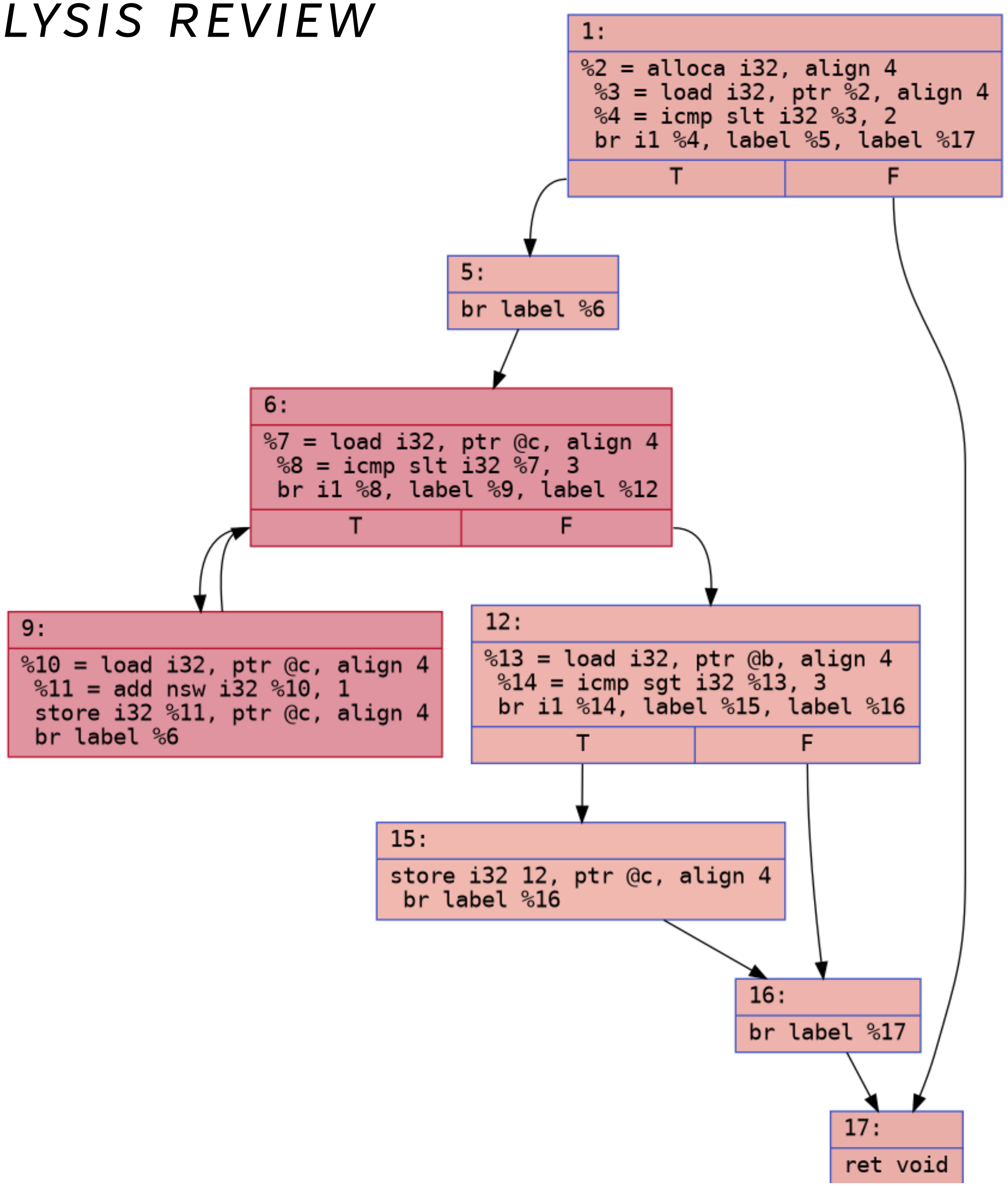
EXERCISE #8

STATIC ANALYSIS REVIEW

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



```
1:
%2 = alloca i32, align 4
 %3 = load i32, ptr %2, align 4
 %4 = icmp slt i32 %3, 2
 br i1 %4, label %5, label %17
```

```
5:
br label %6
```

```
6:
%7 = load i32, ptr @c, align 4
 %8 = icmp slt i32 %7, 3
 br i1 %8, label %9, label %12
```

```
9:
%10 = load i32, ptr @c, align 4
 %11 = add nsw i32 %10, 1
 store i32 %11, ptr @c, align 4
 br label %6
```

```
12:
%13 = load i32, ptr @b, align 4
 %14 = icmp sgt i32 %13, 3
 br i1 %14, label %15, label %16
```

```
15:
store i32 12, ptr @c, align 4
 br label %16
```

```
16:
br label %17
```

```
17:
ret void
```

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

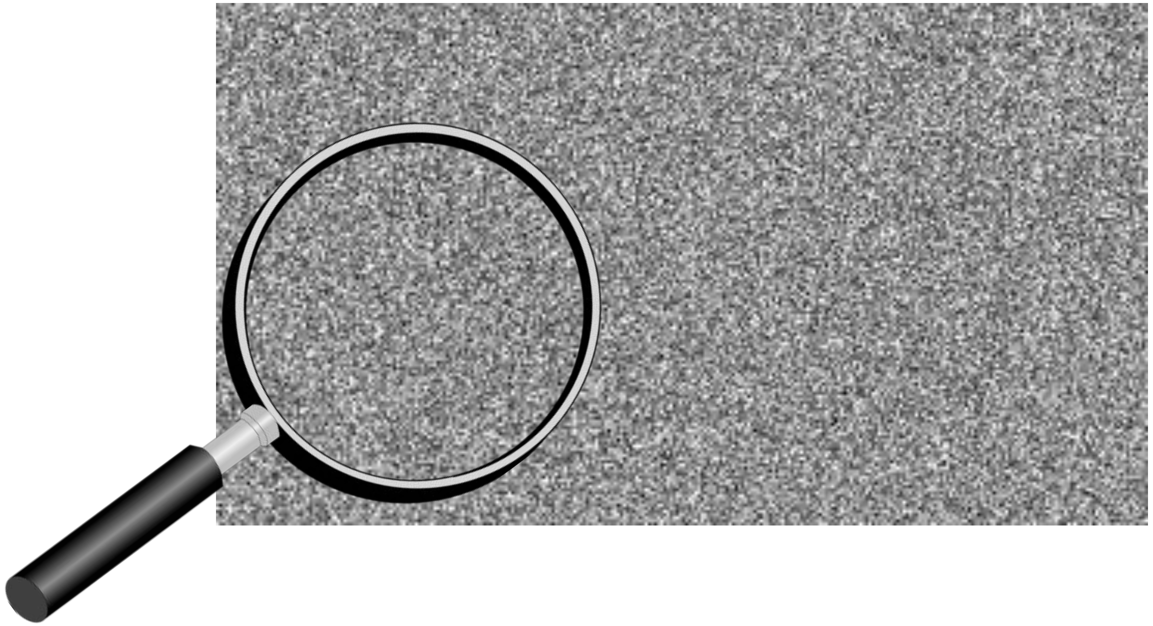
REVIEW: STATIC ANALYSIS

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

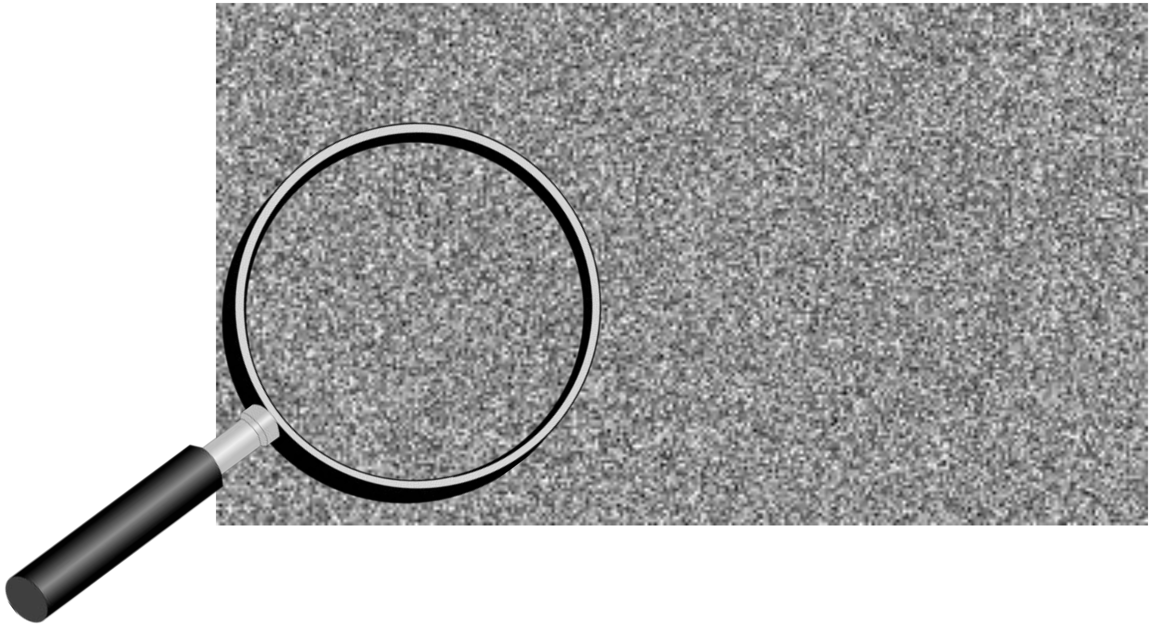


LAST TIME: STATIC ANALYSIS

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

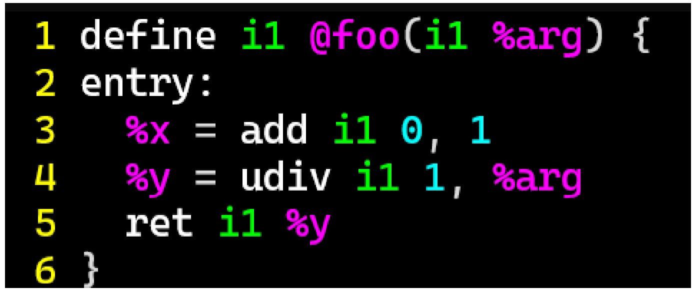
A complete approximation of program behavior

=

*an **over**-approximation of program behavior*

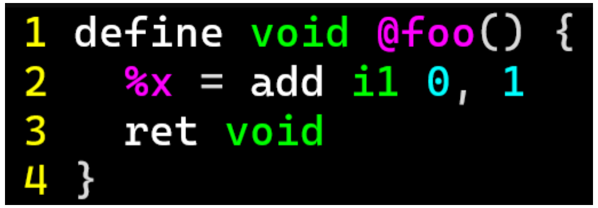
MODELLING VALUES

ABSTRACTING DATA

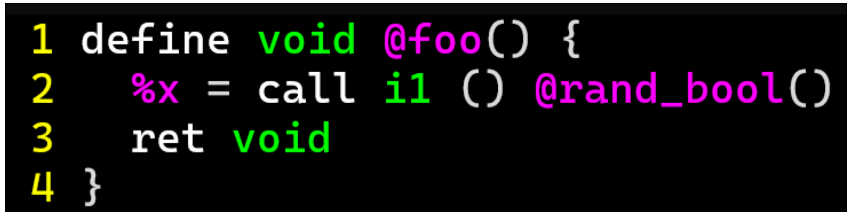


```
define i1 @foo(i1 %arg) {
entry:
  %x = add i1 0, 1
  %y = udiv i1 1, %arg
  ret i1 %y
}
```

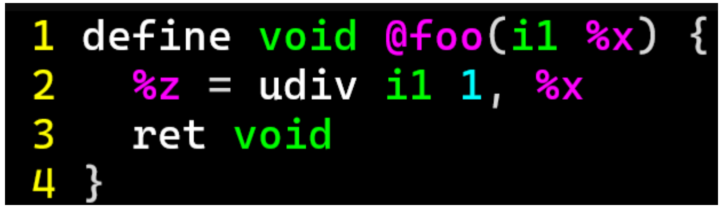
Model the values a location MIGHT hold



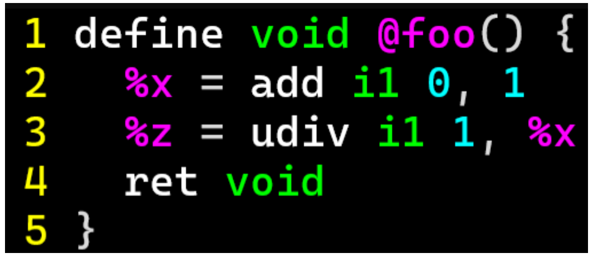
```
define void @foo() {
  %x = add i1 0, 1
  ret void
}
```



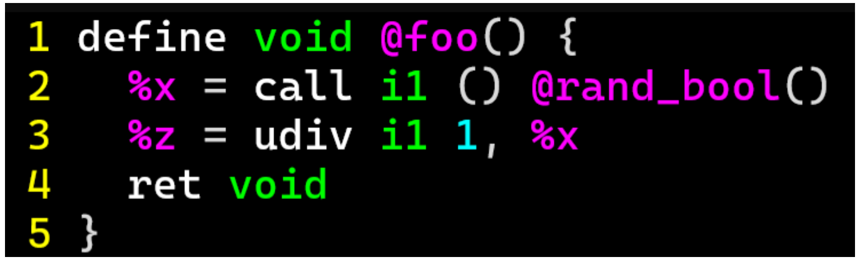
```
define void @foo() {
  %x = call i1 () @rand_bool()
  ret void
}
```



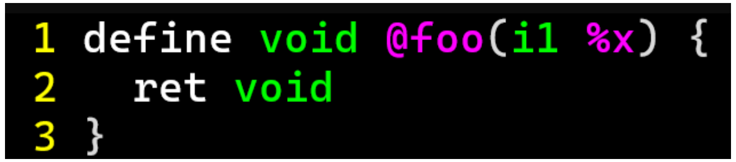
```
define void @foo(i1 %x) {
  %z = udiv i1 1, %x
  ret void
}
```



```
define void @foo() {
  %x = add i1 0, 1
  %z = udiv i1 1, %x
  ret void
}
```



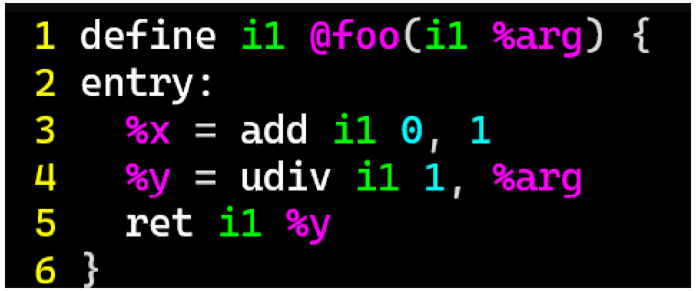
```
define void @foo() {
  %x = call i1 () @rand_bool()
  %z = udiv i1 1, %x
  ret void
}
```



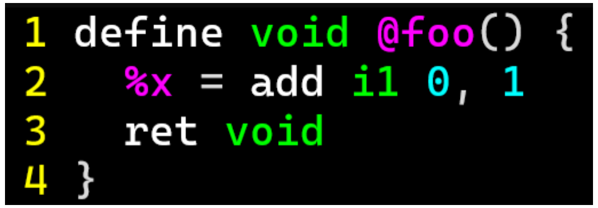
```
define void @foo(i1 %x) {
  ret void
}
```

MODELLING INSTRUCTIONS

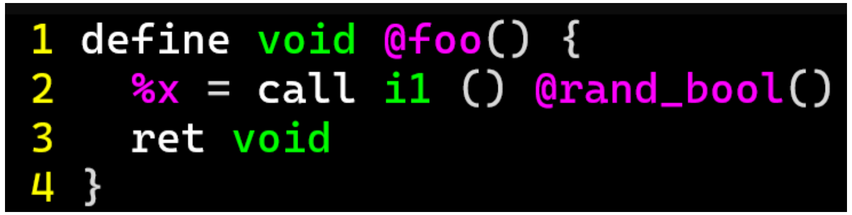
ABSTRACTING DATA



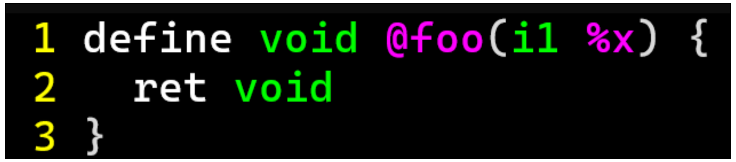
```
define i1 @foo(i1 %arg) {
entry:
  %x = add i1 0, 1
  %y = udiv i1 1, %arg
  ret i1 %y
}
```



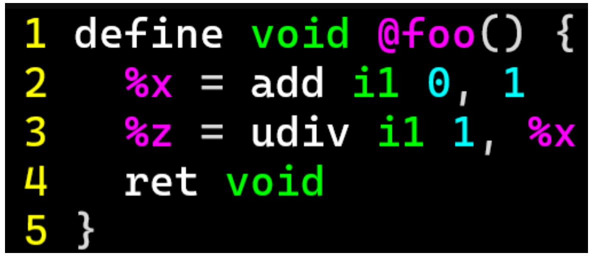
```
define void @foo() {
  %x = add i1 0, 1
  ret void
}
```



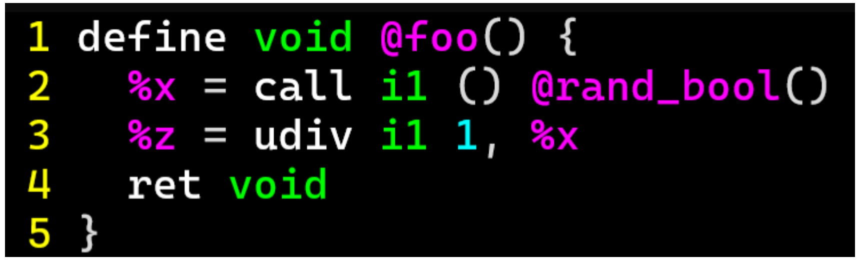
```
define void @foo() {
  %x = call i1 () @rand_bool()
  ret void
}
```



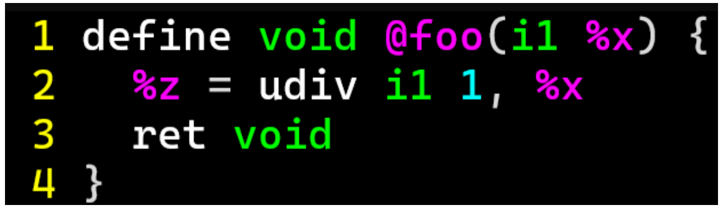
```
define void @foo(i1 %x) {
  ret void
}
```



```
define void @foo() {
  %x = add i1 0, 1
  %z = udiv i1 1, %x
  ret void
}
```



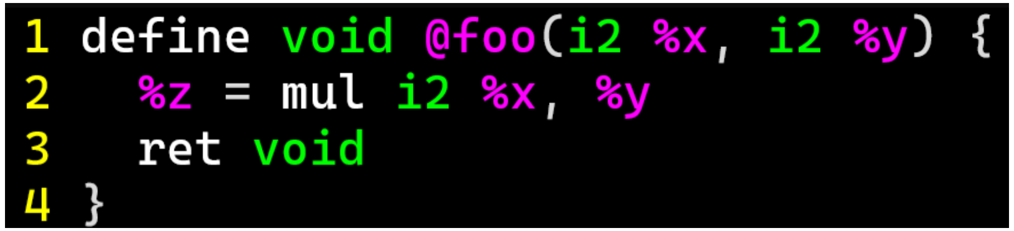
```
define void @foo() {
  %x = call i1 () @rand_bool()
  %z = udiv i1 1, %x
  ret void
}
```



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

MODELLING INSTRUCTIONS

ABSTRACTING DATA



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

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

Stmt$_1$: x = y ;

Stmt$_2$: 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

Stmt$_1$: x = y ;

$\langle \boldsymbol{y}: \mathbf{1} \rangle, \langle \boldsymbol{x}: \mathbf{1} \rangle$

Stmt$_2$: z = 0 ;

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

Stmt$_3$: p = 1 / z ;

$\boldsymbol{CRASH}?!$

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

Stmt_1: 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 lets just say we're using a big ol' if statement to define an operator

Memory state M $\langle k\colon \{1\}\rangle$ $\langle k\colon \{3,4\}\rangle$

Stmt$_1$: k += 1 ;

Memory state M' $\langle k\colon \{2\}\rangle$ $\langle k\colon \{4,5\}\rangle$

COMPOSING VALUE SETS

DATAFLOW ANALYSIS

(example: assume a 1-bit data size)

Stmt$_0$: y = randomBit()

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

Stmt$_1$: x = y ;

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

Stmt$_2$: z = x ;

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

Stmt$_3$: p = 1 / z ;

$\boldsymbol{CRASH}?!$

MODELLING UNCERTAINTY

DATAFLOW ANALYSIS

WE CAN NOW HANDLE OPAQUE DATA SOMEWHAT CLEANLY

z: {0}

y: {0,1}

Stmt$_1$: x = y ;

z: {0} x: {0,1}, y: {0,1}

Stmt$_2$: z = USER_INPUT ;

z: {0,1} x: {0,1}, y: {0,1}

Stmt$_3$: p = 1 / z ;

CRASH?!

Stmt$_1$: x = y ;

Stmt$_2$: z = global ;

Stmt$_3$: 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

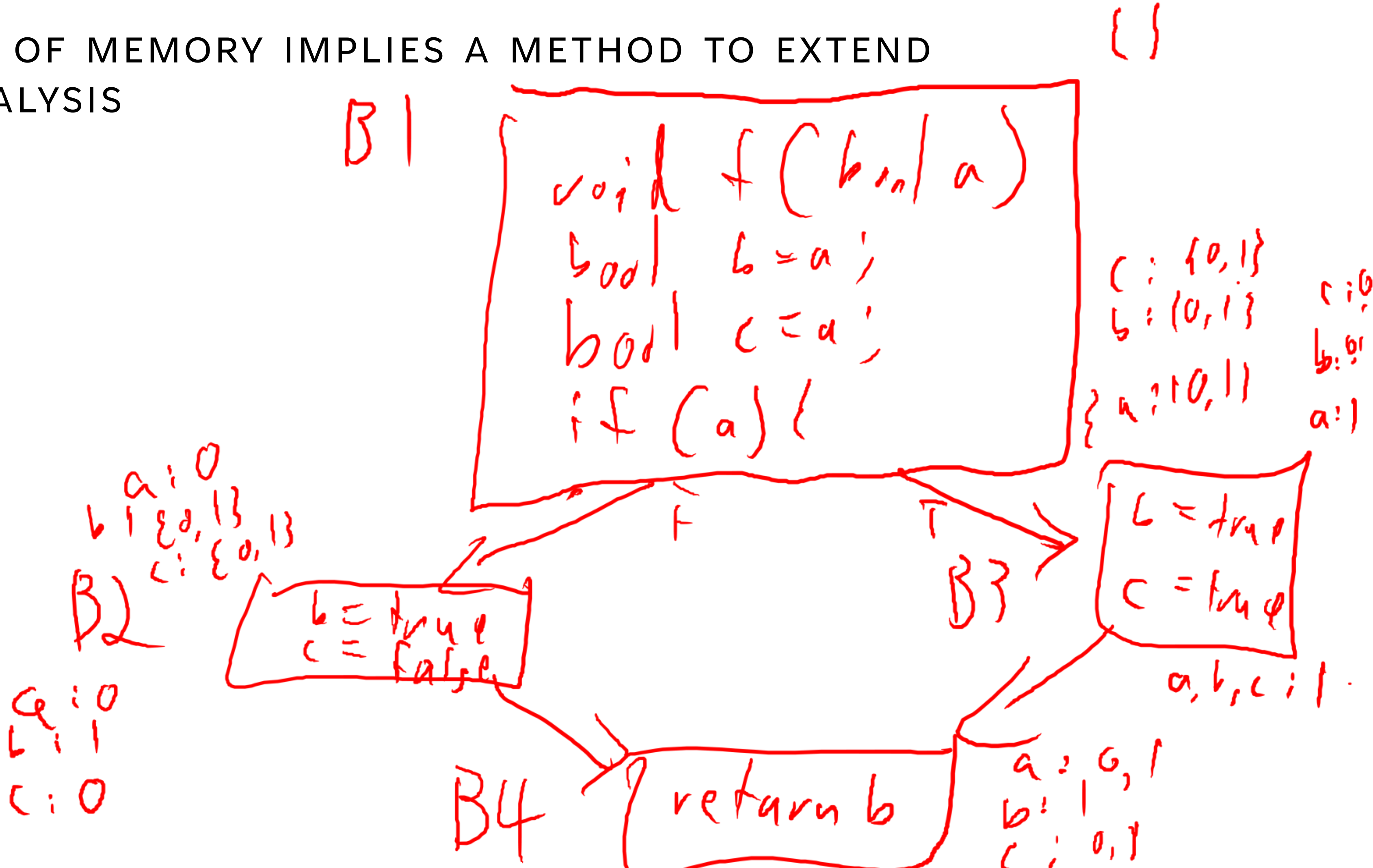
```
void f(bool a){
  bool b = a;
  bool c = a;
  if (a){
    b = true;
    c = true
  } else {
    b = true;
    c = false;
  }
  return b;
}
```

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;
}
```



MAY-BE VS MUST-BE ANALYSIS

GLOBAL DATAFLOW ANALYSIS

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

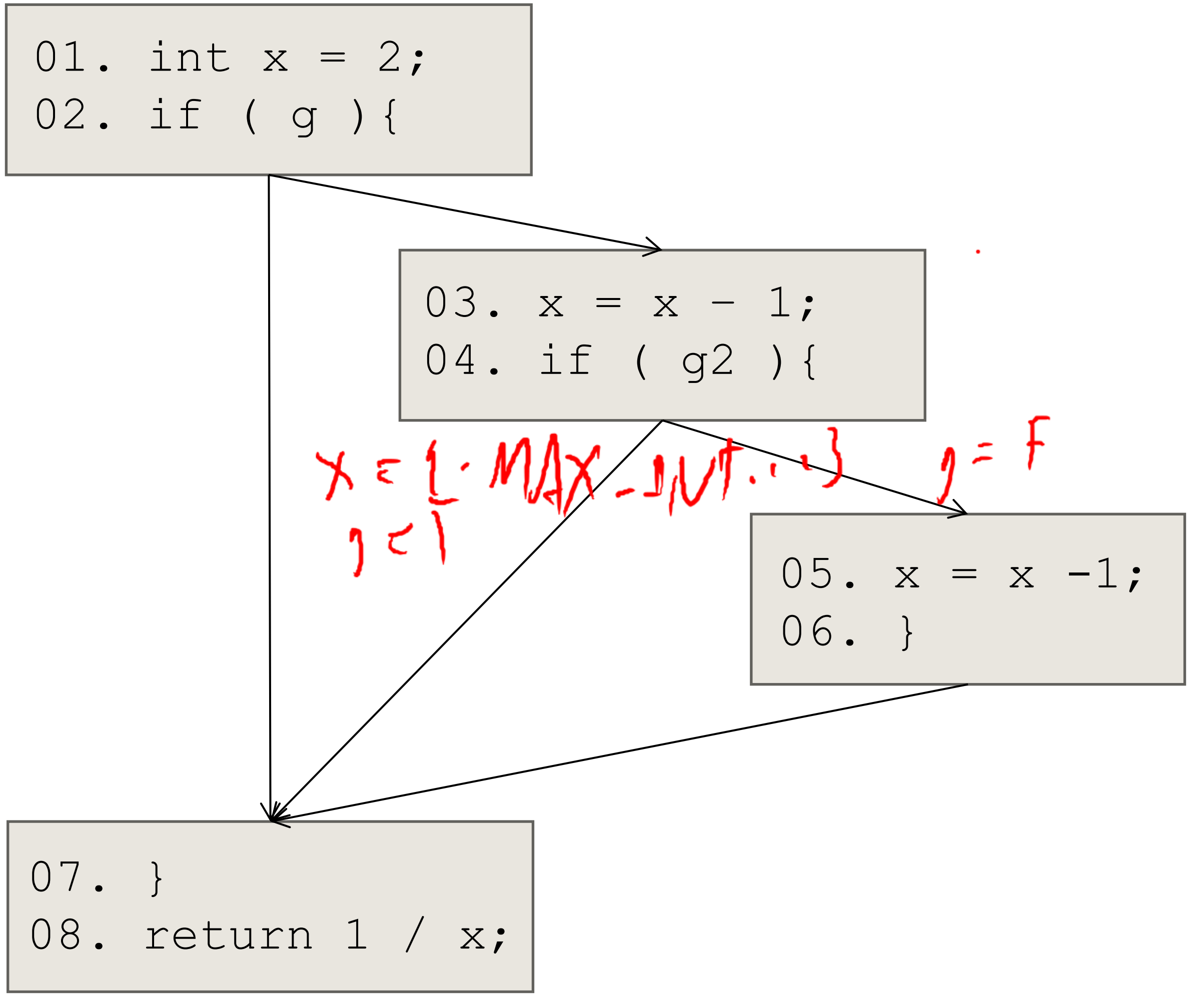
```
void f(bool a){
  bool b = a;
  bool c = a;
  if (a){
    b = true;
    c = true
  } else {
    b = true;
    c = false;
  }
  return b;
}
```

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;
```



```
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

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