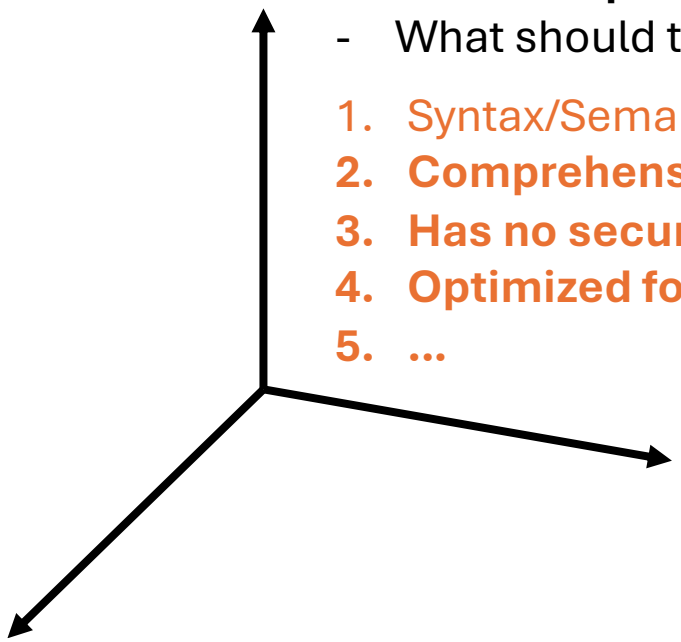
Machine Programming

Lecture 16 – LLM for Software Testing: Static Analysis

Logistics – Week 9

- Assignment 3: Coding LLM Agents
  - https://github.com/machine-programming/assignment-3
  - Fully functional web-app agent. Due: Oct 23 (Thu)
- Forming groups for your final projects!
  - Form a group of 2-3 before This Thursday (~~Oct 19~~ Oct 23)

Module 3: Overview



Behavioral Specification
- What should the program do?

1. Syntax/Semantics/Functional correctness
2. **Comprehensive test coverage**
3. **Has no security flaw**
4. **Optimized for runtime speed**
5. **…**

Structural Specification
- What is the space of the programs?

General Purpose Programming Language
Python / Java / C / Rust / …

Domain Specific Languages
SQL / LEAN / ROCQ / DATALOG / PDDL / …

Synthesis Strategy
- How do we find such a program?

Enumeration: Bottom-up and Top-Down

Programming with Large Language Models
- Next token prediction, prompting, controlled decoding
- Iterative refinement, agentic frameworks and tool use
- Pre-training, fine-tuning, reinforcement learning

Fixed

Software Analysis

Testing and Dynamic Analysis

Forms of Software Analysis

Static Analysis

Examine code without concrete execution

- Taint analysis
- Reachability analysis
- Abstract interpretation
- Symbolic execution
- …

Dynamic Analysis

Observes program behavior while executing

- Unit testing
- Fuzzing
- Property-based testing
- Penetration testing
- …

Dynamic Testing

Unit Testing

Human-written tests in the form of concrete input-output

```
assert f(2) == 4
```

Dynamic Testing

Unit Testing

Human-written tests in the form of concrete input-output

```
assert f(2) == 4
```

Fuzzing

Randomly generating test inputs

```
for i in 0..10^4:
  x = gen_input()
  f(x)
```

Dynamic Testing

Unit Testing

Human-written tests in the form of concrete input-output

```
assert f(2) == 4
```

Fuzzing

Randomly generating test inputs

```
for i in 0..10^4:
  x = gen_input()
  f(x)
```

Property-based testing

Randomly generating test inputs, but with properties checking outputs

```
for i in 0..10^4:
  x = gen_input()
  y = f(x)
  assert property(x,y)
```

Dynamic Testing

Unit Testing

Human-written tests in the form of concrete input-output

```
assert f(2) == 4
```

Fuzzing

Randomly generating test inputs

```
for i in 0..10^4:
  x = gen_input()
  f(x)
```

Property-based testing

Randomly generating test inputs, but with properties checking outputs

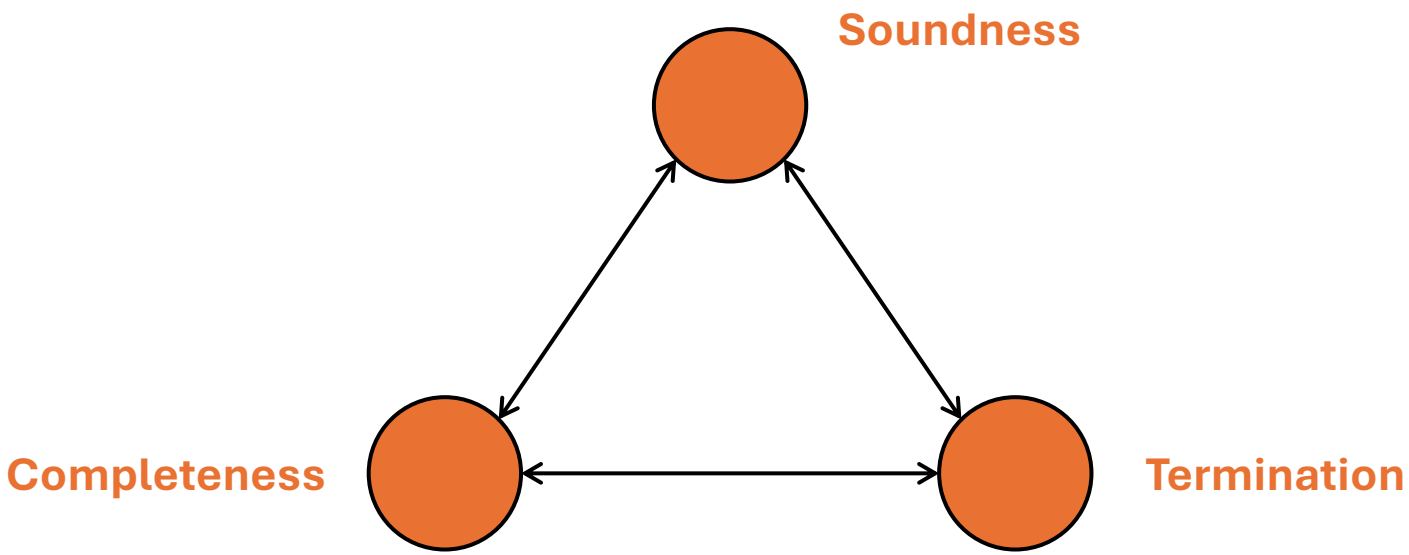
```
for i in 0..10^4:
  x = gen_input()
  y = f(x)
  assert property(x,y)
```

Penetration testing

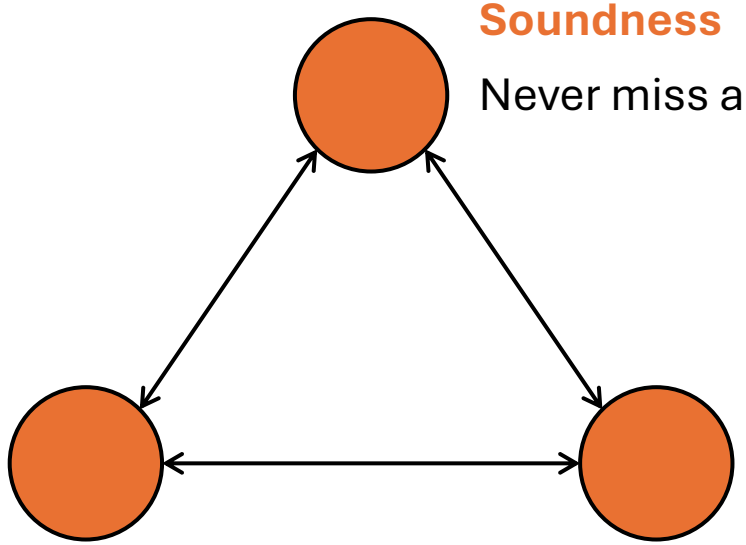
Crafting adversarial inputs to trigger vulnerabilities rather than getting outputs

```
try:
  f(very_bad_input)
  assert false
except:
  assert no_bad_behavior
```

Software Testing: Impossible Triangle



Software Testing: Impossible Triangle



Soundness

Never miss a real bug

Completeness

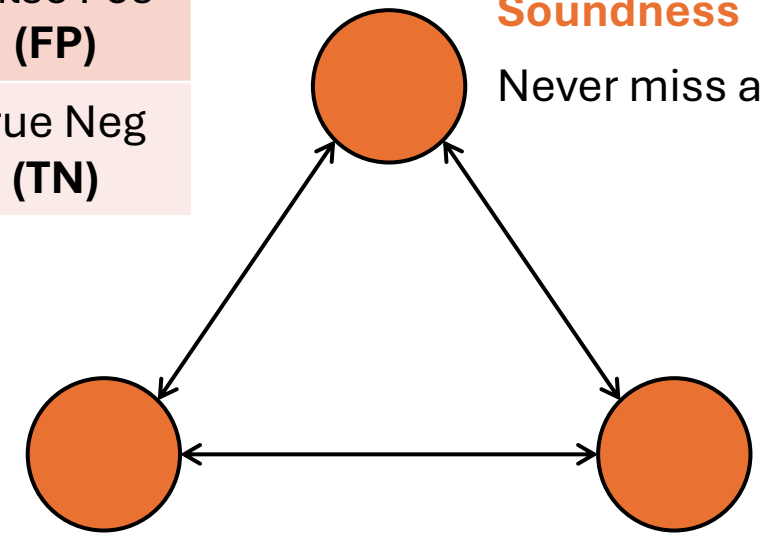
Never raise a false alarm

Termination

Always finish the analysis within finite time

Software Testing: Impossible Triangle

	Real Bug	False Bug
Reported (positive)	True Pos **(TP)**	False Pos **(FP)**
Not Reported (negative)	False Neg **(FN)**	True Neg **(TN)**



Soundness

Never miss a real bug

Completeness

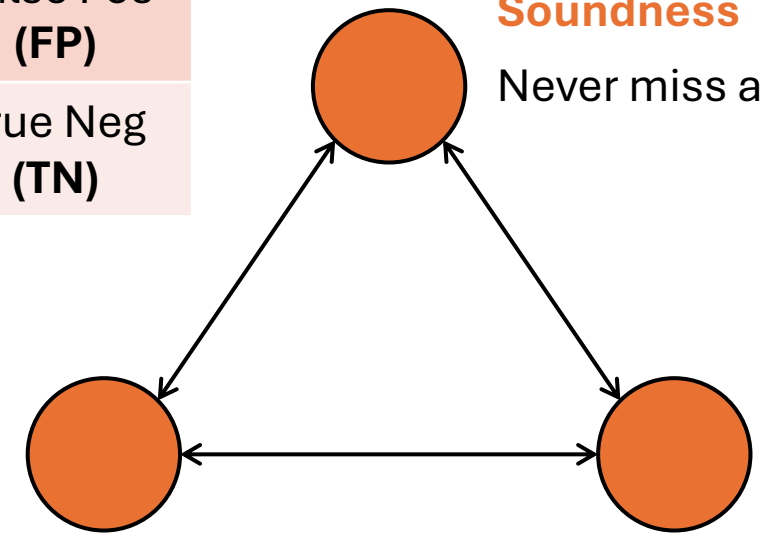
Never raise a false alarm

Termination

Always finish the analysis within finite time

Software Testing: Impossible Triangle

	Real Bug	False Bug
Reported (positive)	True Pos **(TP)**	False Pos **(FP)**
Not Reported (negative)	False Neg **(FN)**	True Neg **(TN)**



Soundness #FN = 0

Never miss a real bug

#FP = 0 **Completeness**

Never raise a false alarm

Termination

Always finish the analysis within finite time

Software Testing: Impossible Triangle

Dynamic Analysis:

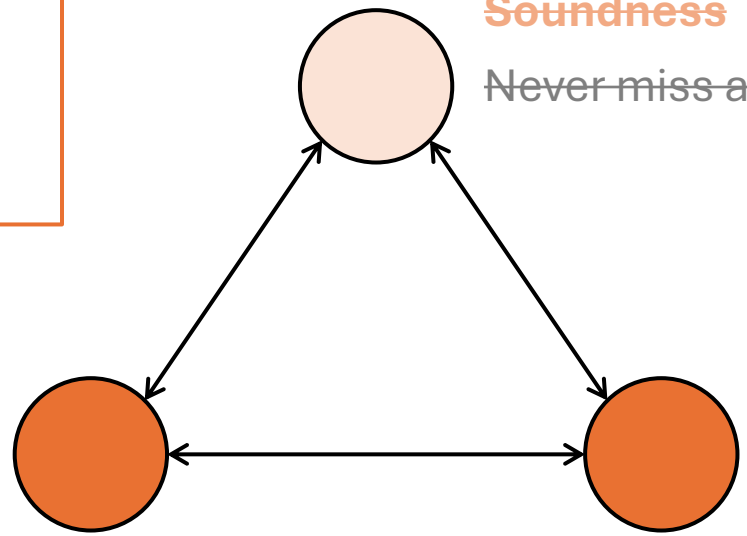
Unit Testing

- (assuming tests are correct)

Fuzzing

- (assuming harness is real)

The unit test cases might not be comprehensive enough.



~~**Soundness**~~

~~Never miss a real bug~~

Completeness

Never raise a false alarm

Termination

Always finish the analysis within finite time

Program – Line coverage: 5/5 (100%)

```python
def calculate_interest(balance: float, months: int) -> float:
  rate = 0.02 # 2% per month
  if months > 12:
    rate = 0.03 # higher rate for long-term deposits
  return balance * rate / months
```

Test 1

```python
def test_short_term_interest():
  assert calculate_interest(1000, 6) == pytest.approx(3.33)
```

Test 2

```python
def test_long_term_interest():
  assert calculate_interest(1000, 24) == pytest.approx(1000 * 0.03 / 24)
```

Test 3 (Assume that this test case is not there)

```python
def test_zero_months_divide_by_zero(): # covers implicit error path
  assert calculate_interest(1000, 0) == 0
```

Software Testing: Impossible Triangle

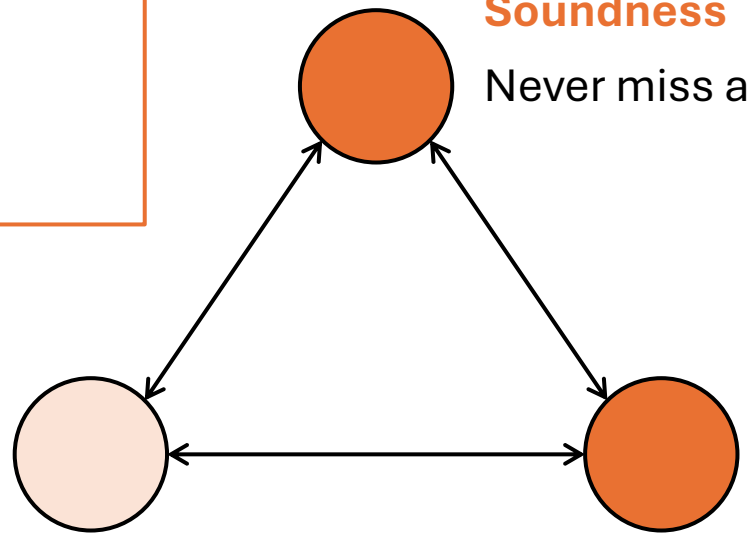
Static Analysis:

Taint Analysis

- (assuming correct specs)

Abstract Interpretation

- (assuming correct domains)



Soundness

Never miss a real bug

~~**Completeness**~~

~~Never raise a false alarm~~

Termination

Always finish the analysis within finite time

May report an alarm for developer confirmation, and that is rejected

Software Testing: Impossible Triangle

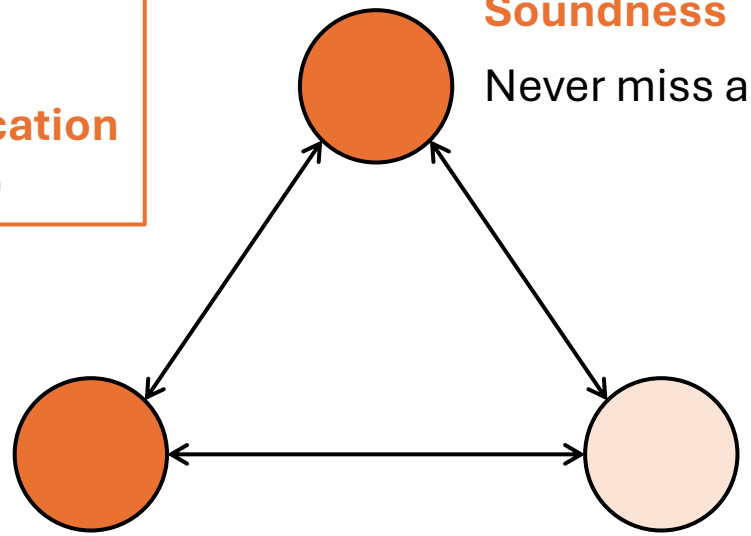
Formal Verification:

Halting Problem

- Is the program terminating?

Functional Correctness Verification

- (assuming specs are correct)



Soundness

Never miss a real bug

Completeness

Never raise a false alarm

~~Termination~~

~~Always finish the analysis within finite time~~

It may be impossible for a verifier to terminate

Software Analysis

Static Analysis

Software Testing: Impossible Triangle

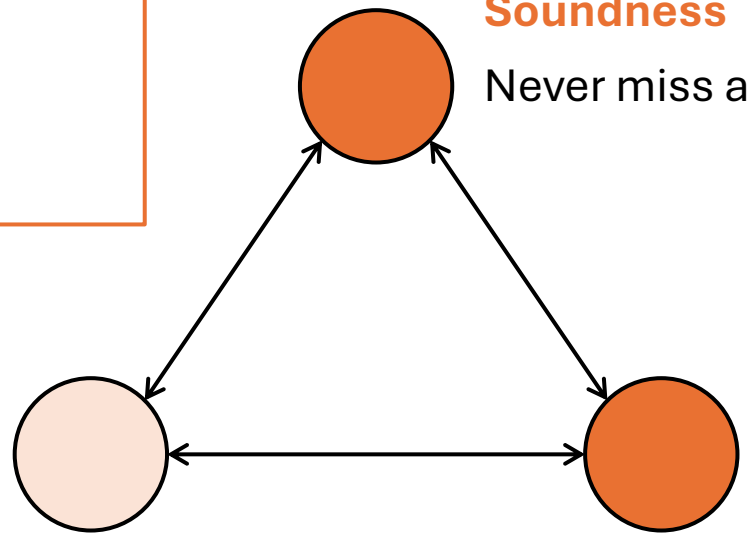
Static Analysis:

Taint Analysis

- (assuming correct specs)

Abstract Interpretation

- (assuming correct domains)



Soundness

Never miss a real bug

~~**Completeness**~~

~~Never raise a false alarm~~

Termination

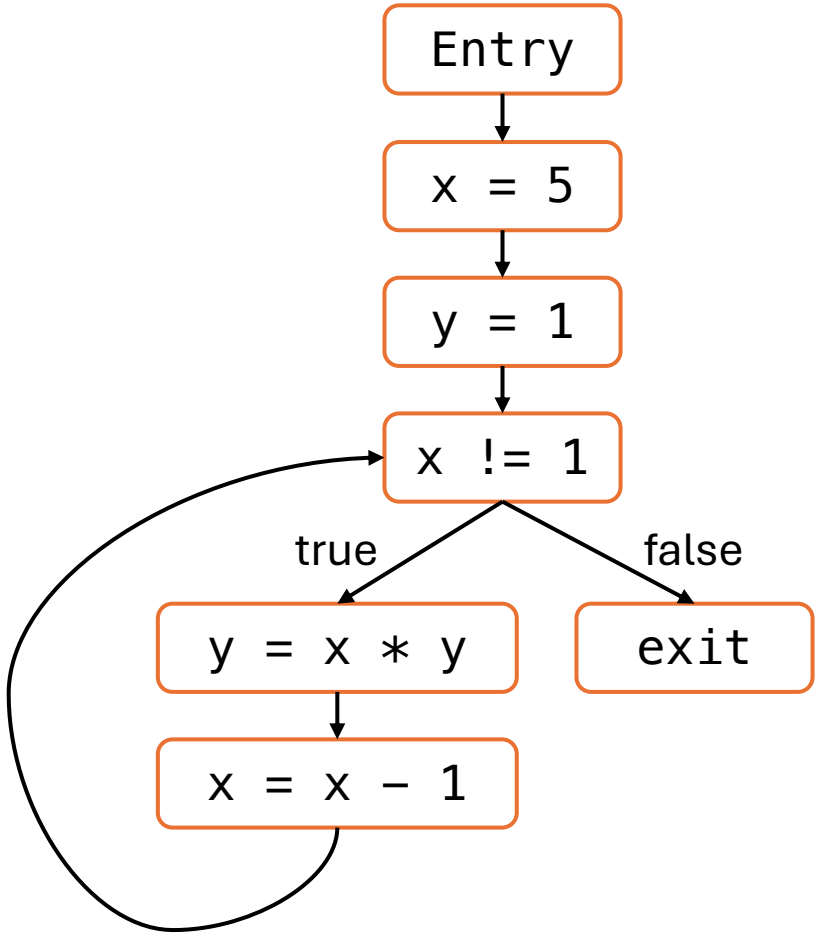
Always finish the analysis within finite time

May report an alarm for developer confirmation, and that is rejected

Control-Flow Graphs

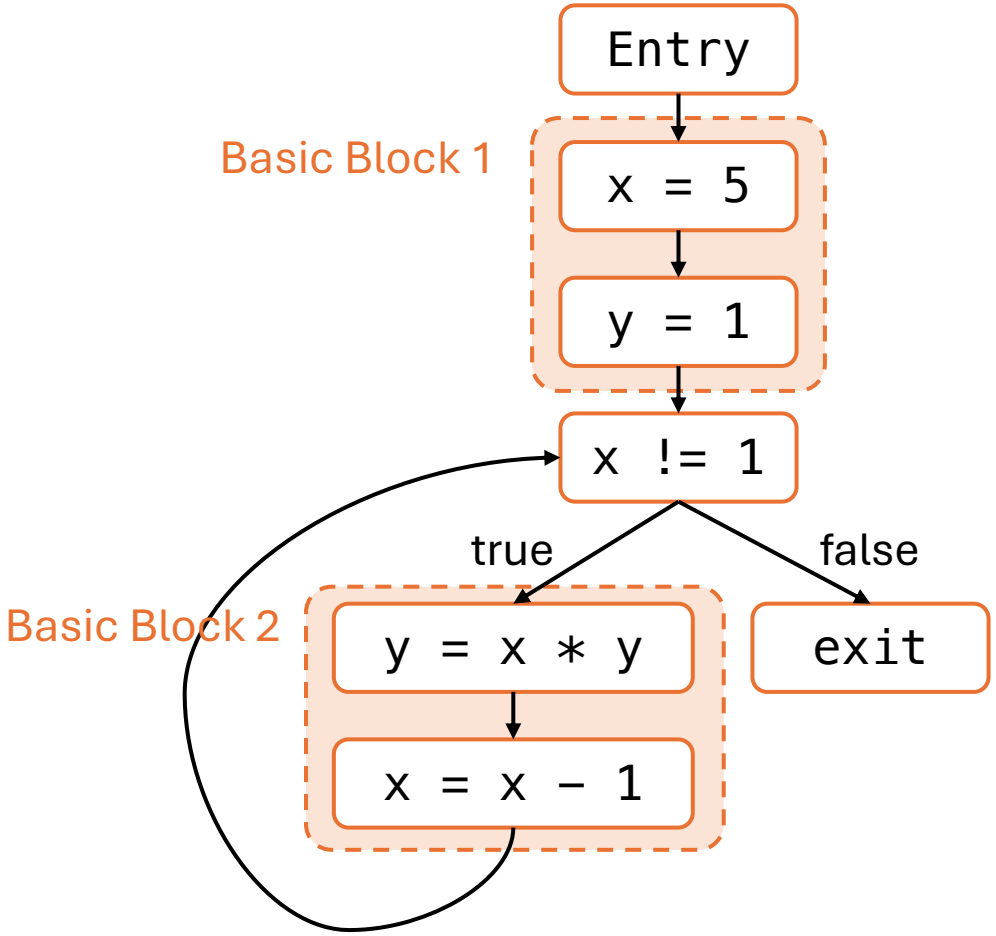
- Premise:
  - An abstract representation of imperative programs
  - Illustrating how “control” (execution) flows through the code
- A graph representation
  - Summarizes the flow of control in all possible runs of the program
- Goal:
  - Help all sorts of static analysis by providing the birds eye view

Control-Flow Graphs



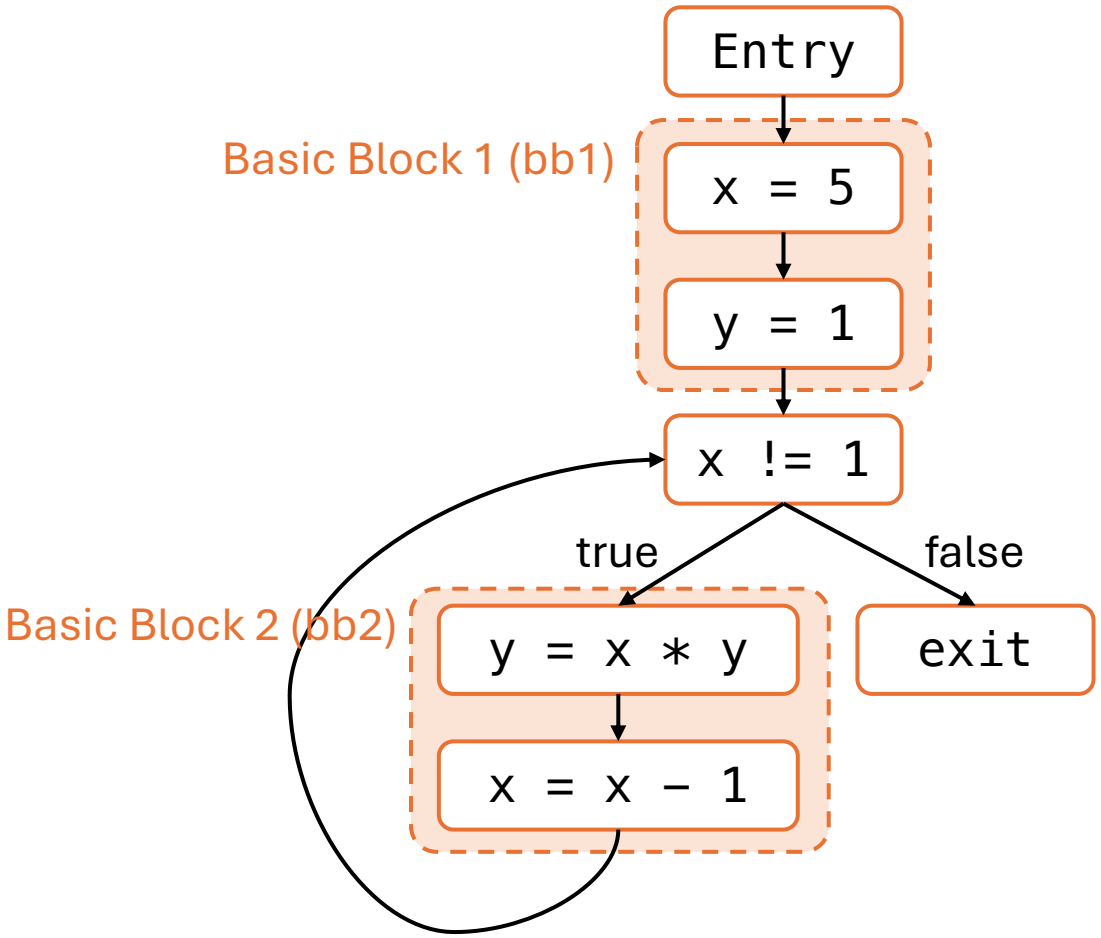

```
int x = 5;
int y = 1;
while (x != 1) {
  y = x * y;
  x = x - 1;
}
```

Control-Flow Graphs




```
int x = 5;
int y = 1;
while (x != 1) {
  y = x * y;
  x = x - 1;
}
```

Control-Flow Graphs

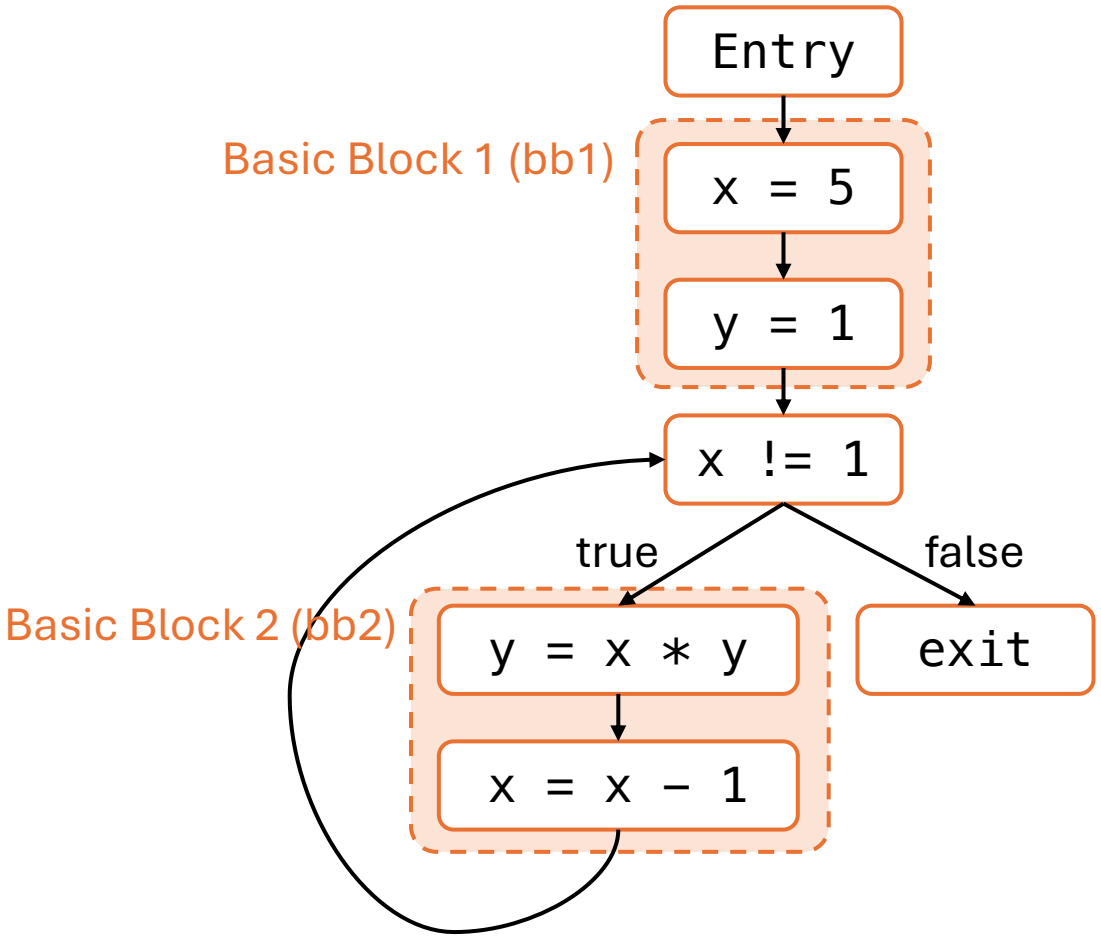


bb1

bb2

```
int x = 5;
int y = 1;
while (x != 1) {
  y = x * y;
  x = x - 1;
}
```

Control-Flow Graphs: Traces



bb1
cond
bb2

```
int x = 5;
int y = 1;
while (x != 1) {
  y = x * y;
  x = x - 1;
}
```

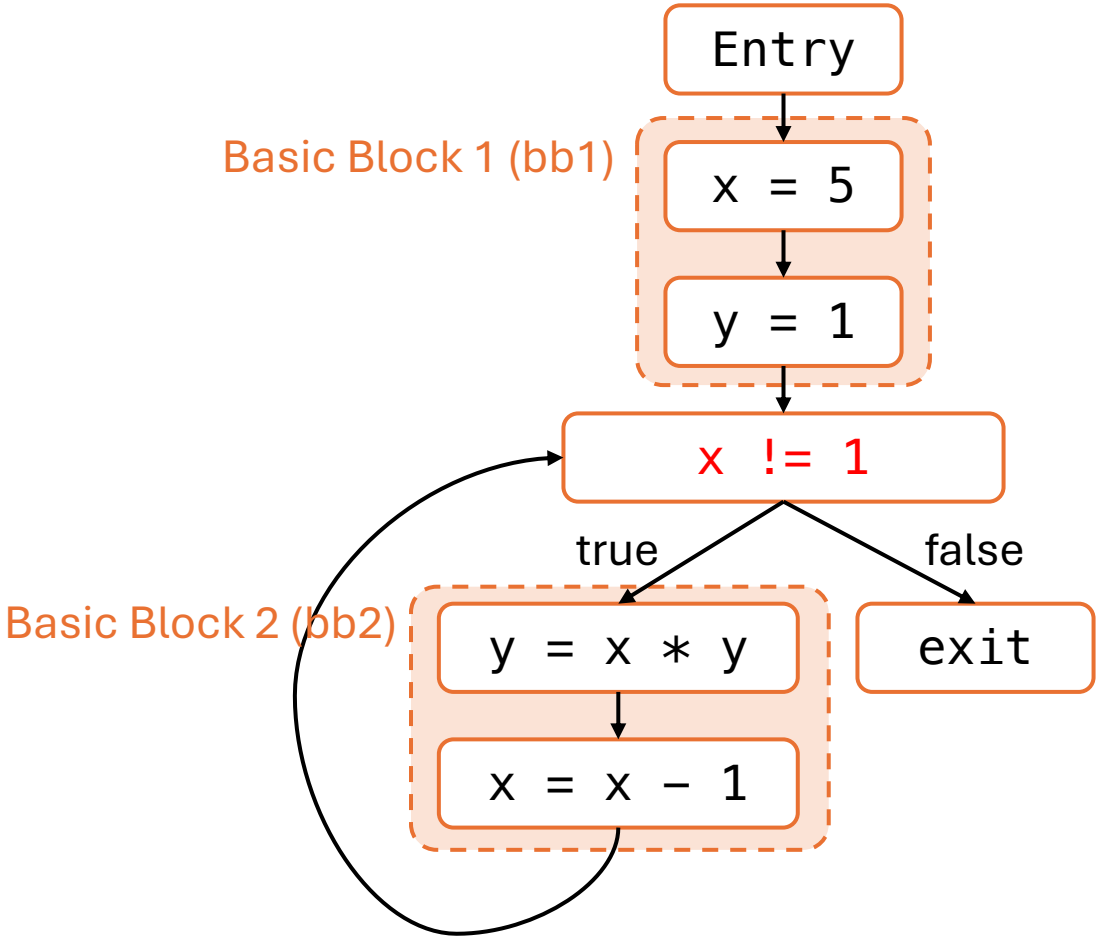
Execution Trace

Entry → bb1 → cond
→ bb2 → cond // (x == 5)
→ bb2 → cond // (x == 4)
→ bb2 → cond // (x == 3)
→ bb2 → cond // (x == 2)
→ bb2 → cond // (x == 1)
→ exit

Dataflow Analysis

- Building on control-flow analysis
  - Abstracts away control-flow conditions
- Considers all paths possible in actual runs (sound)
- Including paths that are never realizable (incomplete)

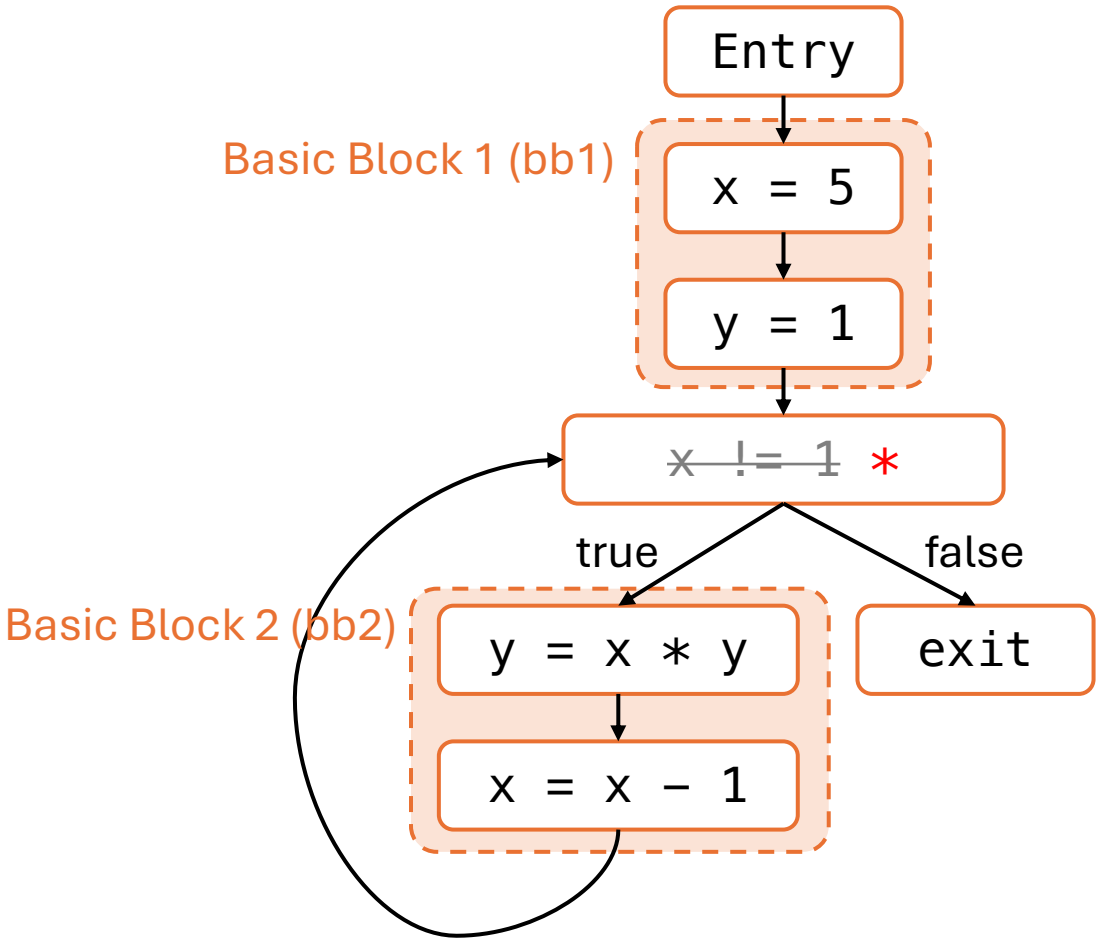
Abstracting Control-Flow Graphs

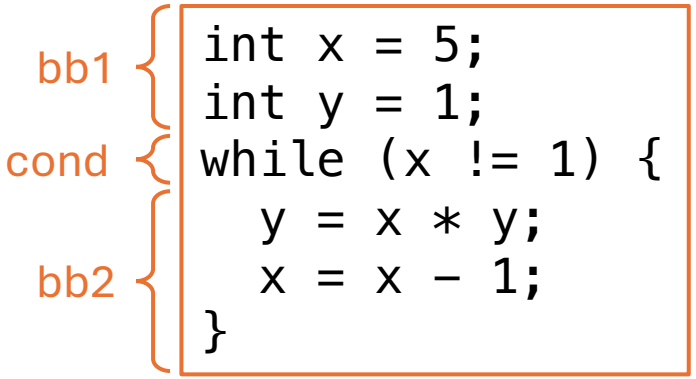


bb1
cond
bb2

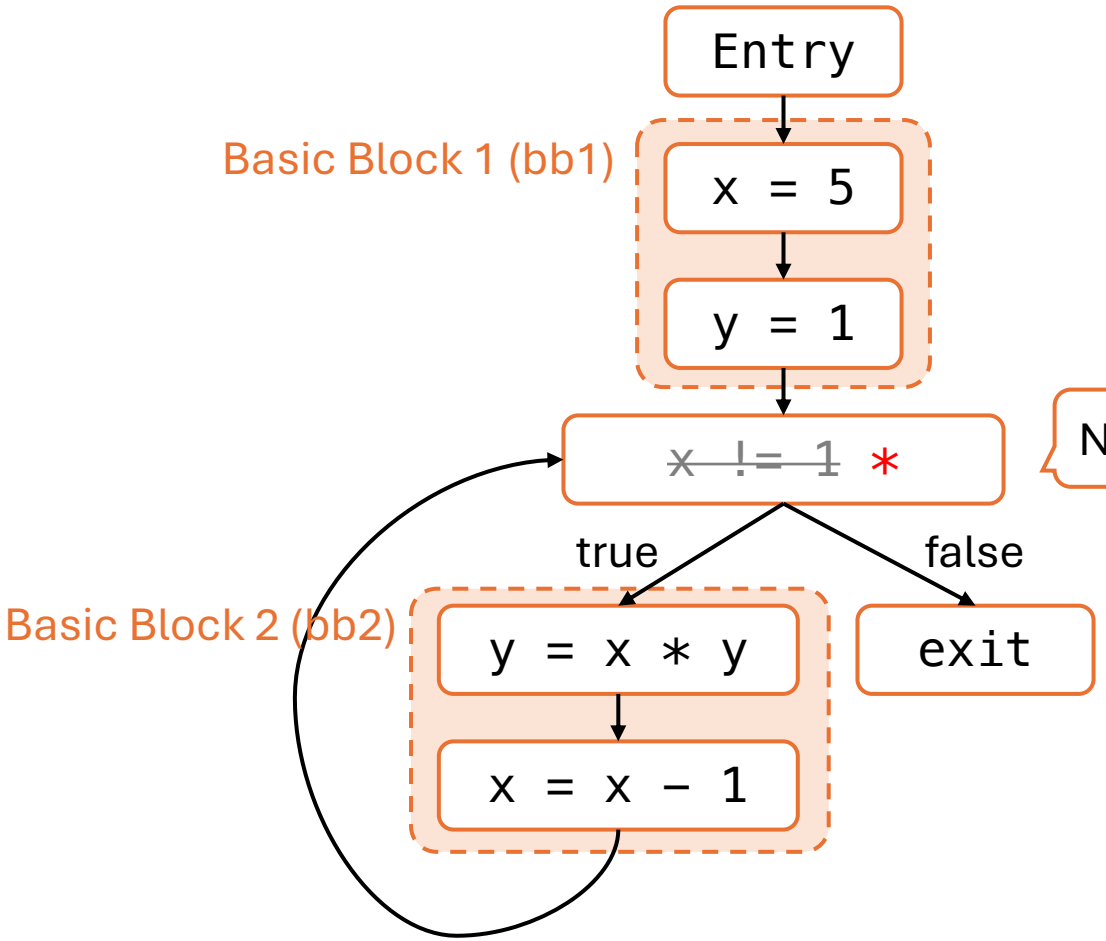
```
int x = 5;
int y = 1;
while (x != 1) {
  y = x * y;
  x = x - 1;
}
```

Abstracting Control-Flow Graphs





Abstracting Control-Flow Graphs



bb1
cond
bb2

```
int x = 5;
int y = 1;
while (x != 1) {
  y = x * y;
  x = x - 1;
}
```

Non-deterministic

Potential Traces

Entry → bb1 → cond → bb2 → cond → exit

Entry → bb1 → cond → exit

Modern Dataflow Analyses

Interval Analysis

- Check memory safety
  (integer overflows, buffer overruns, ...)

Taint Analysis

- Check information flow
  (Sensitive data leak, code injection, ...)

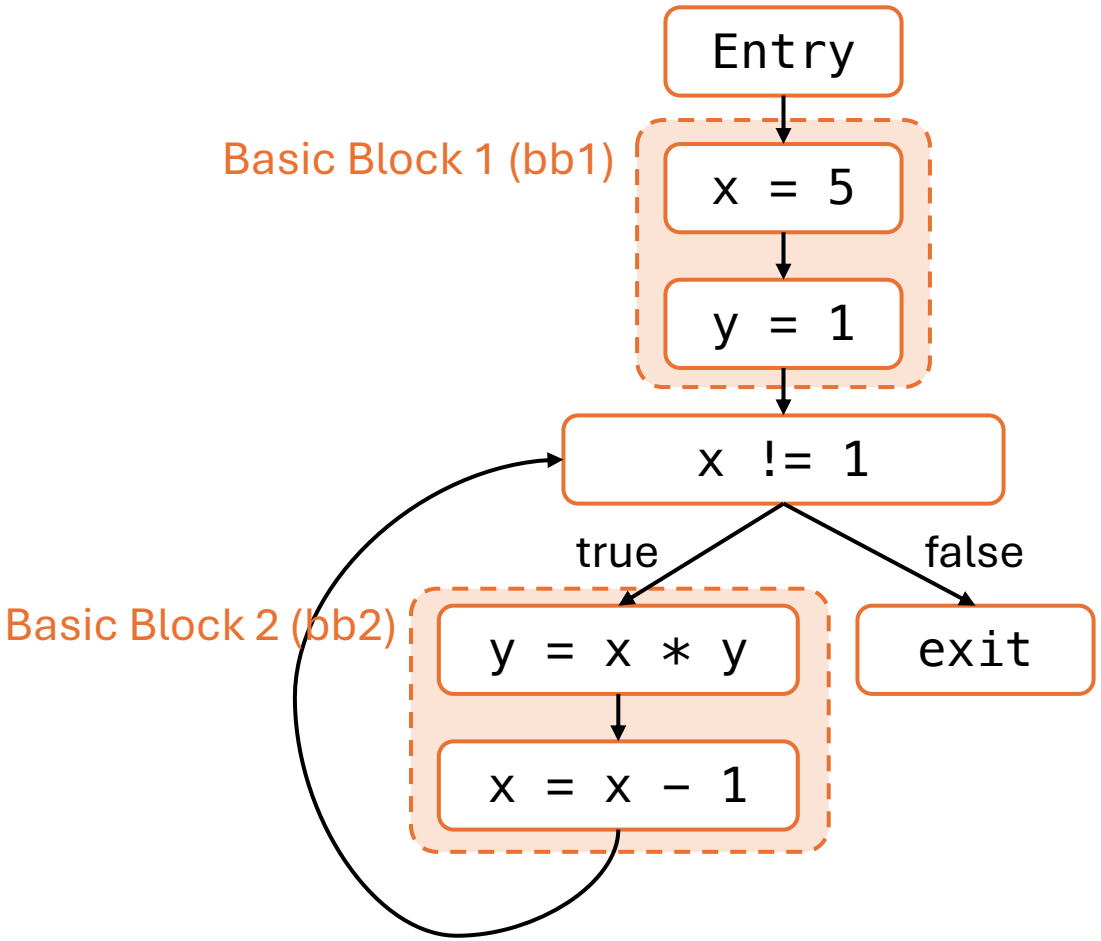
Type-State Analysis

- Temporal safety properties
  (APIs of protocols, libraries, ...)

Concurrency Analysis

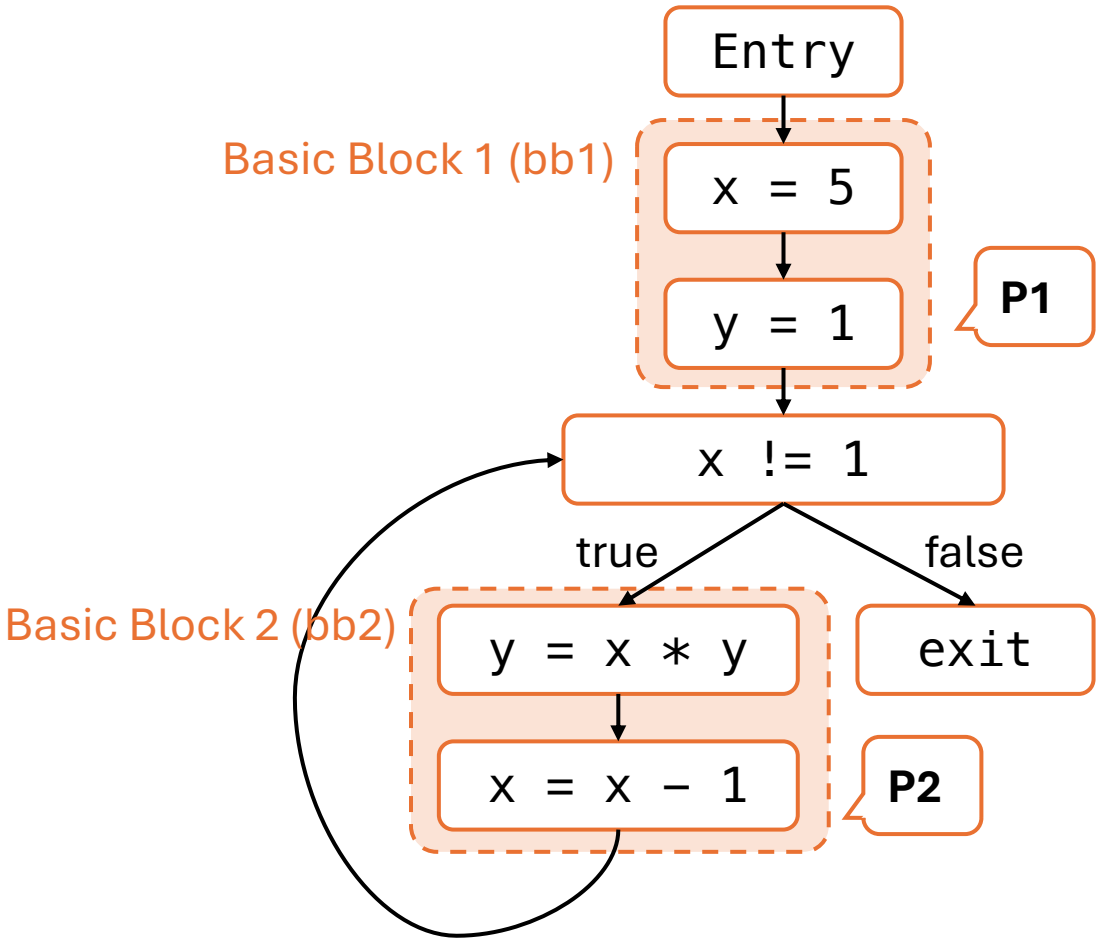
- Concurrency safety properties
  (dataraces, deadlocks, ...)

Reaching Definitions Analysis



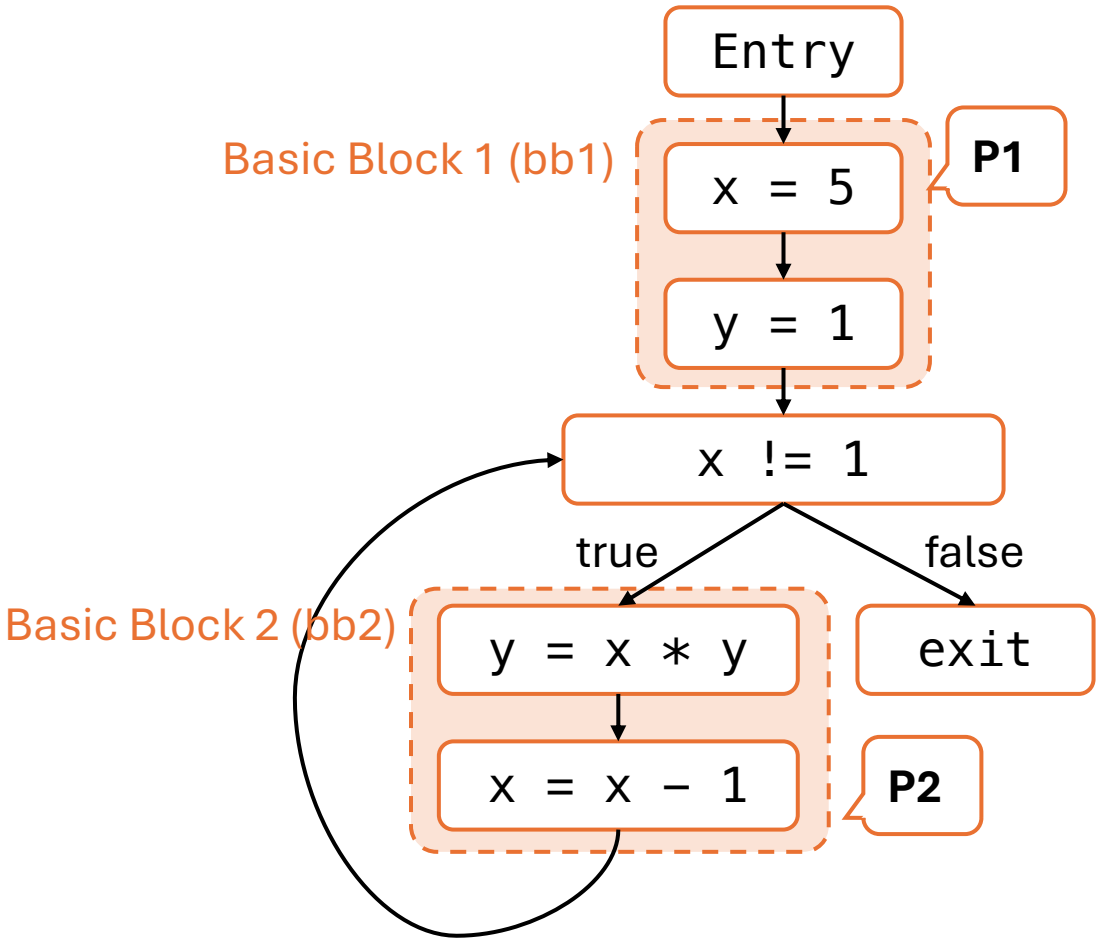
- Goal:
  - Determine whether a program point containing a piece of data (definition of a variable) can flow to another program point
- Reduced to…
  - Path finding problem in a graph

Reaching Definitions Analysis



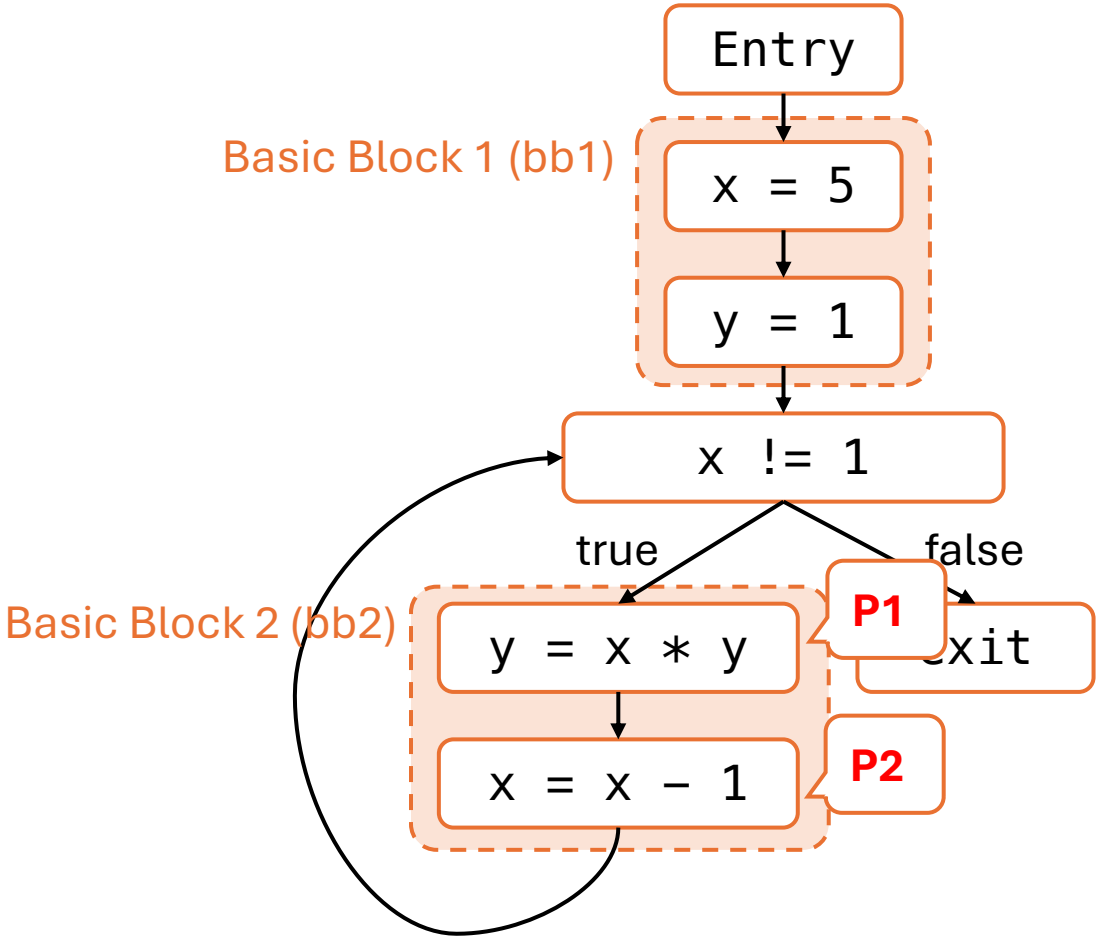
- Goal:
  - Determine whether a program point containing a piece of data (definition of a variable) can flow to another program point
- Reduced to…
  - Path finding problem in a graph

Reaching Definitions Analysis



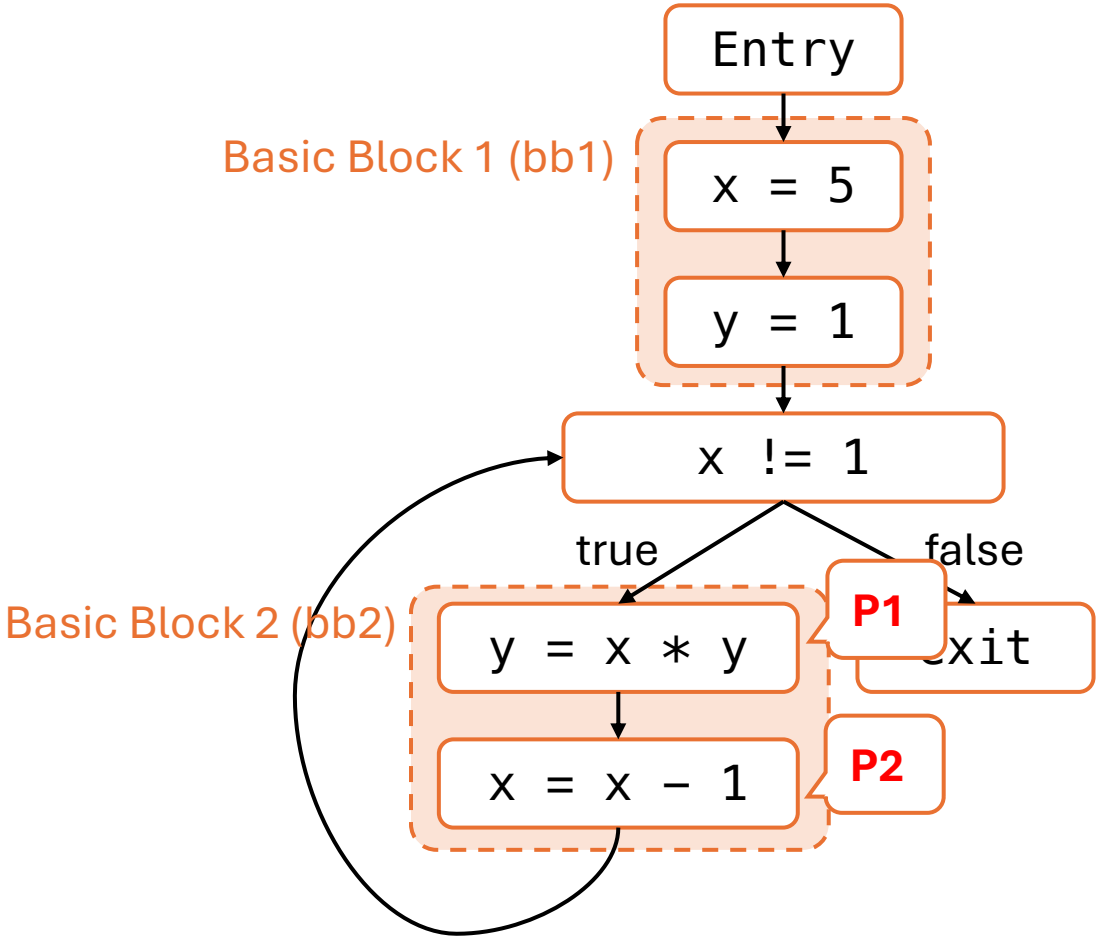
- Goal:
  - Determine whether a program point containing a piece of data (definition of a variable) can flow to another program point
- Reduced to…
  - Path finding problem in a graph
- Example:
  - Does information defined in **P1** flow to **P2**?

Reaching Definitions Analysis: Quiz



- Does information defined in **P1** flow to **P2**?

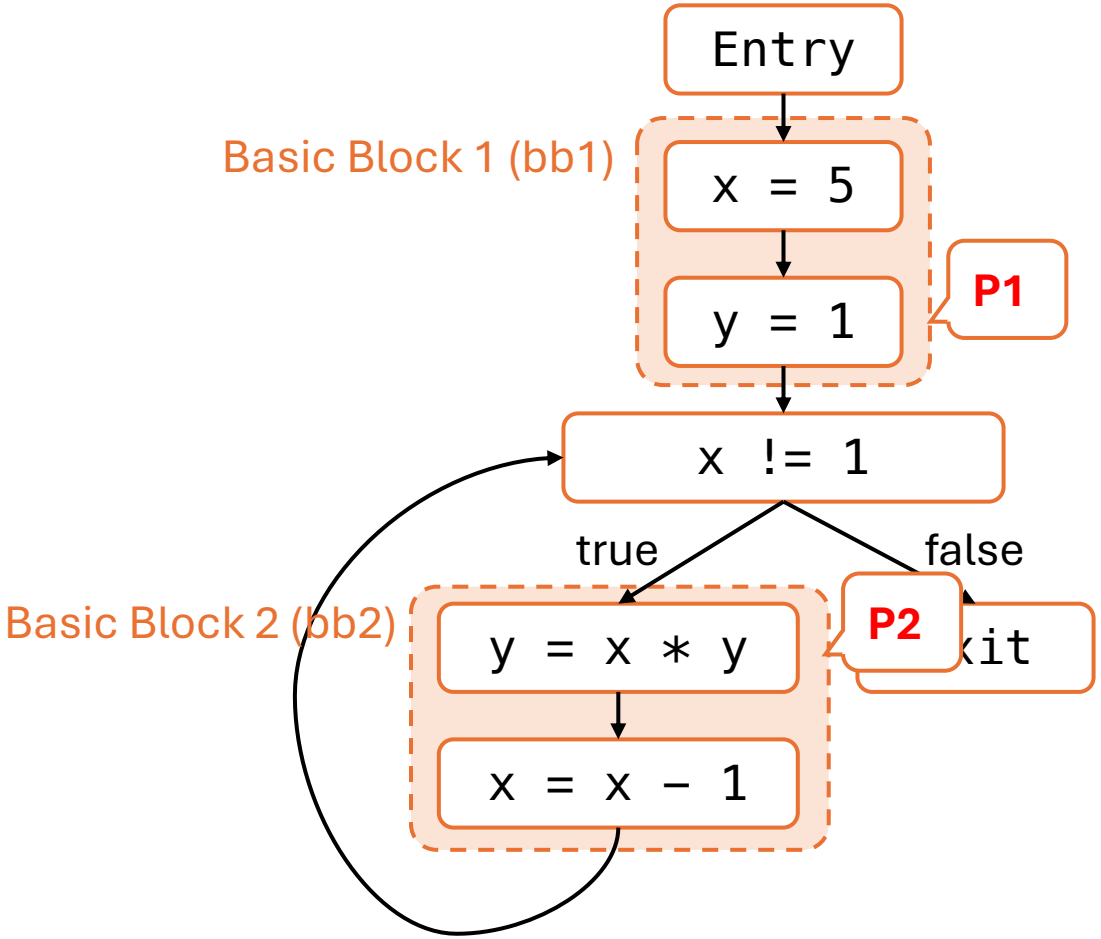
Reaching Definitions Analysis: Quiz



- Does information defined in **P1** flow to **P2**?

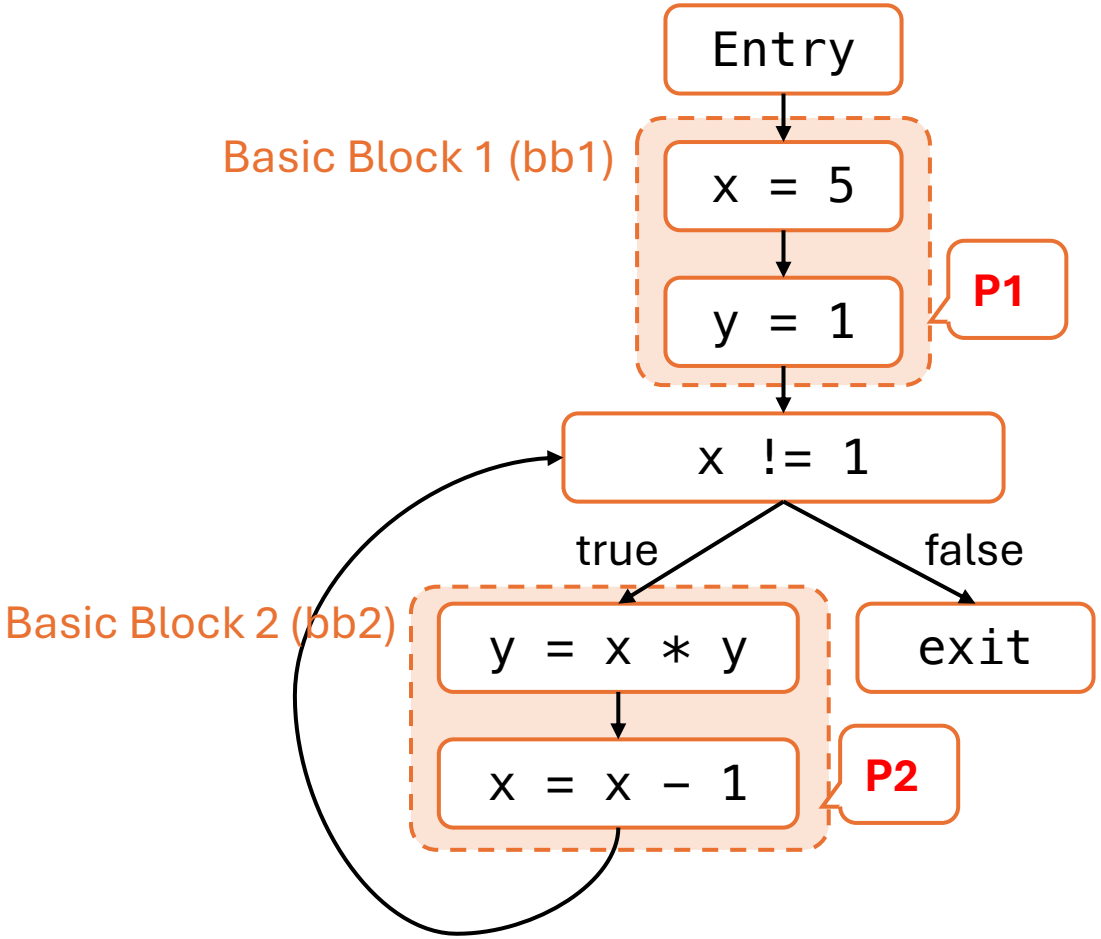
  YES

Reaching Definitions Analysis: Quiz



- Does information defined in **P1** flow to **P2**?

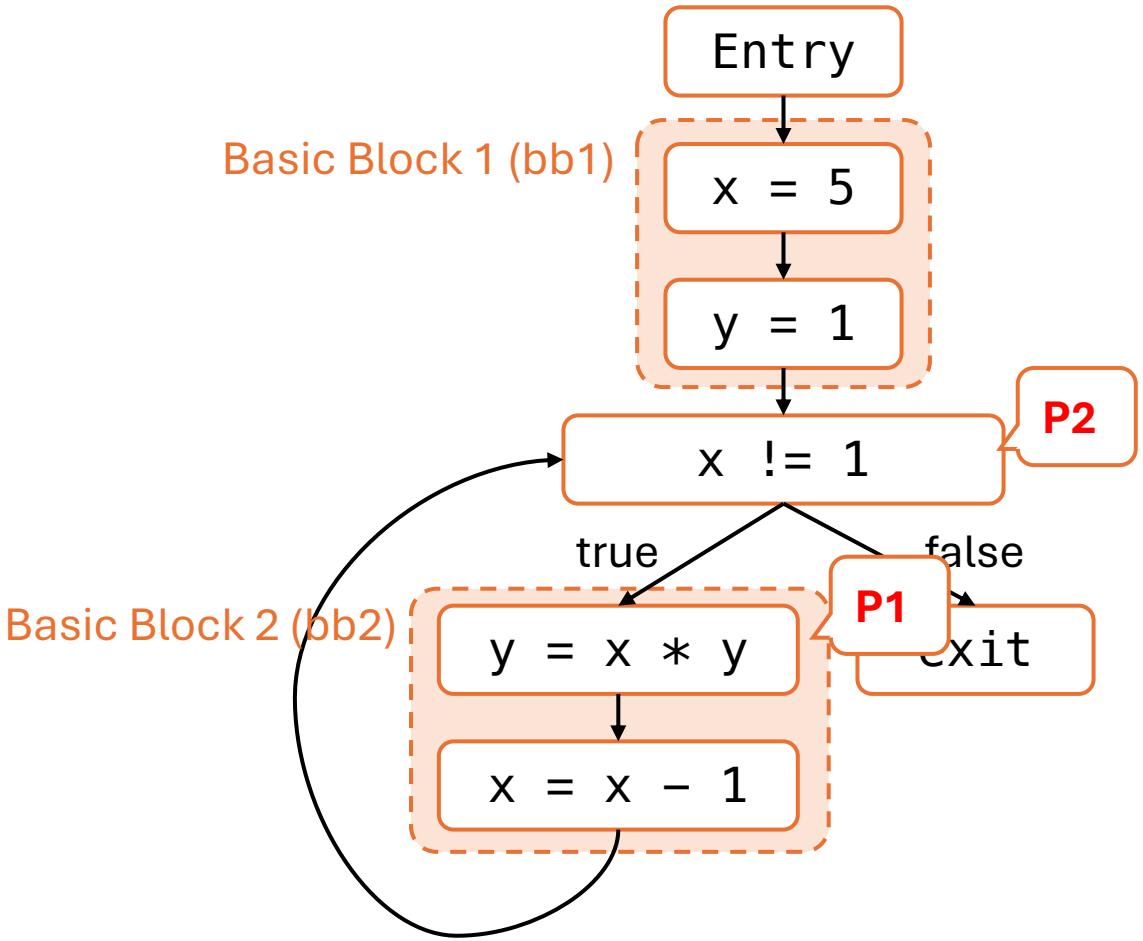
Reaching Definitions Analysis: Quiz



- Does information defined in **P1** flow to **P2**?

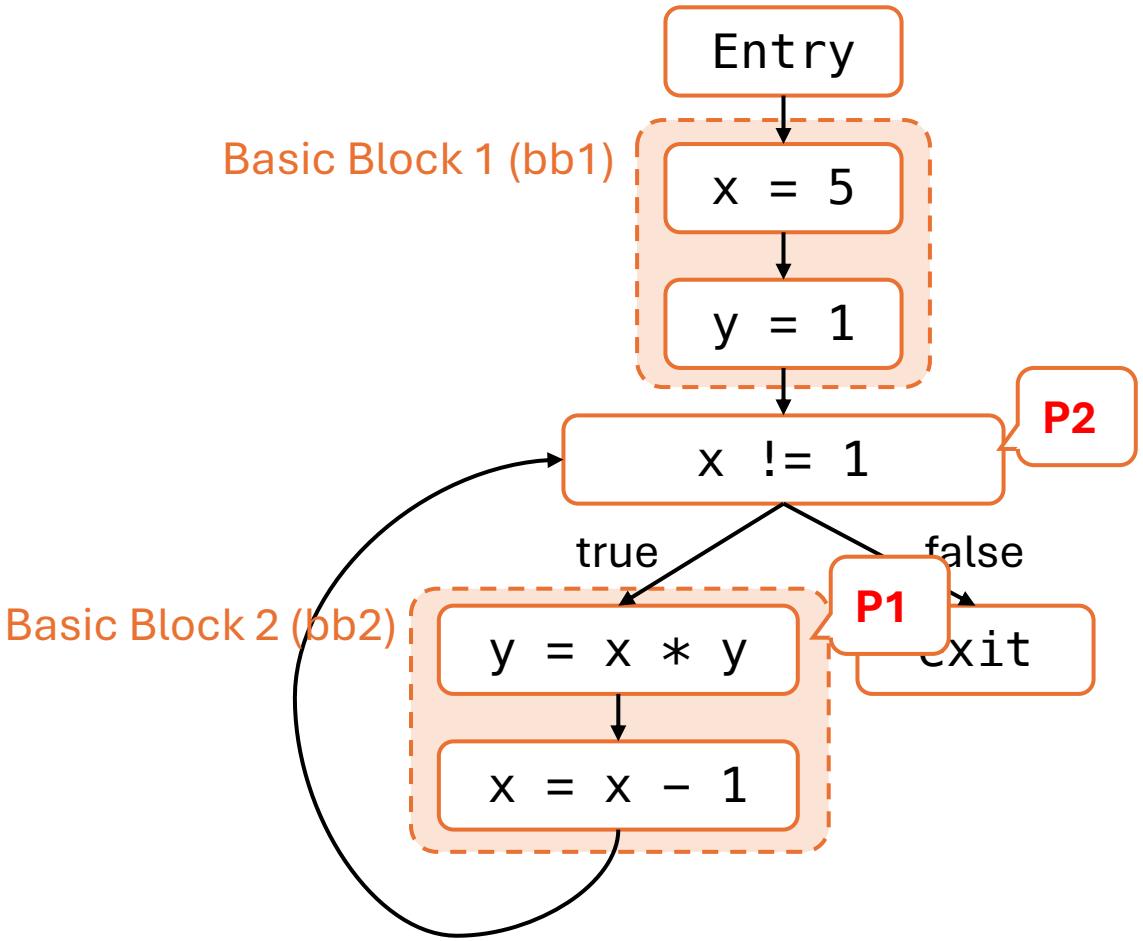
NO

Reaching Definitions Analysis: Quiz



- Does information defined in **P1** flow to **P2**?

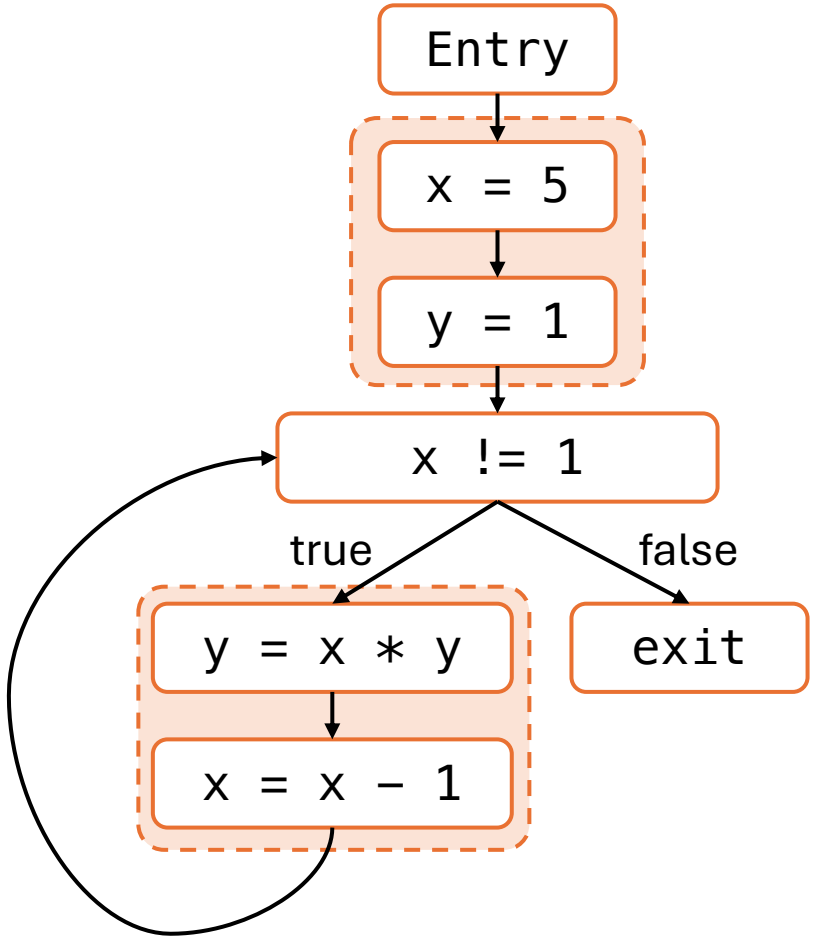
Reaching Definitions Analysis: Quiz



- Does information defined in **P1** flow to **P2**?

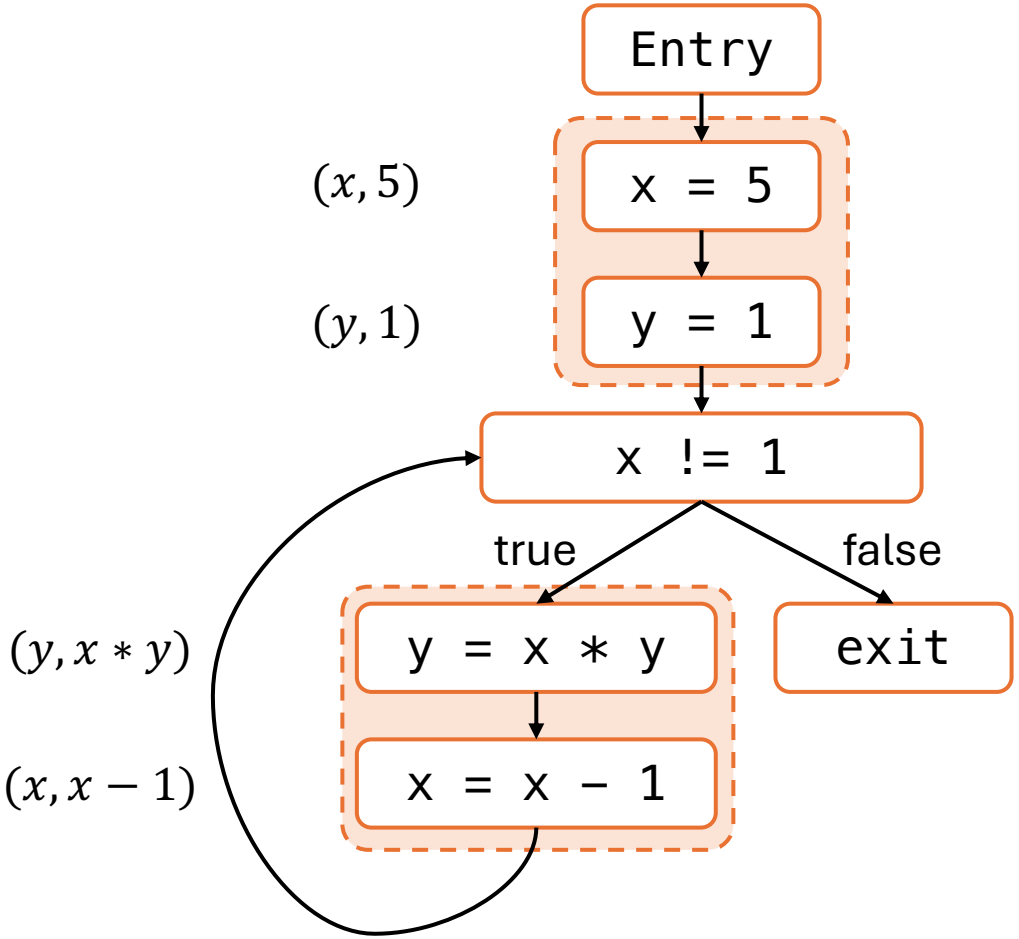
  YES

How to Compute Reaching Definitions?



- For each node n, we denote
  - $\mathrm{IN}(n)$ =nodes which can reach n
  - $\mathrm{OUT}(n)$ =nodes which can go beyond n
- Dataflow analysis computes $\mathrm{IN}(n)$ and $\mathrm{OUT}(n)$ for each node
- Chaotic Iteration:
  - Starting from them being empty
  - Repeat operations to populate them
  - Until the two sets stop changing

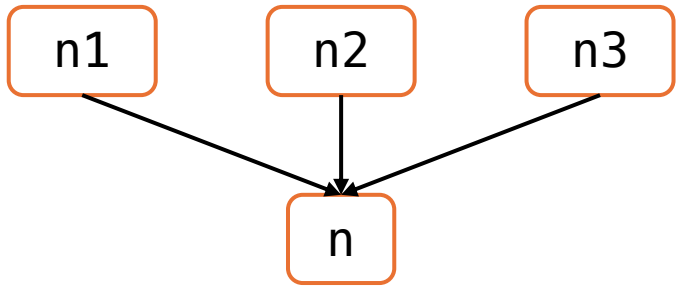
How to Compute Reaching Definitions?



- For each node n, we denote
  - $\mathrm{IN}(n)$ =nodes which can reach n
  - $\mathrm{OUT}(n)$ =nodes which can go beyond n
- Dataflow analysis computes $\mathrm{IN}(n)$ and $\mathrm{OUT}(n)$ for each node
- Chaotic Iteration:
  - Starting from them being empty
  - Repeat operations to populate them
  - Until the two sets stop changing

How to Compute? Flow-in

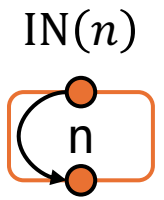
$$\mathrm{IN}(n) = \bigcup_{m \in \mathrm{predecessors}(n)} \mathrm{OUT}(m)$$



$$\mathrm{IN}(n) = \mathrm{OUT}(n_1) \cup \mathrm{OUT}(n_2) \cup \mathrm{OUT}(n_3)$$

How to Compute? Flow-out

$$\mathrm{OUT}(n) = \big(\mathrm{IN}(n) - \mathrm{KILL}(n)\big) \cup \mathrm{GEN}(n)$$



n: `y = x * y`

$\mathrm{GEN}(n) = \{(y, n)\}$

$\mathrm{KILL}(n) = \{(y, 5)\}$

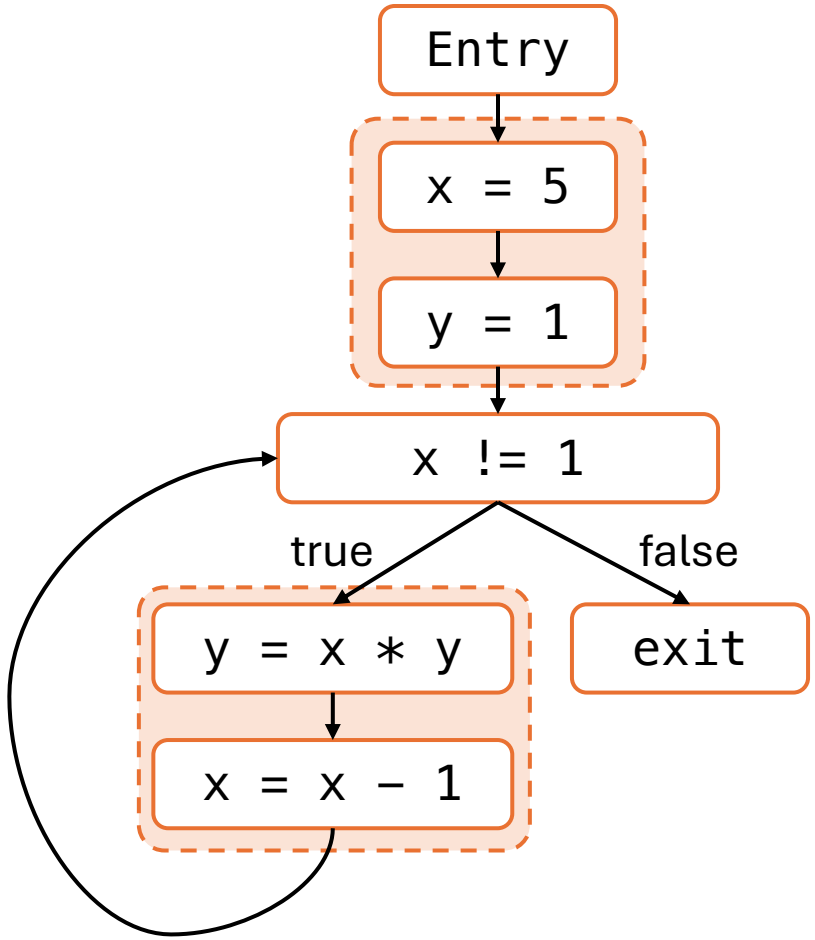
n: `x != 1`

$\mathrm{GEN}(n) = \{\}$

$\mathrm{KILL}(n) = \{\}$

Chaotic Iteration for Dataflow Analysis

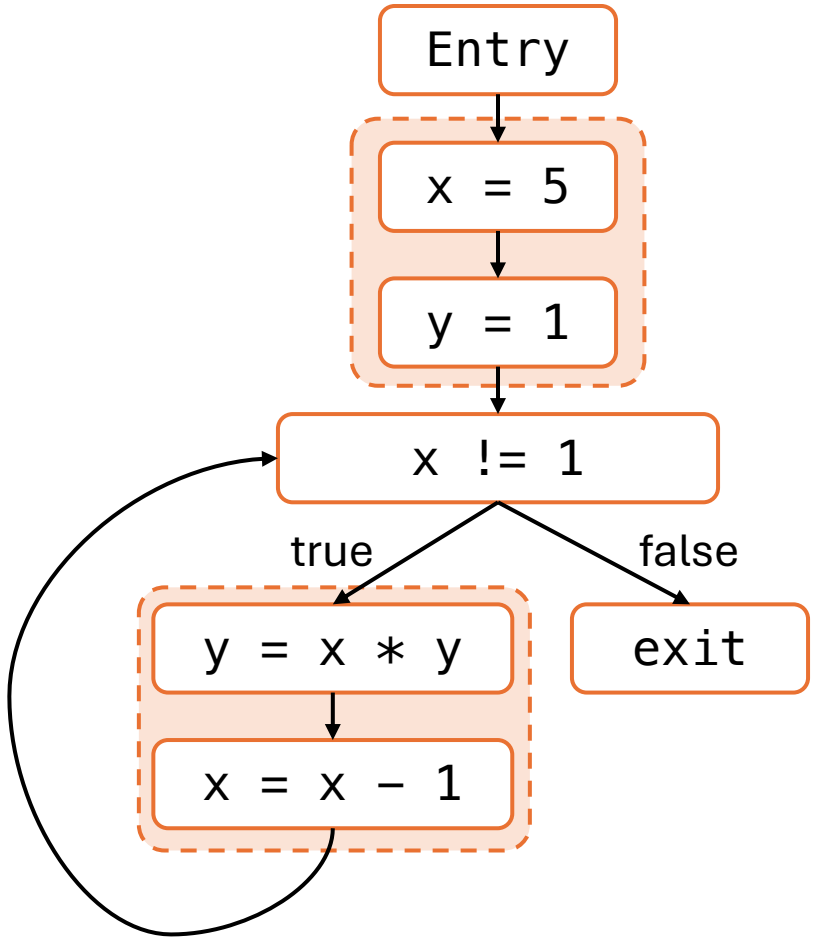
- for (each node n in control-flow graph):
  - $\mathrm{IN}(n) = \{\}$
  - $\mathrm{OUT}(n) = \{\}$
  - $\mathrm{GEN}(n) = \{(v, n)\}$ if n is an assignment else $\{\}$
- repeat:
  - for (each node n):
    - $\mathrm{IN}(n)$ = flow-in(n)
    - $\mathrm{OUT}(n)$ = flow-out(n)
- until all the $\mathrm{IN}(n)$ and $\mathrm{OUT}(n)$ stops changing



Chaotic Iteration for Dataflow Analysis

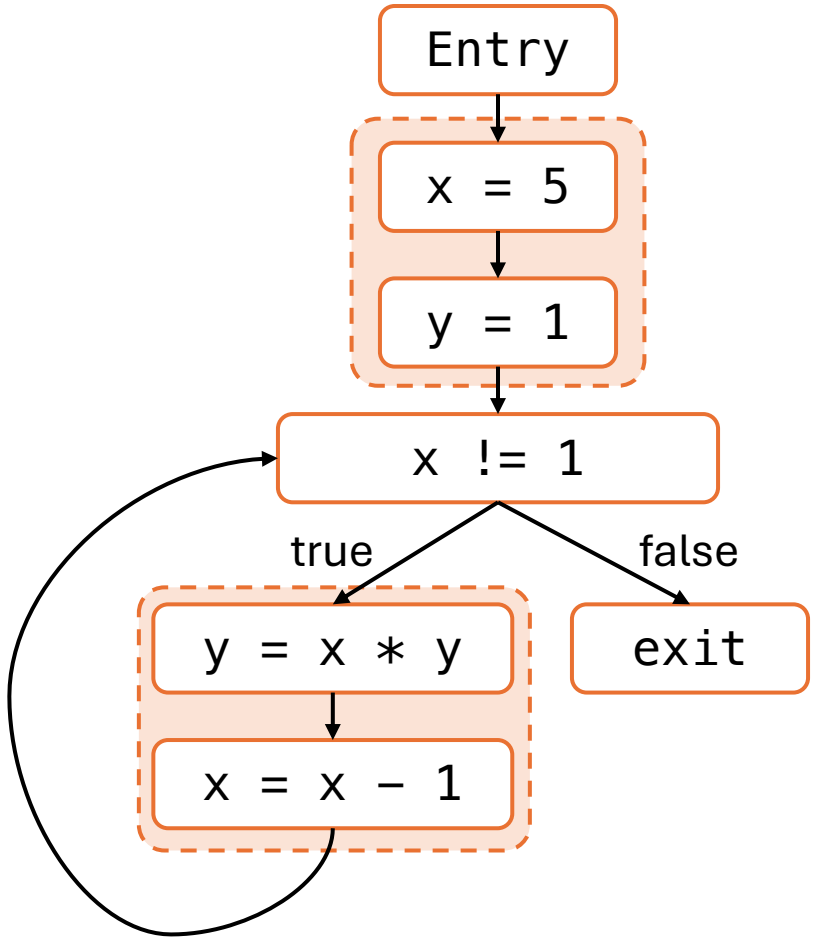
- for (each node n in control-flow graph):
  - $\mathrm{IN}(n) = \{\}$
  - $\mathrm{OUT}(n) = \{\}$
  - $\mathrm{GEN}(n) = \{(v, n)\}$ if n is an assignment else $\{\}$
- repeat:
  - for (each node n):
    - $\mathrm{IN}(n) =$ flow-in(n)
    - $\mathrm{OUT}(n) =$ flow-out(n)
- until all the $\mathrm{IN}(n)$ and $\mathrm{OUT}(n)$ stops changing

Question: Does it always terminate?



Chaotic Iteration for Dataflow Analysis

- for (each node n in control-flow graph):
  - $\mathrm{IN}(n) = \{\}$
  - $\mathrm{OUT}(n) = \{\}$
  - $\mathrm{GEN}(n) = \{(v, n)\}$ if n is an assignment else $\{\}$
- repeat:
  - for (each node n):
    - $\mathrm{IN}(n) =$ flow-in(n)
    - $\mathrm{OUT}(n) =$ flow-out(n)
- until all the $\mathrm{IN}(n)$ and $\mathrm{OUT}(n)$ stops changing

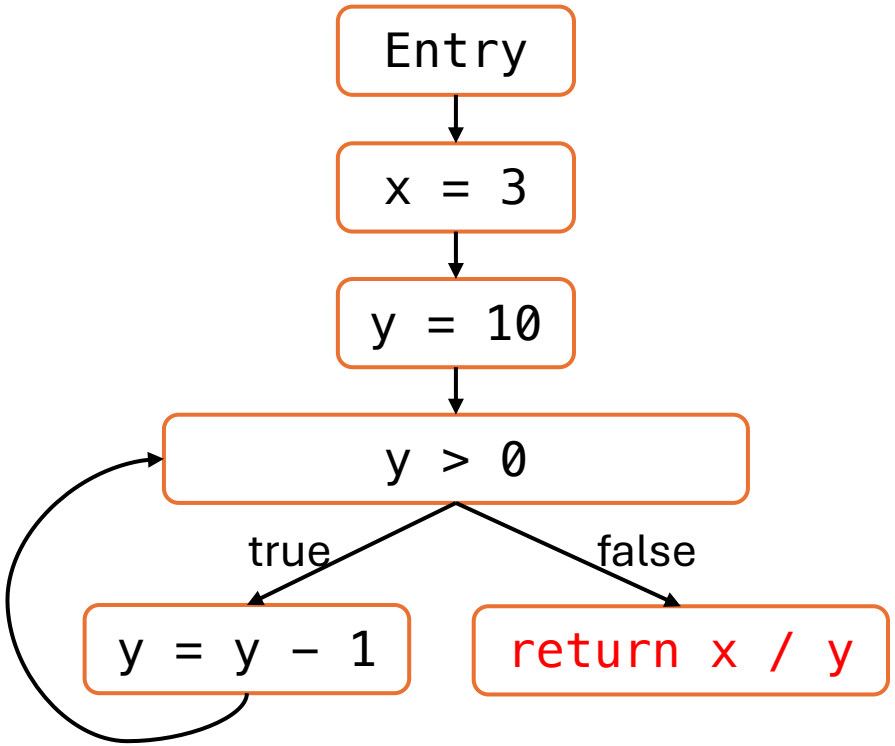


Question: Does it always terminate?

Answer: Yes!

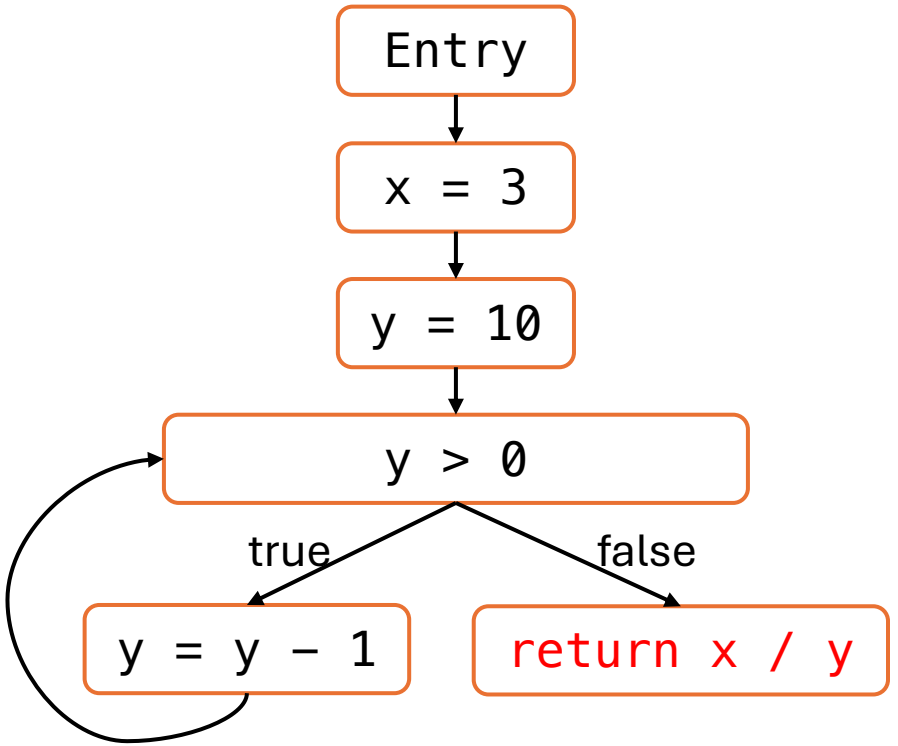
More Complicated Dataflow Analysis

- Question:
  - With Dataflow analysis, can we tell whether there could be a divide-by-zero vulnerability?



More Complicated Dataflow Analysis

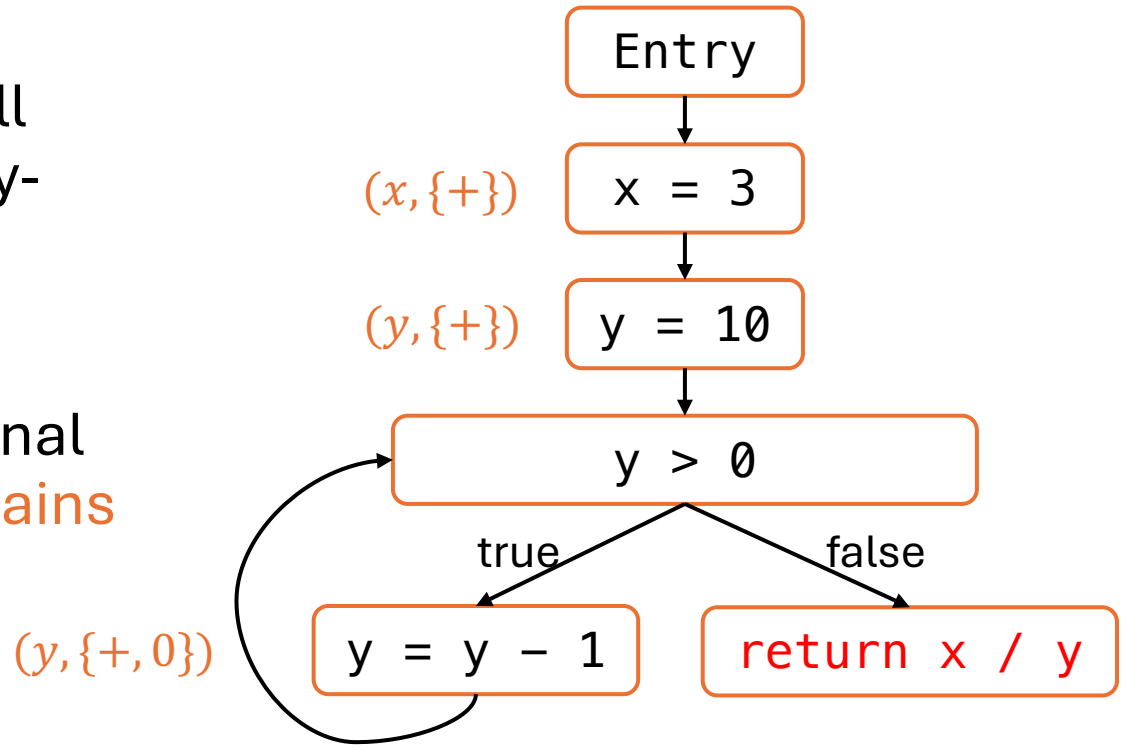
- Question:
  - With Dataflow analysis, can we tell whether there could be a divide-by-zero vulnerability?
- Answer:
  - We can tag the nodes with additional information such as abstract domains



More Complicated Dataflow Analysis

- Question:
  - With Dataflow analysis, can we tell whether there could be a divide-by-zero vulnerability?
- Answer:
  - We can tag the nodes with additional information such as abstract domains

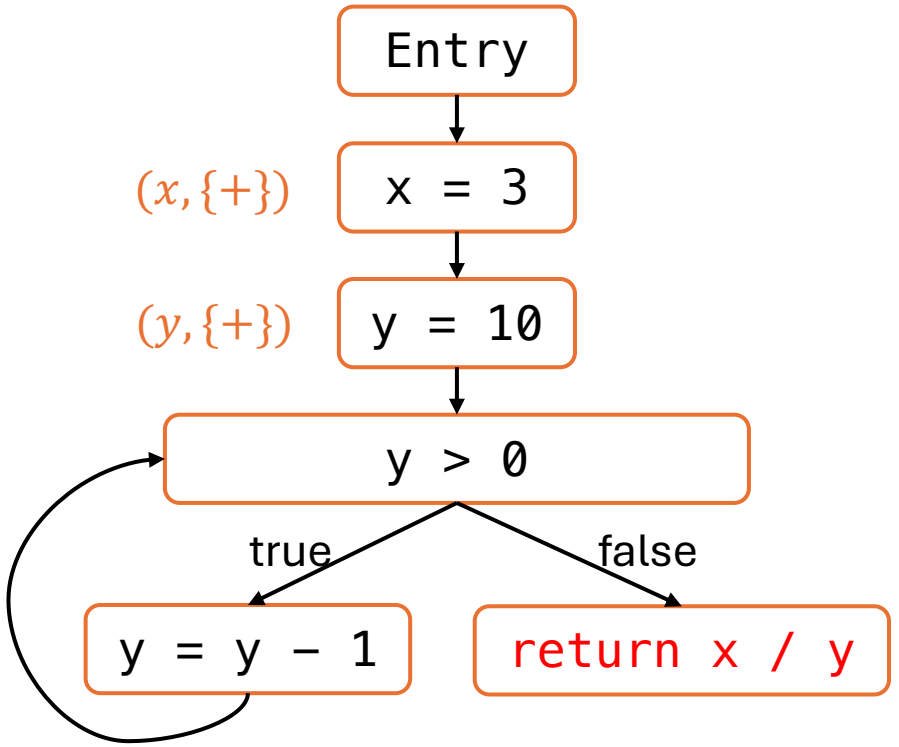
Abstract Domain: $\{+, 0, -\}$



More Complicated Dataflow Analysis

- Question:
  - With Dataflow analysis, can we tell whether there could be a divide-by-zero vulnerability?
- Answer:
  - We can tag the nodes with additional information such as abstract domains

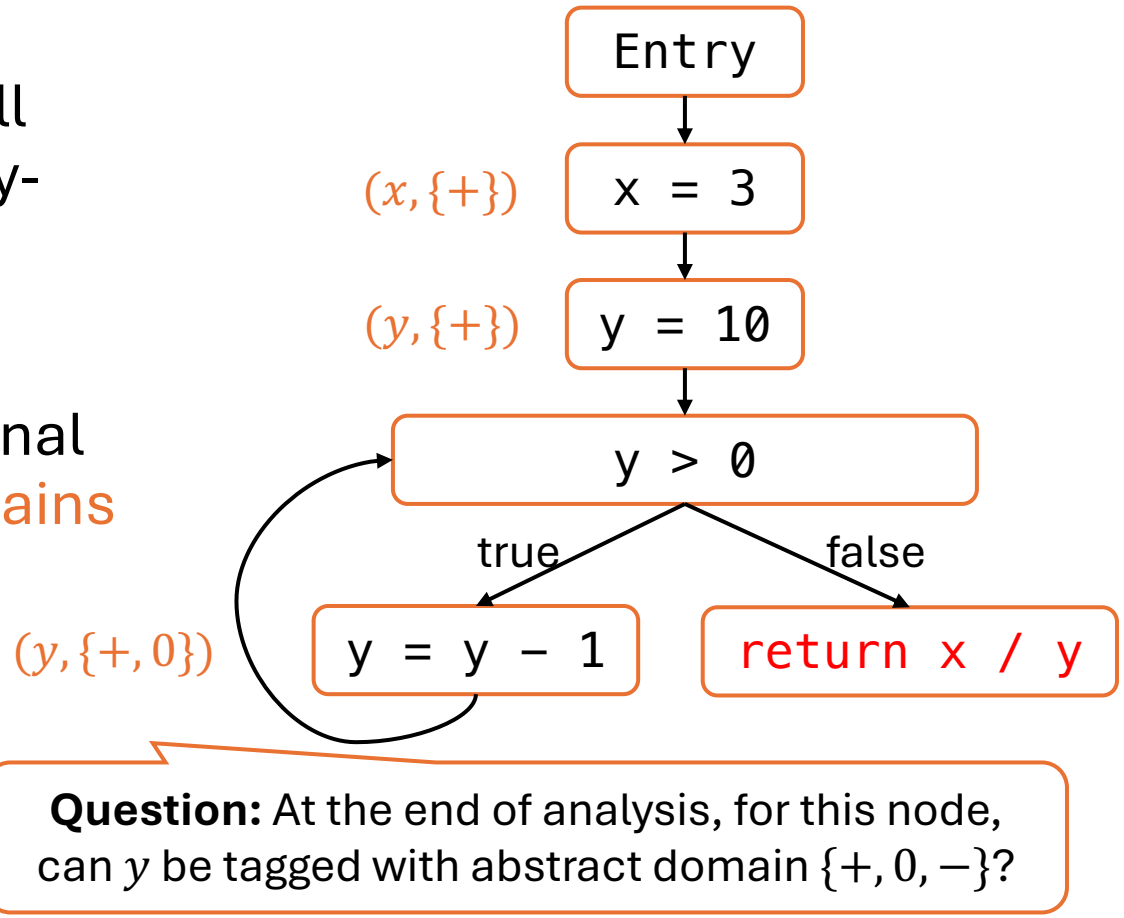
Abstract Domain: $\{+, 0, -\}$



More Complicated Dataflow Analysis

- Question:
  - With Dataflow analysis, can we tell whether there could be a divide-by-zero vulnerability?
- Answer:
  - We can tag the nodes with additional information such as abstract domains

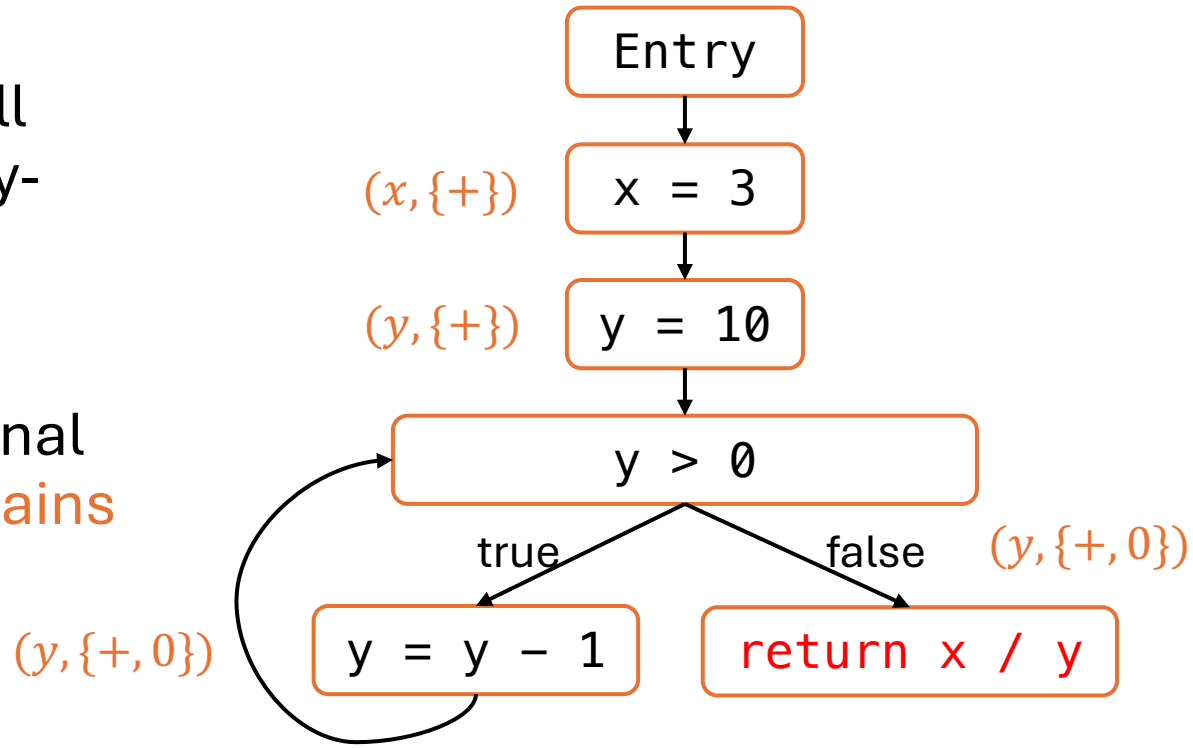
Abstract Domain: $\{+, 0, -\}$



More Complicated Dataflow Analysis

- Question:
  - With Dataflow analysis, can we tell whether there could be a divide-by-zero vulnerability?
- Answer:
  - We can tag the nodes with additional information such as abstract domains

Abstract Domain: $\{+, 0, -\}$



Arithmetic of Abstract Domains

$\{+\} + \{+\} \to \{+\}$

$\{+\} - \{+\} \to \{+, 0, -\}$ Specialization: $\{+\} - 1 \to \{+, 0\}$

$\{+\} * \{+\} \to \{+\}$

$\{+\} + \{0\} \to \{+\}$

$\{+\} - \{0\} \to \{+\}$

$\{+\} * \{0\} \to \{0\}$

$\{+\} + \{-\} \to \{+, 0, -\}$ Specialization: $\{+\} + (-1) \to \{+, 0\}$

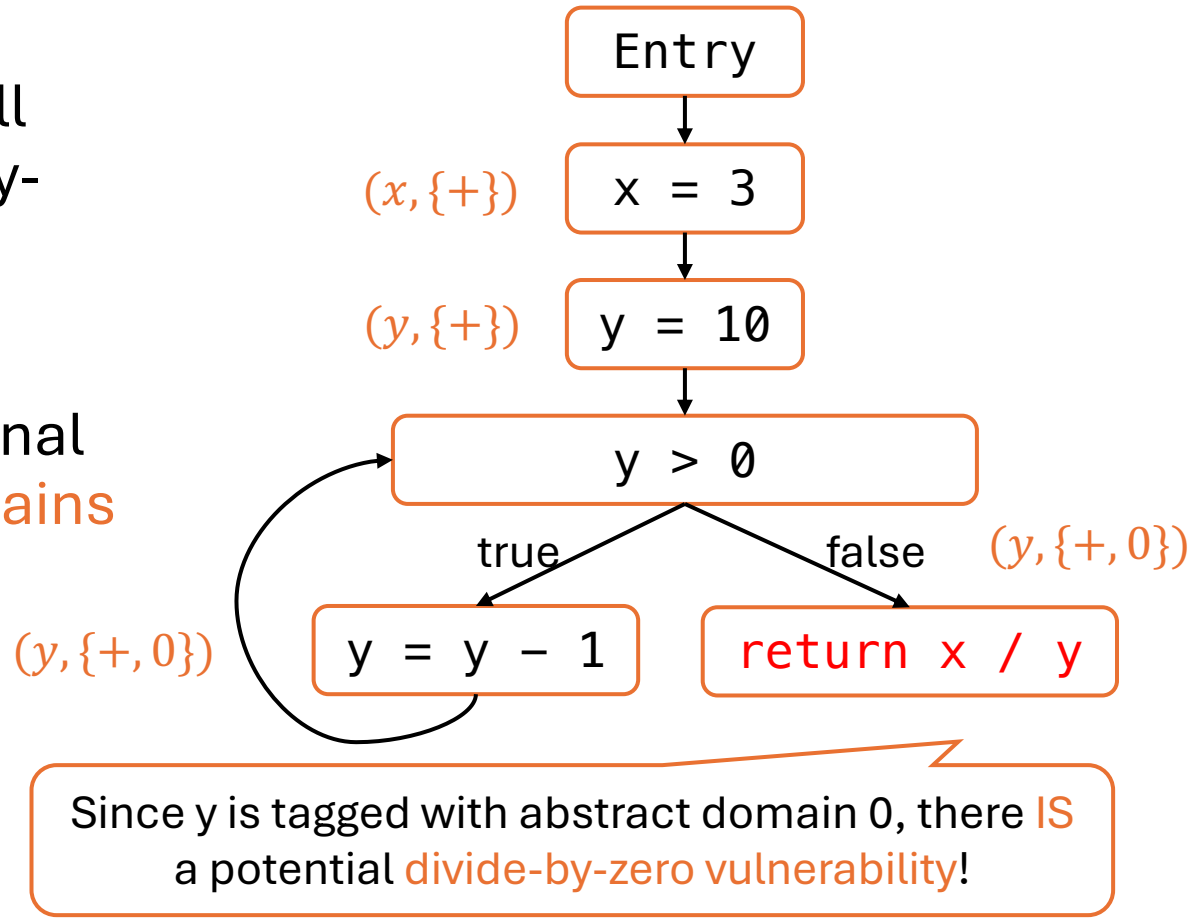
$\{+\} - \{-\} \to \{+\}$

$\{+\} * \{-\} \to \{-\}$

More Complicated Dataflow Analysis

- Question:
  - With Dataflow analysis, can we tell whether there could be a divide-by-zero vulnerability?
- Answer:
  - We can tag the nodes with additional information such as abstract domains

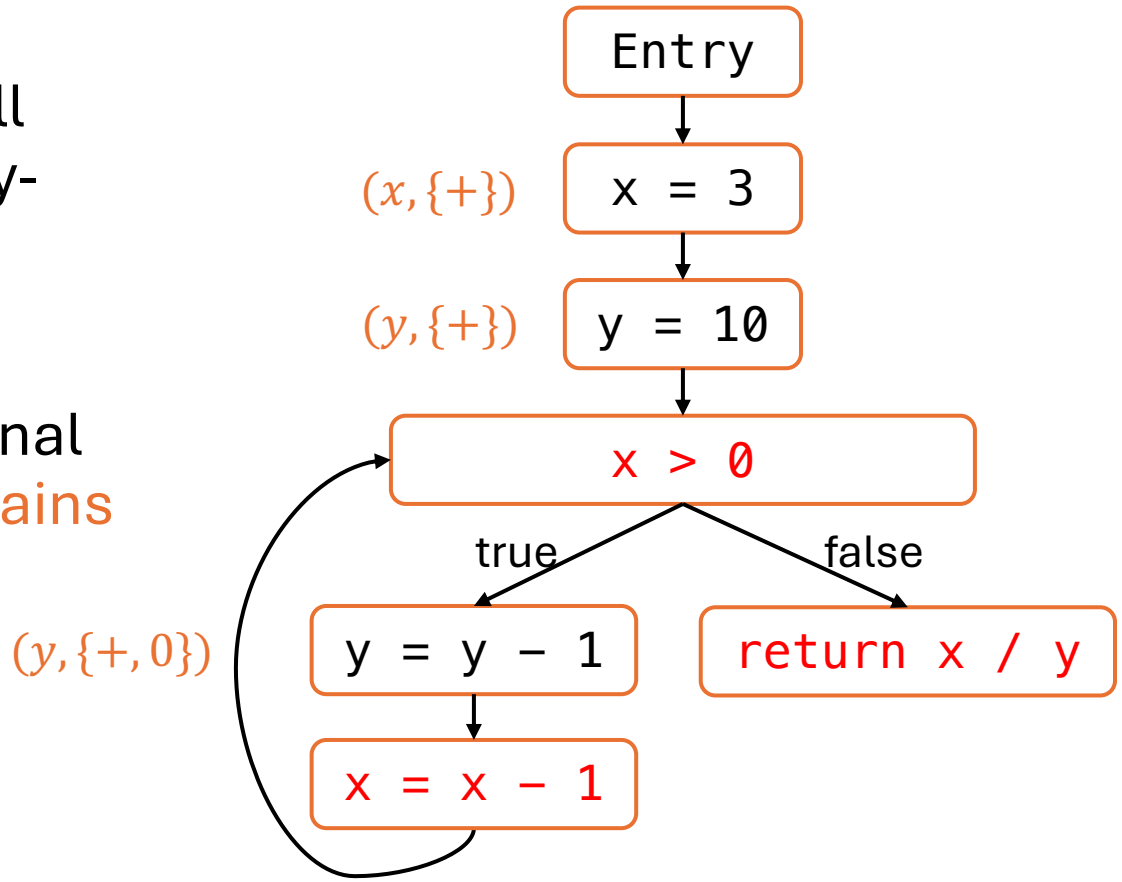
Abstract Domain: $\{+, 0, -\}$



More Complicated Dataflow Analysis: Sensitivity

- Question:
  - With Dataflow analysis, can we tell whether there could be a divide-by-zero vulnerability?
- Answer:
  - We can tag the nodes with additional information such as abstract domains

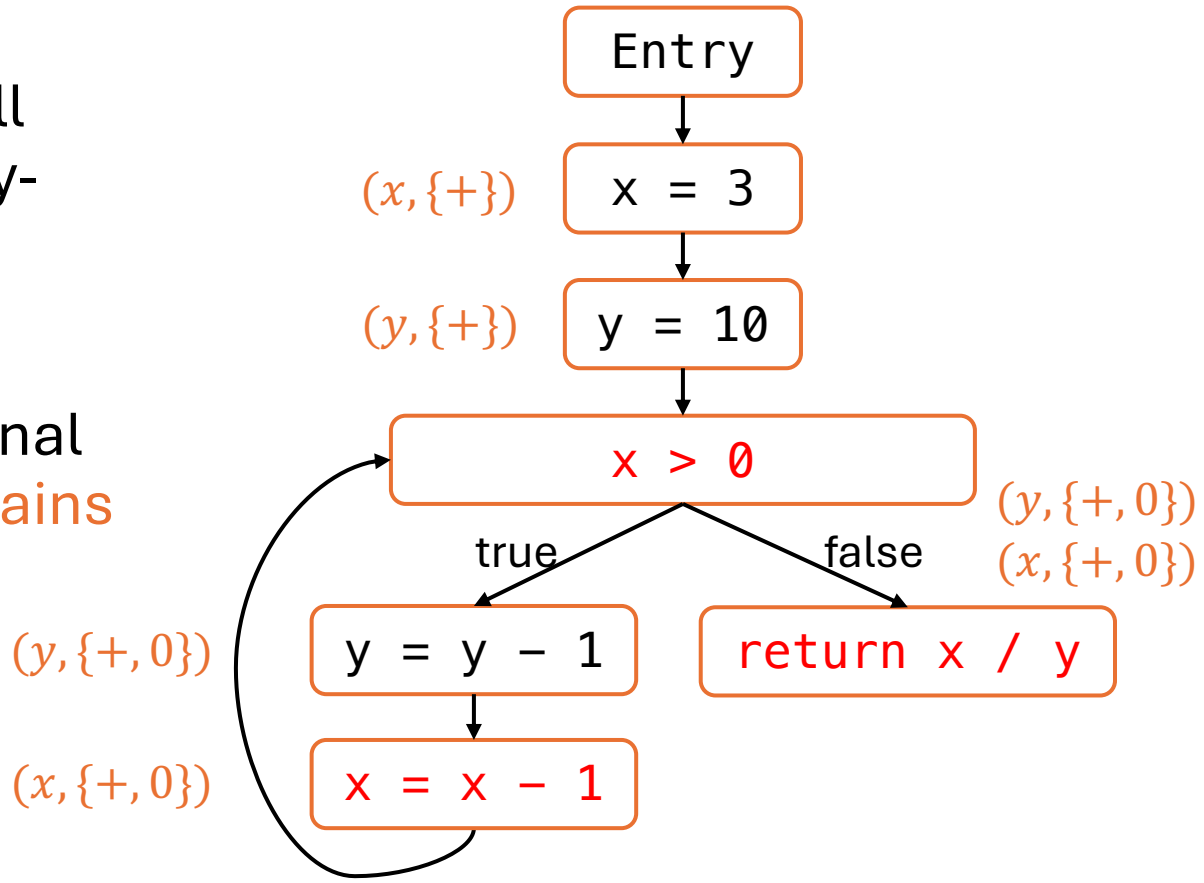
Abstract Domain: $\{+, 0, -\}$



More Complicated Dataflow Analysis: Sensitivity

- Question:
  - With Dataflow analysis, can we tell whether there could be a divide-by-zero vulnerability?
- Answer:
  - We can tag the nodes with additional information such as abstract domains

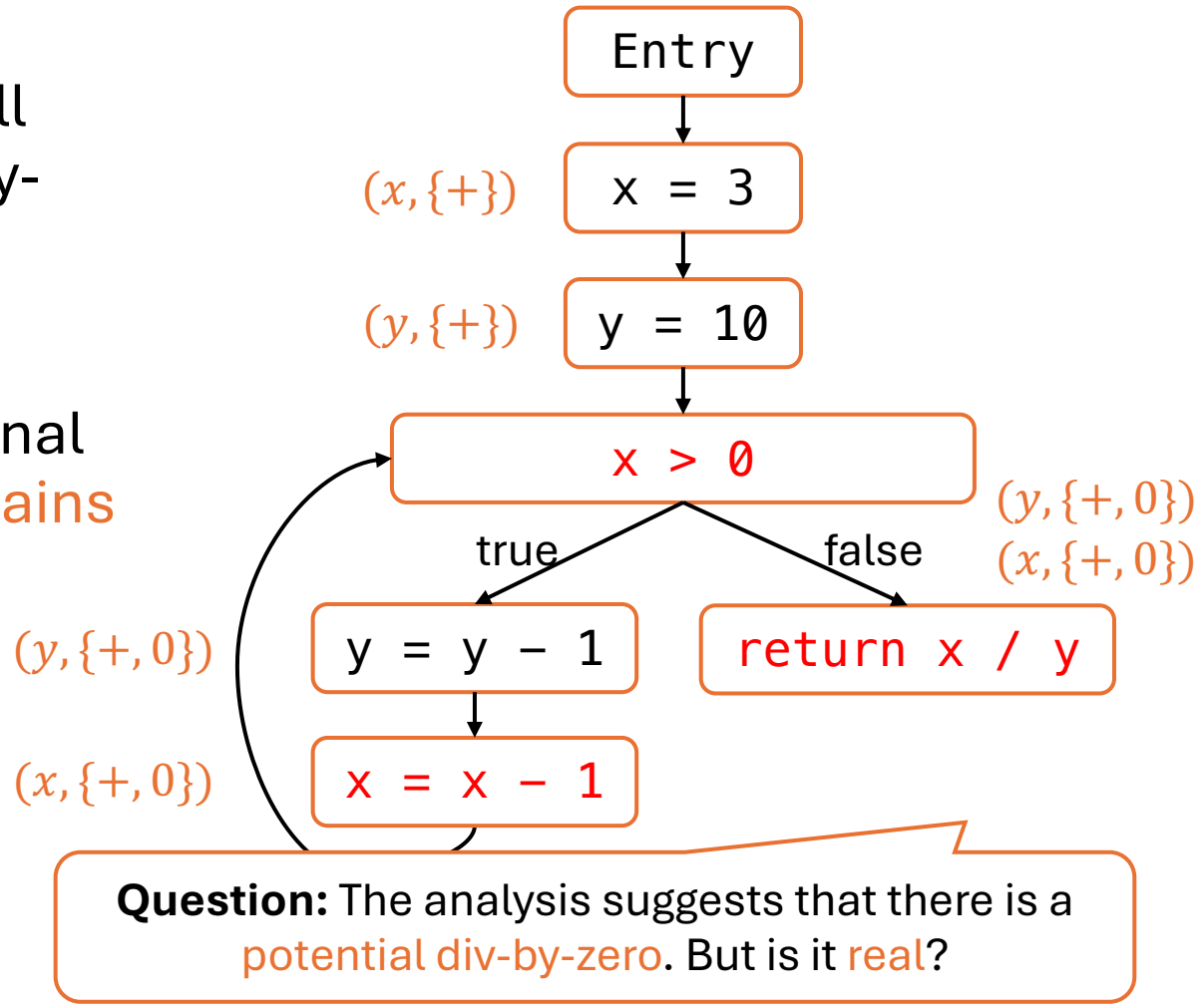
Abstract Domain: $\{+, 0, -\}$



More Complicated Dataflow Analysis: Sensitivity

- Question:
  - With Dataflow analysis, can we tell whether there could be a divide-by-zero vulnerability?
- Answer:
  - We can tag the nodes with additional information such as abstract domains

Abstract Domain: $\{+, 0, -\}$



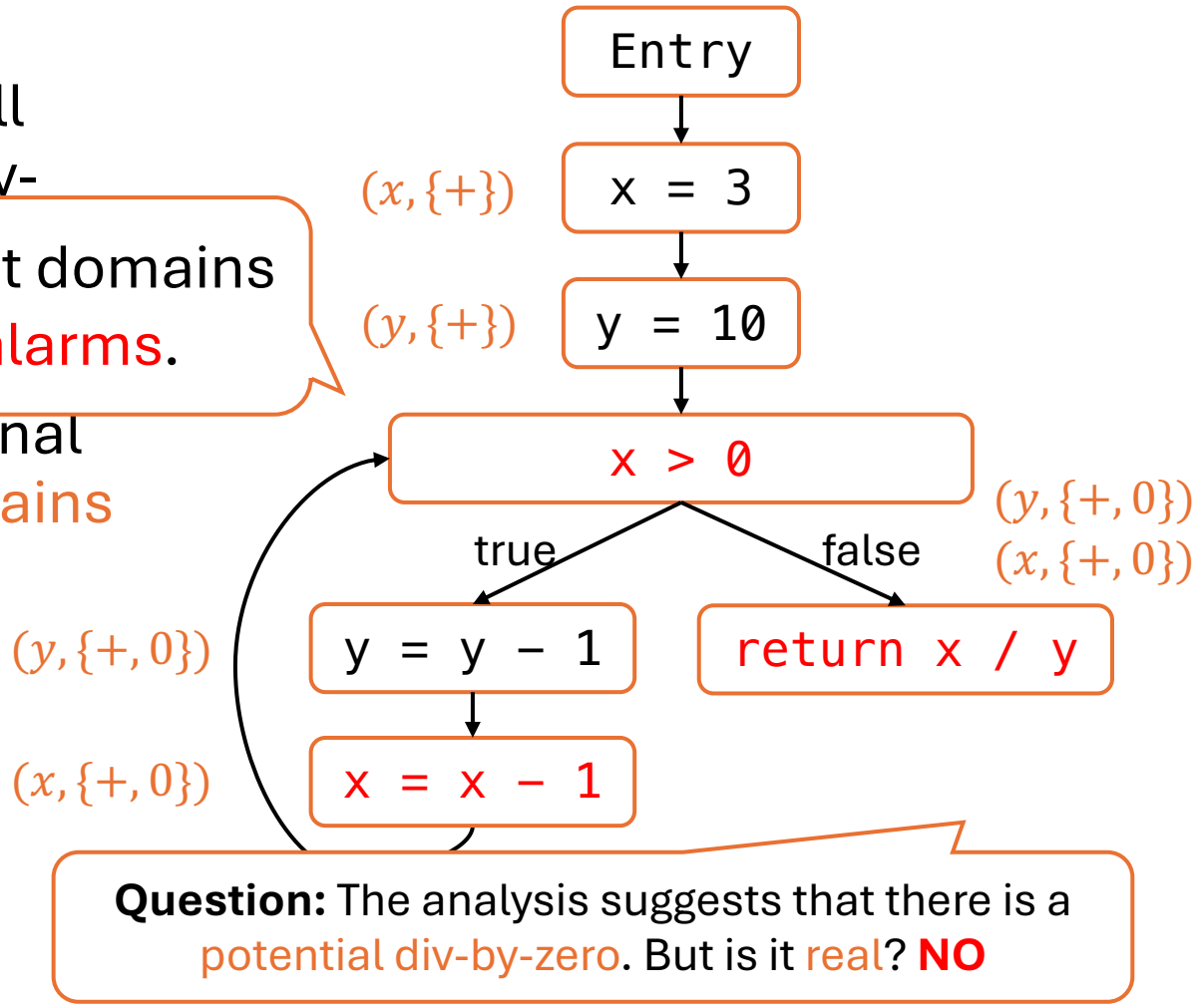
More Complicated Dataflow Analysis: Sensitivity

- Question:
  - With Dataflow analysis, can we tell whether there could be a divide-by-

Dataflow Analysis with $\{+, 0, -\}$ abstract domains reported a false positive div-by-zero alarms.

  - We can tag the nodes with additional information such as abstract domains

Abstract Domain: $\{+, 0, -\}$



Software Testing: Impossible Triangle

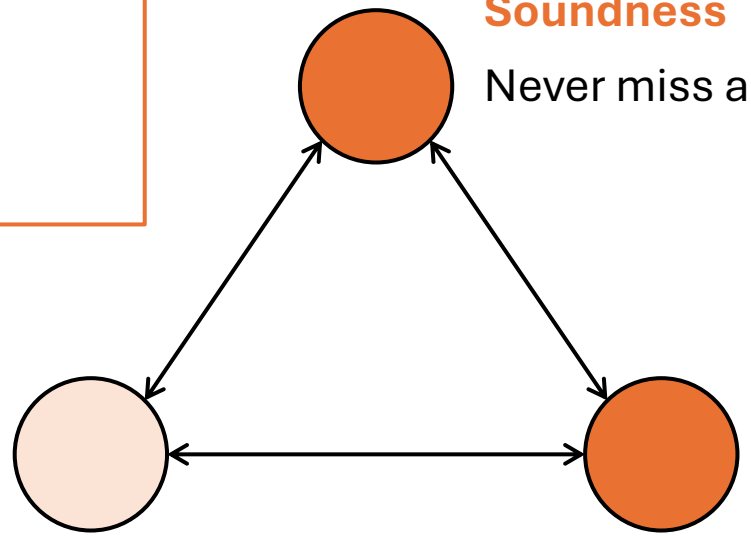
Static Analysis:

Taint Analysis

- (assuming correct specs)

Abstract Interpretation

- (assuming correct domains)



Soundness

Never miss a real bug

~~**Completeness**~~

~~Never raise a false alarm~~

Termination

Always finish the analysis within finite time

May report an alarm for developer confirmation, and that is rejected

Software Testing: Impossible Triangle

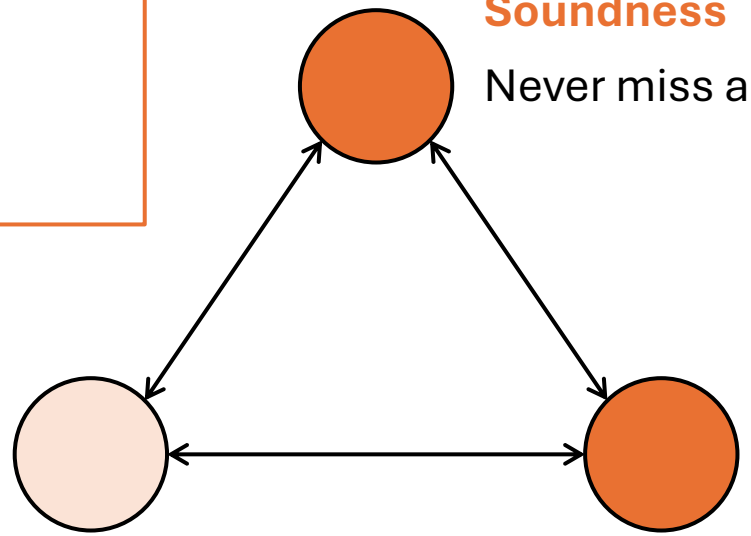
Static Analysis:

Taint Analysis

- (assuming correct specs)

Abstract Interpretation

- (assuming correct domains)



Soundness

Never miss a real bug

~~**Completeness**~~

~~Never raise a false alarm~~

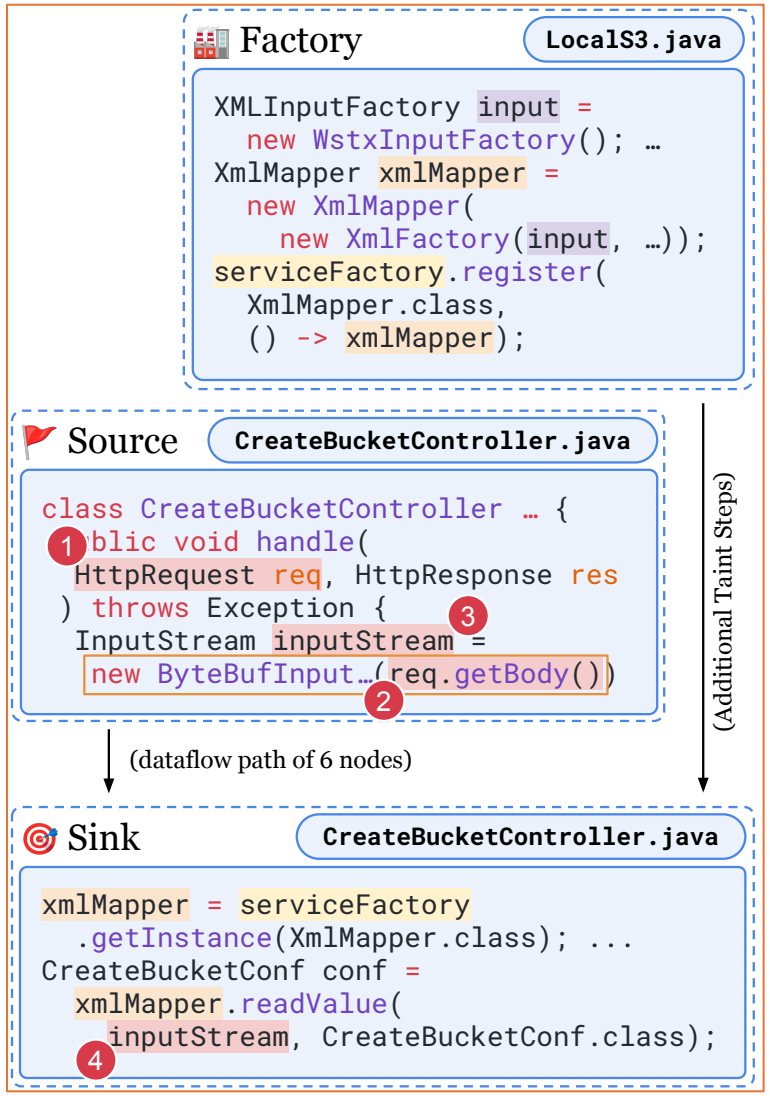
Termination

Always finish the analysis within finite time

Dataflow Analysis & Abstract Domain generalizes to entire space of input, but loses precision

Dataflow Analysis in Real Life: Taint Analysis

Dataflow Analysis in Real Life: Taint Analysis



GitHub Advisory Database / GitHub Reviewed / CVE-2025-27136

LocalS3 CreateBucketConfiguration Endpoint XML External Entity (XXE) Injection

Moderate severity | GitHub Reviewed | Published on Mar 8 in **Robothy/local-s3** • Updated on Mar 14

Steps to Reproduce

1. Create an XML document that includes an external entity declaration pointing to the internal target:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE test [ <!ENTITY xxe SYSTEM "http://internal-web/flag.txt"> ]>
<CreateBucketConfiguration>
    <LocationConstraint>&xxe;</LocationConstraint>
</CreateBucketConfiguration>
```

2. Send a PUT request to create a new bucket with this configuration:

```
curl -X PUT http://app/test-bucket-2 -H 'Content-Type: application/xml' -d @payload.xml
```

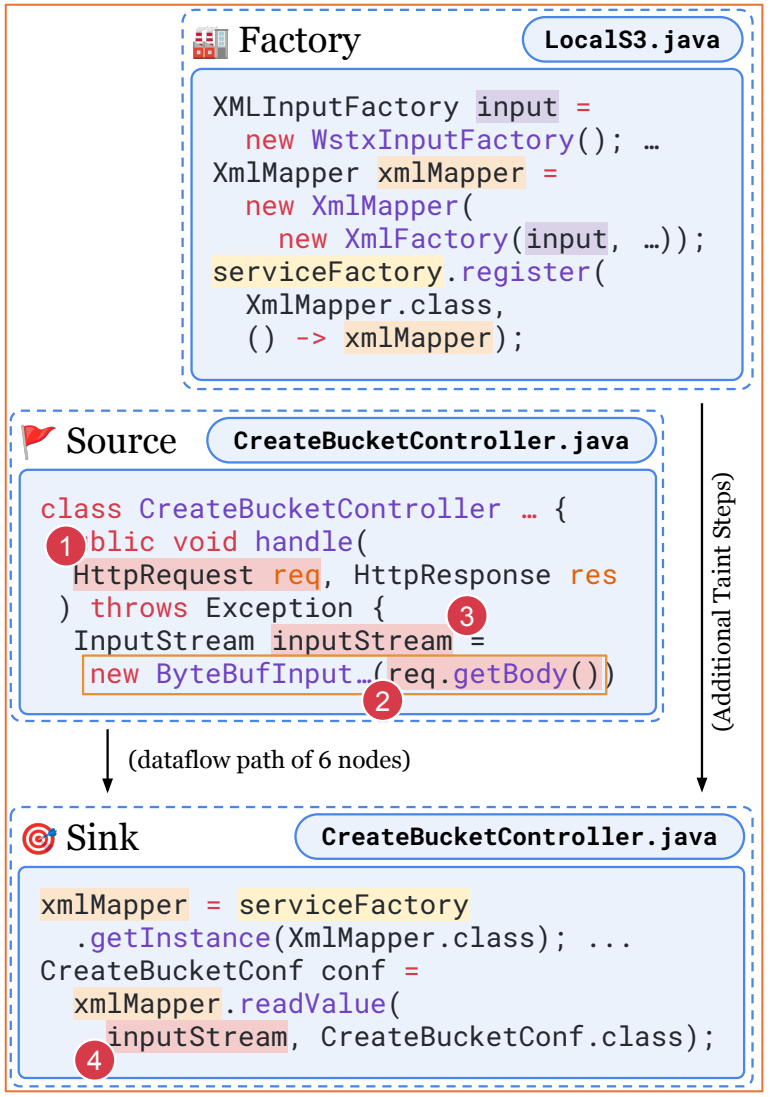
3. Retrieve the bucket location to see the resolved entity content:

```
curl http://app/test-bucket-2/?location
```

When these steps are executed, the server processes the XML, resolves the external entity by making a request to the internal URL, and includes the response in the bucket's location constraint. The attacker can then retrieve this information through the bucket location endpoint.

Dataflow Analysis in Real Life: Taint Analysis

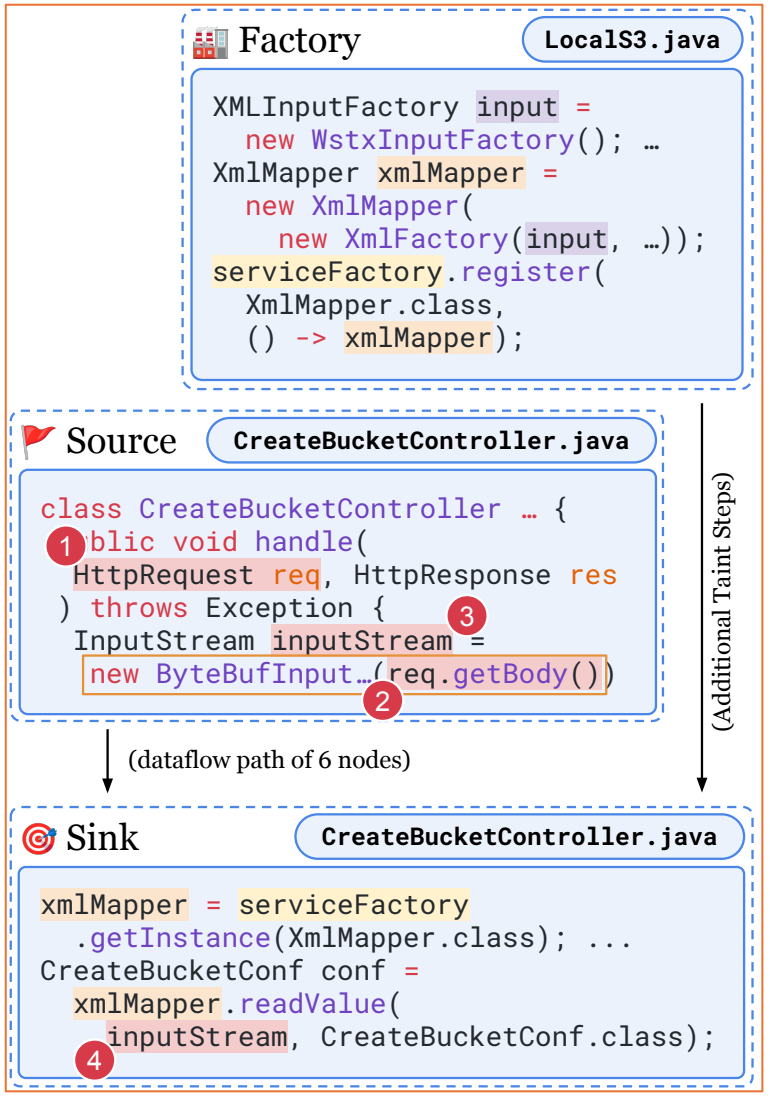
- Goal:
  - See how a program point with “tainted information” may flow through the code
  - How is it “Manifested” into a real vulnerability



CVE-2025-27136

Dataflow Analysis in Real Life: Taint Analysis

- Goal:
  - See how a program point with "tainted information" may flow through the code
  - How is it "Manifested" into a real vulnerability
- Taint Specification:
  - **Source**: the HttpRequest parameter to a handle function of a Java Servlet
  - **Sink**: the readValue function of XmlMapper



CVE-2025-27136

Dataflow Analysis in Real Life: Taint Analysis

IRIS: LLM-Assisted Static Analysis for Detecting Security Vulnerabilities

Ziyang Li
University of Pennsylvania
liby99@cis.upenn.edu

Saikat Dutta
Cornell University
saikatd@cornell.edu

Mayur Naik
University of Pennsylvania
mhnaik@cis.upenn.edu

Dataflow Analysis in Real Life: Taint Analysis

IRIS: LLM-ASSISTED STATIC ANALYSIS FOR

cronutils/validation/CronValidator.java

```
@override
public boolean isValid(
  String value, ConstraintValidatorContext context) {
  try {
    cronParser.parse(value).validate(); // ...
  } catch (IllegalArgumentException e) {
    context
      .buildConstraintViolationWithTemplate(e.getMessage())
      .addConstraintViolation(); // ...
```

cronutils/parser/CronParser.java

```
/** Parse string with cron expression. ... */
public Cron parse(final String expression) {
  try { /* ... */ } catch {
  throw new IllegalArgumentException(
    String.format("Failed to parse '%s'. %s",
      expression, e.getMessage()), e);
  }
}
```

Figure 2: An example of Code Injection (CWE-94) vulnerability found in cron-utils (CVE-2021-41269) that CodeQL fails to detect. We number the program points of the vulnerable path.

Dataflow Analysis in Real Life: Taint Analysis

IRIS: LLM-ASSISTED STATIC ANALYSIS FOR DETECT

Ziyang Li
University of
liby99@ci

cronutils/validation/CronValidator.java

```
@override
public boolean isValid(
  String value, ConstraintValidatorContext context) {
  try {
    cronParser.parse(value).validate(); // ...
  } catch (IllegalArgumentException e) {
    context
      .buildConstraintViolationWithTemplate(e.getMessage())
      .addConstraintViolation(); // ...
```

cronutils/parser/CronParser.java

```
/** Parse string with cron expression. ... */
public Cron parse(final String expression) {
  try { /* ... */ } catch {
  throw new IllegalArgumentException(
    String.format("Failed to parse '%s'. %s",
      expression, e.getMessage()), e);
  }
}
```

Figure 2: An example of Code Injection (CWE-94) vulnerability found in cron-utils (CVE-2021-41269) that CodeQL fails to detect. We number the program points of the vulnerable path.

Taint Specification:

Source: the value parameter to the isValue function of a CronValidator

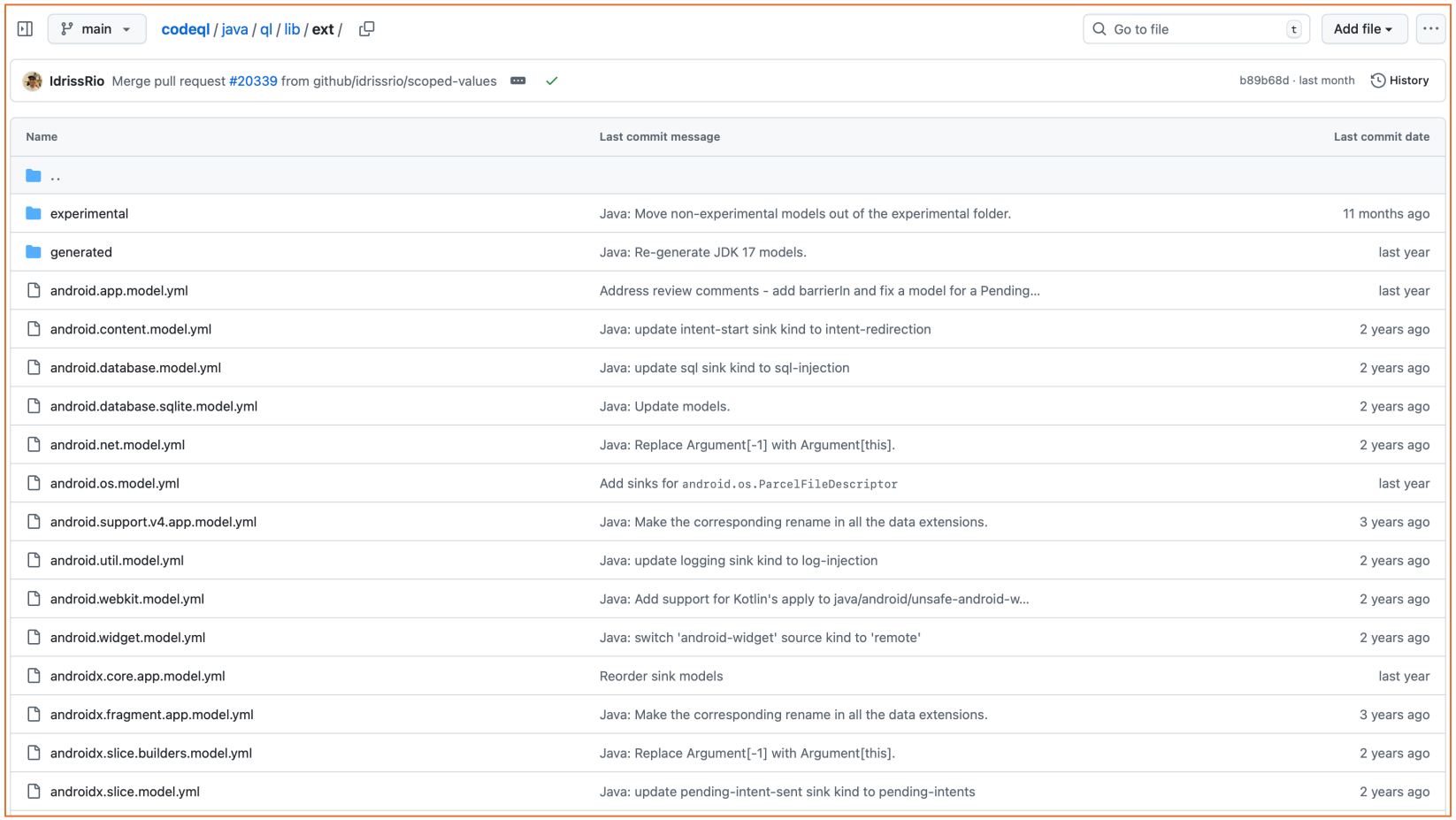
Sink: the buildConstraintViolationWithTemplate function of ConstraintValidatorContext

Dataflow Analysis in Real Life: Taint Analysis

- Where are those taint specifications coming from?

Dataflow Analysis in Real Life: Taint Analysis

- Where are those taint specifications coming from?



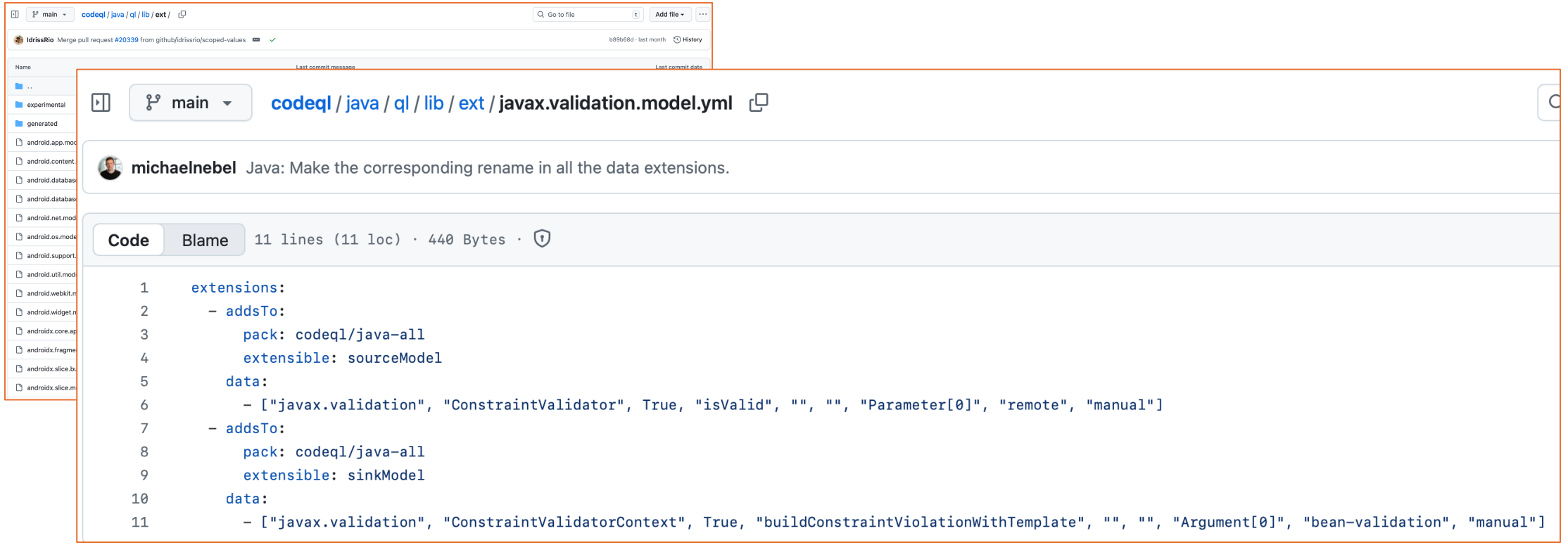
main codeql / java / ql / lib / ext / Go to file t Add file

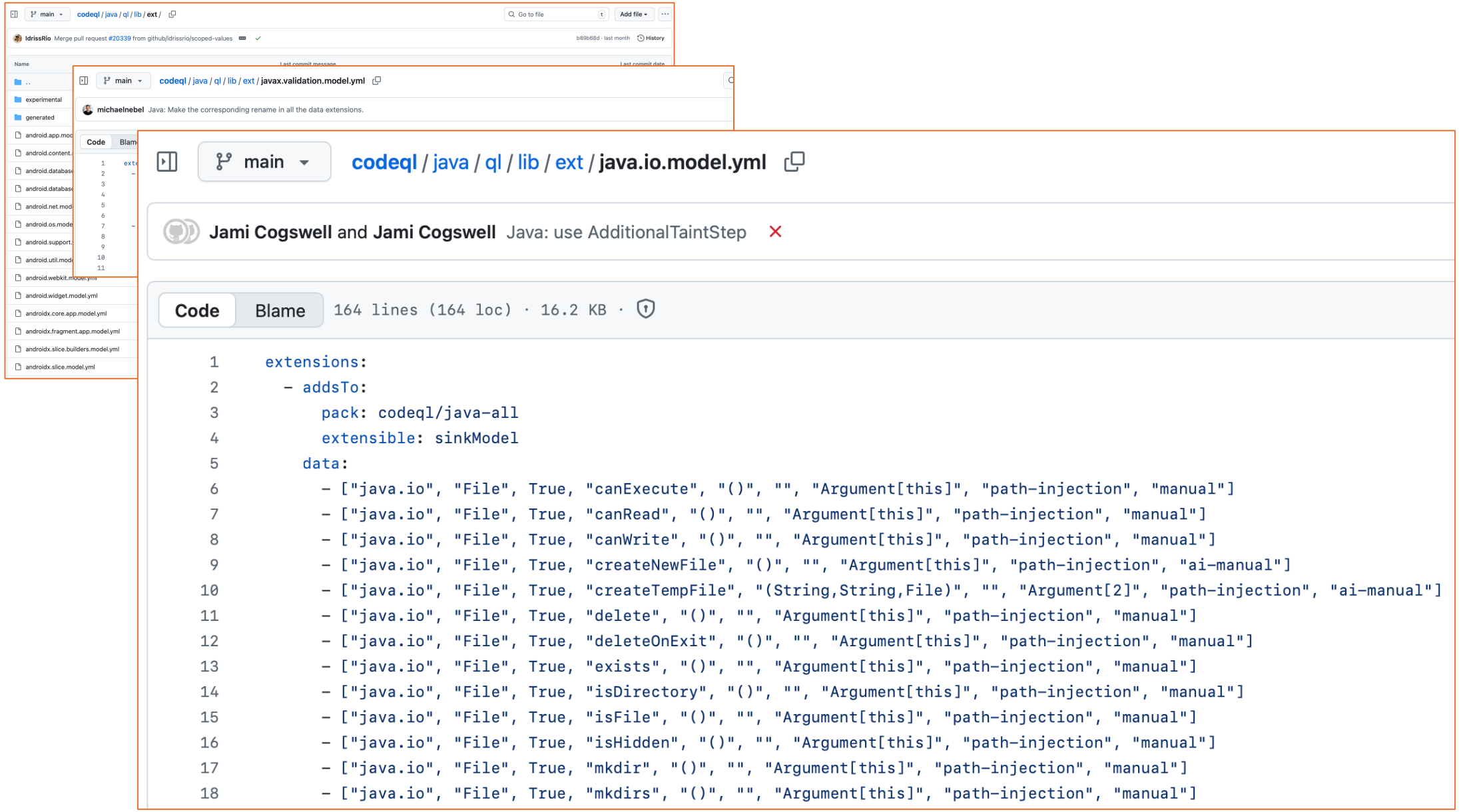
IdrissRio Merge pull request #20339 from github/idrissrio/scoped-values b89b68d · last month History

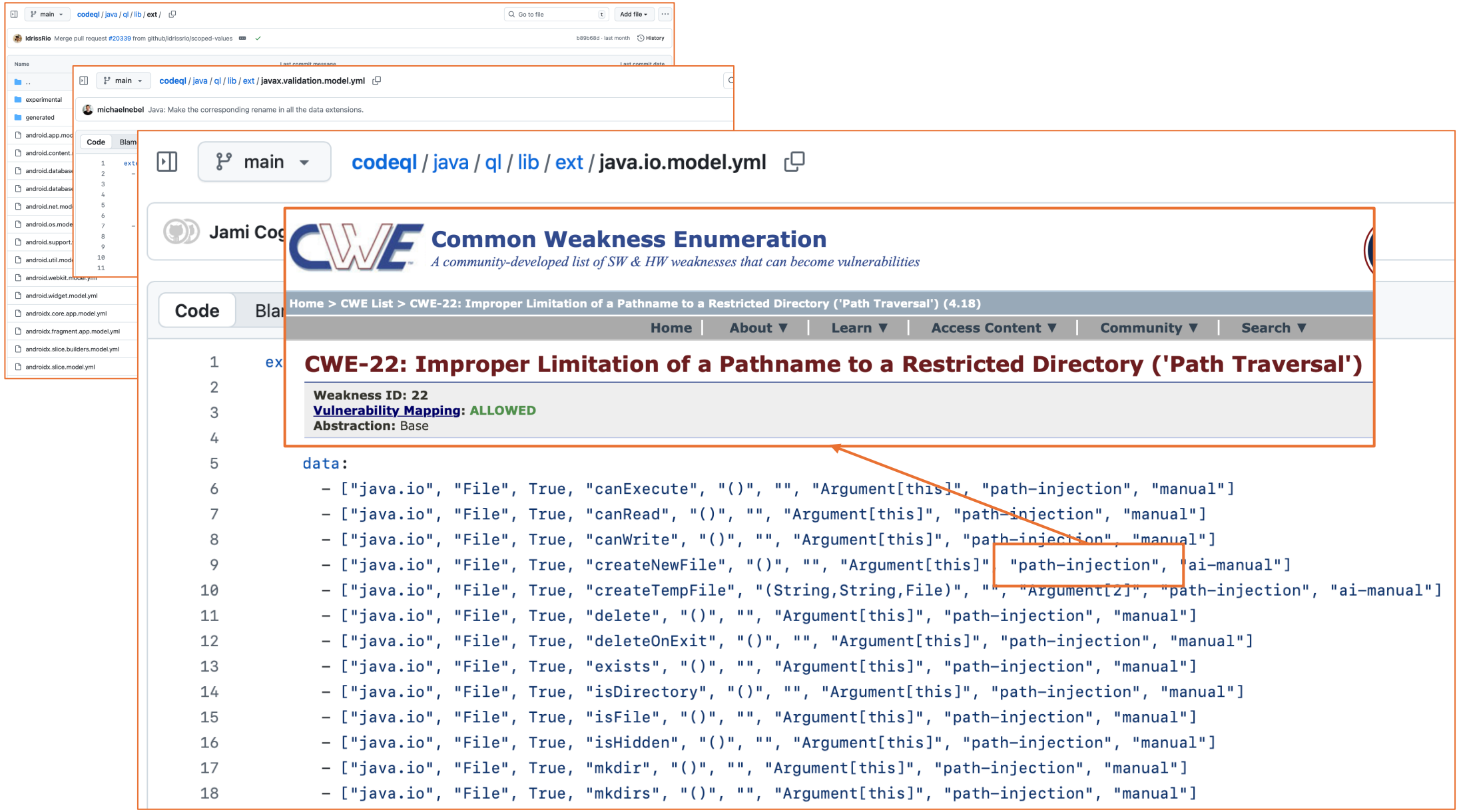
Name	Last commit message	Last commit date
..		
experimental	Java: Move non-experimental models out of the experimental folder.	11 months ago
generated	Java: Re-generate JDK 17 models.	last year
android.app.model.yml	Address review comments - add barrierIn and fix a model for a Pending...	last year
android.content.model.yml	Java: update intent-start sink kind to intent-redirection	2 years ago
android.database.model.yml	Java: update sql sink kind to sql-injection	2 years ago
android.database.sqlite.model.yml	Java: Update models.	2 years ago
android.net.model.yml	Java: Replace Argument[-1] with Argument[this].	2 years ago
android.os.model.yml	Add sinks for `android.os.ParcelFileDescriptor`	last year
android.support.v4.app.model.yml	Java: Make the corresponding rename in all the data extensions.	3 years ago
android.util.model.yml	Java: update logging sink kind to log-injection	2 years ago
android.webkit.model.yml	Java: Add support for Kotlin's apply to java/android/unsafe-android-w...	2 years ago
android.widget.model.yml	Java: switch 'android-widget' source kind to 'remote'	2 years ago
androidx.core.app.model.yml	Reorder sink models	last year
androidx.fragment.app.model.yml	Java: Make the corresponding rename in all the data extensions.	3 years ago
androidx.slice.builders.model.yml	Java: Replace Argument[-1] with Argument[this].	2 years ago
androidx.slice.model.yml	Java: update pending-intent-sent sink kind to pending-intents	2 years ago

Dataflow Analysis in Real Life: Taint Analysis

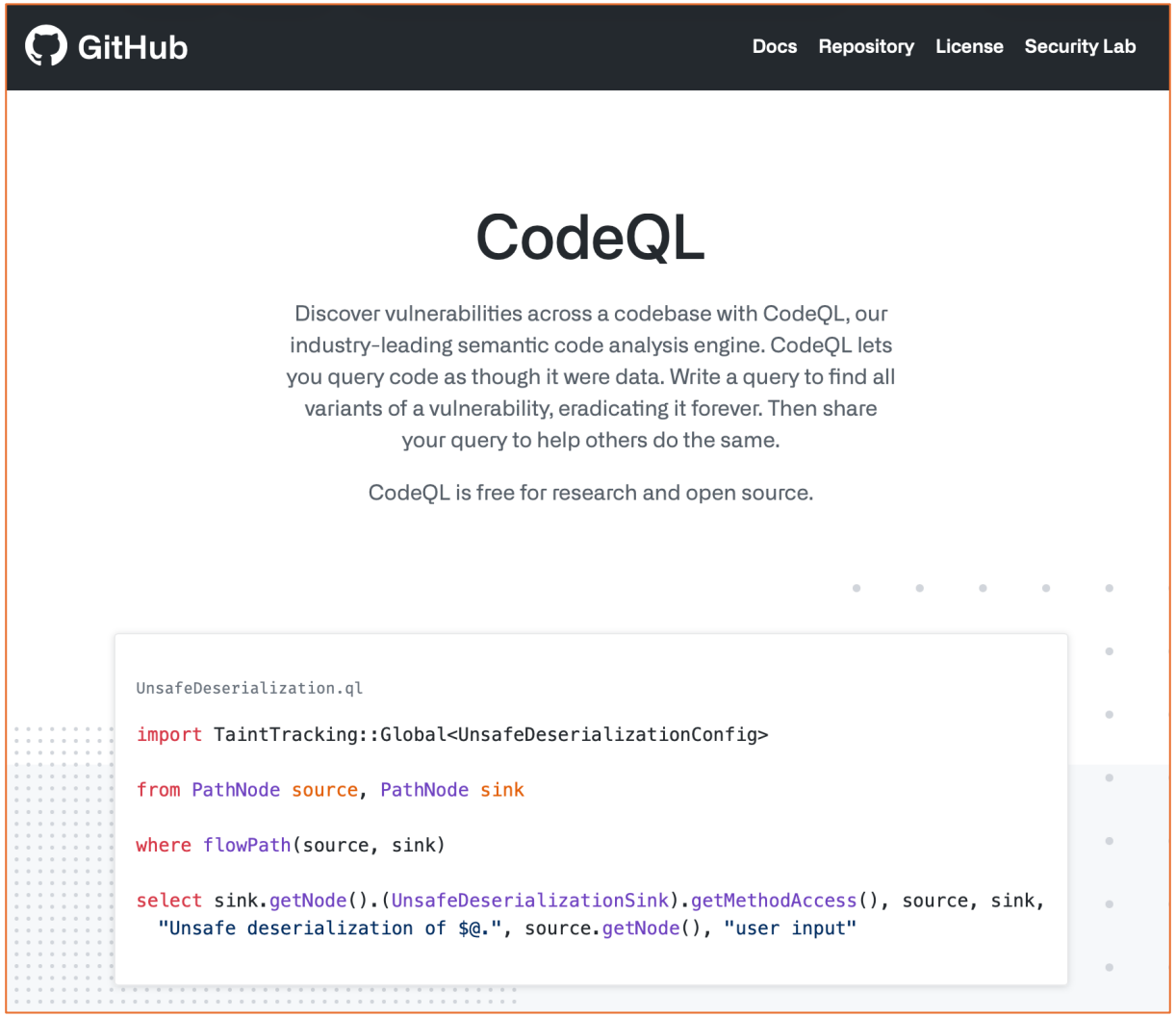
- Where are those taint specifications coming from?




main
codeql / java / ql / lib / ext / java.io.model.yml
Jami Cogswell and Jami Cogswell Java: use AdditionalTaintStep
Code
Blame
164 lines (164 loc) · 16.2 KB
extensions:
- addsTo:
pack: codeql/java-all
extensible: sinkModel
data:
- ["java.io", "File", True, "canExecute", "()", "", "Argument[this]", "path-injection", "manual"]
- ["java.io", "File", True, "canRead", "()", "", "Argument[this]", "path-injection", "manual"]
- ["java.io", "File", True, "canWrite", "()", "", "Argument[this]", "path-injection", "manual"]
- ["java.io", "File", True, "createNewFile", "()", "", "Argument[this]", "path-injection", "ai-manual"]
- ["java.io", "File", True, "createTempFile", "(String,String,File)", "", "Argument[2]", "path-injection", "ai-manual"]
- ["java.io", "File", True, "delete", "()", "", "Argument[this]", "path-injection", "manual"]
- ["java.io", "File", True, "deleteOnExit", "()", "", "Argument[this]", "path-injection", "manual"]
- ["java.io", "File", True, "exists", "()", "", "Argument[this]", "path-injection", "manual"]
- ["java.io", "File", True, "isDirectory", "()", "", "Argument[this]", "path-injection", "manual"]
- ["java.io", "File", True, "isFile", "()", "", "Argument[this]", "path-injection", "manual"]
- ["java.io", "File", True, "isHidden", "()", "", "Argument[this]", "path-injection", "manual"]
- ["java.io", "File", True, "mkdir", "()", "", "Argument[this]", "path-injection", "manual"]
- ["java.io", "File", True, "mkdirs", "()", "", "Argument[this]", "path-injection", "manual"]


main
codeql / java / ql / lib / ext /
Go to file
Add file
IdrissRio Merge pull request #20339 from github/idrissrio/scoped-values
b89b68d · last month
History
Name
Last commit message
Last commit date
..
experimental
generated
android.app.mod
android.content.
android.database
android.database
android.net.mod
android.os.mode
android.support.
android.util.mod
android.webkit.model.yml
android.widget.model.yml
androidx.core.app.model.yml
androidx.fragment.app.model.yml
androidx.slice.builders.model.yml
androidx.slice.model.yml
main
codeql / java / ql / lib / ext / javax.validation.model.yml
michaelnebel Java: Make the corresponding rename in all the data extensions.
Code
Blam
main
codeql / java / ql / lib / ext / java.io.model.yml
Jami Cog
Common Weakness Enumeration
A community-developed list of SW & HW weaknesses that can become vulnerabilities
Home > CWE List > CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') (4.18)
Home | About ▼ | Learn ▼ | Access Content ▼ | Community ▼ | Search ▼
CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
Weakness ID: 22
Vulnerability Mapping: ALLOWED
Abstraction: Base
Code
Bla
ex
data:
- ["java.io", "File", True, "canExecute", "()", "", "Argument[this]", "path-injection", "manual"]
- ["java.io", "File", True, "canRead", "()", "", "Argument[this]", "path-injection", "manual"]
- ["java.io", "File", True, "canWrite", "()", "", "Argument[this]", "path-injection", "manual"]
- ["java.io", "File", True, "createNewFile", "()", "", "Argument[this]", "path-injection", "ai-manual"]
- ["java.io", "File", True, "createTempFile", "(String,String,File)", "", "Argument[2]", "path-injection", "ai-manual"]
- ["java.io", "File", True, "delete", "()", "", "Argument[this]", "path-injection", "manual"]
- ["java.io", "File", True, "deleteOnExit", "()", "", "Argument[this]", "path-injection", "manual"]
- ["java.io", "File", True, "exists", "()", "", "Argument[this]", "path-injection", "manual"]
- ["java.io", "File", True, "isDirectory", "()", "", "Argument[this]", "path-injection", "manual"]
- ["java.io", "File", True, "isFile", "()", "", "Argument[this]", "path-injection", "manual"]
- ["java.io", "File", True, "isHidden", "()", "", "Argument[this]", "path-injection", "manual"]
- ["java.io", "File", True, "mkdir", "()", "", "Argument[this]", "path-injection", "manual"]
- ["java.io", "File", True, "mkdirs", "()", "", "Argument[this]", "path-injection", "manual"]

Dataflow Analysis in Real Life: CodeQL



Dat

GitHub

```
/**
 * @name Uncontrolled data used in path expression
 * @description Accessing paths influenced by users can allow an attacker to access un
 * @kind path-problem
 * @problem.severity error
 * @security-severity 7.5
 * @precision high
 * @id java/path-injection
 * @tags security
 *       external/cwe/cwe-022
 *       external/cwe/cwe-023
 *       external/cwe/cwe-036
 *       external/cwe/cwe-073
 */

import java
import semmle.code.java.security.TaintedPathQuery
import TaintedPathFlow::PathGraph

from TaintedPathFlow::PathNode source, TaintedPathFlow::PathNode sink
where TaintedPathFlow::flowPath(source, sink)
select sink.getNode(), source, sink, "This path depends on a $@.", source.getNode(),
  "user-provided value"
```

https://github.com/github/codeql/blob/main/java/ql/src/Security/CWE/CWE-022/TaintedPath.ql

Dataflow Analysis in Real Life: CodeQL

- Question:
  - When we have a programming language for static analysis…
  - Can we Synthesize Program in this language?

Published as a conference paper at ICLR 2025

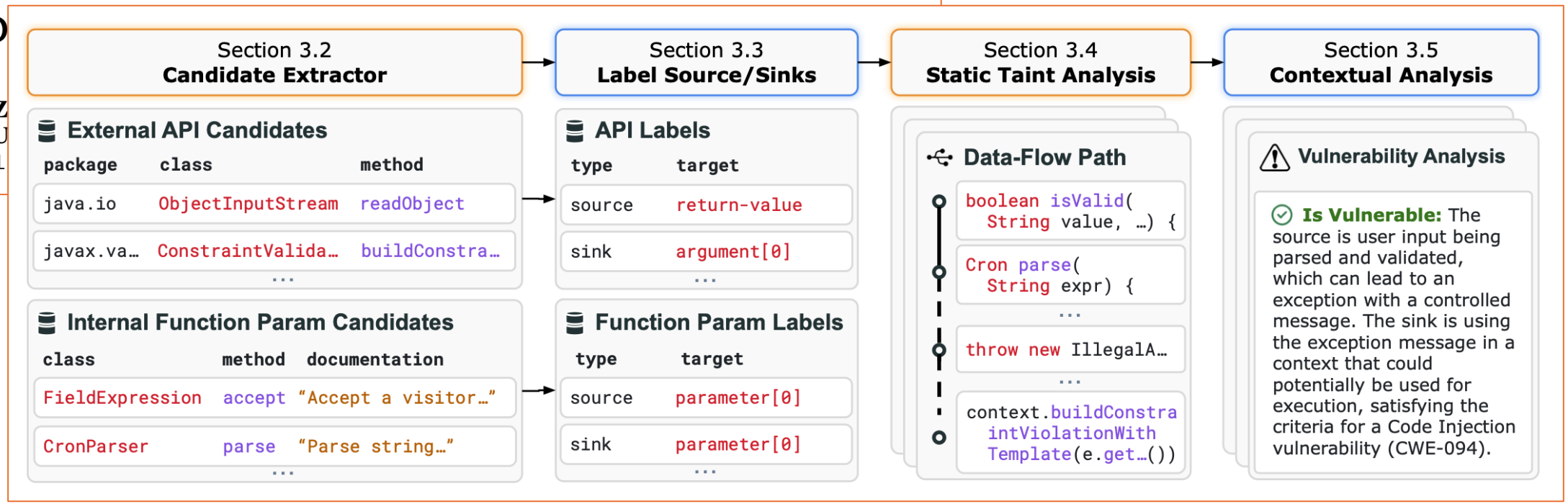
IRIS: LLM-ASSISTED STATIC ANALYSIS FOR DETECTING SECURITY VULNERABILITIES

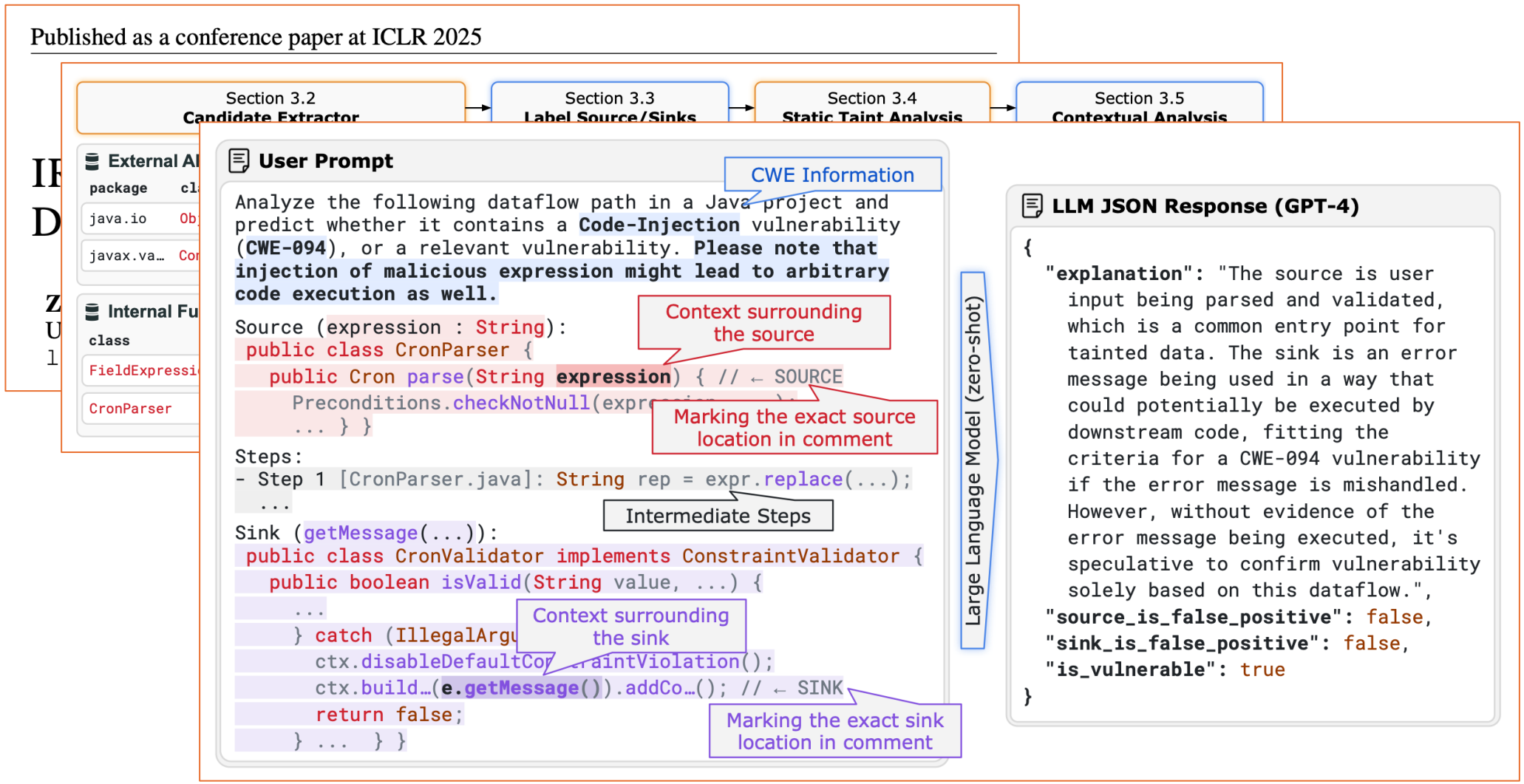
Ziyang Li
University of Pennsylvania
liby99@cis.upenn.edu

Saikat Dutta
Cornell University
saikatd@cornell.edu

Mayur Naik
University of Pennsylvania
mhnaik@cis.upenn.edu

IRIS: LLM-ASSISTED STATIC ANALYSIS FOR D




Section 3.2
Candidate Extractor
Section 3.3
Label Source/Sinks
Section 3.4
Static Taint Analysis
Section 3.5
Contextual Analysis
External Al
package
java.io
javax.va…
Internal Fu
class
FieldExpressi
CronParser
User Prompt
CWE Information
Analyze the following dataflow path in a Java project and predict whether it contains a Code-Injection vulnerability (CWE-094), or a relevant vulnerability. Please note that injection of malicious expression might lead to arbitrary code execution as well.
Source (expression : String):
public class CronParser {
public Cron parse(String expression) { // ← SOURCE
Preconditions.checkNotNull(
... } }
Context surrounding the source
Marking the exact source location in comment
Steps:
- Step 1 [CronParser.java]: String rep = expr.replace(...);
...
Intermediate Steps
Sink (getMessage(...)):
public class CronValidator implements ConstraintValidator {
public boolean isValid(String value, ...) {
...
} catch (IllegalArgu
ctx.disableDefaultCo
ctx.build…(e.getMessage()).addCo…(); // ← SINK
return false;
} ... } }
Context surrounding the sink
Marking the exact sink location in comment
Large Language Model (zero-shot)
LLM JSON Response (GPT-4)
{
"explanation": "The source is user input being parsed and validated, which is a common entry point for tainted data. The sink is an error message being used in a way that could potentially be executed by downstream code, fitting the criteria for a CWE-094 vulnerability if the error message is mishandled. However, without evidence of the error message being executed, it's speculative to confirm vulnerability solely based on this dataflow.",
"source_is_false_positive": false,
"sink_is_false_positive": false,
"is_vulnerable": true
}

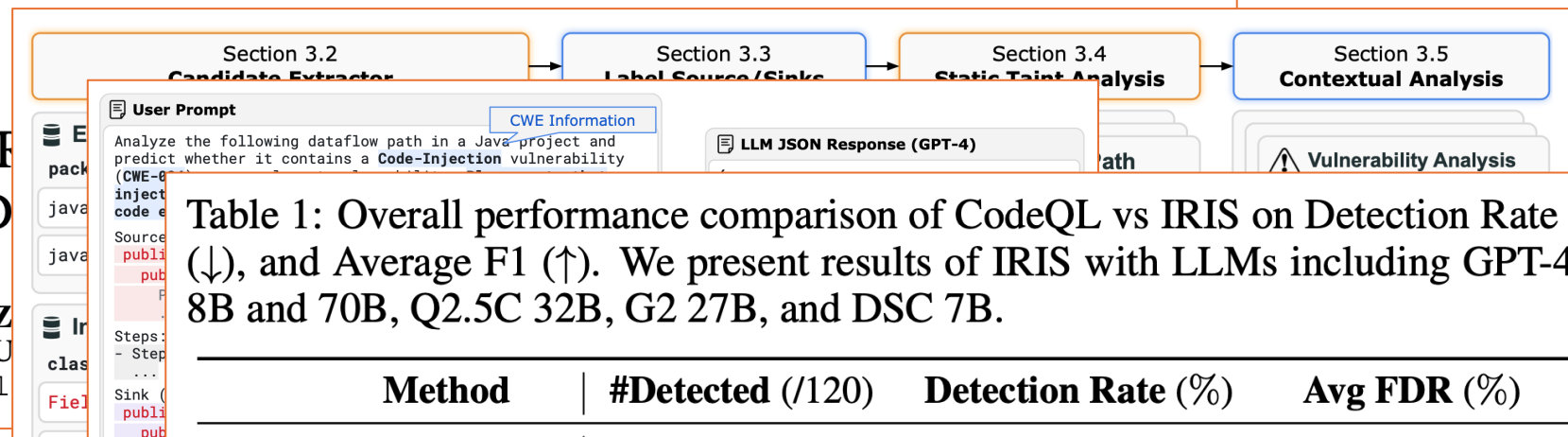


Table 1: Overall performance comparison of CodeQL vs IRIS on Detection Rate (↑), Average FDR (↓), and Average F1 (↑). We present results of IRIS with LLMs including GPT-4 and GPT-3.5, L3 8B and 70B, Q2.5C 32B, G2 27B, and DSC 7B.

	Method	#Detected (/120)	Detection Rate (%)	Avg FDR (%)	Avg F1 Score
	CodeQL	27	22.50	90.03	0.076
IRIS +	GPT-4	**55** (↑ 28)	**45.83** (↑ 23.33)	**84.82** (↓ 5.21)	**0.177** (↑ 0.101)
	GPT-3.5	47 (↑ 20)	39.17 (↑ 16.67)	90.42 (↑ 0.39)	0.096 (↑ 0.020)
	L3 8B	41 (↑ 14)	34.17 (↑ 11.67)	95.55 (↑ 5.52)	0.058 (↓ 0.018)
	L3 70B	54 (↑ 27)	45.00 (↑ 22.50)	90.96 (↑ 0.93)	0.113 (↑ 0.037)
	Q2.5C 32B	47 (↑ 20)	39.17 (↑ 16.67)	92.38 (↑ 2.35)	0.097 (↑ 0.021)
	G2 27B	45 (↑ 18)	37.50 (↑ 15.00)	91.23 (↑ 1.20)	0.100 (↑ 0.024)
	DSC 7B	52 (↑ 25)	43.33 (↑ 20.83)	95.40 (↑ 5.37)	0.062 (↓ 0.014)

Software Testing: Impossible Triangle

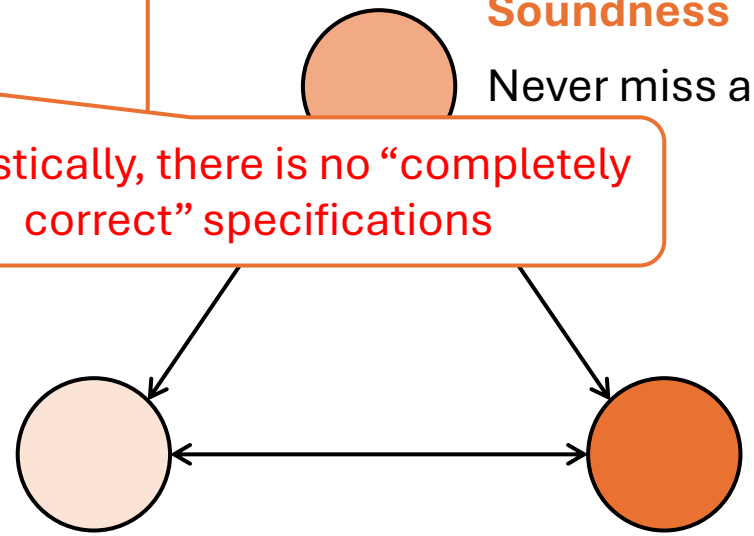
Static Analysis:

Taint Analysis

- **(assuming correct specs)**

Abstract Interpretation

- (assuming correct do



Soundness

Never miss a real bug

Realistically, there is no "completely correct" specifications

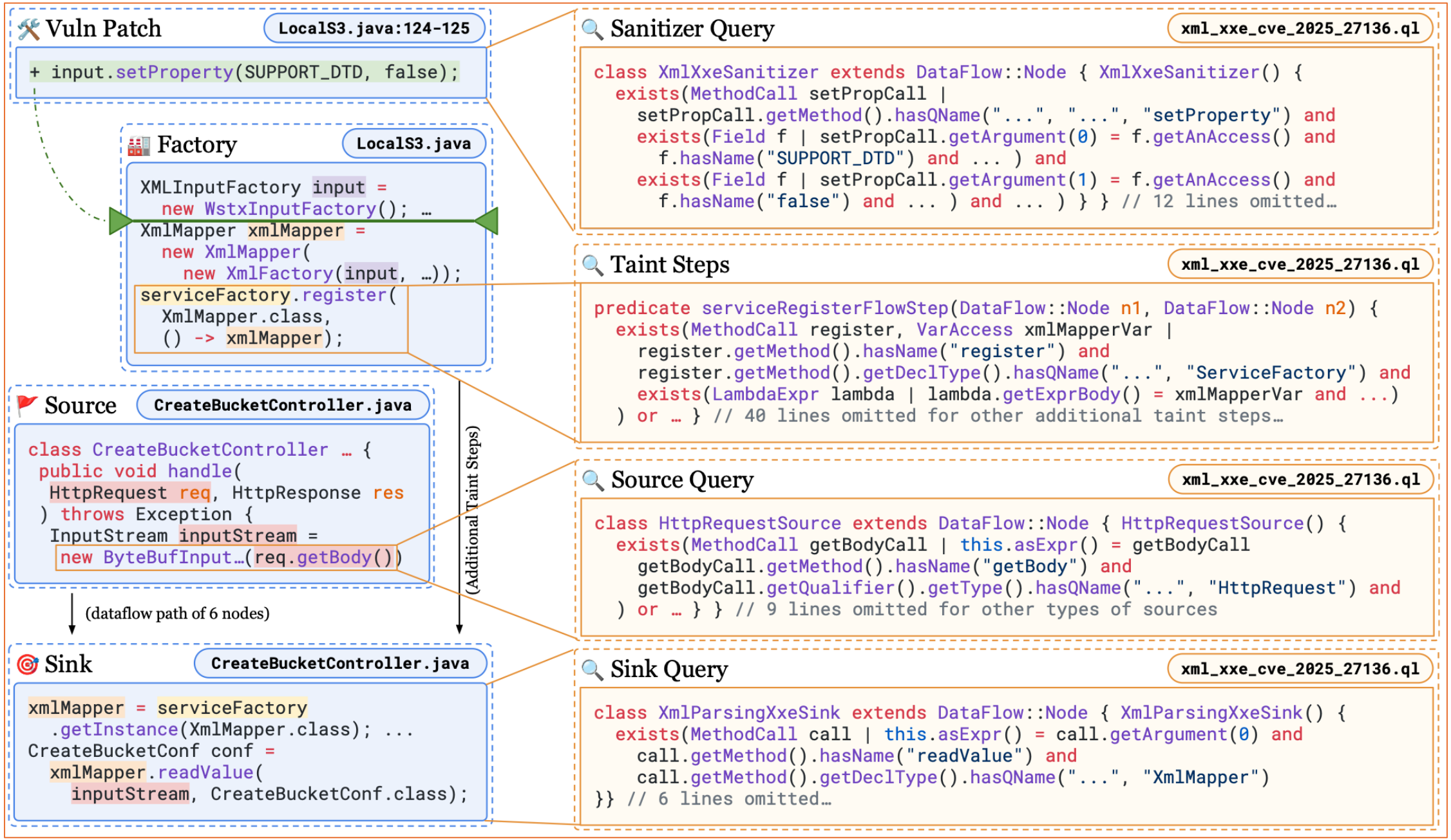
~~**Completeness**~~

~~Never raise a false alarm~~

Termination

Always finish the analysis within finite time

Dataflow Analysis & Abstract Domain generalizes to entire space of input, but loses precision


Vuln Patch
LocalS3.java:124-125
+ input.setProperty(SUPPORT_DTD, false);
Factory
LocalS3.java
XMLInputFactory input =
new WstxInputFactory(); …
XmlMapper xmlMapper =
new XmlMapper(
new XmlFactory(input, …));
serviceFactory.register(
XmlMapper.class,
() -> xmlMapper);
Source
CreateBucketController.java
class CreateBucketController … {
public void handle(
HttpRequest req, HttpResponse res
) throws Exception {
InputStream inputStream =
new ByteBufInput…(req.getBody())
(dataflow path of 6 nodes)
(Additional Taint Steps)
Sink
CreateBucketController.java
xmlMapper = serviceFactory
.getInstance(XmlMapper.class); ...
CreateBucketConf conf =
xmlMapper.readValue(
inputStream, CreateBucketConf.class);
Sanitizer Query
xml_xxe_cve_2025_27136.ql
class XmlXxeSanitizer extends DataFlow::Node { XmlXxeSanitizer() {
exists(MethodCall setPropCall |
setPropCall.getMethod().hasQName("...", "...", "setProperty") and
exists(Field f | setPropCall.getArgument(0) = f.getAnAccess() and
f.hasName("SUPPORT_DTD") and ...) and
exists(Field f | setPropCall.getArgument(1) = f.getAnAccess() and
f.hasName("false") and ...) and ...) } } // 12 lines omitted…
Taint Steps
xml_xxe_cve_2025_27136.ql
predicate serviceRegisterFlowStep(DataFlow::Node n1, DataFlow::Node n2) {
exists(MethodCall register, VarAccess xmlMapperVar |
register.getMethod().hasName("register") and
register.getMethod().getDeclType().hasQName("...", "ServiceFactory") and
exists(LambdaExpr lambda | lambda.getExprBody() = xmlMapperVar and ...)
) or … } // 40 lines omitted for other additional taint steps…
Source Query
xml_xxe_cve_2025_27136.ql
class HttpRequestSource extends DataFlow::Node { HttpRequestSource() {
exists(MethodCall getBodyCall | this.asExpr() = getBodyCall
getBodyCall.getMethod().hasName("getBody") and
getBodyCall.getQualifier().getType().hasQName("...", "HttpRequest") and
) or … } } // 9 lines omitted for other types of sources
Sink Query
xml_xxe_cve_2025_27136.ql
class XmlParsingXxeSink extends DataFlow::Node { XmlParsingXxeSink() {
exists(MethodCall call | this.asExpr() = call.getArgument(0) and
call.getMethod().hasName("readValue") and
call.getMethod().getDeclType().hasQName("...", "XmlMapper")
}} // 6 lines omitted…

Automated Static Vulnerability Detection via a Holistic Neuro-symbolic Approach

Penghui Li
Columbia University
New York, NY, USA
pl2689@columbia.edu

Songchen Yao
Columbia University
New York, NY, USA
sy2743@columbia.edu

Josef Sarfati Korich
Columbia University
New York, NY, USA
jsk2266@columbia.edu

Changhua Luo
Wuhan University
Wuhan, Hubei, China
chluo2502@whu.edu.cn

Jianjia Yu
Johns Hopkins University
Baltimore, MD, USA
jyu122@jhu.edu

Yinzhi Cao
Johns Hopkins University
Baltimore, MD, USA
yinzhi.cao@jhu.edu

Junfeng Yang
Columbia University
New York, NY, USA
junfeng@cs.columbia.edu

Automated Static Vulnerability Detection via a Holistic Neuro-symbolic Approach

Penghui Li
Columbia University
New York, NY, USA
pl2689@columbia.edu

Songchen Yao
Columbia University
New York, NY, USA

Josef Sarfati Korich
Columbia University
New York, NY, USA

Changhua Luo
Wuhan University
Wuhan, Hubei, China

Jianj
Johns Hopki
Baltimore,
jyu122@

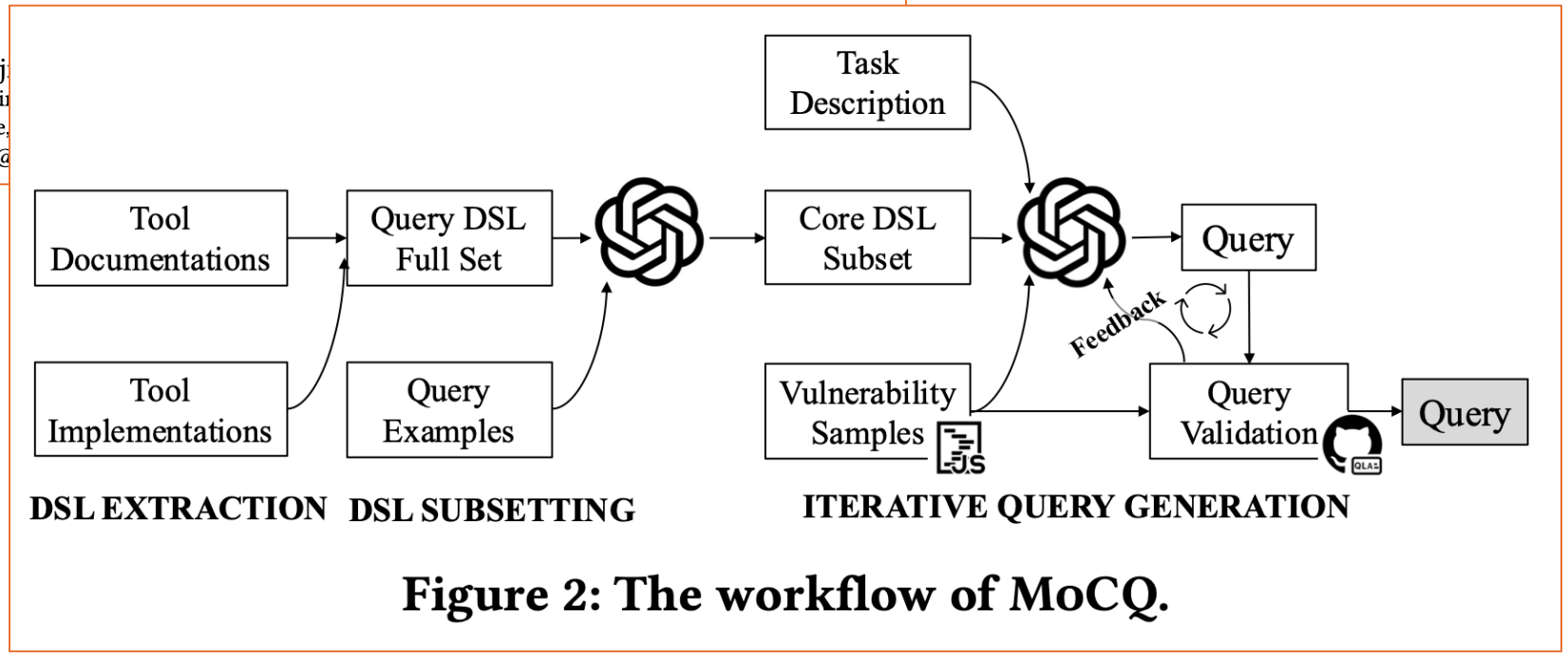


Figure 2: The workflow of MoCQ.

Automated Static Vulnerability Detection via a Holistic Neuro-symbolic Approach

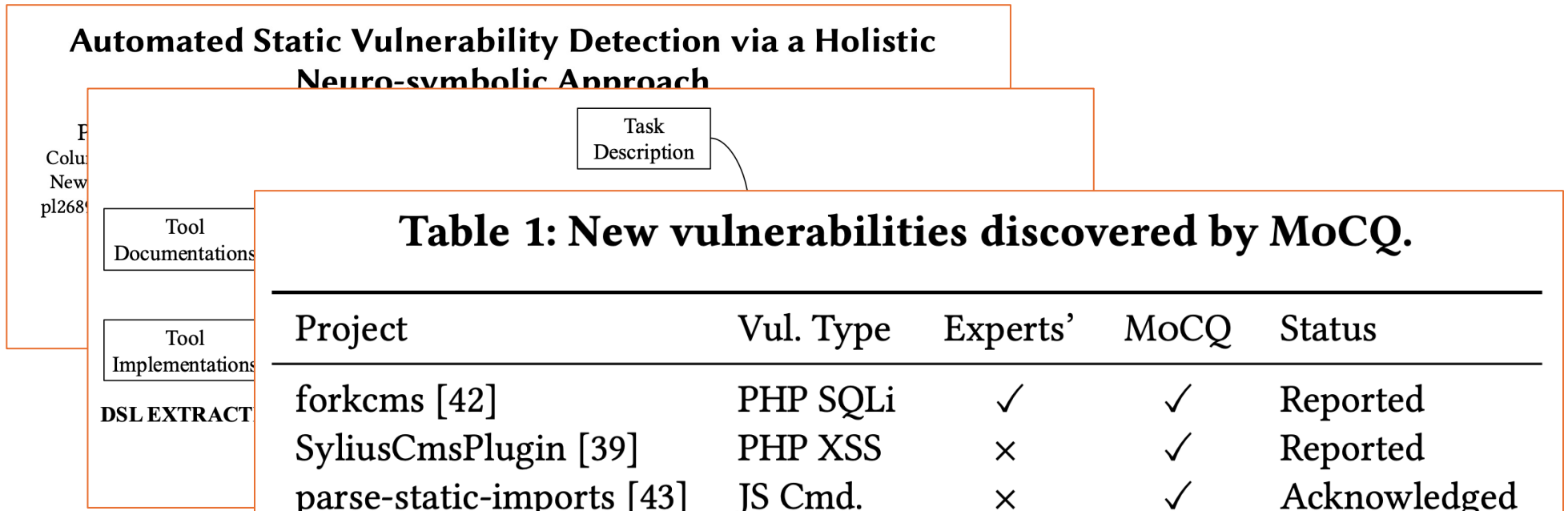
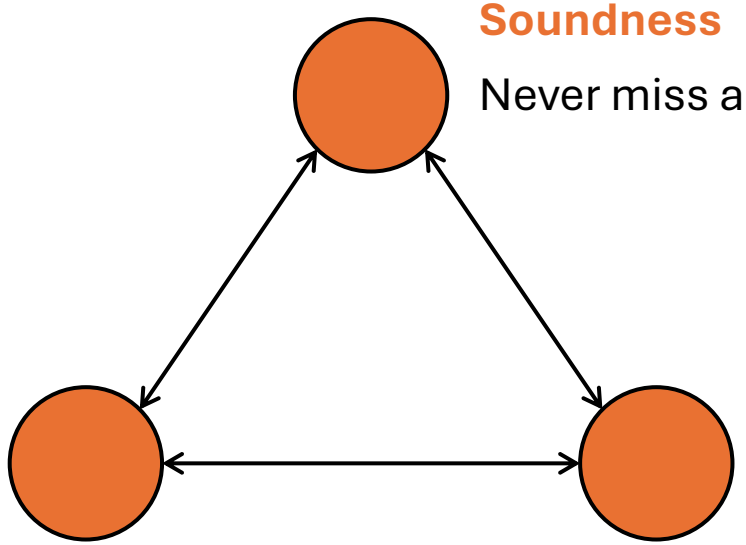


Table 1: New vulnerabilities discovered by MoCQ.

Project	Vul. Type	Experts'	MoCQ	Status
forkcms [42]	PHP SQLi	✓	✓	Reported
SyliusCmsPlugin [39]	PHP XSS	×	✓	Reported
parse-static-imports [43]	JS Cmd.	×	✓	Acknowledged
es2015-proxy [41]	JS Proto.	×	✓	Acknowledged
web-worker [45]	JS Proto.	✓	✓	Reported
koa-send [46]	JS Proto.	✓	✓	Acknowledged
content-type [40]	JS Proto.	×	✓	Reported

Software Testing: Impossible Triangle



Soundness

Never miss a real bug

Completeness

Never raise a false alarm

Termination

Always finish the analysis within finite time

Logistics – Week 9

- Assignment 3: Coding LLM Agents
  - https://github.com/machine-programming/assignment-3
  - Fully functional web-app agent. Due: Oct 23 (Thu)
- Forming groups for your final projects!
  - Form a group of 2-3 before This Thursday (~~Oct 19~~ Oct 23)