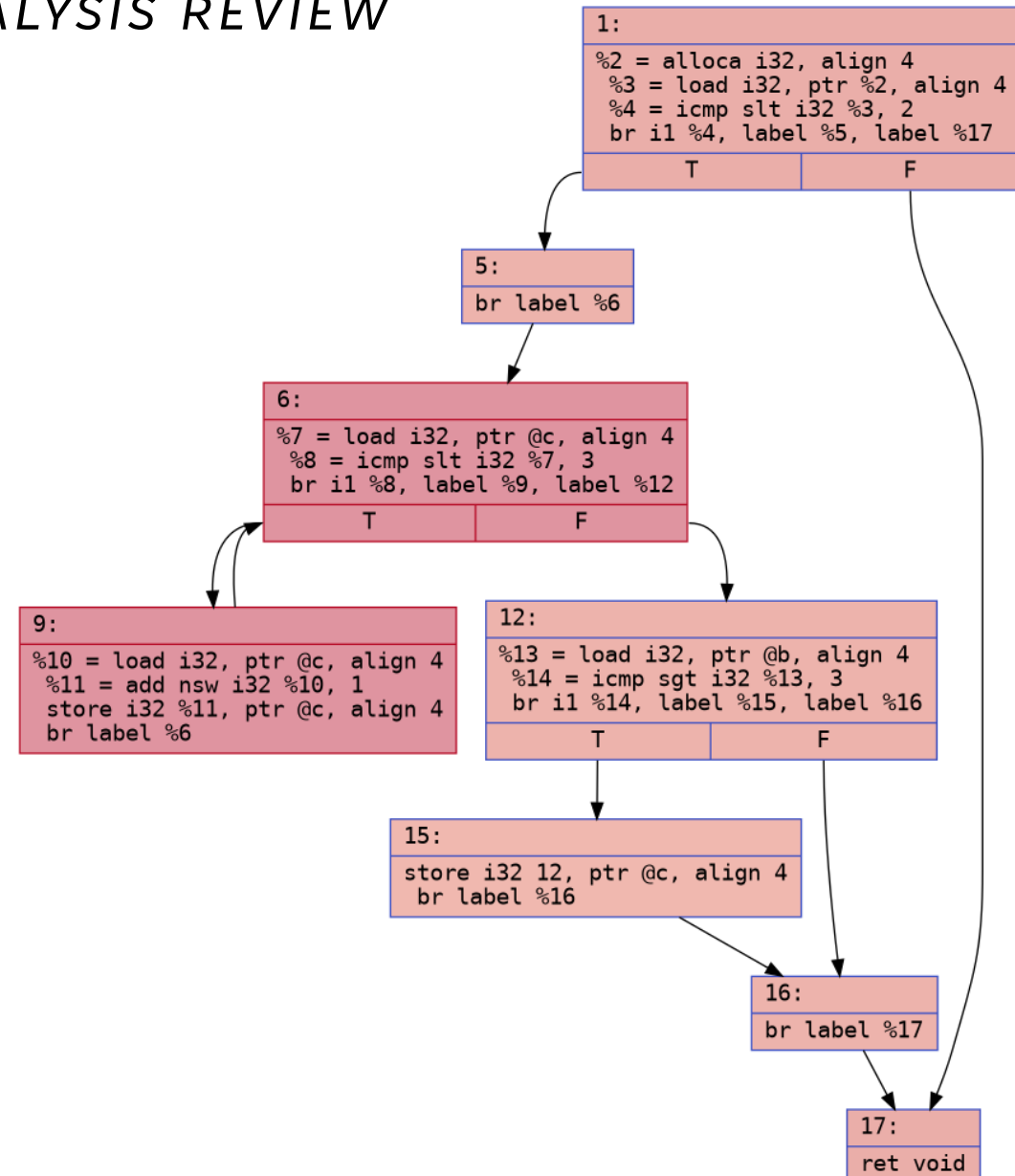
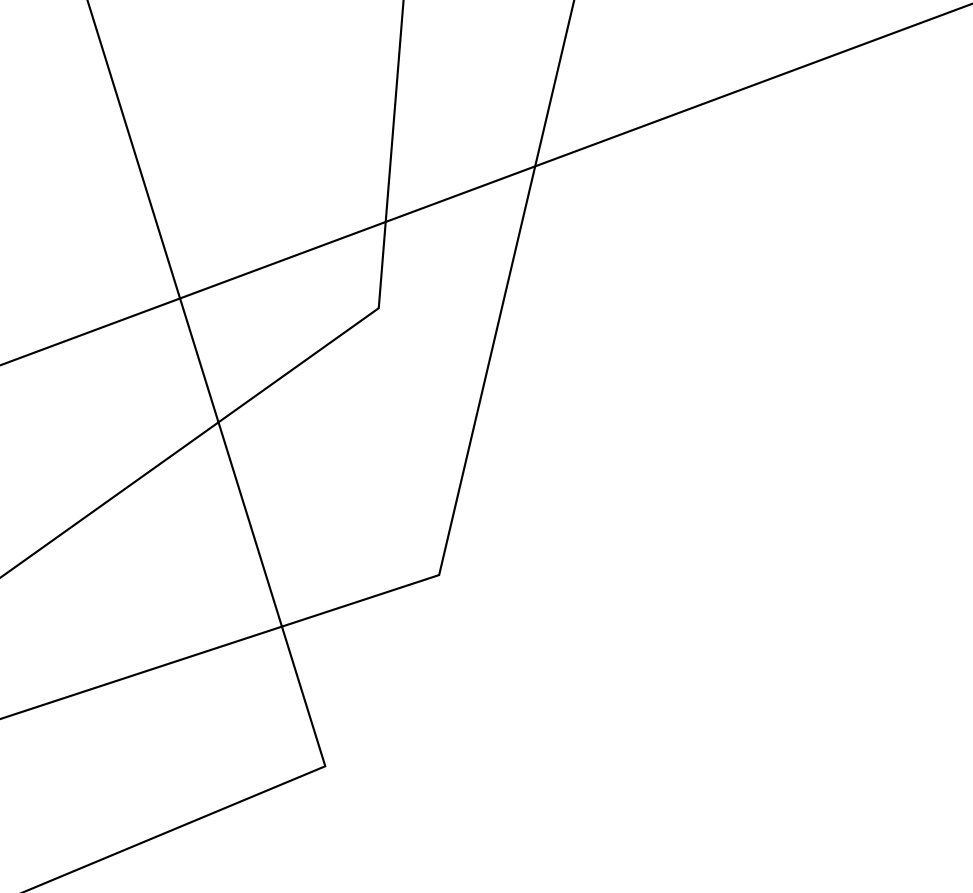


EXERCISE 10

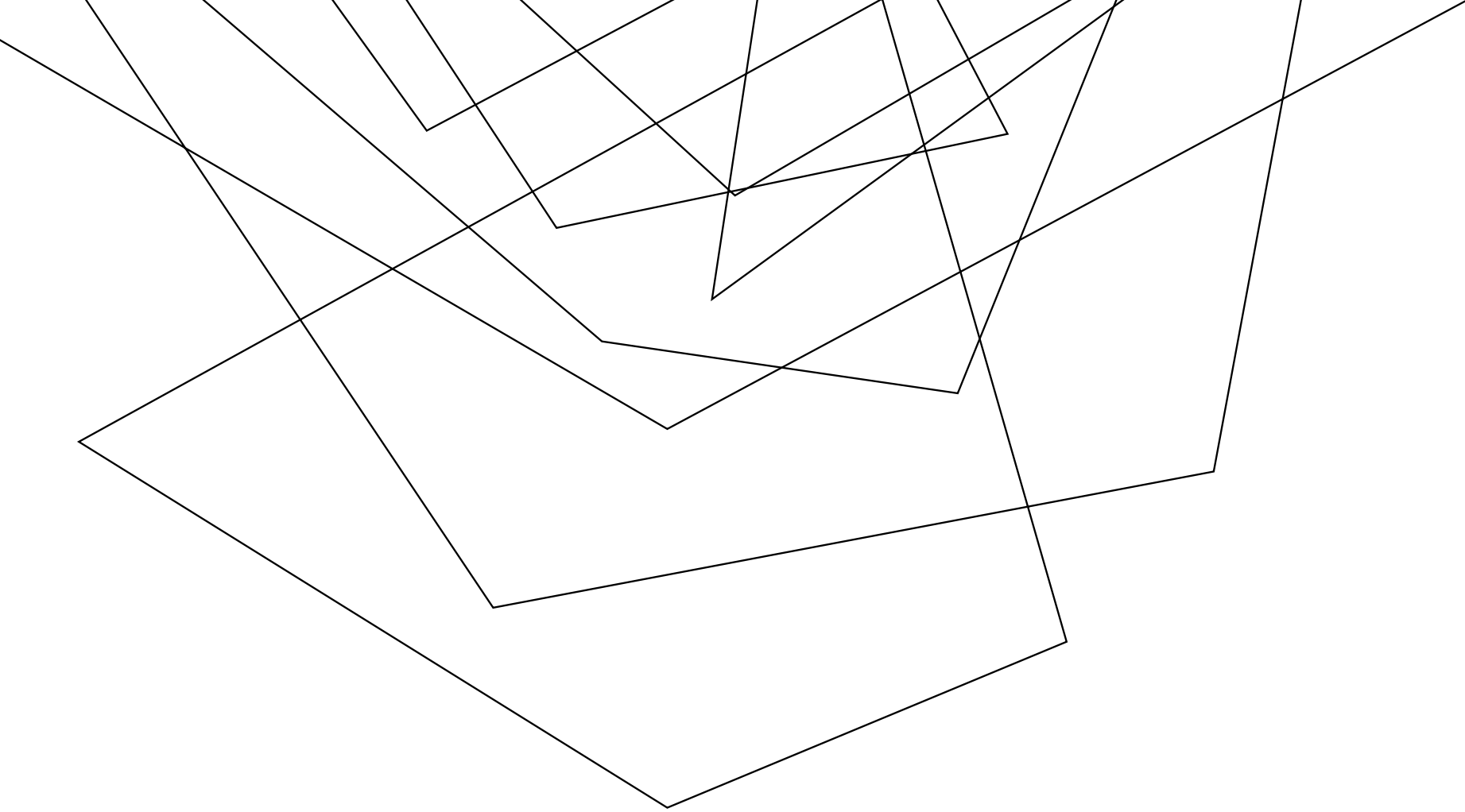
STATIC ANALYSIS REVIEW

- Use path notation to indicate one path (or set of paths) through this CFG:



Abstract geometric lines in the top left corner, consisting of several thin black lines that intersect and form various angles and shapes, creating a modern, architectural feel.

ADMINISTRIVIA AND ANNOUNCEMENTS



DATAFLOW ANALYSIS

EECS 677: Software Security Evaluation

Drew Davidson

CONTINUE TO EXPLORE STATIC ANALYSIS

CLASS PROGRESS

LOOK INTO CONCRETE FORMS OF STATIC ANALYSIS

- Particularly interested in dataflow analysis for now
- Building up the underlying abstractions / techniques to perform such analysis



LAST TIME: STATIC ANALYSIS

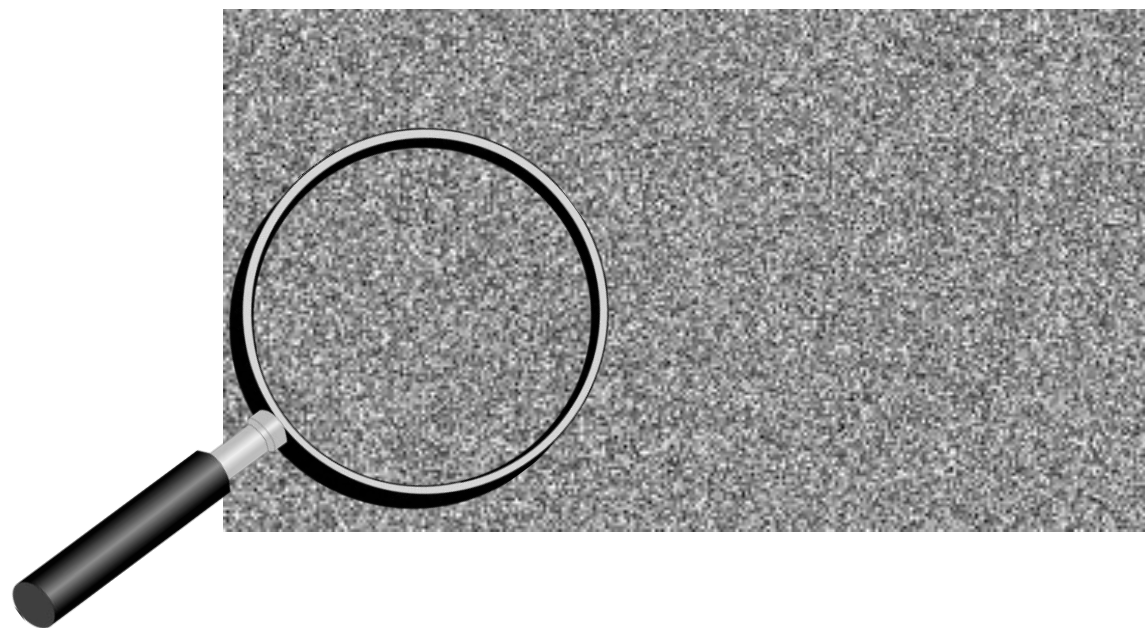
REVIEW: STATIC ANALYSIS

TRUE POWER OF STATIC ANALYSIS

- Unnecessary to supply a given program input
- Summarize the behavior of the program under ANY input
- Capture all possible behaviors of a program

MENTIONED SOME STATIC ANALYSIS TECHNIQUES

- Textual Analysis
- CFG Analysis



LAST TIME: STATIC ANALYSIS

REVIEW: STATIC ANALYSIS

MENTIONED SOME STATIC ANALYSIS TECHNIQUES

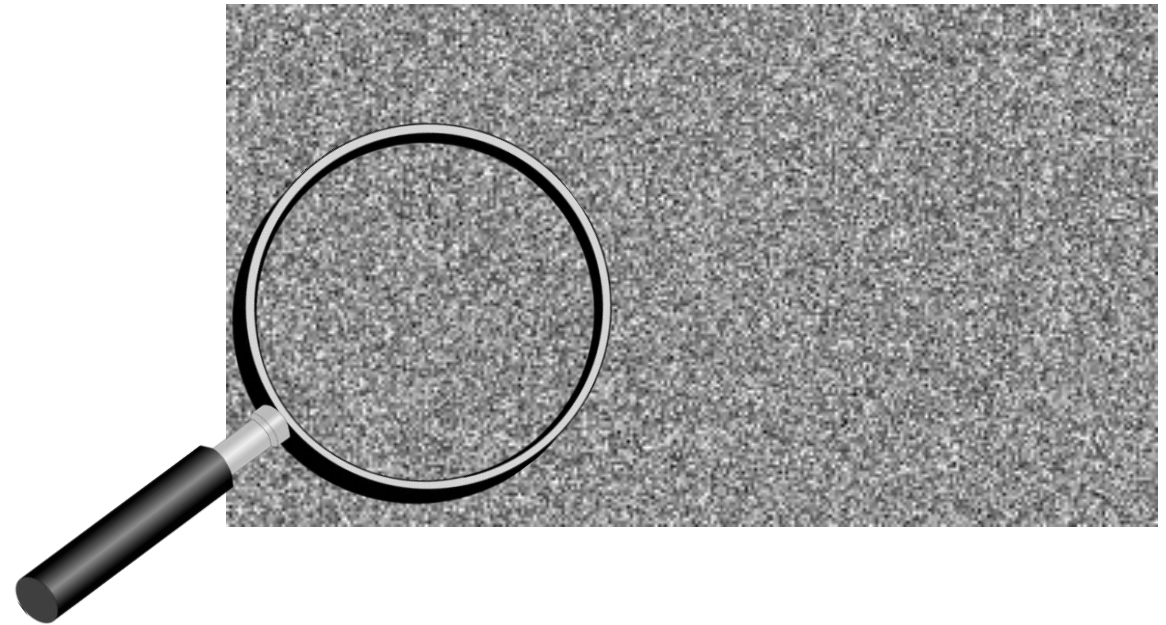
- Textual Analysis
- CFG Analysis

auth.c

```
int main(int argc, char * argv[] ){  
    return (strcmp(argv[1], "secretpw");  
}
```

cmdline

```
$: sudo apt install binutils  
$: gcc auth.c -o auth  
$: strings auth | less
```



LAST TIME: STATIC ANALYSIS

REVIEW: STATIC ANALYSIS

MENTIONED SOME STATIC ANALYSIS TECHNIQUES

- Textual Analysis
- CFG Analysis

auth.c

```
int main(int argc, char * argv[] ){  
    return (strcmp(argv[1], "secretpw");  
}
```

cmdline

```
$: sudo apt install binutils  
$: gcc auth.c -o auth  
$: strings auth | less
```

output

```
/lib64/ld-linux-x86-64.so.2  
__libc_start_main  
__cxa_finalize  
libc.so.6  
GLIBC_2.2.5  
GLIBC_2.34  
_ITM_deregisterTMCloneTable  
__gmon_start__  
_ITM_registerTMCloneTable  
PTE1  
u+UH  
secretpw  
9*3$"  
GCC: (Ubuntu 13.2.0-23ubuntu4) 13.2.0  
Scrt1.o  
...
```

LAST TIME: STATIC ANALYSIS

REVIEW: STATIC ANALYSIS

MENTIONED SOME STATIC ANALYSIS TECHNIQUES

- Textual Analysis
- CFG Analysis

Simplistic Idea: identify isolated issues, refine FPs by CFG reachability

- Build the CFG
- Test if the isolated issue is reachable in the CF

Problems

- A path might be infeasible even if there are edges in the CFG
- Issues might not be isolated

Elaborate Idea: check every path in the program one-by-one

- Build the CFG
- Explore each path through the program

LAST TIME: STATIC ANALYSIS

REVIEW: STATIC ANALYSIS

MENTIONED SOME STATIC ANALYSIS TECHNIQUES

- Textual Analysis
- CFG Analysis

Elaborate Idea: check every path in the program one-by-one

- Build the CFG
- Explore each path through the program

LAST TIME: STATIC ANALYSIS

REVIEW: STATIC ANALYSIS

MENTIONED SOME STATIC ANALYSIS TECHNIQUES

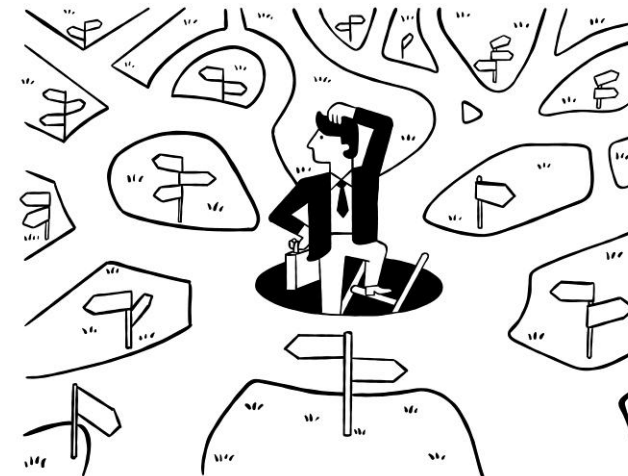
- Textual Analysis
- CFG Analysis

Elaborate Idea: check every path in the program one-by-one

- Build the CFG
- Explore each path through the program

Problems

- Too expensive: many paths, maybe an unbounded set!
- Many program configurations even on a single path!



THIS TIME: ADDRESSING THESE PROBLEMS

REVIEW: STATIC ANALYSIS

MENTIONED SOME STATIC ANALYSIS TECHNIQUES

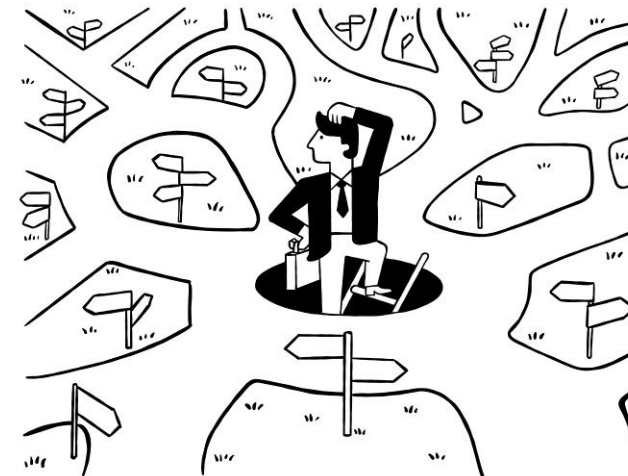
- Textual Analysis
- CFG Analysis

Elaborate Idea: check every path in the program one-by-one

- Build the CFG
- Explore each path through the program

Problems

- Too expensive: many paths, maybe an unbounded set!
- Many program configurations even on a single path!



THIS TIME: ADDRESSING THESE PROBLEMS

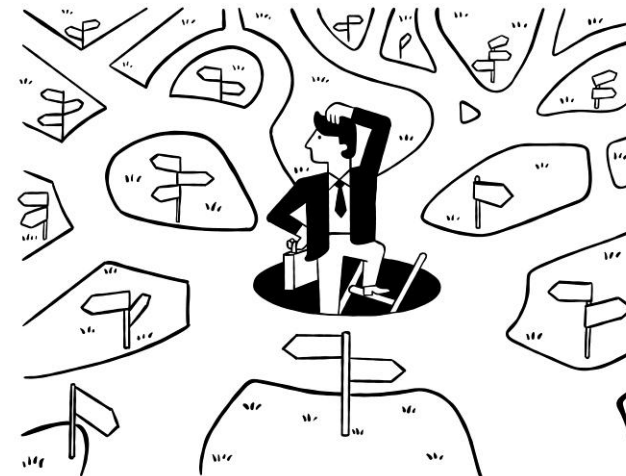
REVIEW: STATIC ANALYSIS

Elaborate Idea: check every path in the program one-by-one

- Build the CFG
- Explore each path through the program

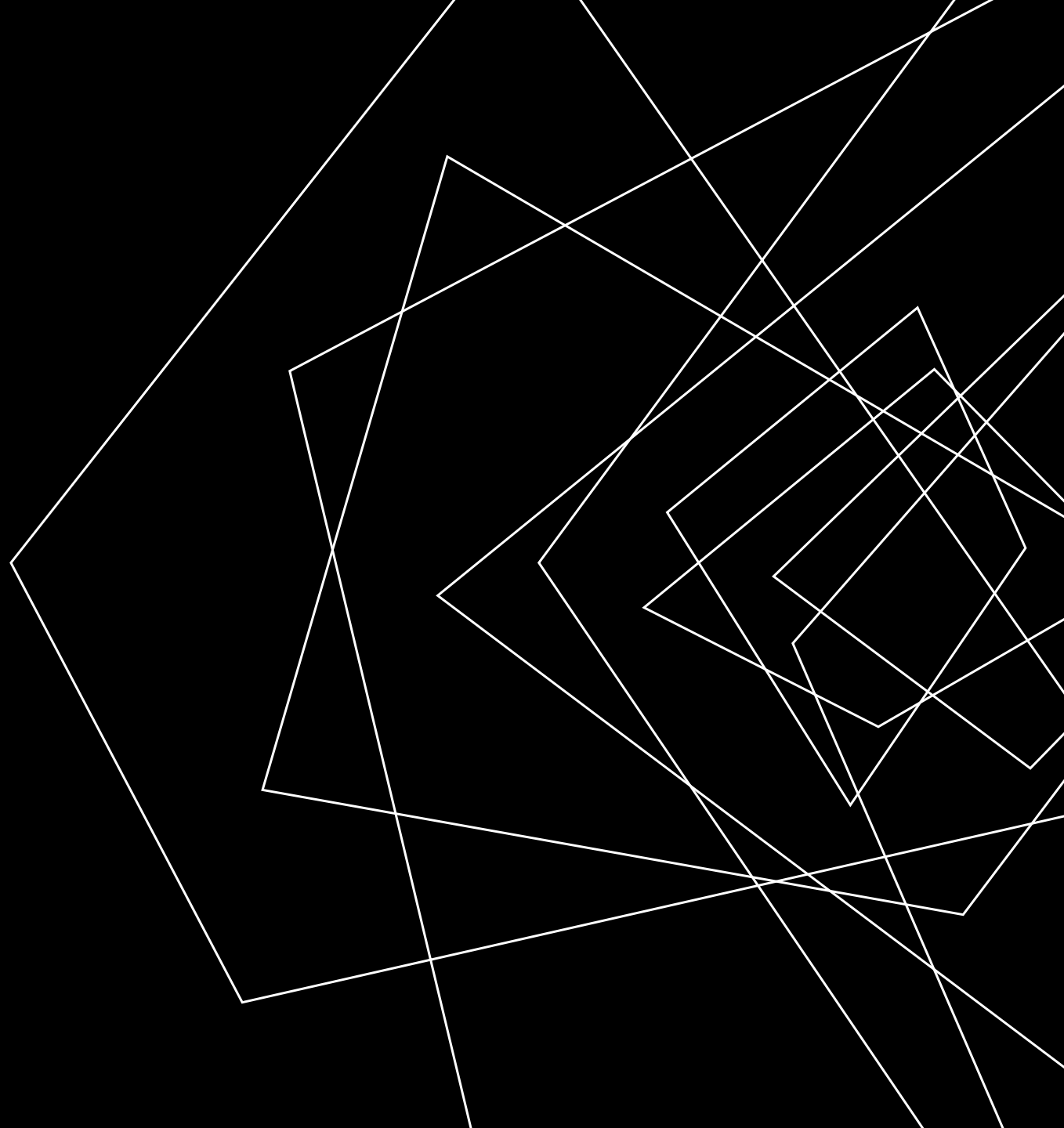
Problems

- Too expensive: many paths, maybe an unbounded set!
- Many program configurations even on a single path!



LECTURE OUTLINE

- Abstracting data
- Abstracting control



THE ART OF ABSTRACTION

CLASS PROGRESS

ENUMERATING ALL PROGRAM
CONFIGURATIONS IS TOO EXPENSIVE

The trick is getting an approximation of the
program's behavior that is both...

- Complete
- Close enough to avoid too many false positives

*A complete approximation of program behavior
=
an **over**-approximation of program behavior*



MODELLING VALUES

ABSTRACTING DATA

```

1 define i1 @foo(i1 %arg) {
2   entry:
3     %x = add i1 0, 1
4     %y = udiv i1 1, %arg
5     ret i1 %y
6 }

```

Model the values a location MIGHT hold

```

1 define void @foo() {
2   %x = add i1 0, 1
3   ret void
4 }

```

```

1 define void @foo() {
2   %x = call i1 () @rand_bool()
3   ret void
4 }

```

```

1 define void @foo(i1 %x) {
2   %z = udiv i1 1, %x
3   ret void
4 }

```

```

1 define void @foo() {
2   %x = add i1 0, 1
3   %z = udiv i1 1, %x
4   ret void
5 }

```

```

1 define void @foo() {
2   %x = call i1 () @rand_bool()
3   %z = udiv i1 1, %x
4   ret void
5 }

```

```

1 define void @foo(i1 %x) {
2   ret void
3 }

```

MODELLING INSTRUCTIONS

ABSTRACTING DATA

```

1 define i1 @foo(i1 %arg) {
2   entry:
3     %x = add i1 0, 1
4     %y = udiv i1 1, %arg
5     ret i1 %y
6 }

```

```

1 define void @foo() {
2   %x = add i1 0, 1
3   ret void
4 }

```

```

1 define void @foo() {
2   %x = call i1 () @rand_bool()
3   ret void
4 }

```

```

1 define void @foo(i1 %x) {
2   ret void
3 }

```

```

1 define void @foo() {
2   %x = add i1 0, 1
3   %z = udiv i1 1, %x
4   ret void
5 }

```

```

1 define void @foo() {
2   %x = call i1 () @rand_bool()
3   %z = udiv i1 1, %x
4   ret void
5 }

```

```

1 define void @foo(i1 %x) {
2   %z = udiv i1 1, %x
3   ret void
4 }

```

MODELLING INSTRUCTIONS

ABSTRACTING DATA

```
1 define void @foo(i2 %x, i2 %y) {  
2   %z = mul i2 %x, %y  
3   ret void  
4 }
```



DATAFLOW ANALYSIS

CLASS PROGRESS

VIEW INSTRUCTIONS AS TRANSFORMERS OF PROGRAM STATE

Several dimensions to tune the state space, we started describing one:

Flow-insensitive analysis



Too squishy

Flow-sensitive analysis



Just right

Path-sensitive analysis



Too hard

FLOW-SENSITIVE ANALYSIS

DATAFLOW ANALYSIS

CONSIDER THE ORDER OF INSTRUCTIONS ALONG ANY
FEASIBLE CONTROL FLOW

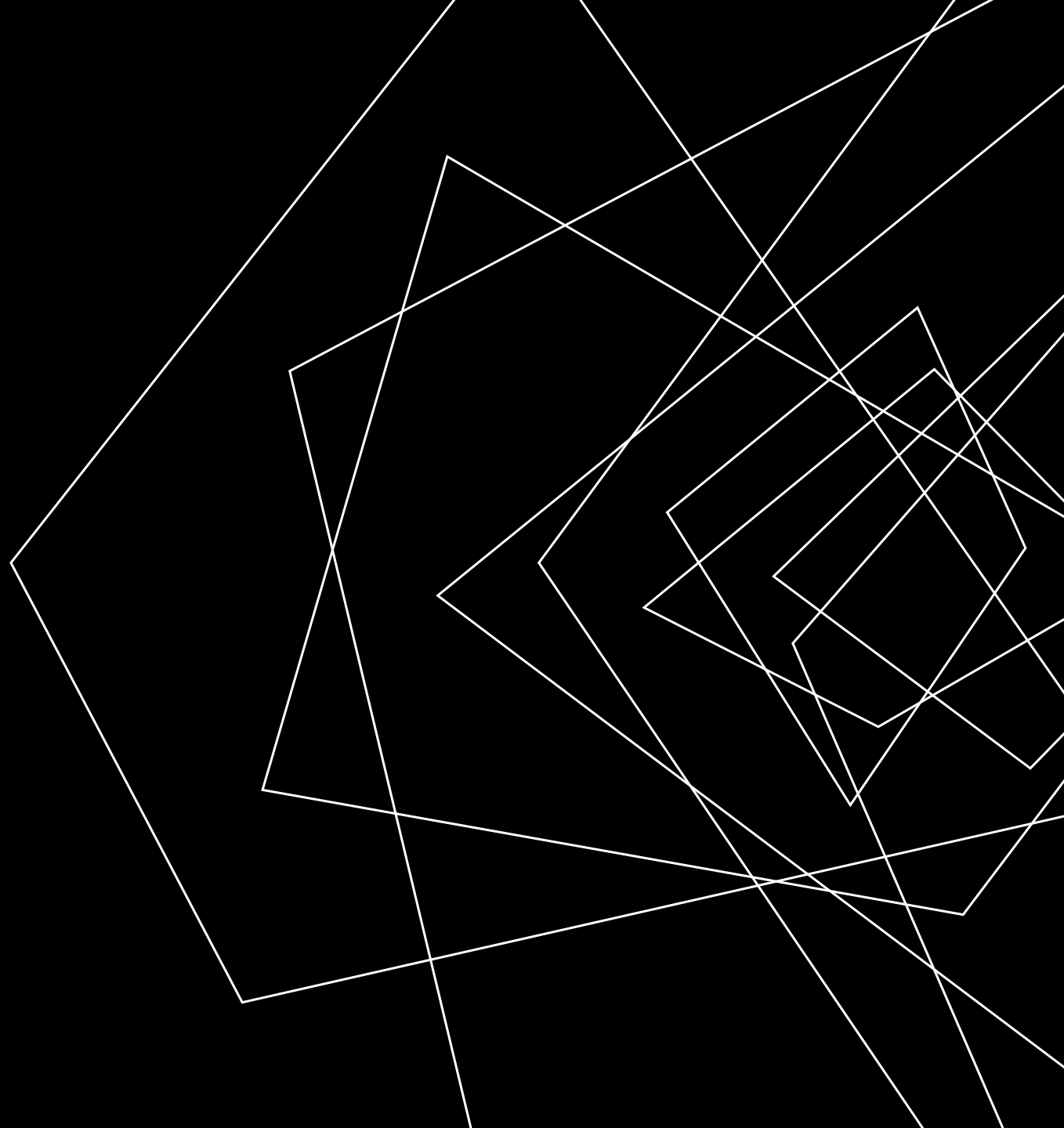
Glom together results of multiple paths

FOR NOW, LET'S START SIMPLE: ANALYSIS WITHIN A BASIC
BLOCK

Known as local analysis

LECTURE OUTLINE

- Intuition: Flow-sensitive analysis
- Local Flow-sensitive analysis
- Global Flow-sensitive analysis



COMPOSING TRANSFER FUNCTIONS

DATAFLOW ANALYSIS

STATEMENTS COMPOSE NATURALLY WITH EACH OTHER
(WITHIN A BASIC BLOCK)

state M

y has the value 1

Stmt₁: x = y ;

Stmt₂: z = x ;

state M'

x has the value 1

y has the value 1

z has the value 1



For now, we'll only think about analysis within a BBL

AN EARLY WIN

DATAFLOW ANALYSIS

EVEN WITH THIS VERY SIMPLE CONCEPT, MIGHT BE ABLE
TO DETECT SOME ISSUES

state M

y has the value 1

Stmt₁: x = y ;

⟨y: 1⟩, ⟨x: 1⟩

Stmt₂: z = 0 ;

⟨y: 1⟩, ⟨x: 1⟩, ⟨z: 0⟩

Stmt₃: p = 1 / z ;

CRASH?!

FORMALIZING TRANSFER FUNCTIONS

DATAFLOW ANALYSIS

IF WE WANT TO BUILD AN AUTOMATED
(LOCAL) DATAFLOW ANALYSIS, WE NEED
PROGRAMMATIC PRECISION

- Some sort of specification of what a statement does
- A statement is a memory state transformer

Memory state M

Stmt₁: k += 1 ;

Memory state M'

Need a semantics!

Representation mapping (large)
set of memory states to each other

Depend somewhat on the analysis

Goals:

- Keep states manageable
- Handle the uncertainty inherent in static analysis

MEMORY AS VALUE SETS

DATAFLOW ANALYSIS

LET EACH MEMORY LOCATION CORRESPOND TO
A SET OF VALUES IT MIGHT CONTAIN

- Define (informally) transfer functions as mapping elements of M to elements of M'

We're still kinda-dodging the larger semantic questions here. for now, let's just say we're using a big ol' if statement to define an operator

Memory state M	$\langle k: \{1\} \rangle$	$\langle k: \{3,4\} \rangle$
Stmt ₁ : $k += 1$;		
Memory state M'	$\langle k: \{2\} \rangle$	$\langle k: \{4,5\} \rangle$

COMPOSING VALUE SETS

DATAFLOW ANALYSIS

(example: assume a 1-bit data size)

Stmt₀: $y = \text{randomBit}()$

$\langle y: \{0, 1\} \rangle$

Stmt₁: $x = y ;$

$\langle y: \{0, 1\}, x: \{0, 1\} \rangle$

Stmt₂: $z = x ;$

$\langle y: \{0, 1\}, x: \{0, 1\}, z: \{0, 1\} \rangle$

Stmt₃: $p = 1 / z ;$

CRASH?!

MODELLING UNCERTAINTY

DATAFLOW ANALYSIS

(example: assume an i1 (1 bit) data size)

WE CAN NOW HANDLE OPAQUE DATA SOMEWHAT CLEANLY

$\langle z: \{0\}, y: \{0, 1\}, z: \{0\} \rangle$

Stmt₁: x = y ;

$\langle x: \{0, 1\}, y: \{0, 1\}, z: \{0\} \rangle$

Stmt₂: z = USER_INPUT ;

$\langle p: \{0, \text{ERR}\}, x: \{0, 1\}, y: \{0, 1\}, z: \{0, 1\} \rangle$

Stmt₃: p = 1 / z ;

$\langle x: \{0, 1\}, y: \{0, 1\}, z: \{0, 1\} \rangle$

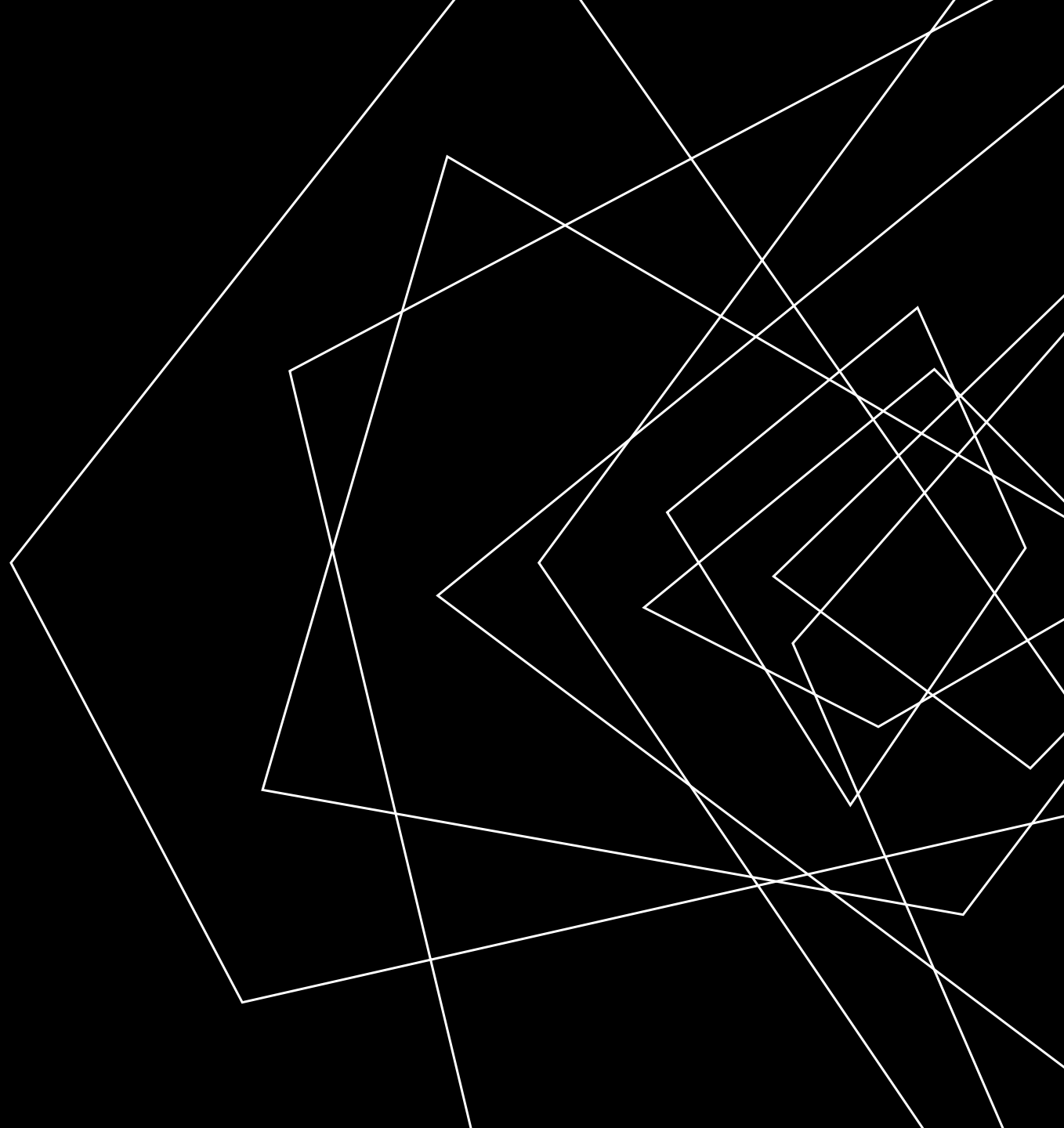
Stmt₁: x = y ;

Stmt₂: z = global ;

Stmt₃: p = 1 / z ;

LECTURE OUTLINE

- (Local) Dataflow analysis
- Global dataflow analysis



COMPOSING BLOCKS

GLOBAL DATAFLOW ANALYSIS

VALUE-SET MODEL OF MEMORY IMPLIES A METHOD TO EXTEND
BEYOND LOCAL ANALYSIS

```
void f(bool a){  
    bool b = a;  
    bool c = a;  
    if (a){  
        b = true;  
        c = true  
    } else {  
        b = true;  
        c = false;  
    }  
    return b;  
}
```



Go Global

COMPOSING BLOCKS

GLOBAL DATAFLOW ANALYSIS

VALUE-SET MODEL OF MEMORY IMPLIES A METHOD TO EXTEND
BEYOND LOCAL ANALYSIS

```
1 define i1 @f(i1 %a) {  
2 B1:  
3   %bPtr = alloca i1  
4   %cPtr = alloca i1  
5   store i1 %a, i1* %bPtr  
6   store i1 %a, i1* %cPtr  
7   br i1 %a, label %B2, label %B3  
8 B2:  
9   store i1 1, i1* %bPtr  
10  store i1 1, i1* %cPtr  
11  br label %B4  
12 B3:  
13  store i1 1, i1* %bPtr  
14  store i1 0, i1* %cPtr  
15  br label %B4  
16 B4:  
17  %bRes = load i1, i1* %bPtr  
18  ret i1 %bRes  
19 }
```

MAY-BE VS MUST-BE ANALYSIS

GLOBAL DATAFLOW ANALYSIS

HOW WE JOIN VALUES IS BASED ON THE GOAL OF OUR ANALYSIS

```
void f(bool a){  
    bool b = a;  
    bool c = a;  
    if (a){  
        b = true;  
        c = true  
    } else {  
        b = true;  
        c = false;  
    }  
    return b;  
}
```

CHAOTIC ITERATION

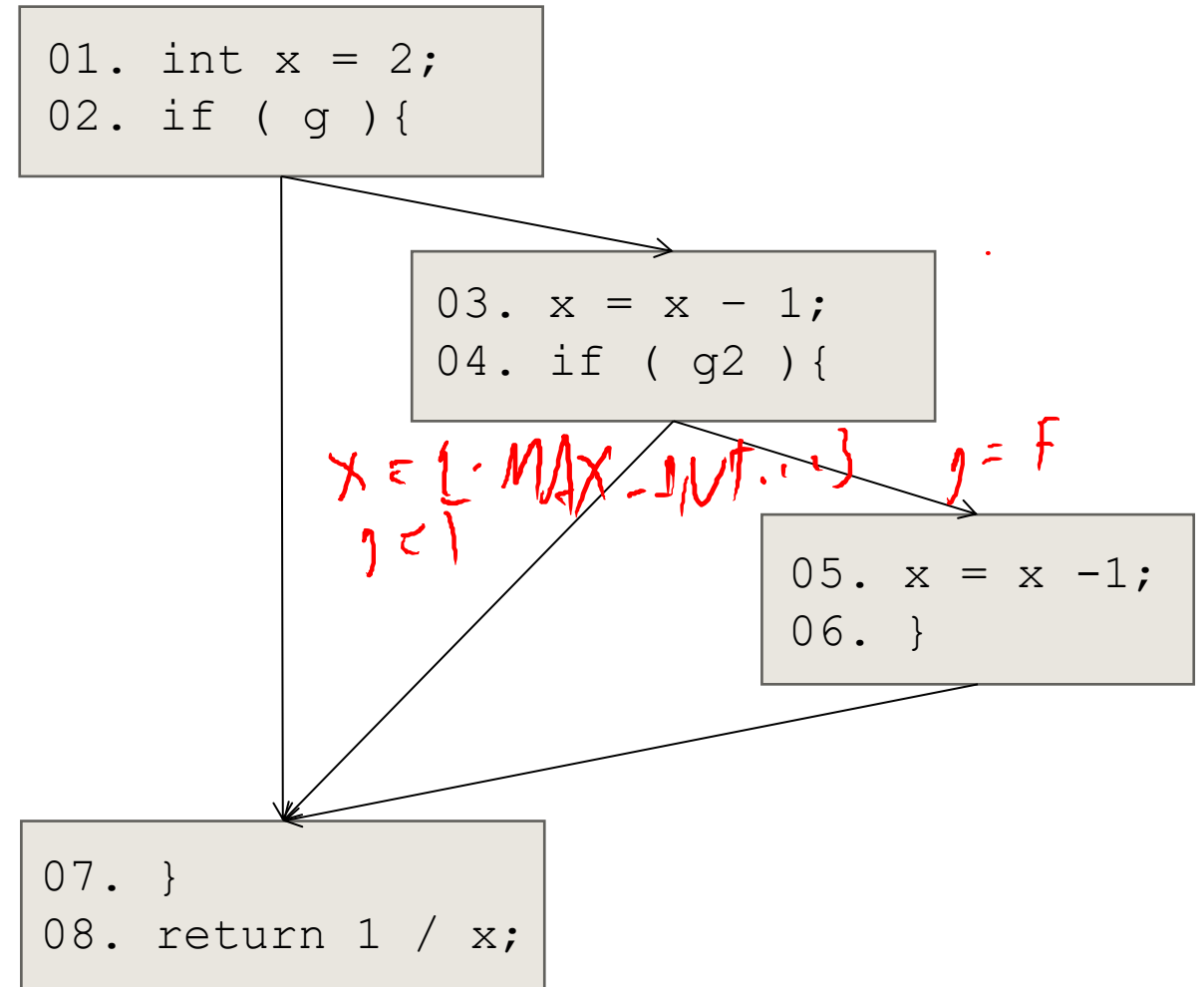
GLOBAL DATAFLOW ANALYSIS

IN WHAT ORDER DO WE PROCESS BLOCKS?

```

01. int x = 2;
02. if ( g ){
03.     x = x - 1;
04.     if ( g2 ){
05.         x = x - 1;
06.     }
07. }
08. return 1 / x;

```



TROUBLE ON THE HORIZON

GLOBAL DATAFLOW ANALYSIS



LOOPS ARE TOUGH TO HANDLE!

GLOBAL DATAFLOW ANALYSIS

ISSUES WITH LOOPS

- Generate lots of paths
- Cyclic data dependency



Oh, brother! You may have some loops

LECTURE END!

- Local Dataflow analysis
- Global Dataflow analysis

