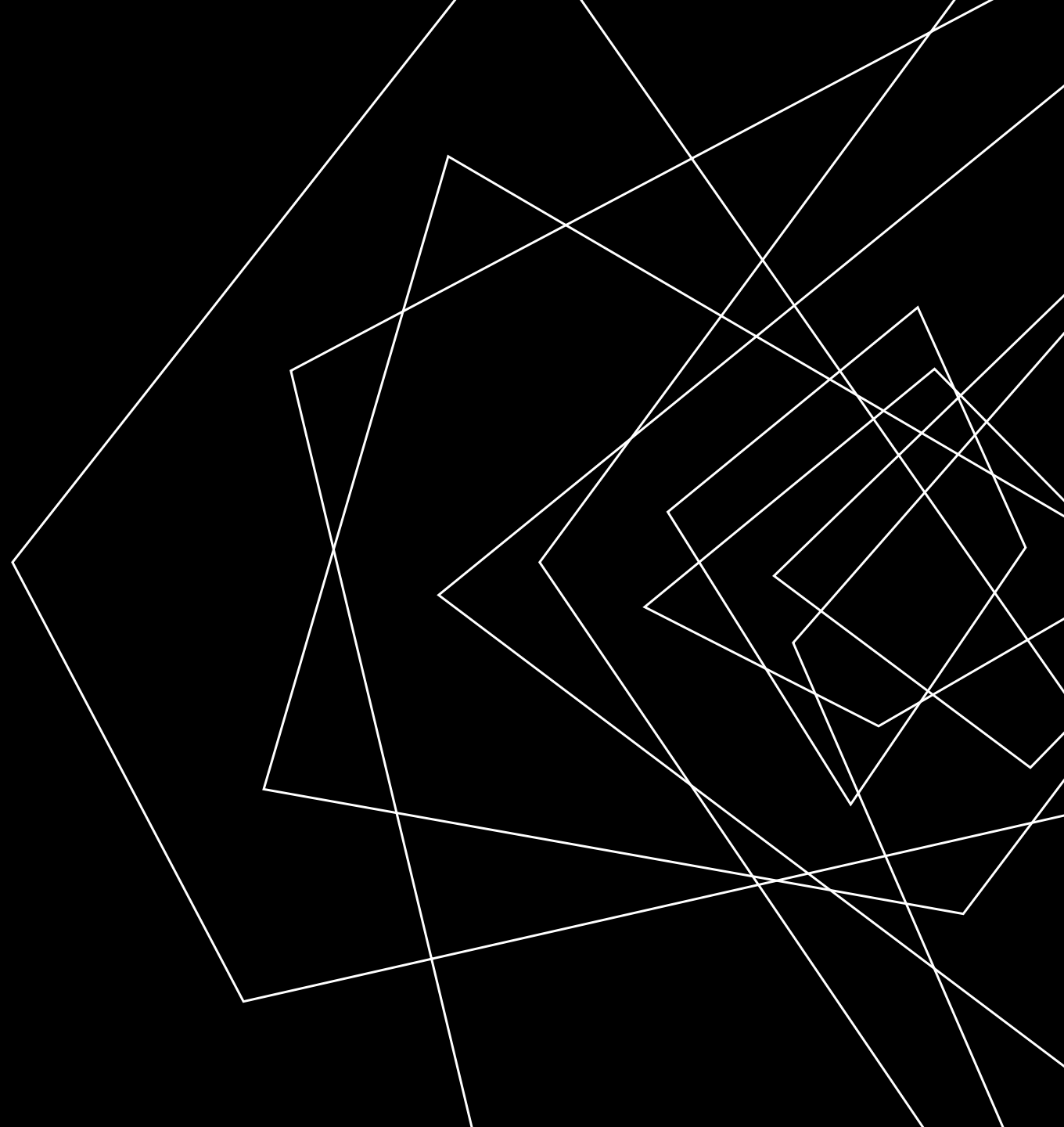
$f(\{a, b\}) = \{ a, b \}$ ← $\{a, b\}$ is a fixpoint of f

$f(f(\{a\}))$ is a fixpoint of f

$f(f(any\ set))$ is a fixpoint of f

LECTURE OUTLINE

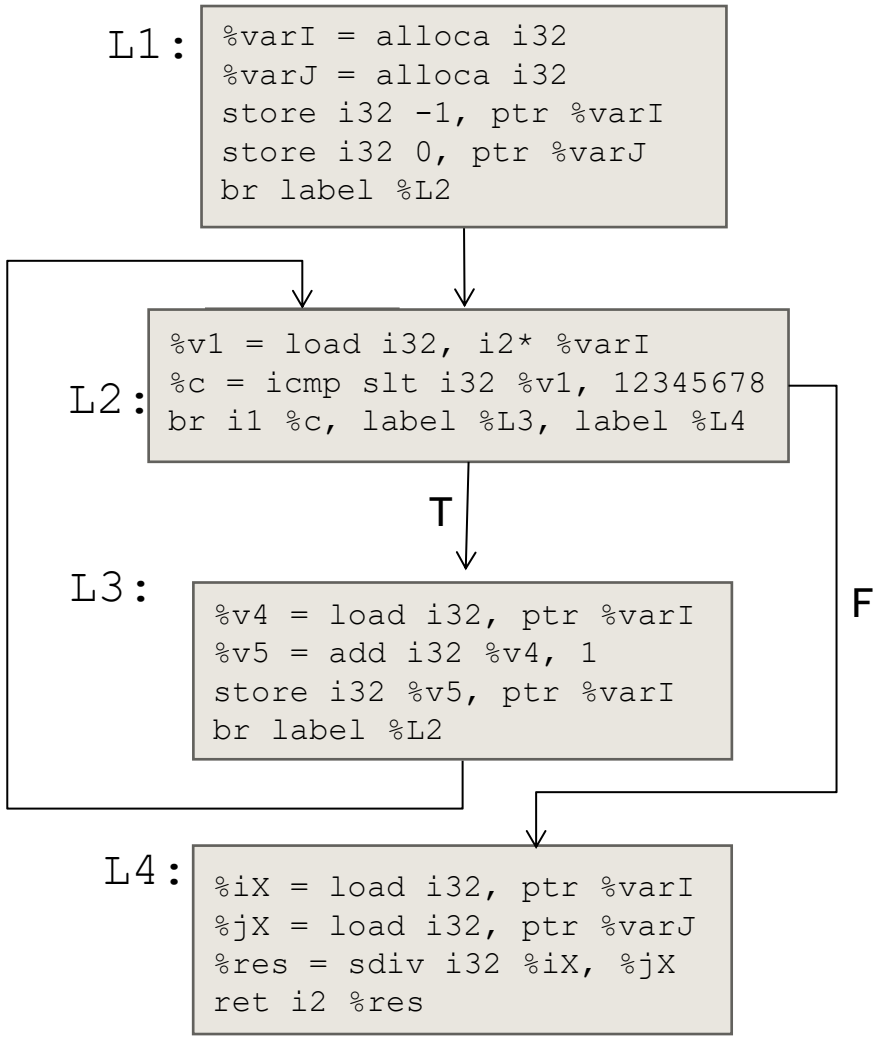
- Formalizing Saturation
- Exploring Larger Programs
- Abstract Domains



EXAMPLE

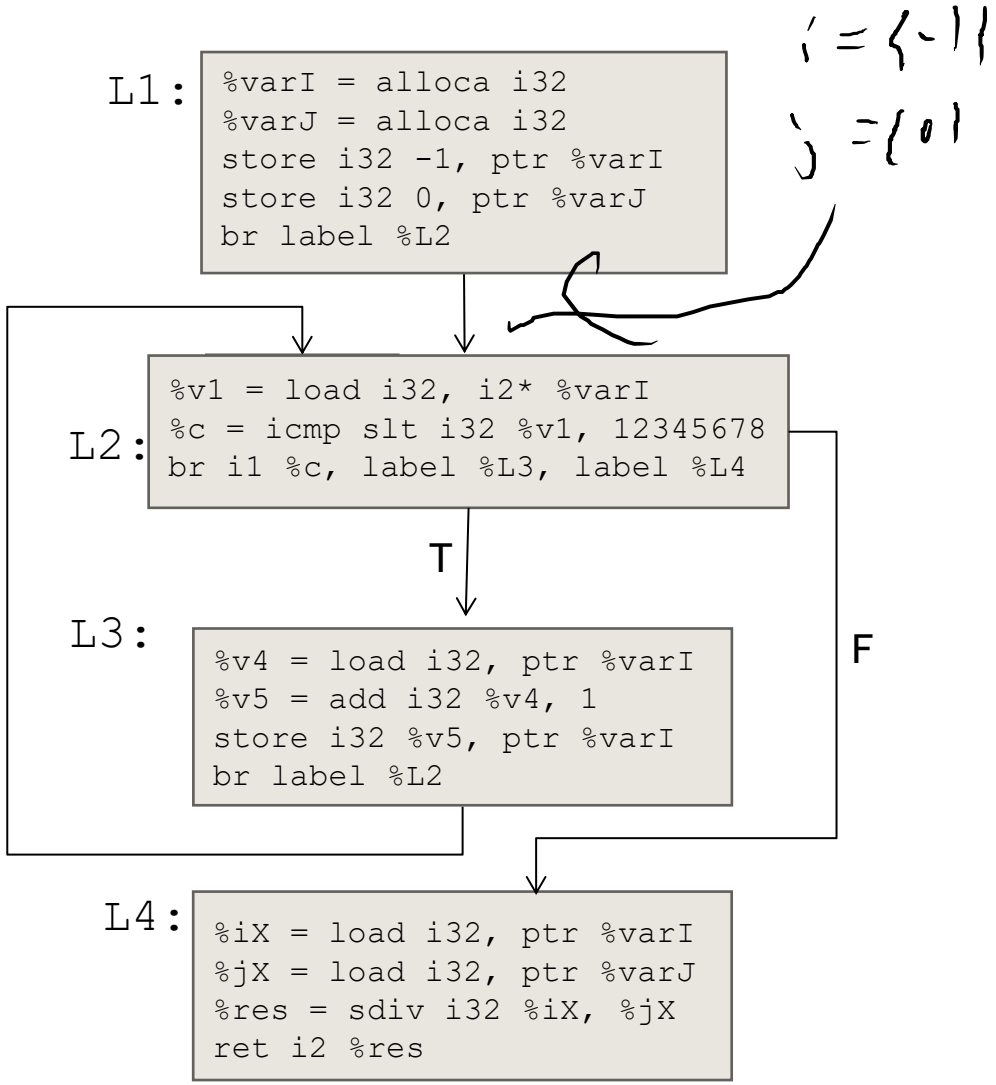
Consider the CFG to the right, roughly equivalent to the C code below.
Indicate the value-set computed for `*varJ` at the end of L6 using the flow-sensitive saturation algorithm described in the last lecture

```
int32 i = -1;
int32 j = 0;
while (i > 12345678){
   i++;
}
return 1/j;
```



EXAMPLE


INFORMATION OVERLOAD

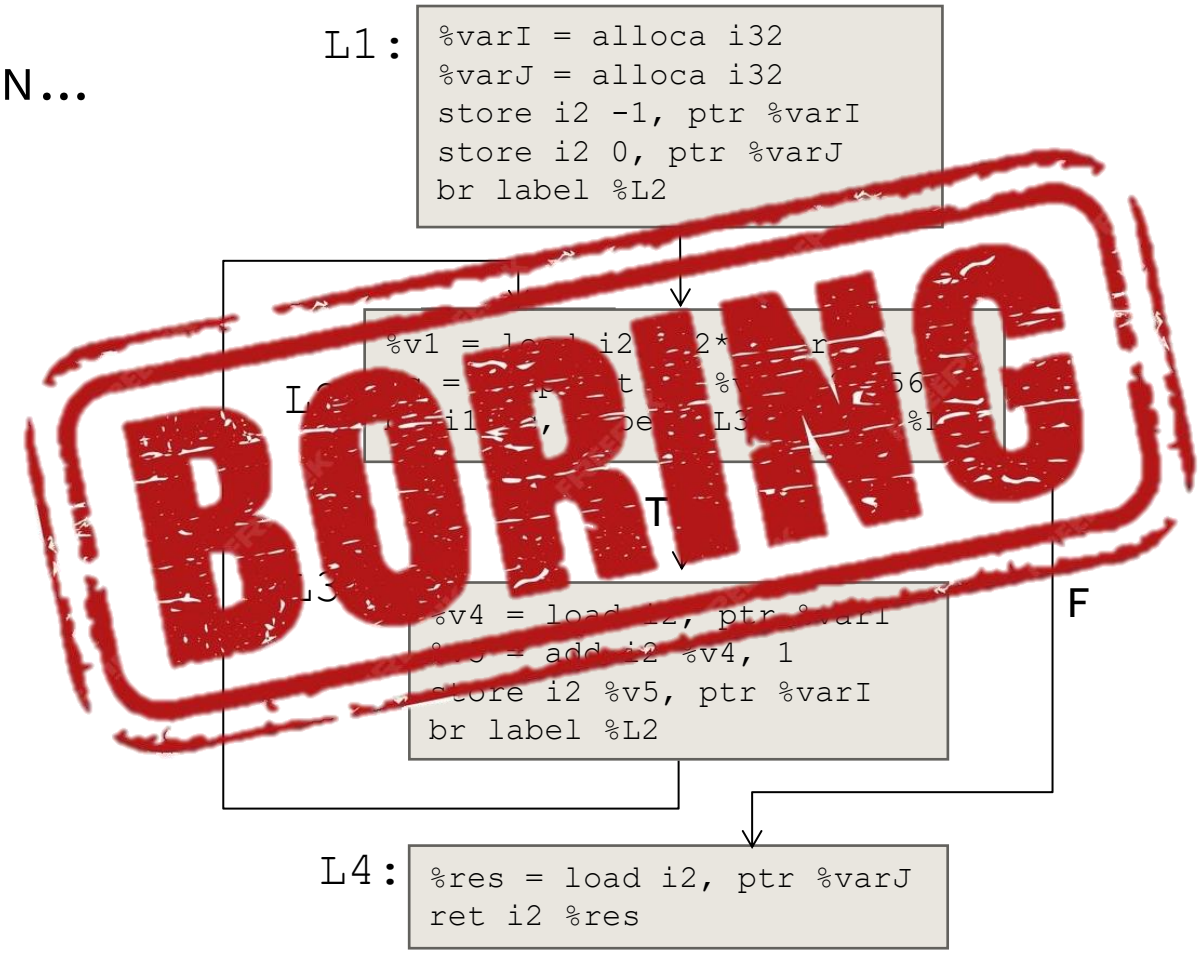

L1:
%varI = alloca i32
%varJ = alloca i32
store i32 -1, ptr %varI
store i32 0, ptr %varJ
br label %L2
L2:
%v1 = load i32, i2* %varI
%c = icmp slt i32 %v1, 12345678
br i1 %c, label %L3, label %L4
T
F
L3:
%v4 = load i32, ptr %varI
%v5 = add i32 %v4, 1
store i32 %v5, ptr %varI
br label %L2
L4:
%iX = load i32, ptr %varI
%jX = load i32, ptr %varJ
%res = sdiv i32 %iX, %jX
ret i2 %res

TMI – TOO MUCH INFORMATION!

ABSTRACT INTERPRETATION

EVENTUALLY WE'LL REACH SATURATION...

By the time we're all old and withered!

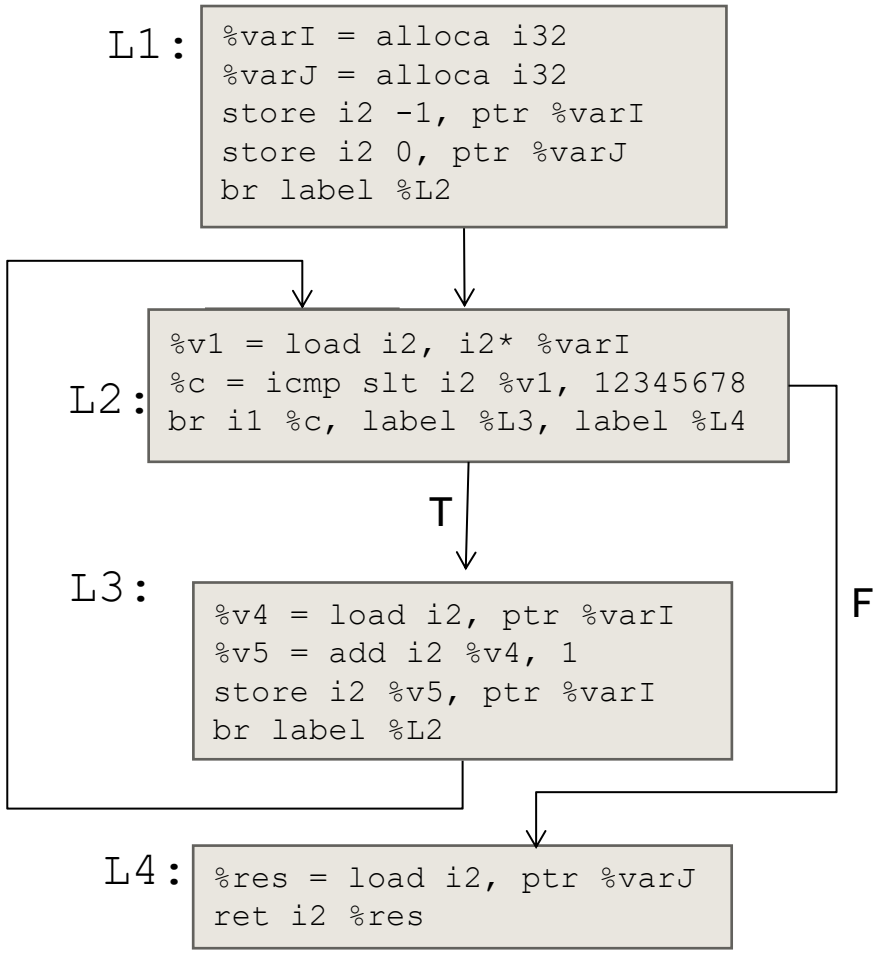


TMI – TOO MUCH INFORMATION!

ABSTRACT INTERPRETATION

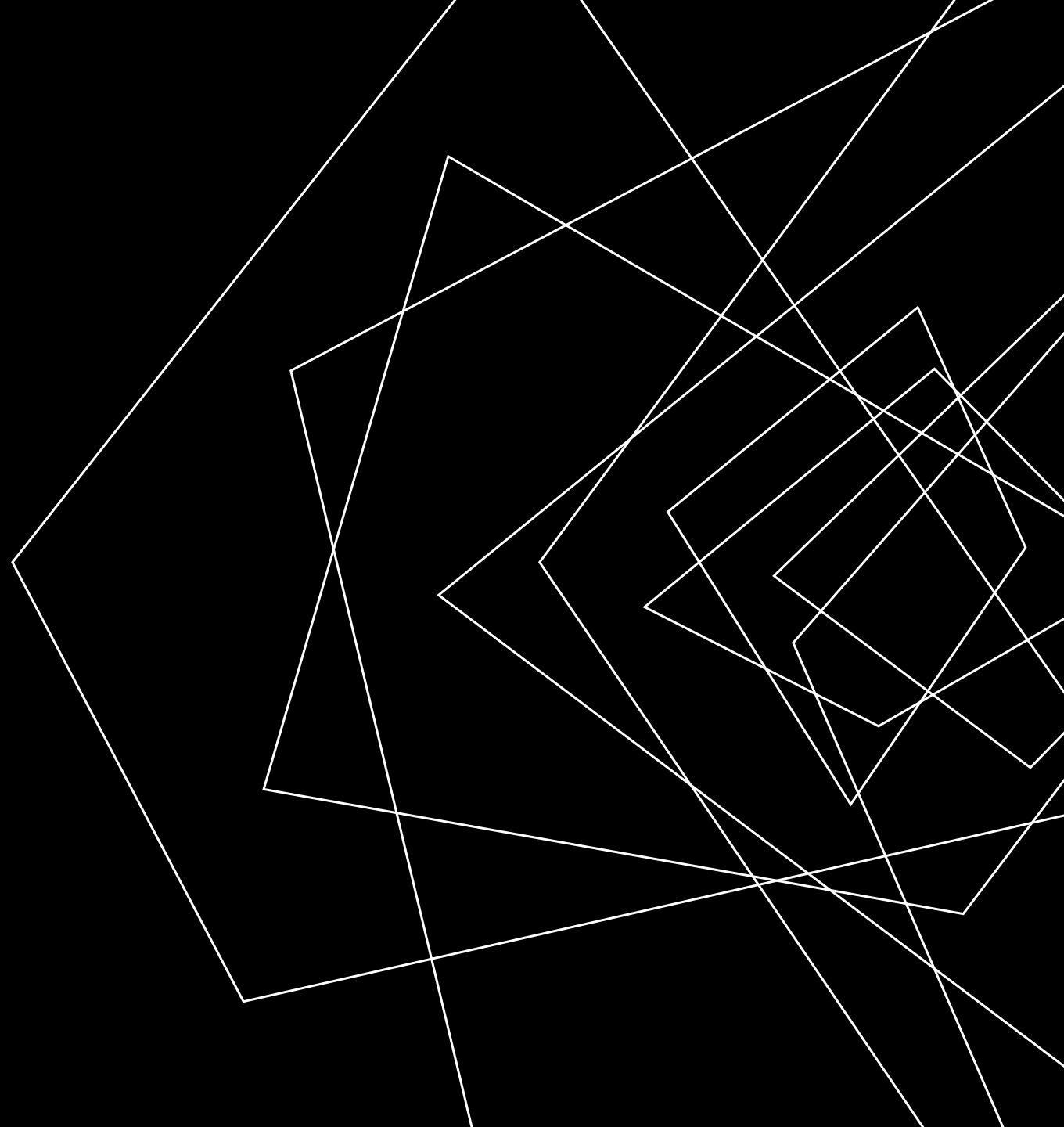
EVENTUALLY WE'LL REACH SATURATION...

By the time we're all old and withered!



LECTURE OUTLINE

- Formalizing Saturation
- Exploring Larger Programs
- Abstract Domains



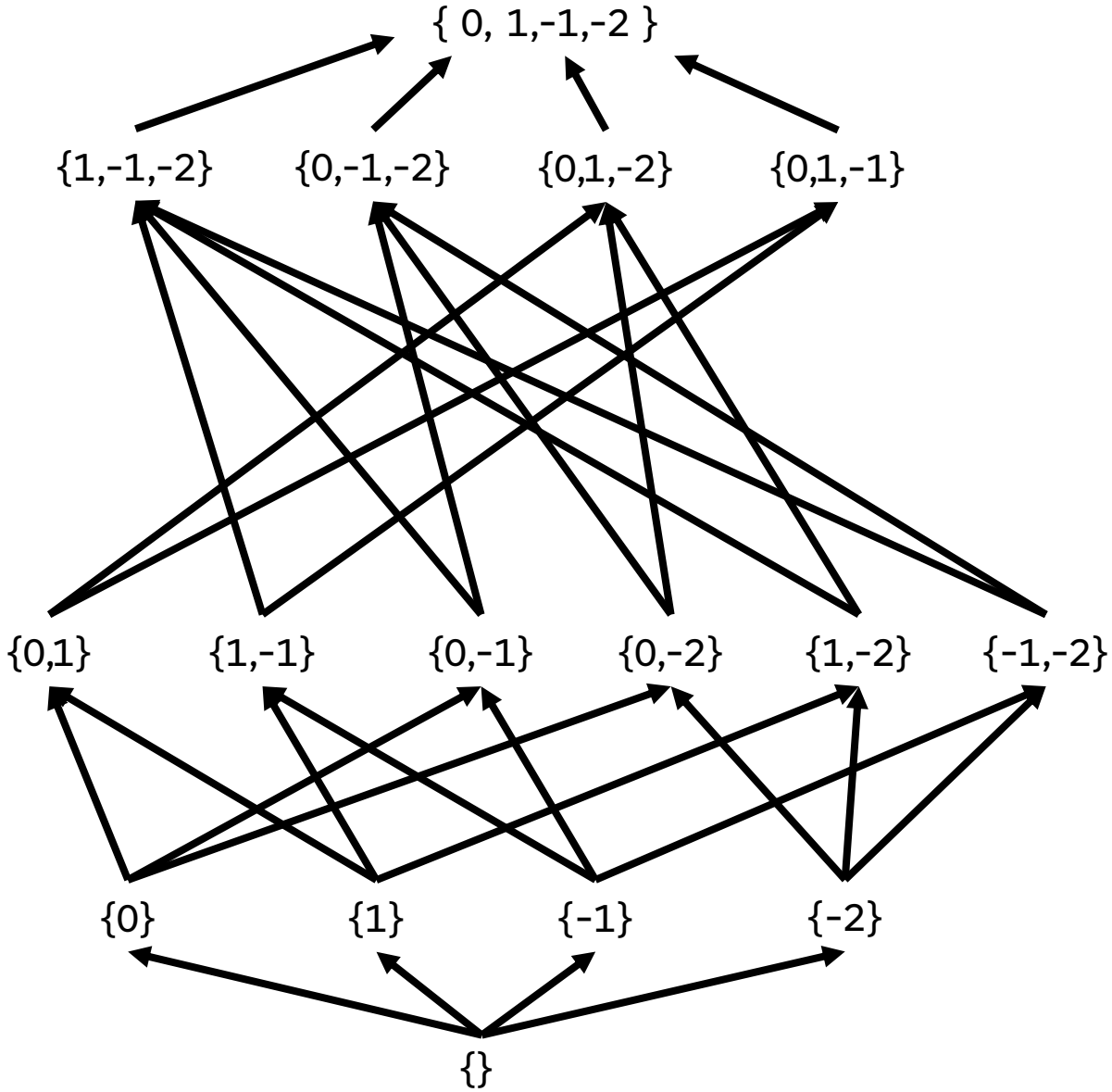
USING THE FORMALISMS

DATAFLOW FRAMEWORKS

WHAT DOES IT MEAN THAT A FACT SET "NEVER SHRINKS"?

- Implicitly, partial ordering by set inclusion...
- Merge operation (here, union) never moves down the partial ordering

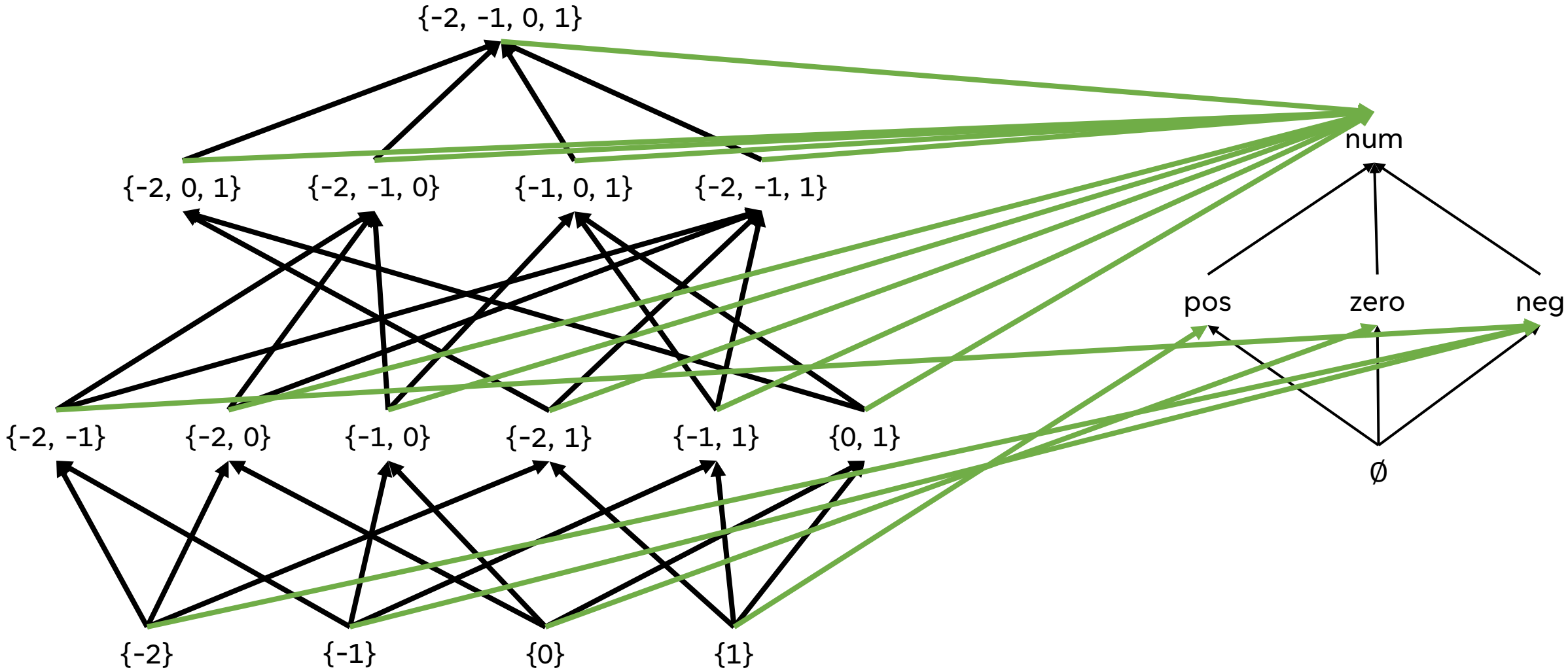
Connecting to formal properties provides the guarantees we crave



A VERY APPROXIMATE LATTICE

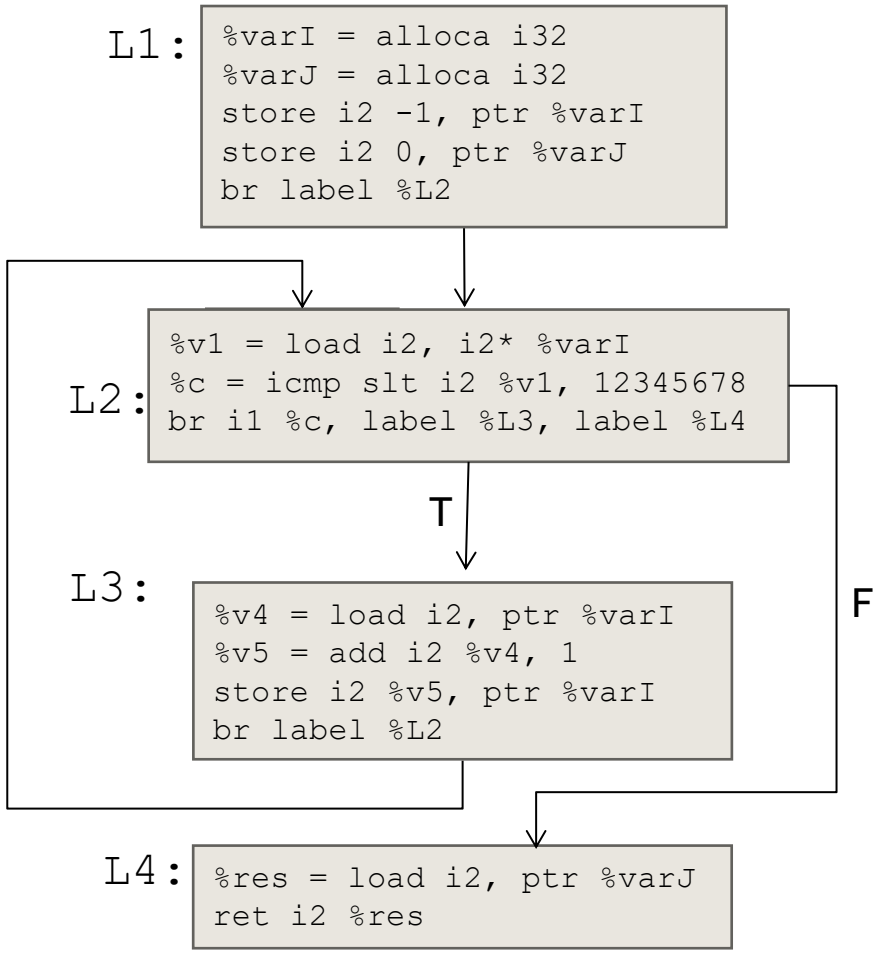
ABSTRACT INTERPRETATION

ABSTRACT DOMAIN OF SIGNS



ANALYSIS WITH ABSTRACT DOMAIN OF SIGNS

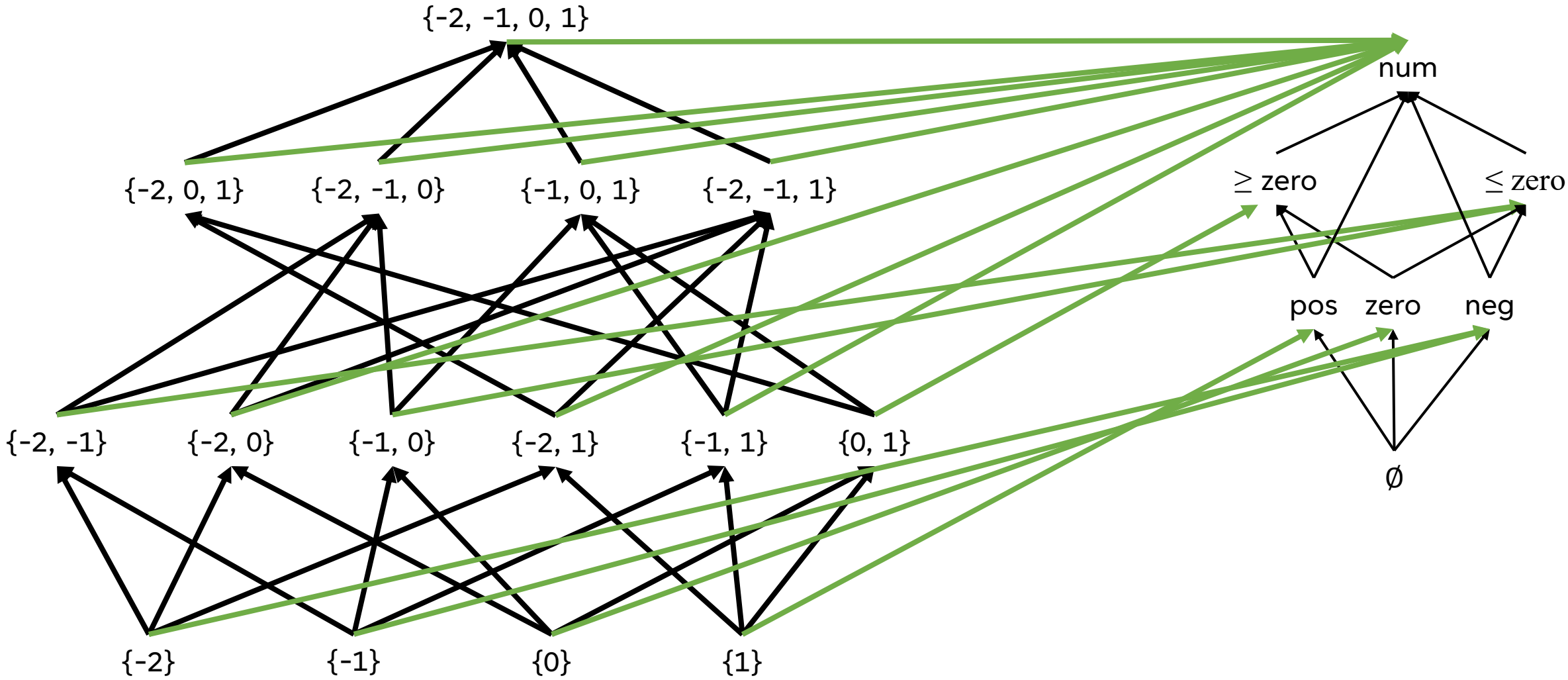
ABSTRACT INTERPRETATION



A LESS-APPROXIMATE LATTICE

ABSTRACT INTERPRETATION

ABSTRACT DOMAIN OF SIGNS



ABSTRACT DOMAINS IN PRACTICE

STATIC ANALYSIS

"Singleton Integer Sets"

- You know the number, or you don't

Intervals

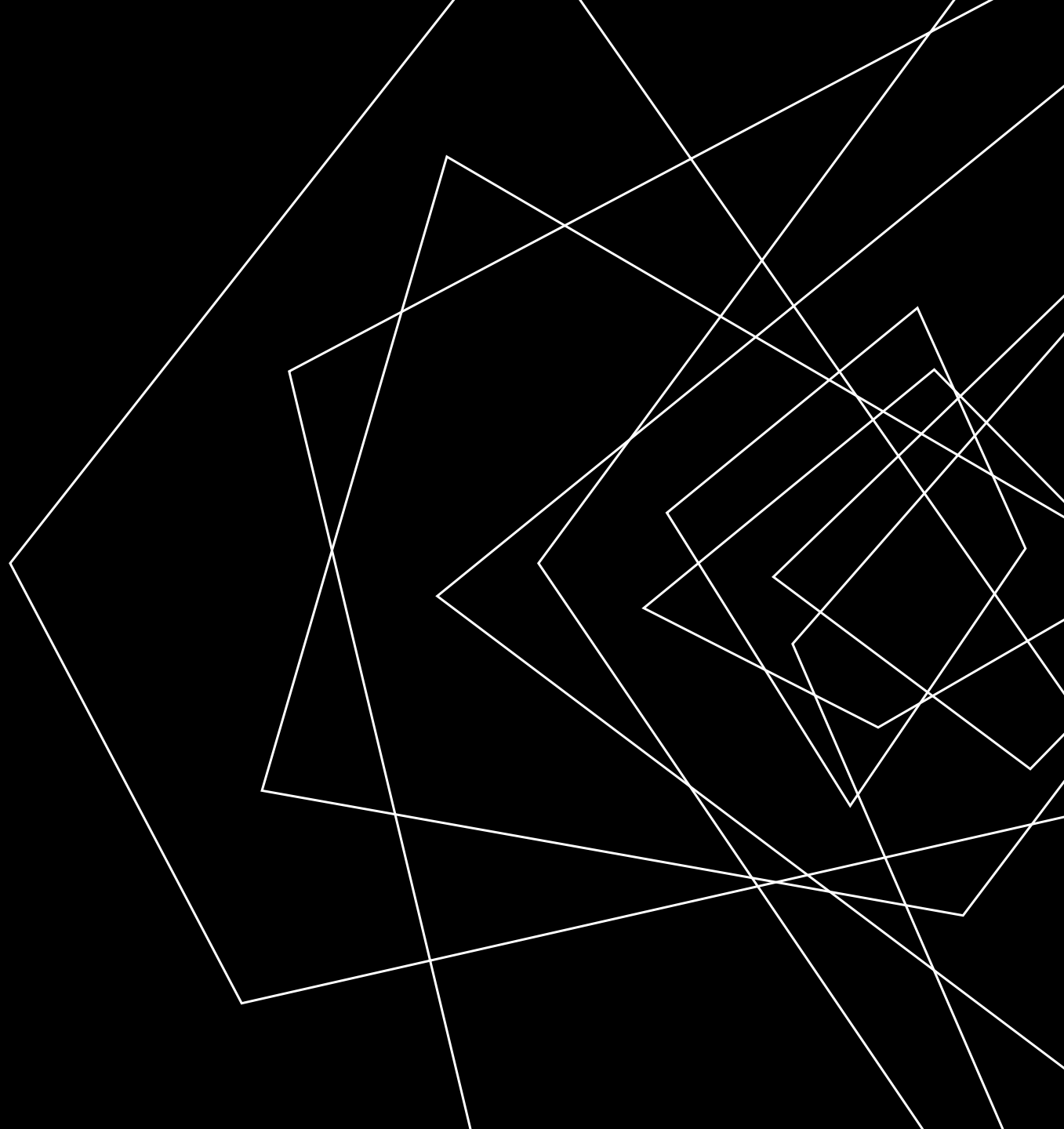
- You know a concrete range

Property Existence

- A property does or does not hold

LECTURE OUTLINE

- Enhancing Dataflow analysis
- Lattices
- Abstract Interpretation



SECTION SUMMARY
STATIC ANALYSIS

STATIC ANALYSIS GIVES US AN IMPORTANT GUARANTEE

- Completeness of bug finding / Soundness of verification

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