# Identifying Minimal Changes in the Zone Abstract Domain

Kenny Ballou
Elena Sherman

Boise State University
Boise, Idaho
United States of America

July 2023



- Thank you.
- I'm Kenny Ballou and I will be presenting our work on Identifying Minimal Changes in the Zone Abstract Domain.

# Outline

# Static analysis computes invariants



- Static analysis computes facts about programs. These facts can be represented in various ways.
- A common approach for numerical abstract interpretation, is a restricted set of inequalities

## Static analysis computes invariants

### Unit difference, two-variables per inequality

$$x - Z_0 = 0$$
$$w - x \leq 2$$



- Static analysis computes facts about programs. These facts can be represented in various ways.
- A common approach for numerical abstract interpretation, is a restricted set of inequalities

# Static analysis computes invariants

## Inequalities as invariants for a simple program

```
int example(int w, int y) {
    int x = 0;
    if (w <= x + 2) {
        if (y <= x) {
            assert y <= 0;
        }
    }
    return x;
}
```

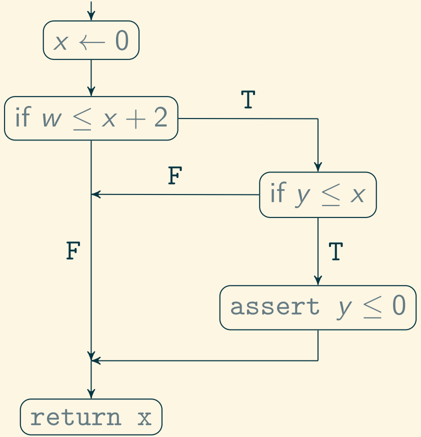


- Given some code, we produce a CFG
- Using DFA, we compute some invariants, for example
  - These two trapezoids represent the in-state and the out-state for the inner if
- Let's focus on this invariant, and describe briefly how Zones work

# Static analysis computes invariants

## Inequalities as invariants for a simple program

```
int example(int w, int y) {
    int x = 0;
    if (w <= x + 2) {
        if (y <= x) {
            assert y <= 0;
        }
    }
    return x;
}
```



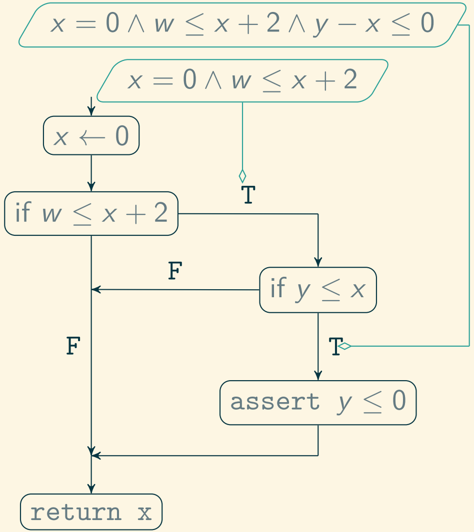



- Given some code, we produce a CFG
- Using DFA, we compute some invariants, for example
  - These two trapezoids represent the in-state and the out-state for the inner if
- Let's focus on this invariant, and describe briefly how Zones work

# Static analysis computes invariants

## Inequalities as invariants for a simple program

```c
int example(int w, int y) {
    int x = 0;
    if (w <= x + 2) {
        if (y <= x) {
            assert y <= 0;
        }
    }
    return x;
}
```



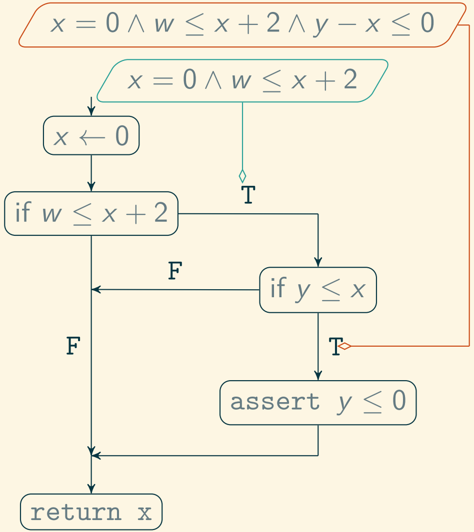



- Given some code, we produce a CFG
- Using DFA, we compute some invariants, for example
  - These two trapezoids represent the in-state and the out-state for the inner if
- Let's focus on this invariant, and describe briefly how Zones work

# Static analysis computes invariants

## Inequalities as invariants for a simple program

```
int example(int w, int y) {
    int x = 0;
    if (w <= x + 2) {
        if (y <= x) {
            assert y <= 0;
        }
    }
    return x;
}
```






- Given some code, we produce a CFG
- Using DFA, we compute some invariants, for example
  - These two trapezoids represent the in-state and the out-state for the inner if
- Let's focus on this invariant, and describe briefly how Zones work

## Zone Domain

$$
\begin{array}{c}
x - Z_0 \leq 0 \\
Z_0 - x \leq 0 \\
w - x \leq 2 \\
y - x \leq 0 \\
\hline
y \leq 0 \\
w \leq 2
\end{array}
$$

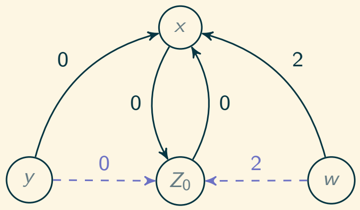
Zonal state representation of data-flow analysis invariant



- Static program analysis computes invariants. We can represent these invariants with different ways. A popular choice for representing these invariants is the zonal domain as introduced by Miné.

## Zone Domain

$$
\begin{array}{c}
x - Z_0 \leq 0 \\
Z_0 - x \leq 0 \\
w - x \leq 2 \\
y - x \leq 0 \\
\hline
y \leq 0 \\
w \leq 2
\end{array}
$$

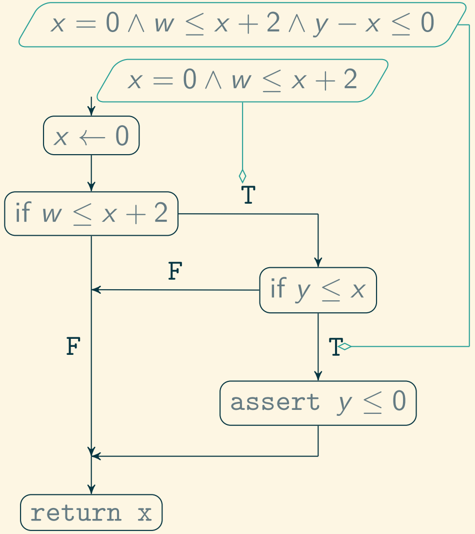



Zonal state representation of data-flow analysis invariant



- Static program analysis computes invariants. We can represent these invariants with different ways. A popular choice for representing these invariants is the zonal domain as introduced by Miné.

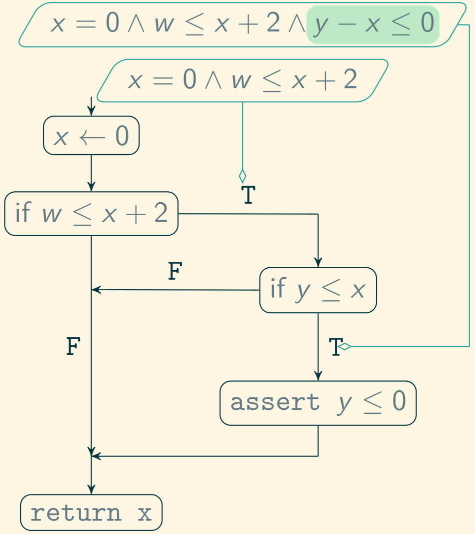
# Data-flow analysis incrementally updates variables






- Consider this simple data flow graph
- Using the intervals domain, for example, we know that the x variable is the only variable changed at each point along the true path.
- Using Zones, we can gain information about $y$.
- However, aside from small examples, it's unlikely that an update affects the entire state.

## Data-flow analysis incrementally updates variables






- Consider this simple data flow graph
- Using the intervals domain, for example, we know that the x variable is the only variable changed at each point along the true path.
- Using Zones, we can gain information about $y$.
- However, aside from small examples, it's unlikely that an update affects the entire state.

## Finding Affected Inequalities

### Problem Definition

$$\frac{\begin{array}{r} x = 0 \\ w - x \leq 2 \\ y - x \leq 0 \end{array}}{w \leq 2} \quad \Longrightarrow \quad \frac{\begin{array}{r} x = 0 \\ w - x \leq 2 \\ y - x \leq 0 \end{array}}{\begin{array}{c} y \leq 0 \\ w \leq 2 \end{array}}$$

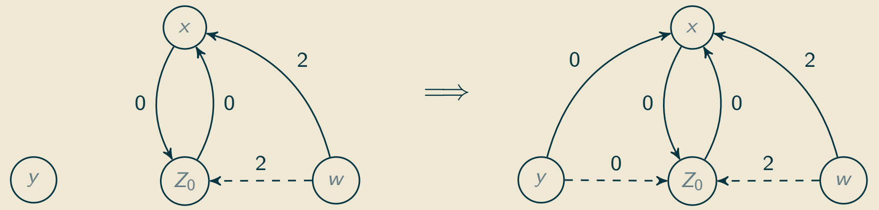
What are the changed set of inequalities?



- Our problem is finding the smallest set of inequalities between two
- We approach this with several algorithms, which we now present.

# Finding Affected Inequalities

## Problem Definition



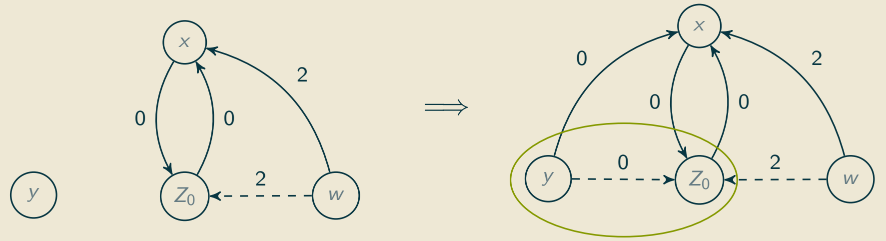

What are the changed set of inequalities?



- Our problem is finding the smallest set of inequalities between two
- We approach this with several algorithms, which we now present.

# Finding Affected Inequalities

## Problem Definition

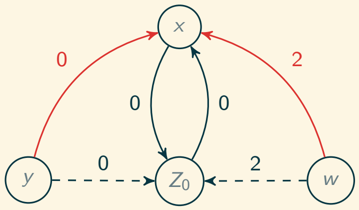



What are the changed set of inequalities?



- Our problem is finding the smallest set of inequalities between two
- We approach this with several algorithms, which we now present.

## Spurious Connected Variables[1]

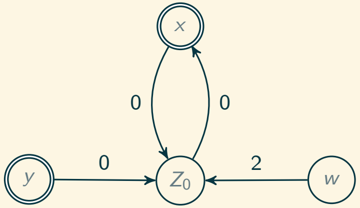



[1] Larsen et al., "Efficient Verification of Real-Time Systems: Compact Data Structure and State-Space Reduction".



- Through inference, we may gain connections to variables which do not have an intentional logical connection
- Borrowing from Larsen et al. work, we remove these "spurious connections" as a pre-processing step to the remainder of our algorithms

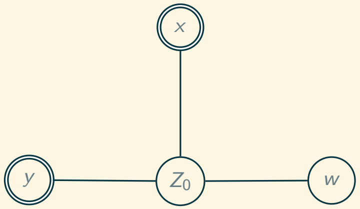
# Connected Components




- Considering the state without the spurious connections
- The approach proposed by Visser et al. considers the undirected variable relation projection of the different constraints

# Connected Components

## Variable Relation Projection

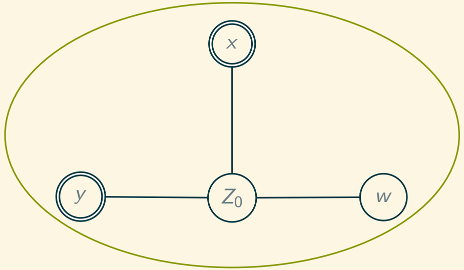





- Considering the state without the spurious connections
- The approach proposed by Visser et al. considers the undirected variable relation projection of the different constraints

## Connected Components

### Variable Relation Projection

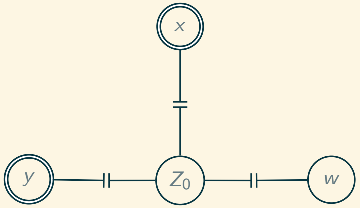





- Considering the state without the spurious connections
- The approach proposed by Visser et al. considers the undirected variable relation projection of the different constraints

# Connected Components

Variable Relation Projection with impassable $Z_0$

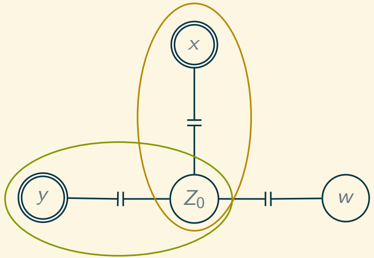





- Considering the state without the spurious connections
- The approach proposed by Visser et al. considers the undirected variable relation projection of the different constraints

## Connected Components

Variable Relation Projection with impassable $Z_0$

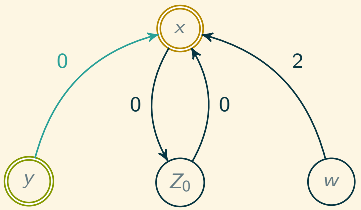





- Considering the state without the spurious connections
- The approach proposed by Visser et al. considers the undirected variable relation projection of the different constraints

## Node Neighbors

Reconsider the out-going state without closed edges

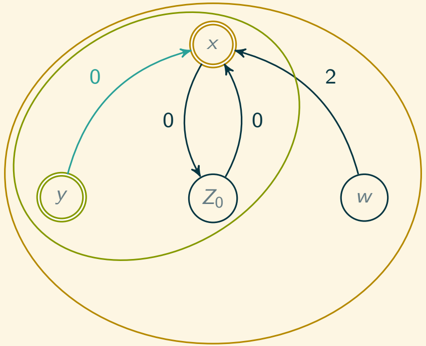





- To demonstrate our next technique, we consider the same state without closure.
- The next technique extracts the forward-reachable and backward-reachable neighborhood of each changed variable.

## Node Neighbors

Reconsider the out-going state without closed edges

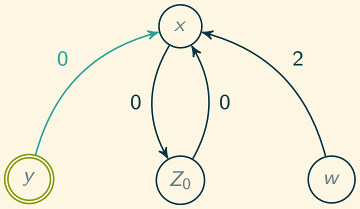





- To demonstrate our next technique, we consider the same state without closure.
- The next technique extracts the forward-reachable and backward-reachable neighborhood of each changed variable.

# Minimal Neighbors

Again, reconsider the out-going state without closed edges.

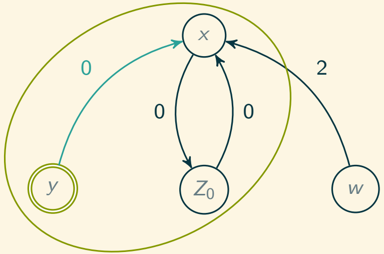





- Consider, instead, the edge of the updated edge.

# Minimal Neighbors

Again, reconsider the out-going state without closed edges.






- Consider, instead, the edge of the updated edge.

## Logically comparing different abstract domains

**Research Questions**

*RQ1* Do the minimization algorithms reduce the size of a Zone state and improve runtime of domain comparisons?

*RQ2* Do the minimization algorithms affect categorization of domain comparison results?



- We consider the following research questions for our application of empirically comparing abstract domains
- We compare Zones and Intervals, two comparable domains; and we compare Zones and Predicates, two incomparable domains

## Experimental Setup

- Benchmarks: 127 Java methods
  - Ranging from 4 to 412 Jimple instructions
- Compared Zones to Intervals and Zones to Predicates
- Compared Total Runtime of Z3 to perform logical entailment of every combination, averaging over 5 executions



- Benchmark: 127 Java Methods selected from previous research
- Subject programs range from 4 to 412 Jimple instructions.

## Experimental results show significant reduction in required number of inequalities for comparison

Average percentage changes in $V$ and $E$ between each technique

| State Type | vs. | $\downarrow\Delta$ % V | $\downarrow\Delta$ % E |
|---|---|---|---|
| **DFA Subject Programs** | | | |
| CC | FS | 70.37 | 29.47 |
| NN | CC | 0.02 | 0.01 |
| MN | NN | 0.10 | 0.05 |
| **EQBench Subject Programs** | | | |
| CC | FS | 43.0 | 2.1 |
| NN | CC | 0.0 | 0.0 |
| MN | NN | 0.13 | 0.13 |



- We see differences between the two benchmark suites due to the different qualities between the suites themselves
  - For example, the DFA benchmark suite is pulled from open-source projects with a high-preponderance of integer operations
  - Conversely, the EqBench consists of simply methods mainly used for equivalence testing within symbolic execution, many operations are not necessarily integer based

# Experimental results show significantly reduced time to solver queries

| State Type | $\sim$ Inter, sec. | $\sim$ Pred, sec. |
|---|---|---|
| **DFA Subject Programs** | | |
| FS | 4.03 | 265.91 |
| CC | 1.41 | 4.09 |
| NN | 1.41 | 4.04 |
| MN | 1.35 | 4.05 |
| **EQBench Subject Programs** | | |
| FS | 0.79 | 5.56 |
| CC | 0.63 | 0.87 |
| NN | 0.58 | 0.9 |
| MN | 0.58 | 0.9 |

# Experimental results show significant improvement in comparison granularity

| State | $\succ$ Intervals | $=$ Intervals |
|---|---|---|
| **DFA Subject Programs** | | |
| FS | 2898 | 1002 |
| CC | 1194 | 2706 |
| NN | 1191 | 2709 |
| MN | 1164 | 2736 |
| **EQBench Subject Programs** | | |
| FS | 374 | 255 |
| CC | 131 | 498 |
| NN | 131 | 498 |
| MN | 131 | 498 |

# Experimental results show significant improvement in comparison granularity

| State | $\succ$ Predicates | $=$ Predicates | $\prec$ Predicates | $\prec\succ$ Predicates |
|---|---|---|---|---|
| **DFA Subject Programs** | | | | |
| FS | 1464 | 237 | 167 | 2032 |
| CC | 1324 | 1930 | 473 | 173 |
| NN | 1322 | 1933 | 473 | 172 |
| MN | 1305 | 1960 | 473 | 162 |
| **EQBench Subject Programs** | | | | |
| FS | 307 | 135 | 46 | 141 |
| CC | 217 | 322 | 72 | 18 |
| NNy | 217 | 322 | 72 | 18 |
| MN | 217 | 322 | 72 | 18 |

# Conclusion

## Experimental Results

- Minimization leads to reduced overall execution time when determining domain categorization.
- Minimization leads to improved granularity when evaluating domain precision.

# Conclusion

## Experimental Results

- Minimization leads to reduced overall execution time when determining domain categorization.
- Minimization leads to improved granularity when evaluating domain precision.

## Algorithms and Approaches

- Spurious Connections → Reduce variable clustering
- Connected Components → Extract subsets using relational projection
- Node Neighbors → Extract subsets based on reachable neighborhoods
- Minimal Neighbors → Extract subsets leveraging semantic information

## Future Work

- Extend to other Weakly-Relational Domains, e.g., Octagons
- Extend for comparison between relational domains



- Extend minimal identification to other weakly-relational domains, e.g., Octagons
- Extend minimal changes to a minimal union between two states for comparison of two weakly-relational domains.

# Thank you

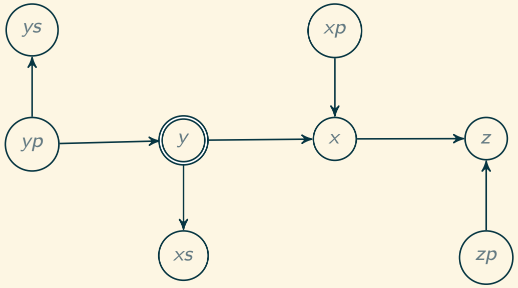
## Questions?

The work reported here was supported by the U.S. National Science Foundation under award CCF-19-42044.

# References I

[1] K.G. Larsen et al. “Efficient Verification of Real-Time Systems: Compact Data Structure and State-Space Reduction”. In: *Proceedings Real-Time Systems Symposium*. IEEE Comput. Soc, 1997, pp. 14–24. ISBN: 081868268X. DOI: 10.1109/real.1997.641265.

# Extended Examples of the Minimal Neighbors Algorithm



# Extended Examples of the Minimal Neighbors Algorithm