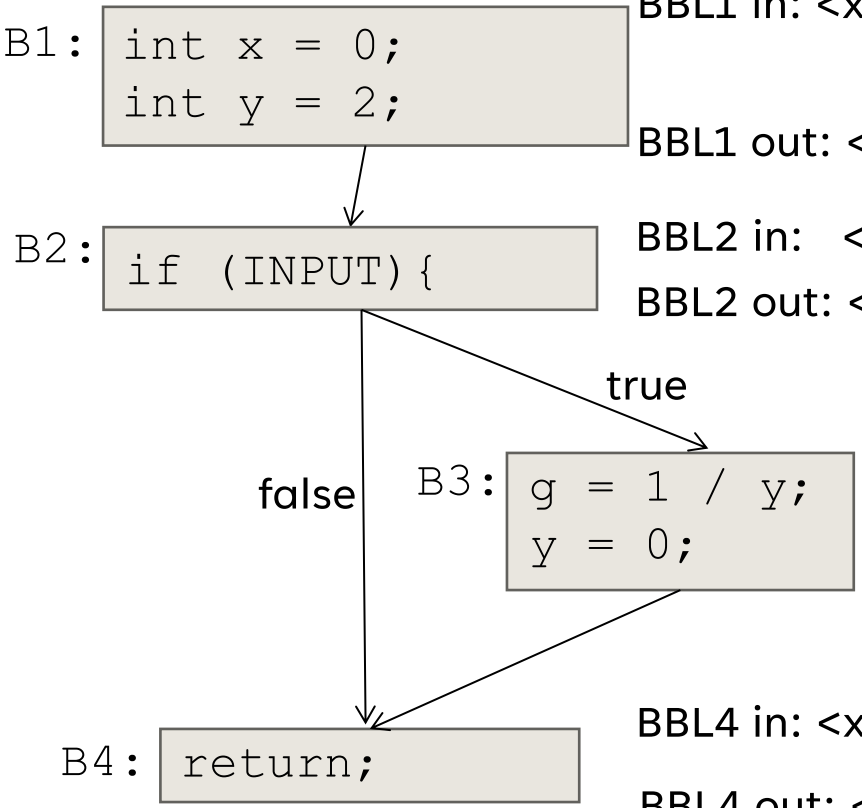
EXERCISE #10

DATAFLOW REVIEW

Perform a value-set dataflow analysis on the following CFG, starting at B1, then B2 then BBL 3, then BBL 4. Give the value sets at the top of each block after 1 round of analysis



```
int x = 0;
int y = 2;
```

```
if (INPUT){
```

```
g = 1 / y;
y = 0;
```

```
return;
```

BBL1 in: <x: { } , y: { }, g: { }>

BBL1 out: <x: { } , y: { }, g: { }>

BBL2 in: <x: { } , y: { }, g: { }>

BBL2 out: <x: { } , y: { }, g: { }>

BBL3 in: <x: { } , y: { }, g: { }>

BBL3 out: <x: { } , y: { }, g: { }>

BBL4 in: <x: { } , y: { }, g: { }>

BBL4 out: <x: { } , y: { }, g: { }>

WARMUPS –

Full point in class

Correctness after class

ADMINISTRIVIA AND ANNOUNCEMENTS

DATAFLOW FIXPOINTS

EECS 677: Software Security Evaluation

Drew Davidson

CLASS PROGRESS

EXPLORING A FORM OF STATIC ANALYSIS THAT SUMMARIZES HOW CONTROL AND DATA FLOWS ACROSS A PROGRAM

- MANIFEST A COMPLETE ANALYSIS BY DENOTING SETS OF ALL VALUES MEMORY MIGHT CONTAIN (**NB – THIS WILL END UP BEING CUMBERSOME!**)

LAST TIME: VALUE SET ANALYSIS

REVIEW: DATAFLOW

CONSERVATIVELY TRACK THE POSSIBLE SET OF VALUES TAKEN

```
uint4 x = randInt();
uint4 y = x % 2;
return x / y;
```

← x, ~~y~~ : {0-15}

← y: {0,1}, x: {0-15}

This approach is a complete over-approximation

LAST TIME: FLOW SENSITIVITY

REVIEW: DATAFLOW

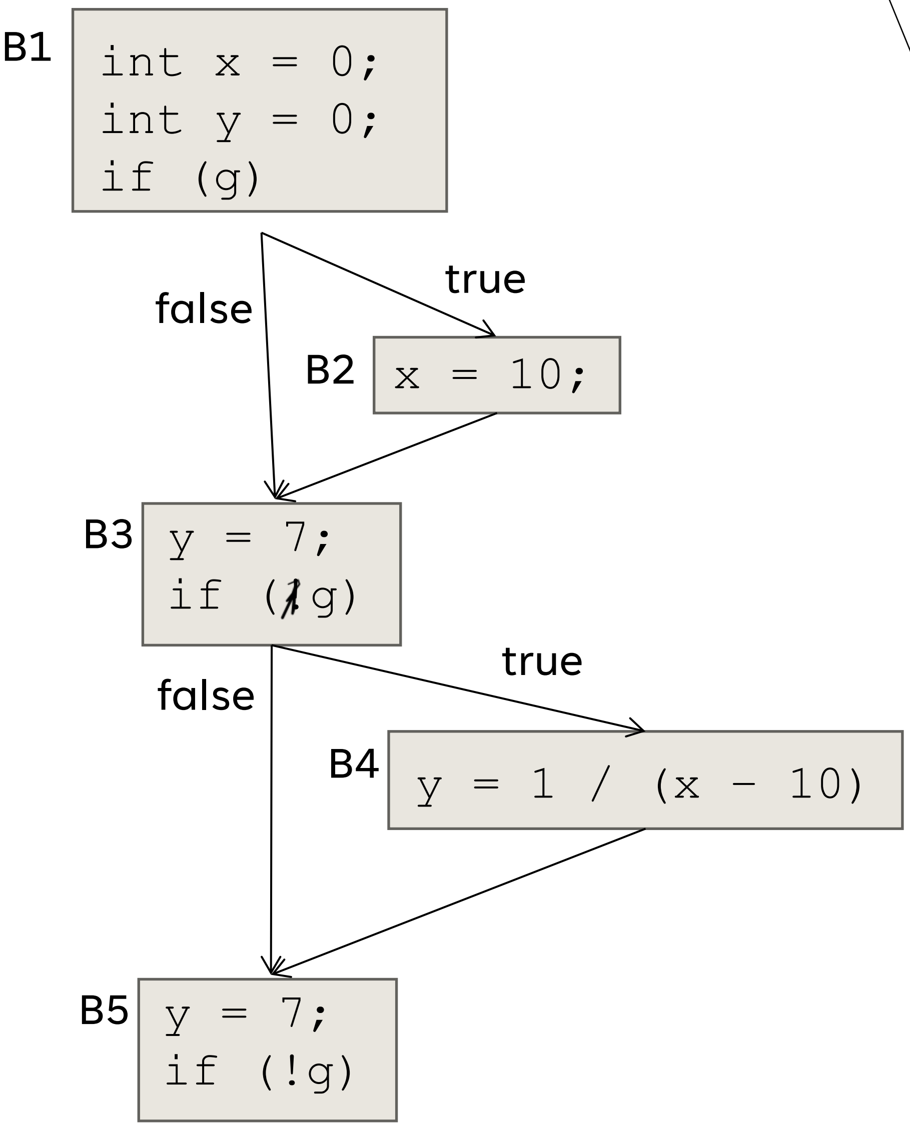
Account for Program Flow, not paths

- When control flow merges, merge the value sets

predicates

2^k

```
int x = 0;
int y = 0;
if (g)
  x = 10;
y = 7
if (!g)
  y = 1 / (x - 10);
return;
```



LOOPS ARE TOUGH TO HANDLE!

REVIEW: DATAFLOW ANALYSIS

Issues with Loops

- Generate lots of paths
- Cyclic data dependency

LECTURE OUTLINE

- Handling cyclic dependency
- Termination
- Handling large value sets

A WORD OF CAUTION

DATAFLOW ANALYSIS

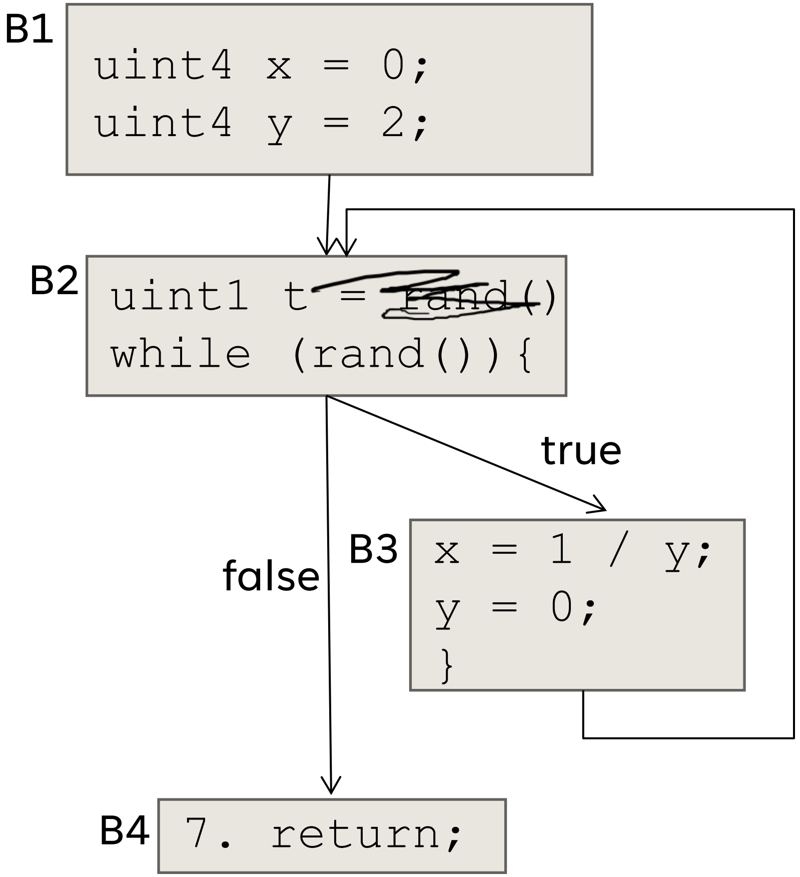
WE NEED TO BUILD UP A LOT OF INTERLOCKING MACHINERY FOR A "REAL" FLOW-SENSITIVE ANALYSIS

- I'll present a simplified algorithm here with some subtle problems, which we'll fix up next time

WHERE TO START ANALYSIS

DATAFLOW ANALYSIS

```
uint4 x = 0;
uint4 y = 2;
uint1 t;
while (t=rand()){
  x = 1 / y;
  y = 0;
}
return;
```



```
uint4 x = 0;
uint4 y = 2;
```

uint1 t ~~= rand~~()
while (rand()){

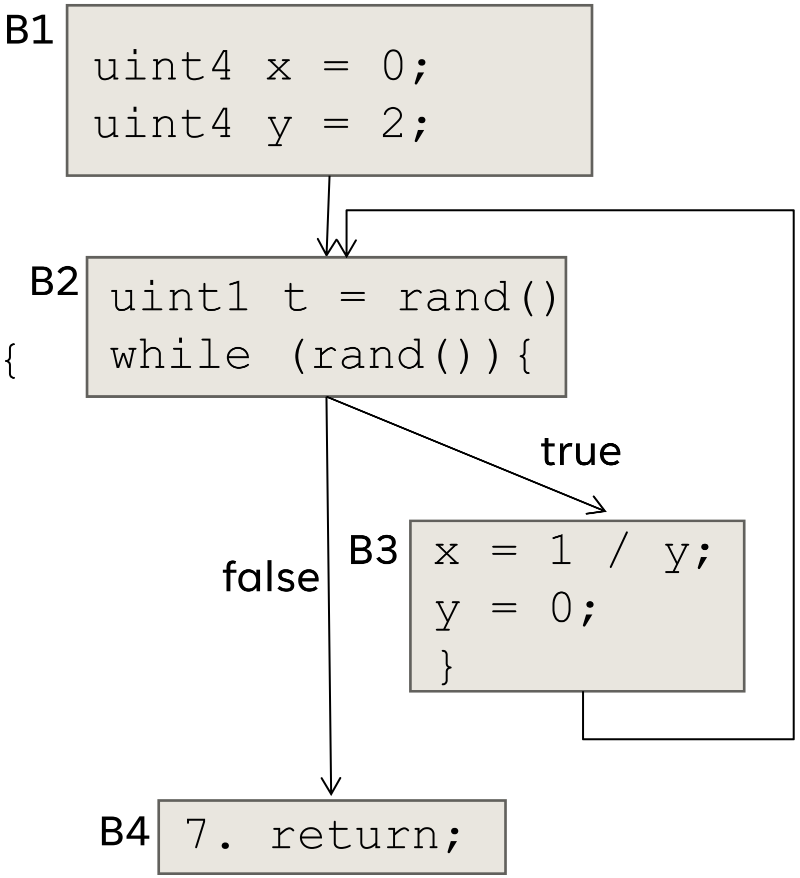
```
x = 1 / y;
y = 0;
}
```

```
7. return;
```

WHERE TO START ANALYSIS

DATAFLOW ANALYSIS

```
uint4 x = 0;
uint4 y = 2;
uint1 t;
while (t=rand()){
  x = 1 / y;
  y = 0;
}
return;
```



```
uint4 x = 0;
uint4 y = 2;
```

```
uint1 t = rand()
while (rand()){
```

```
x = 1 / y;
y = 0;
}
```

```
return;
```

	x	y	t
B1 in	{1-15}	{1-15}	{0,1}
B1 out	{0}	{2}	{0,1}
B2 in	???	???	???
B2 out			
B3 in			
B3 out			
B4 in			
B4 out			

CHAOTIC ITERATION

STATIC ANALYSIS: CONTROL FLOW GRAPHS

A WORKLIST ALGORITHM

- Select the next worklist item in any order
- Necessarily assumes progress towards some goal

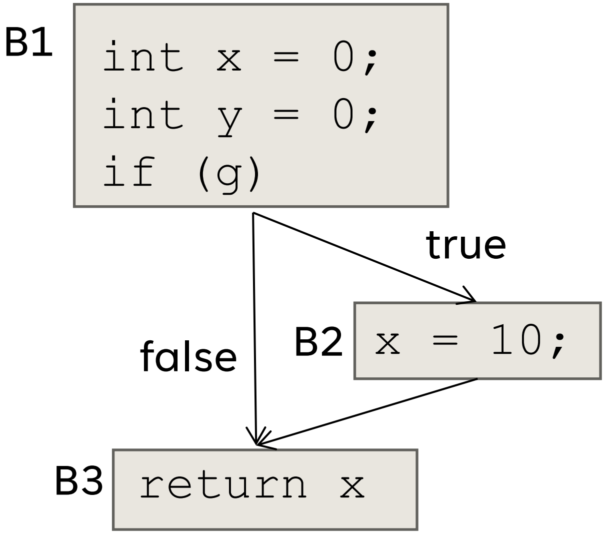
DEALING WITH “UNCOMPUTED” SETS

- Assume a reasonable “initial” value

Surprisingly, not a band with merch at Hot Topic

CHAOTIC ITERATION

STATIC ANALYSIS: CONTROL FLOW GRAPHS



```
int x = 0;
int y = 0;
if (g)
```

```
x = 10;
```

```
return x
```

A Worklist algorithm

- Select the next worklist item in any order
- Necessarily assumes progress towards some goal

Dealing with "uncomputed" sets

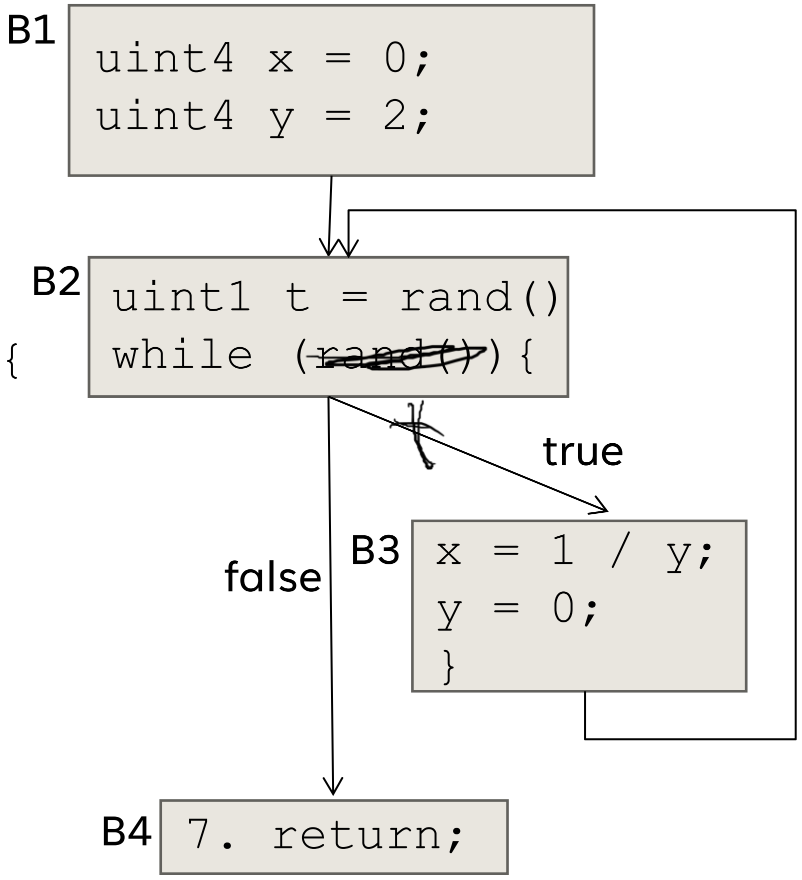
- Assume a reasonable "initial" value
- For the sake of complete over-approximation, let's assume a set that hasn't been computed take could take on ANY value

	x	y	g
B1 in	(MIN-MAX	MIN-MAX	MIN-MAX
B1 out			
B2 in			
B2 out			
B3 in			
B3 out			

CHAOTIC ITERATION: LOOPS

DATAFLOW ANALYSIS

```
uint4 x = 0;
uint4 y = 2;
uint1 t;
while (t=rand()){
  x = 1 / y;
  y = 0;
}
return;
```



	x	y	t
B1 in	{0-15}	{0-15}	{0,1}
B1 out	{0}	{2}	{0,1}

B2 in	~~{0-15}~~ [illegible]	{0-15}
		{0,2}
			{0,1}
			{0,1}

B2 out	{0}	{0,2}	{0,1}
B3 in	{0-15}	{0-15}	~~{0,1}~~ 1
B3 out	0	0	[illegible]
B4 in	~~{1-15}~~ {0}	~~{1-15}~~ {0,2}	~~{0,1}~~ {0}
B4 out	{0}	{0,2}	{0}

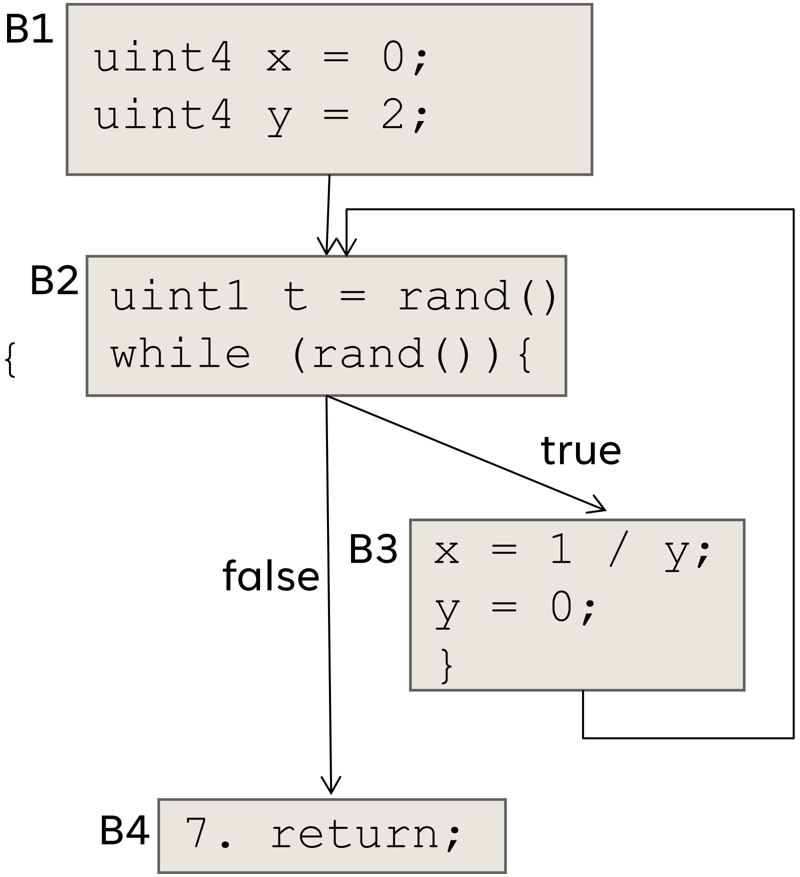
LECTURE OUTLINE

- Handling cyclic dependency
- Termination
- Handling large value sets

WHEN TO STOP ANALYSIS?

DATAFLOW ANALYSIS

```
uint4 x = 0;
uint4 y = 2;
uint1 t;
while (t=rand()){
  x = 1 / y;
  y = 0;
}
return;
```



```
uint4 x = 0;
uint4 y = 2;
```

```
uint1 t = rand()
while (rand()){
```

```
x = 1 / y;
y = 0;
}
```

```
7. return;
```

	x	y	t
B1 in	{1-15}	{1-15}	{0,1}
B1 out	{0}	{2}	{0,1}
B2 in	{0}	{2,0}	{0,1}
B2 out	{0}	{2,0}	{0,1}
B3 in	{0}	{2,0}	{1}
B3 out	{0}	{0}	{1}
B4 in	{0}	{2,0}	{0}
B4 out	{0}	{2,0}	{0}

ANALYSIS PROGRESS

STATIC ANALYSIS: CONTROL FLOW GRAPHS

ANALYSIS ENDS WHEN THE FACT SETS REACH ***SATURATION***

- No additional elements will ever be added
- It sure would be nice if we could guarantee that this will happen!

When your fact sets couldn't possibly hold any more data

FIXED-POINTS

STATIC ANALYSIS: CONTROL FLOW GRAPHS

A Fixed-Point (AKA fixpoint, fixed point)

- A value that does not change under a given transformation

Our value-set analysis WILL have Facts that reach a Fixed-point

Why?

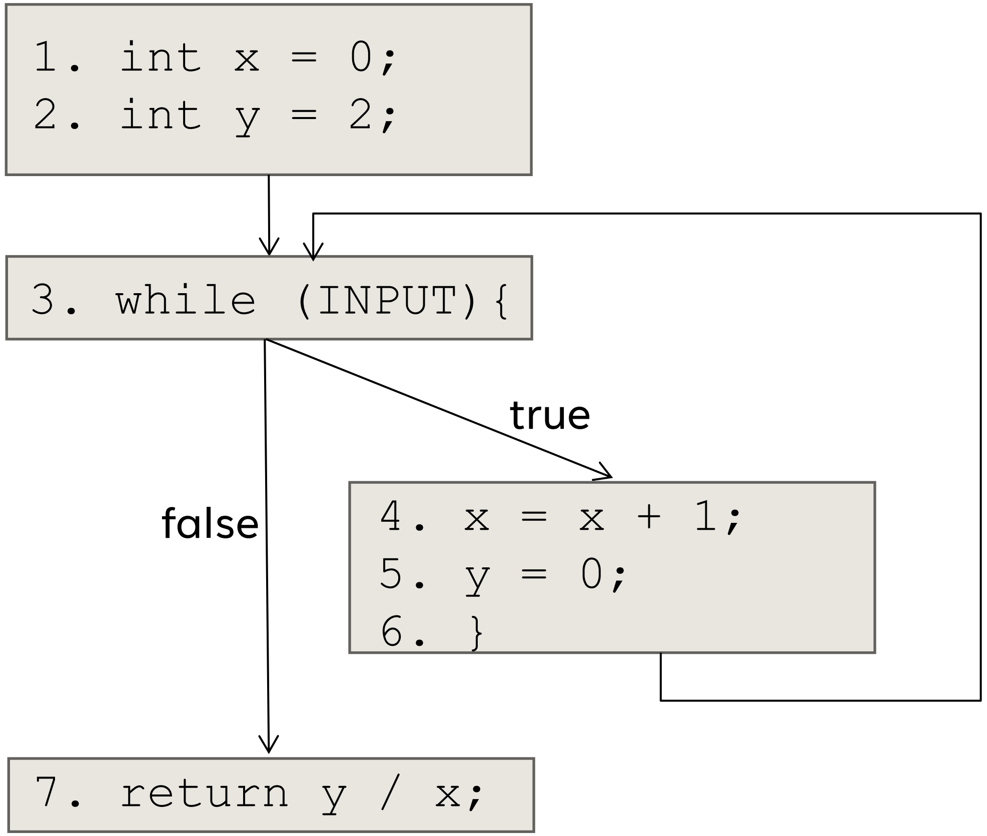
- Finite set of configurations over INT32s
- Data transforms only **add** data to fact sets

LECTURE OUTLINE

- Breaking cyclic dependency
- Termination
- Handling large value sets

WHERE TO STOP THIS ANALYSIS?

ANALYSIS TERMINATION



```
1. int x = 0;
2. int y = 2;
```

```
3. while (INPUT){
```

```
4. x = x + 1;
5. y = 0;
6. }
```

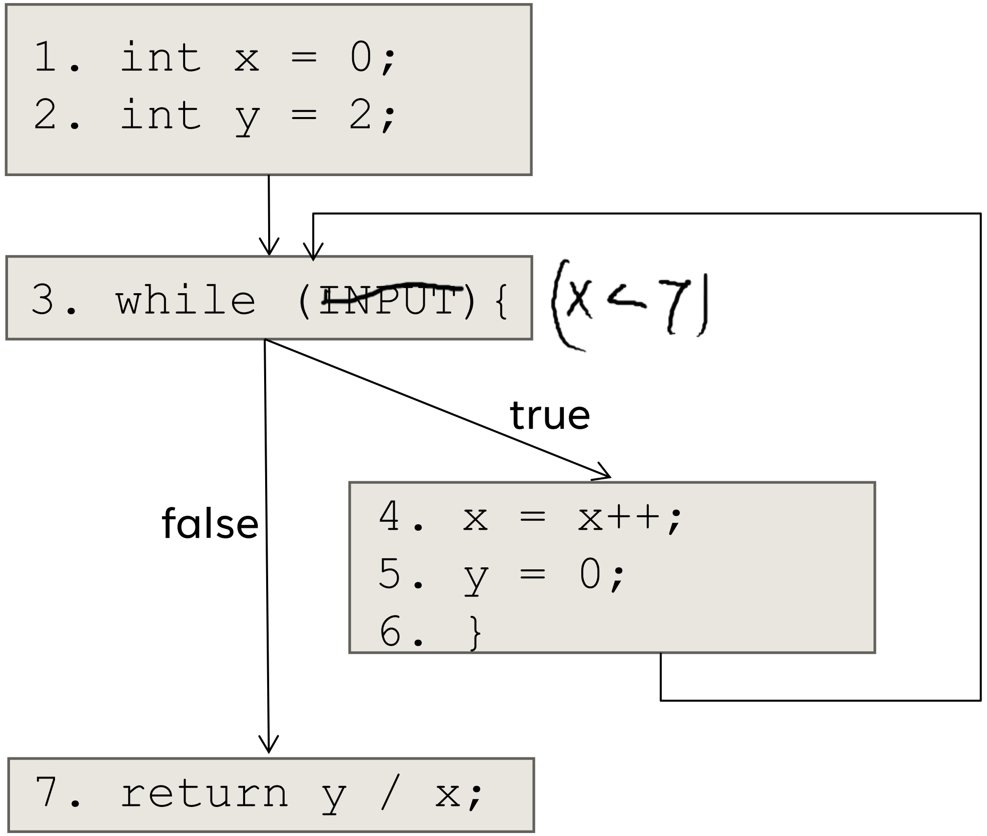
```
7. return y / x;
```

x:0, y:2 {x:{0,1}, y:{0,2}, x:{0,1,2}, y: {0,2}

x:1, y:0 {x:1,2}, y:{0,2}, x:{1,2,3}, y:{0}

WIDENING

ANALYSIS TERMINATION



```
1. int x = 0;
2. int y = 2;
```

```
3. while (~~INPUT~~){
```

(x<7)

```
4. x = x++;
5. y = 0;
6. }
```

```
7. return y / x;
```

Accelerate progress towards Fix-Point

- Lots of (over-approximate) ways to do this
- 1 simple idea: if we hit N values, immediately change the fact set to “All integers”

x: {MIN - MAX}

LECTURE END!

Described some of the issues and Fixes for dataflow in the presence of loops

1) Lack of distinct order blocks

- Chaotic iteration

2) Possibility of large value sets

- widening