Static Analysis – Part 1

Claire Le Goues

Learning goals

- Give a one sentence definition of static analysis. Explain what types of bugs static analysis targets.
- Give an example of syntactic or structural static analysis.
- Construct basic control flow graphs for small examples by hand.
- Distinguish between control- and data-flow analyses; define and then step through on code examples simple control and data-flow analyses.
- Implement a dataflow analysis.
- Explain at a high level why static analyses cannot be sound, complete, and terminating; assess tradeoffs in analysis design.
- Characterize and choose between tools that perform static analyses.

Two fundamental concepts

- **Abstraction.**
  - Elide details of a specific implementation.
  - Capture semantically relevant details; ignore the rest.
- **Programs as data.**
  - Programs are just trees/graphs!
  - …and we know lots of ways to analyze trees/graphs, right?

```
goto fail;
```

```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa,
                                 SSLBuffer signedParams,
                                 uint8_t *signature,
                                 UInt16 signatureLen) {
    OSStatus err;
     .…
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
          goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
          goto fail;
          goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
          goto fail;
    …
fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;
}
```

```
/* from Linux 2.3.99 drivers/block/raid5.c */
static struct buffer_head *
get_free_buffer(struct stripe_head * sh,
                int b_size) {
  struct buffer_head *bh;
  unsigned long flags;
  save_flags(flags);
  cli(); // disables interrupts
  if ((bh = sh->buffer_pool) == NULL)
    return NULL;
  sh->buffer_pool = bh -> b_next;
  bh->b_size = b_size;
  restore_flags(flags); // re-enables interrupts
  return bh;
}
```

ERROR: function returns with interrupts disabled!

With thanks to Jonathan Aldrich; example from Engler et al., *Checking system rules Using System-Specific, Programmer-Written Compiler Extensions*, OSDI ‘000

Could you have found them?

- How often would those bugs trigger?
- Driver bug:
  - What happens if you return from a driver with interrupts disabled?
  - Consider: that's one function
    - …in a 2000 LOC file
    - …in a module with 60,000 LOC
    - …IN THE LINUX KERNEL
- **Moral:** *Some defects are very difficult to find via testing, inspection.*

CNET › News › Security & Privacy › Klocwork: Our source code analyzer caught Apple's '...

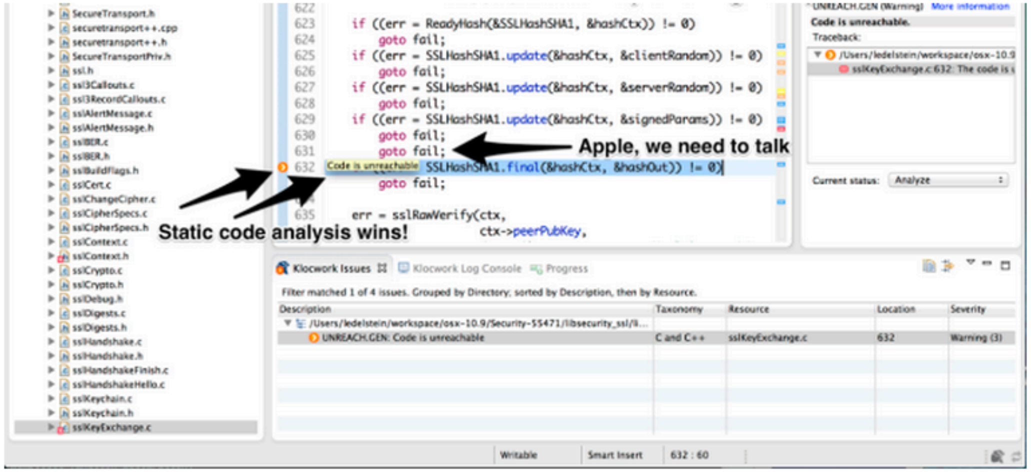
Klocwork: Our source code analyzer caught Apple's 'gotofail' bug

If Apple had used a third-party source code analyzer on its encryption library, it could have avoided the "gotofail" bug.

by Declan McCullagh | February 28, 2014 1:13 PM PST

Follow

57 | 223 | 23 | 5 | More + | Comments 25



```
    if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;

    err = sslRawVerify(ctx,
                       ctx->peerPubKey,
```

UNREACH.GEN (Warning) More information

Code is unreachable.

Traceback:

/Users/ledelstein/workspace/osx-10.9
sslKeyExchange.c:632: The code is u

Current status: Analyze

Filter matched 1 of 4 issues. Grouped by Directory; sorted by Description, then by Resource.

Description	Taxonomy	Resource	Location	Severity
/Users/ledelstein/workspace/osx-10.9/Security-55471/libsecurity_ssl/li...				
UNREACH.GEN: Code is unreachable	C and C++	sslKeyExchange.c	632	Warning (3)

Klocwork's Larry Edelstein sent us this screen snapshot, complete with the arrows, showing how the company's product would have nabbed the "goto fail" bug.

(Credit: Klocwork)

It was a single repeated line of code -- "goto fail" -- that left millions of Apple users vulnerable to Internet attacks until the company finally fixed it Tuesday.

Featured Posts

Google unveils Androi wearables
Internet & Media

Motorol powere
Internet

OK, Glas in my fa
Cutting E

Apple iP product
Apple

iPad wit comeba
Apple

Most Popular

Giant 3D house
6k Faceb

Exclusiv Doesch
716 Twe

Google' four can
771 Goo

Connect With CNET

Facebook
Like Us

Google +

http://www.cnet.com/8301-1009_3-57619754-83/klocwork-our-source-code-analyzer-caught-apples-gotofail-bug/

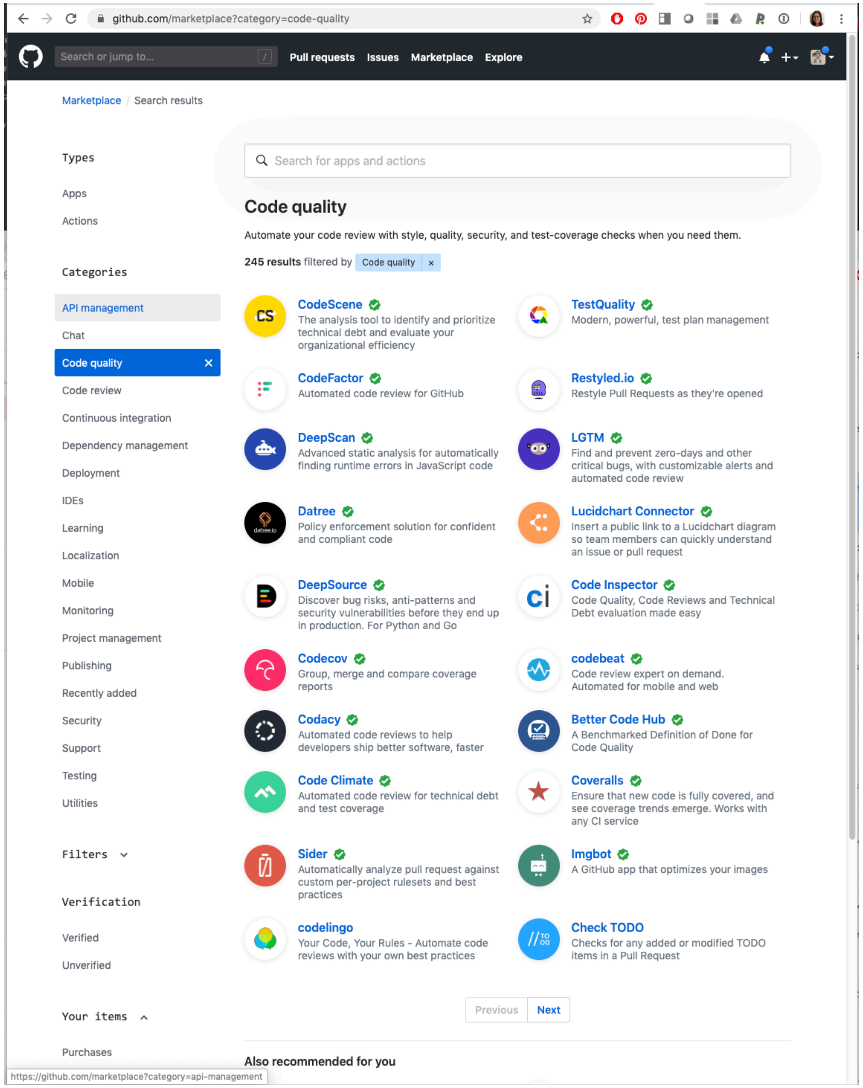
Defects of interest…

- Are on uncommon or difficult-to-force execution paths. (vs testing)
- Executing (or interpreting/otherwise analyzing) all paths concretely to find such defects is infeasible.
- **What we really want to do is check the entire possible state space of the program for particular properties.**

Defects Static Analysis can Catch

- **Defects that result from inconsistently following simple, mechanical design rules.**
  - **Security:** Buffer overruns, improperly validated input.
  - **Memory safety:** Null dereference, uninitialized data.
  - **Resource leaks:** Memory, OS resources.
  - **API Protocols:** Device drivers; real time libraries; GUI frameworks.
  - **Exceptions:** Arithmetic/library/user-defined
  - **Encapsulation:** Accessing internal data, calling private functions.
  - **Data races:** Two threads access the same data without synchronization

Key: check compliance to simple, mechanical design rules



Code quality

Automate your code review with style, quality, security, and test-coverage checks when you need them.

245 results filtered by Code quality ×

CodeScene The analysis tool to identify and prioritize technical debt and evaluate your organizational efficiency

TestQuality Modern, powerful, test plan management

CodeFactor Automated code review for GitHub

Restyled.io Restyle Pull Requests as they're opened

DeepScan Advanced static analysis for automatically finding runtime errors in JavaScript code

LGTM Find and prevent zero-days and other critical bugs, with customizable alerts and automated code review

Datree Policy enforcement solution for confident and compliant code

Lucidchart Connector Insert a public link to a Lucidchart diagram so team members can quickly understand an issue or pull request

DeepSource Discover bug risks, anti-patterns and security vulnerabilities before they end up in production. For Python and Go

Code Inspector Code Quality, Code Reviews and Technical Debt evaluation made easy

Codecov Group, merge and compare coverage reports

codebeat Code review expert on demand. Automated for mobile and web

Codacy Automated code reviews to help developers ship better software, faster

Better Code Hub A Benchmarked Definition of Done for Code Quality

Code Climate Automated code review for technical debt and test coverage

Coveralls Ensure that new code is fully covered, and see coverage trends emerge. Works with any CI service

Sider Automatically analyze pull request against custom per-project rulesets and best practices

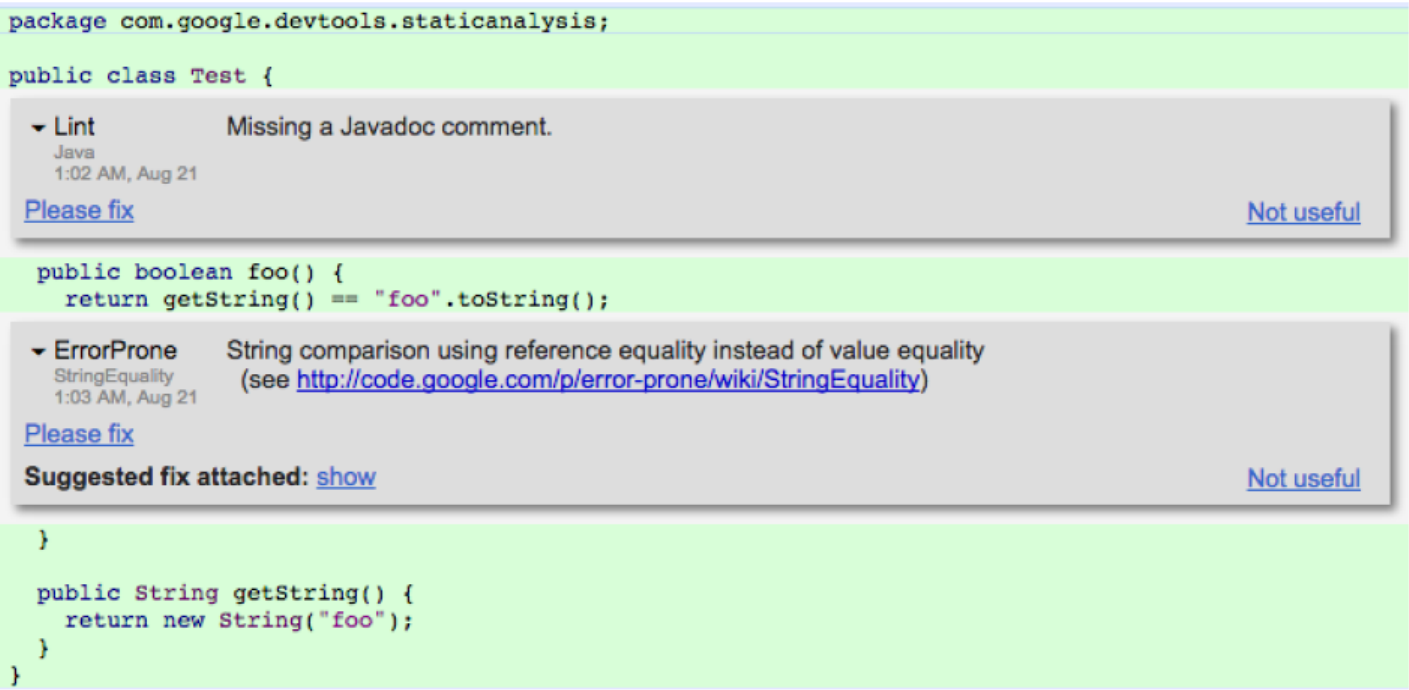
Imgbot A GitHub app that optimizes your images

codelingo Your Code, Your Rules - Automate code reviews with your own best practices

Check TODO Checks for any added or modified TODO items in a Pull Request

Also recommended for you

https://github.com/marketplace?category=code-quality



```
package com.google.devtools.staticanalysis;

public class Test {
```

▾ Lint
Java
1:02 AM, Aug 21

Missing a Javadoc comment.

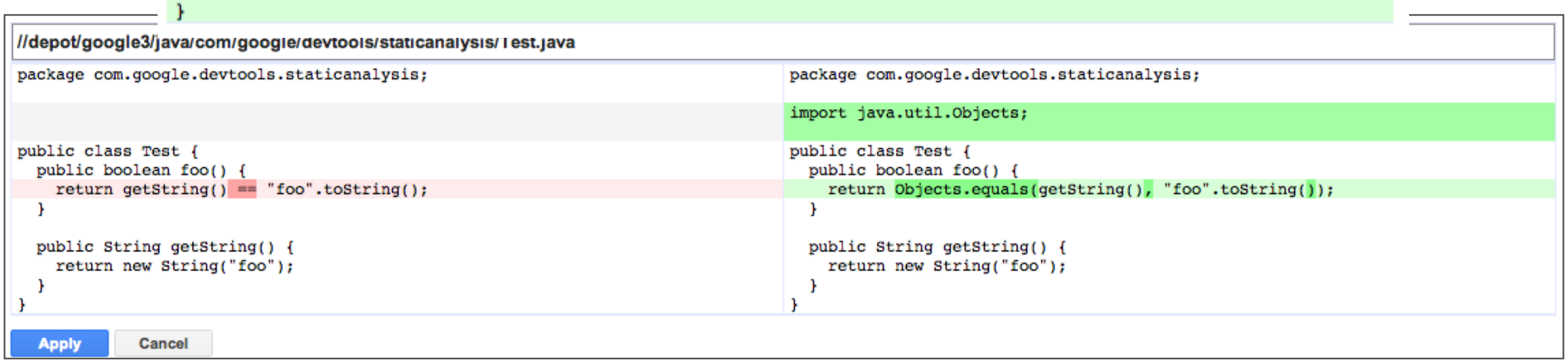
```
  public boolean foo() {
    return getString() == "foo".toString();
```

▾ ErrorProne
StringEquality
1:03 AM, Aug 21

String comparison using reference equality instead of value equality (see http://code.google.com/p/error-prone/wiki/StringEquality)

Suggested fix attached: show

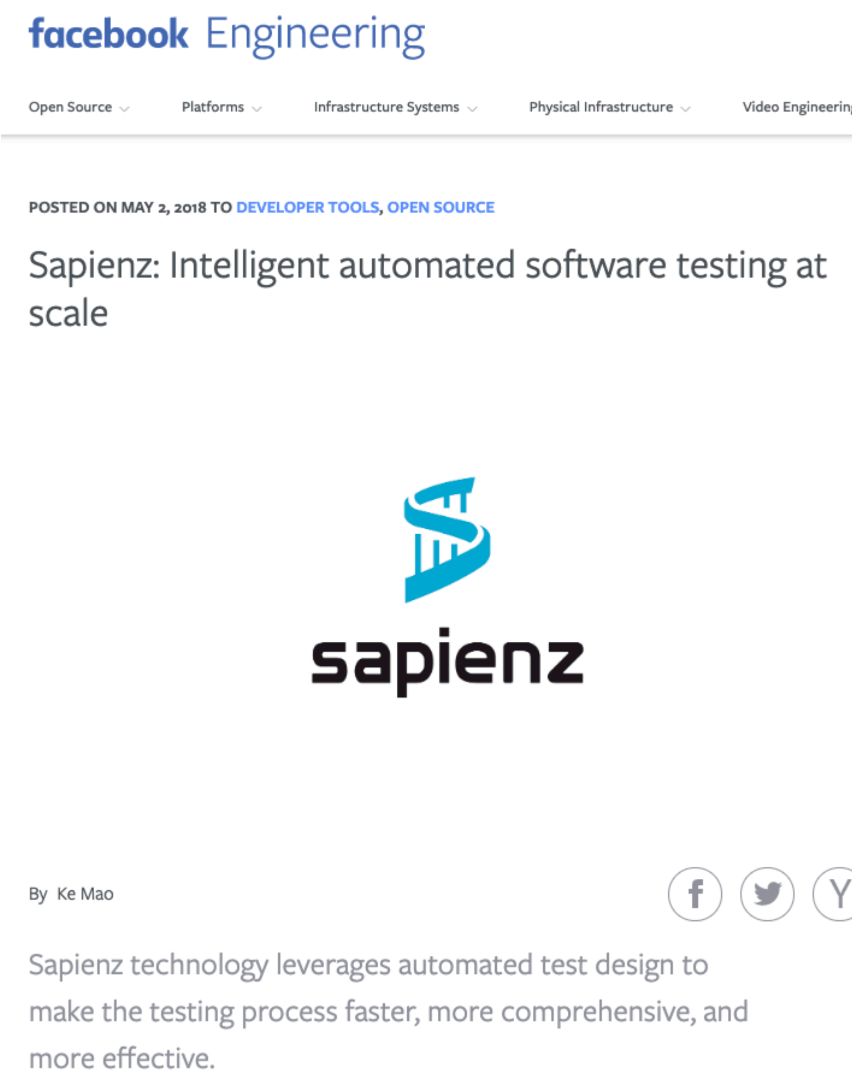
```
  }

  public String getString() {
    return new String("foo");
  }
}
```



//depot/google3/java/com/google/devtools/staticanalysis/Test.java

```
package com.google.devtools.staticanalysis;


public class Test {
  public boolean foo() {
    return getString() == "foo".toString();
  }

  public String getString() {
    return new String("foo");
  }
}
```

```
package com.google.devtools.staticanalysis;

import java.util.Objects;

public class Test {
  public boolean foo() {
    return Objects.equals(getString(), "foo".toString());
  }

  public String getString() {
    return new String("foo");
  }
}
```



facebook Engineering

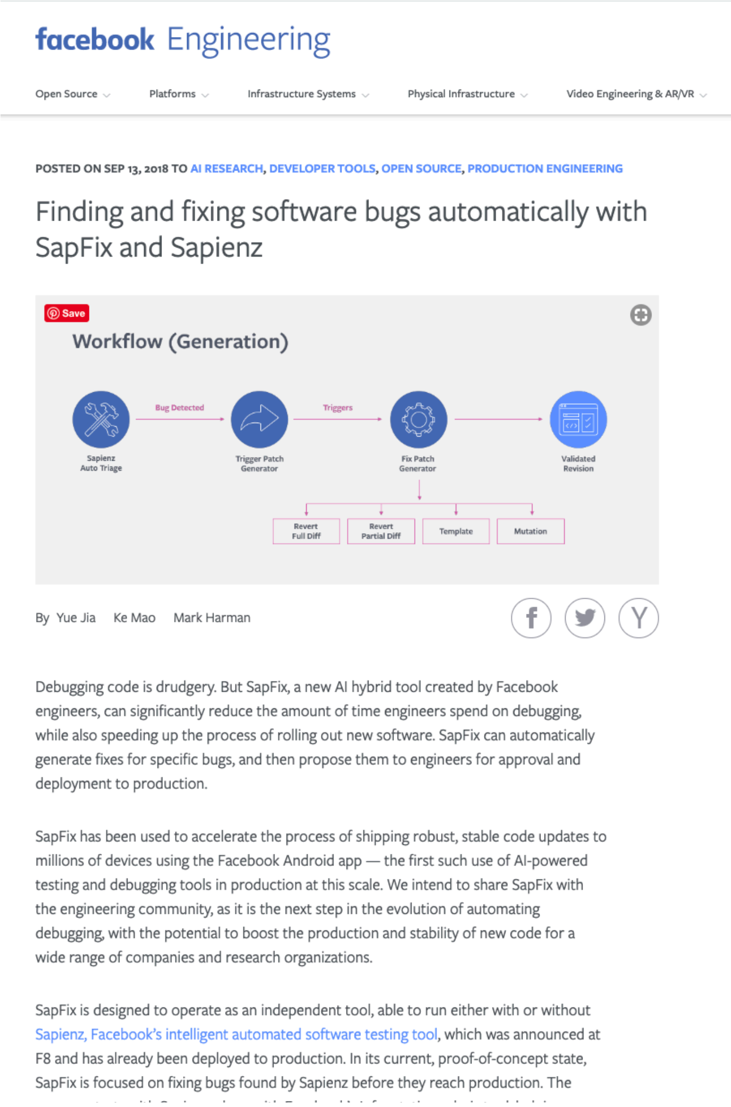
Open Source ⌵ Platforms ⌵ Infrastructure Systems ⌵ Physical Infrastructure ⌵ Video Engineerin

POSTED ON MAY 2, 2018 TO DEVELOPER TOOLS, OPEN SOURCE

Sapienz: Intelligent automated software testing at scale

By Ke Mao

Sapienz technology leverages automated test design to make the testing process faster, more comprehensive, and more effective.



facebook Engineering

Open Source ⌵ Platforms ⌵ Infrastructure Systems ⌵ Physical Infrastructure ⌵ Video Engineering & AR/VR ⌵

POSTED ON SEP 13, 2018 TO AI RESEARCH, DEVELOPER TOOLS, OPEN SOURCE, PRODUCTION ENGINEERING

Finding and fixing software bugs automatically with SapFix and Sapienz

By Yue Jia Ke Mao Mark Harman

Debugging code is drudgery. But SapFix, a new AI hybrid tool created by Facebook engineers, can significantly reduce the amount of time engineers spend on debugging, while also speeding up the process of rolling out new software. SapFix can automatically generate fixes for specific bugs, and then propose them to engineers for approval and deployment to production.

SapFix has been used to accelerate the process of shipping robust, stable code updates to millions of devices using the Facebook Android app — the first such use of AI-powered testing and debugging tools in production at this scale. We intend to share SapFix with the engineering community, as it is the next step in the evolution of automating debugging, with the potential to boost the production and stability of new code for a wide range of companies and research organizations.

SapFix is designed to operate as an independent tool, able to run either with or without Sapienz, Facebook's intelligent automated software testing tool, which was announced at F8 and has already been deployed to production. In its current, proof-of-concept state, SapFix is focused on fixing bugs found by Sapienz before they reach production. The

DEFINING STATIC ANALYSIS

What is Static Analysis?

- **Systematic** examination of an **abstraction** of program **state space**.
  - Does not execute code! (like code review)
- **Abstraction:** produce a representation of a program that is simpler to analyze.
  - Results in fewer states to explore; makes difficult problems tractable.
- Check if a **particular property** holds over the entire state space:
  - Liveness: “something good eventually happens.”
  - Safety: “this bad thing can’t ever happen.”
  - Compliance with mechanical design rules.

The Bad News: Rice's Theorem

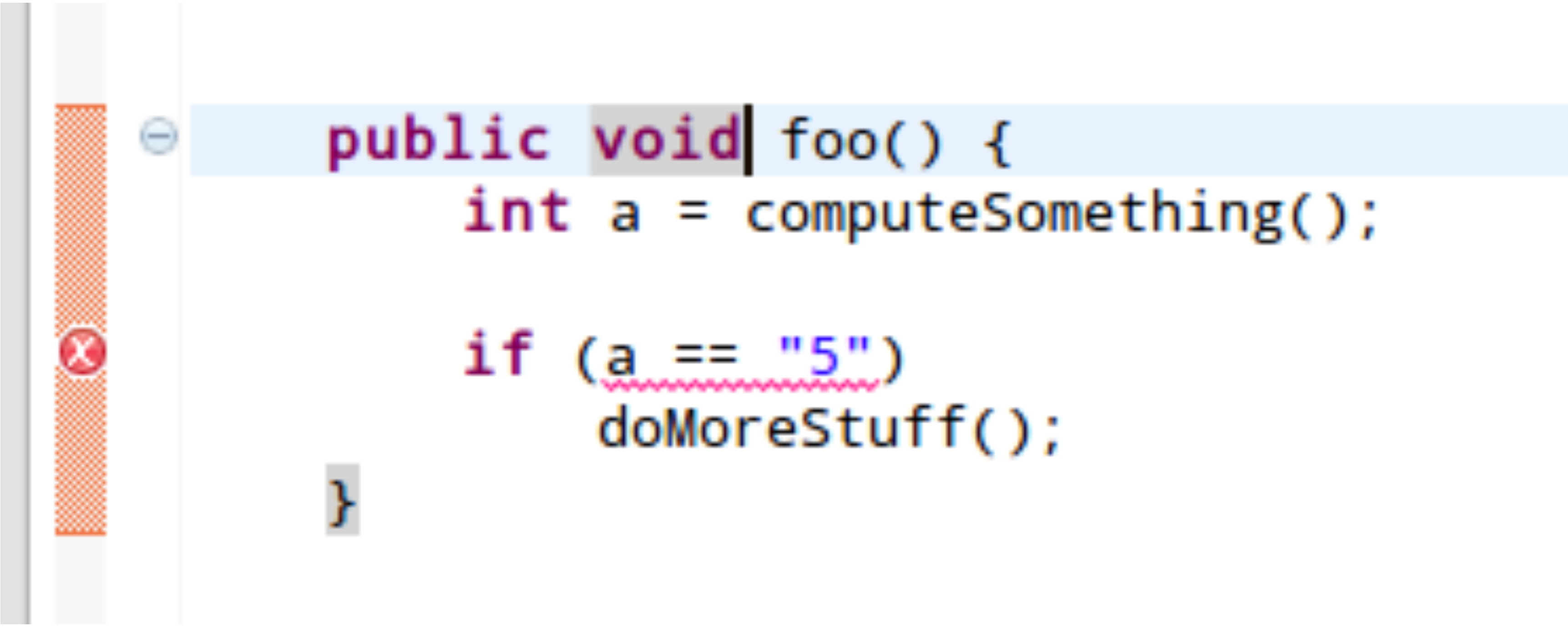
"Any nontrivial property about the language recognized by a Turing machine is undecidable.“

Henry Gordon Rice, 1953

Every static analysis is necessarily incomplete or unsound or undecidable (or multiple of these)

SIMPLE SYNTACTIC AND STRUCTURAL ANALYSES

Type Analysis

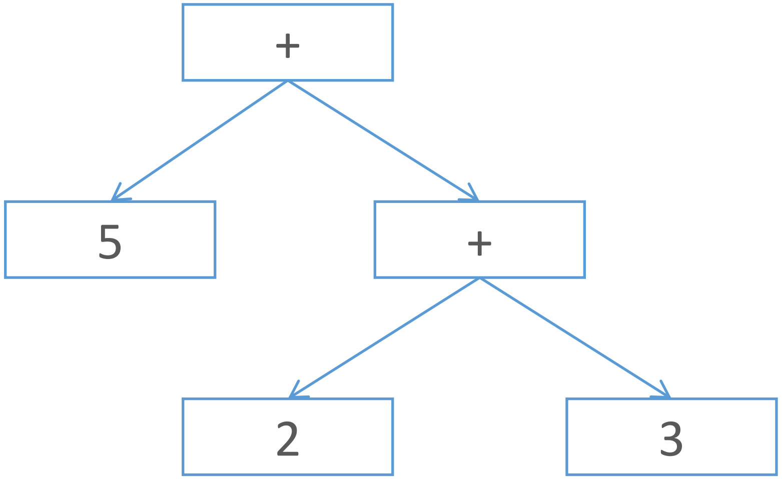


```
public void foo() {
        int a = computeSomething();

        if (a == "5")
            doMoreStuff();
}
```

Abstraction: abstract syntax tree

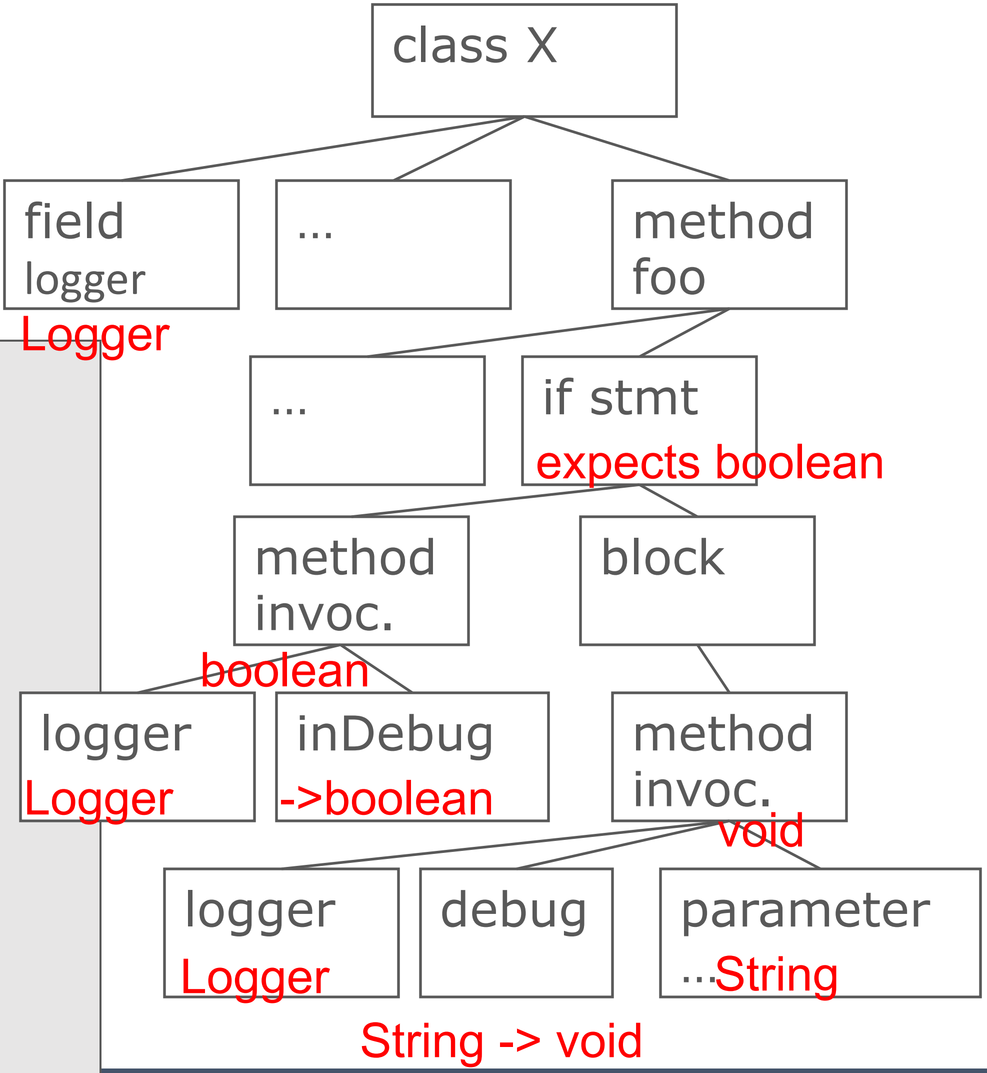
- Tree representation of the syntactic structure of source code.
  - Parsers convert concrete syntax into abstract syntax, and deal with resulting ambiguities.
- Records only the semantically relevant information.
  - Abstract: doesn't represent every detail (like parentheses); these can be inferred from the structure.
- (How to build one? Take compilers!)

- Example: 5 + (2 + 3)



Type checking

```
class X {
  Logger logger;
  public void foo() {
    …
    if (logger.inDebug()) {
      logger.debug(“We have ” +
conn + “connections.”);
    }
  }
}
class Logger {
   boolean inDebug() {…}
   void debug(String msg) {…}
}
```



class X

field logger

Logger

…

method foo

…

if stmt

expects boolean

method invoc.

boolean

block

logger

Logger

inDebug

->boolean

method invoc.

void

logger

Logger

debug

parameter

…String

String -> void

Syntactic Analysis

Find every occurrence of this pattern:

```
public foo() {
   …
   logger.debug(“We have ” + conn + “connections.”);
}
```

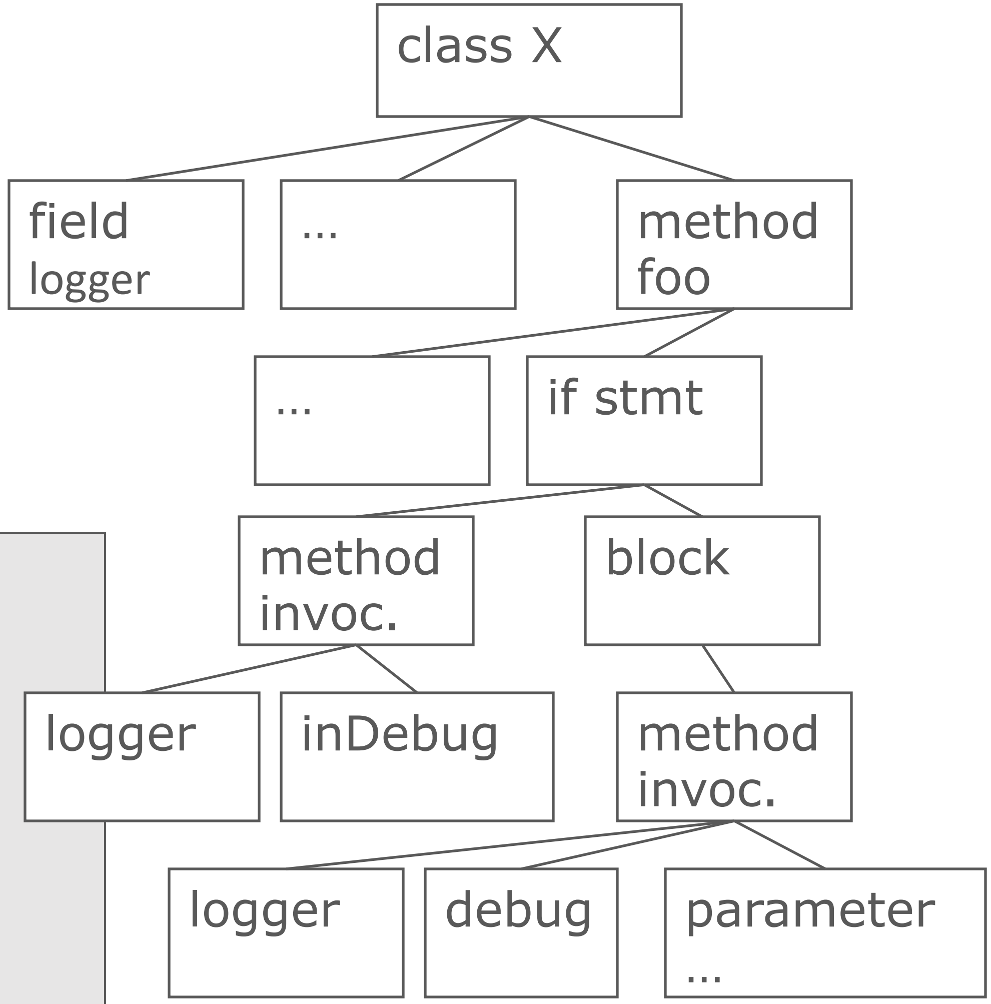
```
public foo() {
   …
   if (logger.inDebug()) {
      logger.debug(“We have ” + conn + “connections.”);
   }
}
```

grep "if \(logger\.inDebug" . -r

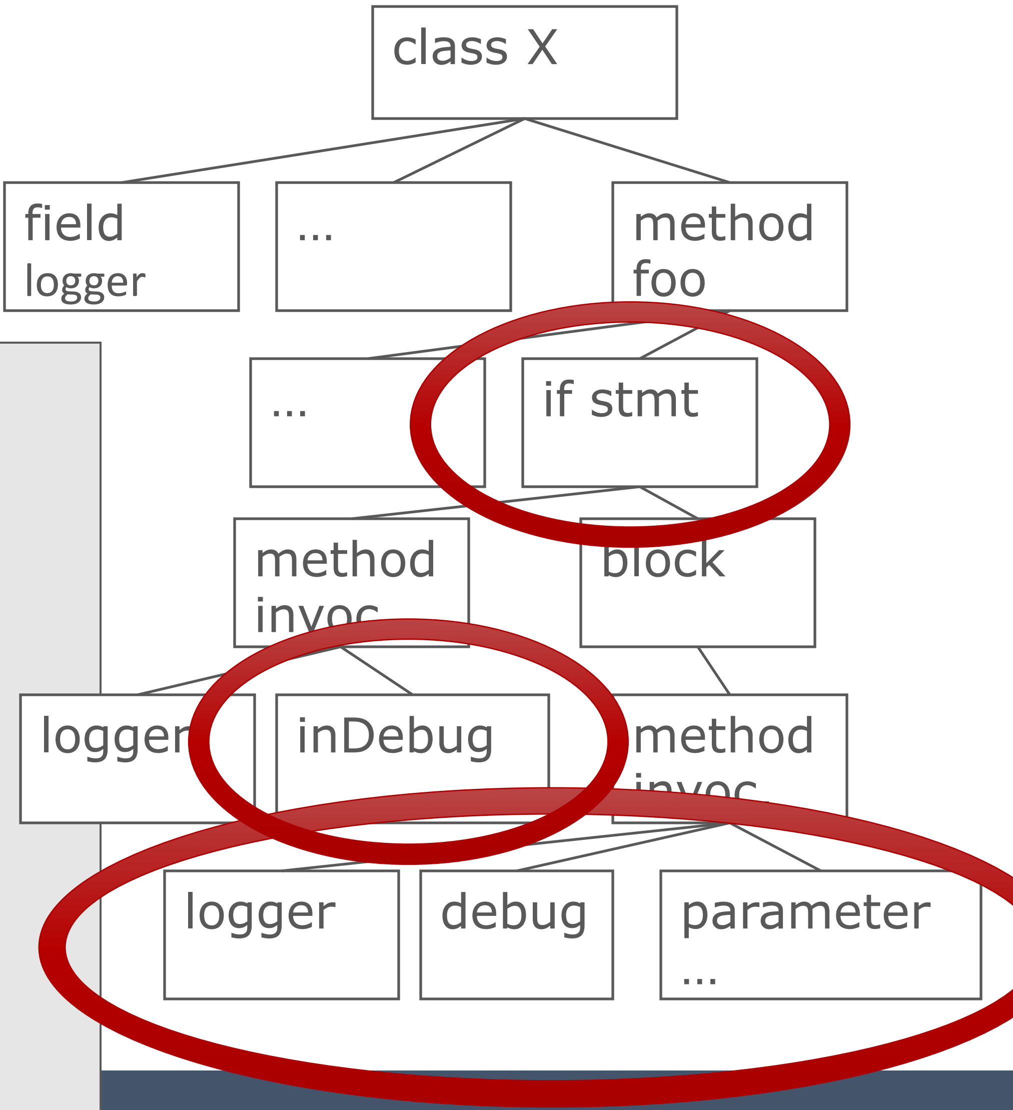
Abstract syntax tree walker

- Check that we don't create strings outside of a `Logger.inDebug` check
- Abstraction:
  - Look only for calls to `Logger.debug()`
  - Make sure they're all surrounded by if (`Logger.inDebug()`)
- Systematic: Checks all the code
- Known as an Abstract Syntax Tree (AST) walker
  - Treats the code as a structured tree
  - Ignores control flow, variable values, and the heap
  - Code style checkers work the same way

Structural Analysis



```
class X {
  Logger logger;
  public void foo() {
    …
    if (logger.inDebug()) {
      logger.debug(“We have ” +
conn + “connections.”);
    }
  }
}
```



class X

field logger

…

method foo

…

if stmt

method invoc

block

logger

inDebug

method invoc

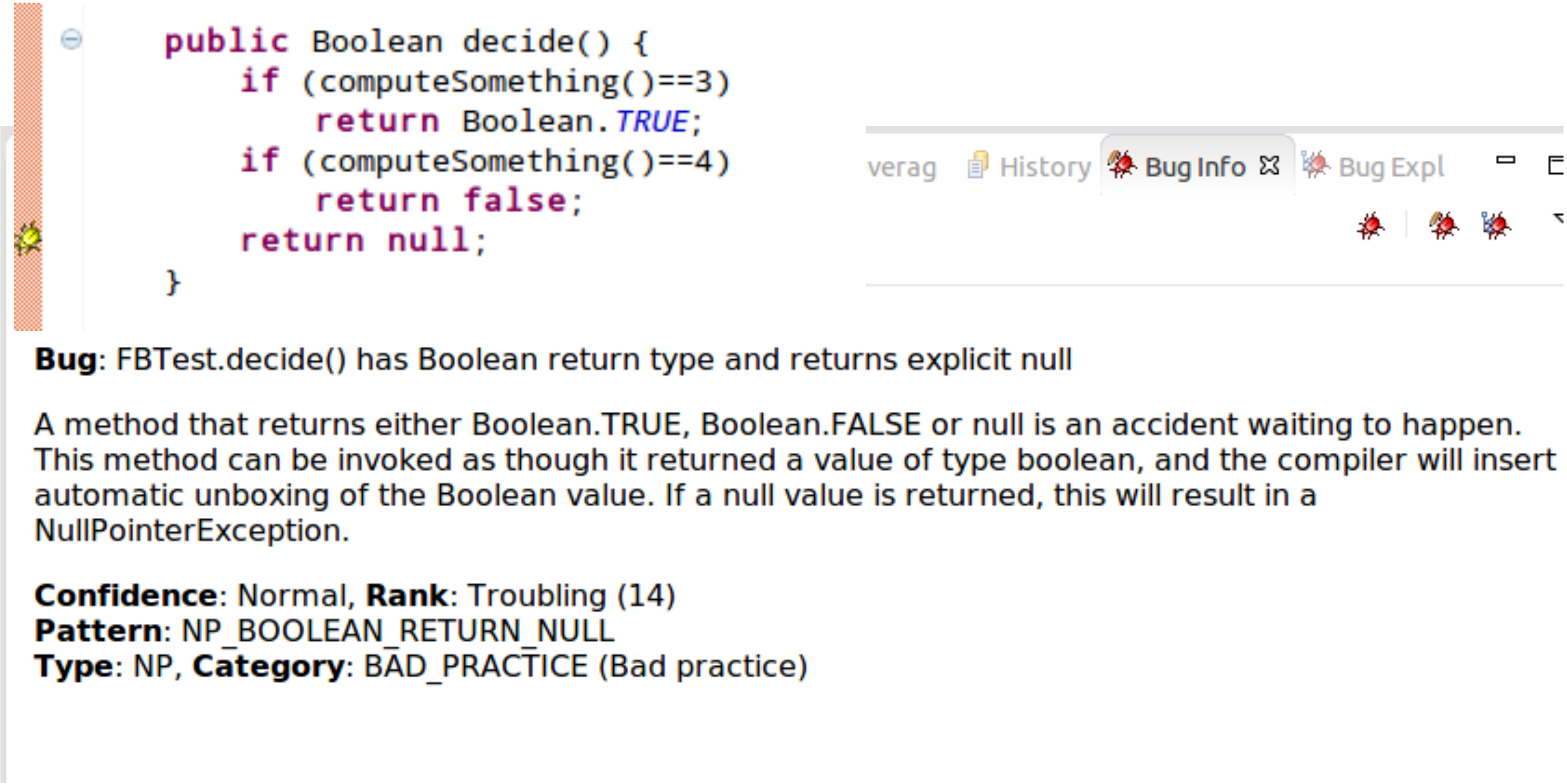
logger

debug

parameter …

```
class X {
  Logger logger;
  public void foo() {
    …
    if (logger.inDebug()) {
      logger.debug("We have " +
conn + "connections.");
    }
  }
}
class Logger {
   boolean inDebug() {…}
   void debug(String msg) {…}
}
```

Bug finding



```
public Boolean decide() {
    if (computeSomething()==3)
        return Boolean.TRUE;
    if (computeSomething()==4)
        return false;
    return null;
}
```

Bug: FBTest.decide() has Boolean return type and returns explicit null

A method that returns either Boolean.TRUE, Boolean.FALSE or null is an accident waiting to happen. This method can be invoked as though it returned a value of type boolean, and the compiler will insert automatic unboxing of the Boolean value. If a null value is returned, this will result in a NullPointerException.

Confidence: Normal, **Rank**: Troubling (14)
Pattern: NP_BOOLEAN_RETURN_NULL
Type: NP, **Category**: BAD_PRACTICE (Bad practice)

Structural Analysis to Detect Goto Fail?

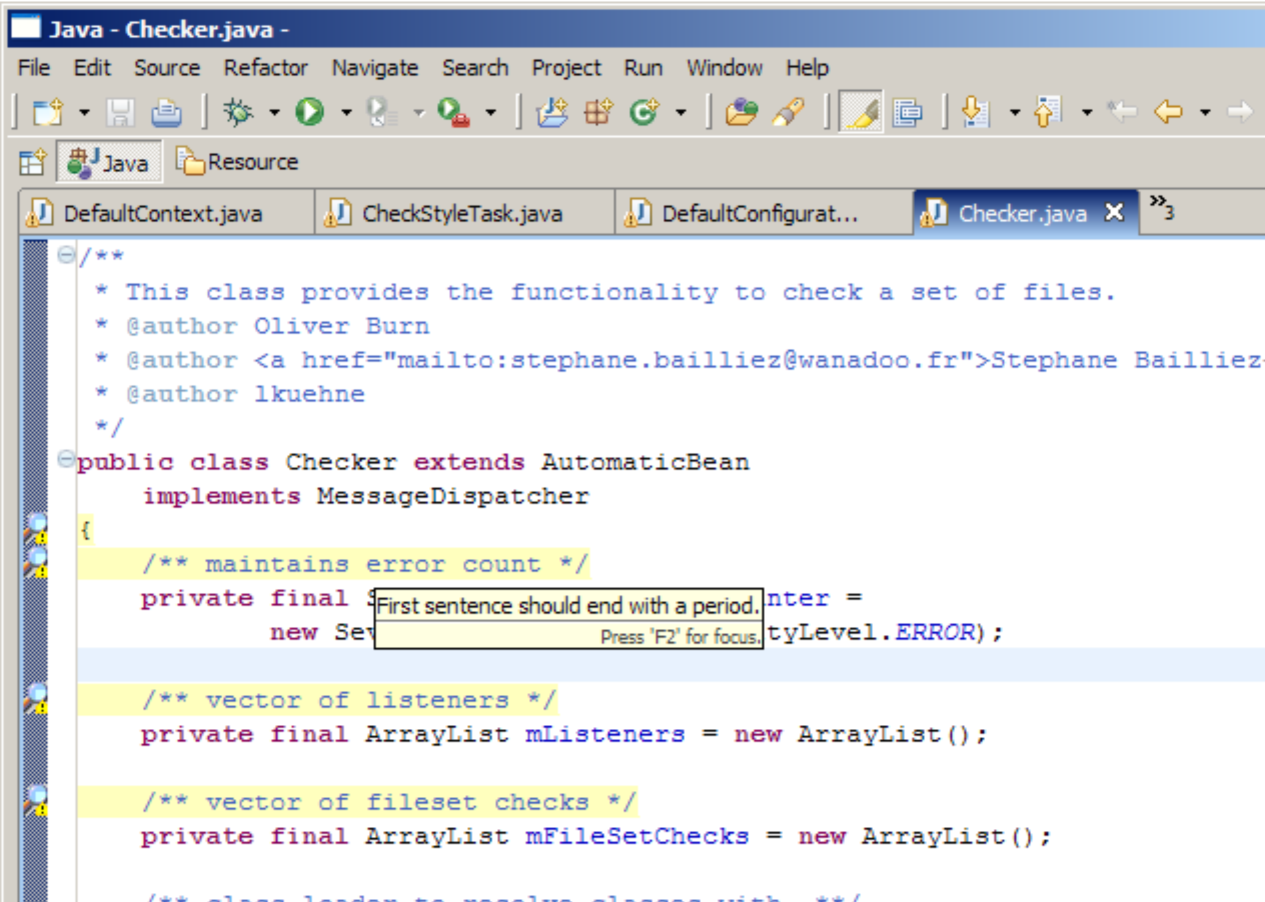
```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa,
                                 SSLBuffer signedParams,
                                 uint8_t *signature,
                                 UInt16 signatureLen) {
   OSStatus err;
    .…
   if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
          goto fail;
   if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
          goto fail;
          goto fail;
   if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
          goto fail;
   …
fail:
   SSLFreeBuffer(&signedHashes);
   SSLFreeBuffer(&hashCtx);
```

Summary: Syntactic/Structural Analyses

- Analyzing token streams or code structures (ASTs)
- Useful to find patterns
- Local/structural properties, independent of execution paths

Summary: Syntactic/Structural Analyses

- Tools include Checkstyle, many linters (C, JS, Python, …), Findbugs, others



```
/**
 * This class provides the functionality to check a set of files.
 * @author Oliver Burn
 * @author <a href="mailto:stephane.bailliez@wanadoo.fr">Stephane Bailliez
 * @author lkuehne
 */
public class Checker extends AutomaticBean
    implements MessageDispatcher
{
    /** maintains error count */
    private final S                                        nter =
        new Sev                                            tyLevel.ERROR);

    /** vector of listeners */
    private final ArrayList mListeners = new ArrayList();

    /** vector of fileset checks */
    private final ArrayList mFileSetChecks = new ArrayList();
```

First sentence should end with a period.

Press 'F2' for focus.

Tools: Compilers

- Type checking, proper initialization API, correct API usage

Program	Compiler output

`int add(int x,int y) {`
`  return x+y;`
`}`
 	
`void main() {`
`  add(2);`
`}`
	`$> error: too few arguments to function ‘int add(int, int)’`

CONTROL-FLOW ANALYSIS

Control/Dataflow analysis

- **Reason** about all possible executions, via paths through a *control flow graph*.
  - Track information relevant to a property of interest at every *program point*.
  - Including exception handling, function calls, etc
- Define an **abstract domain** that captures only the values/states relevant to the property of interest.
- **Track** the abstract state, rather than all possible concrete values, for all possible executions (paths!) through the graph.

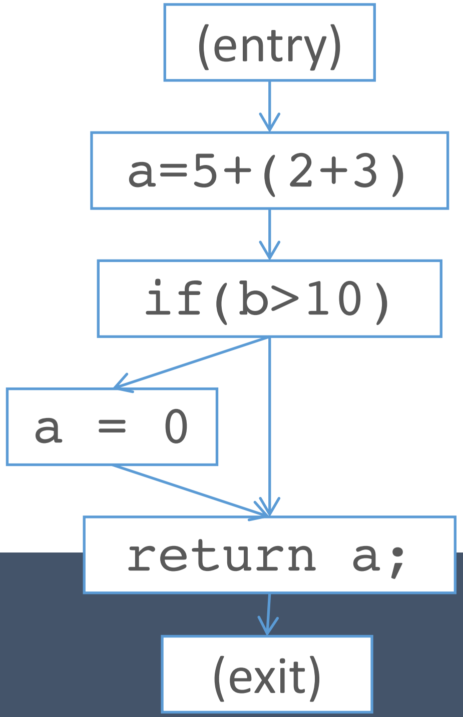
Control/Dataflow analysis

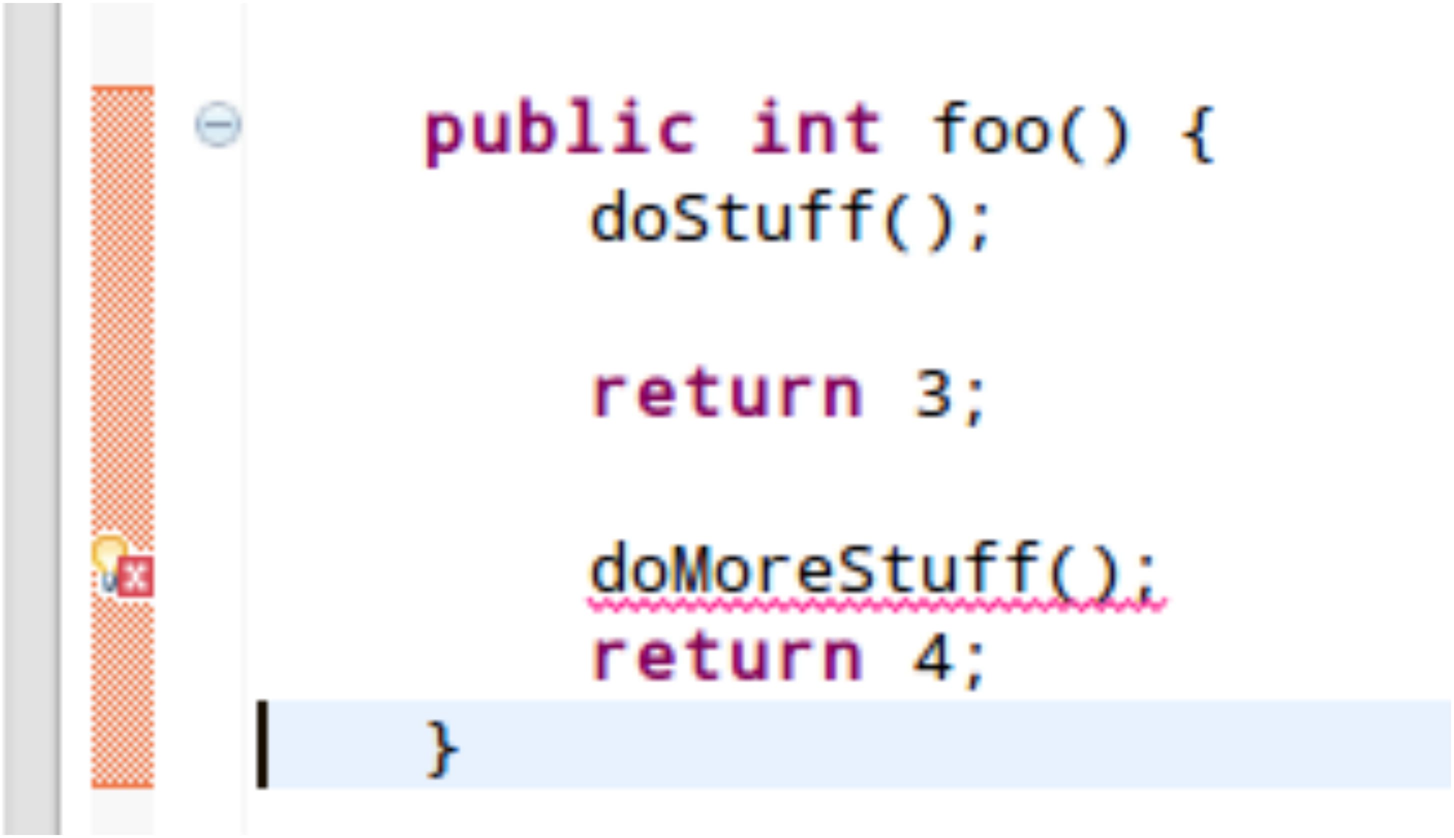
- **Reason** about all possible executions, via paths through a *control flow graph*.
  - Track information relevant to a property of interest at every *program point*.
- Define an **abstract domain** that captures only the values/states relevant to the property of interest.
- **Track** the abstract state, rather than all possible concrete values, for all possible executions (paths!) through the graph.

Control flow graphs

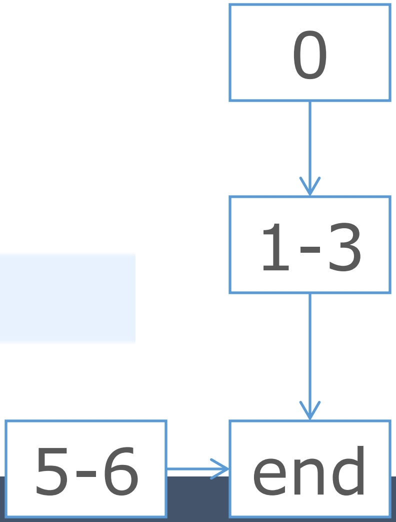
- A tree/graph-based representation of the flow of control through the program.
  - Captures all possible execution paths.
- Each node is a basic block: no jumps in or out.
- Edges represent control flow options between nodes.
- Intra-procedural: within one function.
  - cf. inter-procedural

```
a = 5 + (2 + 3)
if (b > 10) {
    a = 0;
}
return a;
```





```
public int foo() {
    doStuff();

    return 3;

    doMoreStuff();
    return 4;
}
```



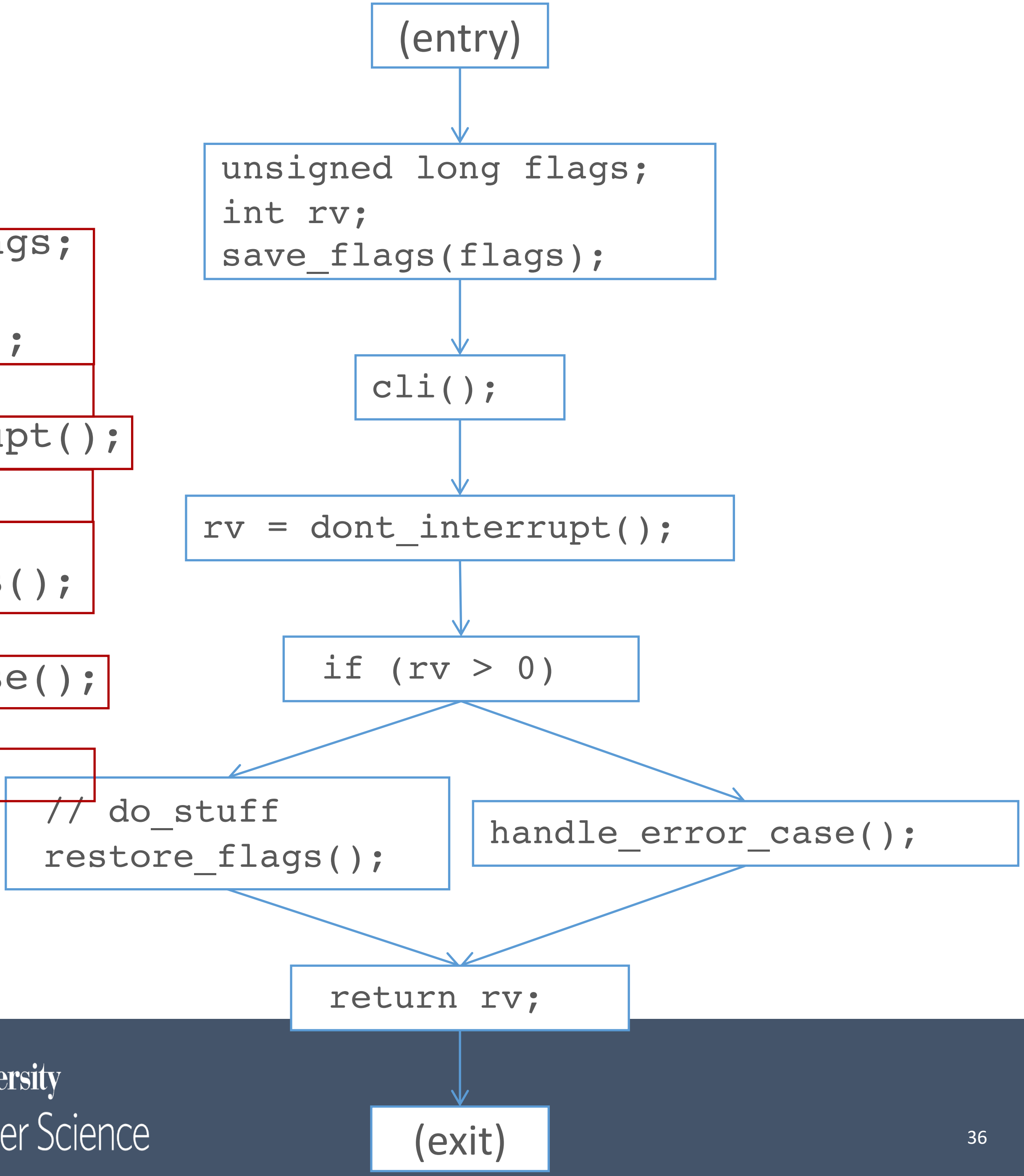
```
/* from Linux 2.3.99 drivers/block/raid5.c */
static struct buffer_head *
get_free_buffer(struct stripe_head * sh,
                int b_size) {
  struct buffer_head *bh;
  unsigned long flags;
  save_flags(flags);
  cli(); // disables interrupts
  if ((bh = sh->buffer_pool) == NULL)
    return NULL;
  sh->buffer_pool = bh -> b_next;
  bh->b_size = b_size;
  restore_flags(flags); // re-enables interrupts
  return bh;
}
```

Draw control-flow graph for this function

gler et

00

```
int foo() {
    unsigned long flags;
    int rv;
    save_flags(flags);
    cli();
    rv = dont_interrupt();
    if (rv > 0) {
       // do_stuff
          restore_flags();
    } else {
     handle_error_case();
    }
    return rv;
}
```



(entry)

```
unsigned long flags;
int rv;
save_flags(flags);
```

```
cli();
```

```
rv = dont_interrupt();
```

```
if (rv > 0)
```

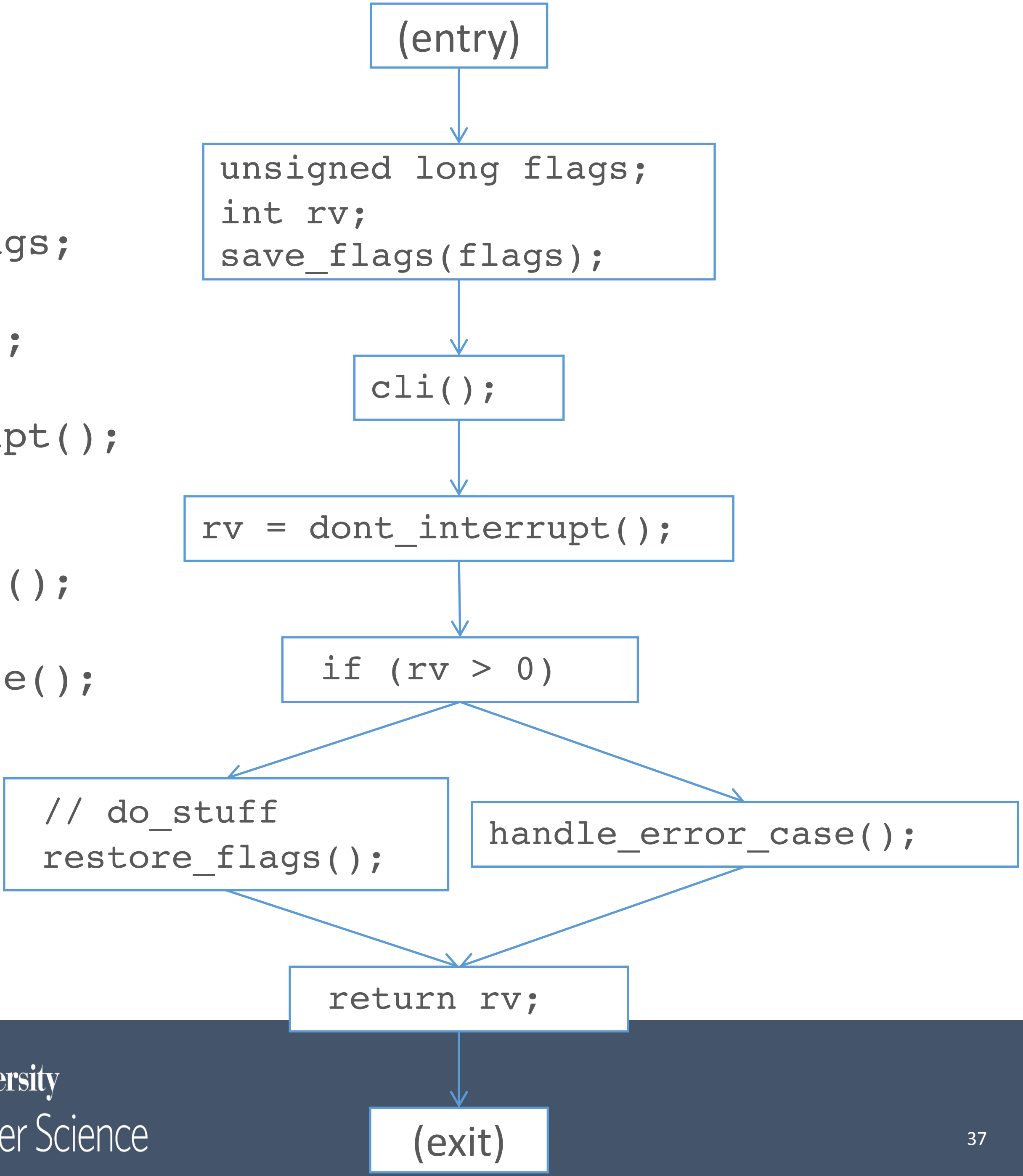
```
// do_stuff
restore_flags();
```

```
handle_error_case();
```

```
return rv;
```

(exit)

```
int foo() {
   unsigned long flags;
   int rv;
   save_flags(flags);
   cli();
   rv = dont_interrupt();
   if (rv > 0) {
      // do_stuff
       restore_flags();
   } else {
    handle_error_case();
   }
   return rv;
}
```



(entry)

```
unsigned long flags;
int rv;
save_flags(flags);
```

```
cli();
```

```
rv = dont_interrupt();
```

```
if (rv > 0)
```

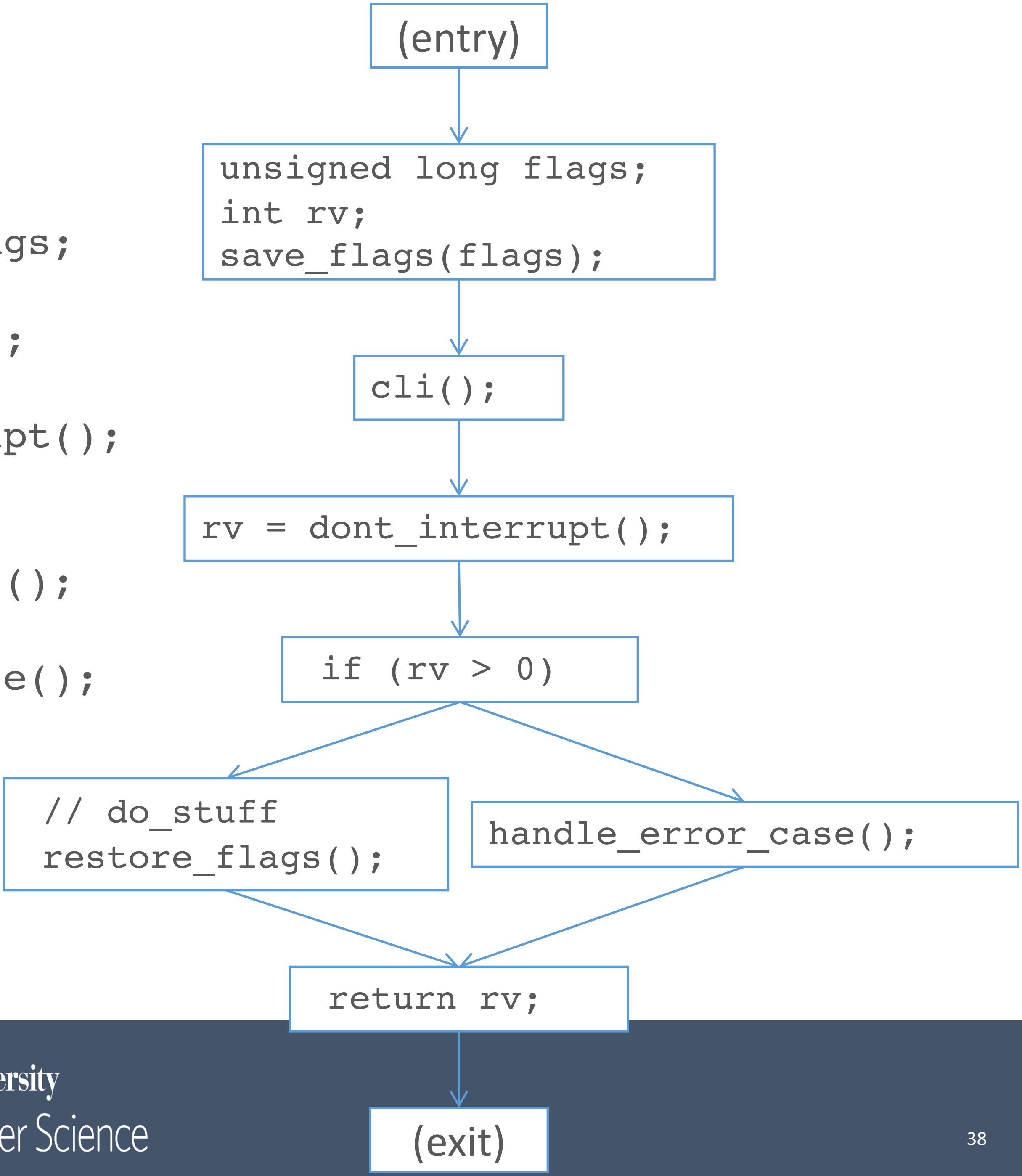
```
// do_stuff
restore_flags();
```

```
handle_error_case();
```

```
return rv;
```

(exit)

```
int foo() {
   unsigned long flags;
   int rv;
   save_flags(flags);
   cli();
   rv = dont_interrupt();
   while (rv > 0) {
      // do_stuff
       restore_flags();
   } else {
    handle_error_case();
   }
   return rv;
}
```



```
unsigned long flags;
int rv;
save_flags(flags);
```

```
cli();
```

```
rv = dont_interrupt();
```

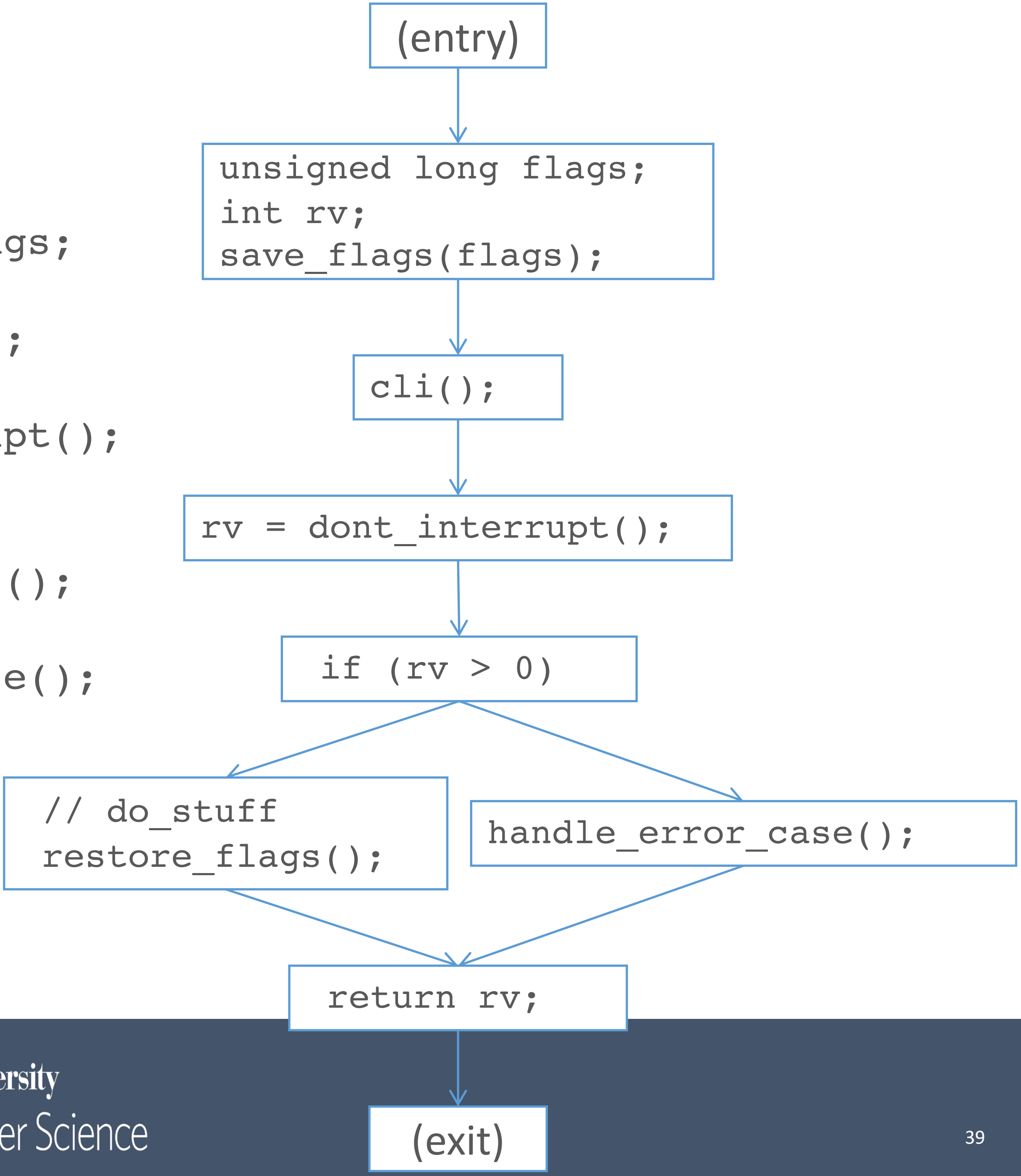
```
if (rv > 0)
```

```
// do_stuff
restore_flags();
```

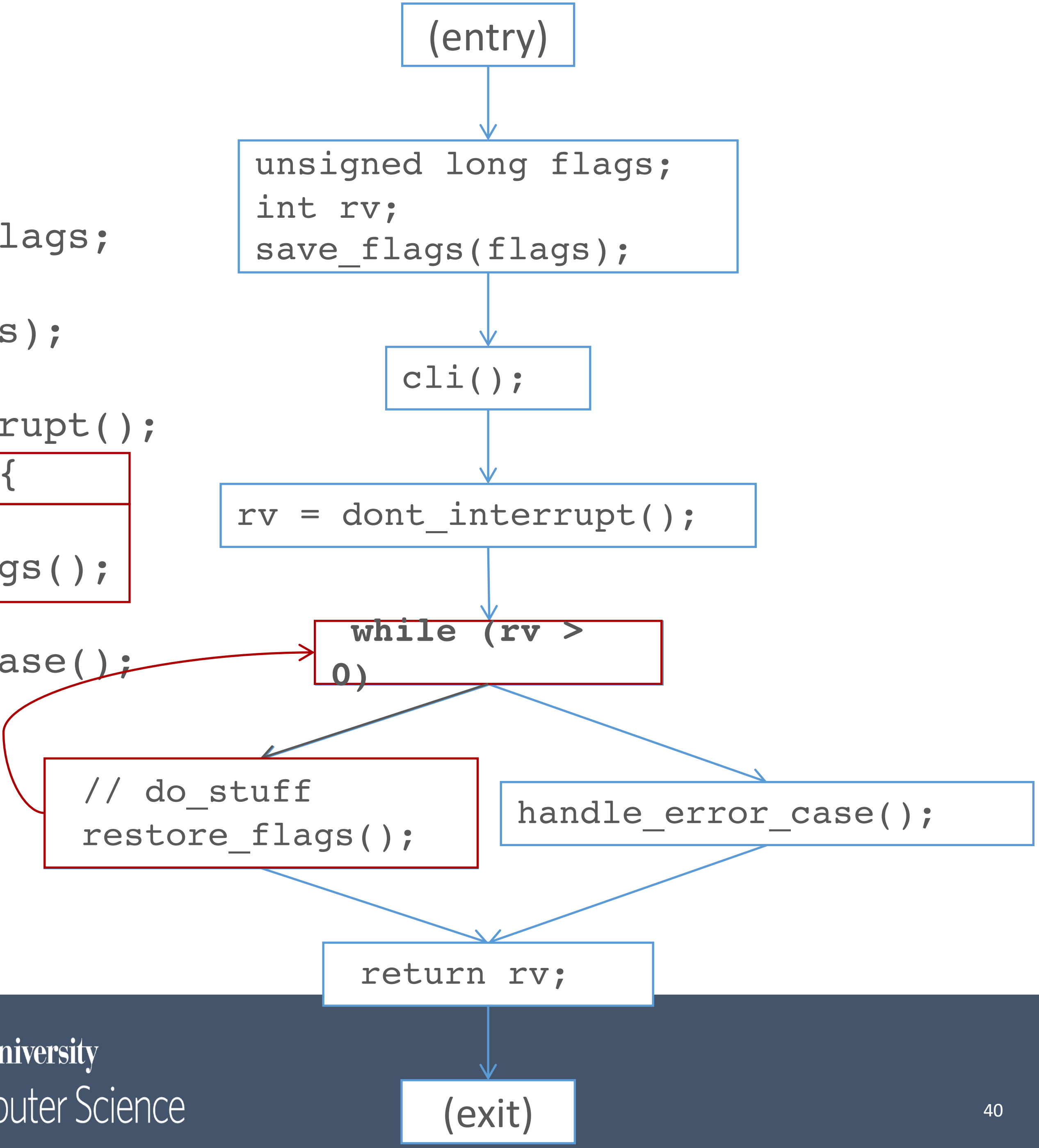
```
handle_error_case();
```

```
return rv;
```

```
int foo() {
   unsigned long flags;
   int rv;
   save_flags(flags);
   cli();
   rv = dont_interrupt();
   while (rv > 0) {
      // do_stuff
       restore_flags();
   } ~~else {~~
    handle_error_case();
   ~~}~~
   return rv;
}
```



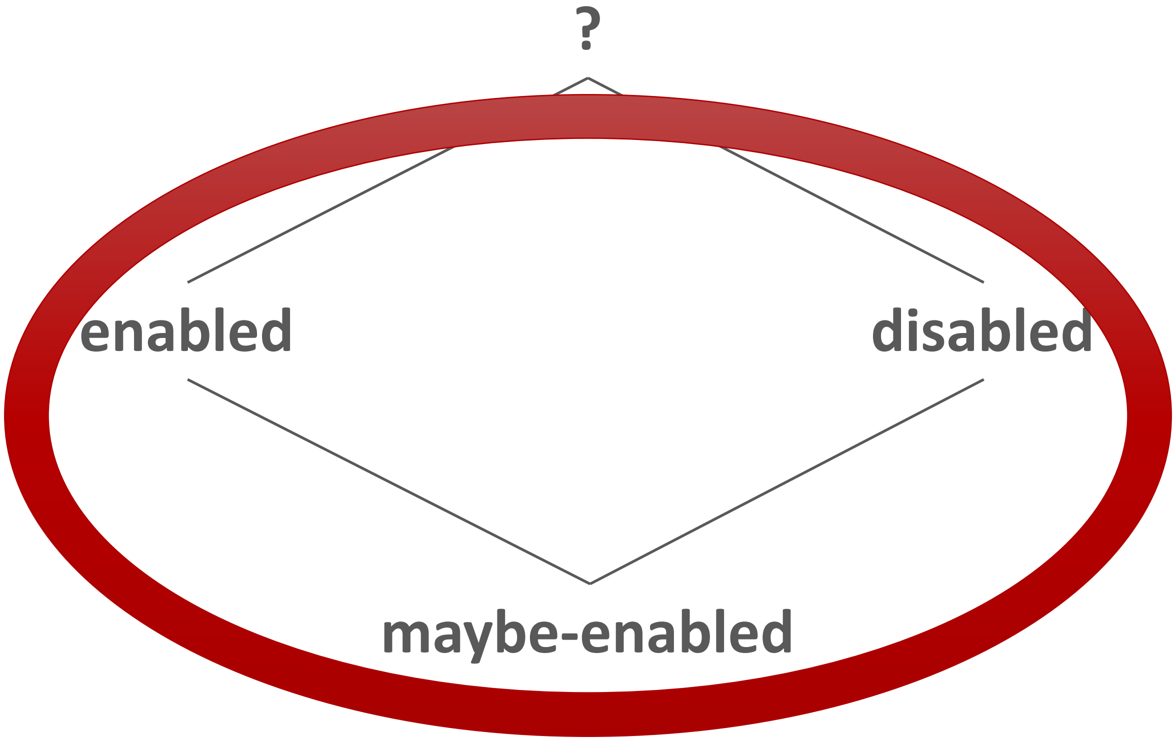
```
int foo() {
    unsigned long flags;
    int rv;
    save_flags(flags);
    cli();
    rv = dont_interrupt();
    while (rv > 0) {
       // do_stuff
        restore_flags();
    }
     handle_error_case();

    return rv;
}
```



Control/Dataflow analysis

- **Reason** about all possible executions, via paths through a *control flow graph*.
  - Track information relevant to a property of interest at every program point.
- Define an **abstract domain** that captures only the values/states relevant to the property of interest.
- **Track** the abstract state, rather than all possible concrete values, for all possible executions (paths!) through the graph.

Abstract Domain: interrupt checker

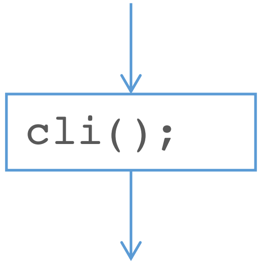


Reasoning about a CFG

- Analysis updates state at *program points:* points between nodes.
- For each node:
  - determine state on entry by examining/combining state from predecessors.
  - evaluate state on exit of node based on effect of the operations (*transfer*).
- *Iterate through successors and over entire graph until the state at each program point stops changing.*
- **Output: state at each program point**

Transfer function

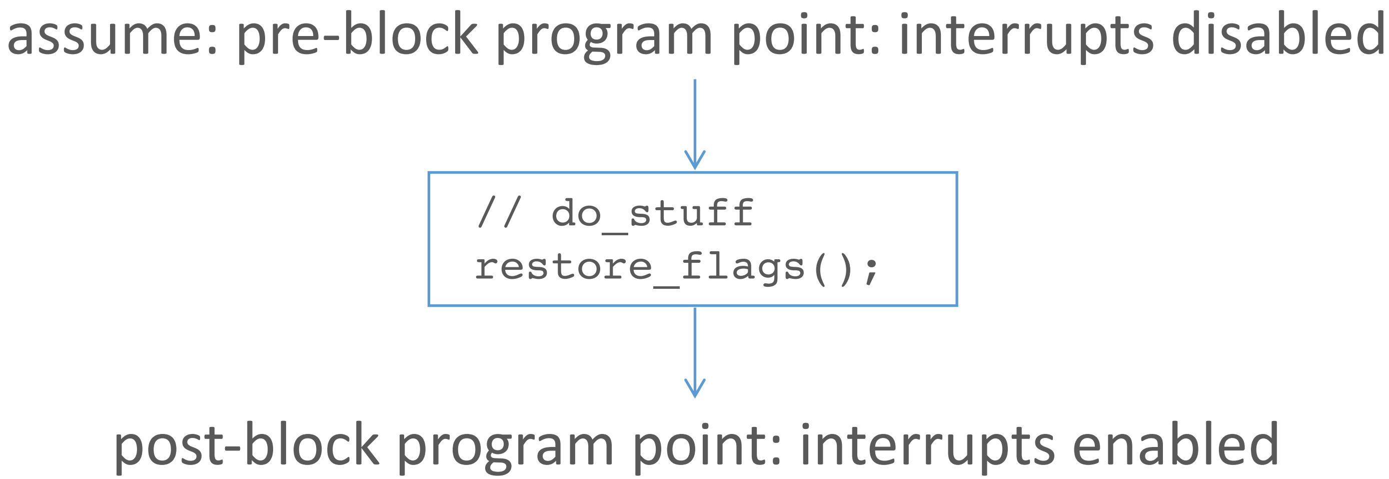
assume: pre-block program point: interrupts enabled



```
cli();
```

post-block program point: interrupts disabled

Transfer function



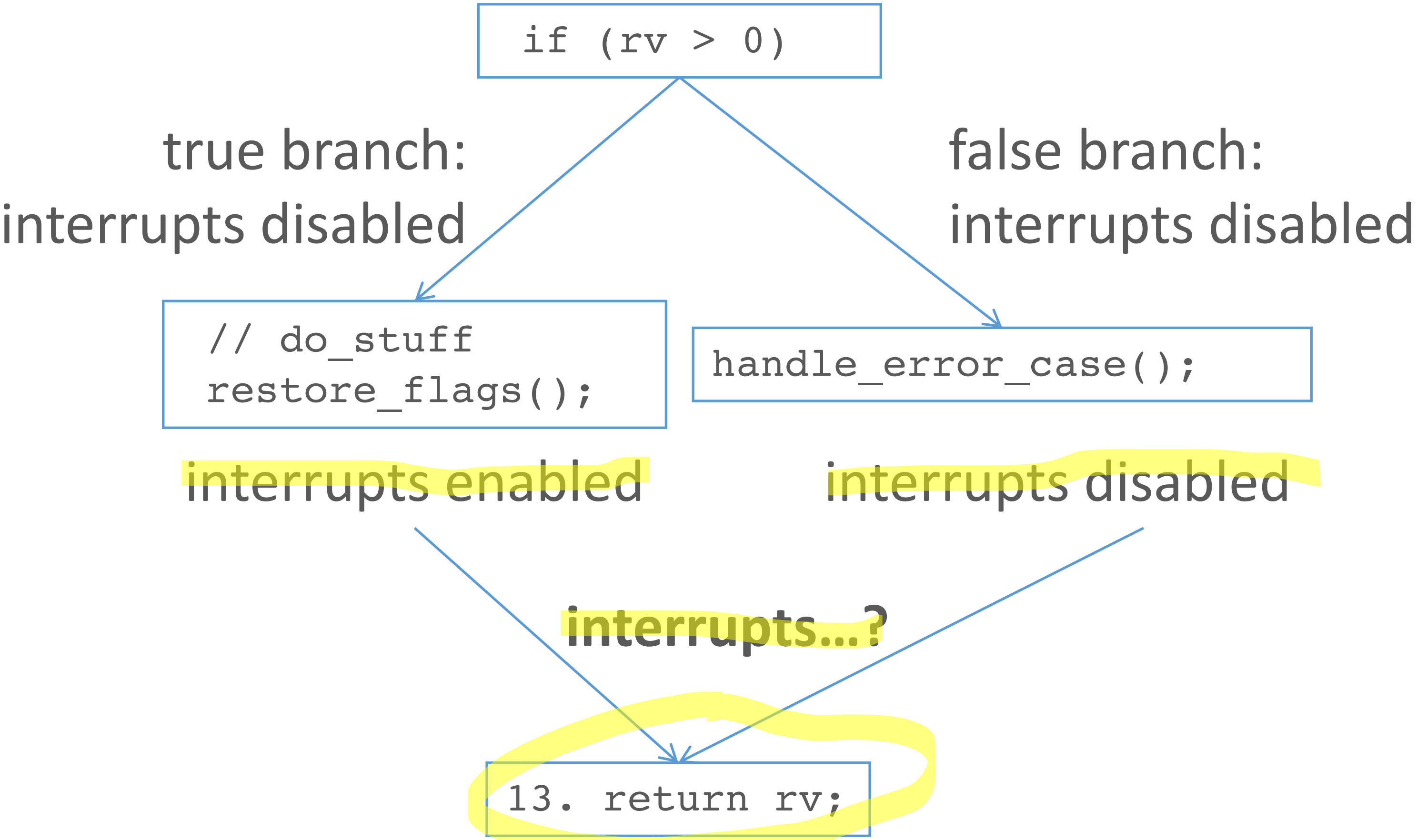
assume: pre-block program point: interrupts disabled

```
// do_stuff
restore_flags();
```

post-block program point: interrupts enabled

Join

assume: pre-block program point: interrupts disabled



```
if (rv > 0)
```

true branch:
interrupts disabled

false branch:
interrupts disabled

```
// do_stuff
restore_flags();
```

```
handle_error_case();
```

interrupts enabled

interrupts disabled

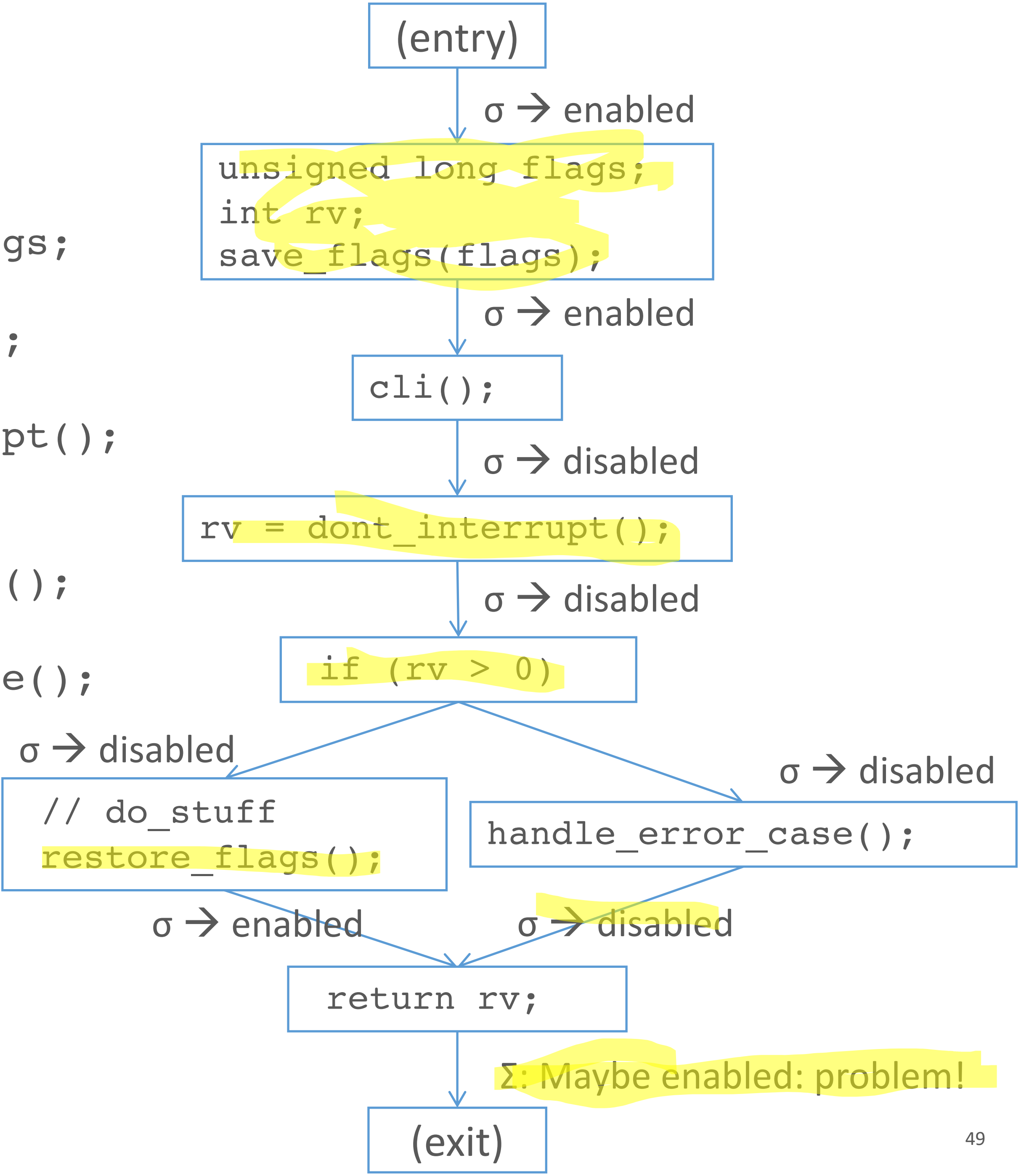
interrupts…?

```
13. return rv;
```

Interrupt analysis: join function

- Abstraction
  - 3 states: enabled, disabled, maybe-enabled
  - Program counter
- **Join:** If at least one predecessor to a basic block has interrupts enabled and at least one has them disabled…
  - Join(enabled, enabled) → enabled
  - Join(disabled, disabled) → disabled
  - Join(disabled, enabled) → maybe-enabled
  - Join(maybe-enabled, *) → maybe-enabled

```
int foo() {
    unsigned long flags;
    int rv;
    save_flags(flags);
    cli();
    rv = dont_interrupt();
    if (rv > 0) {
       // do_stuff
        restore_flags();
    } else {
     handle_error_case();
    }
    return rv;
}
```



(entry)

σ → enabled

```
unsigned long flags;
int rv;
save_flags(flags);
```

σ → enabled

```
cli();
```

σ → disabled

```
rv = dont_interrupt();
```

σ → disabled

```
if (rv > 0)
```

σ → disabled

```
// do_stuff
restore_flags();
```

σ → disabled

```
handle_error_case();
```

σ → enabled

σ → disabled

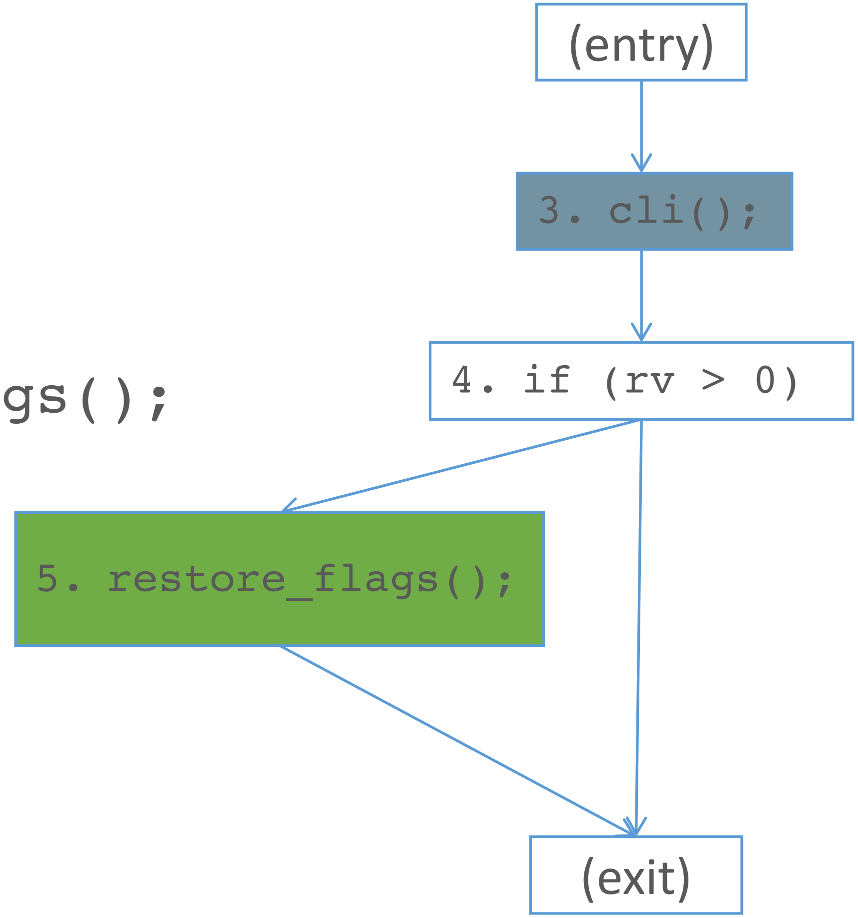
```
return rv;
```

Σ: Maybe enabled: problem!

(exit)

Abstraction

```
1. void foo() {
2.     …
3.     cli();
4.    if (a) {
5.        restore_flags();
6.     }
7. }
```



Too simple?

- Even just tracking a global state like this per function (*control flow analysis)* is useful, e.g.:
  - Dead-code detection in many compilers (e.g. Java)
  - Instrumentation for dynamic analysis before and after decision points; loop detection
  - Actual interrupt analysis in the linux kernel!
- One immediate step up in complexity is to track some state *per variable* (*dataflow analysis*).
- For example: could a variable ever be 0? *(what kinds of errors could this check for?)*
  - Original domain: N maps every variable to an integer. Number of possible concrete states gigantic
    - n 32 bit variables results in 2^{32*n} states
    - With loops, states can change indefinitely
  - Abstract state space is much smaller: a variable is zero, not zero, or maybe-zero: $2^{(n*3)}$