# Check-in

Review: Dataflow

Give an example of a forward dataflow analysis and an example of a backward dataflow analysis.

# Announcements

Review: Dataflow

Drew Davidson | University of Kansas


# Abstract Interpretation

# Previously…

Review: Dataflow

**Global Dataflow analysis**

- Intuition
- Operations

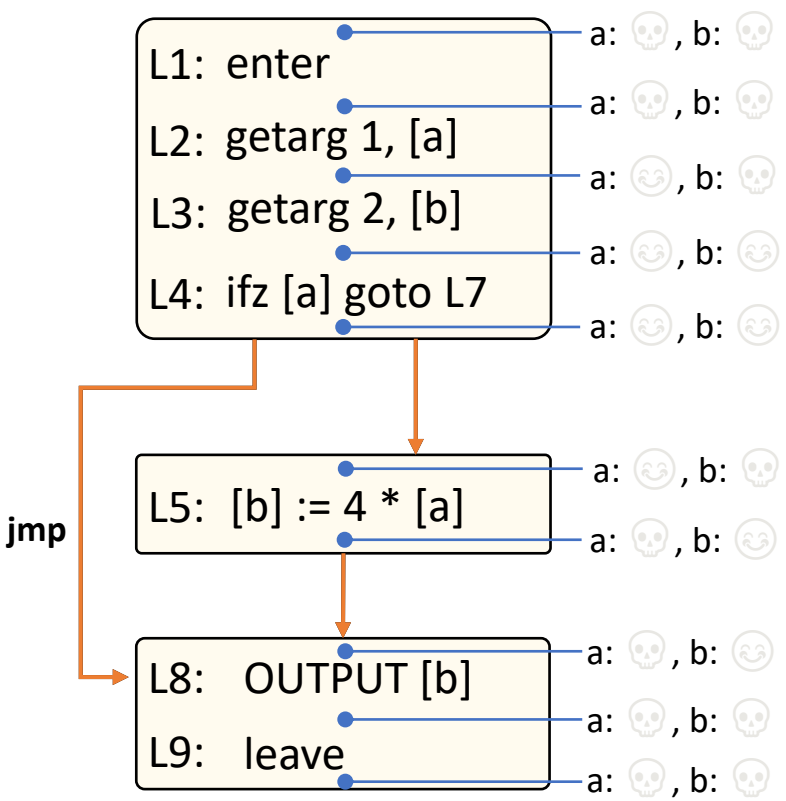
**<u>You should know</u>**

- The basic concepts of dataflow facts
  - Backwards and Forward analysis
  - Augment local analysis with “IN” and “OUT” sets
  - You need to merge fact sets

**Optimization**

# Merging Fact Sets

## Dataflow Intuition




**Fact sets may be different when multiple successors/predecessors join**

- Need to merge incoming fact sets

**Merge as conservatively as possible**

- Don't do anything without a guarantee!
- Plan for all possible flows

**Example: is L3 live?** *(consider both block paths)*

- L3 definition clobbered on the fallthrough branch (at L5)
- L3 definition not clobbered on the jump branch

# Today's Outline

IR Optimization

**Rounding out dataflow analysis concepts**

- Some more examples
- Considering more complex code
- Dataflow Framework

**Abstract Interpretation**

- Concepts
- Examples

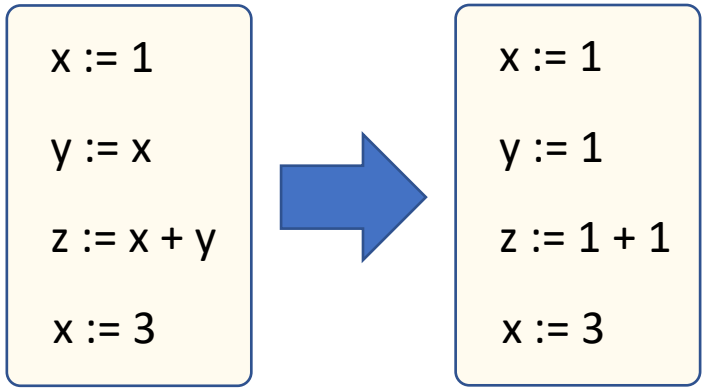
**Optimization**

# Refresh Constant/Copy Propagation

Dataflow: Formalization

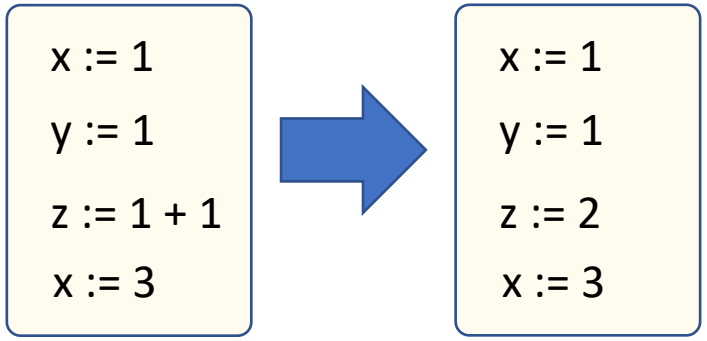
## Copy Propagation

- Replace RHS of simple assigns with value of assign (if known)
- Forward analysis




## Constant folding

- Replace constant RHS expressions with value
- Traversal order isn't important



# Example Analyses

Dataflow: Formalization

## Dead Code Elimination

- Backwards analysis
- Fact sets: the liveness of each variable

| Known Live | Known Dead | Not Enough Info |
|---|---|---|
| 😋 | 💀 | 👶 |

- Merge:

😋 ∪ 💀 = 😋

😋 ∪ 👶 = 😋

💀 ∪ 👶 = 👶

## Constant Propagation

- Forward analysis
- Fact sets: the known value of each variable

| Set of Known Values | Not Enough Info |
|---|---|
| {<value>, <value2}, … | 👶 |

- Merge:

Set Union

{ 1 } ∪ { 1,2 } = { 1,2 }

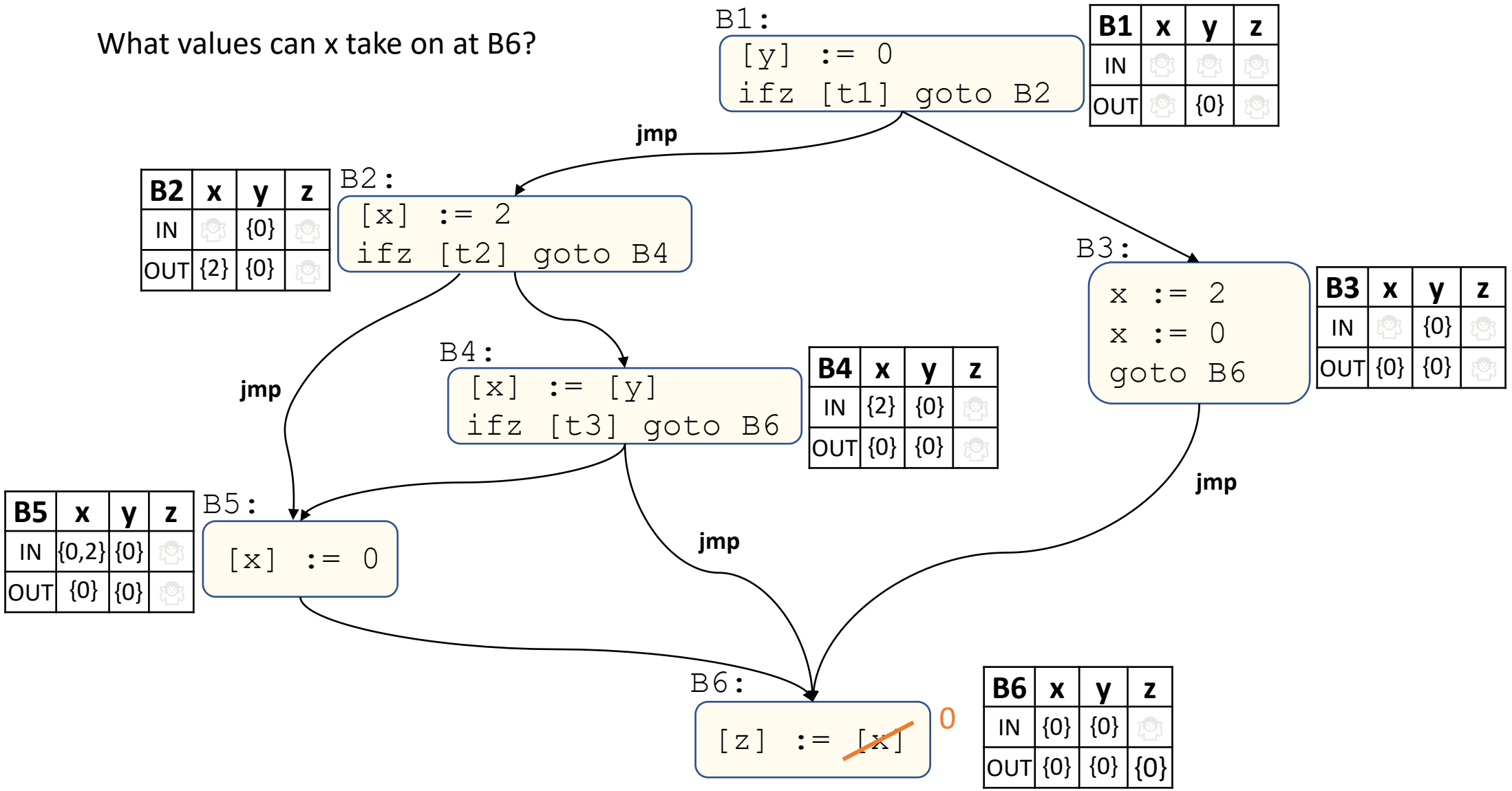
…except

{ * } ∪ 👶 = 👶

# Example Constant Propagation

## Dataflow: Formalization - Example

What values can x take on at B6?




| B1 | x | y | z |
|---|---|---|---|
| IN | | | |
| OUT | | {0} | |

| B2 | x | y | z |
|---|---|---|---|
| IN | | {0} | |
| OUT | {2} | {0} | |

| B3 | x | y | z |
|---|---|---|---|
| IN | | {0} | |
| OUT | {0} | {0} | |

| B4 | x | y | z |
|---|---|---|---|
| IN | {2} | {0} | |
| OUT | {0} | {0} | |

| B5 | x | y | z |
|---|---|---|---|
| IN | {0,2} | {0} | |
| OUT | {0} | {0} | |

| B6 | x | y | z |
|---|---|---|---|
| IN | {0} | {0} | |
| OUT | {0} | {0} | {0} |

# Today's Outline

IR Optimization

**Rounding out dataflow analysis concepts**

- Some more examples
- Considering more complex code
- Instantiating Dataflow Framework

**Abstract Interpretation**

- Concepts
- Examples

**Optimization**

# Handling Practical Programs

Global Dataflow: Formalization

**Global variables**

- We only have visibility into 1 procedure
- Be conservative about the effect of other procedures
  - Reset fact sets across a call
  - Consider global variables live at function end

# Analysis Termination

Dataflow: Formalization

**In the previous examples, we completed in one pass over the CFG**

- This won't always be the case, due to a fundamental construct…

# Analysis Termination

Dataflow: Formalization

**In the previous examples, we completed in one pass over the CFG**

- This won't always be the case, due to a fundamental construct… loops
- Loops (specifically back edges) create cyclic dependencies

***Oh bröther, you might have some lööps***

# Loops: Dependency cycles

Dataflow: Formalization

**Solution: Saturate fact sets**

- Start sets "TBD" ( ) value
- Run the algorithm until sets don't change

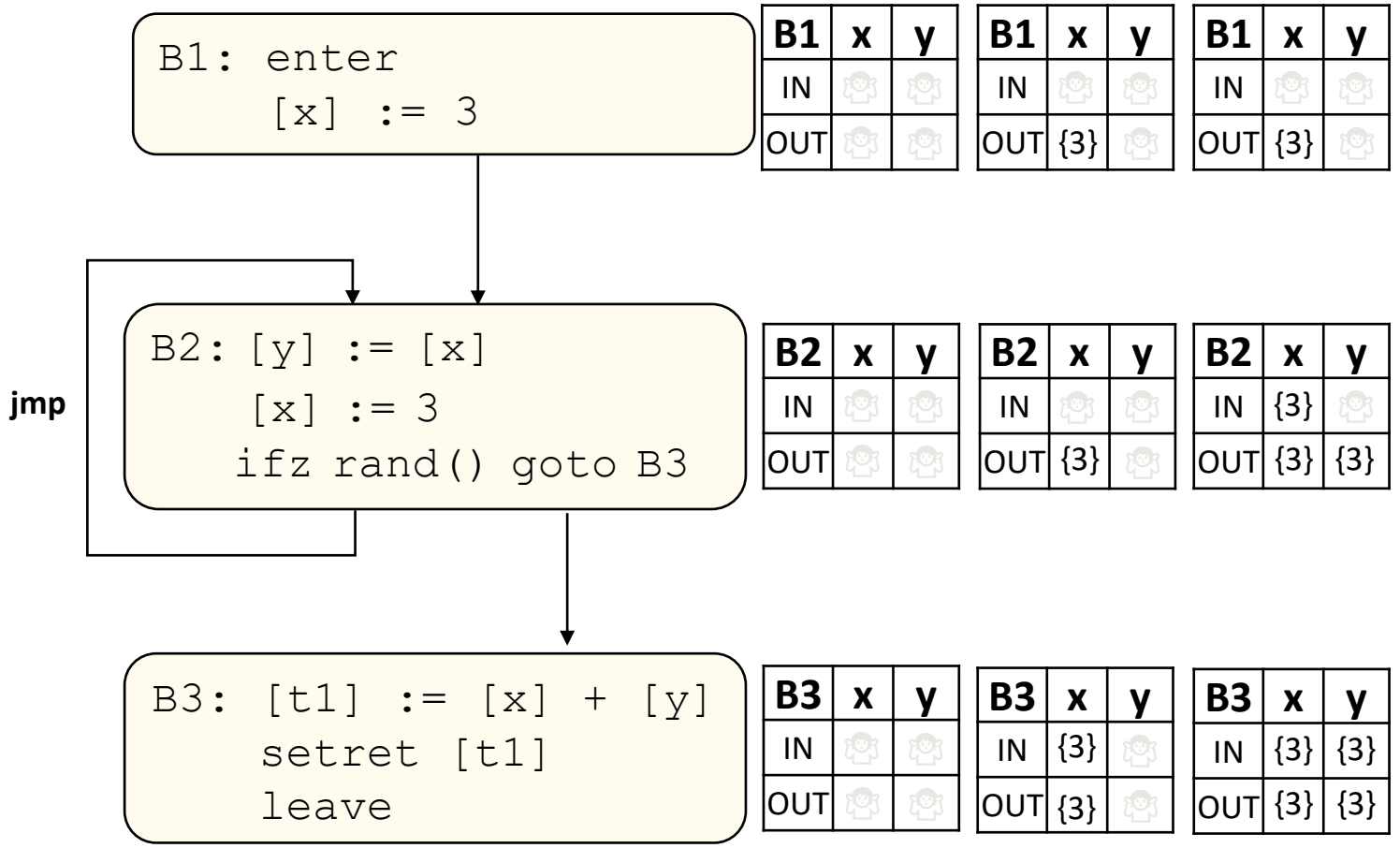
**We've seen the saturation approach before**

- (FIRST and FOLLOW sets)

**Constant propagation**

IN(B2) requires knowing OUT(B2)

OUT(B2) requires knowing IN(B2)



# Handling Practical Data Abstractions

Global Dataflow: Formalization

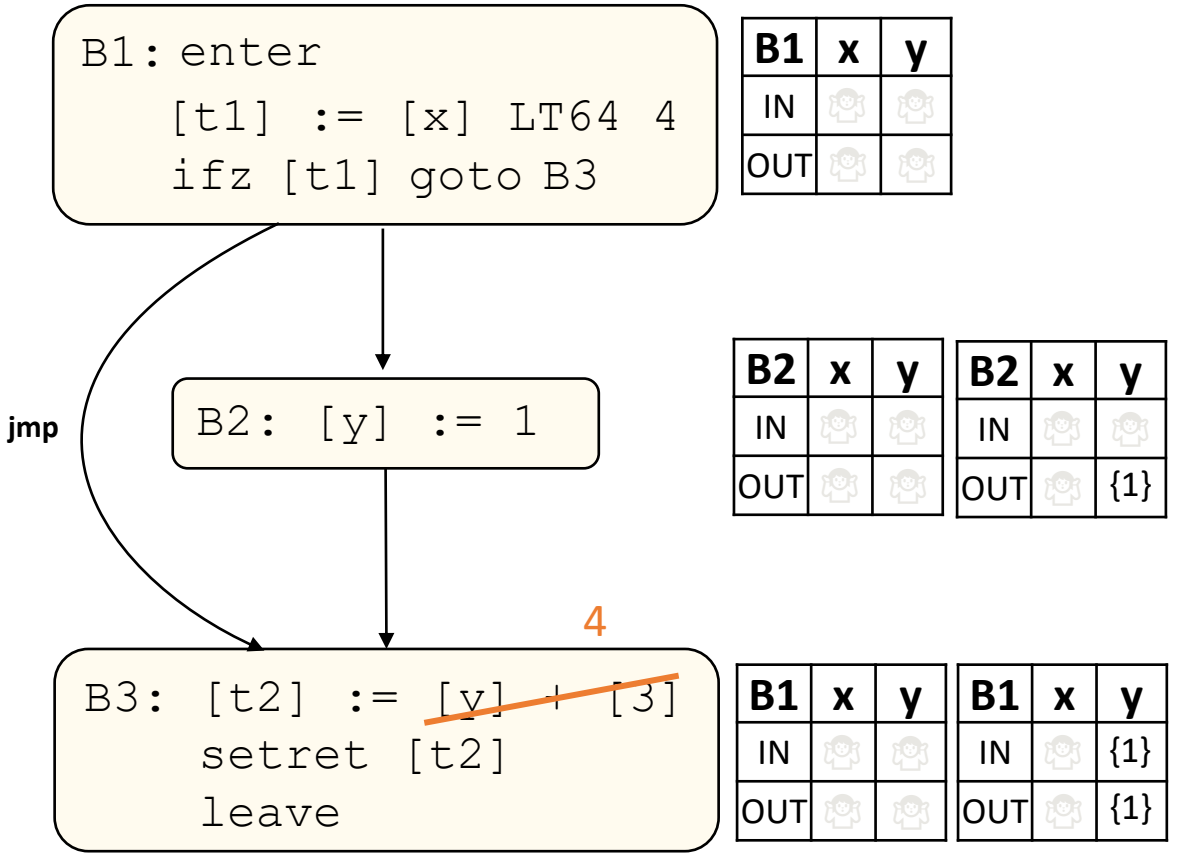
## Undefined Behavior

```
int main(){
  int x,y;
  if (x == 4){
    y = 1;
  }
  return y + 3;
}
```

- Could we fold y + 3?

Ain't no law against it!

Would need to have types of unknowns



# Today's Outline

IR Optimization

**Rounding out dataflow analysis concepts**

- Some more examples
- Considering more complex code
- Instantiating Dataflow Framework

**Abstract Interpretation**

- Concepts
- Examples

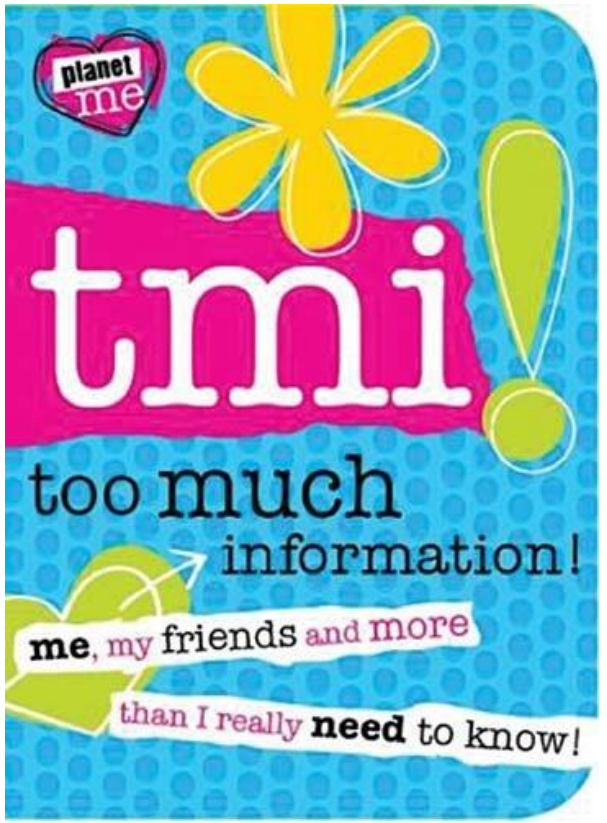
**Optimization**

# Complicated Fact Sets

Dataflow: Formalization

**Occasionally, fact sets exceed their usefulness, e.g.:**

- Constant propagation: once we have > 1 value in a set, we don't really care what the values are
- Change the domain of values to match what we can learn / use in analysis

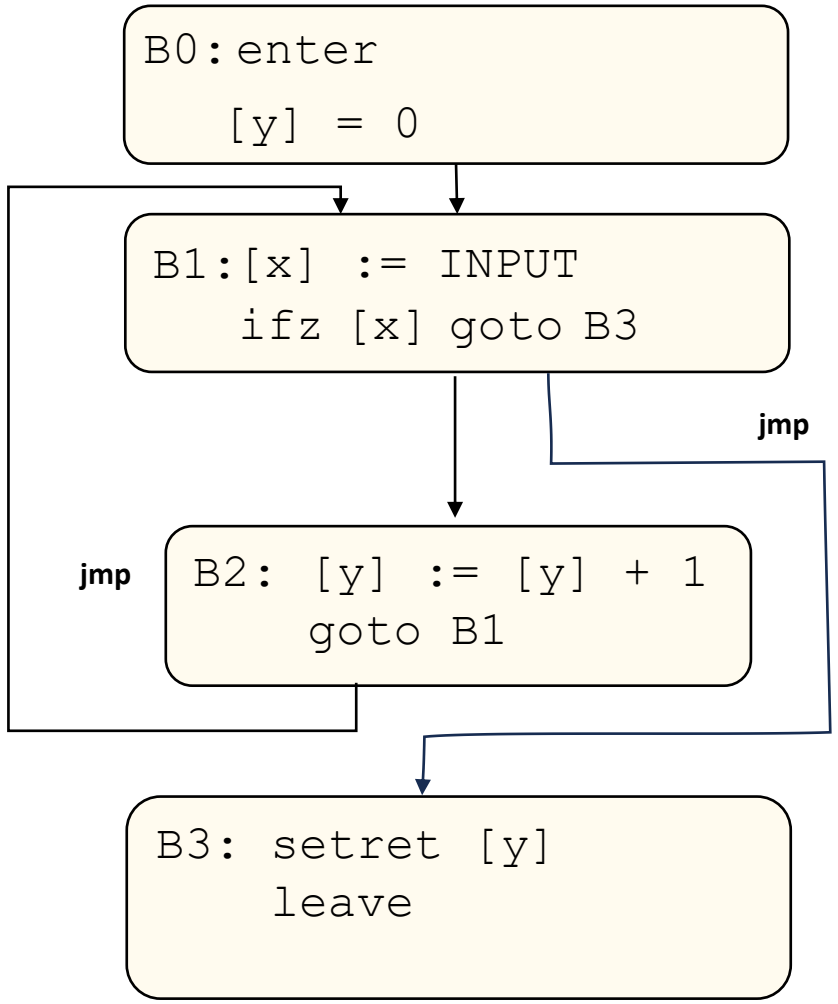


# Complicated Fact Sets

Dataflow: Formalization

**Occasionally, fact sets exceed their usefulness, e.g.:**

- Constant propagation: once we have > 1 value in a set, we don't really care what the values are
- Change the domain of values to match what we can learn / use in analysis




| B0 | x | y |
|---|---|---|
| IN | | |
| OUT | | |

| B1 | x | y |
|---|---|---|
| IN | | |
| OUT | | |

| B2 | x | y |
|---|---|---|
| IN | | |
| OUT | | |

| B3 | x | y |
|---|---|---|
| IN | | |
| OUT | | |

# Complicated Fact Sets

Dataflow: Formalization

**Occasionally, fact sets exceed their usefulness, e.g.:**

- Constant propagation: once we have > 1 value in a set, we don't really care what the values are
- Change the domain of values to match what we can learn / use in analysis

**Before**

| Set of Known Values | We Don't Know |
|---|---|
| {1}, {1,2}, … | |

**After**

| Single Constant Value | We Don't Know | Could be Anything |
|---|---|---|
| 1, 2, 3, … | ⊥ | ⊤ |

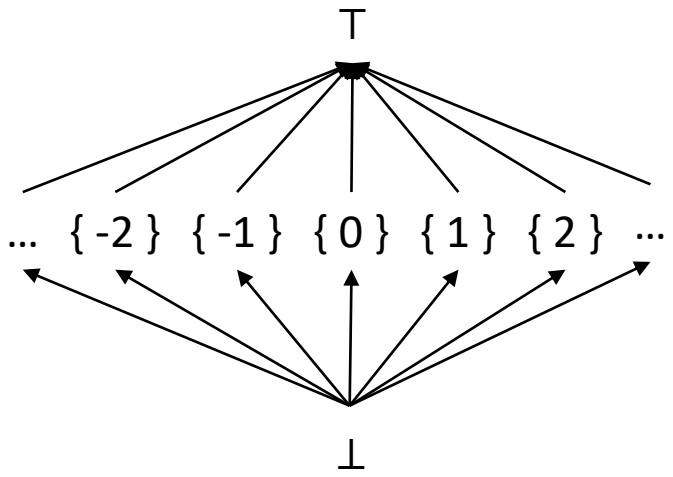
Allows "ranking" fact sets

# Ranking Fact Sets

Dataflow: Formalization

Values form a *lattice*

Values merge *to their least upper bound*




**Before**

| Set of Known Values | We Don't Know |
|---|---|
| {1}, {1,2}, … | |

**After**

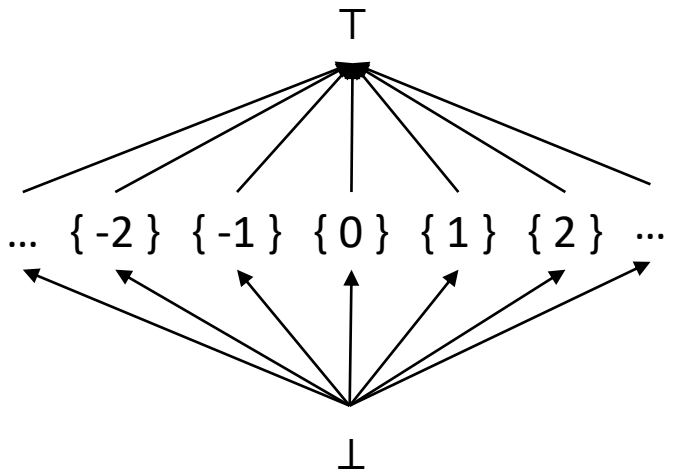
| Single Constant Value | We Don't Know | Could be Anything |
|---|---|---|
| 1, 2, 3, … | ⊥ | ⊤ |

# Reaching a Fixpoint

Dataflow: Formalization

Values form a *lattice*

Values merge *to their least upper bound*

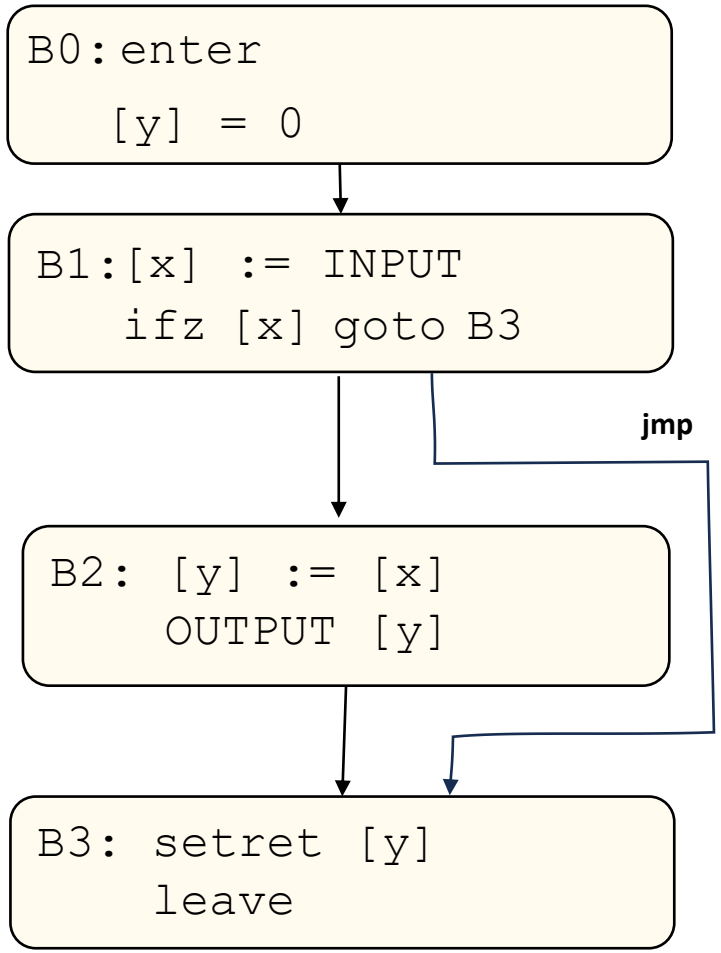



**When the lattice has a finite size:**

- Guarantees termination of the analysis
  - Merges are monotonically non-decreasing
  - Local steps add finite element from the lattice
  - Stop when no set grows

# Incorporating Predicates

## Dataflow: Formalization




| B0 | x | y |
|---|---|---|
| IN | ⊥ | ⊥ |
| OUT | ⊥ | ⊥ |

| B1 | x | y |
|---|---|---|
| IN | ⊥ | ⊥ |
| OUT | ⊥ | ⊥ |

| B2 | x | y |
|---|---|---|
| IN | ⊥ | ⊥ |
| OUT | ⊥ | ⊥ |

| B3 | x | y |
|---|---|---|
| IN | ⊥ | ⊥ |
| OUT | ⊥ | ⊥ |

# Summary

IR Optimization

**Covered some key optimization concepts**

- Inter-block (global) analysis
- Dataflow frameworks:
  - Define fact sets and how they interact

**Next Time – Static Single Assignment (SSA)**

- A program form that eases and enhances optimization