Program Analysis

Adapted from slides by Hilton, Aldrich, and Le Goues

Spot the Bug

```
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;
}
```

Spot the Bug

```
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;
}
```

KEVIN POULSEN SECURITY FEB 22, 2014 11:27 AM

Behind iPhone's Critical Security Bug, a Single Bad 'Goto'

Like everything else on the iPhone, the critical crypto flaw announced in iOS 7 yesterday turns out to be a study in simplicity and elegant design: a single spurious "goto" in one part of Apple's authentication code that accidentally bypasses the rest of it.

ZDNET

tomorrow
belongs to those who embrace it
today

trending tech innovation business security advice buyin

/ tech

Home / Tech / Security

Apple's 'goto fail' tells us nothing good about Cupertino's software delivery process

The fact that Apple's infamous SSL validation bug actually got out into the real world is pretty terrifying.

Written by **Matt Baxter-Reynolds,** Contributor
March 19, 2014 at 3:00 a.m. PT

How Should Apple Have Found the Bug?

- Better code review?
- Better testing?
- Formal verification?
- Today's approach: *analyze the program*'s *source code*

Spot the Bug

```
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;
}
```

This code is unreachable. Isn't that a warning sign?

Hard-To-Find Bugs

- Often on a hard-to-execute codepath (need specific test cases)
- Can't actually test code exhaustively (too many paths, way too many states)
- Instead:
  - Identify relevant properties (e.g. code never dereferences NULL)
  - Try to prove program has those properties

Static Analysis

- Key properties:
  - Liveness: "this good thing eventually happens" (e.g. server generates a response)
  - Safety: "this bad thing never happens" (e.g. dividing by zero)

Example

```python
def n2s(n: int, b: int):
  if n <= 0: return '0'
  r = ''
  while n > 0:
    u = n % b
    if u >= 10:
      u = chr(ord('A') + u-10)
    n = n // b
    r = str(u) + r
return r
```

- What types can 'u' have at each line?
- Can 'u' be negative?
- Will `n2s` always return a value?
- Can there be division by zero?
- Will the returned value ever include a '-'?

Example credit: Hilton et al.

Static Analysis Techniques

- Linters
  - Shallow syntax analysis (unsound, incomplete, unclear properties)
- Type checking (lots of research here)
  - Ensures program has well-defined semantics
- Data flow analysis, abstract interpretation (lots of research here too)
  - Is a[i] always within bounds?
  - Typical answers: "yes", "no", "maybe"

Rice's Theorem (Henry Rice, 1953)

- "Any nontrivial property about the language recognized by a Turing machine is undecidable."
- Implication: interesting static analyses will be imperfect (some false positives, false negatives, or sometimes not terminate)

Proof Sketch (by Contradiction)

- Suppose you have a function, `divides_by_zero`, that determines whether an input program divides by zero.

```
int oops(program p, input i) {
  p(i);
  return 5/0;
}

bool halts(program p, input i) {
  return divides_by_zero(oops(p,i));
}
```

slide adapted from Aldrich and Le Goues

Soundness and Completeness

- A **sound** analysis finds all bugs (in a category of bugs).
  - No false negatives (doesn't fail to find a bug)
- A **complete** analysis only reports bugs (in a category of bugs).
  - No false positives (doesn't report bogus bugs)
- Generally, analyses are either **unsound** or **incomplete** (or both!)

Pattern-Based Bug Detection

- e.g. SpotBugs
- Example: if a method acquires a lock, it should release it on all paths

```
Lock l = ...;
l.lock();
try {
    // do something
    l.unlock();
}
```

Oops! `l` remains locked if an exception is thrown

→

```
Lock l = ...;
l.lock();
try {
    // do something
} finally {
    l.unlock();
}
```

Tradeoffs

- Analysis must be super fast
- In general, these pattern-based detectors are unsound and incomplete
- Google recommends static analyzers have < 10% false positives [Sadowski]
  - Otherwise developers will turn them off!

https://abseil.io/resources/swe-book/html/ch20.html https://checkerframework.org/tutorial/webpages/get-started-cmd.html

Type-Based Approaches

- Idea: Extend the type system to enable reasoning about important properties

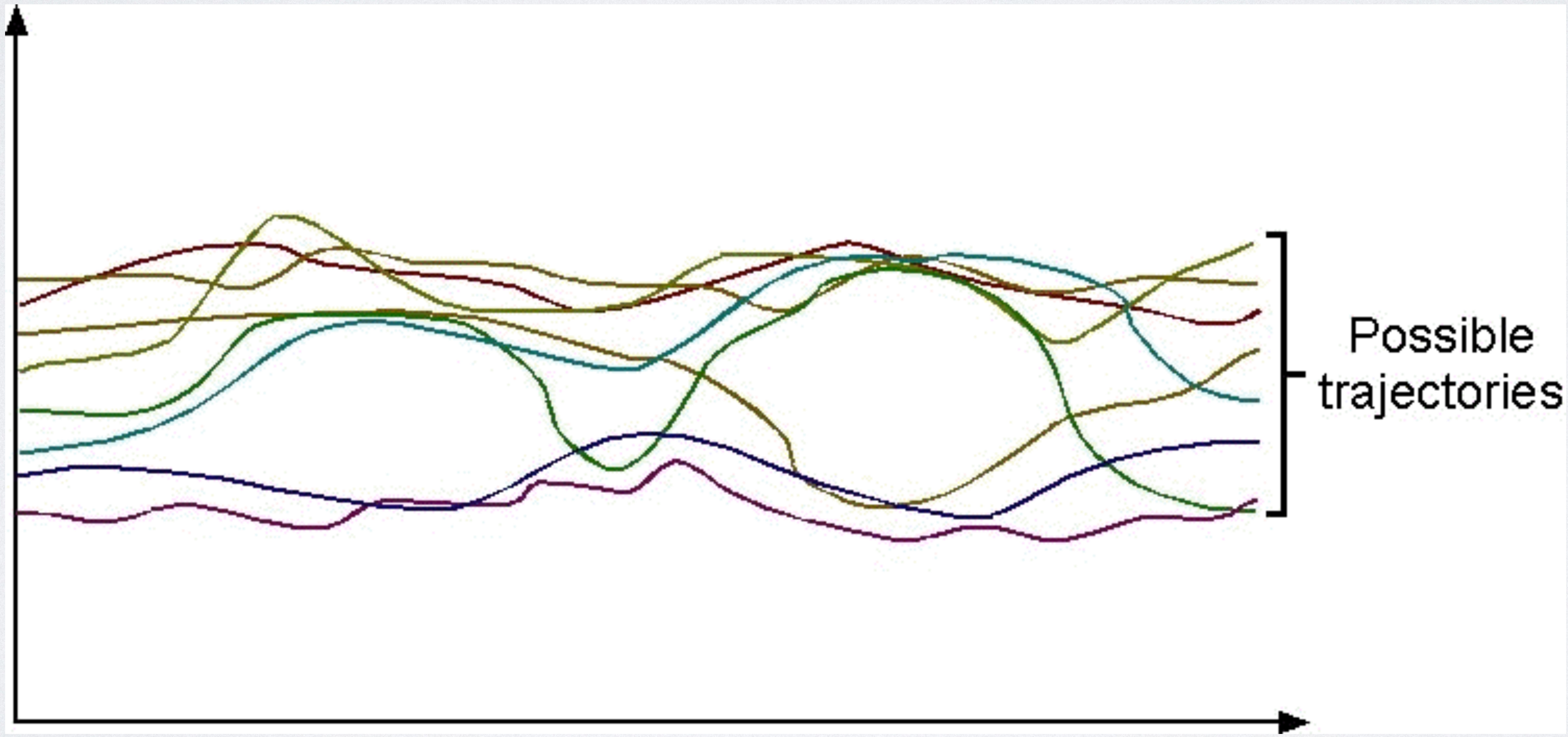
```
public class NullnessExample {
    public static void main(String[] args) {
        Object myObject = null;
        System.out.println(myObject.toString());
    }
}
```

```
$ javacheck -processor org.checkerframework.checker.nullness.NullnessChecker NullnessExample.java
```

```
NullnessExample.java:9: error: [dereference.of.nullable] dereference of possibly-null reference myObject
        System.out.println(myObject.toString());
                           ^
1 error
```

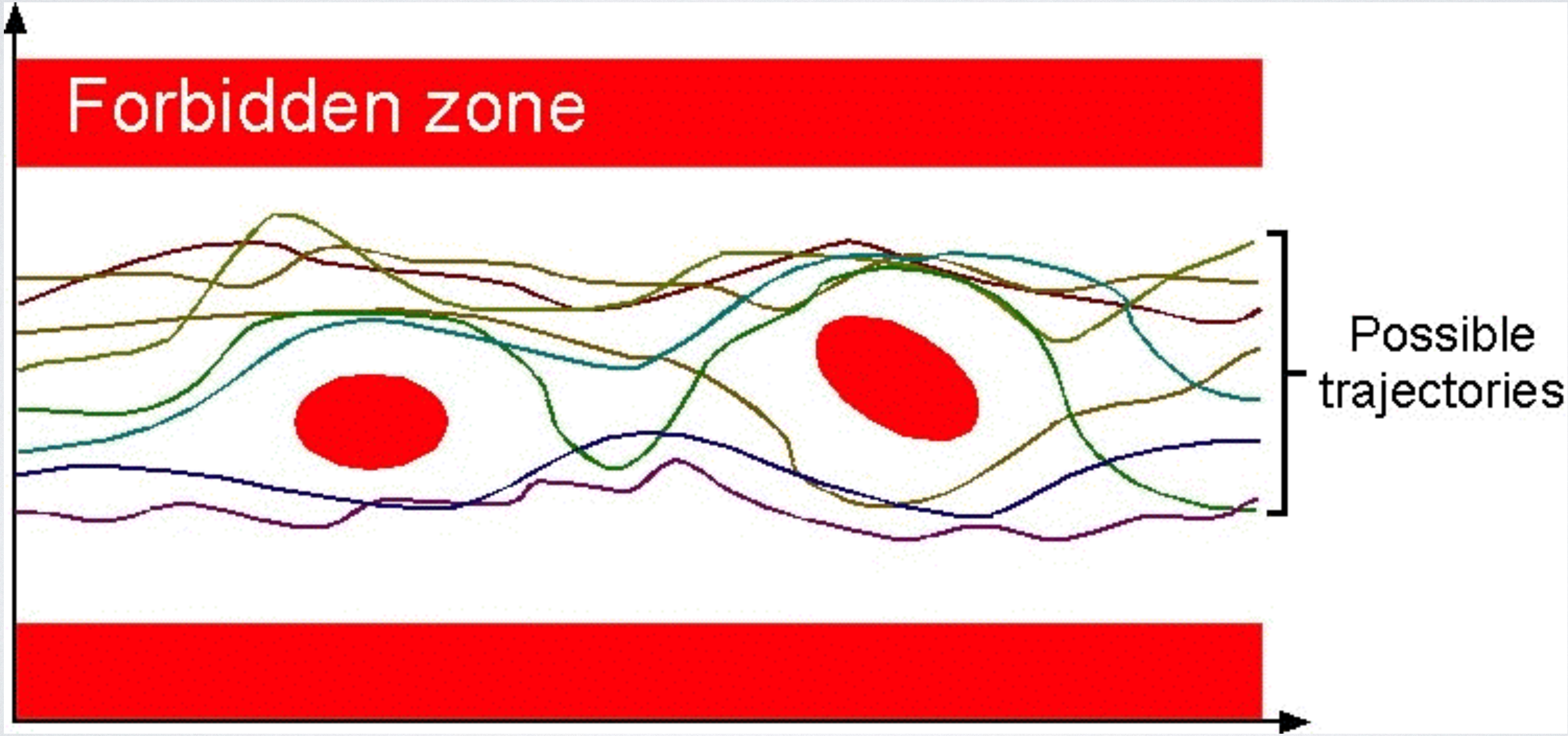
Abstract Interpretation

- Concrete semantics: all possible executions of a program



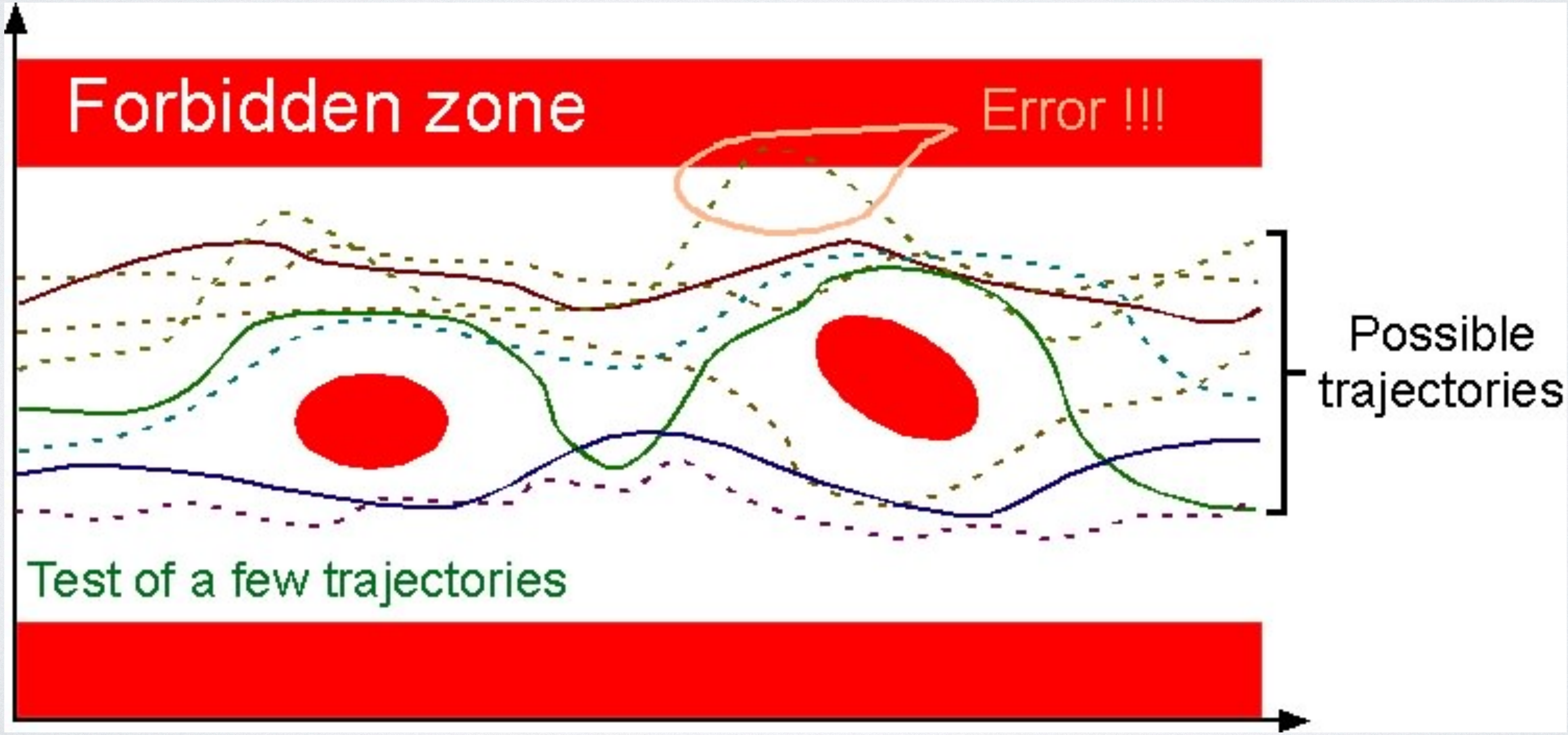
https://www.di.ens.fr/~cousot/AI/IntroAbsInt.html

Safety Properties



Testing

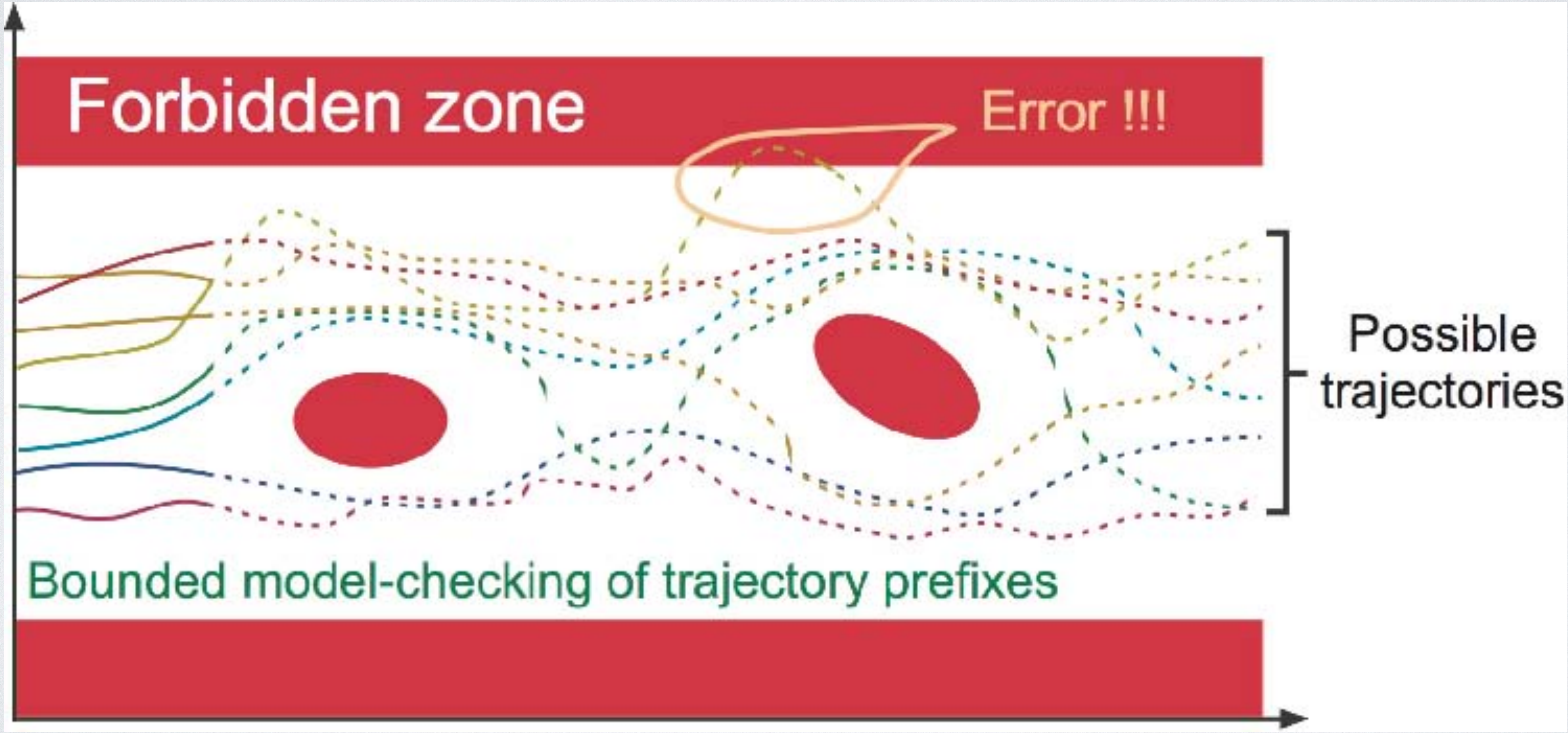
- Can only test some of the possible trajectories



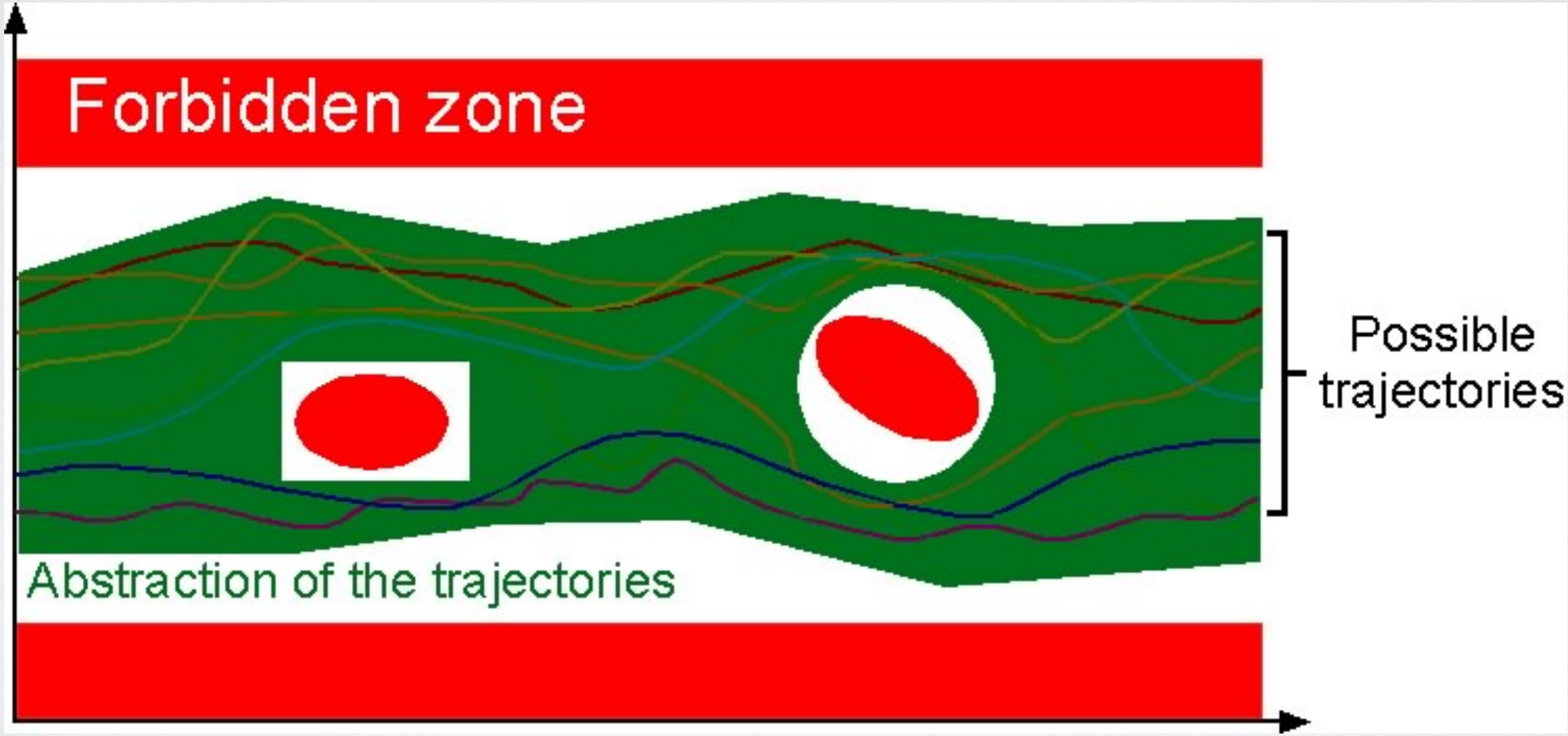
Model Checking

- Goal: Explore all possible execution paths (via logic)
- Problem: too many execution paths (loops, recursion)
- Approach: *bounded* model checking (execute loops at most N times)

Bounded Model Checking



Abstract Interpretation



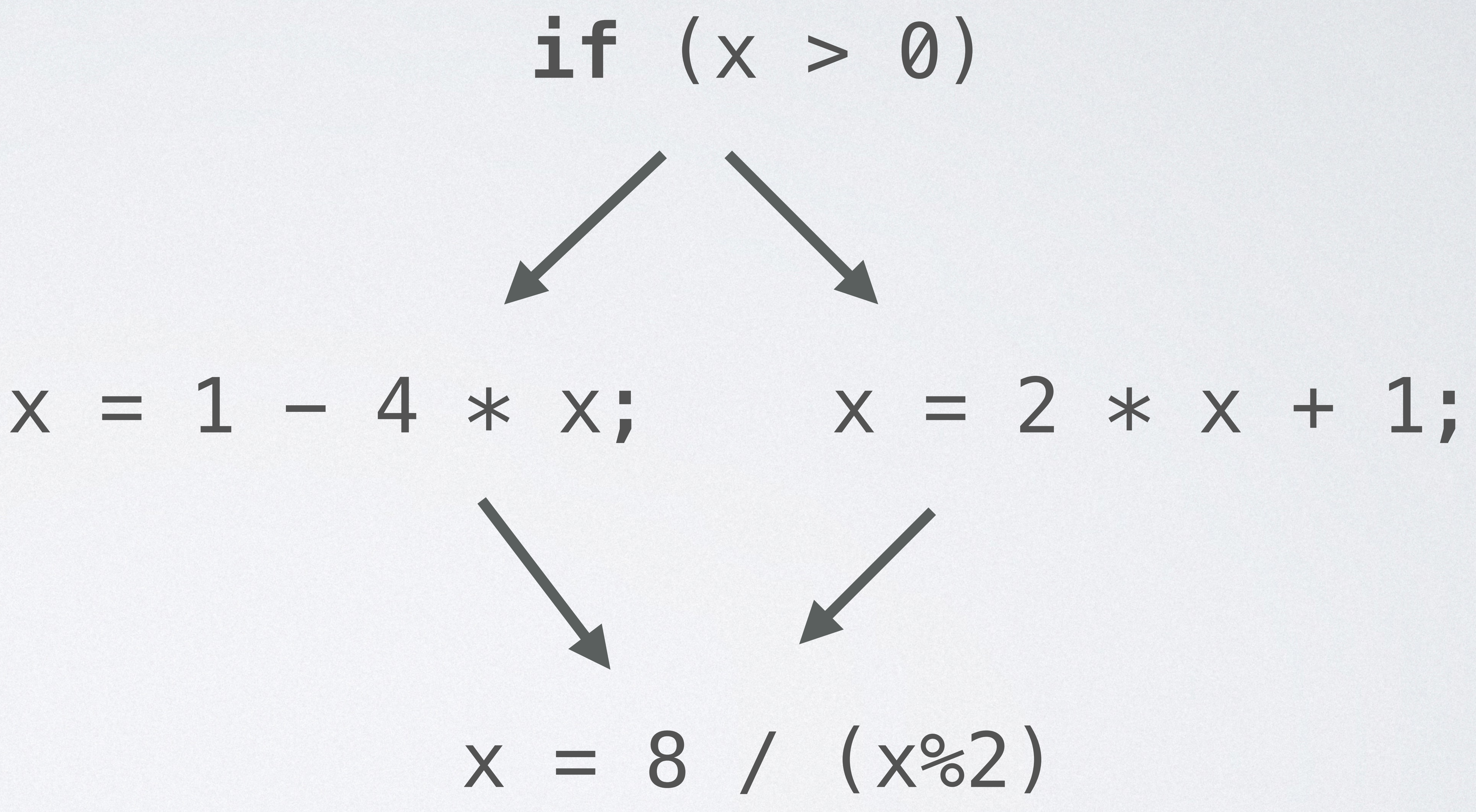
Example: Numerical Intervals

- Ideally: figure out what values variables can have
  - But that requires running the program with all inputs ☹️
- Instead, track bounds [L, H] for each variable

Will This Code Divide by Zero?

```
if (x > 0) {
    x = 2 * x + 1;
}
else {
   x = 1 - 4 * x;
}

x = 8 / (x%2)
```



Defining an Abstract Domain

- We need to know if (x % 2) could be 0
- Let's track whether x could be even or odd.
  - Don't track all the values x could have.
- Abstract domain: {even, odd}

Analysis

{-∞, ∞}; {even, odd}

```
if (x > 0)
```

{-∞, 0}; {even, odd}

{1, ∞}; {even, odd}

```
x = 1 - 4 * x;   x = 2 * x + 1;
```

{1, ∞}; {odd}

{3, ∞}; {odd}

{1, ∞}; {odd}

```
x = 8 / (x%2)
```

Conclusion

- We can find lots of bugs by analyzing code
- But analyses are generally unsound, incomplete, or both
- Software engineers hate false positives, so choose analyses wisely