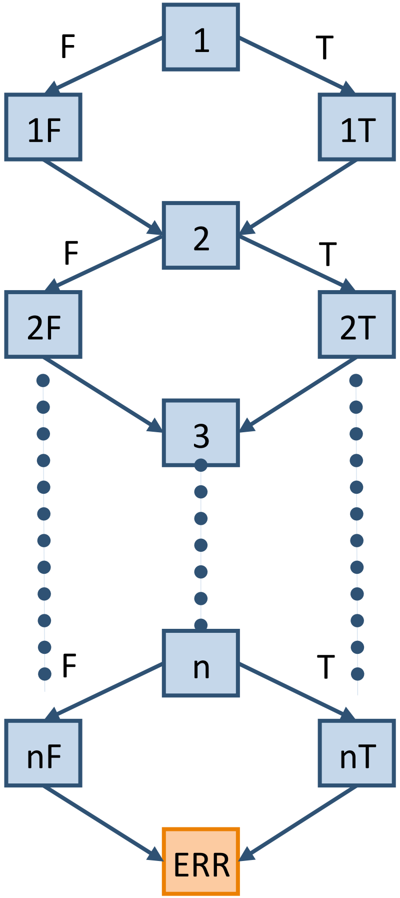
Vulnerability Analysis (III): Static Analysis

Slide credit: Vijay D’Silva

1	Efficiency of Symbolic Execution
2	A Static Analysis Analogy
3	Syntactic Analysis
4	Semantics-Based Analysis

1	Efficiency of Symbolic Execution
2	A Static Analysis Analogy
3	Syntactic Analysis
4	Semantics-Based Analysis

Quiz: Branches and Paths

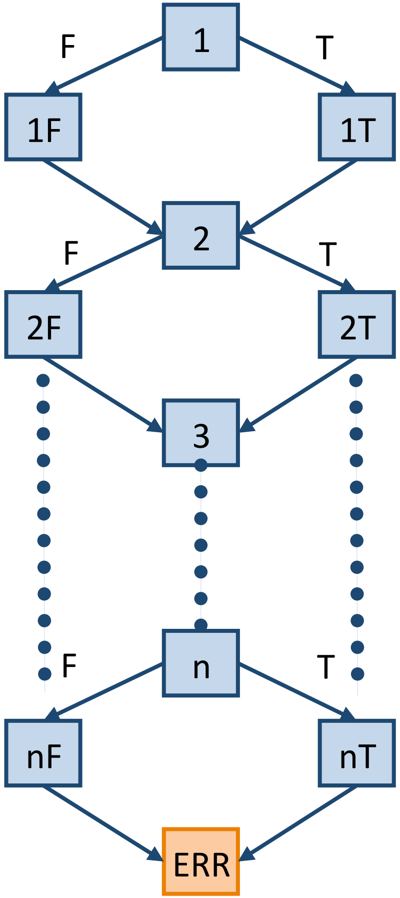


Suppose we want to know if there is a feasible path to the location ERR in this program.

Suppose we generate one path predicate for each path through this program.

How many path predicates are generated?

Quiz: Branches and Paths



Suppose we want to know if there is a feasible path to the location ERR in this program.

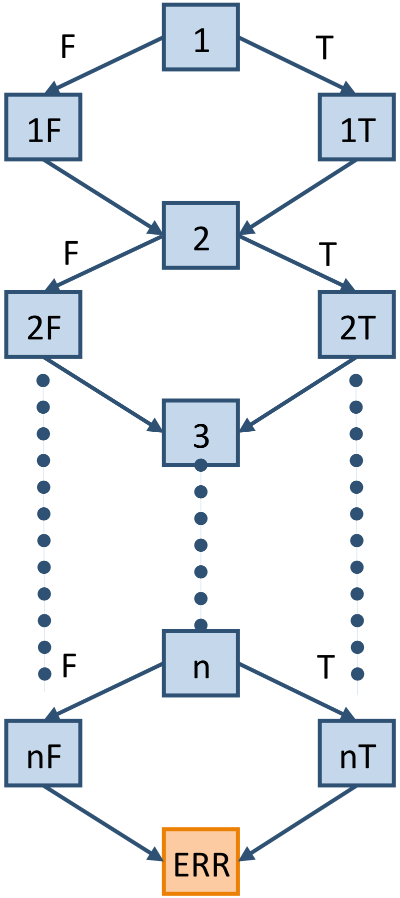
Suppose we generate one path predicate for each path through this program.

How many path predicates are generated?

2^n

Dawn Song

Quiz: Branches and Paths



Suppose we want to know if there is a feasible path to the location ERR in this program.

Suppose we generate one path predicate for each path through this program.

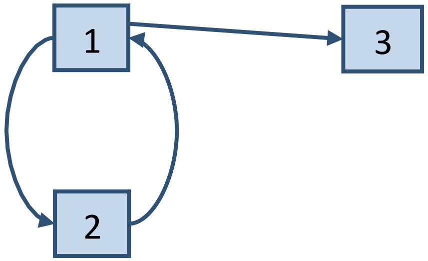
How many path predicates are generated?

2^n

Number of predicates can be *exponential in the number of branches.*

Dawn Song

Quiz: Loops and Paths

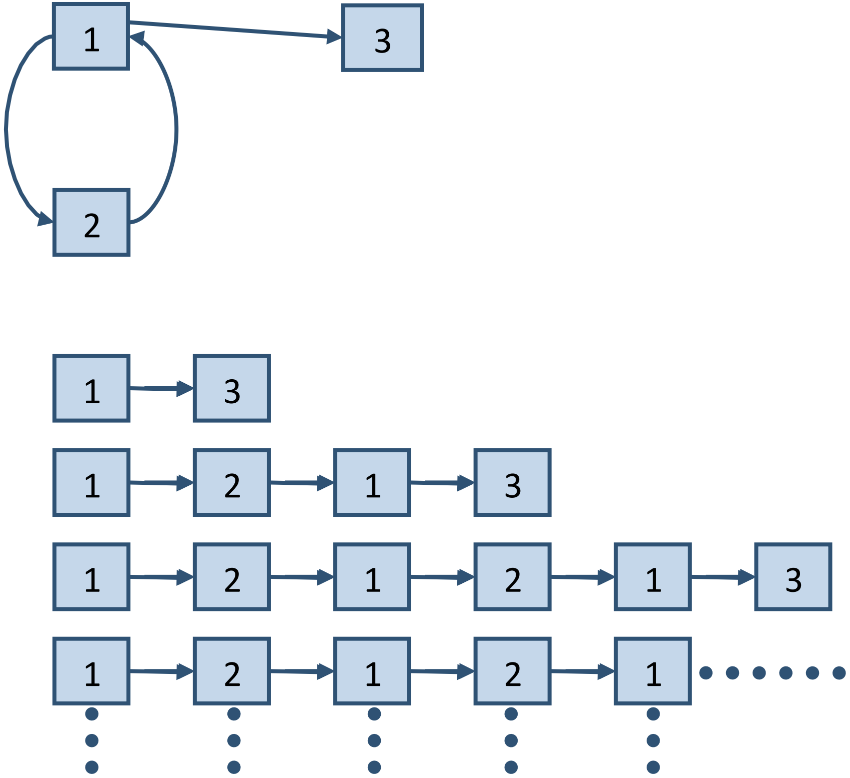


This is the structure of a program with a simple loop.

Suppose the error location is in block 3.

How many path predicates are generated?

Quiz: Loops and Paths

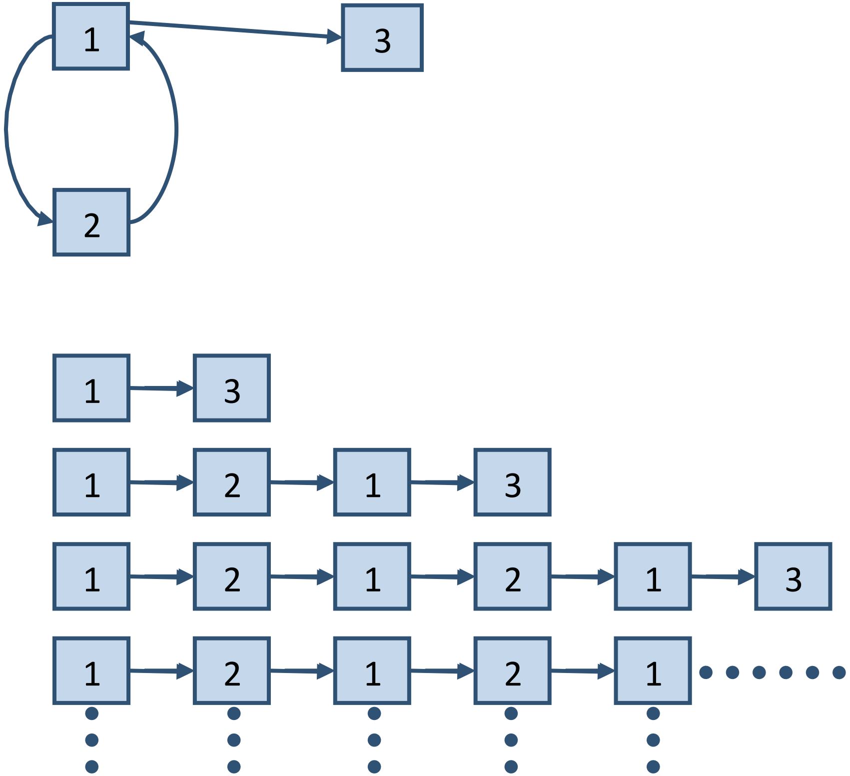


This is the structure of a program with a simple loop.

Suppose the error location is in block 3.

How many path predicates are generated?

Quiz: Loops and Paths



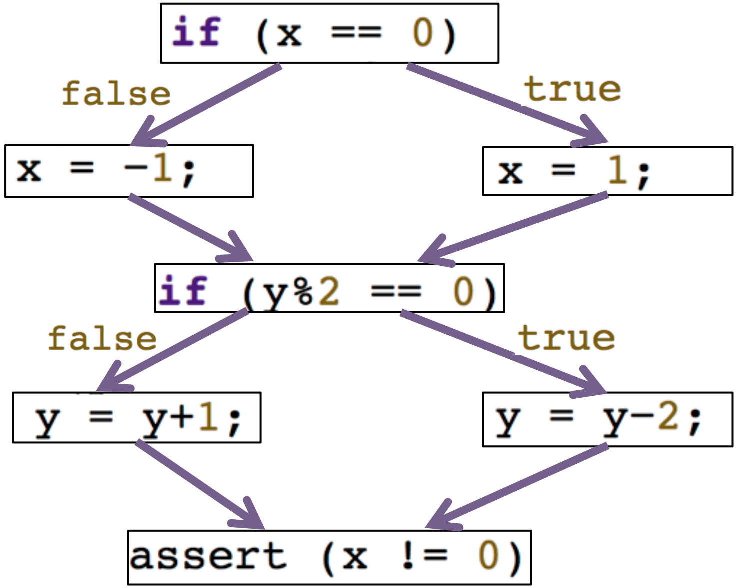
This is the structure of a program with a simple loop.

Suppose the error location is in block 3.

How many path predicates are generated?

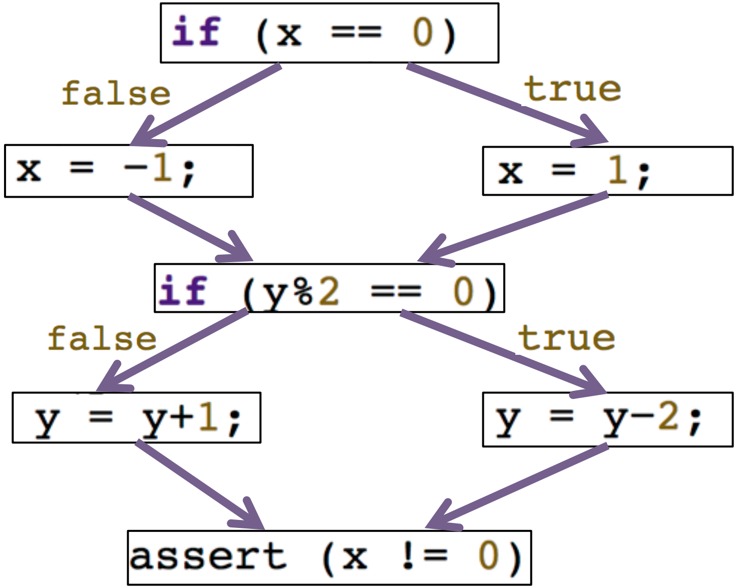
- A loop can generate an *infinite number of path predicates*
- Number of path predicates is finite only if the program terminates

Independence of Variables



How many paths to the assertion?

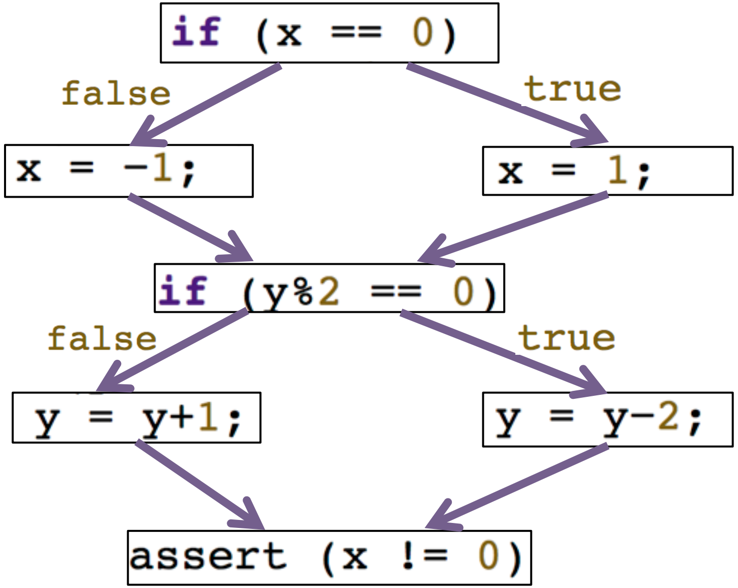
Independence of Variables



How many paths to the assertion?

4

Independence of Variables



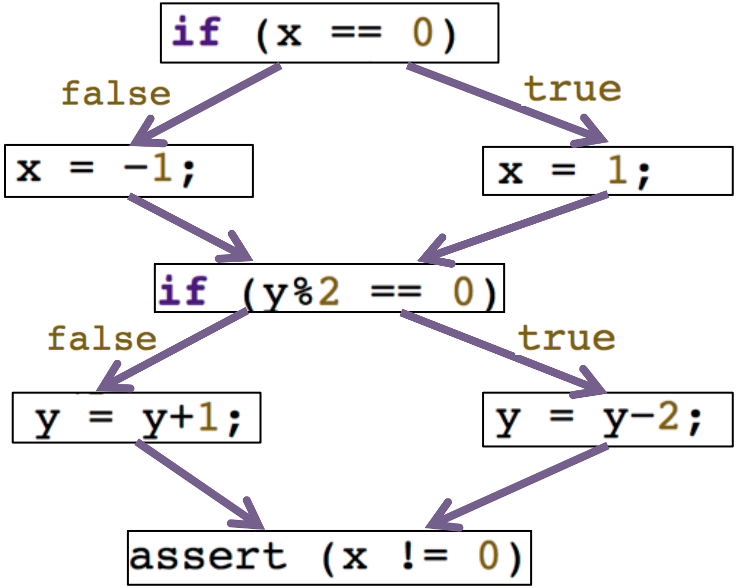
How many paths to the assertion?

4

The second branch does not affect the assertion. How many paths without the second branch?

Dawn Song

Independence of Variables



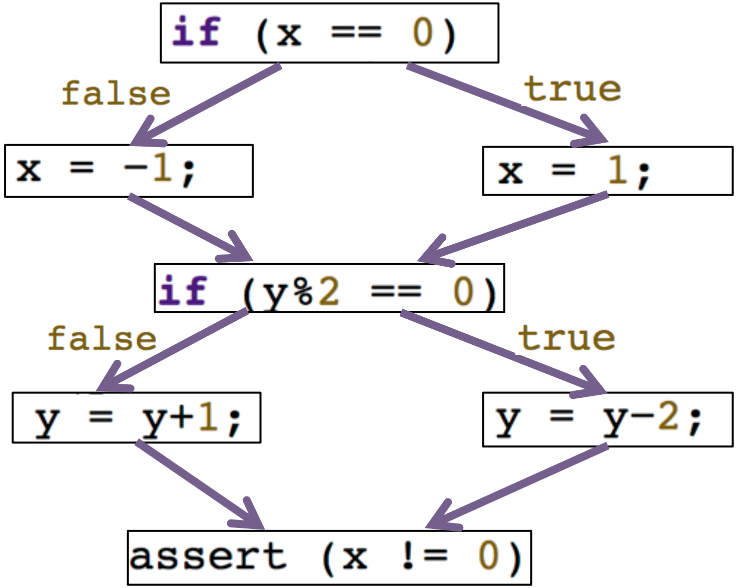
How many paths to the assertion?

4

The second branch does not affect the assertion. How many paths without the second branch?

2

Independence of Variables



How many paths to the assertion?

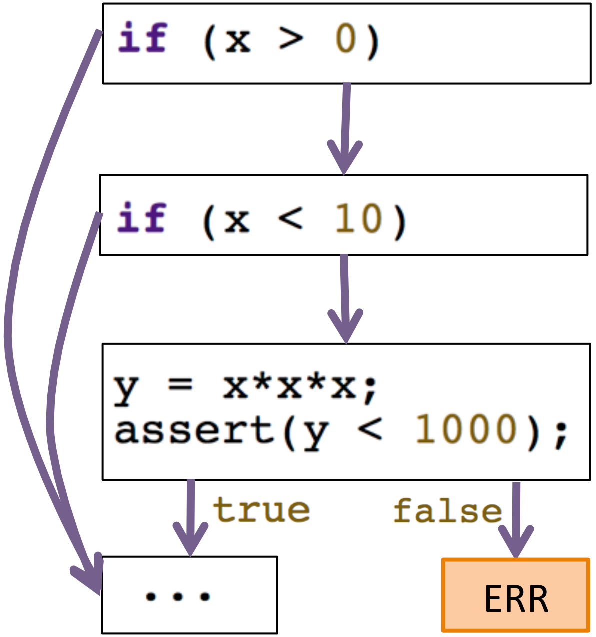
4

The second branch does not affect the assertion. How many paths without the second branch?

2

- Including all statements on a path leads to larger constraints than necessary
- Data dependencies can be used to prune paths and simplify constraints

Structure of Formulas



```
if (x > 0)
```

```
if (x < 10)
```

```
y = x*x*x;
assert(y < 1000);
```

true

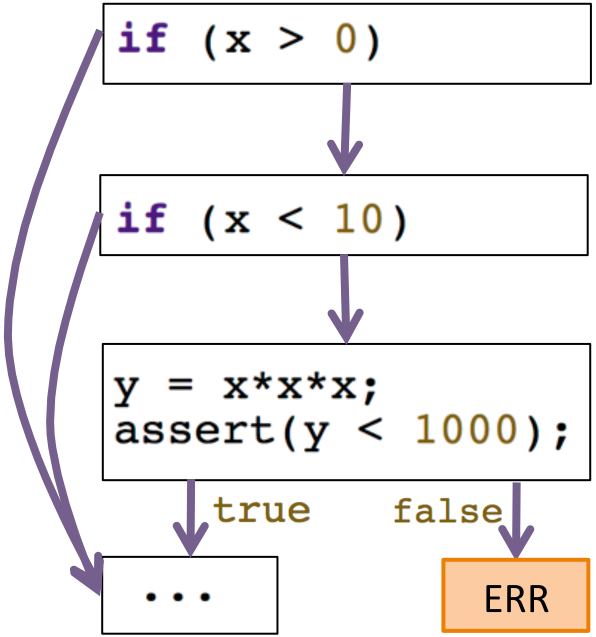
false

...

ERR

- The path predicate for this assertion violation involves bit-vector multiplication
- Reasoning about multiplication of *variables* is computationally expensive (think of multiplier circuits)

Structure of Formulas



```
if (x > 0)
```

```
if (x < 10)
```

```
y = x*x*x;
assert(y < 1000);
```

true

false

...

ERR

- The path predicate for this assertion violation involves bit-vector multiplication
- Reasoning about multiplication of *variables* is computationally expensive (think of multiplier circuits)
- Only need to show an upper bound on y
- Imprecise reasoning can be more efficient and enough

Challenges for Symbolic Execution

Control	• *Path explosion* due to branches and loops • *Redundant exploration* of same path prefixes • *Search strategy* determines if vulnerabilities are found
Data	• *Algorithmic complexity* of arithmetic and string reasoning • *Constraint explosion* because of irrelevant variables and operations • *Memory modeling* is labor intensive but necessary

How can we address these issues?

1	Efficiency of Symbolic Execution
2	Static Analysis by Analogy
3	Syntactic Analysis
4	Semantics-Based Analysis

SODA Hall

Single Program Execution

Moscone Center

Bottlenecks for Dynamic Analysis

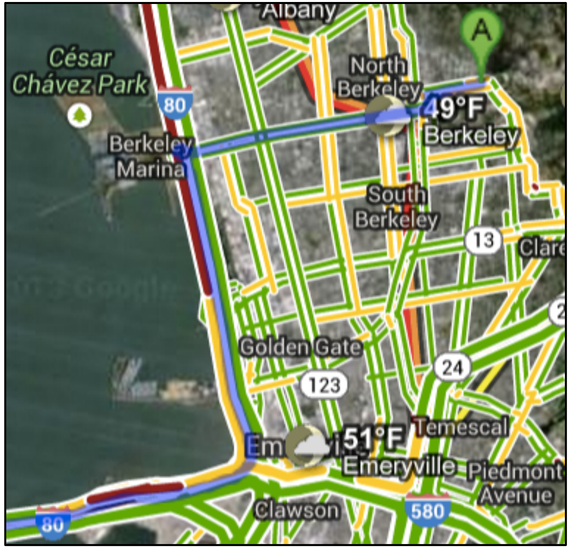
Weather

Traffic

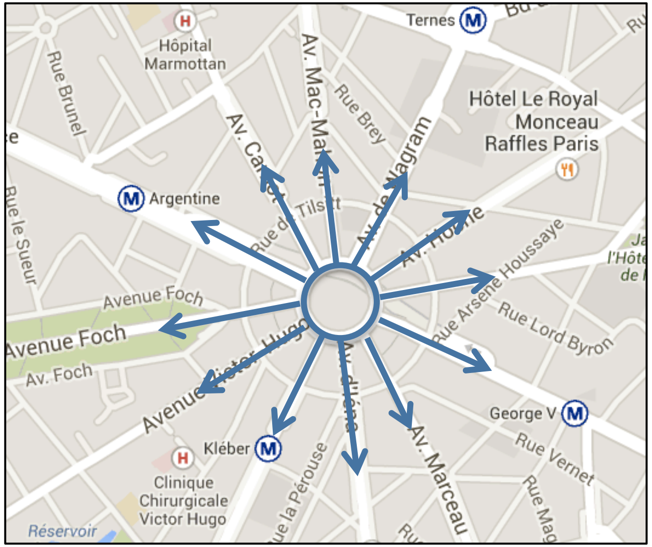
Roads

Terrain

....



Information Overload



Route Explosion

Dawn Song

Bottlenecks for Dynamic Analysis

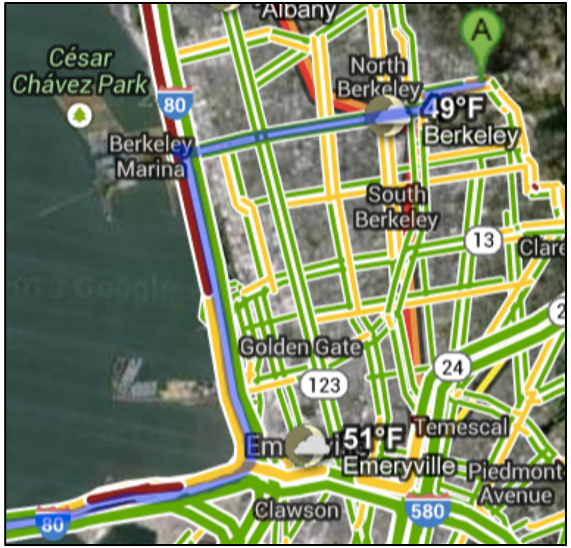
Weather

Traffic

Roads

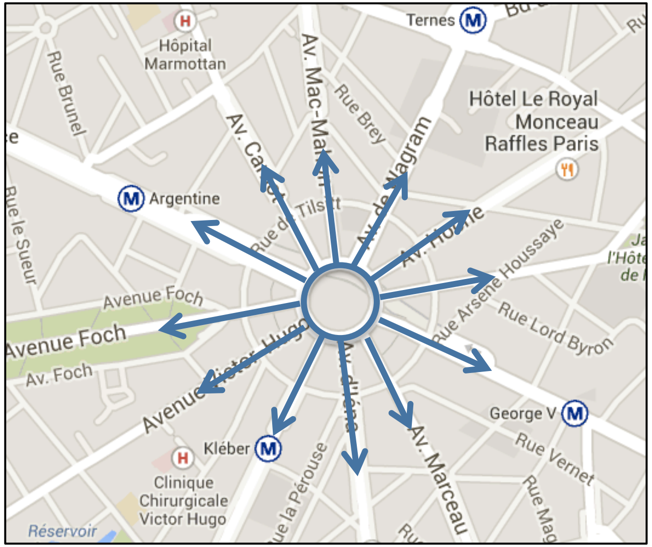
Terrain

....



Information Overload

"Data"



Route Explosion

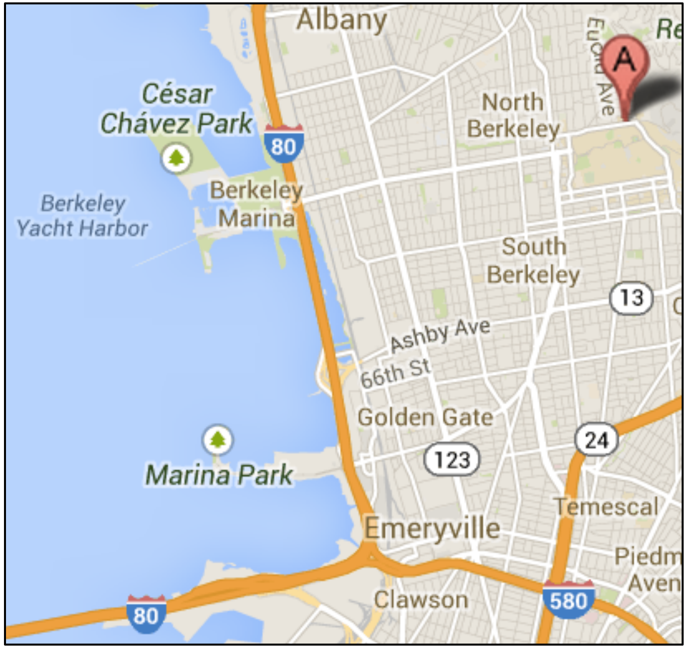
"Control"

Static Analysis

Loss of information allows for more efficient computation of some answers

Static analysis algorithms operate directly on abstract representations

For example, we can analyze all possible road-routes without even sitting in a car

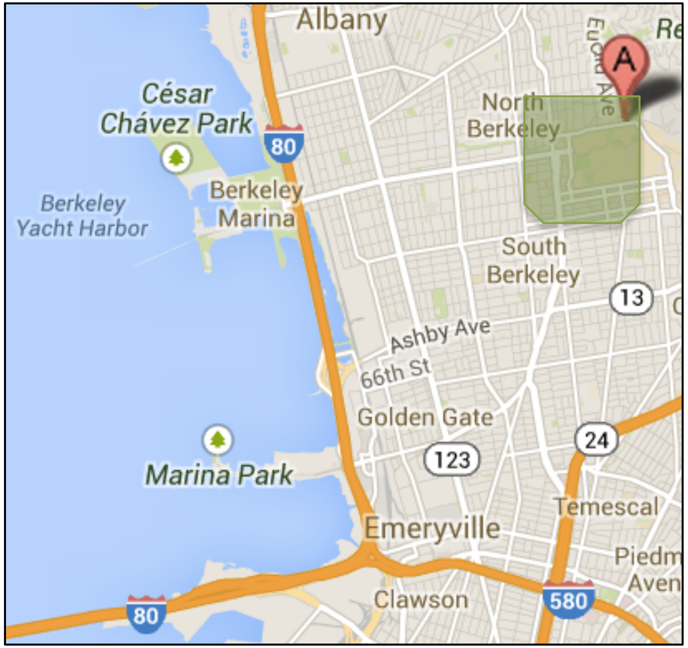


Static Analysis

Loss of information allows for more efficient computation of some answers

Static analysis algorithms operate directly on abstract representations

For example, we can analyze all possible road-routes without even sitting in a car

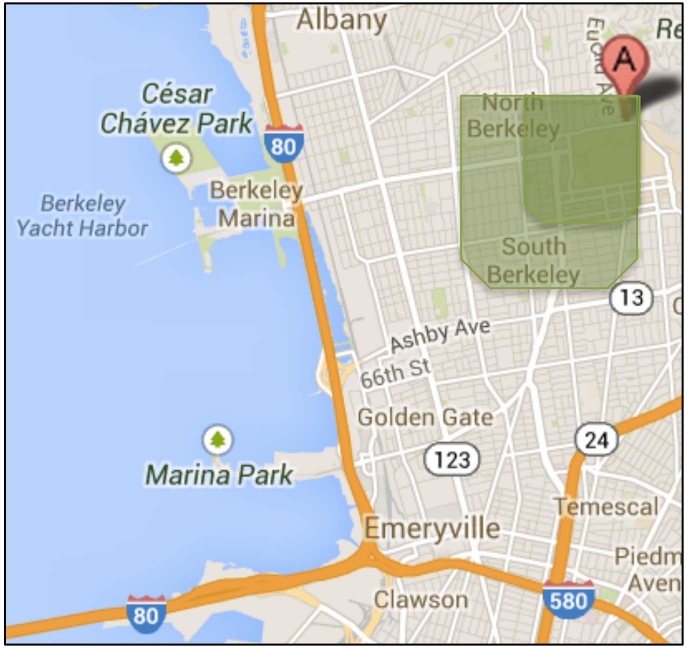


Static Analysis

Loss of information allows for more efficient computation of some answers

Static analysis algorithms operate directly on abstract representations

For example, we can analyze all possible road-routes without even sitting in a car

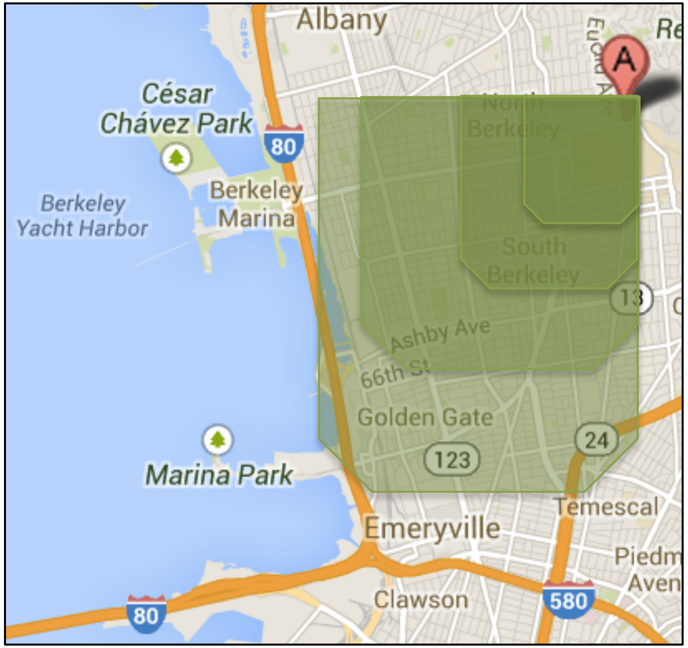


Static Analysis

Loss of information allows for more efficient computation of some answers

Static analysis algorithms operate directly on abstract representations

For example, we can analyze all possible road-routes without even sitting in a car



Static Analysis

Some questions can be answered efficiently.

"Can we drive, on land, from Melboure to Hobart?"

Not enough information to answer questions about traffic, terrain, the weather, routes from Melbourne to Sydney etc.

1	Efficiency of Symbolic Execution
2	A Static Analysis Analogy
3	Syntactic Analysis
4	Semantics-Based Analysis

Static Analysis

A *static analysis* is one that does not execute the program.

A *syntactic analysis* uses the code text but does not interpret statements

A *semantic analysis* interprets statements and updates facts based on statements in the code

Syntactic Example: Optional Arguments

- The system call open() has optional arguments

```
int open( const char ''* path'', int ''oflag'', ... /* mode_t mode */);
```

- Typical mistake:

```
fd = open("file", O_CREAT);
```

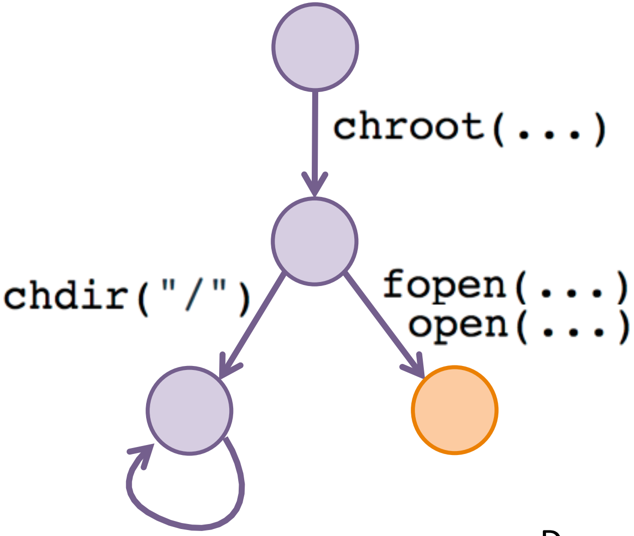
- Result: file has random permissions
- To detect this problem: Look for oflag == O_CREAT without mode argument

Dawn Song

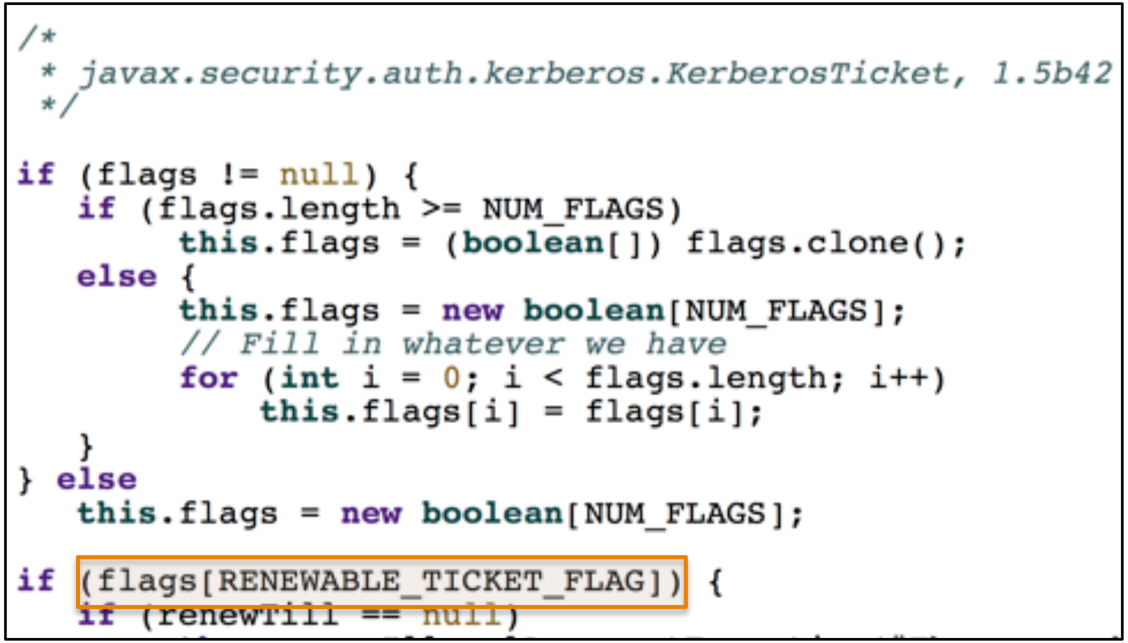
Syntactic Example: Calling Conventions

- Goal: confine a process to a “jail” in the filesystem
- Use chroot() to change the filesystem root for a process
- Problem: chroot() does not itself change the current working directory
- Result: fopen may refer to a file outside the “jail”
- Detection: look for patterns matching the specification

```
chroot("/tmp/sandbox");

fd = fopen("../etc/passwd", "r");
```



Syntactic Example: Name Confusion



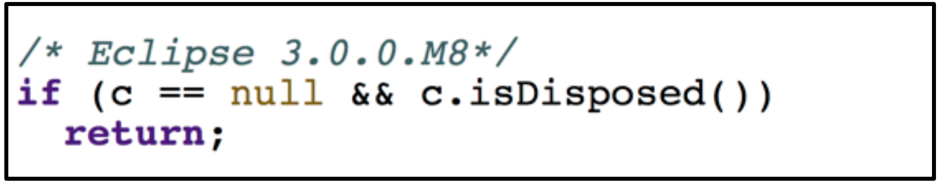
```
/*
 * javax.security.auth.kerberos.KerberosTicket, 1.5b42
 */

if (flags != null) {
   if (flags.length >= NUM_FLAGS)
        this.flags = (boolean[]) flags.clone();
   else {
        this.flags = new boolean[NUM_FLAGS];
        // Fill in whatever we have
        for (int i = 0; i < flags.length; i++)
            this.flags[i] = flags[i];
   }
} else
   this.flags = new boolean[NUM_FLAGS];

if (flags[RENEWABLE_TICKET_FLAG]) {
   if (renewTill == null)
```

source: *Squashing Bugs with Static Analysis,* William Pugh, 2006

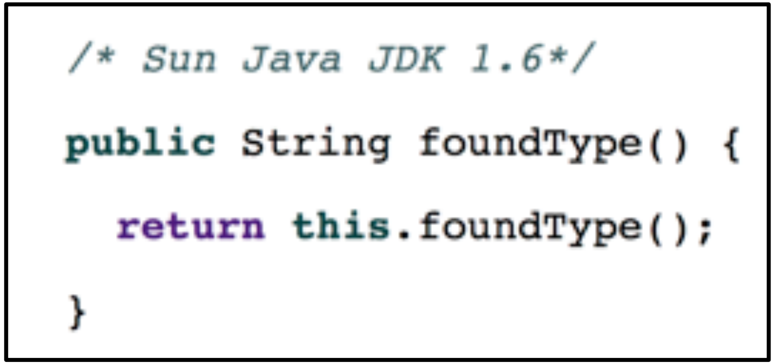
- `flags` is a parameter, `this.flags` is a field
- Problem: check does not prevent `null` dereference
- Result: Potential Null Pointer Dereference
- Detection: find similar names on code paths where security-relevant conditions are checked

Quiz

Can you identify the problems in the following code? (all taken from well tested, production software)



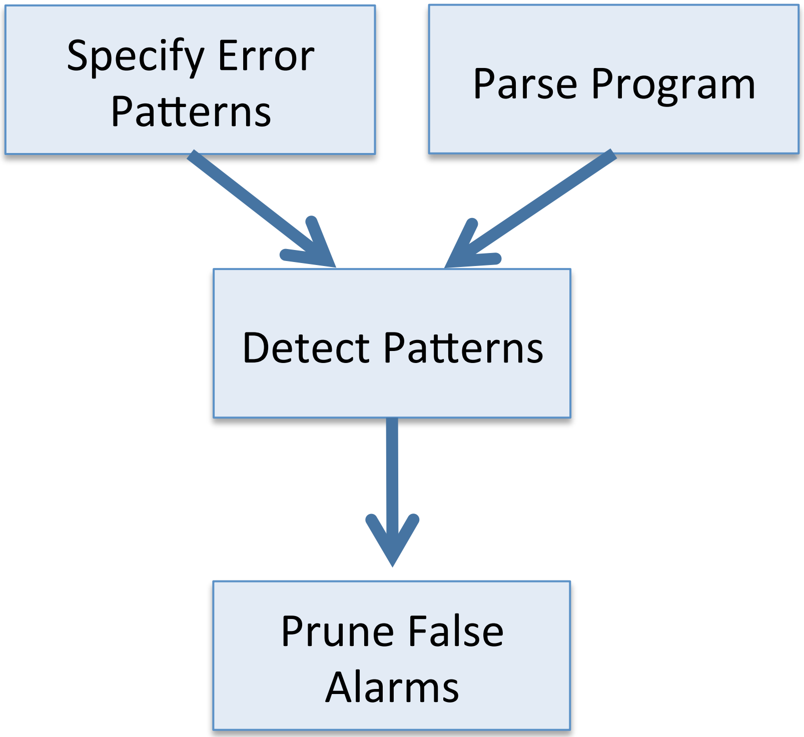
```
/* Eclipse 3.0.0.M8*/
if (c == null && c.isDisposed())
  return;
```



```
/* Sun Java JDK 1.6*/

public String foundType() {

   return this.foundType();

}
```

source: *Squashing Bugs with Static Analysis,* William Pugh, 2006

Syntactic Analysis



Specify Error Patterns

Parse Program

Detect Patterns

Prune False Alarms

Error patterns: Heuristically observed common error patterns in practice

Parsing: generates data structure used for error detection

Detection: match pattern against program representation

Pruning: Used to eliminate common false alarms

Error Pattern Types

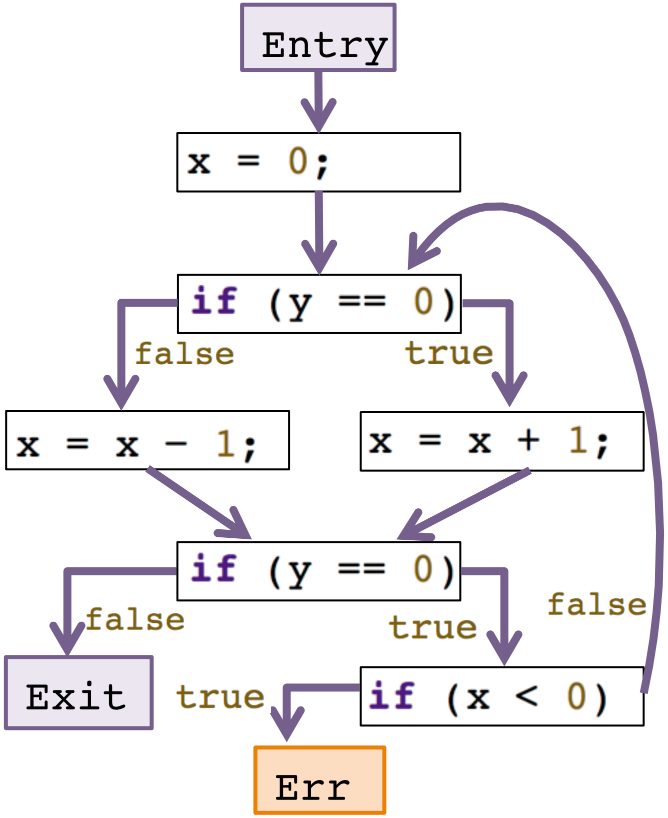
Error Type	Examples
Typos	= vs == , &x vs. x , missing/extra semi-colons
API Usage	chroot, multiple locking, etc.
Copy-Paste	variable names/increments not updated
Identifier confusion	global and local variables, fields and parameters

Pattern Representation and Detection

Representation	Types of Algorithms
String	Subsequence mining, edit distance, matching
Parse Tree	Pattern matching,
Control Flow Graphs	Automata algorithms, sub-graph isomorphism

1	Efficiency of Symbolic Execution
2	A Static Analysis Analogy
3	Syntactic Analysis
4	Semantics-Based Analysis

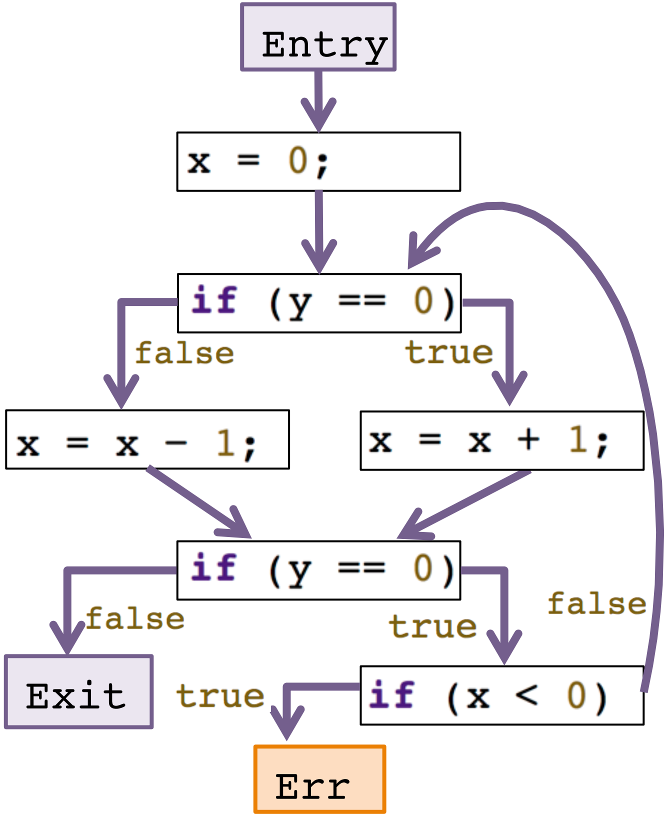
Example Program



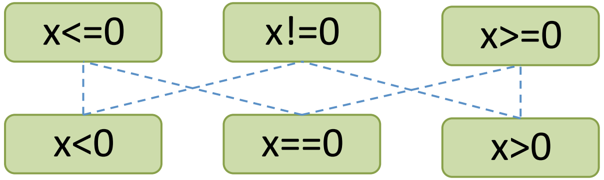
How can we automatically check if the error location is reachable in this program?

An analysis must reason about
- control flow
  - branches
  - a loop
- data
  - increment, decrement
  - comparisons with 0

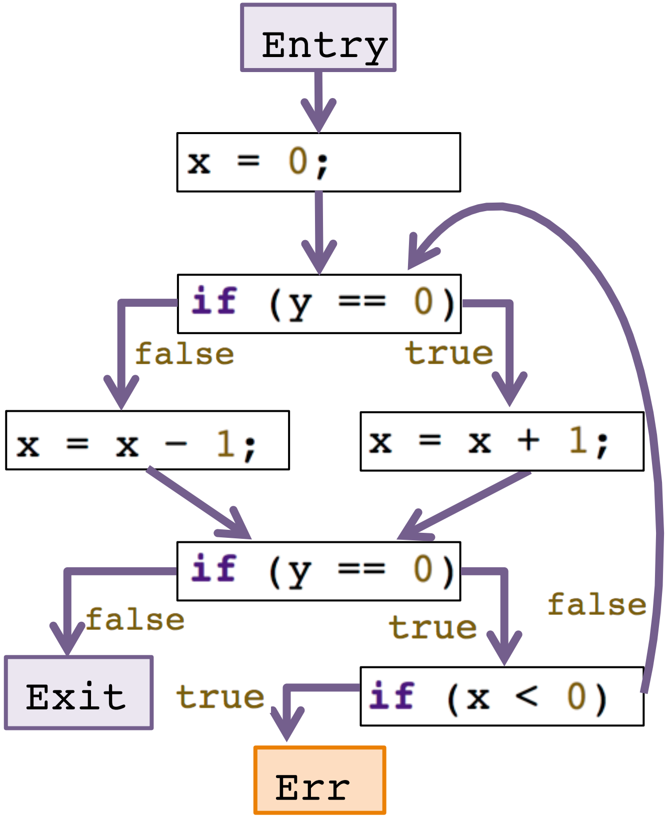
Abstracting Data



Only track relevant properties of x

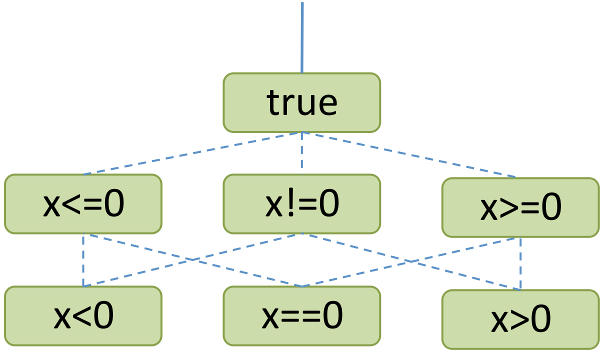


Abstracting Data

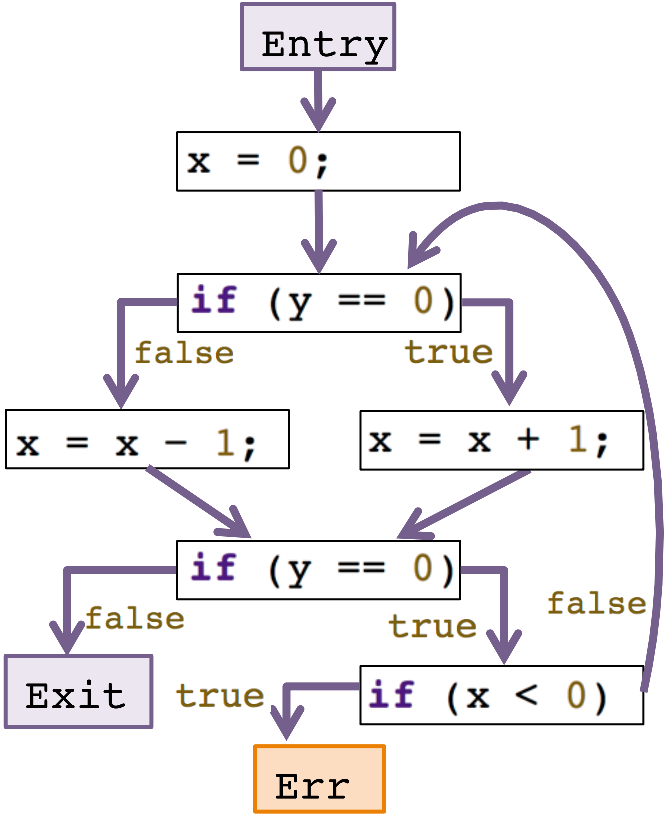


Only track relevant properties of x

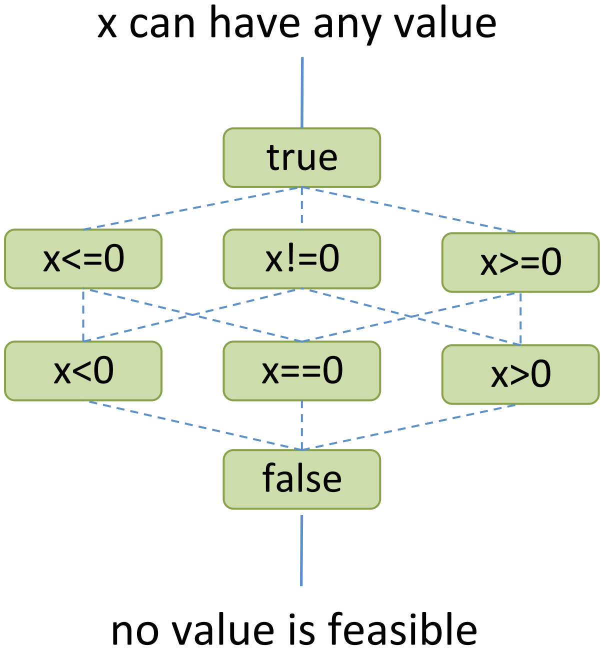
x can have any value



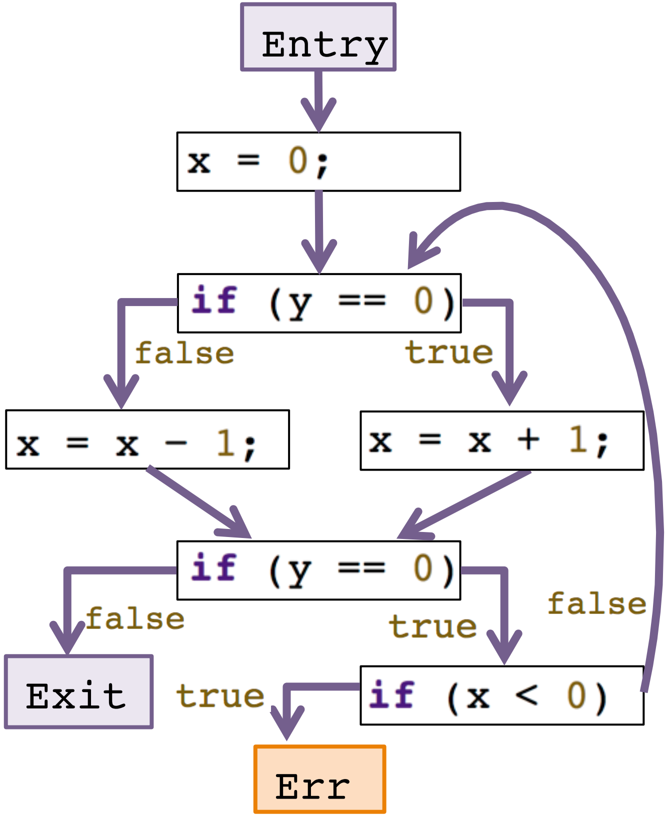
Abstracting Data



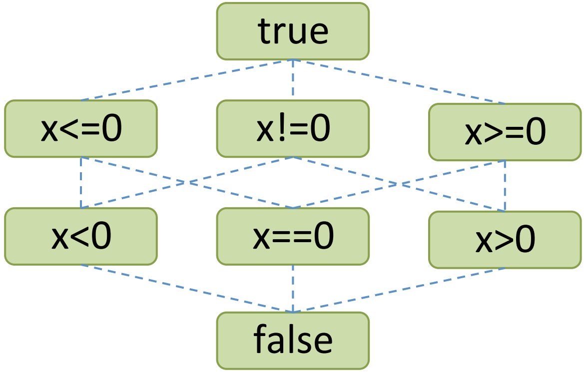
Only track relevant properties of x



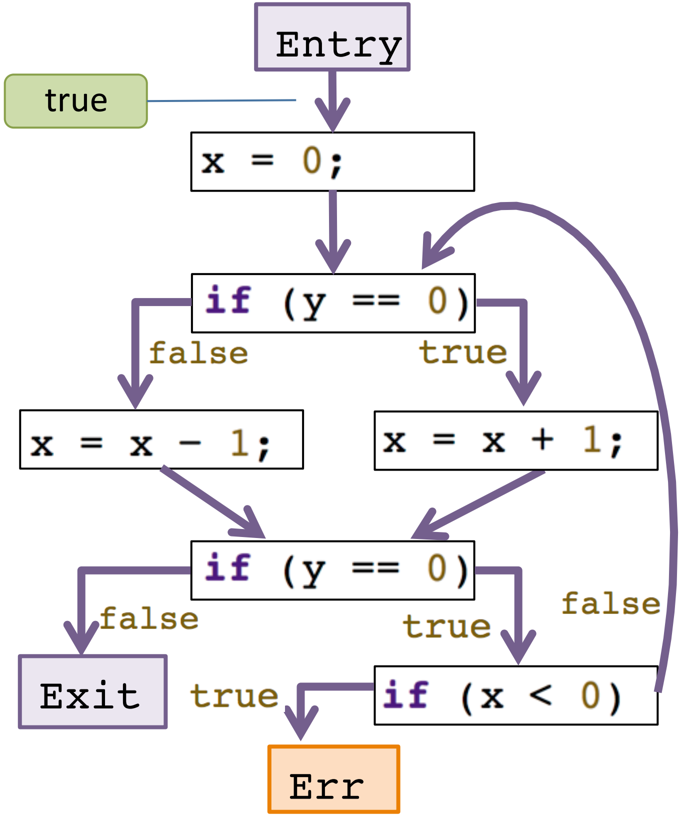
Sign Analysis



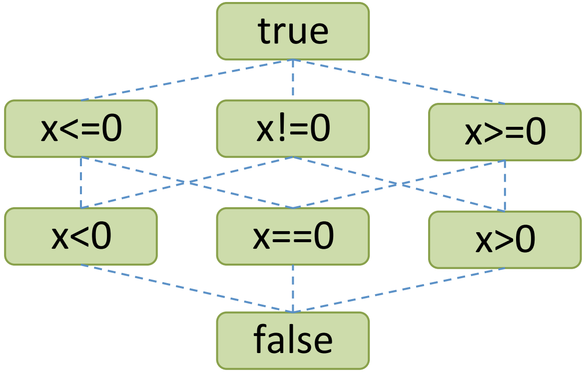
Analysis: update data about x based on control flow



Sign Analysis



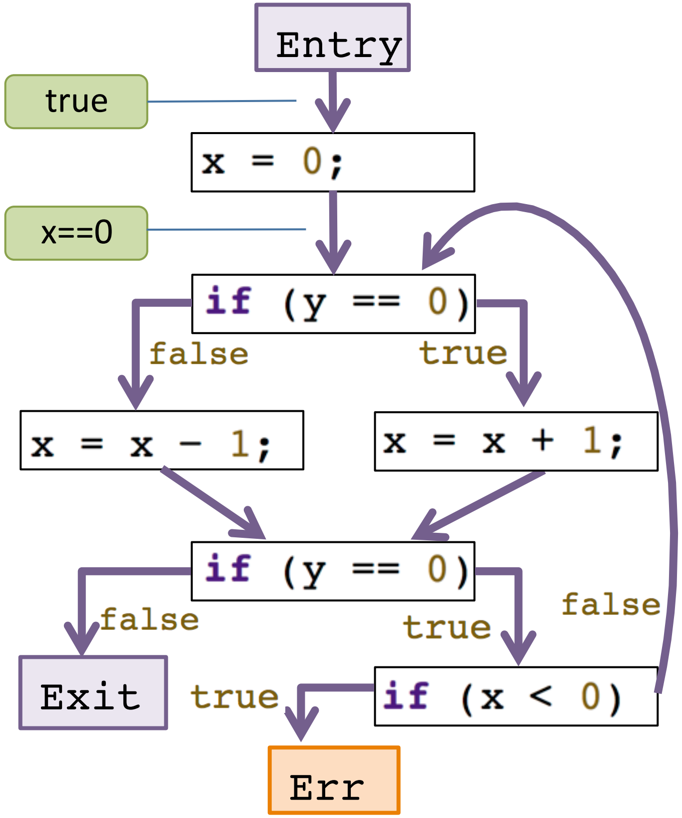
Analysis: update data about x based on control flow



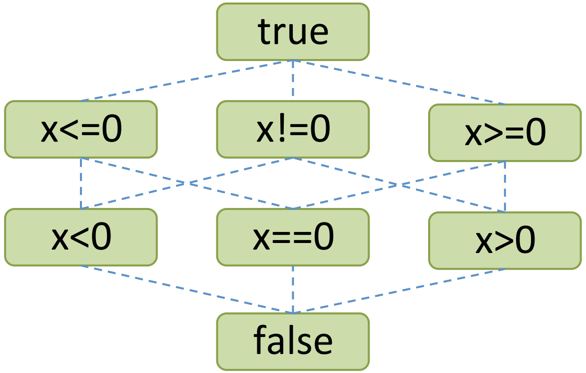
Assuming arbitrary initialization, anything can be true about x

Dawn Song

Sign Analysis

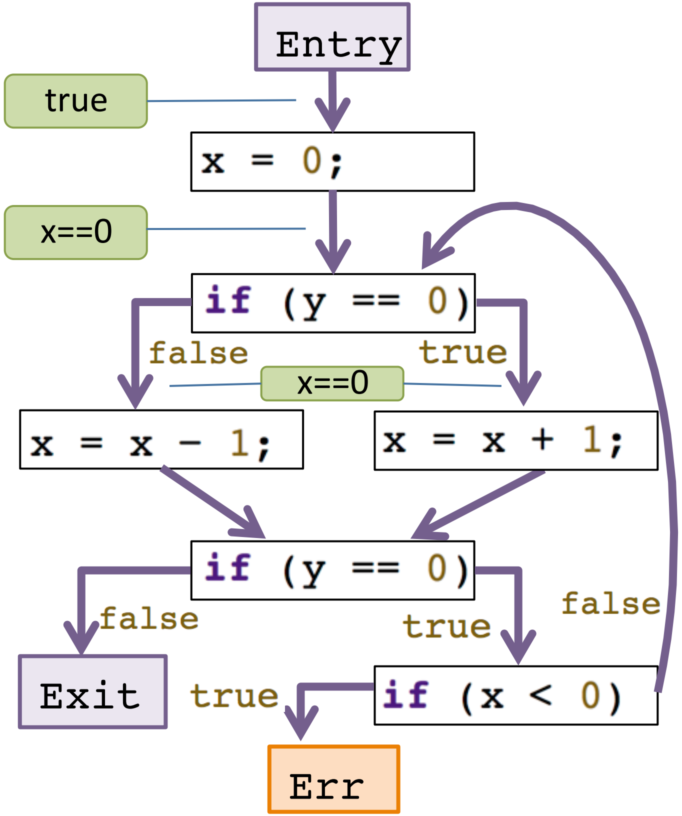


Analysis: update data about x based on control flow

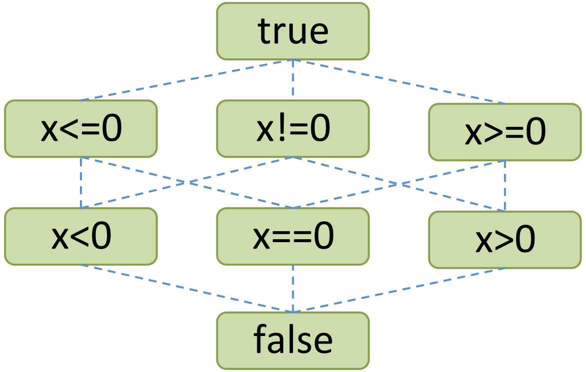


The assignment *updates* the fact about x

Sign Analysis



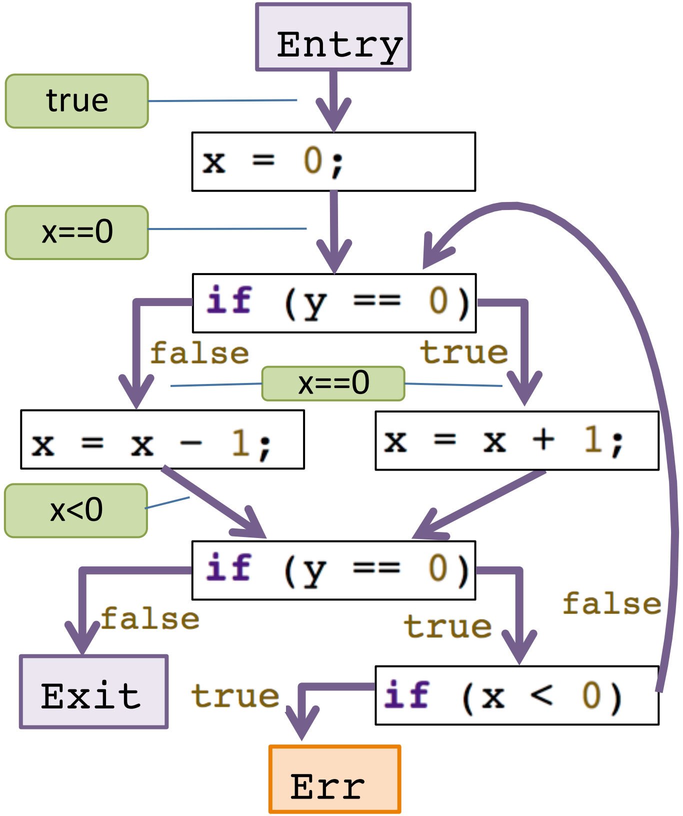
Analysis: update data about x based on control flow



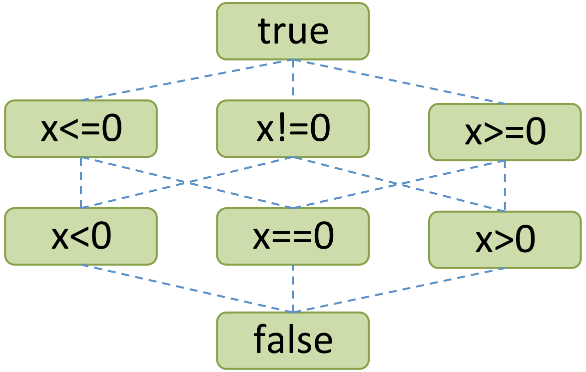
The condition does not affect x so the fact "flows through"

Dawn Song

Sign Analysis

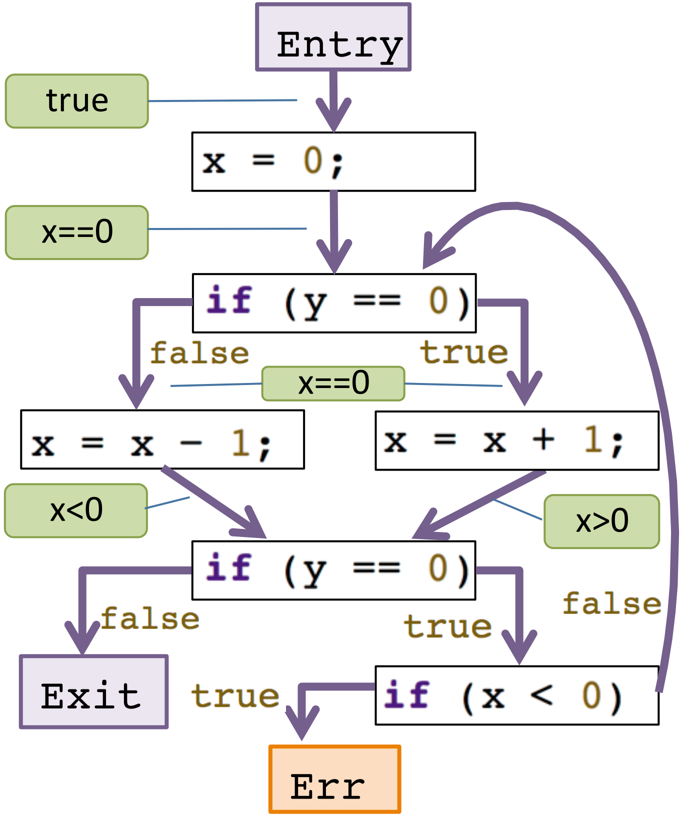


Analysis: update data about x based on control flow

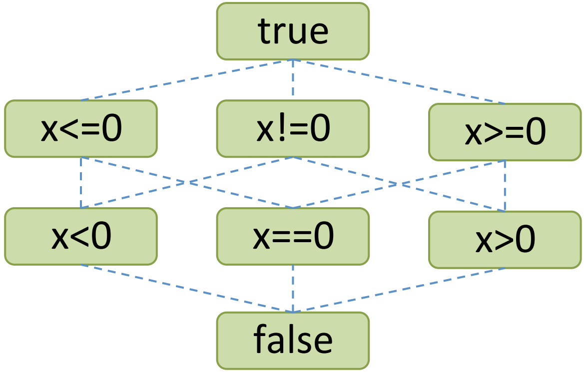


Loss of precision! We cannot write x==-1 so we *approximate* it by x<0

Sign Analysis

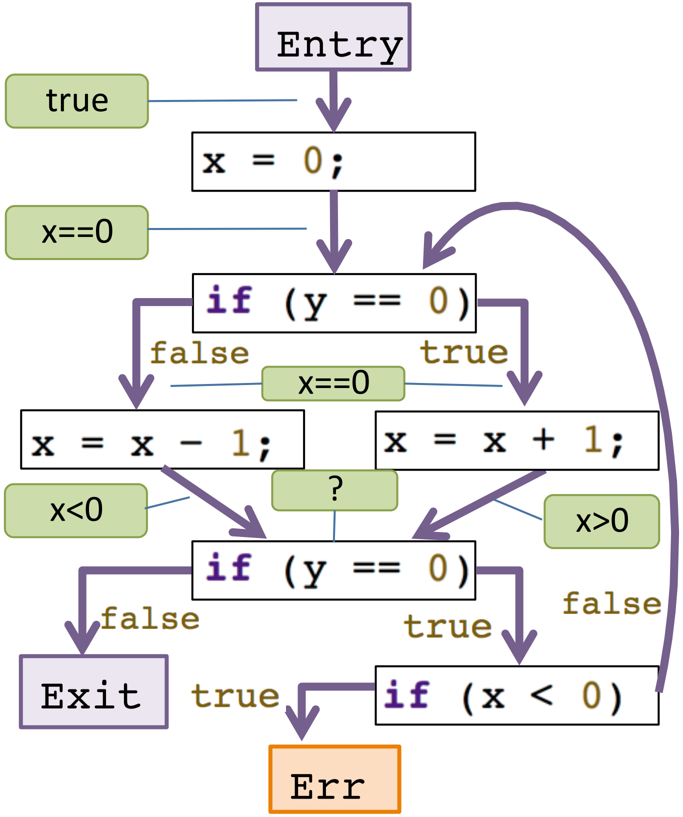


Analysis: update data about x based on control flow

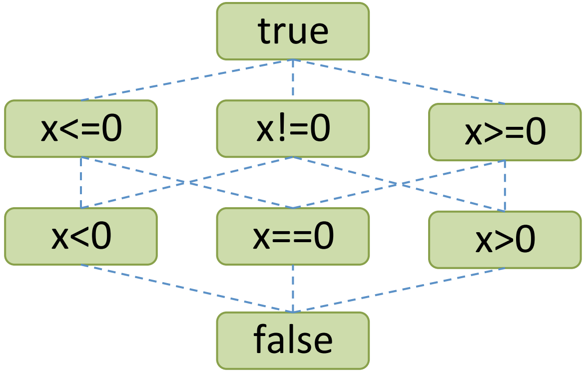


Dawn Song

Sign Analysis

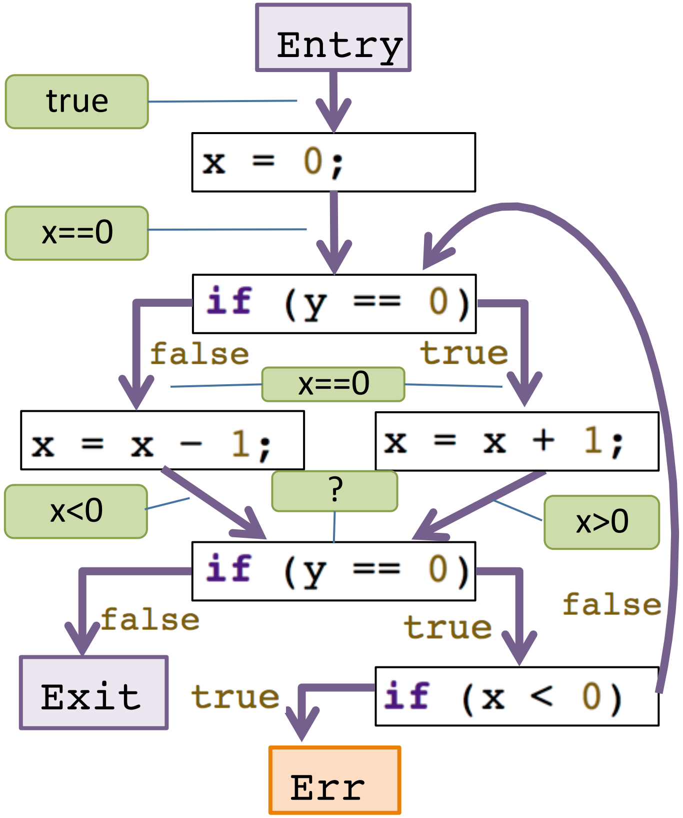


Analysis: update data about x based on control flow

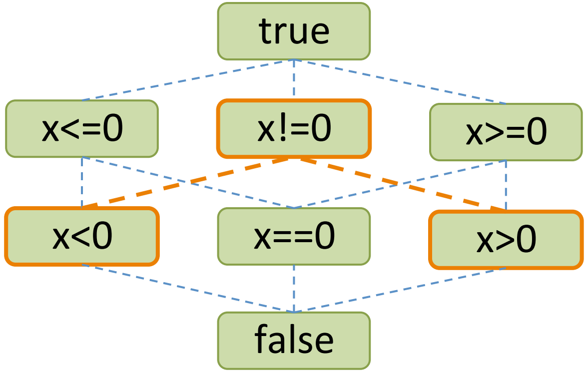


At the *join point* x is either strictly positive or strictly negative

Sign Analysis



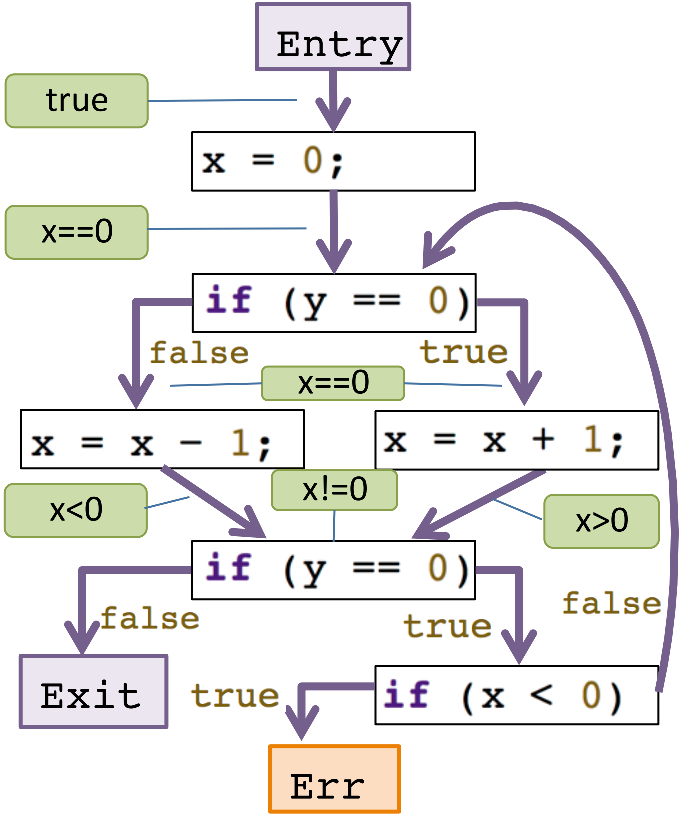
Analysis: update data about x based on control flow



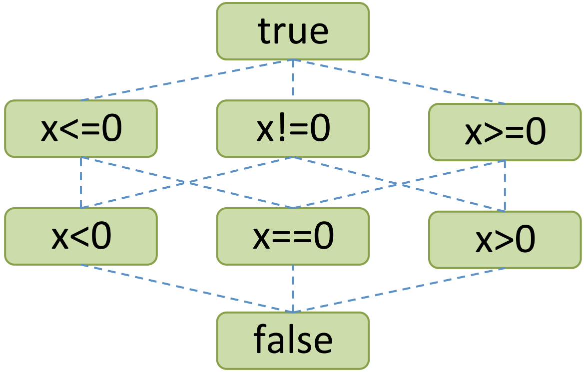
At the *join point* x is either strictly positive or strictly negative

Dawn Song

Sign Analysis



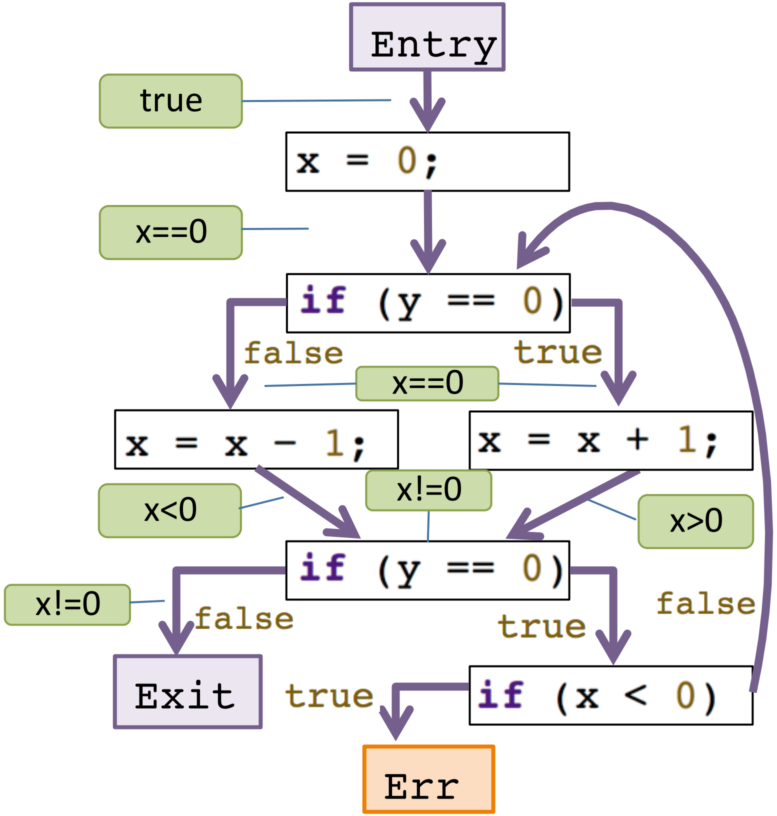
Analysis: update data about x based on control flow



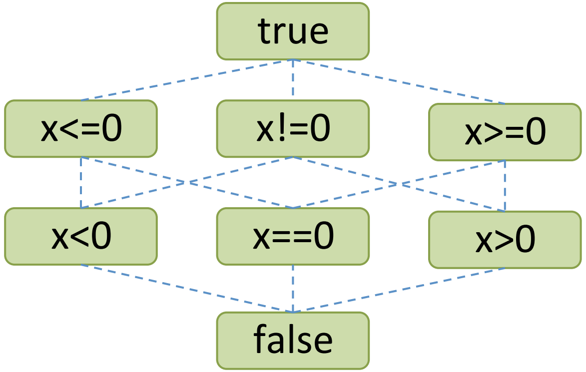
At the *join point* x is either strictly positive or strictly negative

Dawn Song

Sign Analysis

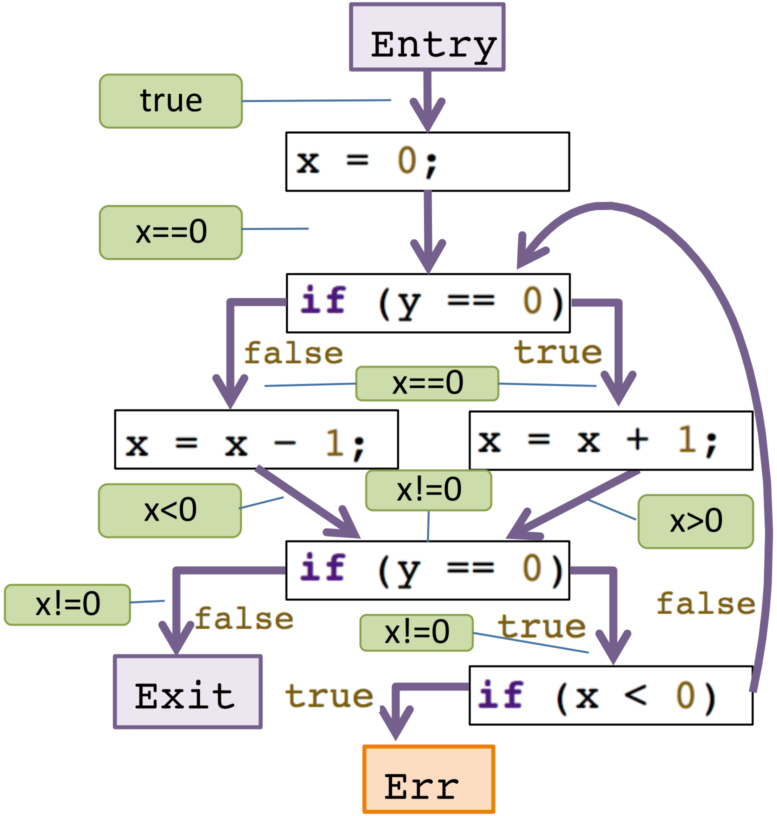


Analysis: update data about x based on control flow

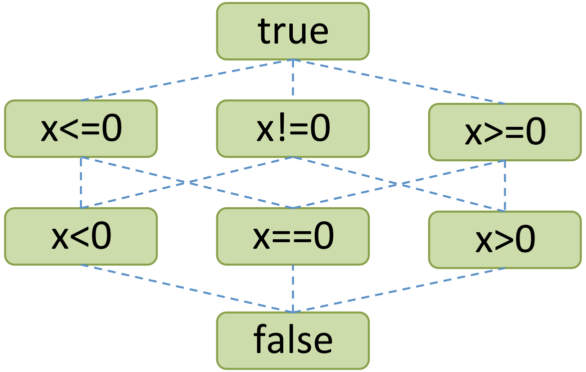


Dawn Song

Sign Analysis

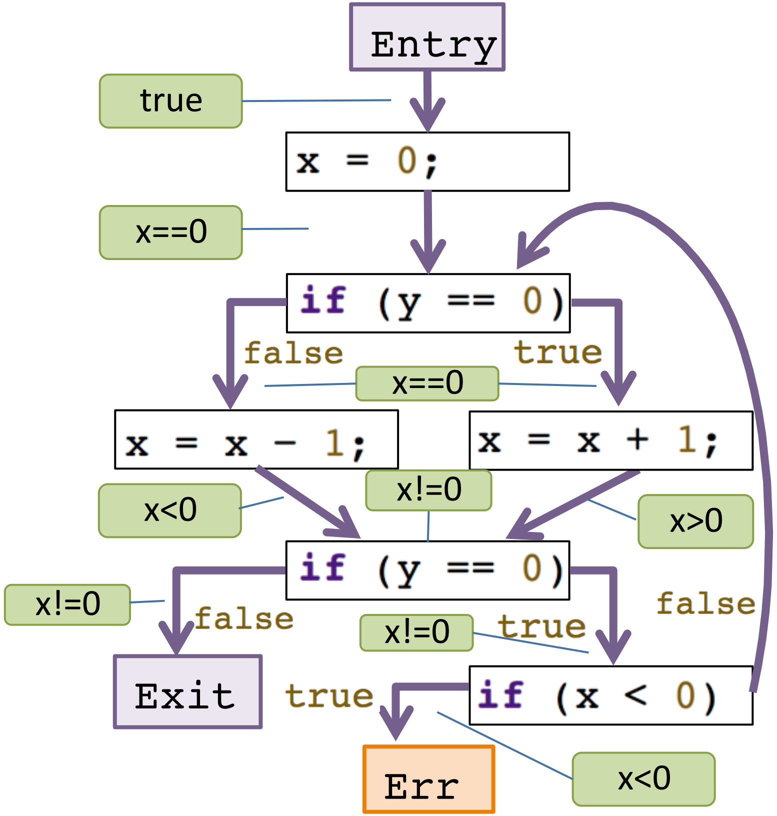


Analysis: update data about x based on control flow

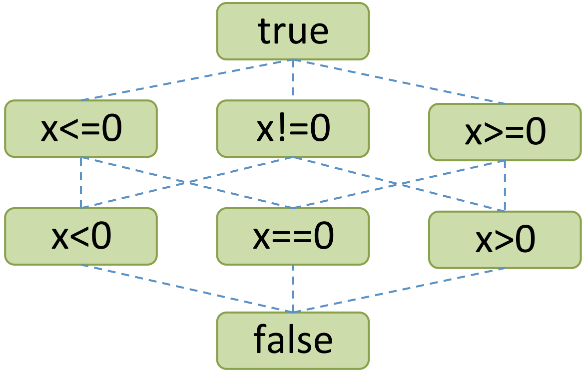


Dawn Song

Sign Analysis



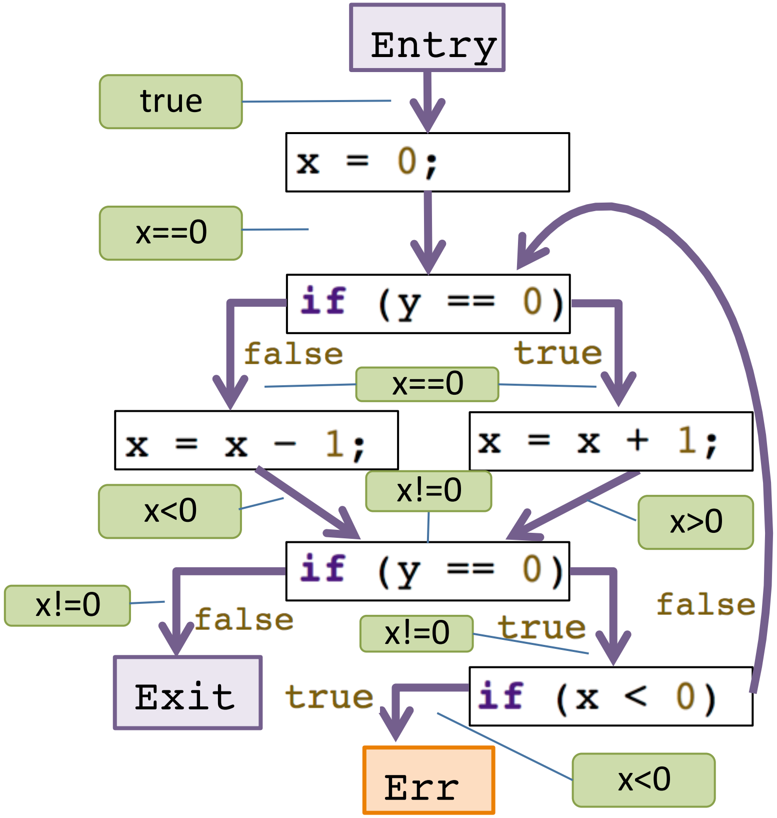
Analysis: update data about x based on control flow



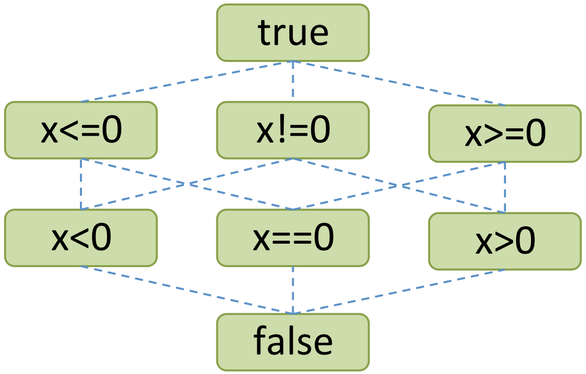
The conditional restricts x

Dawn Song

Sign Analysis

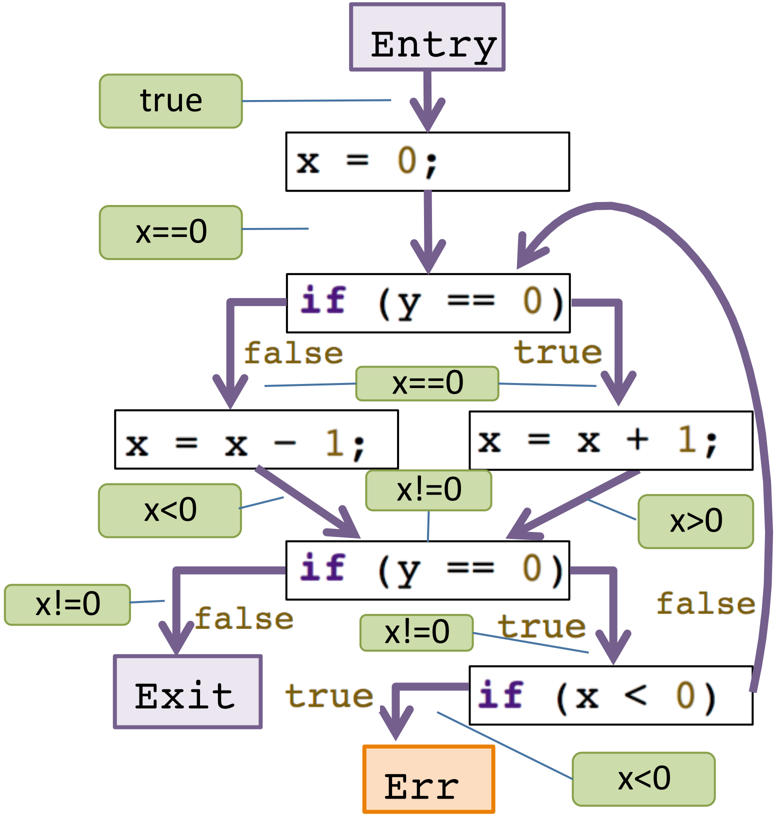


Analysis: update data about x based on control flow



The analysis concludes that it *may be possible* to reach Err with x<0

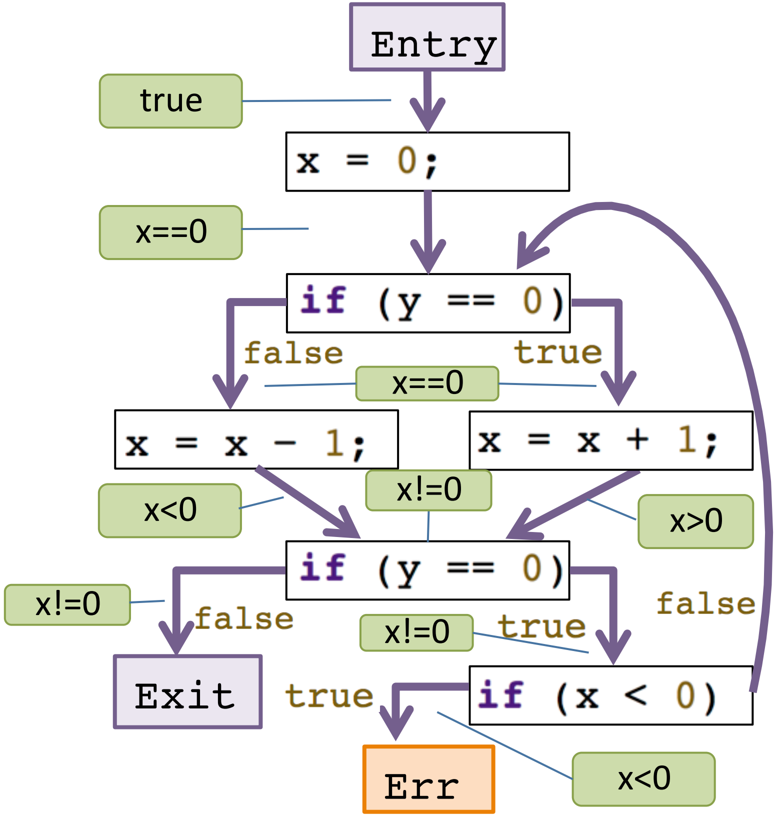
Sign Analysis vs. Symbolic Execution



Compare the sign analysis to symbolic execution

- Data was not precisely represented
- Some variables were ignored
- Control flow paths were *joined*
- It is not clear if there is an error
- It is not clear which path leads to the error

Sign Analysis vs. Symbolic Execution

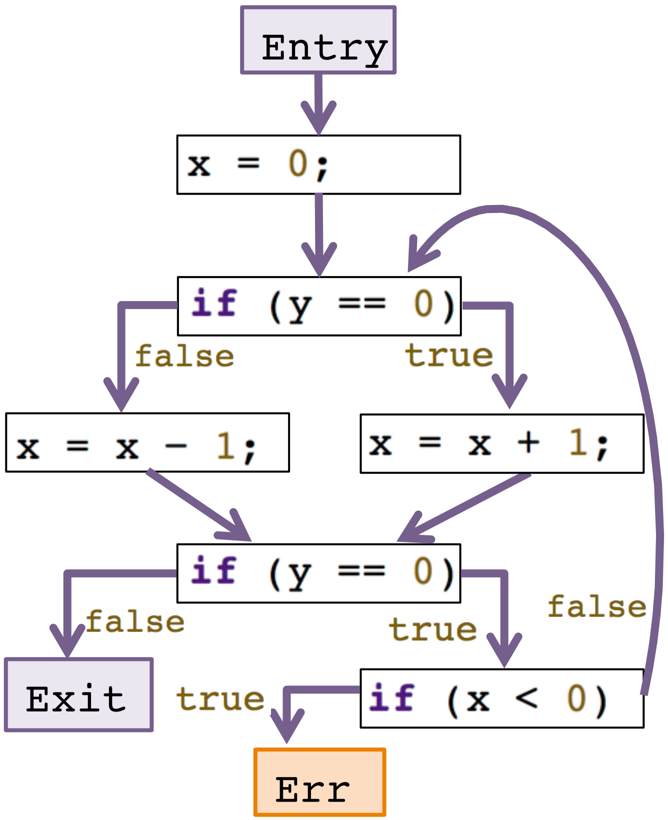


Compare the sign analysis to symbolic execution

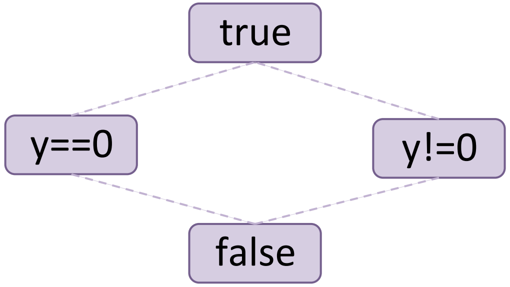
- Data was not precisely represented
- Some variables were ignored
- Control flow paths were *joined*
- It is not clear if there is an error
- It is not clear which path leads to the error

Problem: no information about y

Zero Propagation

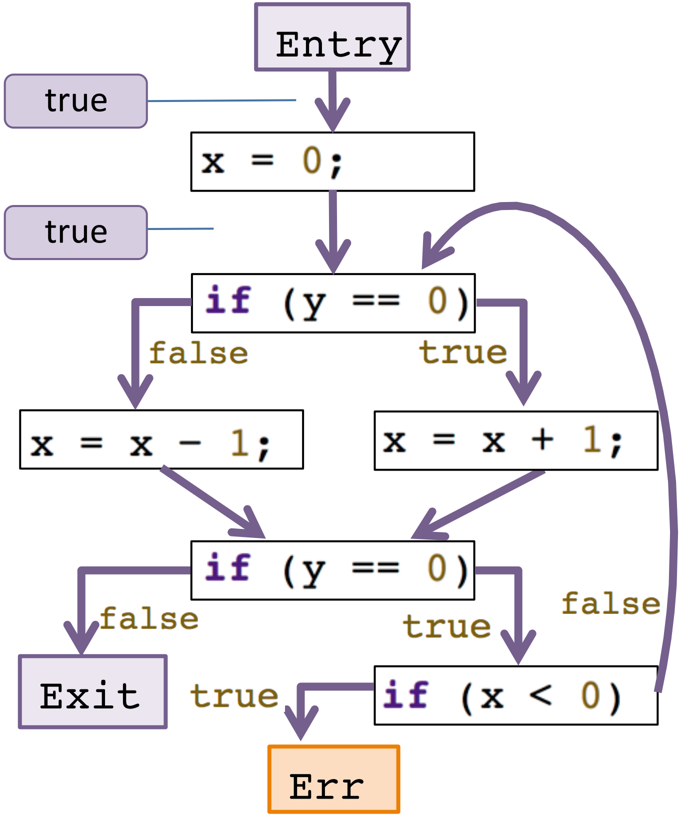


Suppose we only track if y is zero or not

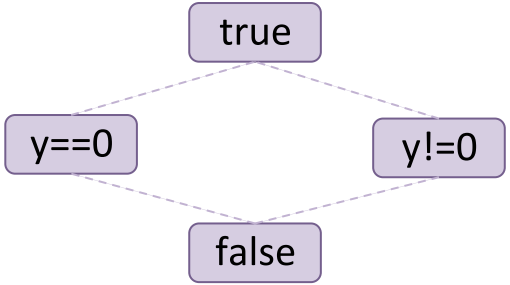


Dawn Song

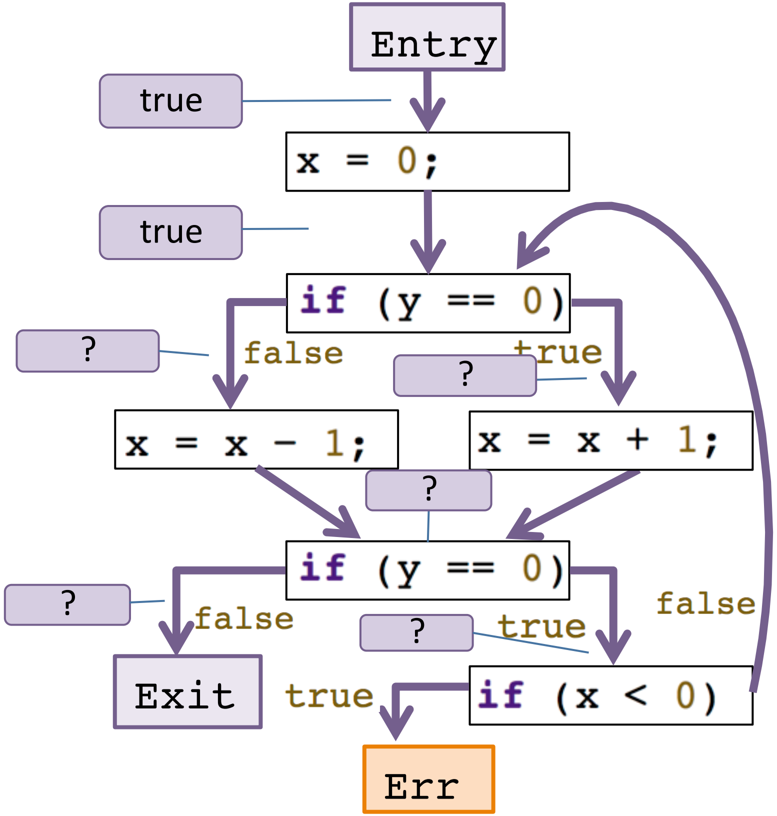
Zero Propagation



Suppose we only track if y is zero or not

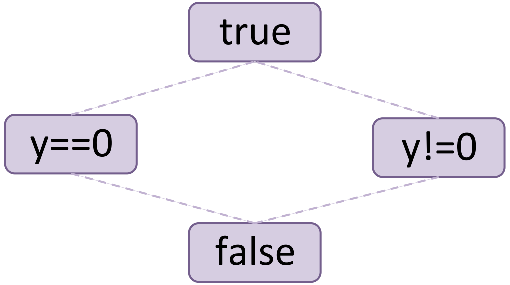


Quiz: Zero Propagation

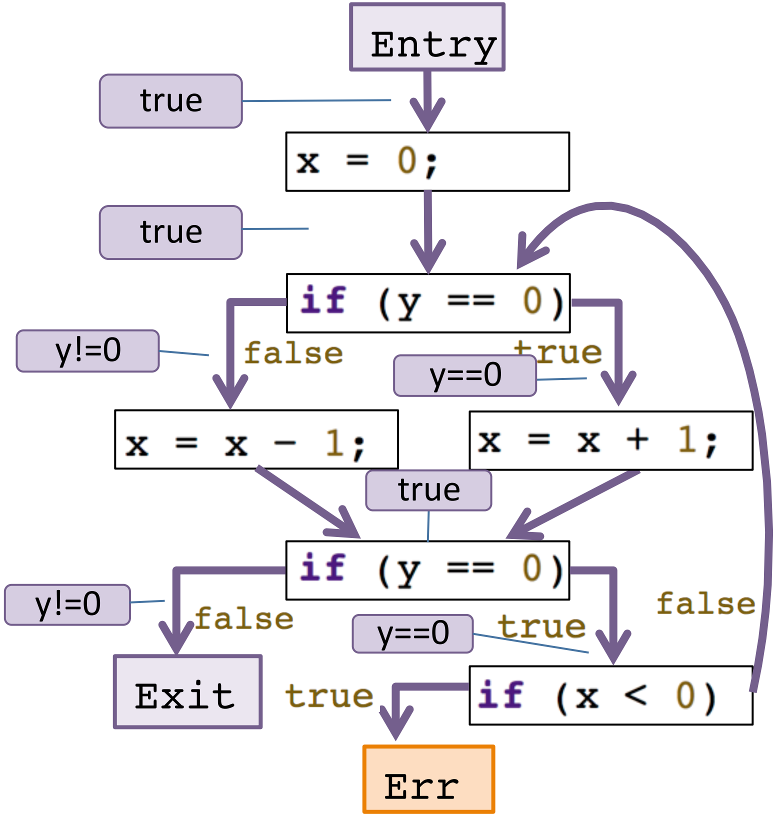


Suppose we only track if y is zero or not

Can you fill in the blanks for the first steps of the analysis?

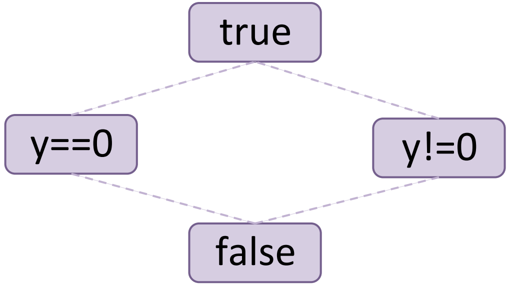


Zero Propagation

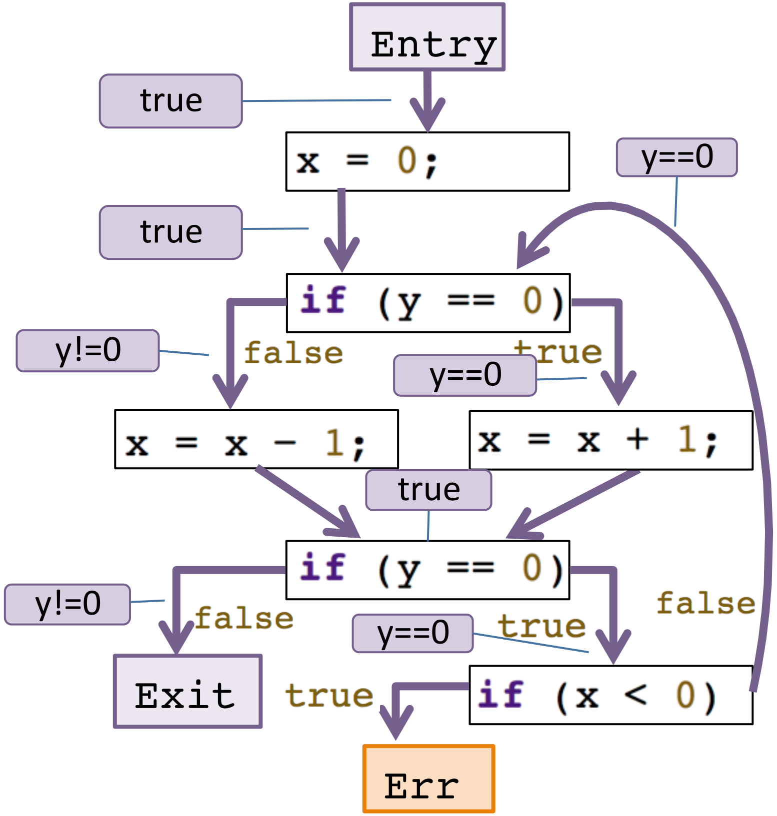


Suppose we only track if y is zero or not

Can you fill in the blanks for the first steps of the analysis?

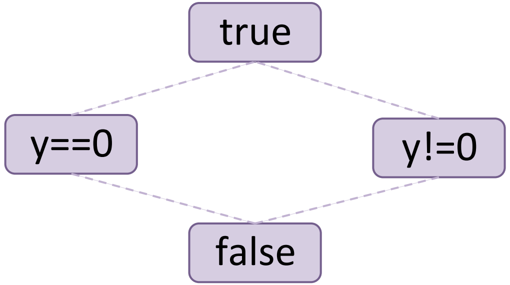


Zero Propagation



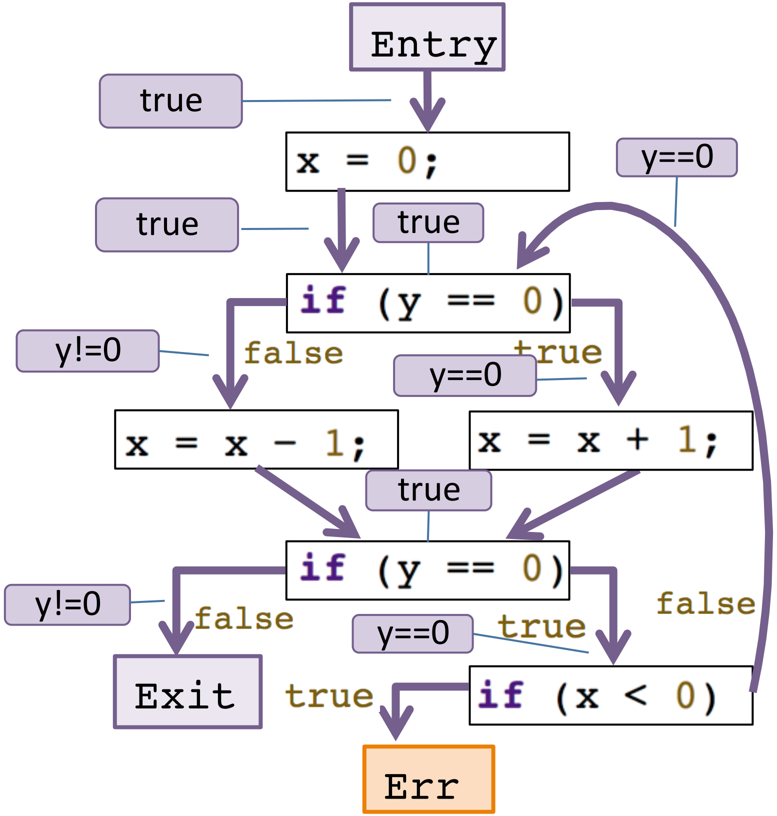
Suppose we only track if y is zero or not

Can you fill in the blanks for the first steps of the analysis?



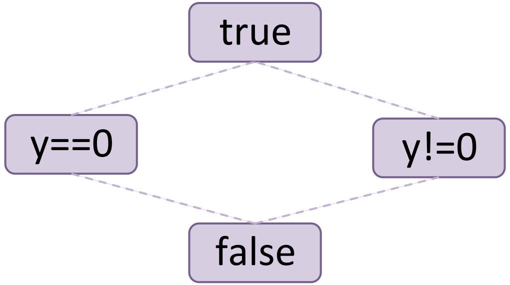
A loop head is also a join-point

Zero Propagation



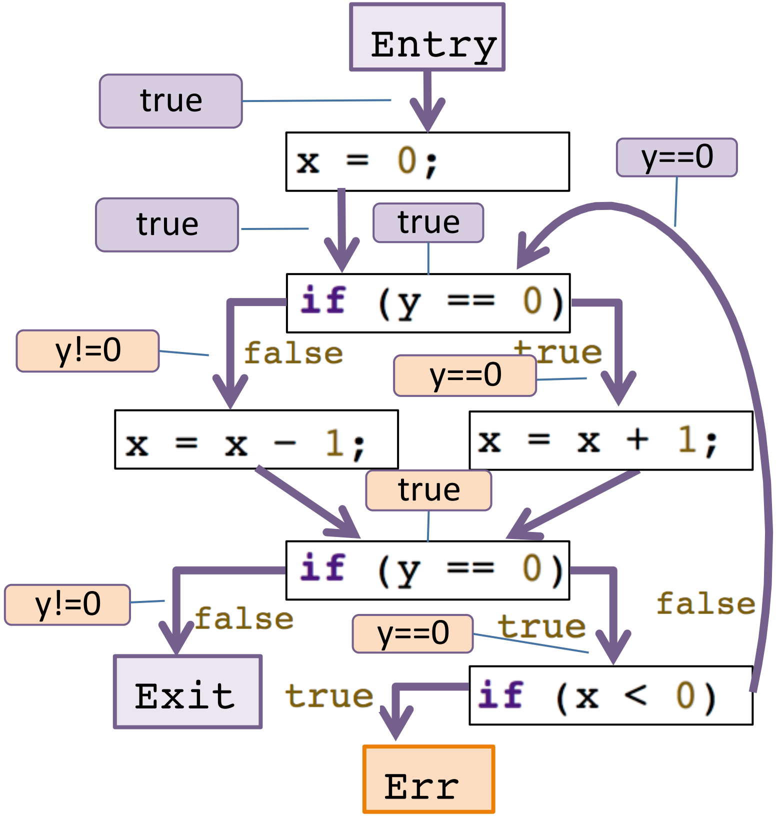
Suppose we only track if y is zero or not

Can you fill in the blanks for the first steps of the analysis?



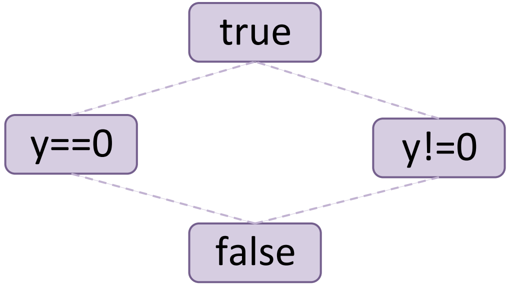
A loop head is also a join-point

Zero Propagation



Suppose we only track if y is zero or not

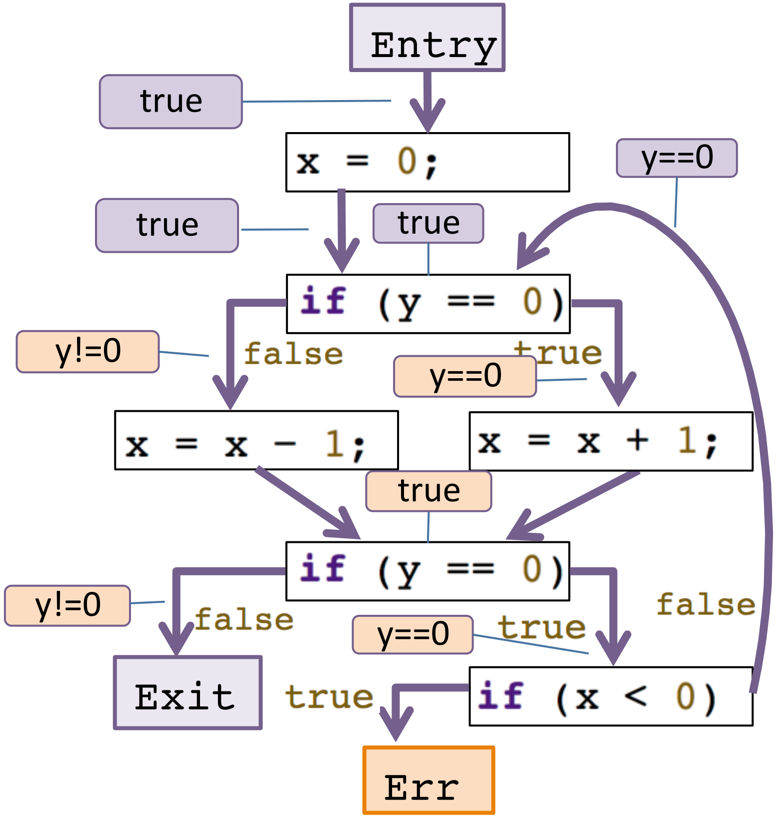
Can you fill in the blanks for the first steps of the analysis?



Since the loop head was updated, what follows may change.

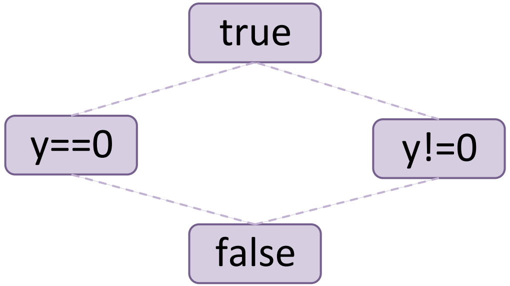
Dawn Song

Zero Propagation



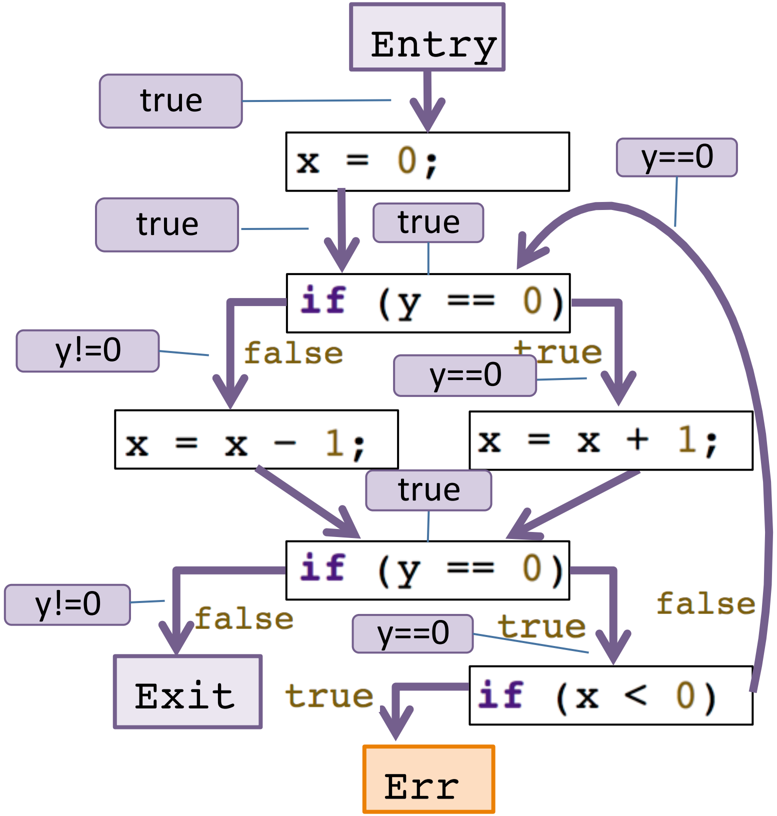
Suppose we only track if y is zero or not

Can you fill in the blanks for the first steps of the analysis?



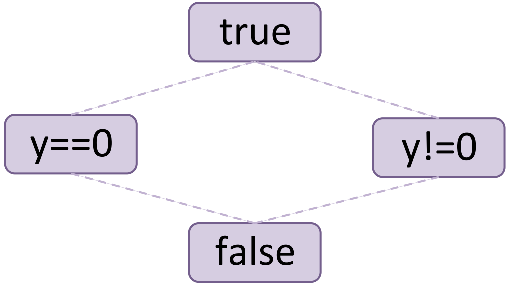
Since the loop head was updated, what follows may change. In this case, the update does not change the result of the analysis.

Zero Propagation



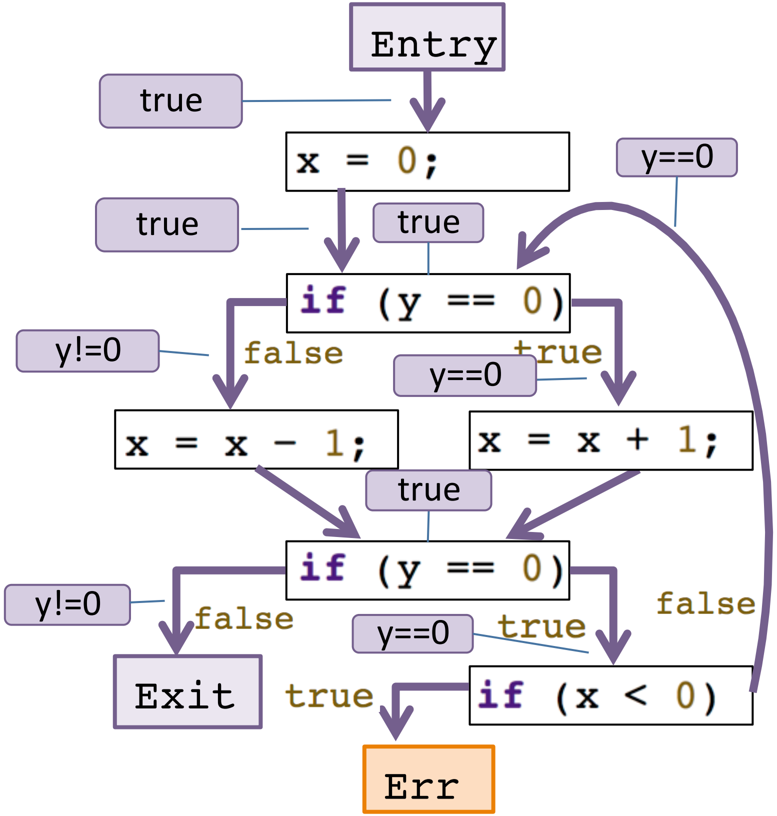
Suppose we only track if y is zero or not

Can you fill in the blanks for the first steps of the analysis?



When propagation does not change the results, a *fixed point* is reached.

Sign Analysis vs. Zero Propagation

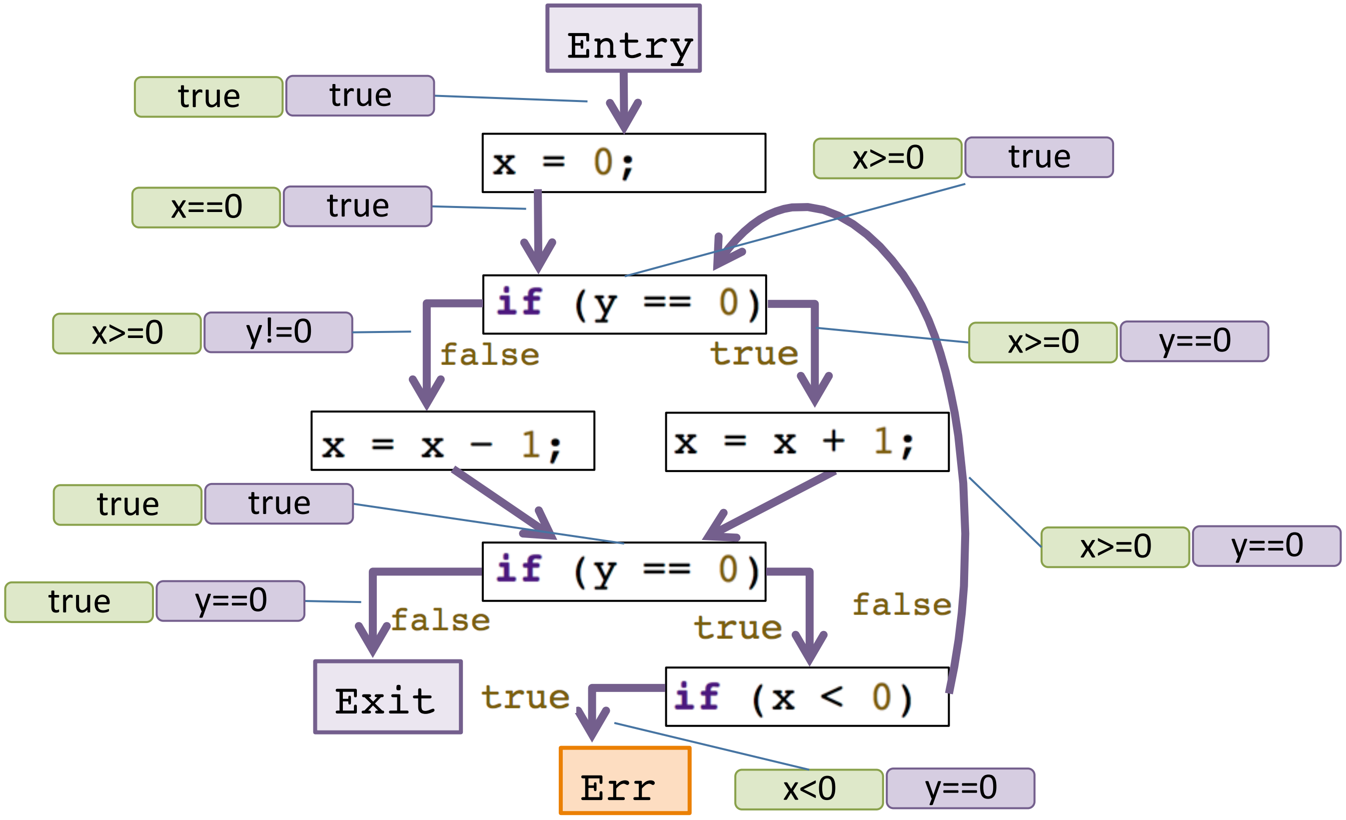


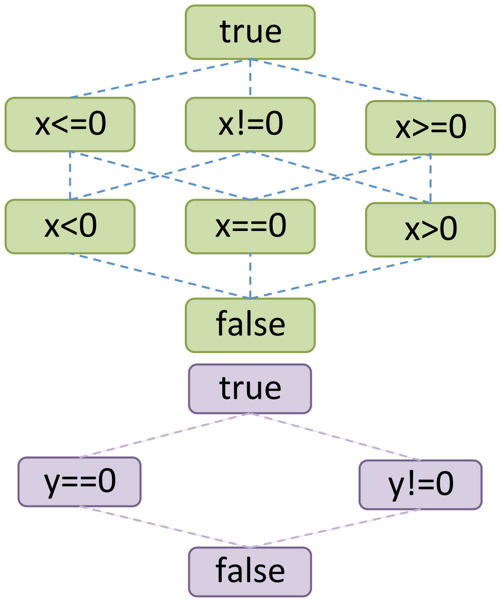
Sign analysis and zero propagation both report that the error may be reached.

Each analysis ignores one variable.

Can we do better by tracking both variables at the same time?

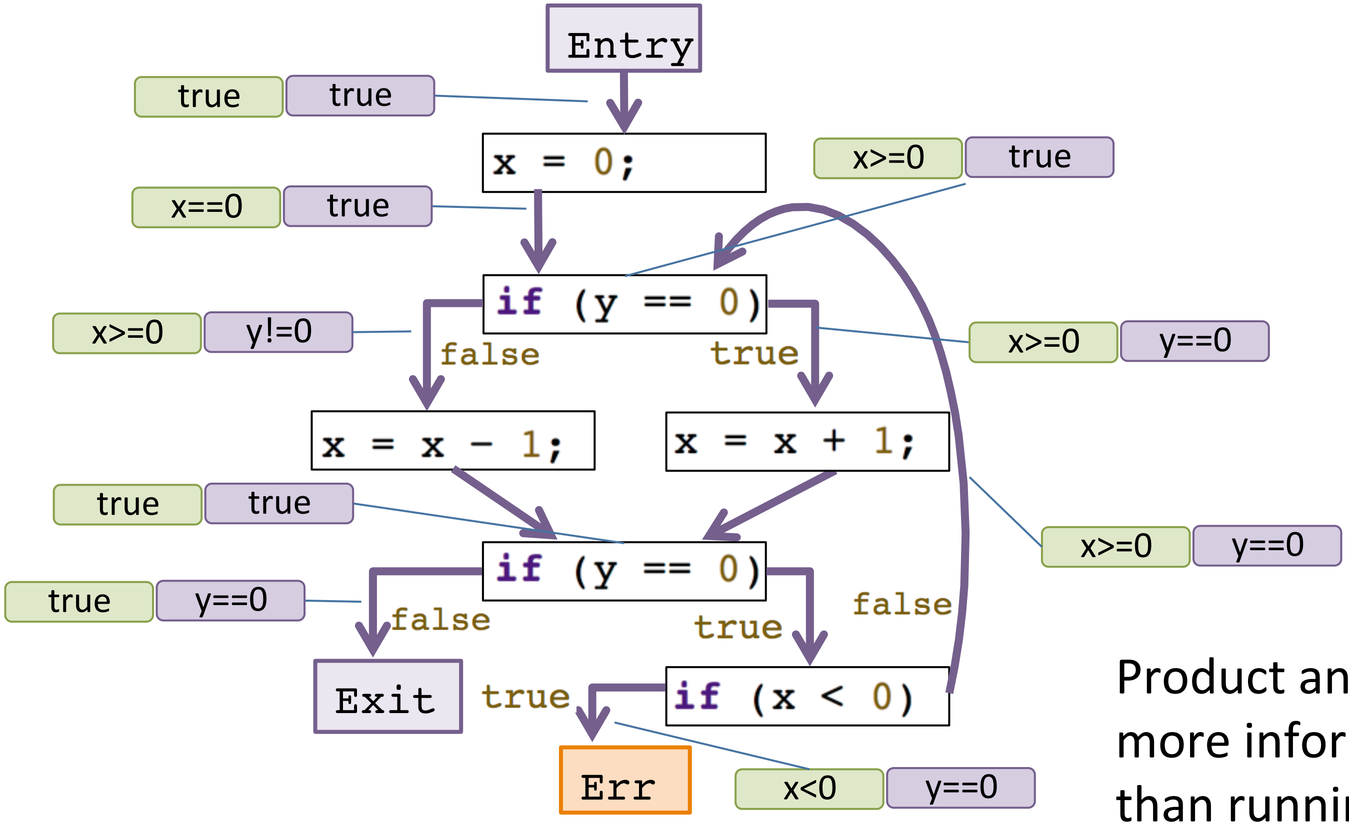
A Product Analysis

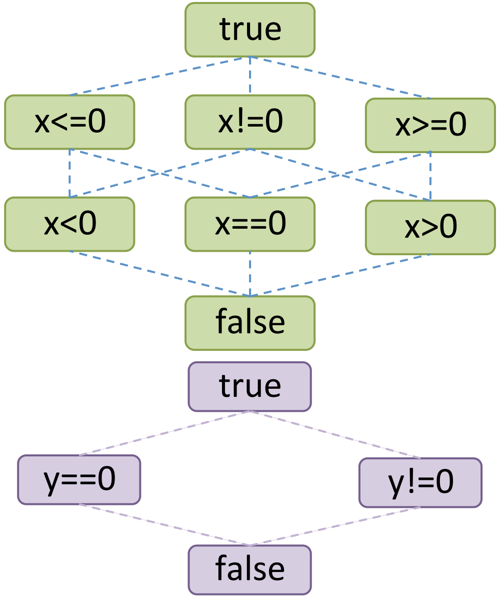




Dawn Song

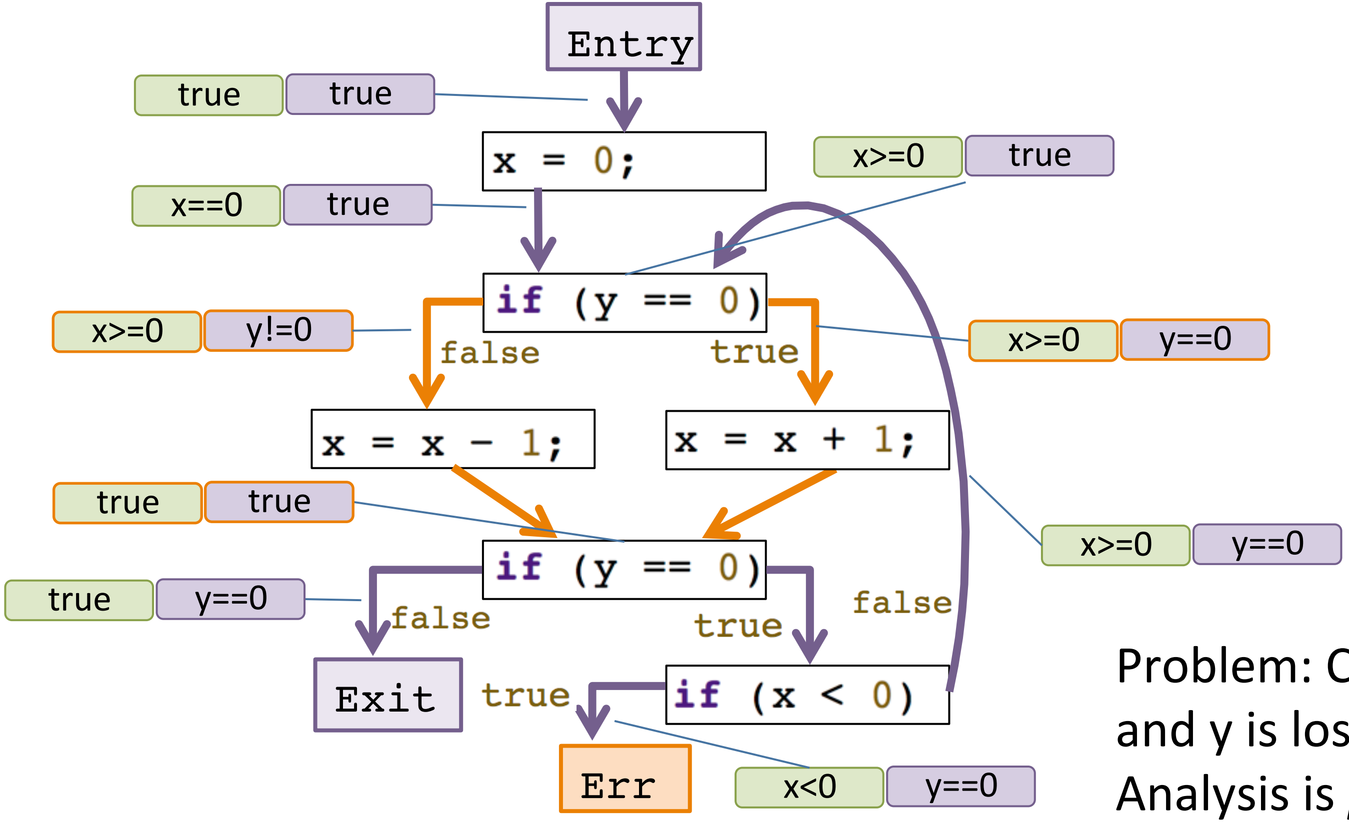
A Product Analysis

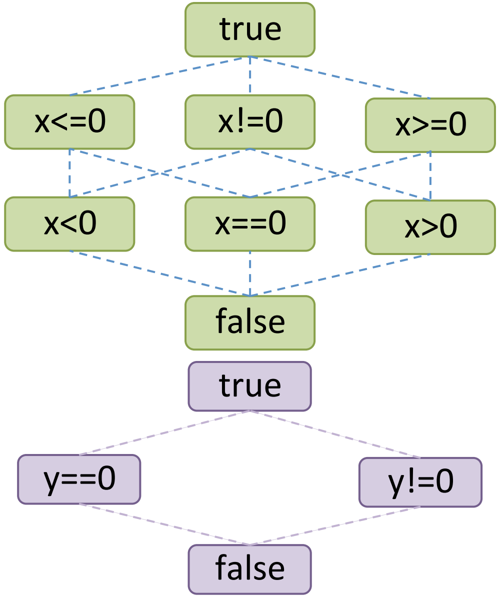




Product analysis does not yield more information (in this case) than running both separately

A Product Analysis

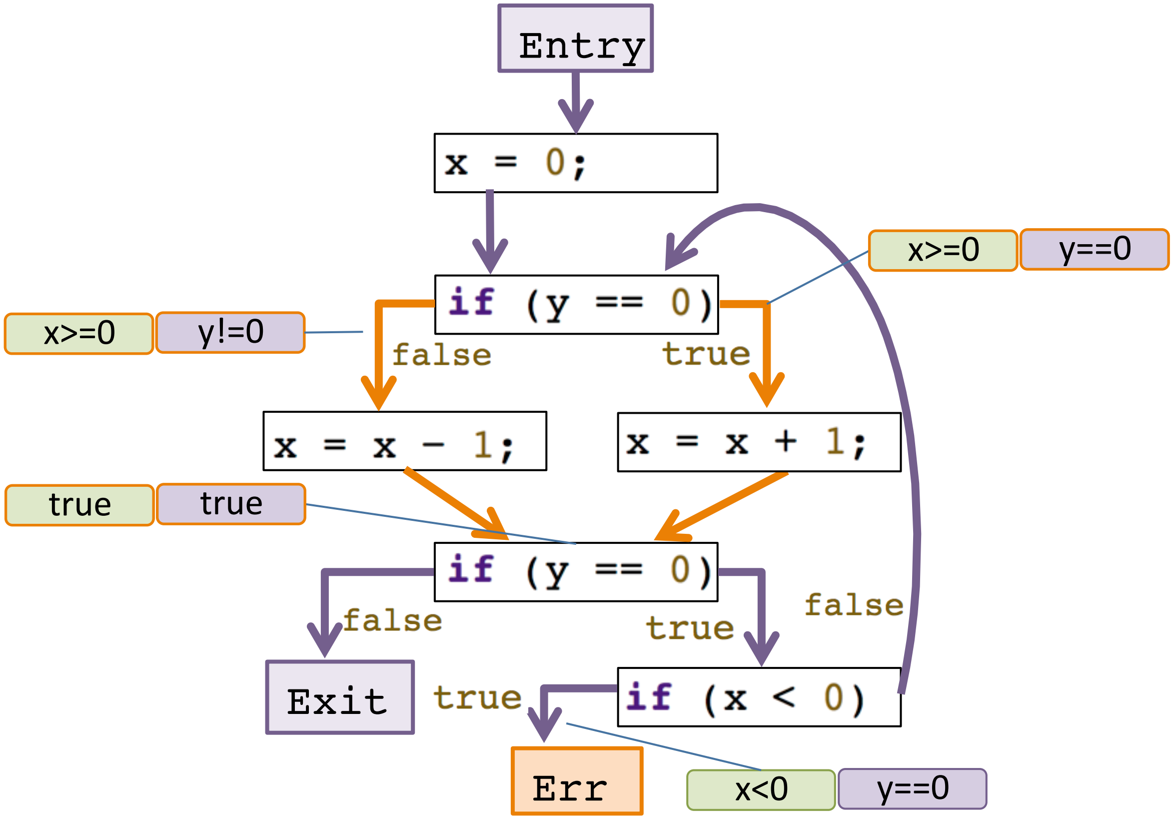




Problem: Correlation between x and y is lost at this joint point. Analysis is *path insensitive.*

Dawn Song

Disjunctive Refinement



Disjunctive refinement allows disjunctions of facts

x>=0	y!=0

join

x<0	y==0

=

x>=0	y!=0
x<0	y==0

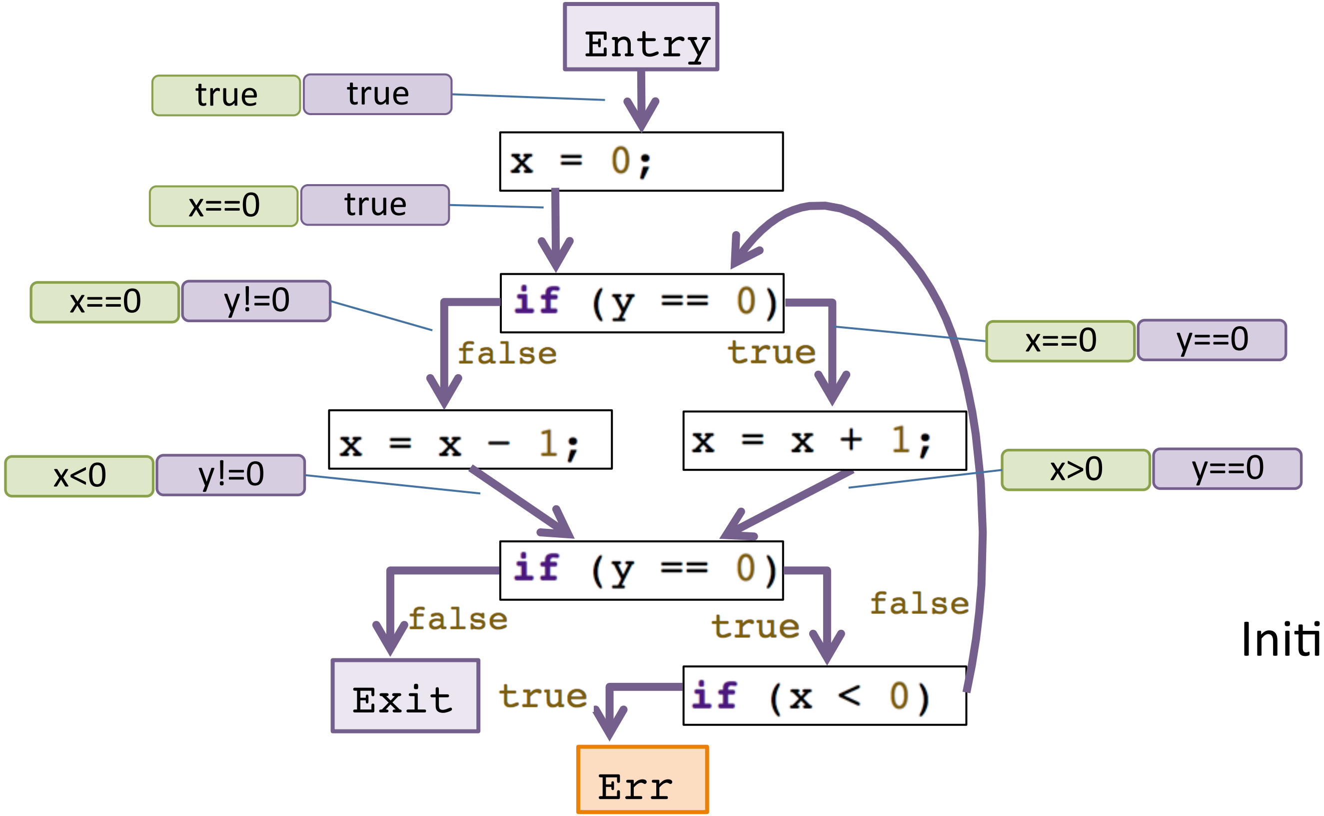
or

instead of

true	true

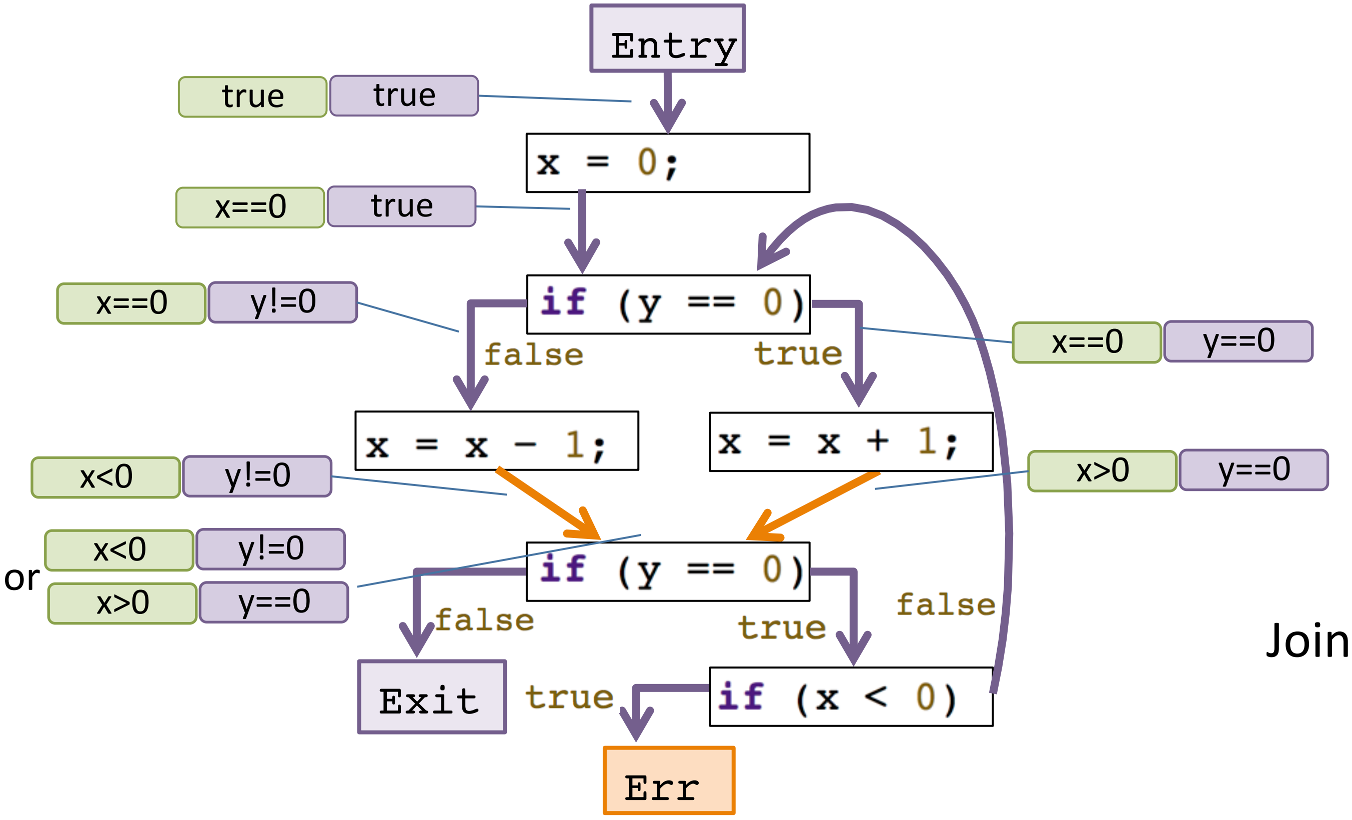
Dawn Song

Analysis with Disjunctive Refinement



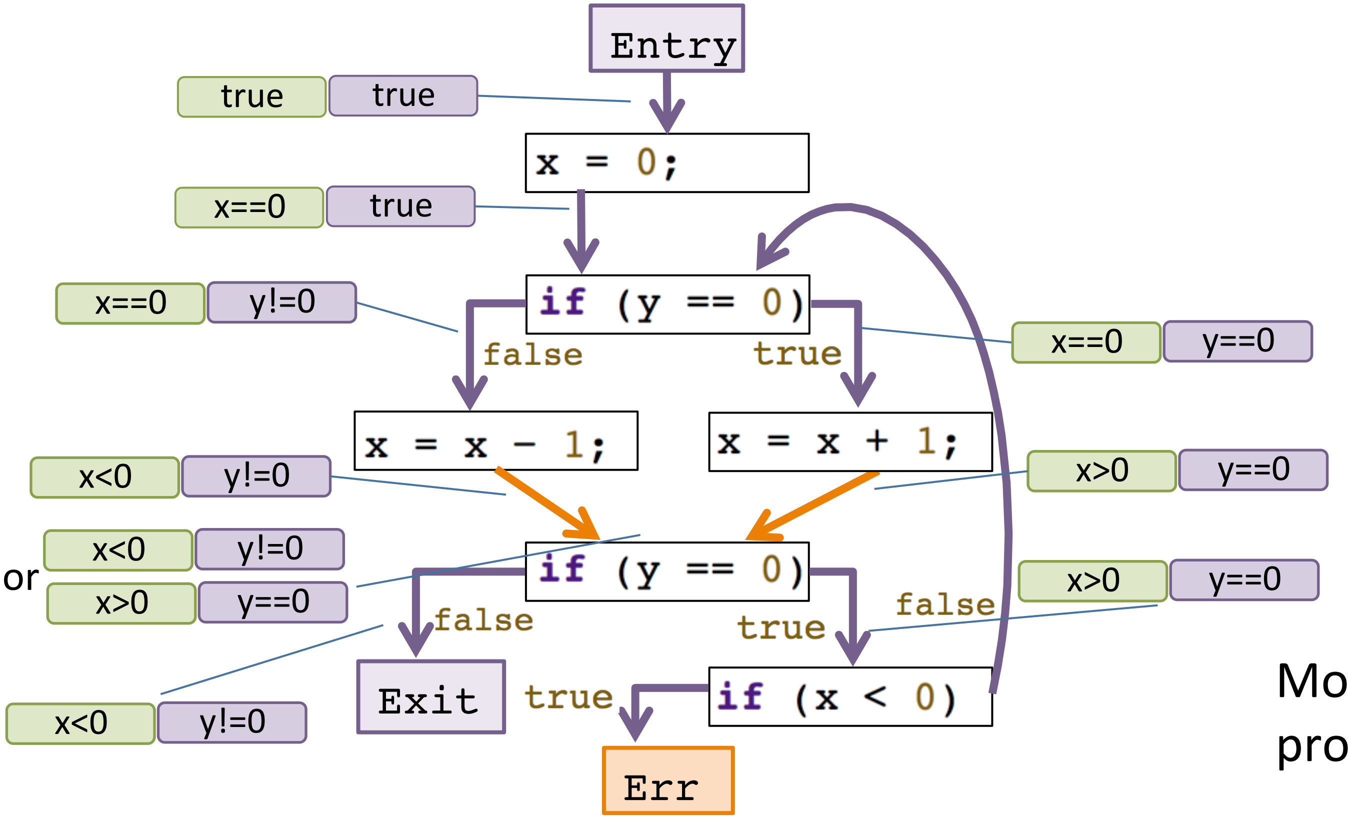
Initial steps are the same

Analysis with Disjunctive Refinement



Join is now disjunction

Analysis with Disjunctive Refinement



Entry

true true

`x = 0;`

x==0 true

`if (y == 0)`

x==0 y!=0

false true

x==0 y==0

`x = x - 1;`

`x = x + 1;`

x<0 y!=0

x>0 y==0

or

x<0 y!=0

x>0 y==0

`if (y == 0)`

false true false

x>0 y==0

Exit

true

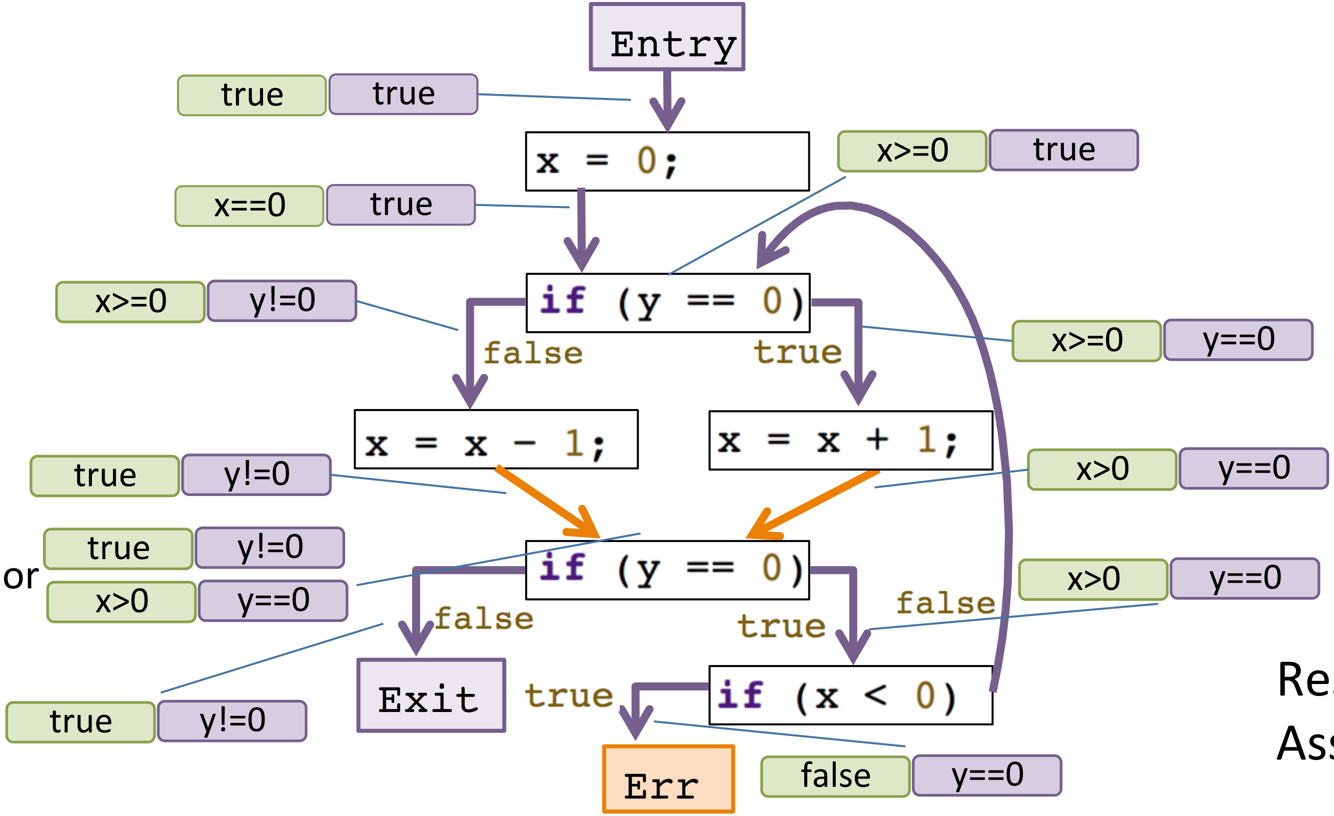
`if (x < 0)`

x<0 y!=0

Err

More precise information propagated after join

Analysis with Disjunctive Refinement



Result of final analysis.
Assertion is not violated.

Dawn Song

1	Analysis Frameworks
2	Types of Analyses
3	Precision
4	Summary of Program Analysis

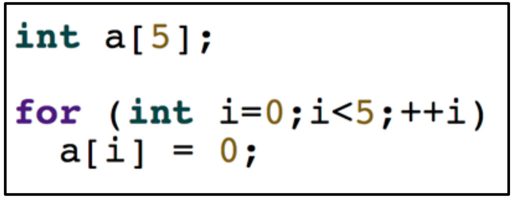
1	Analysis Frameworks

a	Lattices
b	Transformers
c	Systems of Equations
d	Solving Equations

1	Analysis Frameworks

a	Lattices
b	Transformers
c	Systems of Equations
d	Solving Equations

States, Transitions, Executions



```
int a[5];

for (int i=0;i<5;++i)
  a[i] = 0;
```

States, Transitions, Executions

```
int a[5];

for (int i=0;i<5;++i)
  a[i] = 0;
```

States	values of local and global variables, program counter, stack, heap

pc	d
i	1
a[0]	0
a[1]	undef
a[2]	undef
a[3]	undef
a[4]	undef

States, Transitions, Executions

```
int a[5];

for (int i=0;i<5;++i)
  a[i] = 0;
```

States	values of local and global variables, program counter, stack, heap

pc	d
i	1
a[0]	0
a[1]	undef
a[2]	undef
a[3]	undef
a[4]	undef

States, Transitions, Executions

```
int a[5];

for (int i=0;i<5;++i)
  a[i] = 0;
```

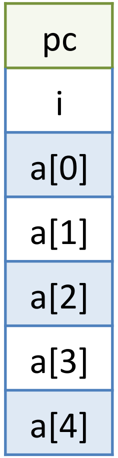
States	values of local and global variables, program counter, stack, heap
Transitions	state changes

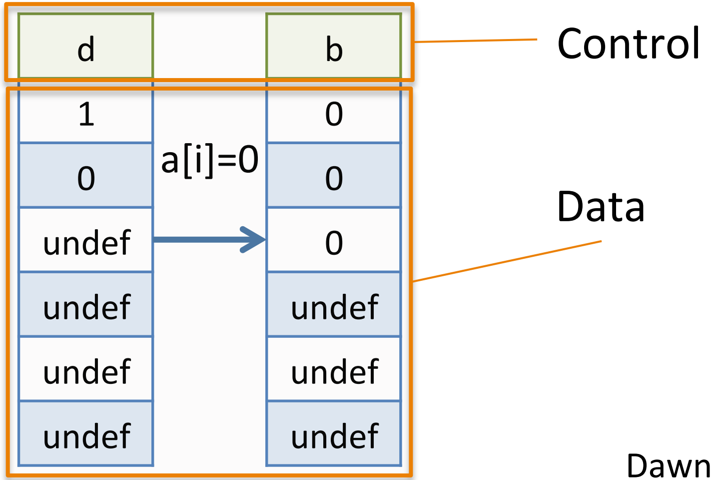
pc	d	b
i	1	0
a[0]	0	0
a[1]	undef	0
a[2]	undef	undef
a[3]	undef	undef
a[4]	undef	undef

States, Transitions, Executions

```
int a[5];

for (int i=0;i<5;++i)
  a[i] = 0;
```

States	values of local and global variables, program counter, stack, heap
Transitions	state changes

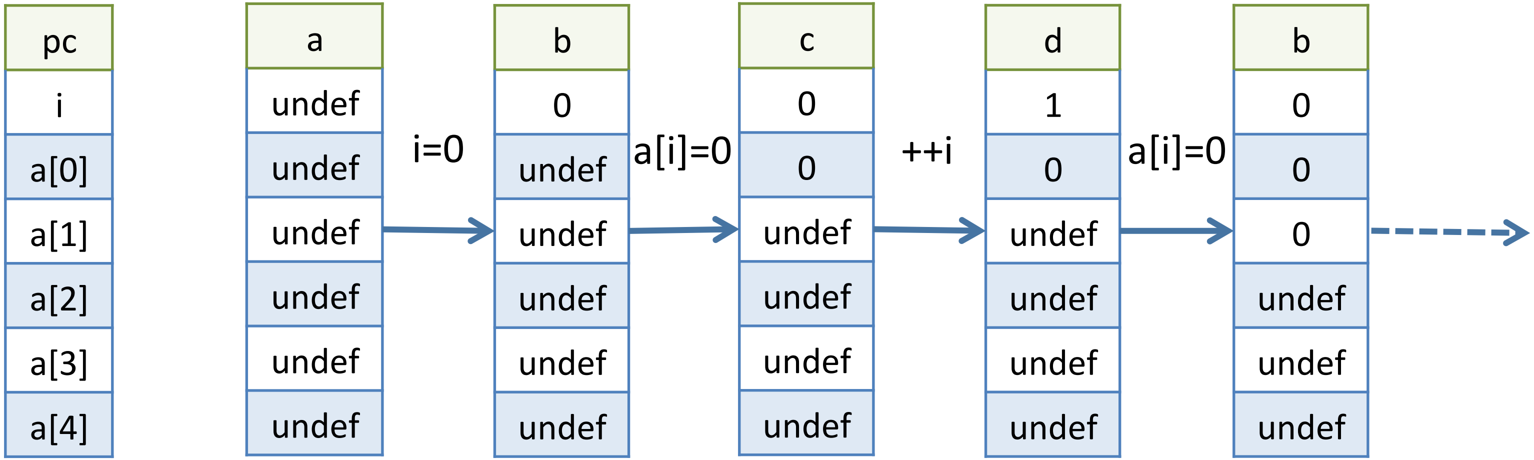




States, Transitions, Executions

```
int a[5];

for (int i=0;i<5;++i)
  a[i] = 0;
```

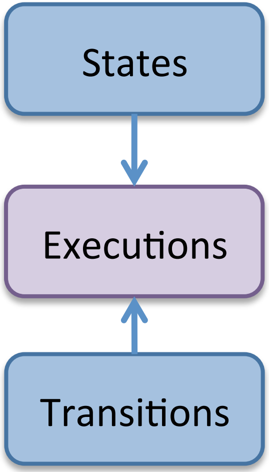
States	values of local and global variables, program counter, stack, heap
Transitions	state changes
Executions	Sequence of state changes



pc	a	b	c	d	b
i	undef	0	0	1	0
a[0]	undef	undef	0	0	0
a[1]	undef	undef	undef	undef	0
a[2]	undef	undef	undef	undef	undef
a[3]	undef	undef	undef	undef	undef
a[4]	undef	undef	undef	undef	undef

Control and Data in Programs

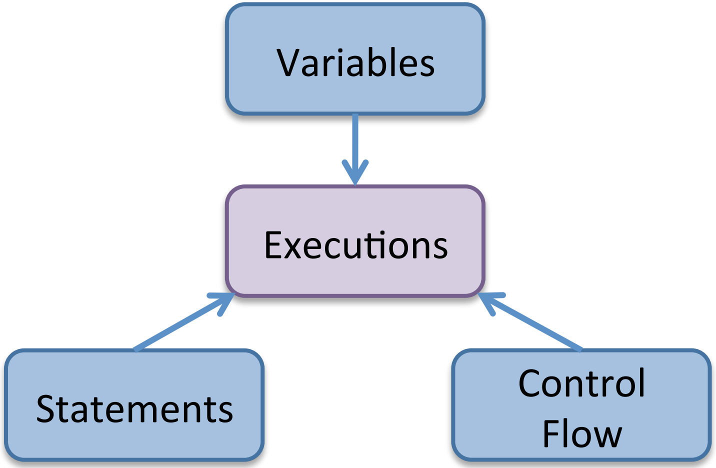
Variables	have values, define state
Statements	modify values, define transitions on data
Control flow	modify program counter, define control transitions



States

Executions

Transitions



Variables

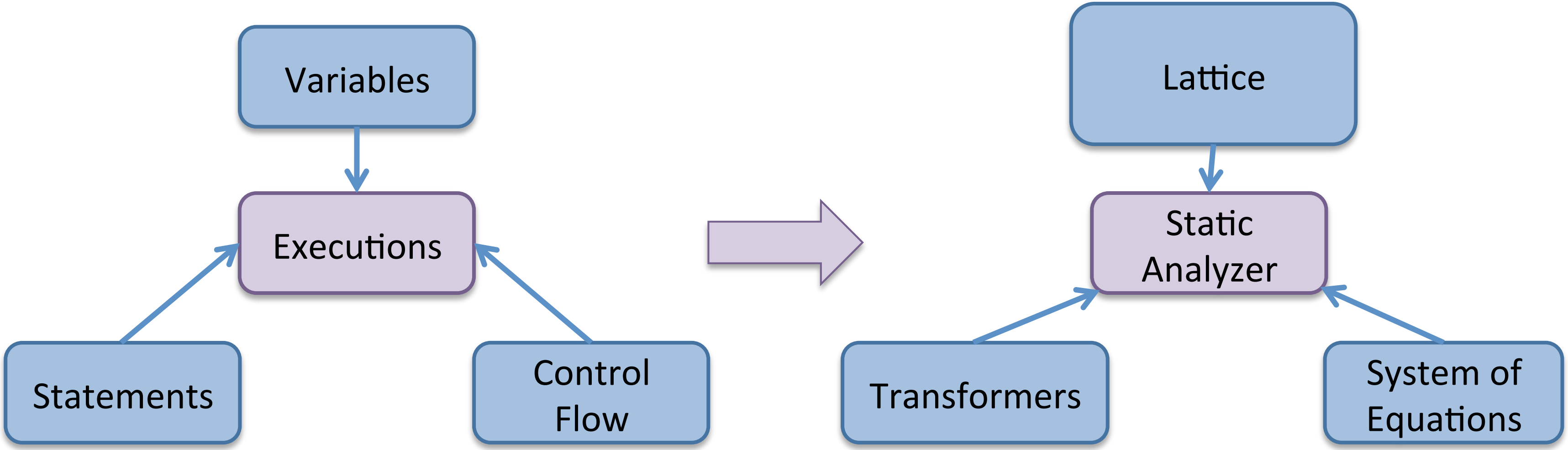
Executions

Statements

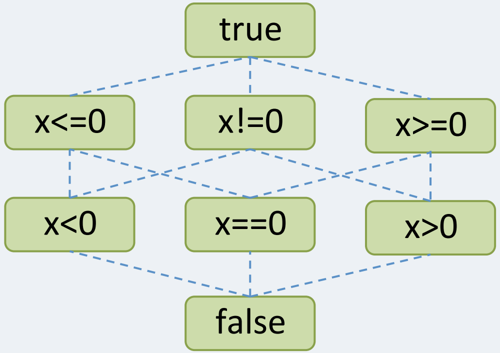
Control Flow

Architecture of a Static Analyzer

The behavior of a program can be approximated by separately approximating variable values, statements and control flow.

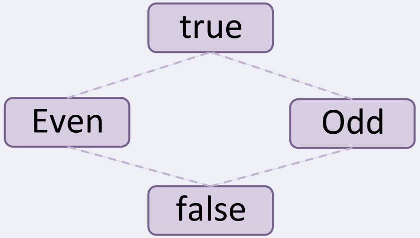


Lattices in Static Analysis



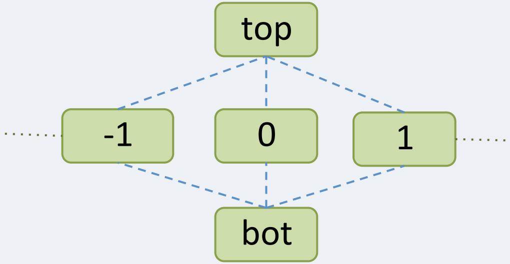
Signs

- positive/negative/zero
- cannot represent non-zero values
- no relationships between variables



Parity

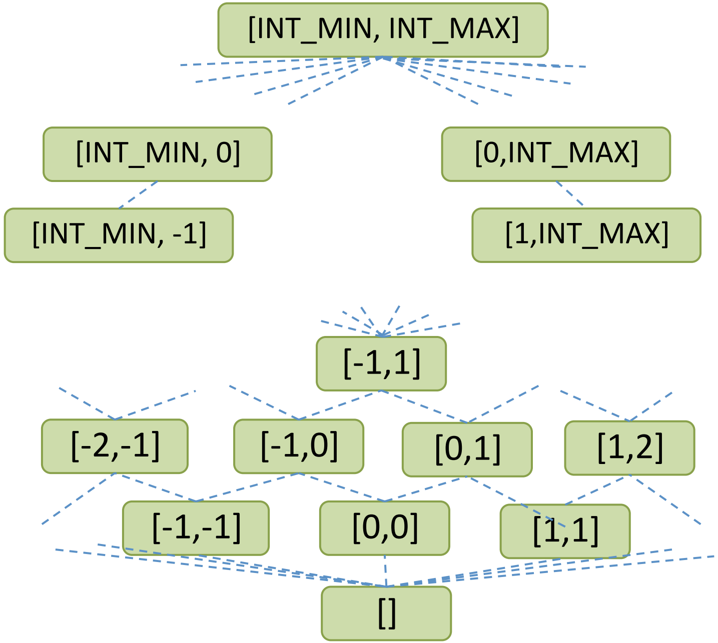
- even or odd
- cannot represent values
- no relationships between variables



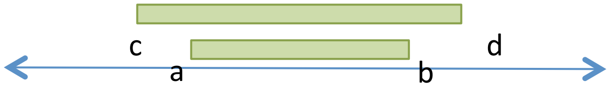
Constants

- a single value
- cannot represent more values: x==3||x==4
- no relationships between variables

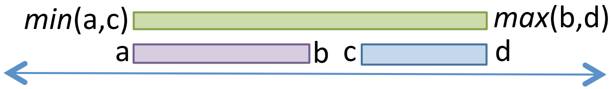
The Interval Lattice



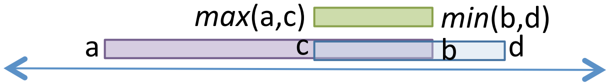
An *interval* is a pair [a,b] with a <= b



There is a *partial order* between intervals

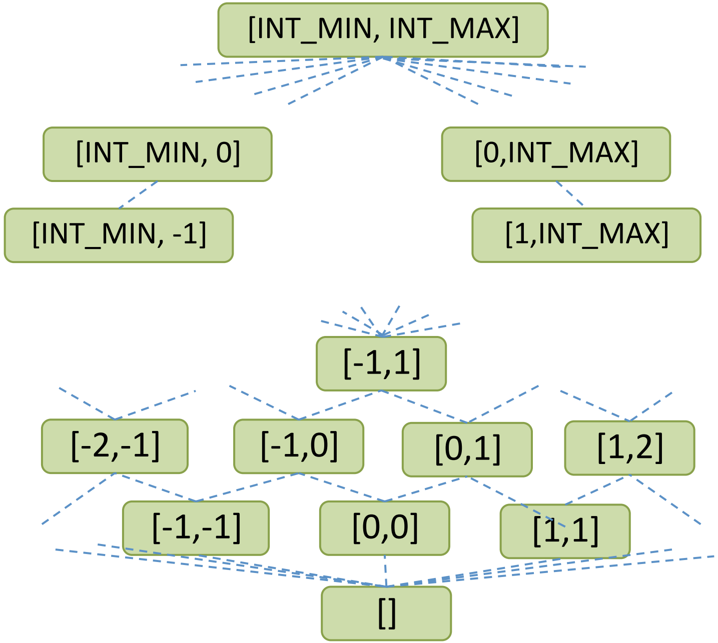


The *join* is the smallest enclosing interval



The *meet* is the largest shared interval

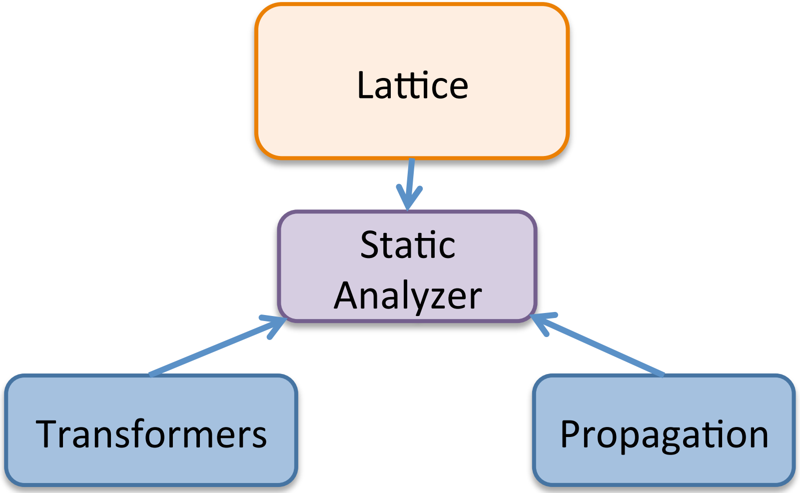
Loss of Information in the Interval Lattice



Intervals are useful for tracking the range of variables. They lose information about concrete values.

Arbitrary sets	{1,5}, {1,3,5} {1,2,4,5} are represented by [1,5]
Union	[1,3] join [6,7] = [1,7] includes values 4 and 5
Relations	x=y can only be written as x:[INT_MIN,INT_MAX], y:[INT_MIN,INT_MAX]

Lattice in a Static Analyzer



A lattice is a set with

- a *partial order* for comparing elements
- a least upper bound called *join*
- a greatest lower bound called *meet*

In static analysis

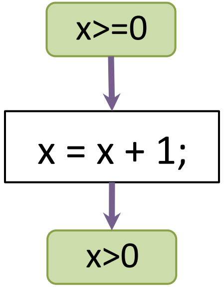
- lattice elements abstract states
- order is used to check if results change
- meet and join are used at branch and join points

Most analyses use only meet or only join

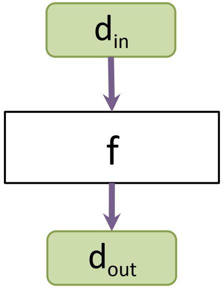
1	Analysis Frameworks

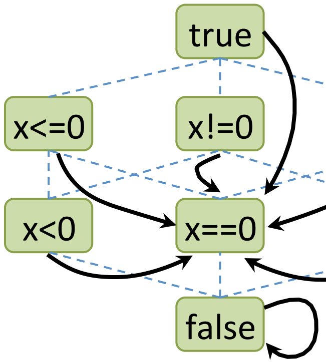
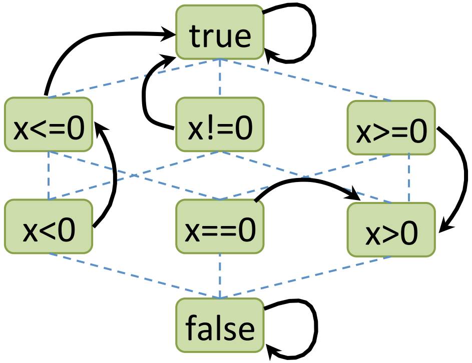
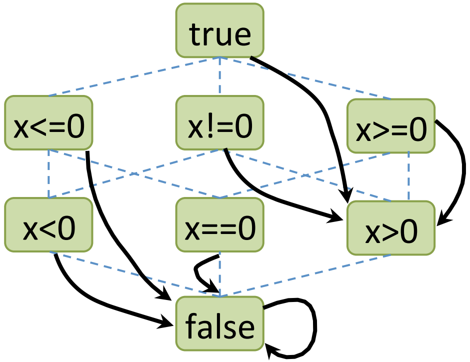
a	Lattices
b	Transformers
c	Systems of Equations
d	Solving Equations

Sign Analysis Transformers

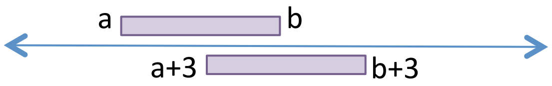
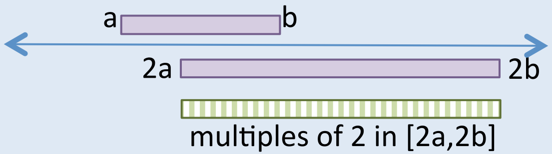
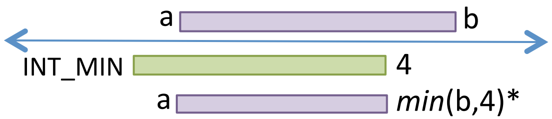
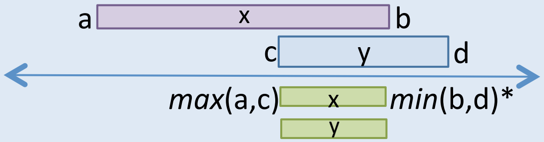


A *transformer* (or *transfer function*) describes how a statement modifies lattice elements



`x = 0;`	`x = x+1;`	`if (x > 4)`
		

Interval Analysis Transformers

Statement	Transformer	Loss of Precision
`x = x+3`		No loss of precision
`x=2*x`		[3,4] is transformed to [6,8] and includes 7, which is not a multiple of 2
`if (x<=4)`		No loss of precision
`if (x==y)`		Cannot express that x and y must have the same value, not just bounds

* [a,b] means False when a>b.