Concolic Testing

Dynamic Symbolic Execution

Marco Probst

Albert-Ludwigs-Universität Freiburg

January 25th, 2016

Overview

Overview

Program Under Test

- Developers writing code [1] . . .

```
1  f(int x, int y) {
2    if (x*x*x > 0) {
3      if (x > 0 && y == 10) {
4        fail();
5      }
6    } else {
7      if (x > 0 && y == 20) {
8        fail();
9      }
10   }
11
12   complete();
13 }
```

Program Under Test

- Developers writing code [1] . . .

```
1  f(int x, int y) {
2    if (x*x*x > 0) {
3      if (x > 0 && y == 10) {
4        fail();
5      }
6    } else {
7      if (x > 0 && y == 20) {
8        fail();
9      }
10   }
11
12   complete();
13 }
```

- . . . need to test

Overview

Unit Testing

- Ensure overall software quality
- Individual components (e.g. functions)

Unit Testing

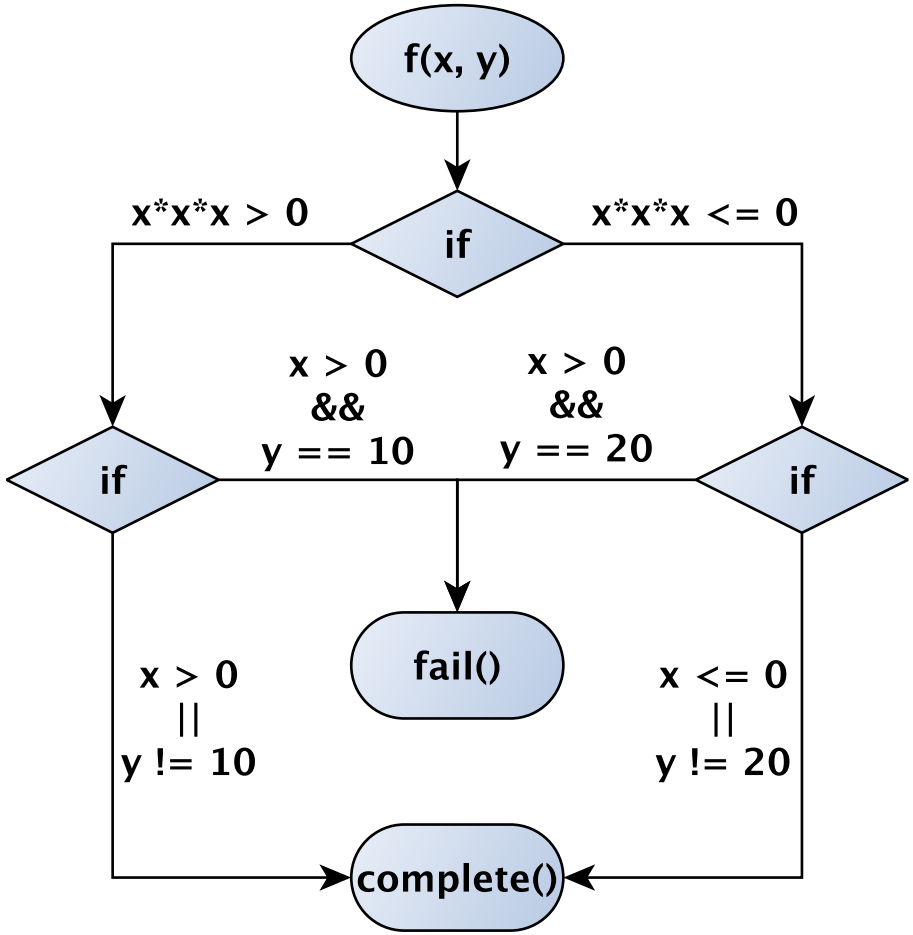
- Ensure overall software quality
- Individual components (e.g. functions)
- Goals
  - Detect errors
  - Check corner cases
  - Provide high code coverage (e.g. path coverage)

Path Coverage

Path Coverage

Code Example $\Rightarrow$ Control Flow $\Rightarrow$ Execution Paths

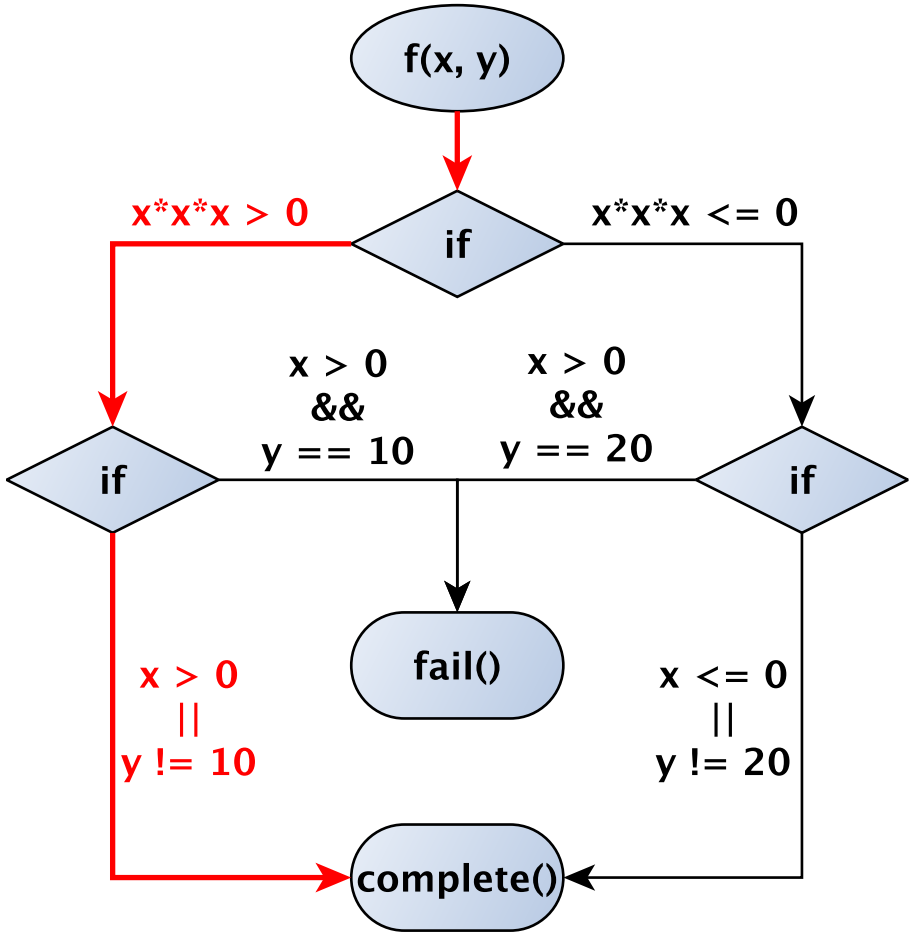
```
1  f(int x, int y) {
2    if (x*x*x > 0) {
3      if (x > 0 && y == 10) {
4        fail();
5      }
6    } else {
7      if (x > 0 && y == 20) {
8        fail();
9      }
10   }
11
12   complete();
13 }
```



Path Coverage

Code Example ⇒ Control Flow ⇒ Execution Paths

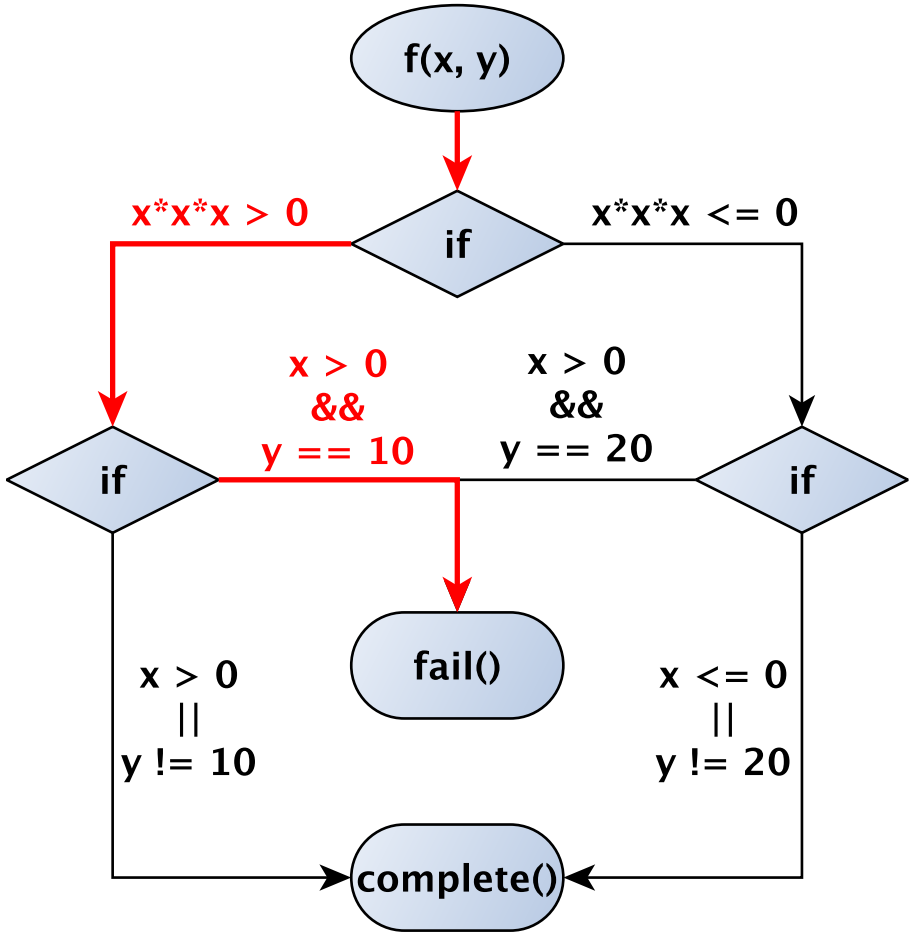
```
1  f(int x, int y) {
2    if (x*x*x > 0) {
3      if (x > 0 && y == 10) {
4        fail();
5      }
6    } else {
7      if (x > 0 && y == 20) {
8        fail();
9      }
10   }
11
12   complete();
13 }
```



Path Coverage

Code Example ⇒ Control Flow ⇒ Execution Paths

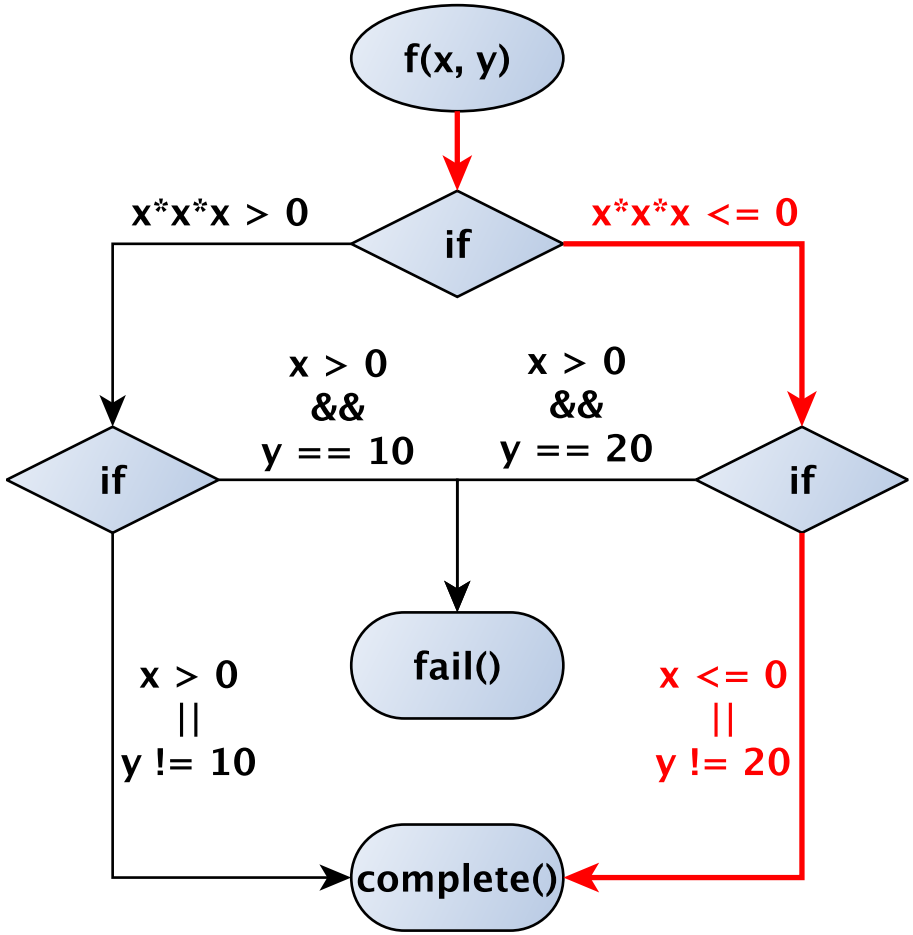
```
1  f(int x, int y) {
2    if (x*x*x > 0) {
3      if (x > 0 && y == 10) {
4        fail();
5      }
6    } else {
7      if (x > 0 && y == 20) {
8        fail();
9      }
10   }
11
12   complete();
13 }
```



Path Coverage

Code Example ⇒ Control Flow ⇒ Execution Paths

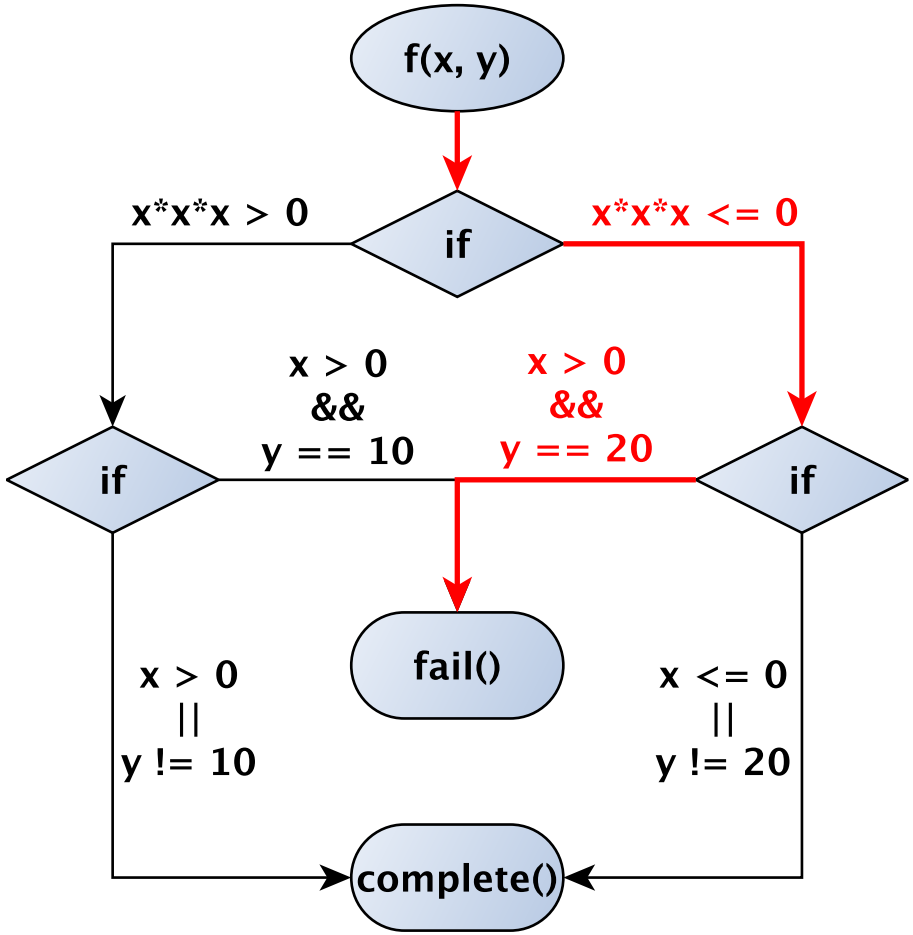
```
1  f(int x, int y) {
2    if (x*x*x > 0) {
3      if (x > 0 && y == 10) {
4        fail();
5      }
6    } else {
7      if (x > 0 && y == 20) {
8        fail();
9      }
10   }
11
12   complete();
13 }
```



Path Coverage

Code Example ⇒ Control Flow ⇒ Execution Paths

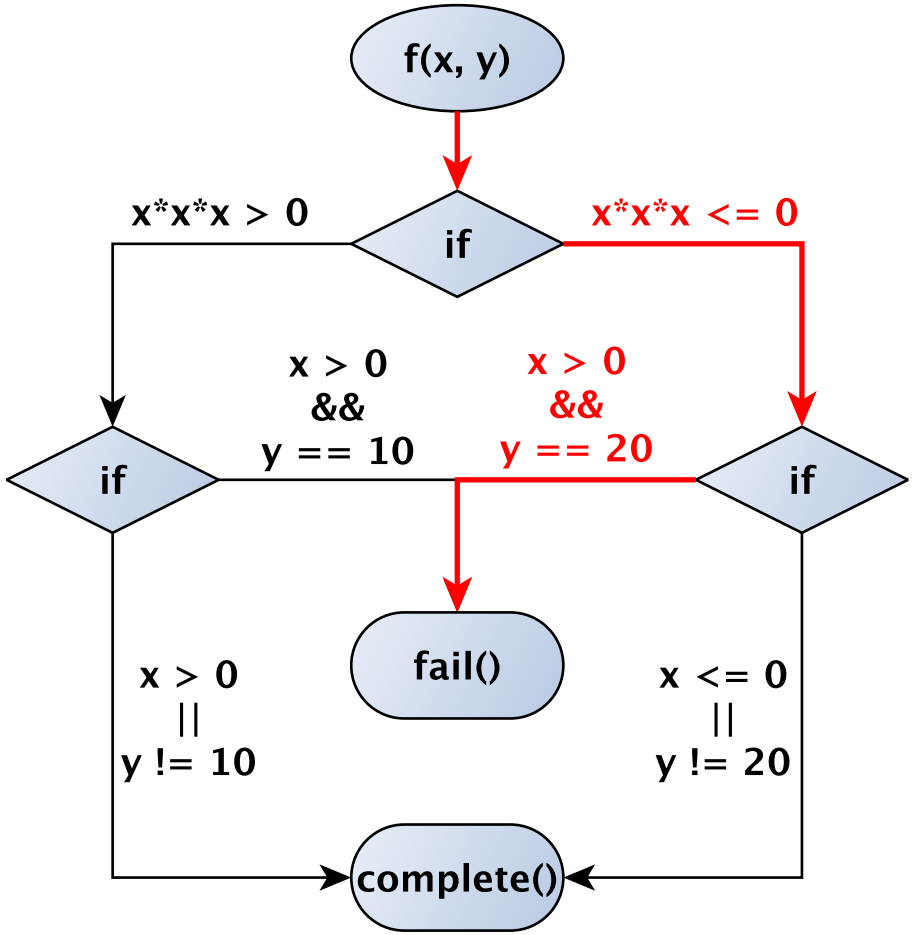
```
1  f(int x, int y) {
2    if (x*x*x > 0) {
3      if (x > 0 && y == 10) {
4        fail();
5      }
6    } else {
7      if (x > 0 && y == 20) {
8        fail();
9      }
10   }
11
12   complete();
13 }
```



Path Coverage

Code Example ⇒ Control Flow ⇒ Execution Paths

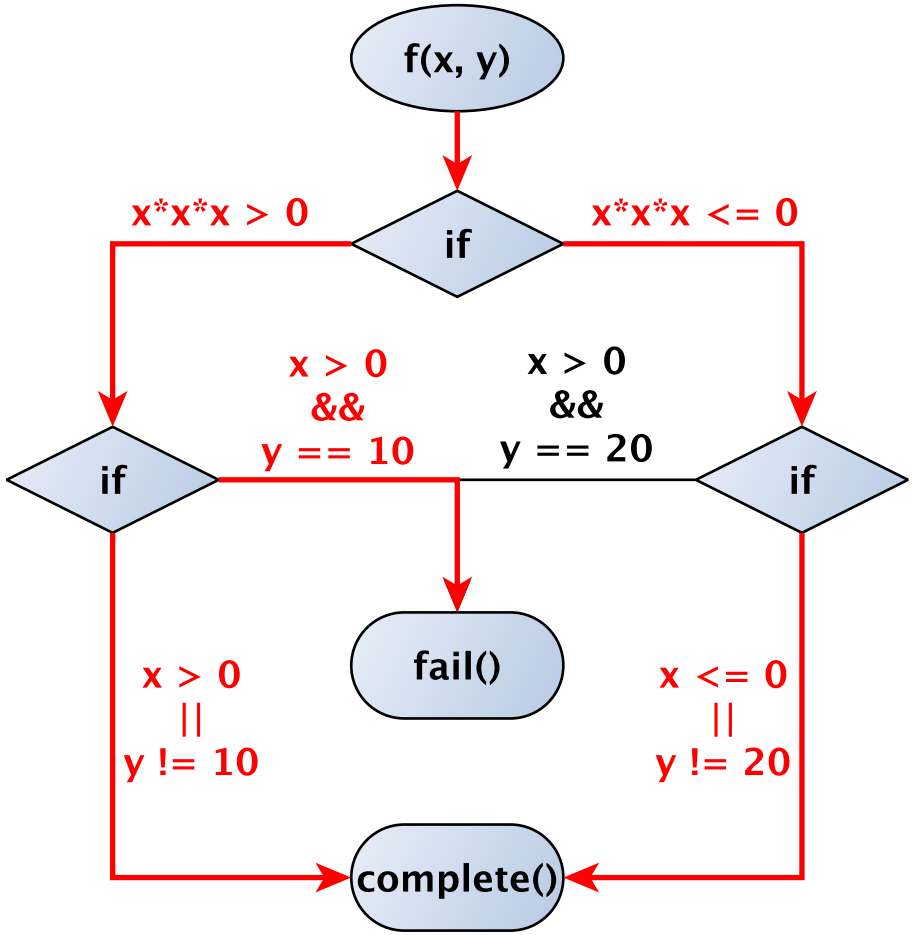
```
1  f(int x, int y) {
2    if (x*x*x > 0) {
3      if (x > 0 && y == 10) {
4        fail();
5      }
6    } else {
7      if (x > 0 && y == 20) {
8        fail();
9      }
10   }
11
12   complete();
13 }
```



Contradiction: `x <= 0 && x > 0` ⇒ not executable

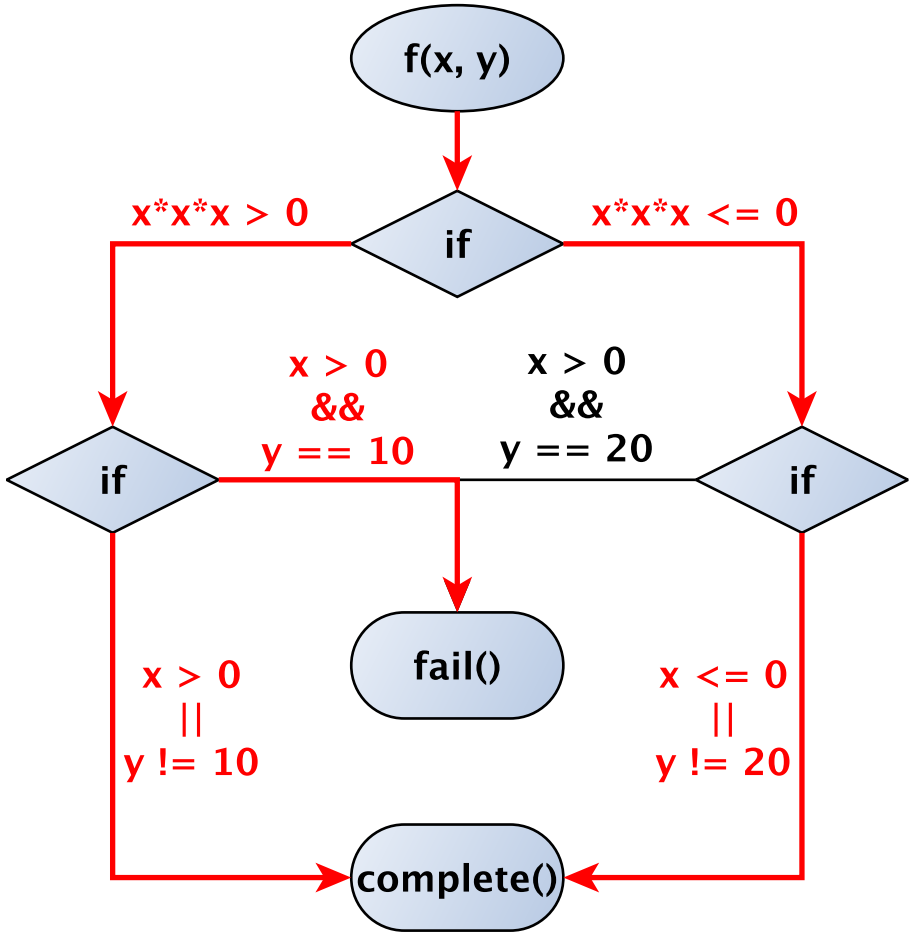
Path Coverage

- 3 possible execution paths
- Corresponding path conditions



Path Coverage

- 3 possible execution paths
- Corresponding path conditions
- Optimal: cover all paths
- Find input set to run program along different paths



Random Testing

Random Testing

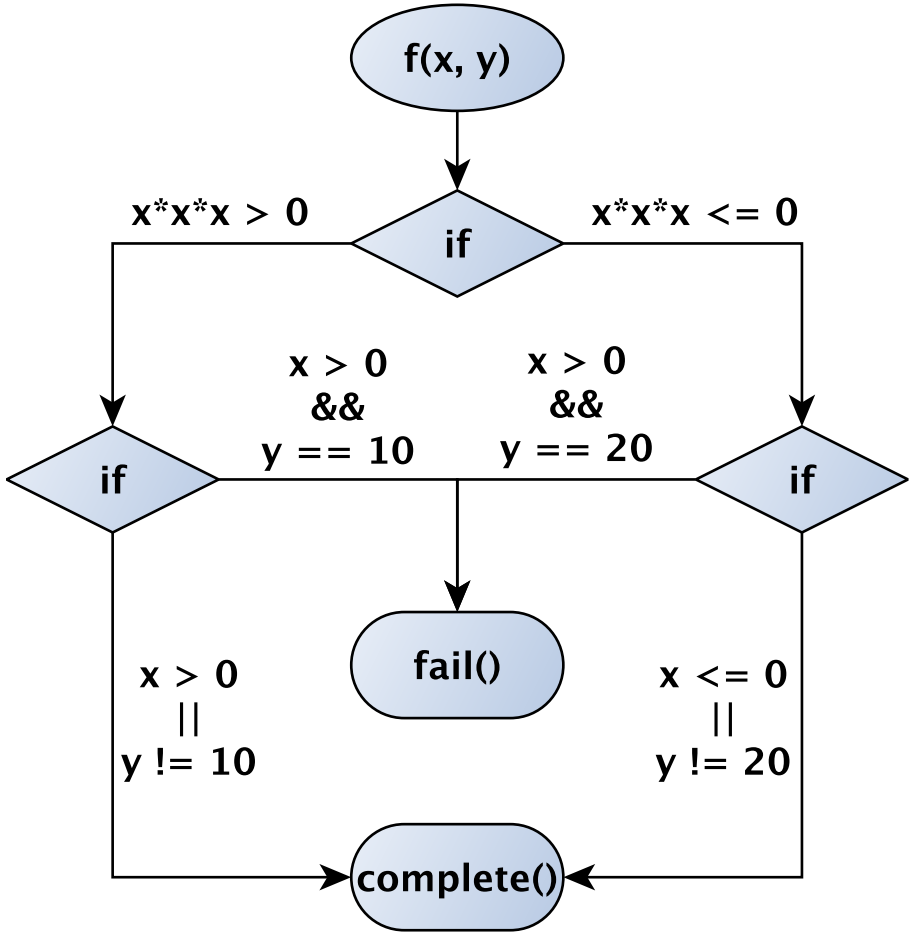
- Most naive way of testing
- Generate random inputs
- Concrete input values
- Dynamic execution of program
- Observe behavior
- Compare against expected behavior
  e.g. output or "do not crash"

Random Testing on Code Example

Random Testing on Code Example

- Random inputs for

```
f(int x, int y)
```

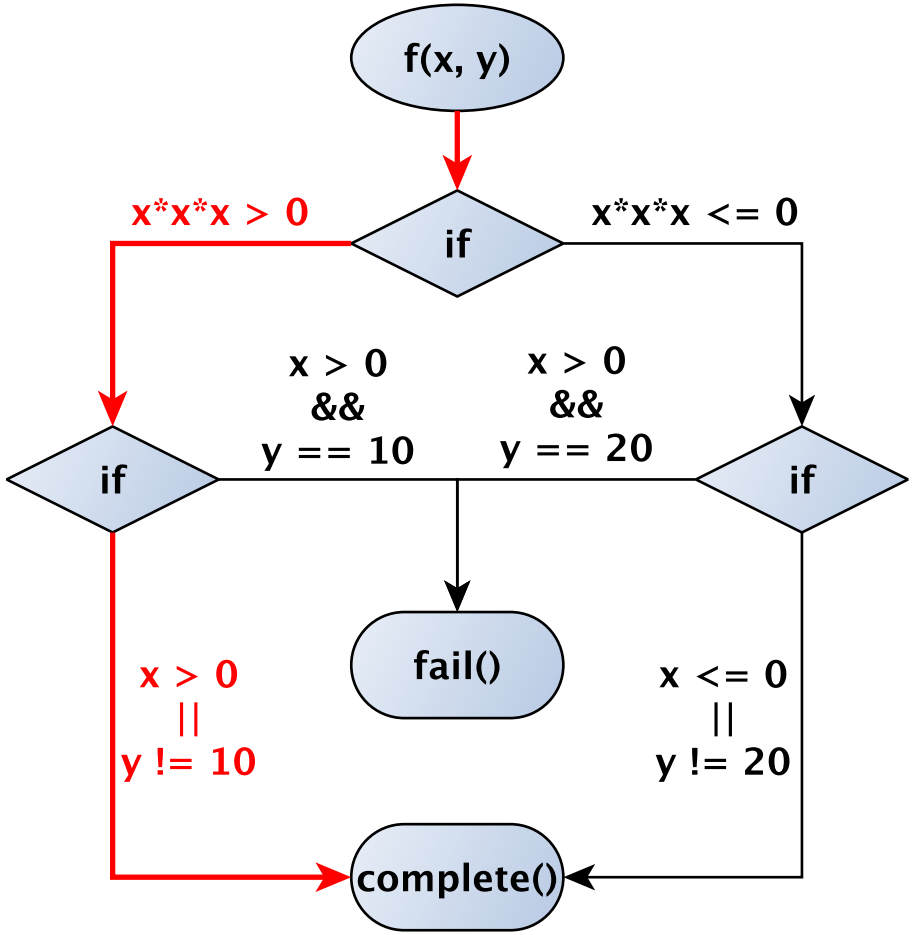


Random Testing on Code Example

- Random inputs for

  ```
  f(int x, int y)
  ```

- `x = 700, y = 500`

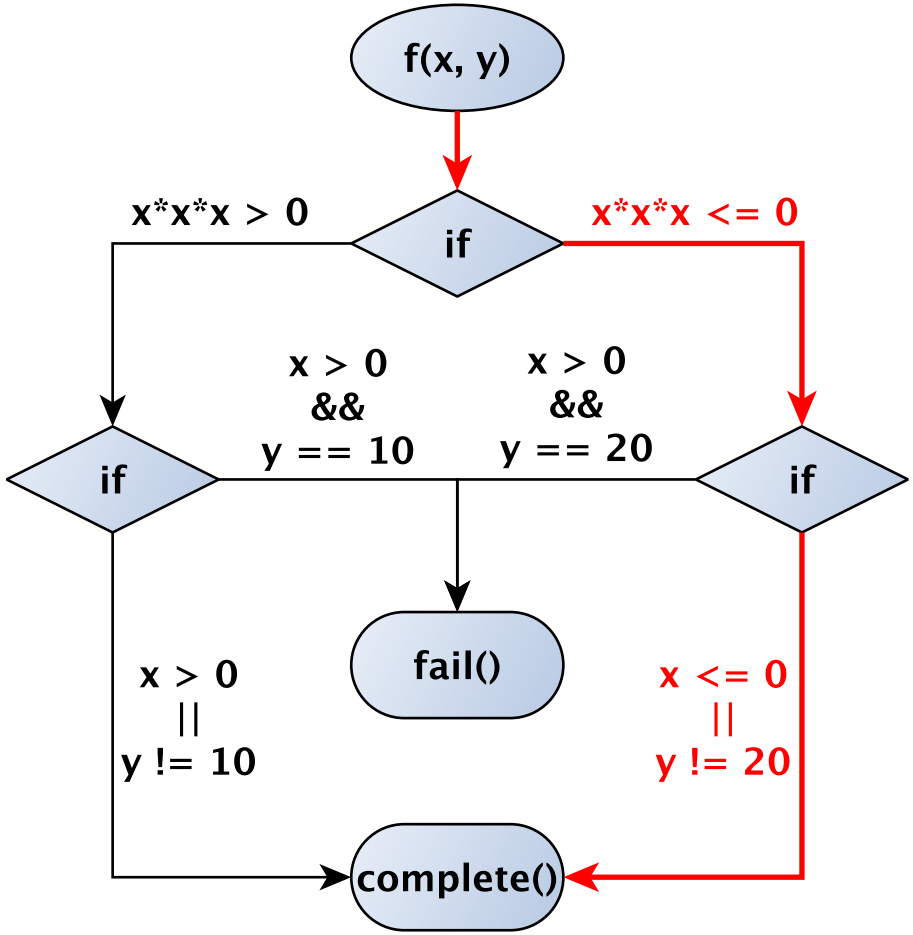


Random Testing on Code Example

- Random inputs for

  ```
  f(int x, int y)
  ```

- `x = 700, y = 500`
- `x = -700, y = 500`

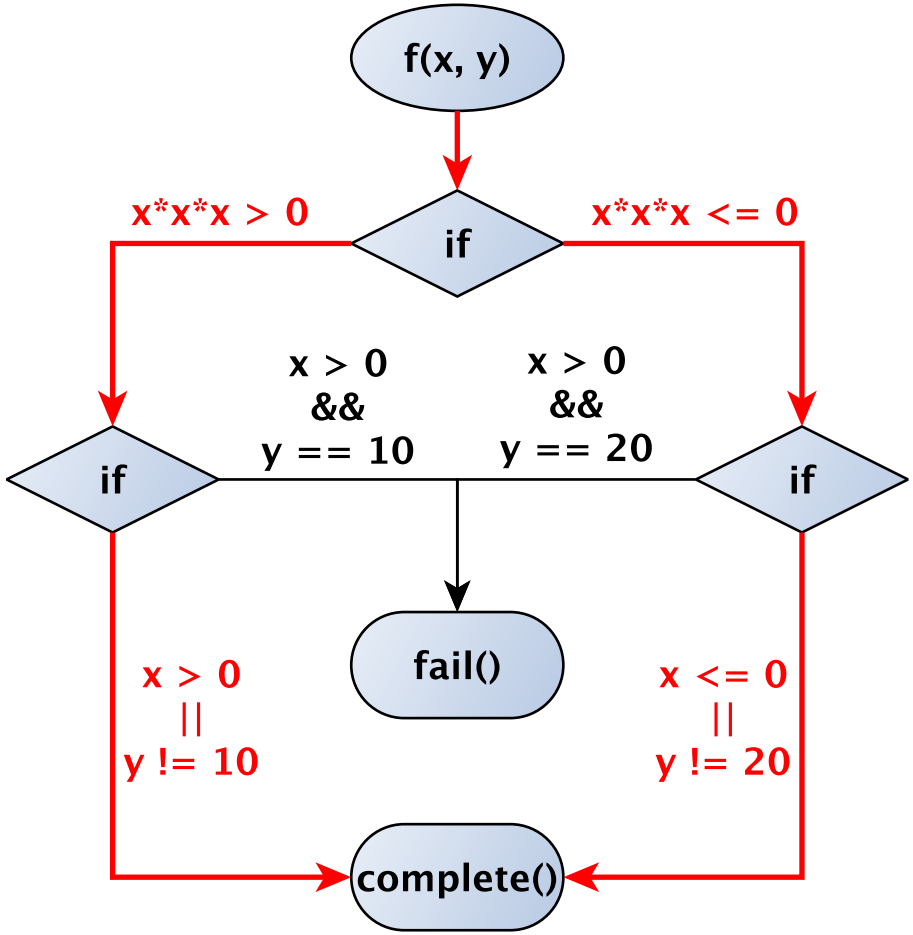


Random Testing on Code Example

- Random inputs for

  ```
  f(int x, int y)
  ```

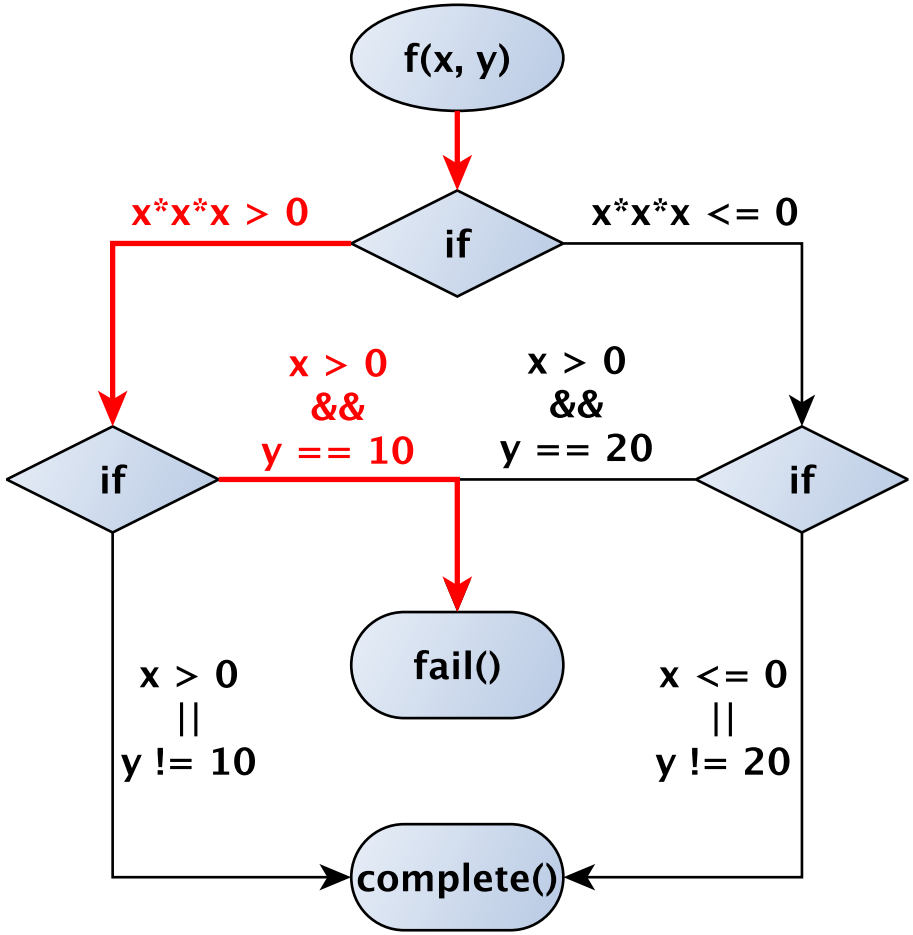
- `x = 700, y = 500`
- `x = -700, y = 500`
- Similar values are very likely



Random Testing on Code Example

- Necessary inputs

  `x > 0, y = 10`



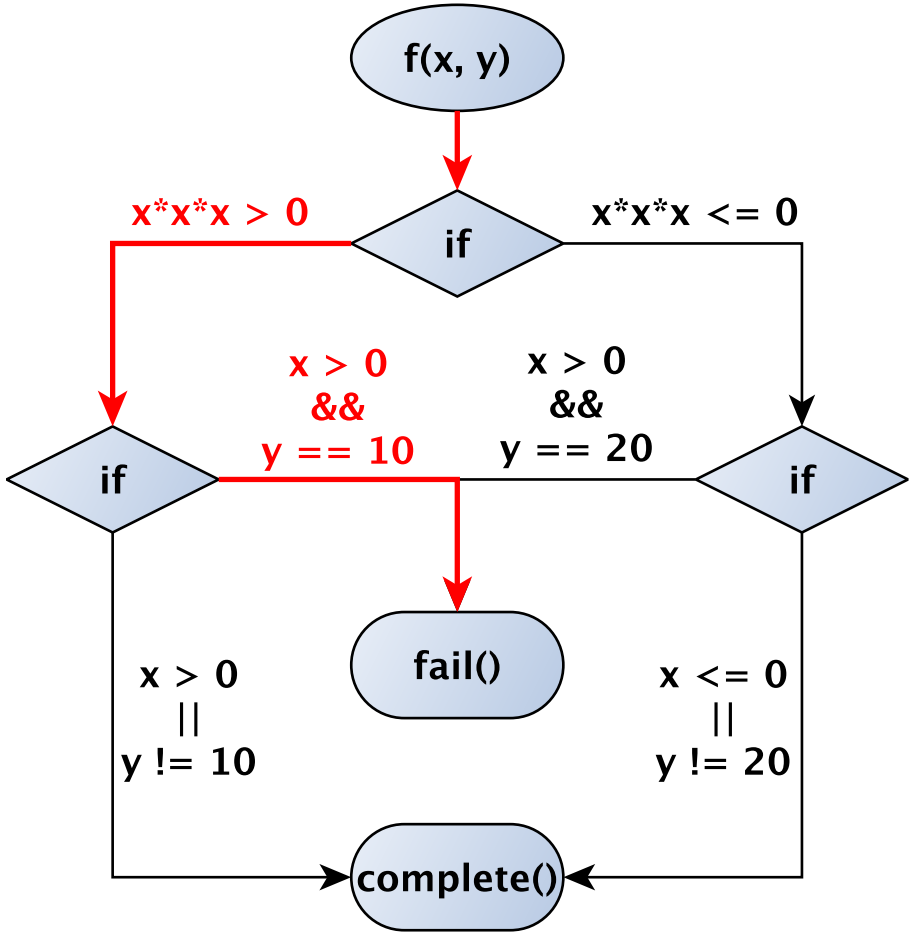
Random Testing on Code Example

- Necessary inputs

  `x > 0, y = 10`

- Assume 32-bit integers

  $\Rightarrow$ 1 out of 2^{32}

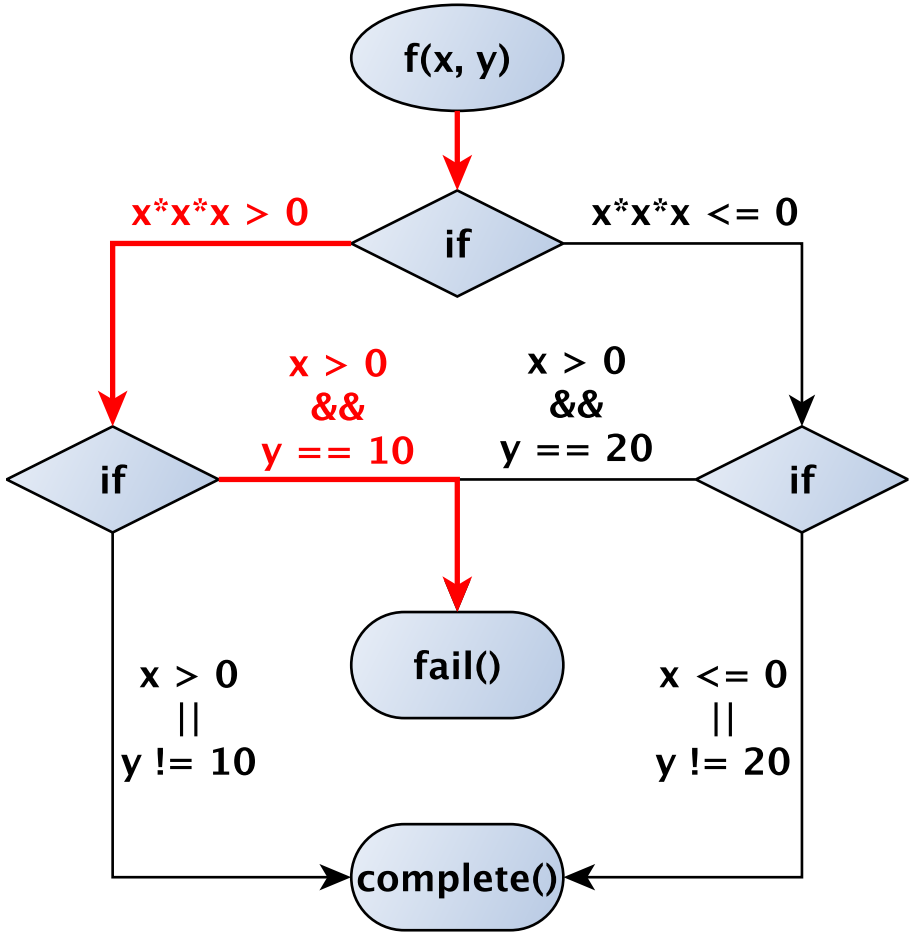


Random Testing on Code Example

- Necessary inputs

  `x > 0, y = 10`

- Assume 32-bit integers
  $\Rightarrow$ 1 out of 2^{32}
- Very low probability

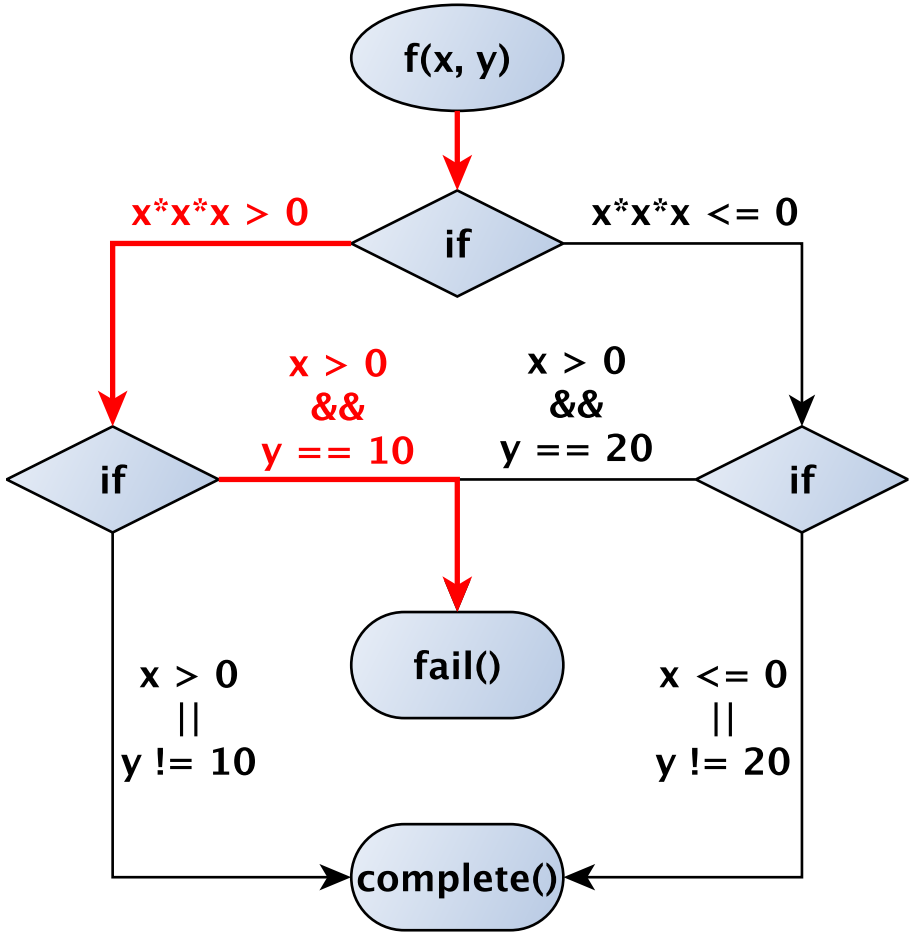


Random Testing on Code Example

- Necessary inputs

  `x > 0, y = 10`
- Assume 32-bit integers

  $\Rightarrow$ 1 out of 2^{32}
- Very low probability
- Long run . . . ☕

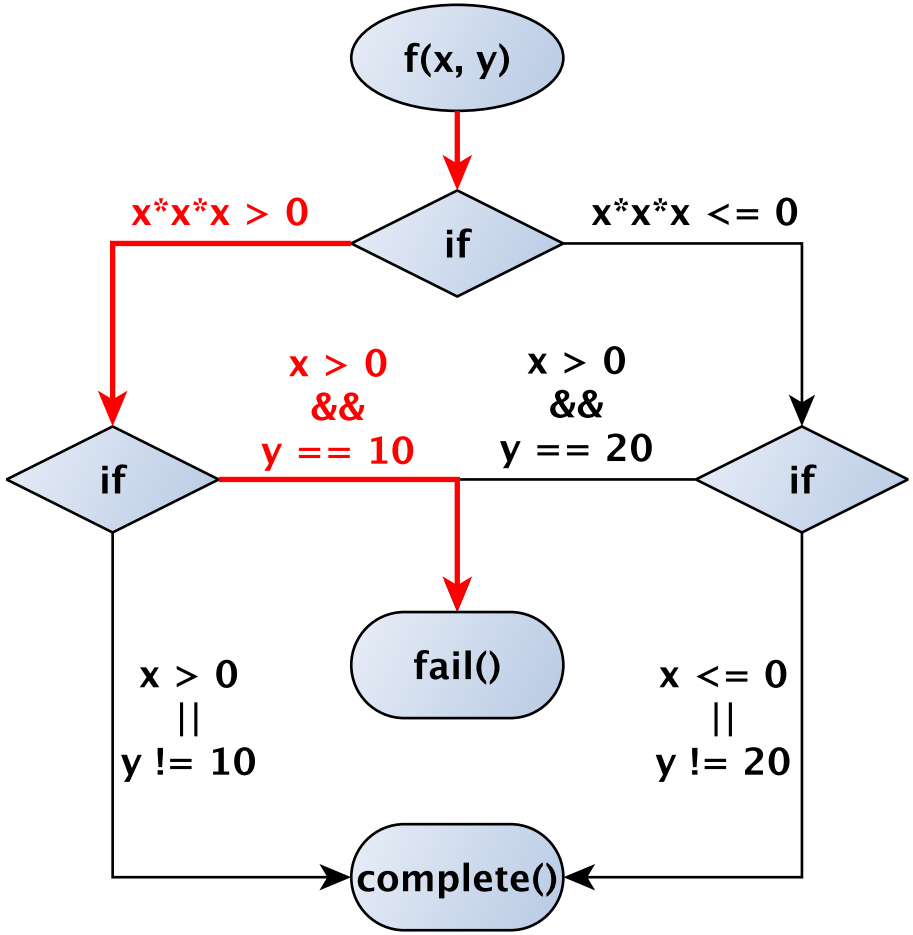


Random Testing on Code Example

- Necessary inputs

  `x > 0, y = 10`
- Assume 32-bit integers

  $\Rightarrow$ 1 out of 2^{32}
- Very low probability
- Long run . . . ☕
- Another technique!



Symbolic Execution [2] & [3]

- Symbols instead of concrete values

Symbolic Execution [2] & [3]

- Symbols instead of concrete values
- Connected to path constraints (or path conditions)

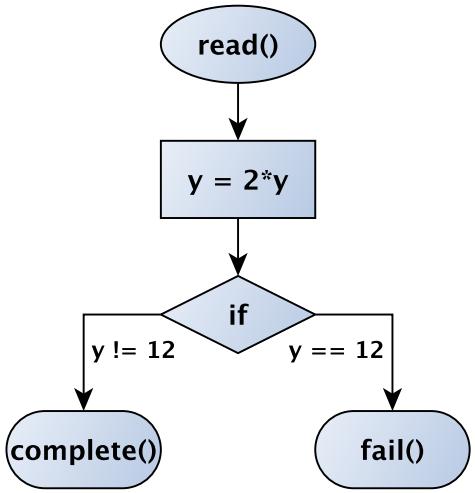
Symbolic Execution [2] & [3]

- Symbols instead of concrete values
- Connected to path constraints (or path conditions)
- Constraint solver computes concrete values

Symbolic Execution [2] & [3]

- Symbols instead of concrete values
- Connected to path constraints (or path conditions)
- Constraint solver computes concrete values

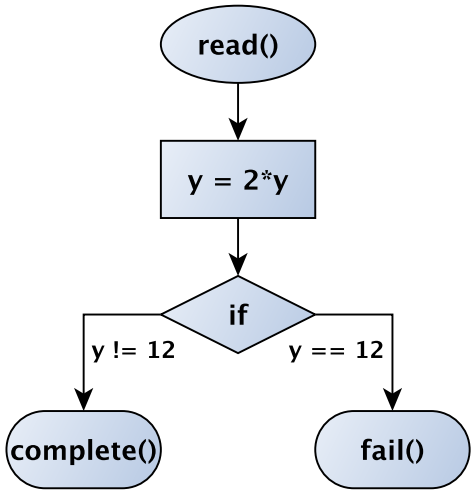
```
1 y = read();
2 y = 2 * y;
3
4 if (y == 12) {
5   fail();
6 }
7
8 complete();
```



Symbolic Execution [2] & [3]

- Symbols instead of concrete values
- Connected to path constraints (or path conditions)
- Constraint solver computes concrete values

```
1 y = read();
2 y = 2 * y;
3
4 if (y == 12) {
5   fail();
6 }
7
8 complete();
```



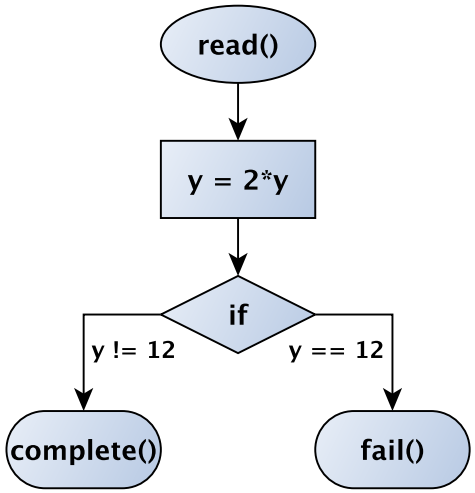
- Introduces symbol `s` for `read()`

  `y = read()` $\Rightarrow$ `y = s`

Symbolic Execution [2] & [3]

- Symbols instead of concrete values
- Connected to path constraints (or path conditions)
- Constraint solver computes concrete values

```
1 y = read();
2 y = 2 * y;
3
4 if (y == 12) {
5   fail();
6 }
7
8 complete();
```



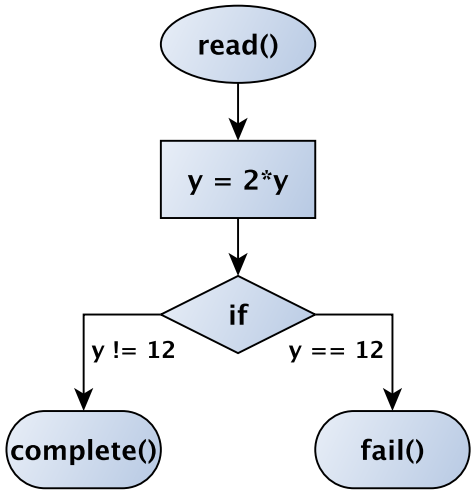
- Introduces symbol `s` for `read()`

  `y = read()` $\Rightarrow$ `y = s`
- `y = 2 * y` $\Rightarrow$ `y = 2 * s`

Symbolic Execution [2] & [3]

- Symbols instead of concrete values
- Connected to path constraints (or path conditions)
- Constraint solver computes concrete values

```
1 y = read();
2 y = 2 * y;
3
4 if (y == 12) {
5   fail();
6 }
7
8 complete();
```



- Introduces symbol `s` for `read()`

  `y = read()` $\Rightarrow$ `y = s`
- `y = 2 * y` $\Rightarrow$ `y = 2 * s`
- Branching point in line 4

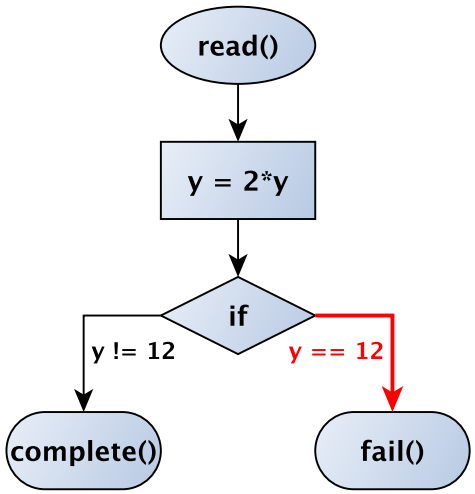
  `y == 12` $\Rightarrow$ `2 * s == 12`

  `y != 12` $\Rightarrow$ `2 * s != 12`

Symbolic Execution [2] & [3]

- Symbols instead of concrete values
- Connected to path constraints (or path conditions)
- Constraint solver computes concrete values

```
1 y = read();
2 y = 2 * y;
3
4 if (y == 12) {
5   fail();
6 }
7
8 complete();
```



- Introduces symbol `s` for `read()`

  `y = read()` $\Rightarrow$ `y = s`
- `y = 2 * y` $\Rightarrow$ `y = 2 * s`
- Branching point in line 4

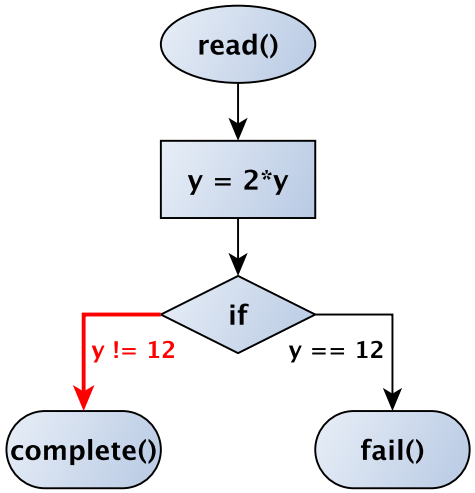
  `y == 12` $\Rightarrow$ `2 * s == 12`

  `y != 12` $\Rightarrow$ `2 * s != 12`

Symbolic Execution [2] & [3]

- Symbols instead of concrete values
- Connected to path constraints (or path conditions)
- Constraint solver computes concrete values

```
1 y = read();
2 y = 2 * y;
3
4 if (y == 12) {
5   fail();
6 }
7
8 complete();
```



- Introduces symbol `s` for `read()`

  $y = \texttt{read}() \Rightarrow y = s$

- $y = 2 * y \Rightarrow y = 2 * s$
- Branching point in line 4

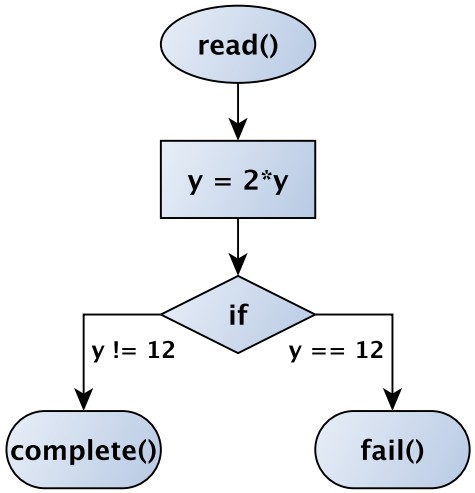
  $y == 12 \Rightarrow 2 * s == 12$

  $y != 12 \Rightarrow 2 * s != 12$

Symbolic Execution [2] & [3]

- Symbols instead of concrete values
- Connected to path constraints (or path conditions)
- Constraint solver computes concrete values

```
1 y = read();
2 y = 2 * y;
3
4 if (y == 12) {
5   fail();
6 }
7
8 complete();
```



- Introduces symbol `s` for `read()`

  `y = read()` $\Rightarrow$ `y = s`
- `y = 2 * y` $\Rightarrow$ `y = 2 * s`
- Branching point in line 4

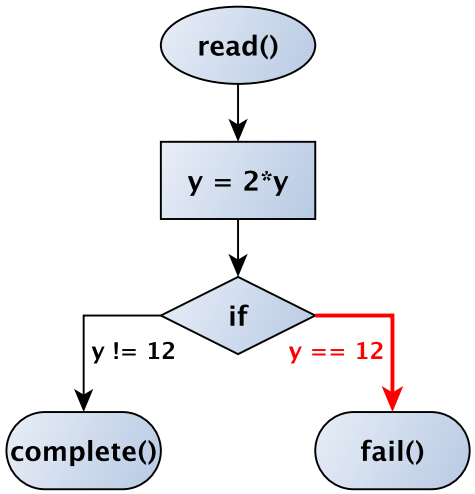
  `y == 12` $\Rightarrow$ `2 * s == 12`

  `y != 12` $\Rightarrow$ `2 * s != 12`
- Which input leads to `fail()`?

Symbolic Execution [2] & [3]

- Symbols instead of concrete values
- Connected to path constraints (or path conditions)
- Constraint solver computes concrete values

```
1 y = read();
2 y = 2 * y;
3
4 if (y == 12) {
5   fail();
6 }
7
8 complete();
```



- Introduces symbol `s` for `read()`

  `y = read()` $\Rightarrow$ `y = s`
- `y = 2 * y` $\Rightarrow$ `y = 2 * s`
- Branching point in line 4

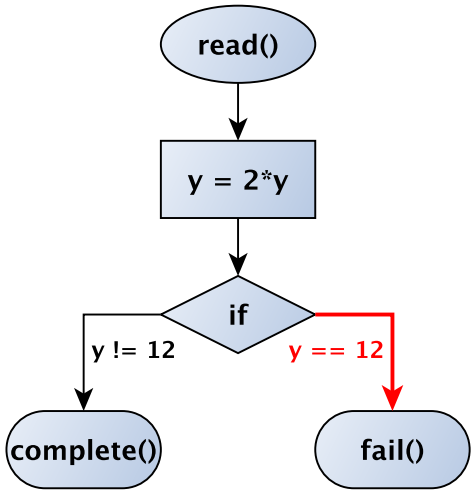
  `y == 12` $\Rightarrow$ `2 * s == 12`

  `y != 12` $\Rightarrow$ `2 * s != 12`
- Which input leads to `fail()`?
- Constraint solver yields `6`

Symbolic Execution [2] & [3]

- Symbols instead of concrete values
- Connected to path constraints (or path conditions)
- Constraint solver computes concrete values

```
1 y = read();
2 y = 2 * y;
3
4 if (y == 12) {
5   fail();
6 }
7
8 complete();
```



- Introduces symbol `s` for `read()`

  `y = read()` $\Rightarrow$ `y = s`
- `y = 2 * y` $\Rightarrow$ `y = 2 * s`
- Branching point in line 4

  `y == 12` $\Rightarrow$ `2 * s == 12`

  `y != 12` $\Rightarrow$ `2 * s != 12`
- Which input leads to `fail()`?
- Constraint solver yields `6`

$\Rightarrow$ Promising for code example!

Symbolic Execution On Code Example

```
1  f(int x, int y) {
2    if (x*x*x > 0) {
3      if (x > 0 && y == 10) {
4        fail();
5      }
6    } else {
7      if (x > 0 && y == 20) {
8        fail();
9      }
10   }
11
12   complete();
13 }
```

Symbolic Execution On Code Example

```
1  f(int x, int y) {
2    if (x*x*x > 0) {
3      if (x > 0 && y == 10) {
4        fail();
5      }
6    } else {
7      if (x > 0 && y == 20) {
8        fail();
9      }
10   }
11
12   complete();
13 }
```

- Non-linear constraint

Symbolic Execution On Code Example

```
1  f(int x, int y) {
2    if (x*x*x > 0) {
3      if (x > 0 && y == 10) {
4        fail();
5      }
6    } else {
7      if (x > 0 && y == 20) {
8        fail();
9      }
10   }
11
12   complete();
13 }
```

- Non-linear constraint
- Undecidable problem
  for most constraint solvers
  $\Rightarrow$ Cannot reason about

Symbolic Execution On Code Example

```
1  f(int x, int y) {
2    if (x*x*x > 0) {
3      if (x > 0 && y == 10) {
4        fail();
5      }
6    } else {
7      if (x > 0 && y == 20) {
8        fail();
9      }
10   }
11
12   complete();
13 }
```

- Non-linear constraint
- Undecidable problem
  for most constraint solvers
  $\Rightarrow$ Cannot reason about
- Execution stops
  $\Rightarrow$ No path covered

Symbolic Execution On Code Example

```
1  f(int x, int y) {
2    if (x*x*x > 0) {
3      if (x > 0 && y == 10) {
4        fail();
5      }
6    } else {
7      if (x > 0 && y == 20) {
8        fail();
9      }
10   }
11
12   complete();
13 }
```

- Non-linear constraint
- Undecidable problem
  for most constraint solvers
  $\Rightarrow$ Cannot reason about
- Execution stops
  $\Rightarrow$ No path covered
- Only possibility:
  Problem with constraint solver

Symbolic Execution On Code Example

```
1  f(int x, int y) {
2    if (x*x*x > 0) {
3      if (x > 0 && y == 10) {
4        fail();
5      }
6    } else {
7      if (x > 0 && y == 20) {
8        fail();
9      }
10   }
11
12   complete();
13 }
```

- Non-linear constraint
- Undecidable problem
  for most constraint solvers
  $\Rightarrow$ Cannot reason about
- Execution stops
  $\Rightarrow$ No path covered
- Only possibility:
  Problem with constraint solver

$\Rightarrow$ How to improve?

Overview

Concolic Testing

- Combination of two techniques
  - Random Testing
    - ★ Concrete values
    - ★ Dynamic execution
  - Symbolic Execution
    - ★ Symbols
    - ★ Static analysis

Concolic Testing

- Combination of two techniques
  - Random Testing
    - ★ Concrete values
    - ★ Dynamic execution
  - Symbolic Execution
    - ★ Symbols
    - ★ Static analysis
- Concolic $\Leftarrow$ Concrete & Symbolic

Concolic Testing

- Combination of two techniques
  - Random Testing
    - ★ Concrete values
    - ★ Dynamic execution
  - Symbolic Execution
    - ★ Symbols
    - ★ Static analysis
- Concolic $\Leftarrow$ Concrete & Symbolic
- Symbolic Execution beside Random Testing
  $\Rightarrow$ Execute dynamically & explore symbolically

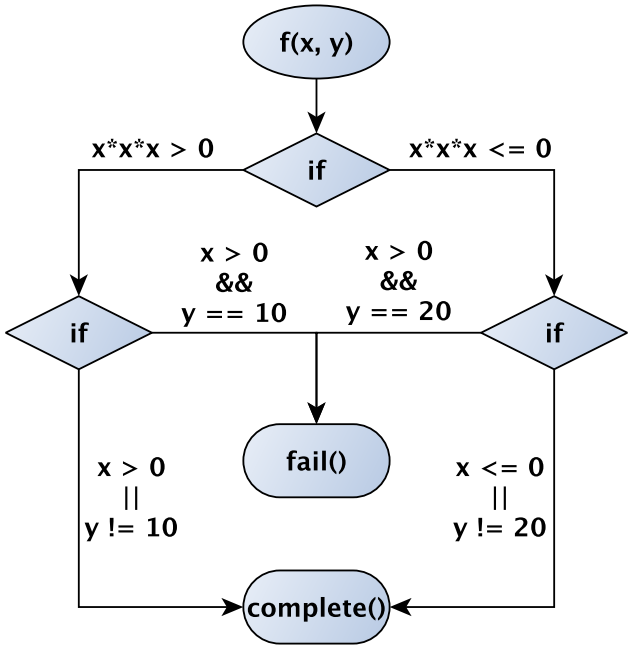
Concolic Testing

- Combination of two techniques
  - Random Testing
    - ★ Concrete values
    - ★ Dynamic execution
  - Symbolic Execution
    - ★ Symbols
    - ★ Static analysis
- Concolic $\Leftarrow$ Concrete & Symbolic
- Symbolic Execution beside Random Testing
  $\Rightarrow$ Execute dynamically & explore symbolically
- Also: Dynamic Symbolic Execution

DART in Action (1)

Symbolic Execution

Dynamic Execution



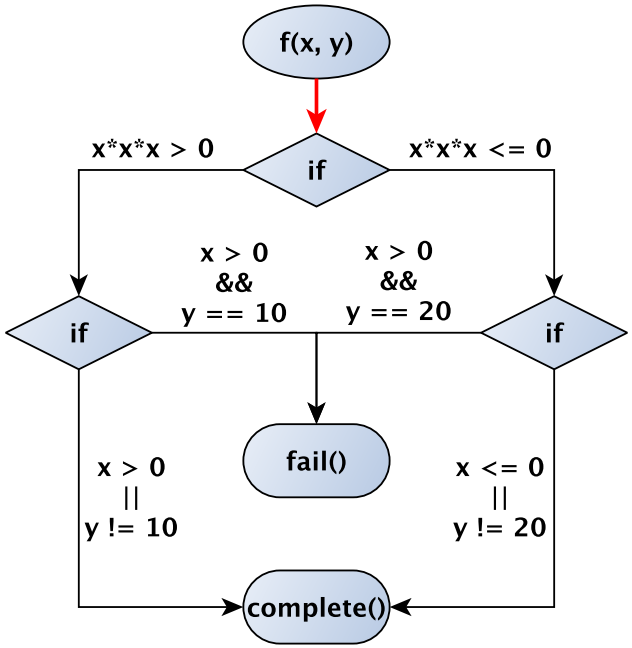
DART in Action (1)

Symbolic Execution

Dynamic Execution

- Random Testing
- Random inputs

  `x = 700, y = 500`

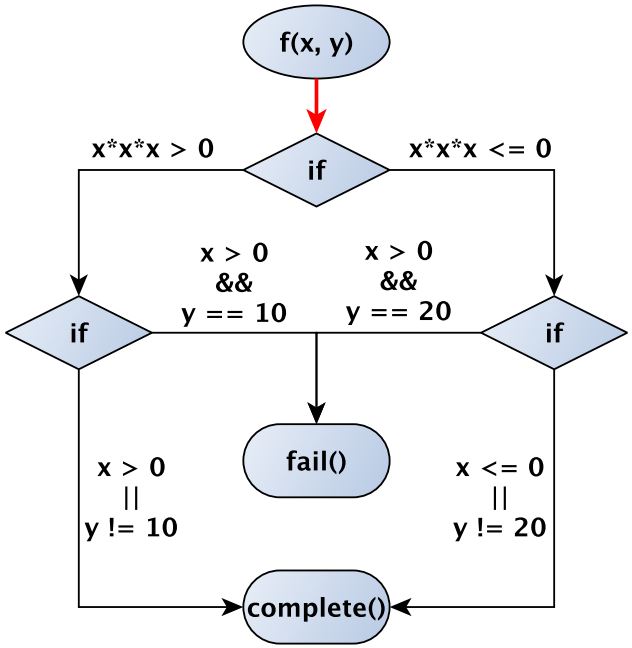


DART in Action (1)

Dynamic Execution

- Random Testing
- Random inputs

```
x = 700, y = 500
```



Symbolic Execution

- Introduce symbols

```
x1 = X, y1 = Y
```

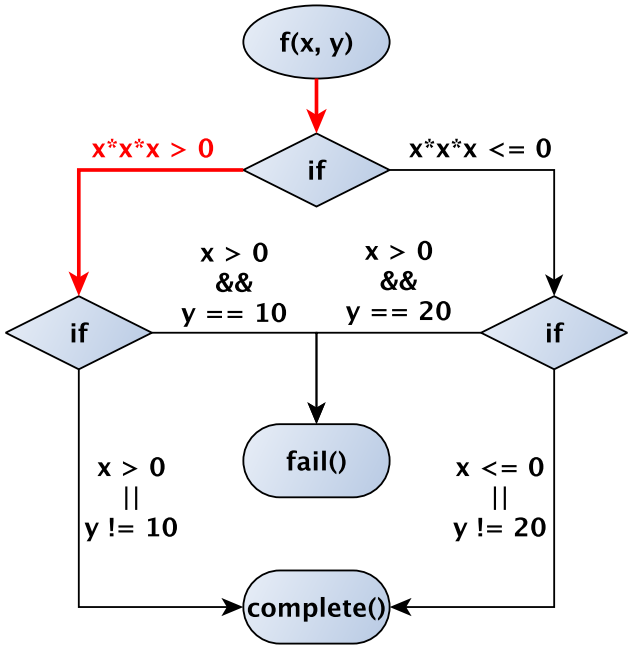
DART in Action (1)

Dynamic Execution

- Random Testing
- Random inputs

  `x = 700, y = 500`

- `x*x*x > 0`



Symbolic Execution

- Introduce symbols

  `x1 = X, y1 = Y`

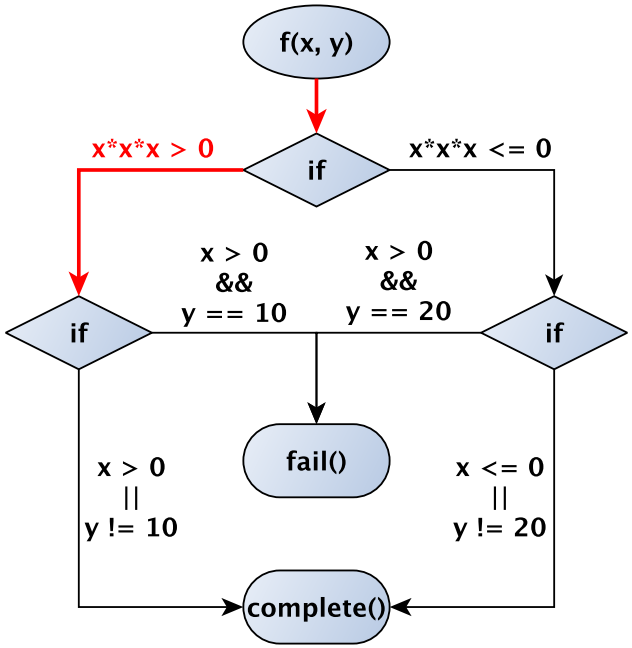
DART in Action (1)

Dynamic Execution

- Random Testing
- Random inputs

  `x = 700, y = 500`

- `x*x*x > 0`



Symbolic Execution

- Introduce symbols

  `x1 = X, y1 = Y`

- Constrain `x`

  `X*X*X <= 0`

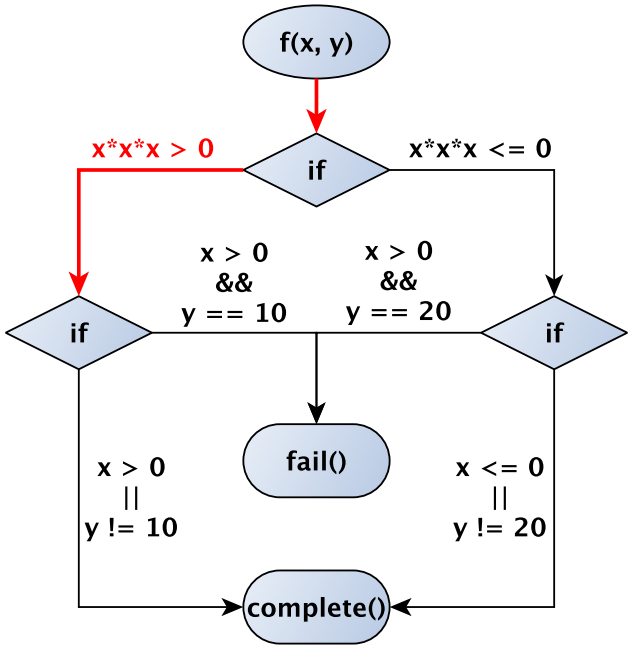
DART in Action (1)

Dynamic Execution

- Random Testing
- Random inputs

  `x = 700, y = 500`

- `x*x*x > 0`



Symbolic Execution

- Introduce symbols

  `x1 = X, y1 = Y`

- Constrain `X`

  `X*X*X <= 0`

- Solve constraint

  Non-Linear $\Rightarrow$ Fail

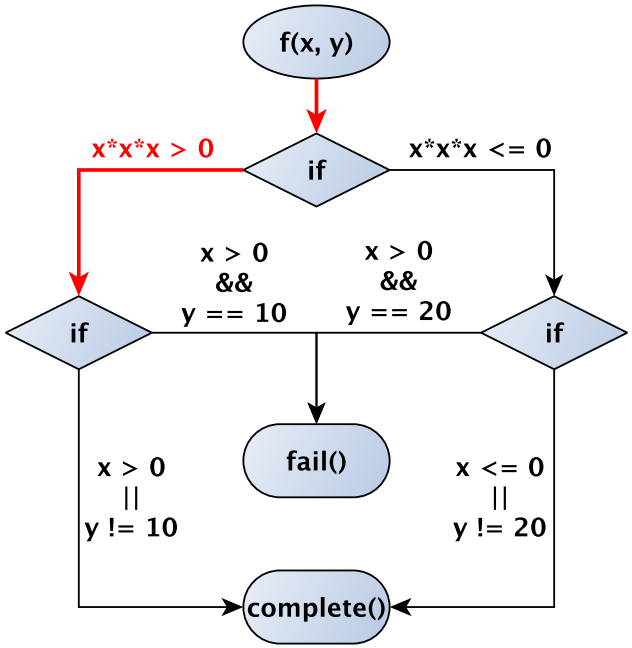
DART in Action (1)

Dynamic Execution

- Random Testing
- Random inputs

  `x = 700, y = 500`

- `x*x*x > 0`



Symbolic Execution

- Introduce symbols

  `x1 = X, y1 = Y`

- Constrain `X`

  `X*X*X <= 0`

- Solve constraint
  Non-Linear $\Rightarrow$ Fail
- Concrete fallback

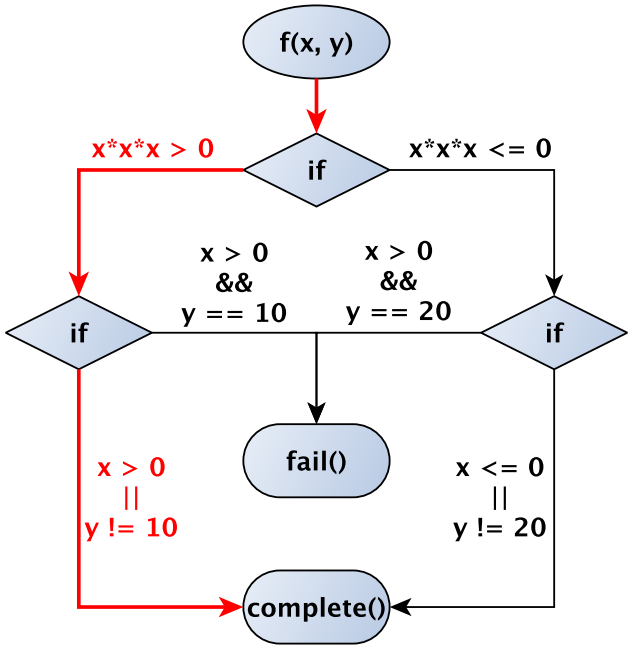
  `x1 = 700`

DART in Action (1)

Dynamic Execution

- Random Testing
- Random inputs

  `x = 700, y = 500`
- `x*x*x > 0`
- `y != 10`



Symbolic Execution

- Introduce symbols

  `x1 = X, y1 = Y`
- Constrain `x`

  `X*X*X <= 0`
- Solve constraint
  Non-Linear $\Rightarrow$ Fail
- Concrete fallback

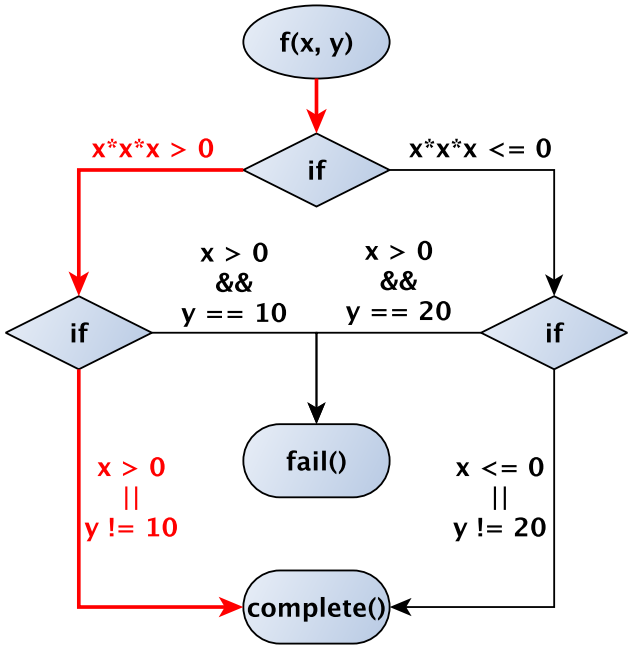
  `x1 = 700`

DART in Action (1)

Dynamic Execution

- Random Testing
- Random inputs

  `x = 700, y = 500`
- `x*x*x > 0`
- `y != 10`



Symbolic Execution

- Introduce symbols

  `x1 = X, y1 = Y`
- Constrain `X`

  `X*X*X <= 0`
- Solve constraint
  Non-Linear $\Rightarrow$ Fail
- Concrete fallback

  `x1 = 700`
- Constrain `Y`

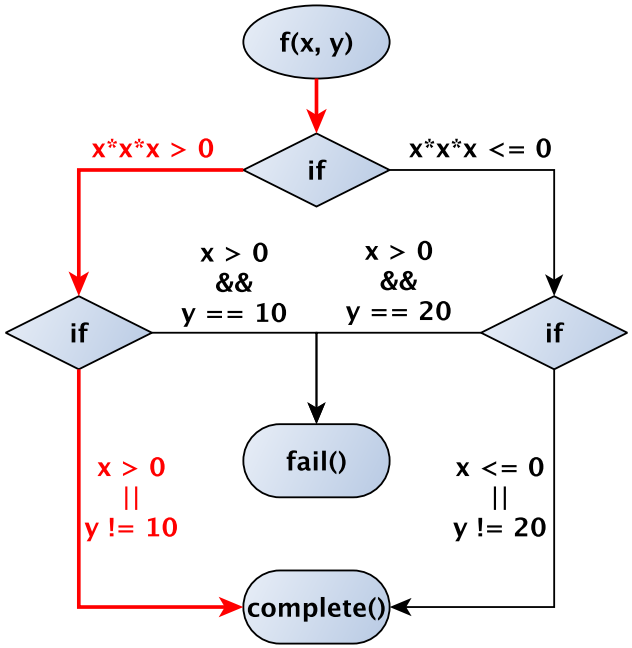
  `Y == 10`

DART in Action (1)

Dynamic Execution

- Random Testing
- Random inputs

  `x = 700, y = 500`
- `x*x*x > 0`
- `y != 10`



Symbolic Execution

- Introduce symbols

  `x1 = X, y1 = Y`
- Constrain `X`

  `X*X*X <= 0`
- Solve constraint

  Non-Linear $\Rightarrow$ Fail
- Concrete fallback

  `x1 = 700`
- Constrain `Y`

  `Y == 10`
- Solve constraint

  `x1 = 700, y1 = 10`

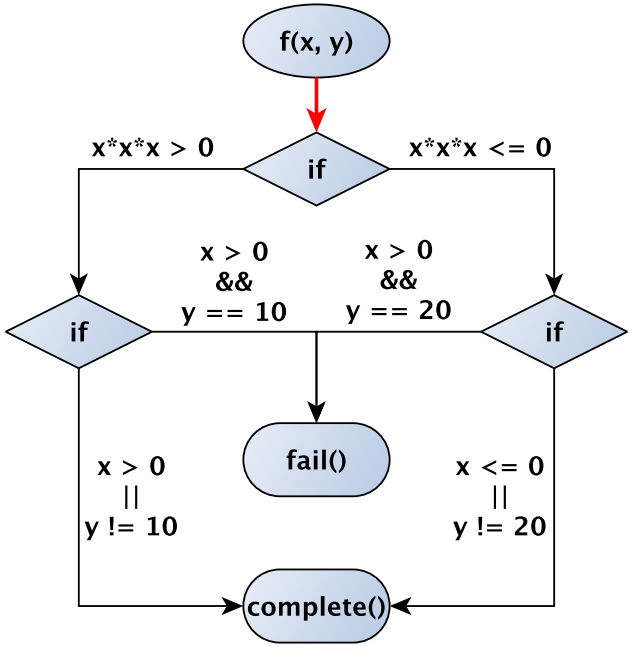
DART in Action (2)

Symbolic Execution

Dynamic Execution

- Given inputs

  `x = 700, y = 10`

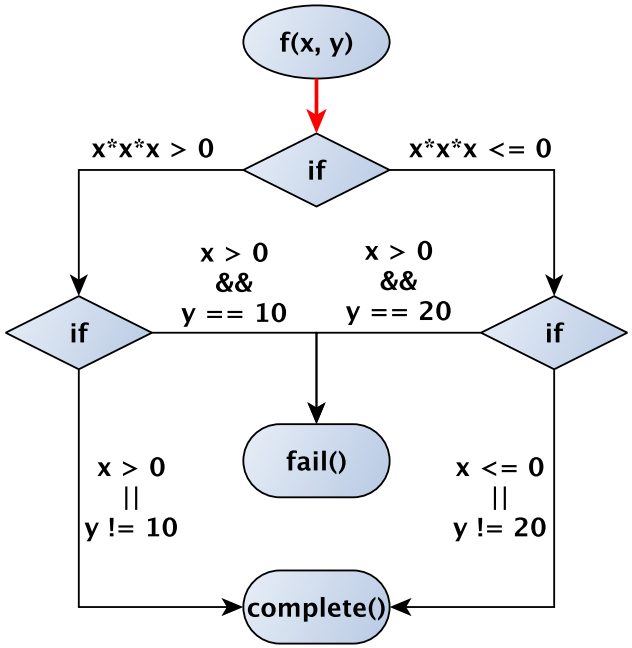


DART in Action (2)

Dynamic Execution

- Given inputs

  `x = 700, y = 10`



Symbolic Execution

- Introduce symbols

  `x2 = X, y2 = Y`

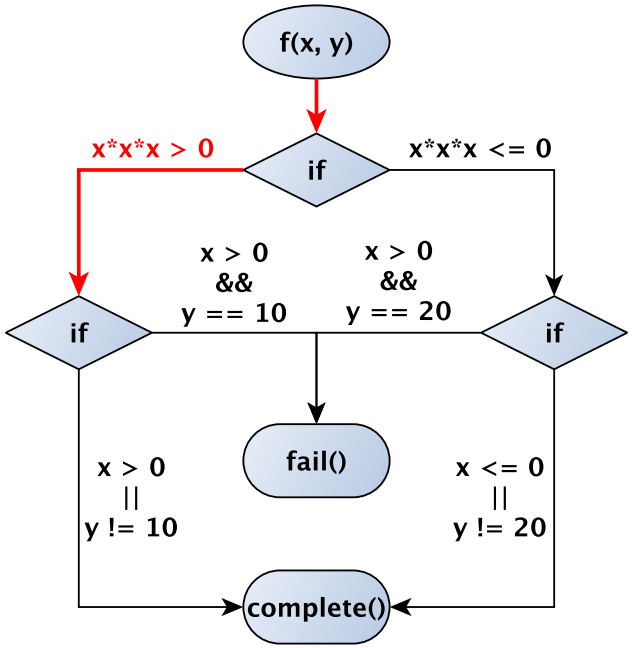
DART in Action (2)

Dynamic Execution

- Given inputs

  `x = 700, y = 10`

- `x*x*x > 0`



Symbolic Execution

- Introduce symbols

  `x2 = X, y2 = Y`

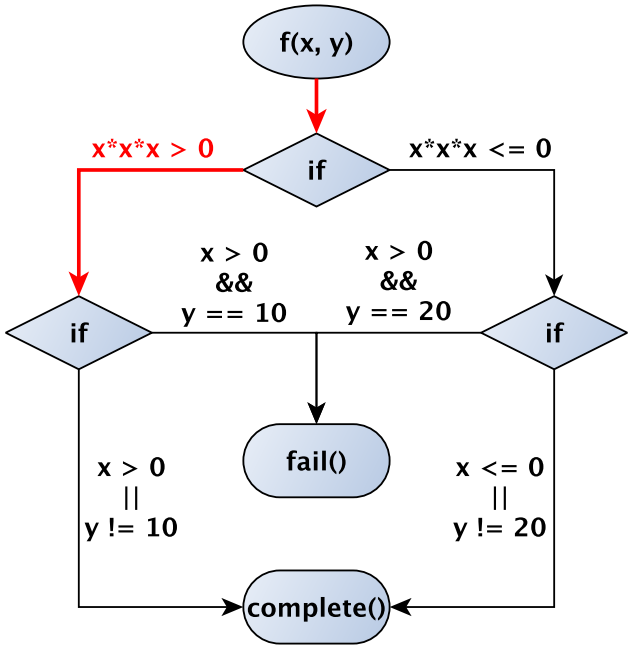
DART in Action (2)

Dynamic Execution

- Given inputs

  `x = 700, y = 10`

- `x*x*x > 0`



Symbolic Execution

- Introduce symbols

  `x2 = X, y2 = Y`

- Constrain `X`

  `X*X*X <= 0`

- Solve constraint
  Non-Linear $\Rightarrow$ Fail

- Concrete fallback

  `x2 = 700`

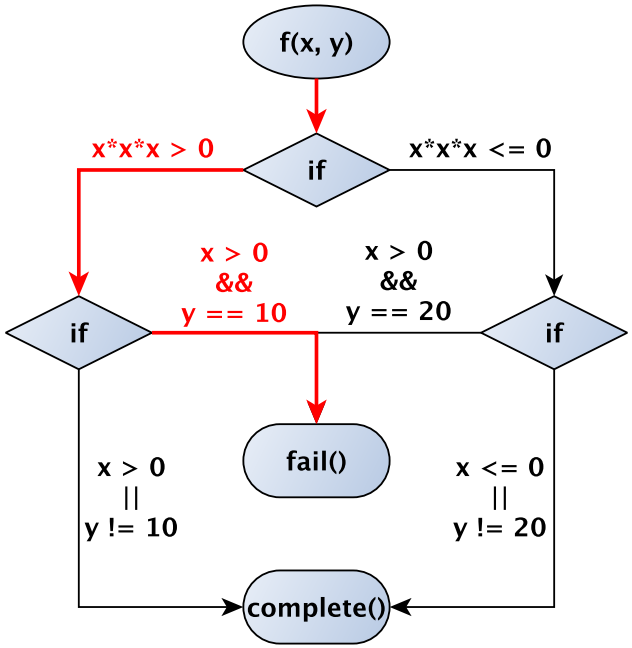
DART in Action (2)

Dynamic Execution

- Given inputs

  `x = 700, y = 10`

- `x*x*x > 0`
- `x > 0 && y == 10`



Symbolic Execution

- Introduce symbols

  `x2 = X, y2 = Y`

- Constrain `X`

  `X*X*X <= 0`

- Solve constraint
  Non-Linear $\Rightarrow$ Fail
- Concrete fallback

  `x2 = 700`

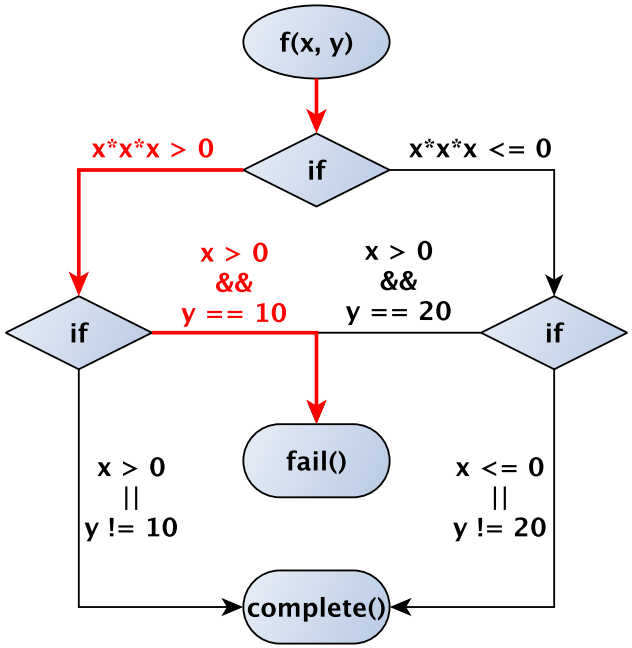
DART in Action (2)

Dynamic Execution

- Given inputs

  `x = 700, y = 10`

- `x*x*x > 0`
- `x > 0 && y == 10`



Symbolic Execution

- Introduce symbols

  `x2 = X, y2 = Y`

- Constrain `X`

  `X*X*X <= 0`

- Solve constraint
  Non-Linear $\Rightarrow$ Fail
- Concrete fallback

  `x2 = 700`

- Branch explored
  $\Rightarrow$ Nothing to do

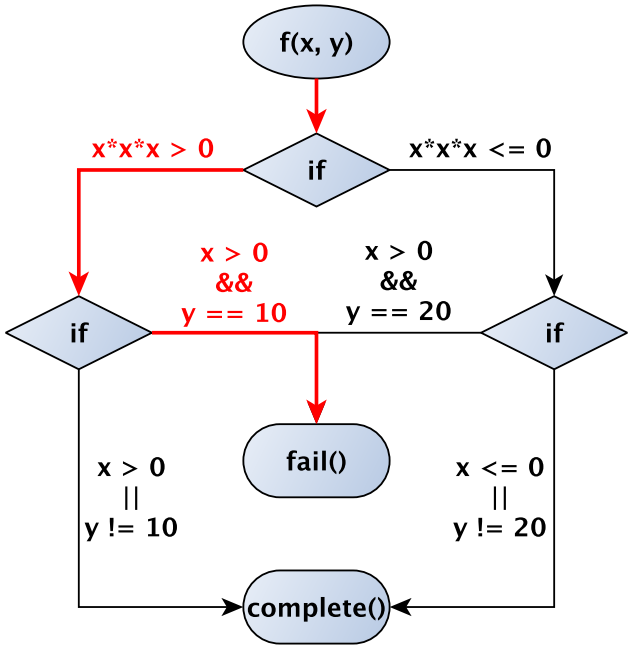
DART in Action (2)

Dynamic Execution

- Given inputs

  `x = 700, y = 10`

- `x*x*x > 0`
- `x > 0 && y == 10`



Symbolic Execution

- Introduce symbols

  `x2 = X, y2 = Y`

- Constrain `X`

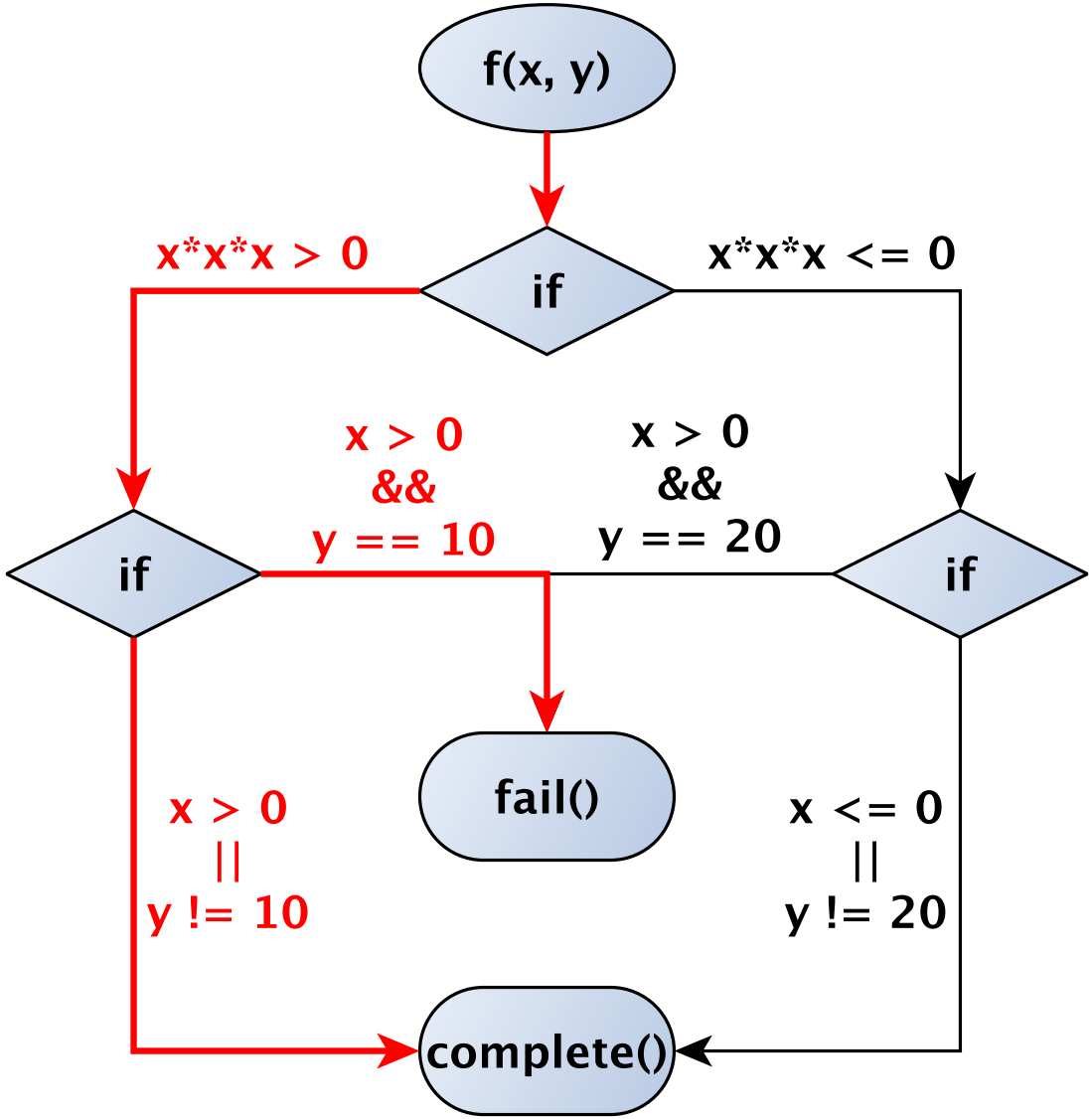
  `X*X*X <= 0`

- Solve constraint
  Non-Linear $\Rightarrow$ Fail
- Concrete fallback

  `x2 = 700`

- Branch explored
  $\Rightarrow$ Nothing to do
- No new inputs

DART in Action (3 . . .)



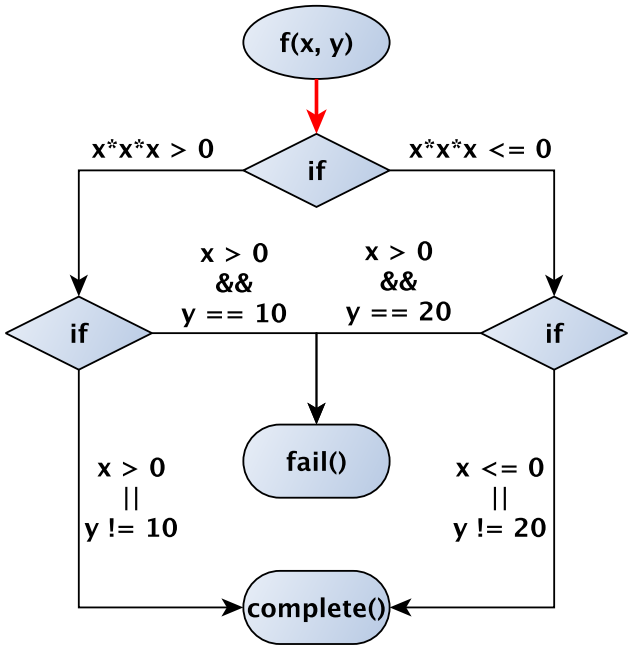
DART in Action (N)

Symbolic Execution

Dynamic Execution

- Random Testing
- Random inputs

  `x = -700, y = 500`

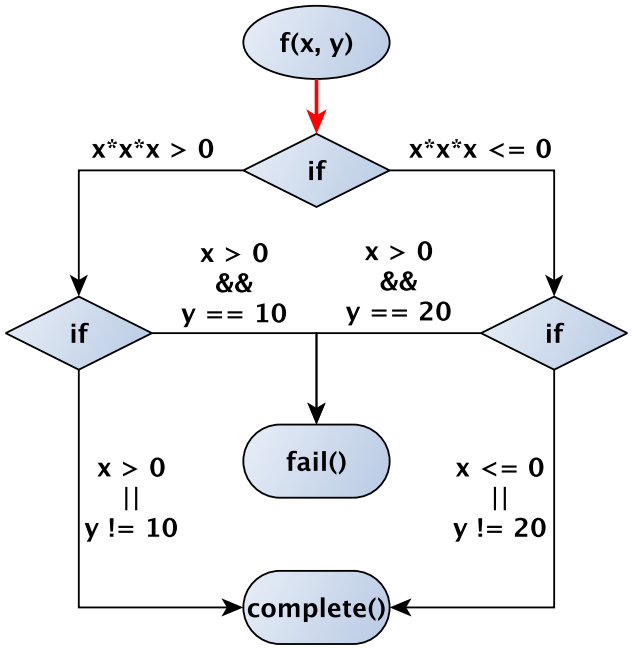


DART in Action (N)

Dynamic Execution

- Random Testing
- Random inputs

`x = -700, y = 500`



Symbolic Execution

- Introduce symbols

`xN = X, yN = Y`

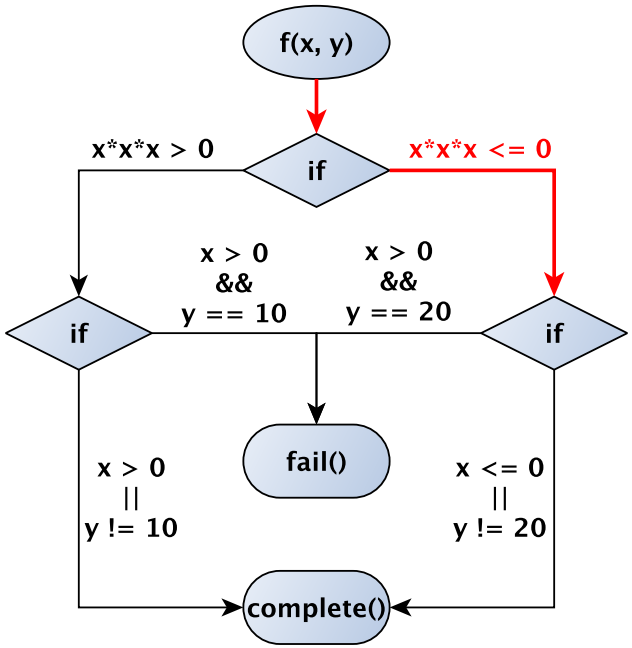
DART in Action (N)

Dynamic Execution

- Random Testing
- Random inputs

  `x = -700, y = 500`

- `x*x*x <= 0`



Symbolic Execution

- Introduce symbols

  `xN = X, yN = Y`

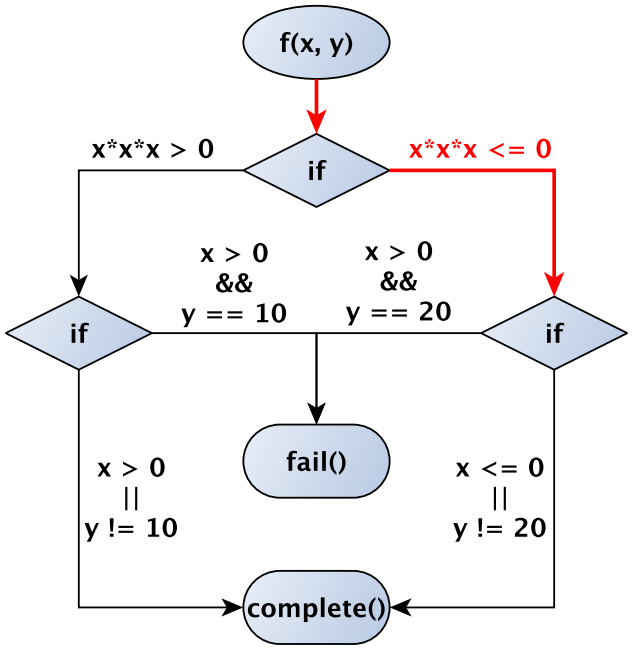
DART in Action (N)

Dynamic Execution

- Random Testing
- Random inputs

  `x = -700, y = 500`

- `x*x*x <= 0`



Symbolic Execution

- Introduce symbols

  `xN = X, yN = Y`

- Branch explored

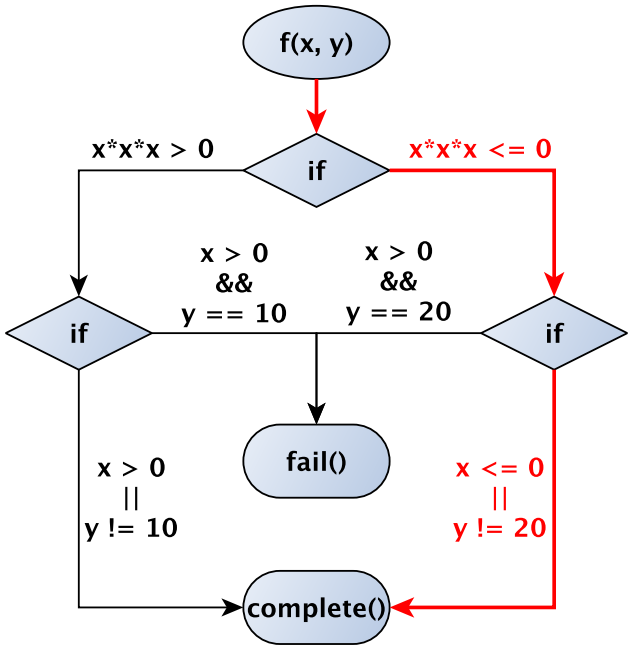
  $\Rightarrow$ Nothing to do

DART in Action (N)

Dynamic Execution

- Random Testing
- Random inputs

  `x = -700, y = 500`
- `x*x*x <= 0`
- `x < 0 && y != 20`



Symbolic Execution

- Introduce symbols

  `xN = X, yN = Y`
- Branch explored

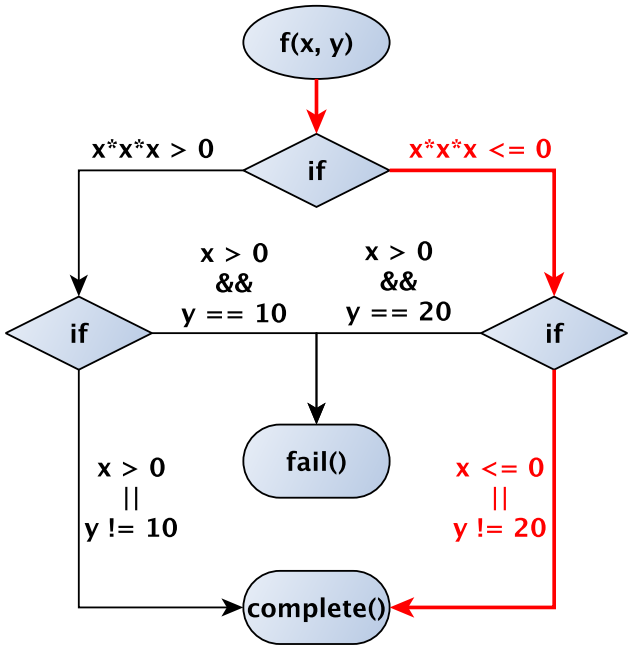
  $\Rightarrow$ Nothing to do

DART in Action (N)

Dynamic Execution

- Random Testing
- Random inputs

  `x = -700, y = 500`
- `x*x*x <= 0`
- `x < 0 && y != 20`



Symbolic Execution

- Introduce symbols

  `xN = X, yN = Y`
- Branch explored

  $\Rightarrow$ Nothing to do
- Constrain `X, Y`

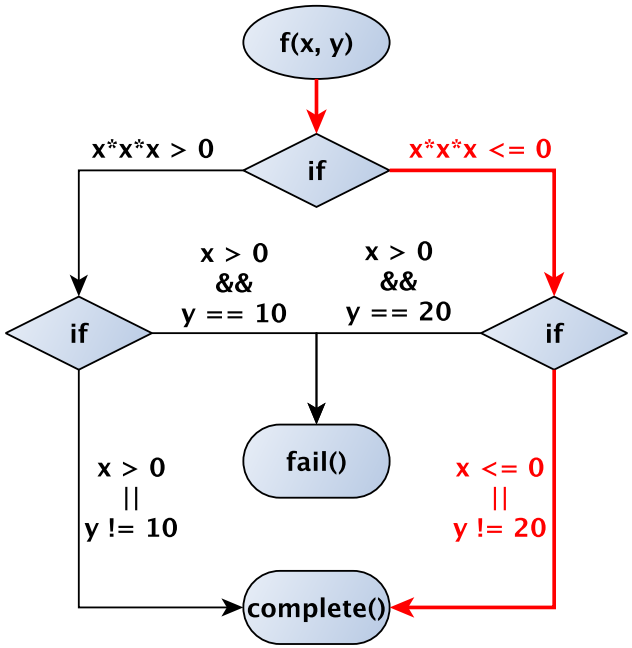
  `X > 0, Y == 20`

DART in Action (N)

Dynamic Execution

- Random Testing
- Random inputs

  `x = -700, y = 500`
- `x*x*x <= 0`
- `x < 0 && y != 20`



Symbolic Execution

- Introduce symbols

  `xN = X, yN = Y`
- Branch explored

  $\Rightarrow$ Nothing to do
- Constrain `X, Y`

  `X > 0, Y == 20`
- Solve constraints

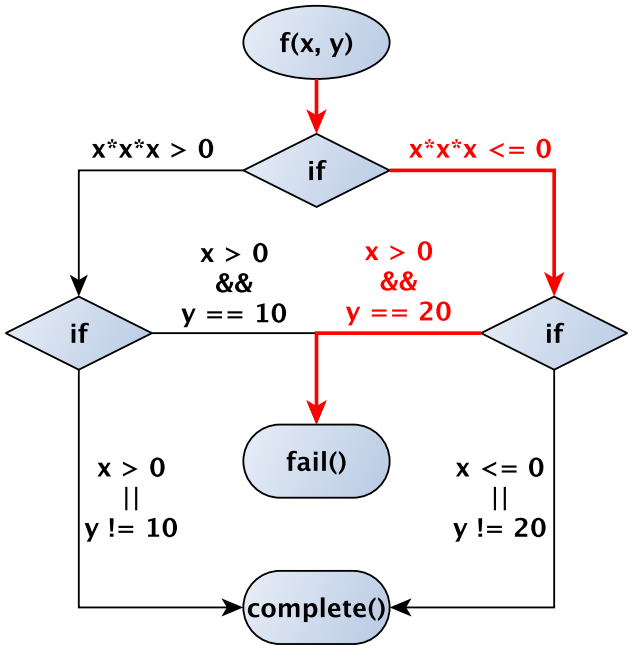
  `xN = 700, yN = 20`

DART in Action (N)

Dynamic Execution

- Random Testing
- Random inputs

  `x = -700, y = 500`
- `x*x*x <= 0`
- `x < 0 && y != 20`



Symbolic Execution

- Introduce symbols

  `xN = X, yN = Y`
- Branch explored

  $\Rightarrow$ Nothing to do
- Constrain `X, Y`

  `X > 0, Y == 20`
- Solve constraints

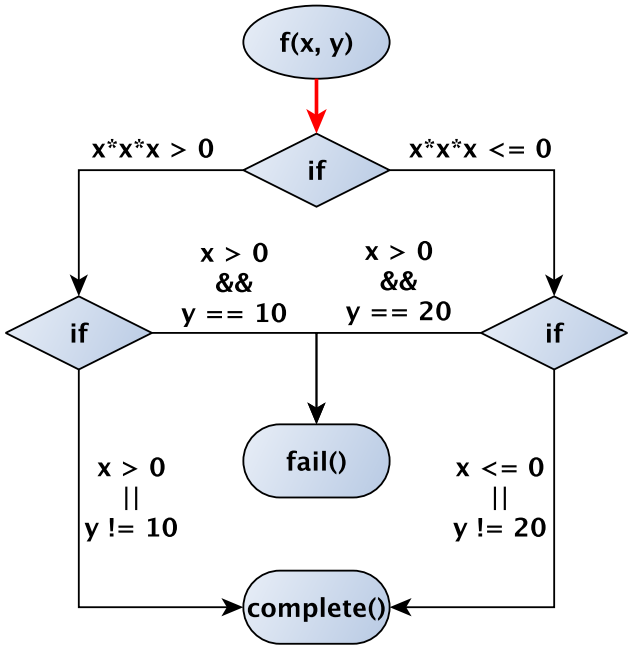
  `xN = 700, yN = 20`
- Assumed path

DART in Action (N+1)

Dynamic Execution

- Given inputs

  `x = 700, y = 20`



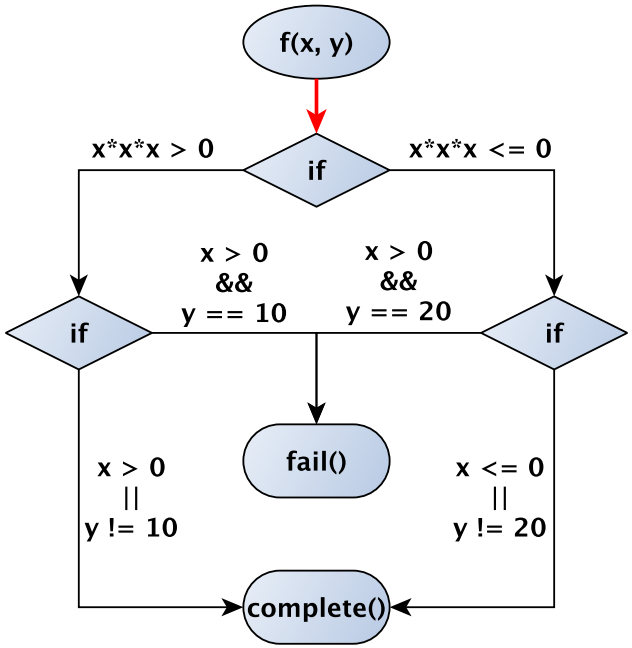
Symbolic Execution

DART in Action (N+1)

Dynamic Execution

- Given inputs

  `x = 700, y = 20`



Symbolic Execution

- Introduce symbols

  `xM = X, yM = Y`

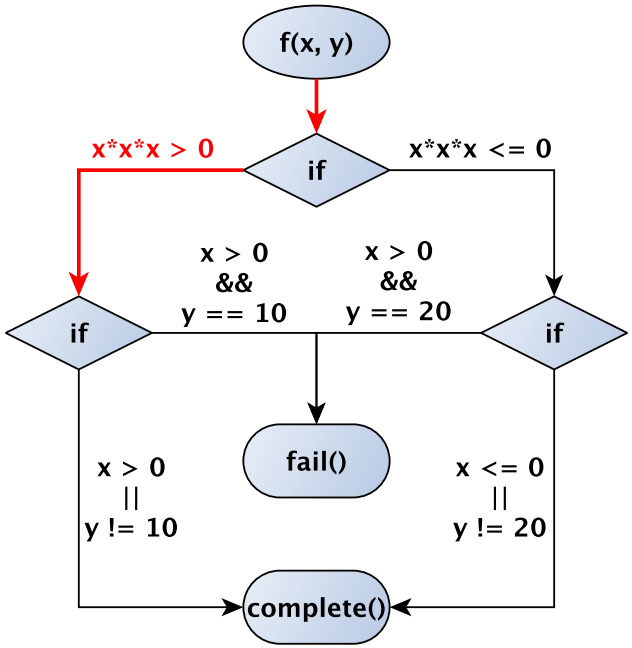
DART in Action (N+1)

Dynamic Execution

- Given inputs

  `x = 700, y = 20`

- `x*x*x > 0`



Symbolic Execution

- Introduce symbols

  `xM = X, yM = Y`

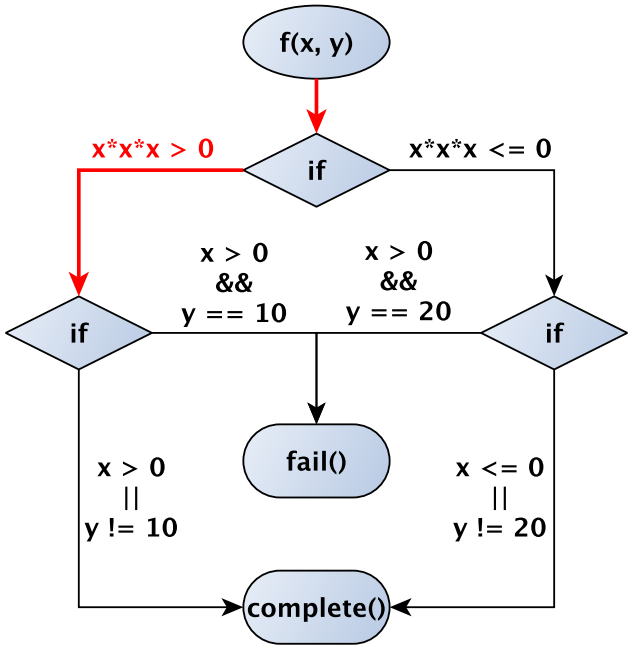
DART in Action (N+1)

Dynamic Execution

- Given inputs

  `x = 700, y = 20`

- `x*x*x > 0`



Symbolic Execution

- Introduce symbols

  `xM = X, yM = Y`

- Branch explored

  $\Rightarrow$ Nothing to do

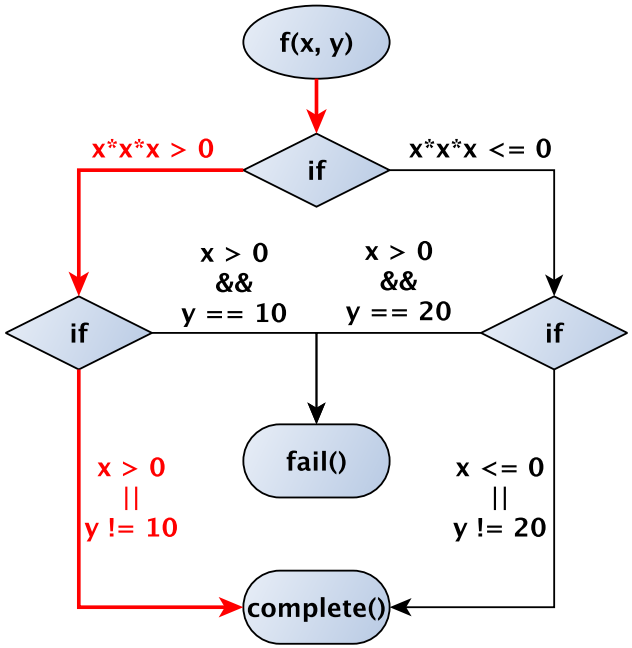
DART in Action (N+1)

Dynamic Execution

- Given inputs

  `x = 700, y = 20`

- `x*x*x > 0`
- `x > 0 && y != 10`



Symbolic Execution

- Introduce symbols

  `xM = X, yM = Y`

- Branch explored

  $\Rightarrow$ Nothing to do

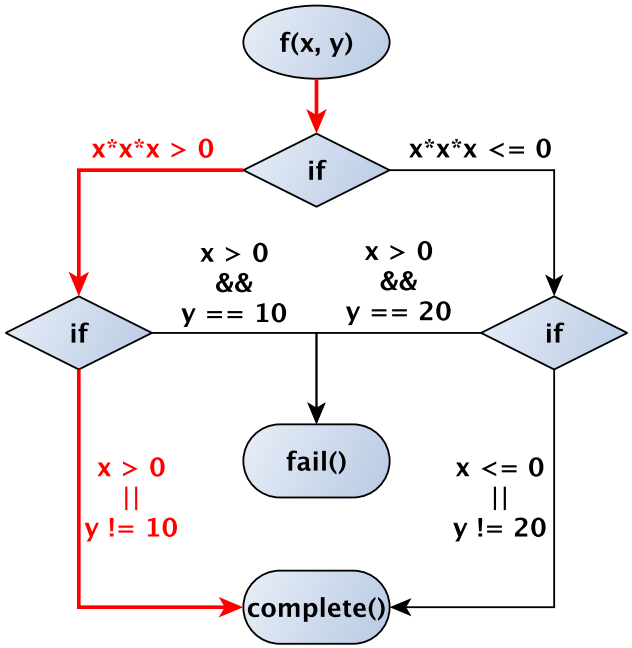
DART in Action (N+1)

Dynamic Execution

- Given inputs

  `x = 700, y = 20`

- `x*x*x > 0`
- `x > 0 && y != 10`



Symbolic Execution

- Introduce symbols

  `xM = X, yM = Y`

- Branch explored

  $\Rightarrow$ Nothing to do

- Branch explored

  $\Rightarrow$ Nothing to do

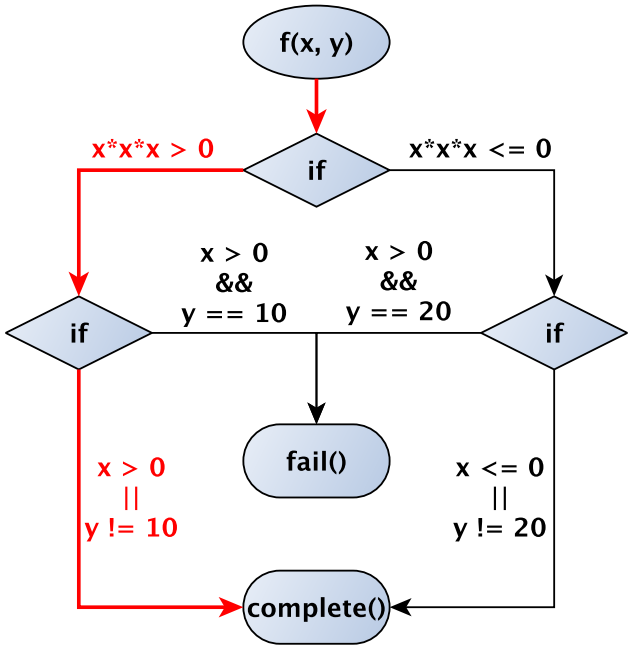
DART in Action (N+1)

Dynamic Execution

- Given inputs

  `x = 700, y = 20`

- `x*x*x > 0`
- `x > 0 && y != 10`



⇒ Same as 1st path!

Symbolic Execution

- Introduce symbols

  `xM = X, yM = Y`

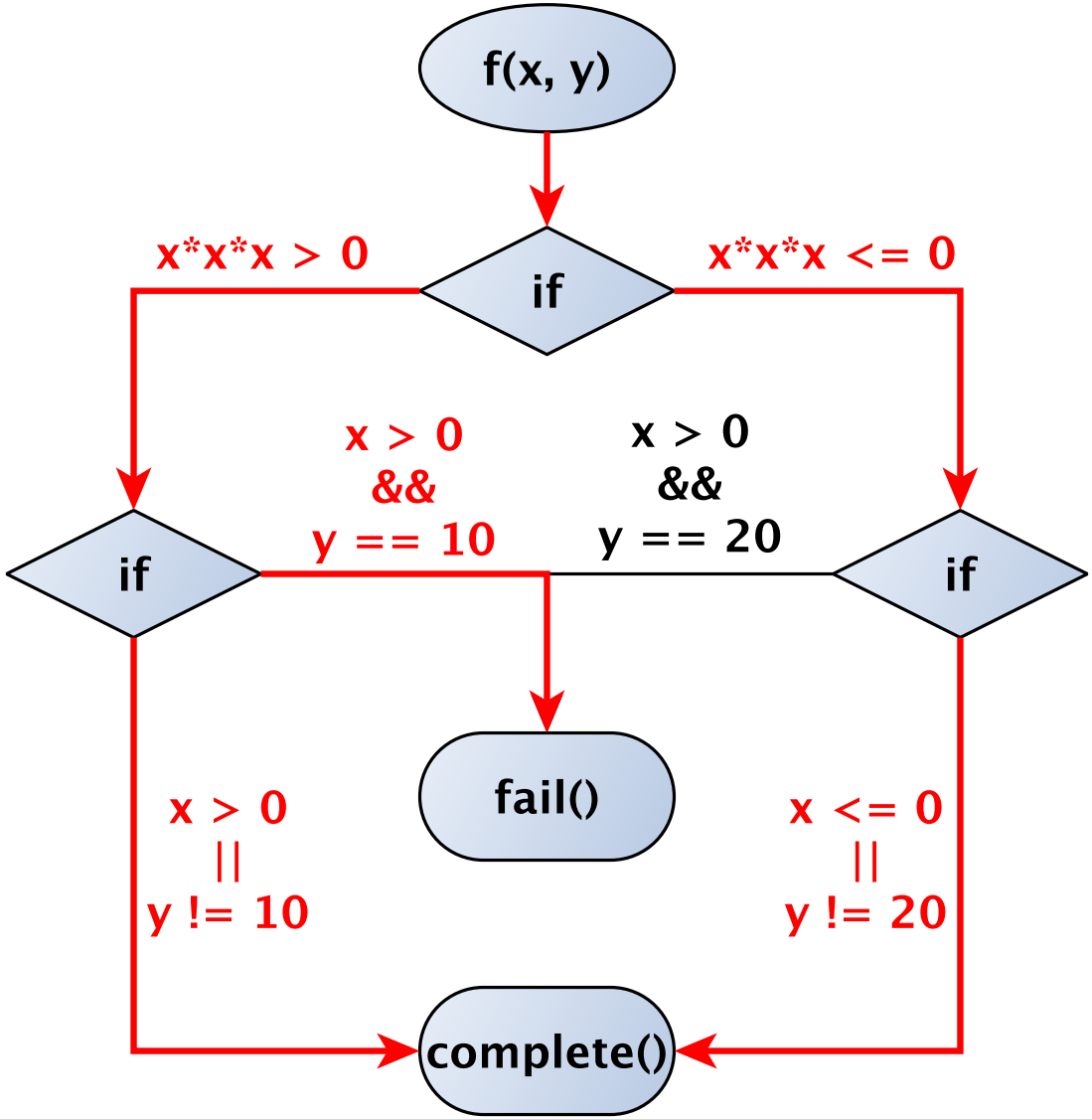
- Branch explored

  ⇒ Nothing to do

- Branch explored

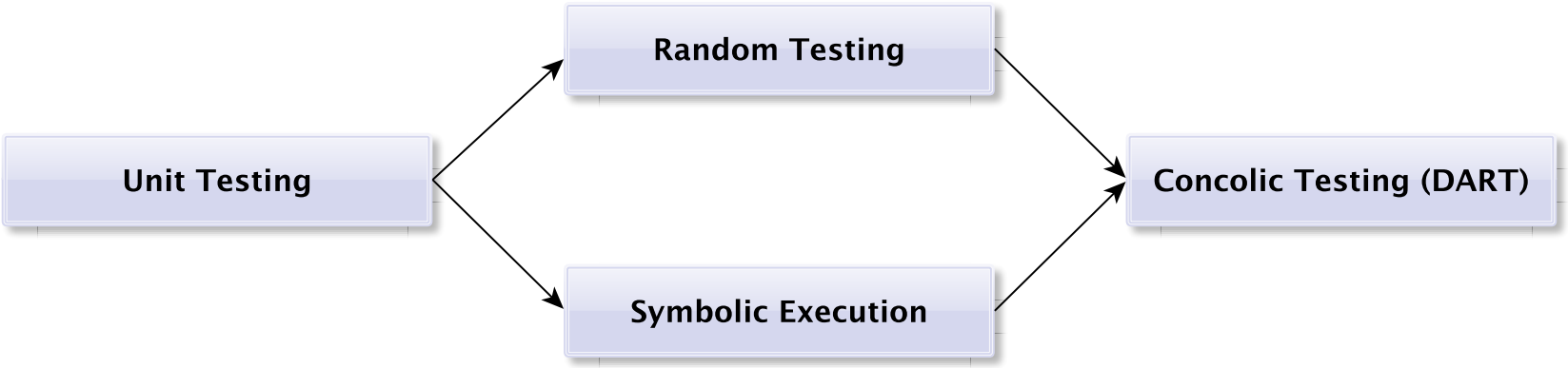
  ⇒ Nothing to do

DART Completed



Overview

Summary



$\Rightarrow$ Classification of DART

$\Rightarrow$ Drawbacks of Basic Techniques Solely

$\Rightarrow$ Improvement with Concolic Testing

References

[1] Patrice Godefroid, Nils Klarlund, and Koushik Sen. “DART: Directed Automated Random Testing”. In: *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI ’05. Chicago, IL, USA: ACM, 2005, pp. 213–223. ISBN: 1-59593-056-6. DOI: 10.1145/1065010.1065036. URL: http://doi.acm.org/10.1145/1065010.1065036.

[2] James C. King. “Symbolic Execution and Program Testing”. In: *Commun. ACM* 19.7 (July 1976), pp. 385–394. ISSN: 0001-0782. DOI: 10.1145/360248.360252. URL: http://doi.acm.org/10.1145/360248.360252.

[3] Wikipedia. *Symbolic Execution*. June 2015. URL: http://en.wikipedia.org/wiki/Symbolic_execution.