Vulnerability Analysis (III): Static Analysis
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Efficiency of Symbolic Execution</td>
</tr>
<tr>
<td>2</td>
<td>A Static Analysis Analogy</td>
</tr>
<tr>
<td>3</td>
<td>Syntactic Analysis</td>
</tr>
<tr>
<td>4</td>
<td>Semantics-Based Analysis</td>
</tr>
<tr>
<td></td>
<td>Efficiency of Symbolic Execution</td>
</tr>
<tr>
<td>---</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>2</td>
<td>A Static Analysis Analogy</td>
</tr>
<tr>
<td>3</td>
<td>Syntactic Analysis</td>
</tr>
<tr>
<td>4</td>
<td>Semantics-Based Analysis</td>
</tr>
</tbody>
</table>
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?

$2^n$
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.
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?
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?

```java
if (x == 0) {
    x = -1;
} else {
    x = 1;
}

if (y % 2 == 0) {
    y = y + 1;
} else {
    y = y - 2;
}

assert (x != 0);
```
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?
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
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

- 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

<table>
<thead>
<tr>
<th>Control</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>• <em>Path explosion</em> due to branches and loops</td>
<td>• <em>Algorithmic complexity</em> of arithmetic and string reasoning</td>
</tr>
<tr>
<td>• <em>Redundant exploration</em> of same path prefixes</td>
<td>• <em>Constraint explosion</em> because of irrelevant variables and operations</td>
</tr>
<tr>
<td>• <em>Search strategy</em> determines if vulnerabilities are found</td>
<td>• <em>Memory modeling</em> is labor intensive but necessary</td>
</tr>
</tbody>
</table>

How can we address these issues?
<table>
<thead>
<tr>
<th></th>
<th>Analysis Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Efficiency of Symbolic Execution</td>
</tr>
<tr>
<td>2</td>
<td>Static Analysis by Analogy</td>
</tr>
<tr>
<td>3</td>
<td>Syntactic Analysis</td>
</tr>
<tr>
<td>4</td>
<td>Semantics-Based Analysis</td>
</tr>
</tbody>
</table>
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”

Dawn Song
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
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
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.
Some questions can be answered efficiently.

“Can we drive, on land, from Melbourne to Hobart?”

Not enough information to answer questions about traffic, terrain, the weather, routes from Melbourne to Sydney etc.
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Efficiency of Symbolic Execution</td>
</tr>
<tr>
<td>2</td>
<td>A Static Analysis Analogy</td>
</tr>
<tr>
<td>3</td>
<td>Syntactic Analysis</td>
</tr>
<tr>
<td>4</td>
<td>Semantics-Based Analysis</td>
</tr>
</tbody>
</table>
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

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

- Typical mistake:

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

- Result: file has random permissions

- To detect this problem: Look for oflag == O_CREAT without mode argument
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

```c
chroot("/tmp/sandbox");
fd = fopen("../etc/passwd", "r");
```
Syntactic Example: Name Confusion

```java
/*
* 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)
        // Do something
}
```

- **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

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

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

```java
/* 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

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

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typos</td>
<td>= vs == , &amp;x vs. x , missing/extra semi-colons</td>
</tr>
<tr>
<td>API Usage</td>
<td>chroot, multiple locking, etc.</td>
</tr>
<tr>
<td>Copy-Paste</td>
<td>variable names/increments not updated</td>
</tr>
<tr>
<td>Identifier confusion</td>
<td>global and local variables, fields and parameters</td>
</tr>
</tbody>
</table>
# Pattern Representation and Detection

<table>
<thead>
<tr>
<th>Representation</th>
<th>Types of Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Subsequence mining, edit distance, matching</td>
</tr>
<tr>
<td>Parse Tree</td>
<td>Pattern matching,</td>
</tr>
<tr>
<td>Control Flow Graphs</td>
<td>Automata algorithms, sub-graph isomorphism</td>
</tr>
</tbody>
</table>

Dawn Song
<table>
<thead>
<tr>
<th></th>
<th>Efficiency of Symbolic Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>A Static Analysis Analogy</td>
</tr>
<tr>
<td>3</td>
<td>Syntactic Analysis</td>
</tr>
<tr>
<td>4</td>
<td>Semantics-Based Analysis</td>
</tr>
</tbody>
</table>
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

Entry

\[ x = 0; \]

\[ \text{if} \ (y == 0) \]

\[ \text{false} \quad \text{true} \]

\[ x = x - 1; \quad x = x + 1; \]

\[ \text{if} \ (y == 0) \]

\[ \text{false} \quad \text{true} \quad \text{false} \]

Exit

Err

\[ x <= 0 \quad x != 0 \quad x >= 0 \]

\[ x < 0 \quad x == 0 \quad x > 0 \]
Abstracting Data

Entry

\[
x = 0;
\]

if \( y == 0 \)

\[
x = x - 1;
\]

\[
x = x + 1;
\]

if \( y == 0 \)

\[
x = x - 1;
\]

\[
x = x + 1;
\]

if \( x < 0 \)

Exit

Err

Only track relevant properties of \( x \)

\( x \) can have any value
Abstracting Data

Only track relevant properties of x

x can have any value

no value is feasible

true

false

x<=0

x!=0

x>=0

x<0

x==0

x>0
Sign Analysis

Analysis: update data about x based on control flow

Entry

x = 0;

if (y == 0)

x = x - 1;

if (y == 0)

x = x + 1;

true

false

false

true

false

true

false

true

Exit

if (x < 0)

Err

x <= 0

x != 0

x >= 0

x < 0

x == 0

x > 0

true

false

Dawn Song
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”
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

true

x = 0;

false

x = x - 1;

true

x = x + 1;

false

x = x - 1;

true

if (y == 0)

false

x <= 0

true

if (x < 0)

false

x >= 0

false

x != 0

false

x == 0

false

x <= 0

true

x > 0

true

if (y == 0)

false

x == 0

true

if (x < 0)

false

x <= 0

true

x != 0

true

false

x >= 0

true

if (y == 0)

false

x == 0

true

if (x < 0)

false

true

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

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
Dawn Song

Sign Analysis

Analysis: update data about x based on control flow
Sign Analysis

Entry

\( x = 0; \)

**if \( y == 0 \)**

- **false:**
  - \( x = x - 1; \)
  - \( x = x + 1; \)
  - \( x<0 \)
  - \( x!=0 \)

- **true:**
  - \( x==0 \)

**if \( y == 0 \)**

- **false:**
  - \( x!=0 \)

- **true:**
  - \( x<0 \)
  - \( x==0 \)
  - \( x>0 \)

Exit

**Err**

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
Sign Analysis

Analysis: update data about $x$ based on control flow

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

Entry

$x = 0$;

if ($y == 0$)

x = x - 1;

false

x = x + 1;

true

if ($y == 0$)

false

true

false

if ($x < 0$)

Err

true

ty==0

ty!=0

false

Dawn Song
Zero Propagation

Suppose we only track if $y$ is zero or not

Suppose we only track if $y$ is zero or not

```
x = 0;

if (y == 0)
    if (x < 0)
        Err
    else
        x = x + 1;
else
    x = x - 1;
```
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?

```plaintext
x = 0;

if (y == 0)
    x = x - 1;
else
    x = x + 1;

if (y == 0)
    Err
```

Dawn Song
Suppose we only track if \( y \) is zero or not.

Can you fill in the blanks for the first steps of the analysis?
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.
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.
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.
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 \textit{fixed point} is reached.
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

Entry

```cpp
x = 0;
```

```
if (y == 0)
```

```
if (x < 0)
```

```
x = x - 1;
```

```
x = x + 1;
```

Exit

Err

true

true

false

true

true

true

true

true

true

true

true

true

true

true

true

true

true

true

true

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

Dawn Song
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

```
if (y == 0)
  \{ x = x - 1; \}
\{ x = x + 1; \}
```

instead of

```
true
```

Entry

```
x = 0;
```

Exit

```
x >= 0
y != 0
```

```
x < 0
y == 0
```

```
x >= 0
y == 0
```

```
x < 0
y == 0
```

```
x < 0
y == 0
```

Dawn Song
Analysis with Disjunctive Refinement

Entry

x = 0;

if (y == 0)

false
false
false true false

if (y == 0)

if (x < 0)

true
true
true

Exit

Err

Initial steps are the same
Analysis with Disjunctive Refinement

Entry

\( x = 0; \)

if \( y == 0 \)

\( x = x - 1; \) \quad \text{false}\n
\( x = x + 1; \) \quad \text{true}\n
if \( y == 0 \)

Exit \quad \text{true}\n
if \( x < 0 \)

Err

Join is now disjunction
Analysis with Disjunctive Refinement

Entry

true
x==0
true
x==0

if (y == 0)

false

x = x - 1;
x = x + 1;

true
x==0
y==0

if (y == 0)

true

x = x - 1;
x = x + 1;

false

x>0
y==0

if (y == 0)

false

x<0
y!=0

Exit true

if (x < 0)

true

Err

More precise information propagated after join

x<0
y!=0

x>0
y==0

x>0
y==0

x<0
y==0

Dawn Song
Analysis with Disjunctive Refinement

Result of final analysis. Assertion is not violated.
<table>
<thead>
<tr>
<th></th>
<th>Analysis Frameworks</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Types of Analyses</td>
</tr>
<tr>
<td>3</td>
<td>Precision</td>
</tr>
<tr>
<td>4</td>
<td>Summary of Program Analysis</td>
</tr>
<tr>
<td></td>
<td>Analysis Frameworks</td>
</tr>
<tr>
<td>---</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>a</td>
<td>Lattices</td>
</tr>
<tr>
<td>b</td>
<td>Transformers</td>
</tr>
<tr>
<td>c</td>
<td>Systems of Equations</td>
</tr>
<tr>
<td>d</td>
<td>Solving Equations</td>
</tr>
<tr>
<td></td>
<td>Analysis Frameworks</td>
</tr>
<tr>
<td>---</td>
<td>---------------------</td>
</tr>
<tr>
<td>a</td>
<td>Lattices</td>
</tr>
<tr>
<td>b</td>
<td>Transformers</td>
</tr>
<tr>
<td>c</td>
<td>Systems of Equations</td>
</tr>
<tr>
<td>d</td>
<td>Solving Equations</td>
</tr>
</tbody>
</table>
States, Transitions, Executions

```c
int a[5];

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

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

States

Values of local and global variables, program counter, stack, heap

d
1
0
undefined
undefined
undefined
undefined

pc
i
a[0]
a[1]
a[2]
a[3]
a[4]
States, Transitions, Executions

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

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

Control

Data

<table>
<thead>
<tr>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>undef</td>
</tr>
<tr>
<td>undef</td>
</tr>
<tr>
<td>undef</td>
</tr>
<tr>
<td>undef</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>pc</th>
<th>States</th>
<th>Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>values of local and global variables, program counter, stack, heap</td>
<td></td>
</tr>
<tr>
<td>a[0]</td>
<td>state changes</td>
<td></td>
</tr>
<tr>
<td>a[1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a[2]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a[3]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a[4]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>d</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>undefined</td>
<td>undefined</td>
</tr>
<tr>
<td>undefined</td>
<td>undefined</td>
</tr>
<tr>
<td>undefined</td>
<td>undefined</td>
</tr>
</tbody>
</table>

Dawn Song
States, Transitions, Executions

```c
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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Control**

**Data**

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

States, Transitions, Executions

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

<table>
<thead>
<tr>
<th>pc</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>undef</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>a[0]</td>
<td>undef</td>
<td>undef</td>
<td>undefined</td>
<td>undefined</td>
<td>undefined</td>
</tr>
<tr>
<td>a[1]</td>
<td>undef</td>
<td>undef</td>
<td>undefined</td>
<td>undefined</td>
<td>undefined</td>
</tr>
<tr>
<td>a[2]</td>
<td>undef</td>
<td>undef</td>
<td>undefined</td>
<td>undefined</td>
<td>undefined</td>
</tr>
<tr>
<td>a[3]</td>
<td>undef</td>
<td>undef</td>
<td>undefined</td>
<td>undefined</td>
<td>undefined</td>
</tr>
<tr>
<td>a[4]</td>
<td>undef</td>
<td>undef</td>
<td>undefined</td>
<td>undefined</td>
<td>undefined</td>
</tr>
</tbody>
</table>

Sequence of state changes:
- i=0: a[i]=0
- ++i: a[i]=0

Dawn Song
Variables have values, define state

Statements modify values, define transitions on data

Control flow modify program counter, define control transitions
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

<table>
<thead>
<tr>
<th>Signs</th>
<th>Parity</th>
<th>Constants</th>
</tr>
</thead>
</table>
| • positive/negative/zero  
• cannot represent non-zero values  
• no relationships between variables | • even or odd  
• cannot represent values  
• no relationships between variables | • a single value  
• cannot represent more values: $x=3 \mid \mid x=4$  
• no relationships between variables |

![Lattice Diagram](attachment:image.png)
The Interval Lattice

An interval is a pair \([a,b]\) with \(a \leq b\)

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.

<table>
<thead>
<tr>
<th>Arbitrary sets</th>
<th>{1,5}, {1,3,5}, {1,2,4,5} are represented by [1,5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Union</td>
<td>[1,3] join [6,7] = [1,7] includes values 4 and 5</td>
</tr>
<tr>
<td>Relations</td>
<td>x=y can only be written as x:[INT_MIN,INT_MAX], y:[INT_MIN,INT_MAX]</td>
</tr>
</tbody>
</table>
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
<table>
<thead>
<tr>
<th></th>
<th>Analysis Frameworks</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Lattices</td>
</tr>
<tr>
<td>b</td>
<td>Transformers</td>
</tr>
<tr>
<td>c</td>
<td>Systems of Equations</td>
</tr>
<tr>
<td>d</td>
<td>Solving Equations</td>
</tr>
</tbody>
</table>
A transformer (or transfer function) describes how a statement modifies lattice elements.

<table>
<thead>
<tr>
<th>x = 0;</th>
<th>x = x+1;</th>
<th>if (x &gt; 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x &gt;= 0</td>
<td>x &gt;= 0</td>
<td>true</td>
</tr>
<tr>
<td>x &lt;= 0</td>
<td>x &lt;= 0</td>
<td>true</td>
</tr>
<tr>
<td>x != 0</td>
<td>x != 0</td>
<td>true</td>
</tr>
<tr>
<td>x &gt; 0</td>
<td>x &gt; 0</td>
<td>true</td>
</tr>
<tr>
<td>x &lt; 0</td>
<td>x &lt; 0</td>
<td>true</td>
</tr>
<tr>
<td>x == 0</td>
<td>x == 0</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

Dawn Song
# Interval Analysis Transformers

<table>
<thead>
<tr>
<th>Statement</th>
<th>Transformer</th>
<th>Loss of Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x = x + 3 )</td>
<td>[\text{a}\leftarrow\text{a+3}\rightarrow\text{b+3}]</td>
<td>No loss of precision</td>
</tr>
<tr>
<td>( x = 2^x )</td>
<td>[\text{a}\leftarrow\text{2a}\rightarrow\text{2b}] (2\text{a}\text{ to }2\text{b}) multiples of 2 in [2\text{a},2\text{b}]</td>
<td>[3,4] is transformed to [6,8] and includes 7, which is not a multiple of 2</td>
</tr>
<tr>
<td>if (( x \leq 4 ))</td>
<td>[\text{a}\leftarrow\text{4}\rightarrow\text{min(b,4)}^*]</td>
<td>No loss of precision</td>
</tr>
<tr>
<td>if (( x = y ))</td>
<td>[\text{a}\leftarrow\text{max(a,c)}\rightarrow\text{x}\leftarrow\text{min(b,d)}^*]</td>
<td>Cannot express that ( x ) and ( y ) must have the same value, not just bounds</td>
</tr>
</tbody>
</table>

* \([a,b]\) means False when \(a > b\).