Analyzing Configurable Systems with Dynamic Taint Analysis

Miguel Velez, Christian Kästner, Pooyan Jamshidi, Florian Sattler, Norbert Siegmund, Sven Apel

Find How Options Influence Control-Flow Decisions

Analyze How Options Interact

Program understanding

Configuration testing

Performance analysis

Exploiting Structure and Behavior of Highly Configurable Systems to Measure Performance

Miguel Velez
Software Engineering Ph.D. student
Carnegie Mellon University
Joint work with: Christian Kästner, Pooyan Jamshidi, and Norbert Siegmund

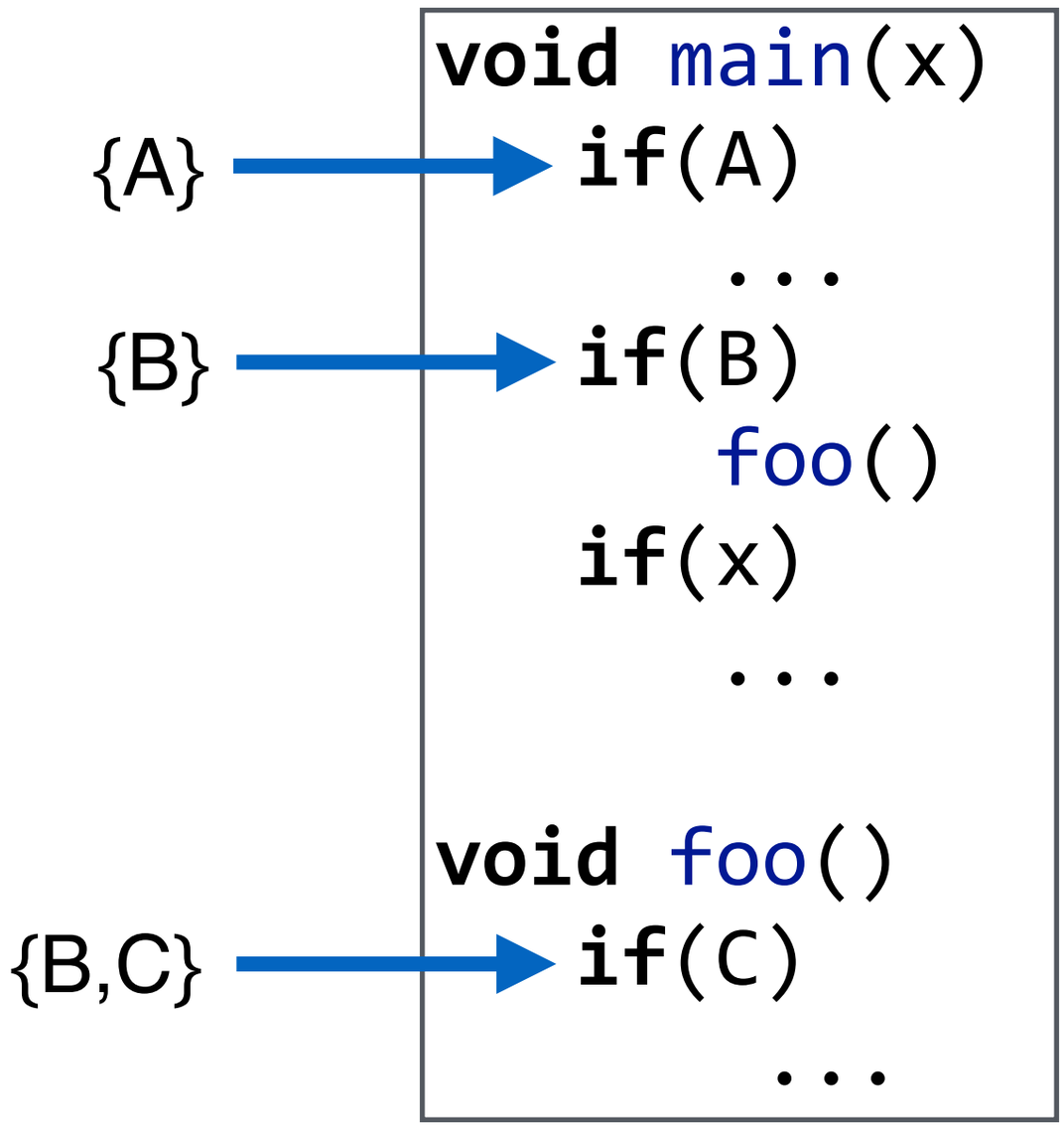
ConfigCrusher: White-Box Performance Analysis for Configurable Systems

Miguel Velez, Pooyan Jamshidi, Florian Sattler, Norbert Siegmund, Sven Apel, Christian Kästner

Analyze How Options Interact

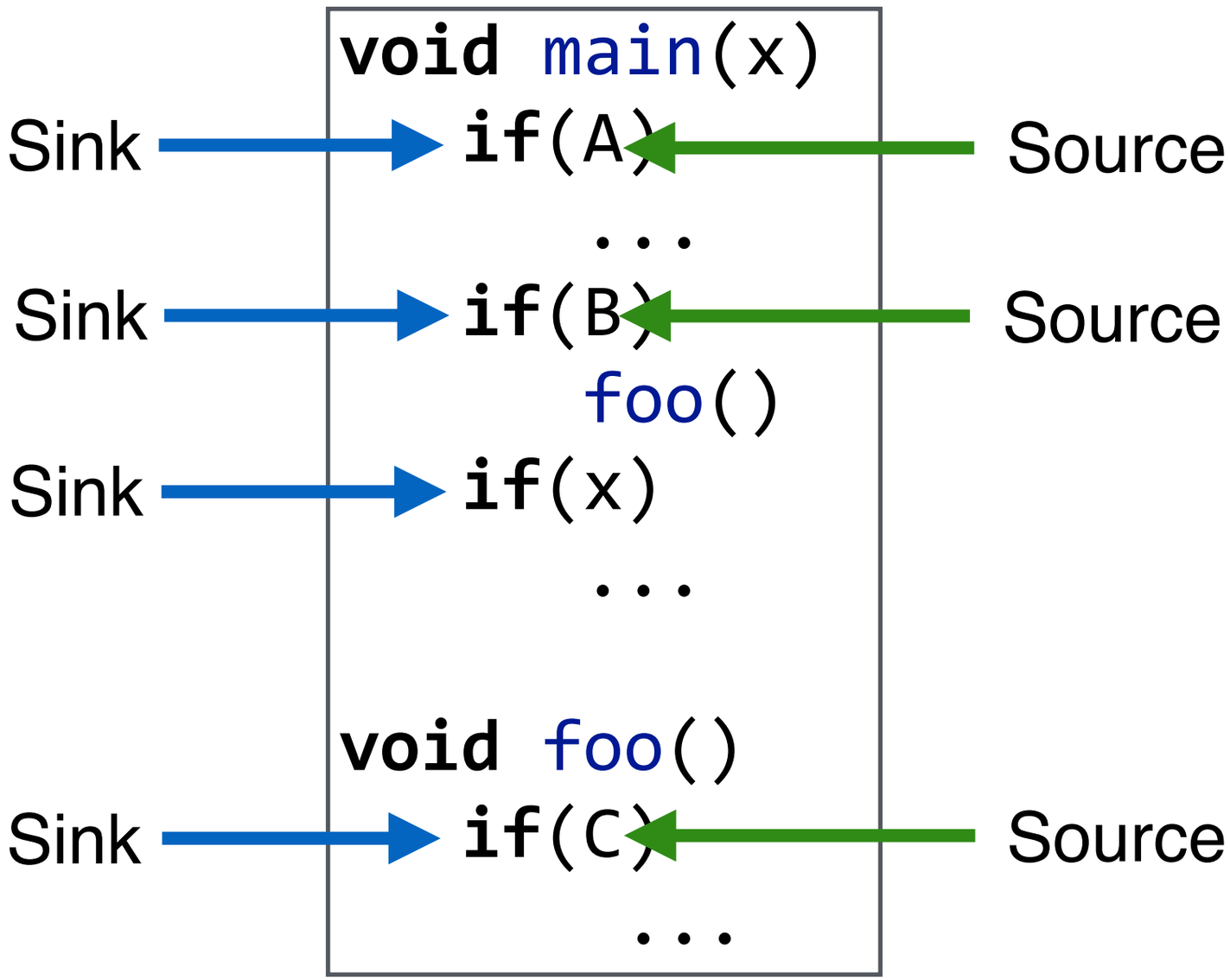
```
void main(x)
   if(A)
      ...
   if(B)
      foo()
   if(x)
      ...

void foo()
   if(C)
      ...
```

Analyze How Options Interact

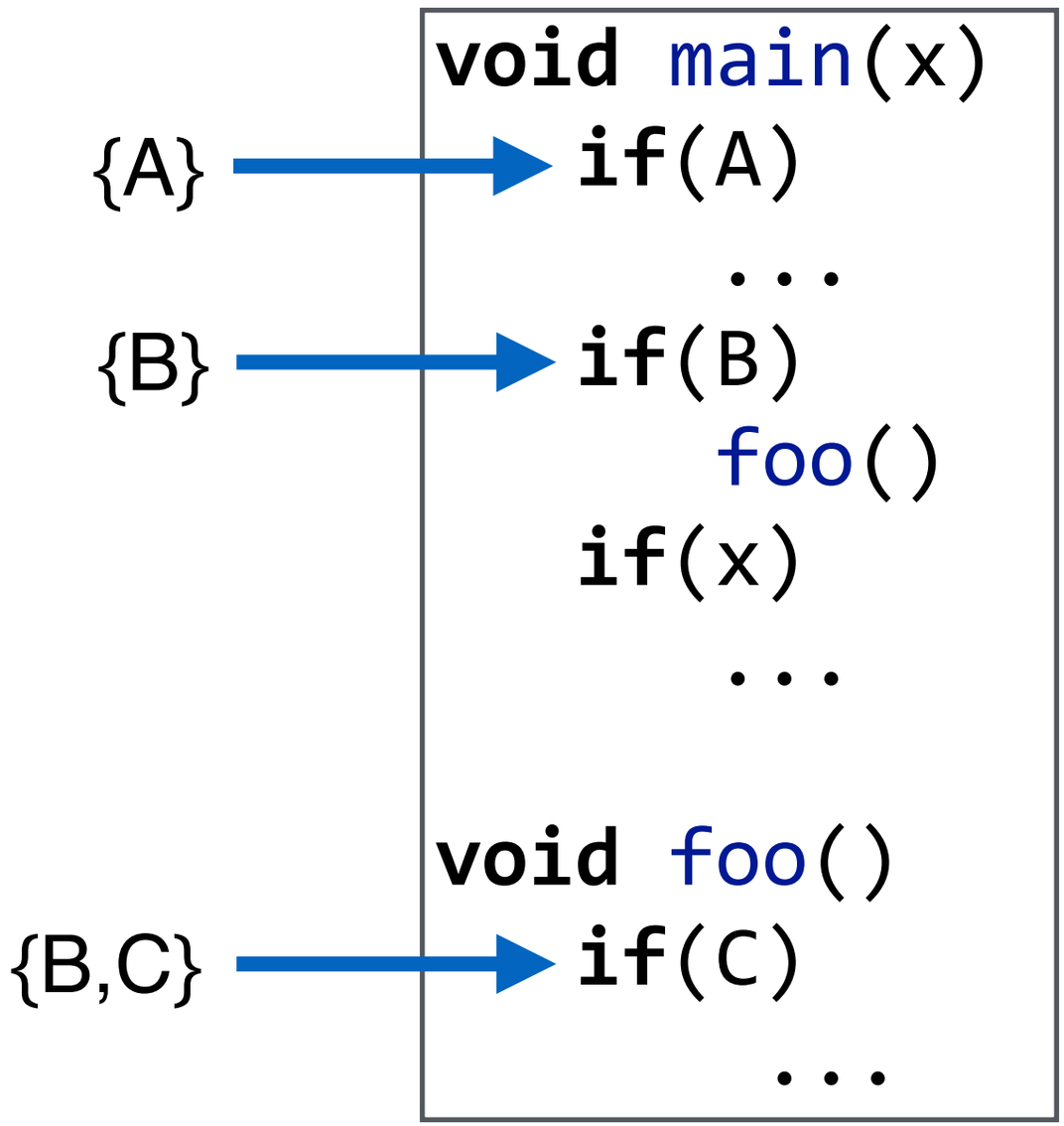


Static Taint Analysis

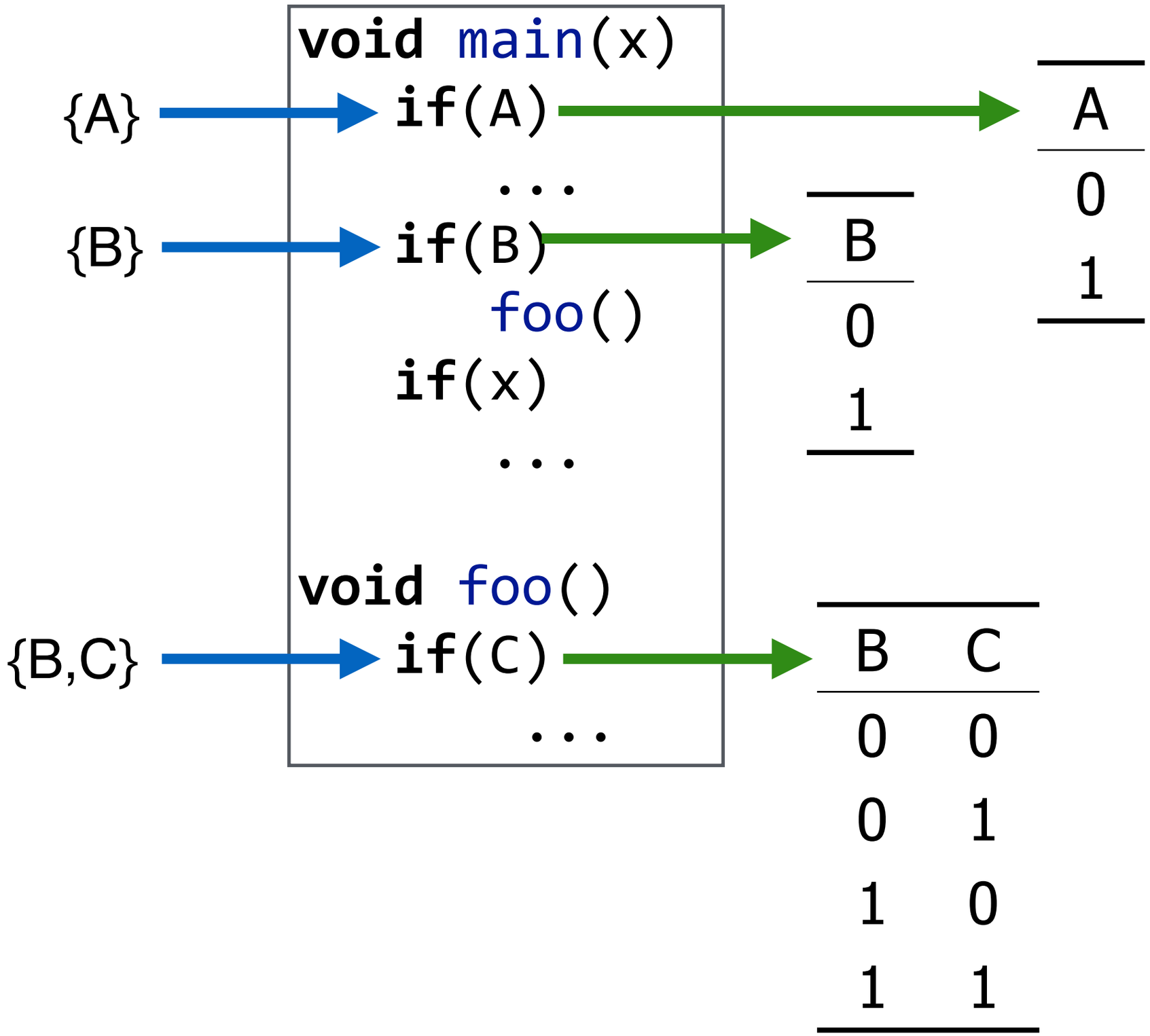
Taint Analysis



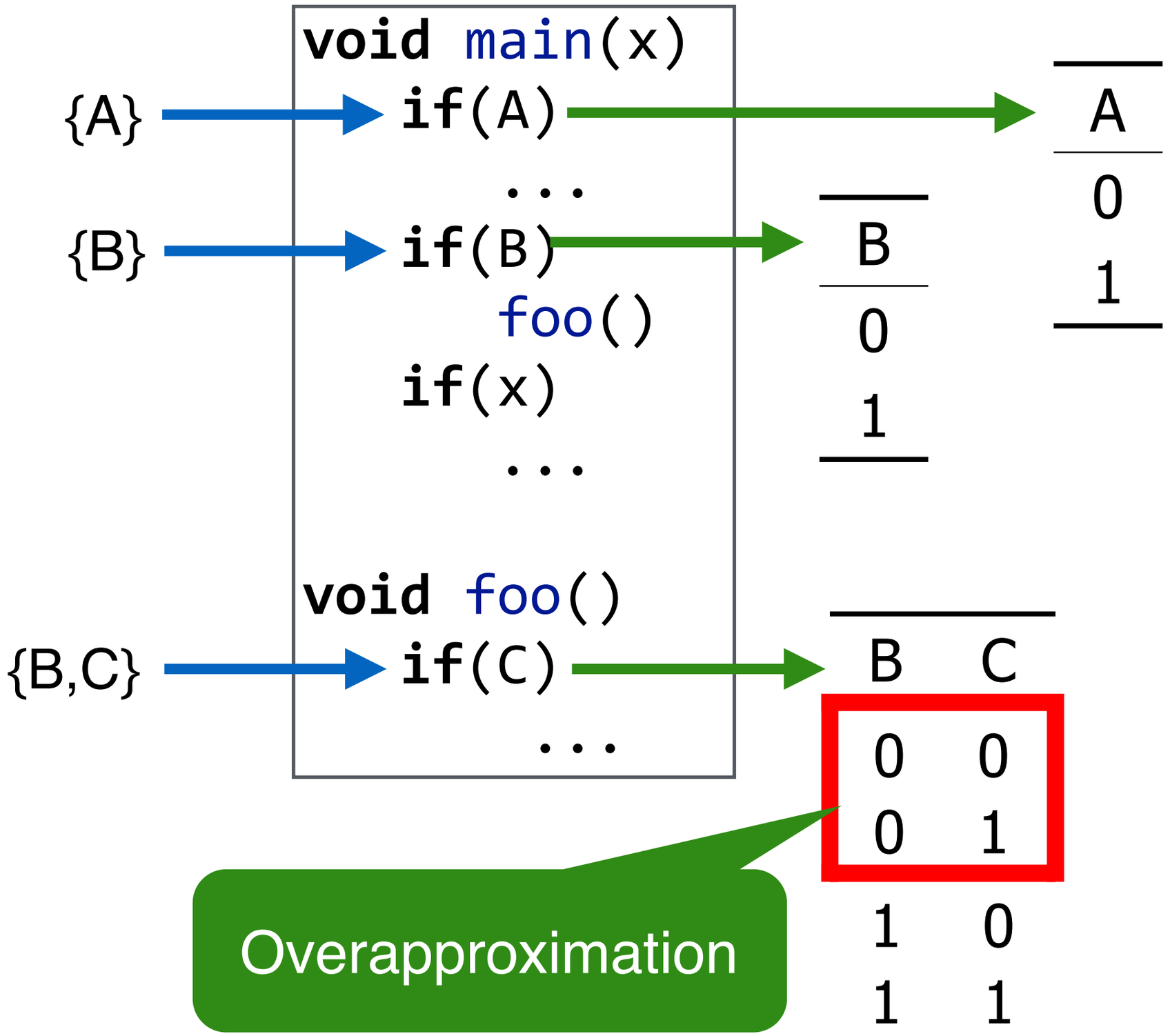
Static Taint Analysis



Find Configurations to Efficiently Explore Decisions



Find Configurations to Efficiently Explore Decisions



SPLat

SPLat: Lightweight Dynamic Analysis for Reducing Combinatorics in Testing Configurable Systems

Chang Hwan Peter Kim
University of Texas
Austin, TX, USA

Darko Marinov[1,2]
[1]University of Illinois
and [2]Groupon, Inc., USA

Sarfraz Khurshid Don Batory
University of Texas
Austin, TX, USA

Sabrina Souto Paulo Barros Marcelo d'Amorim
Federal University of Pernambuco
Recife, PE, Brazil

ABSTRACT

Many programs can be configured through dynamic and/or static selection of configuration variables. A *software product line (SPL)*, for example, specifies a family of programs where each program is defined by a unique combination of features. Systematically testing SPL programs is expensive as it can require running each test against a combinatorial number of configurations. Fortunately, a test is often independent of many configuration variables and need not be run against every combination. Configurations that are not

example, specifies a family of programs where each program is defined by a unique combination of features (increments in program functionality). Many codebases that power modern websites are also highly configurable. For instance, the code behind the `groupon.com` website has over 170 boolean configuration variables and can, in theory, be deployed in over 2^{170} different configurations.

Systematically testing configurable systems and SPLs is challenging because running each test can, in principle, require many actual executions—one execution for each pos-

SPLat Infers Control-Flow Interactions

```
void main(x)
   if(A)
      ...
   if(B)
      foo()
   if(x)
      ...

void foo()
   if(C)
      ...
```

B	C
1	0
1	1

SPLat is a Pseudo-Brute-Force Approach

```
void main(x)
   if(A)
      ...
   if(B)
      foo()
   if(x)
      ...

void foo()
   if(C)
      ...
```

A	B
0	0
0	1
1	0
1	1

Overapproximation

Dynamic Taint Analysis

Iterative Dynamic Taint Analysis

Iterative Dynamic Taint Analysis

Pick initial configuration to execute

Iterative Dynamic Taint Analysis

Pick initial configuration to execute

Run dynamic taint analysis

Iterative Dynamic Taint Analysis

Pick initial configuration to execute

Run dynamic taint analysis

Derive configurations to explore

Iterative Dynamic Taint Analysis

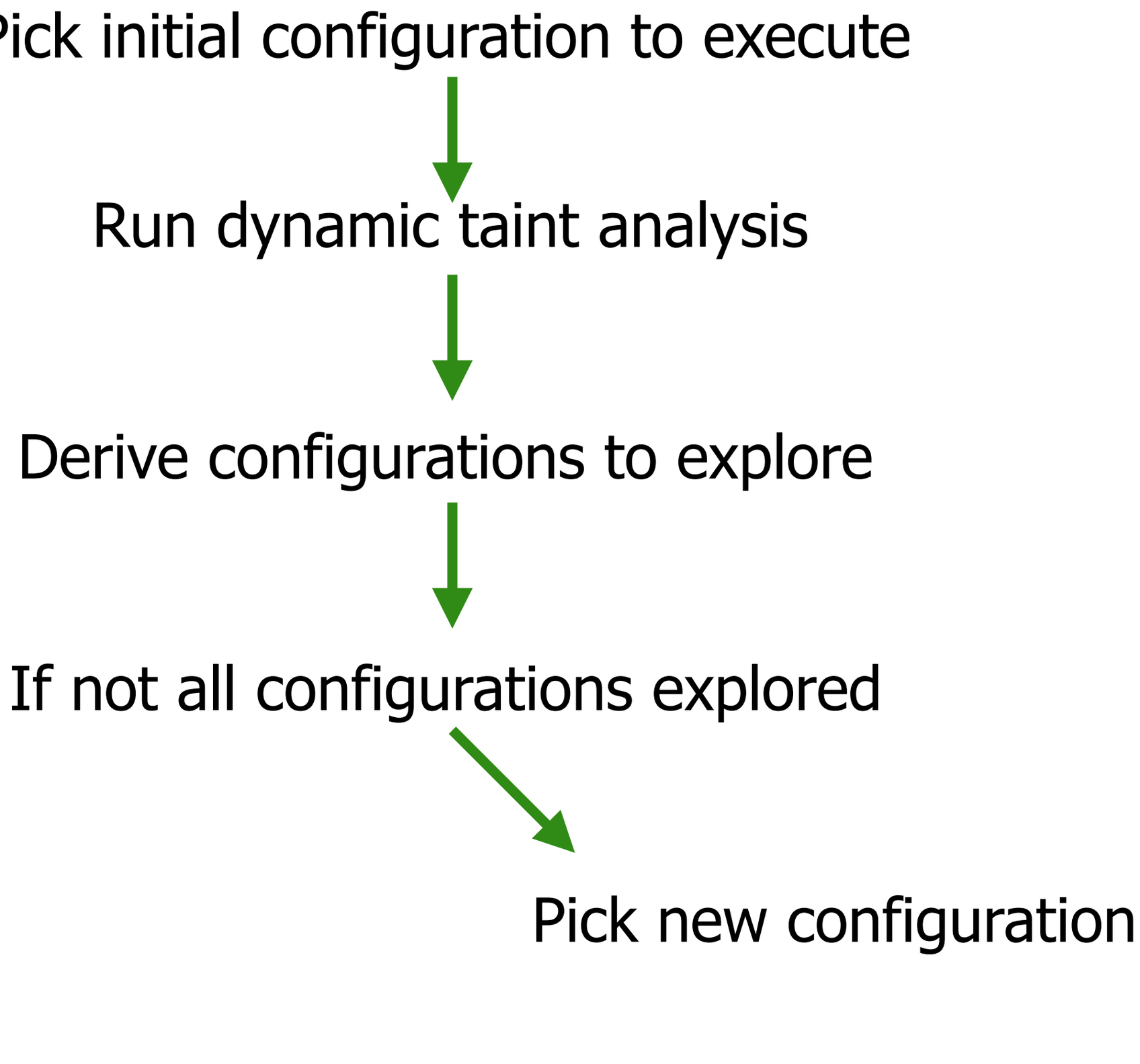
Pick initial configuration to execute

Run dynamic taint analysis

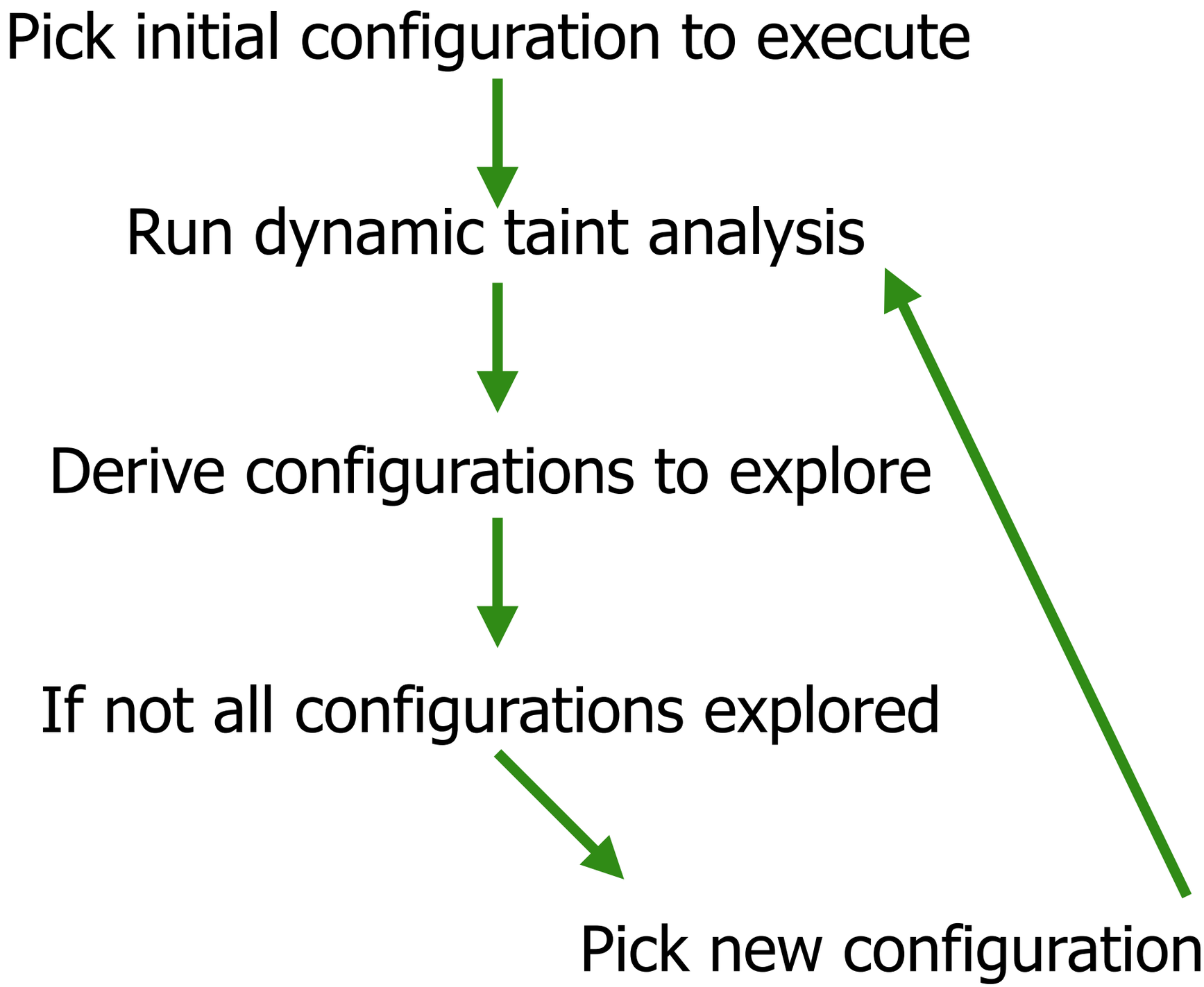
Derive configurations to explore

If not all configurations explored

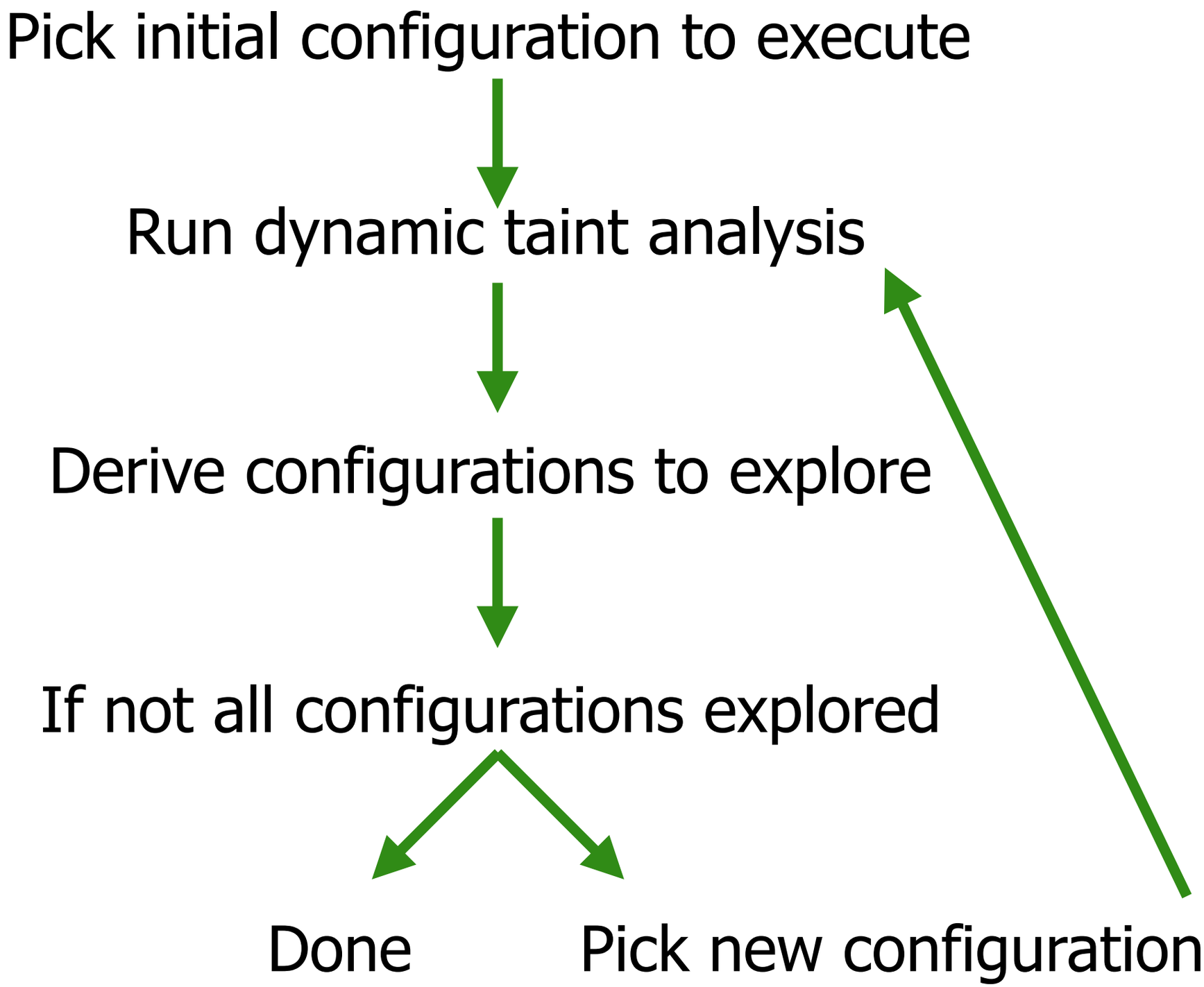
Iterative Dynamic Taint Analysis



Iterative Dynamic Taint Analysis



Iterative Dynamic Taint Analysis



Pick Configuration to Execute

A	B	C
0	0	0

```
void main(x)
   if(A)
      ...
   if(B)
      foo()
   if(x)
      ...

void foo()
   if(C)
      ...
```

Dynamic Taint Analysis

A	B	C
0	0	0

```
void main(x)
    if(A) // A=0
        ...
    if(B) // B=0
        foo()
    if(x)
        ...

void foo()
    if(C) // C=0
        ...
```

Dynamic Taint Analysis

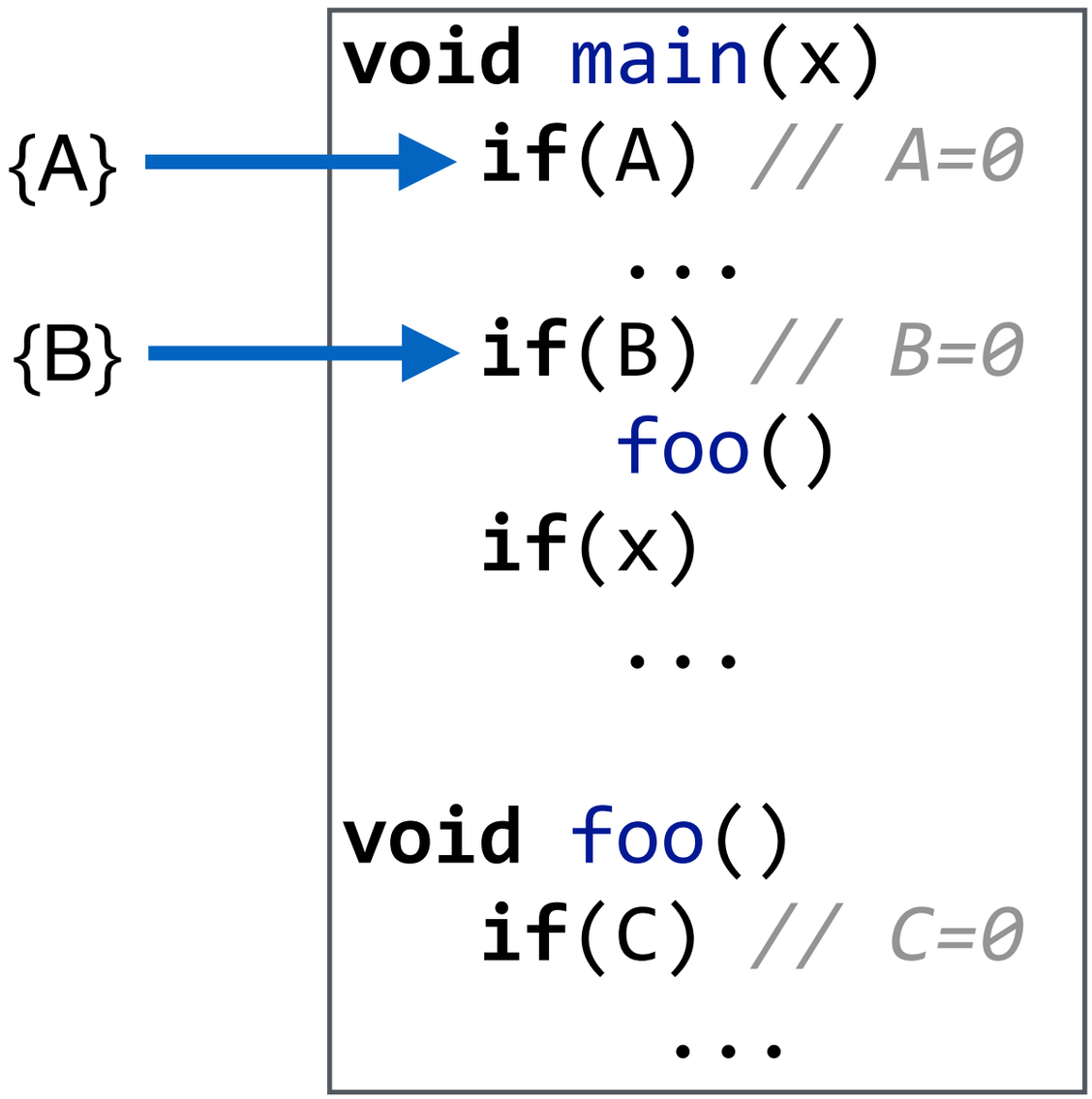
A	B	C
0	0	0

{A}

```
void main(x)
    if(A) // A=0
        ...
    if(B) // B=0
        foo()
    if(x)
        ...

void foo()
    if(C) // C=0
        ...
```

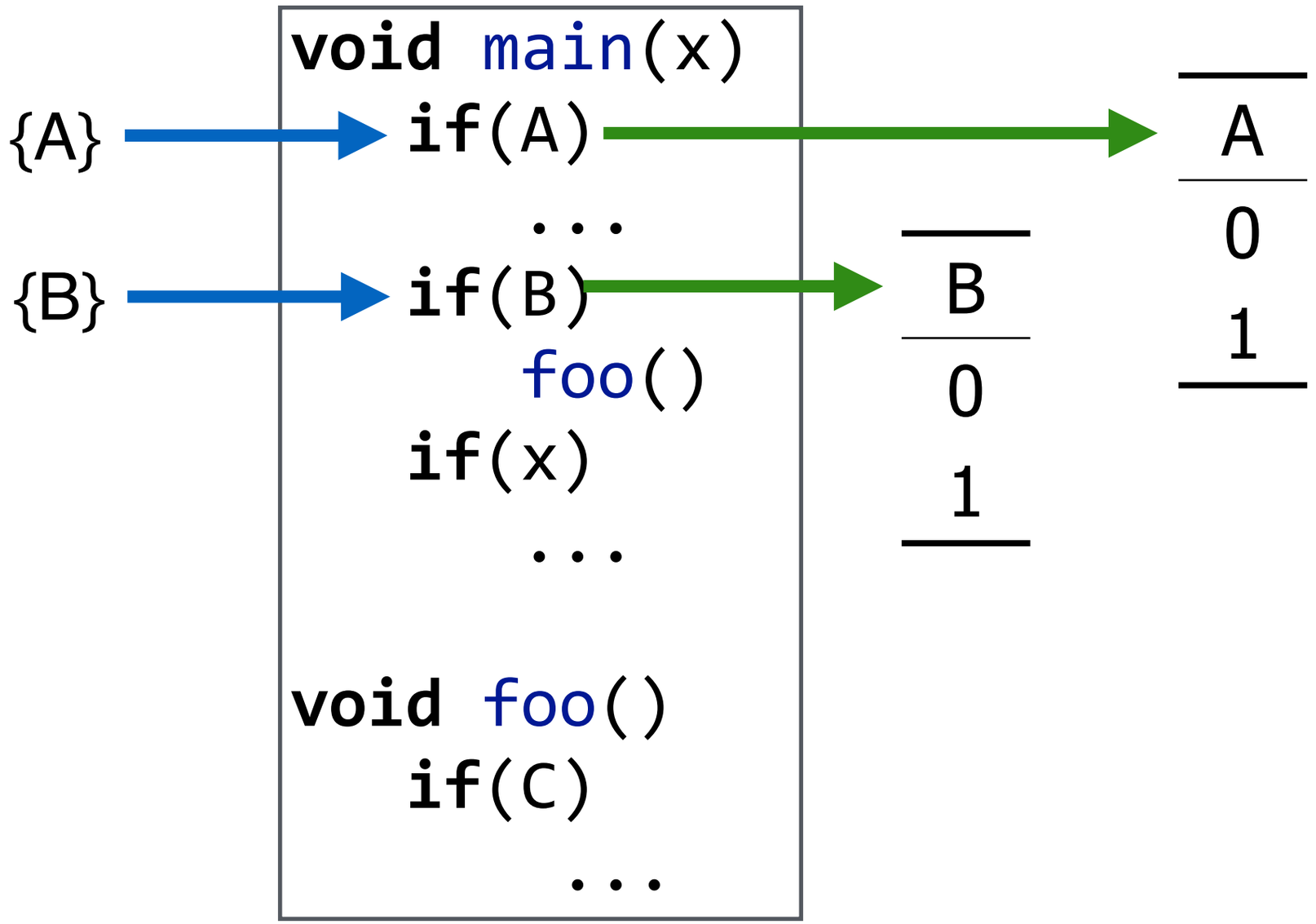
Dynamic Taint Analysis

A	B	C
0	0	0



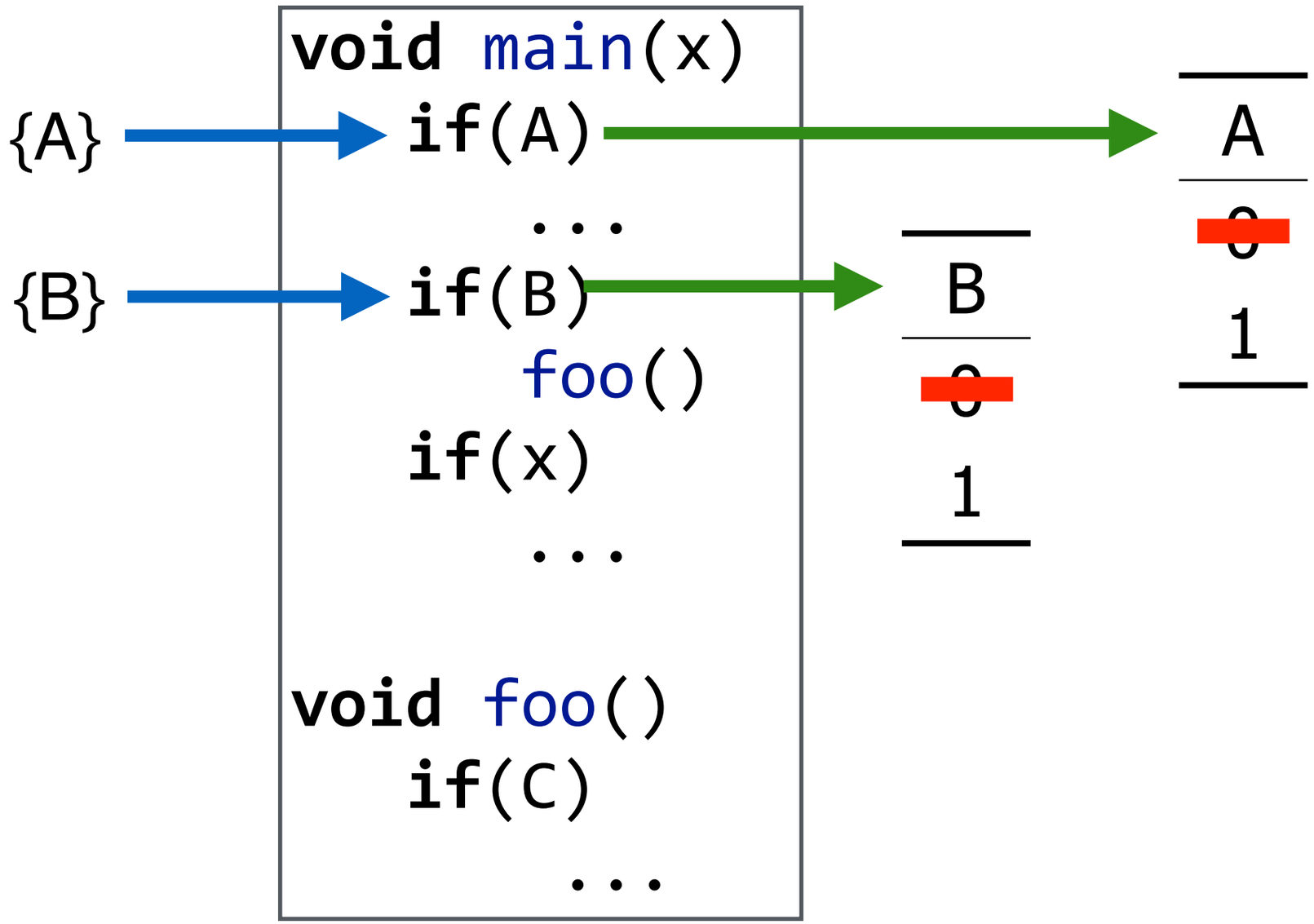
Derive Configurations to Explore

A	B	C
0	0	0



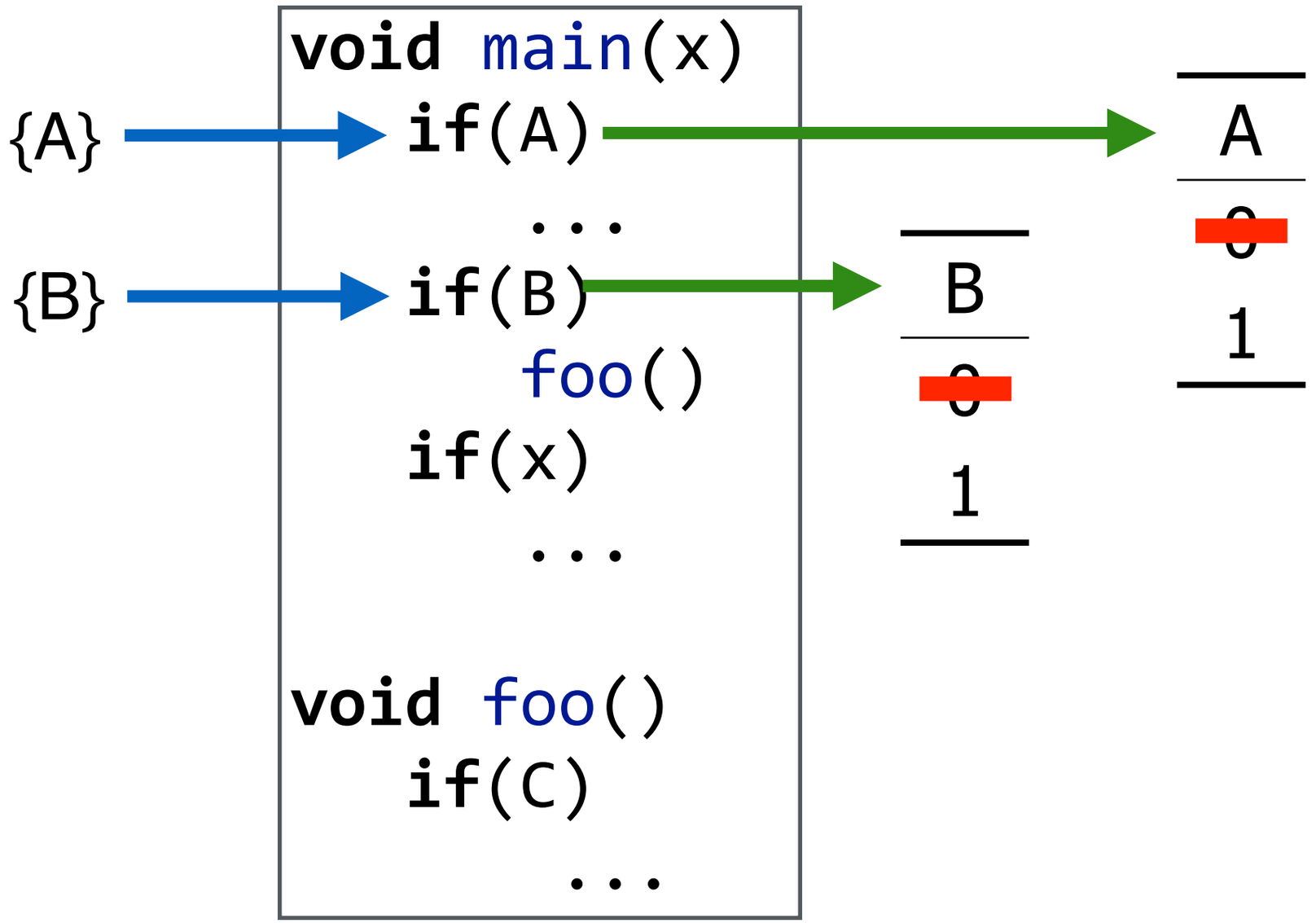
Derive Configurations to Explore

A	B	C
0	0	0



Pick Next Configuration to Execute

A	B	C
1	1	0



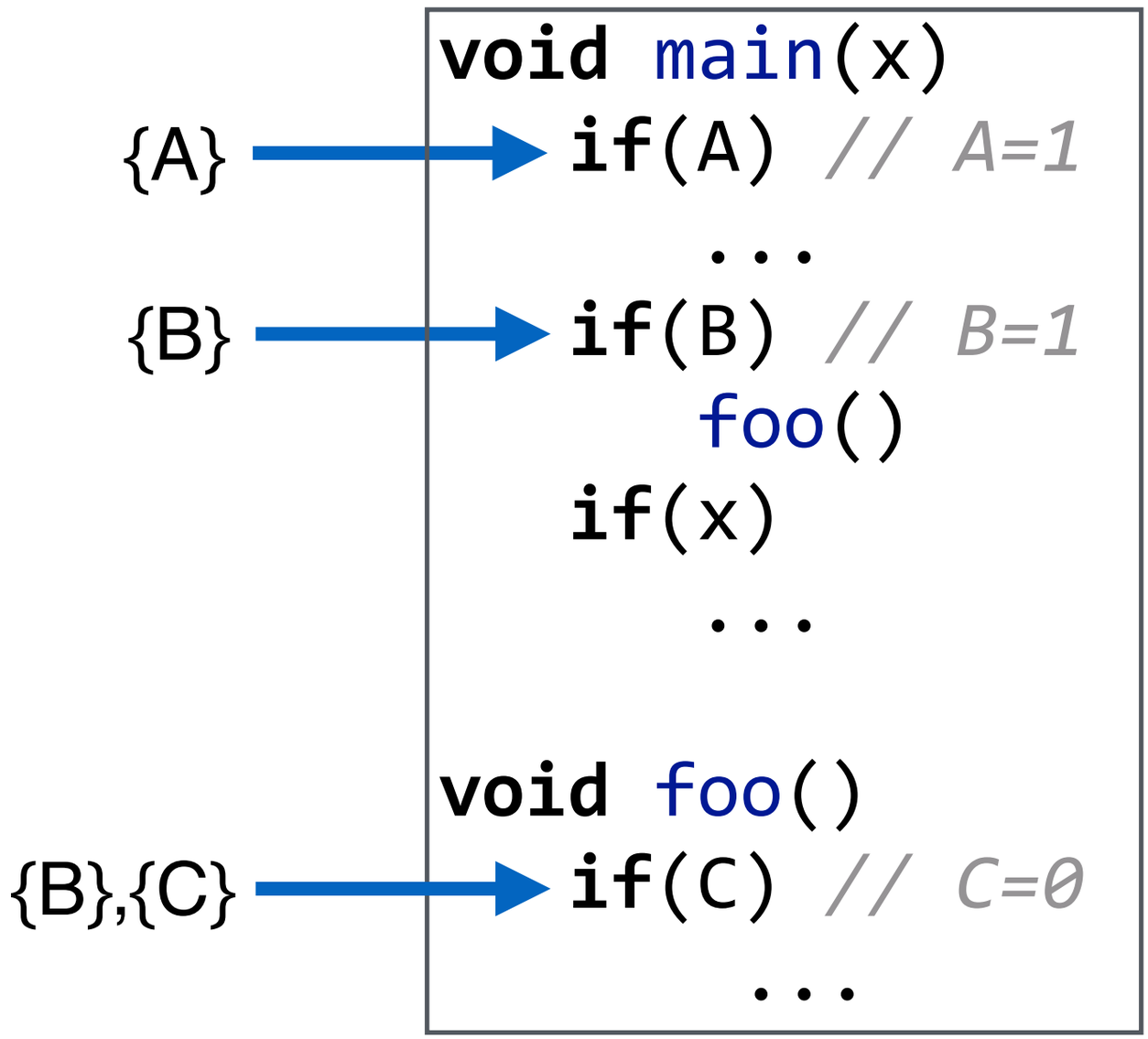
Dynamic Taint Analysis

A	B	C
1	1	0

```
void main(x)
    if(A) // A=1
        ...
    if(B) // B=1
        foo()
    if(x)
        ...

void foo()
    if(C) // C=0
        ...
```

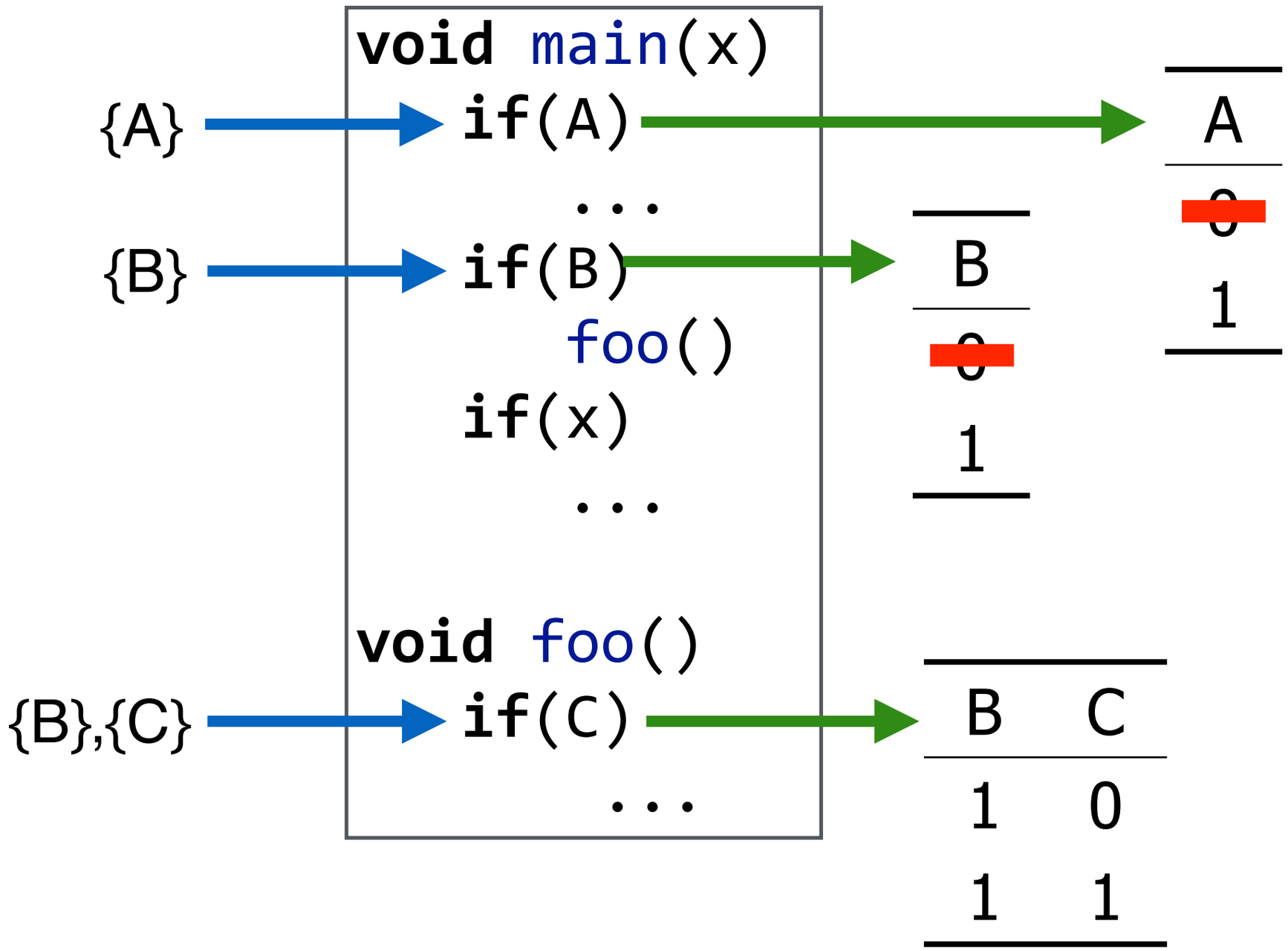
Dynamic Taint Analysis

A	B	C
1	1	0



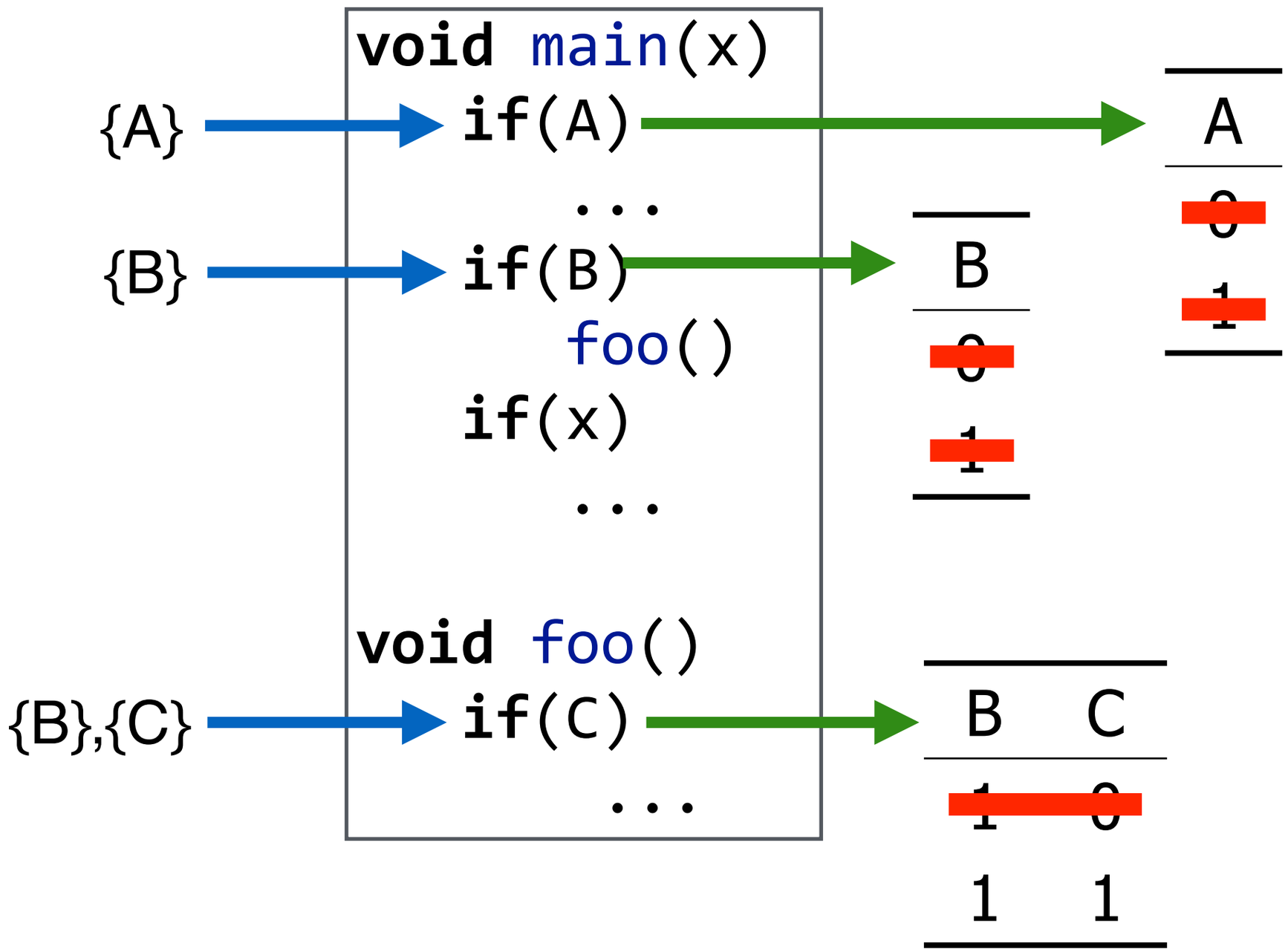
Derive Configurations to Explore

A	B	C
1	1	0



Pick Next Configuration to Execute

A	B	C
0	1	1



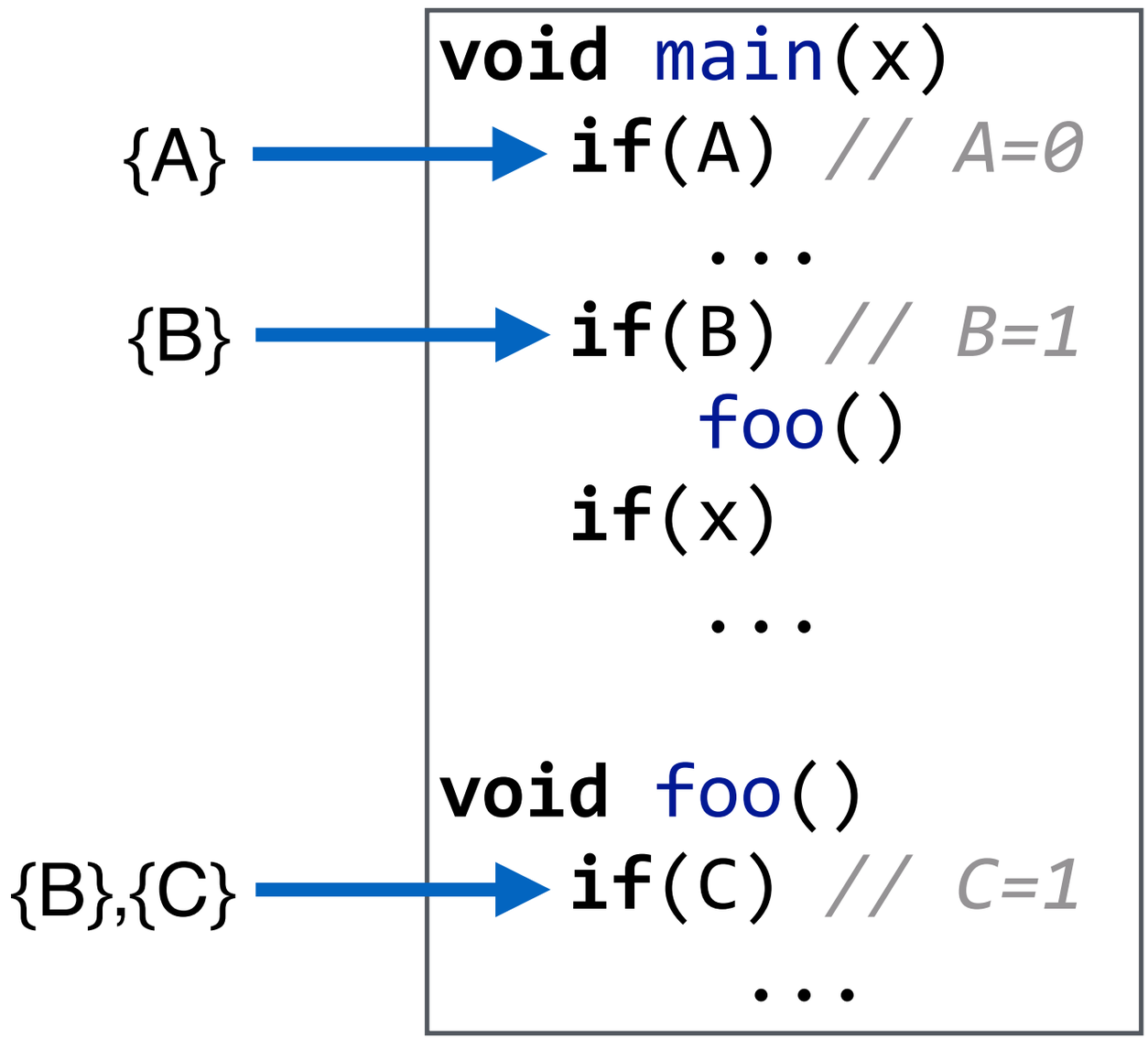
Dynamic Taint Analysis

A	B	C
0	1	1

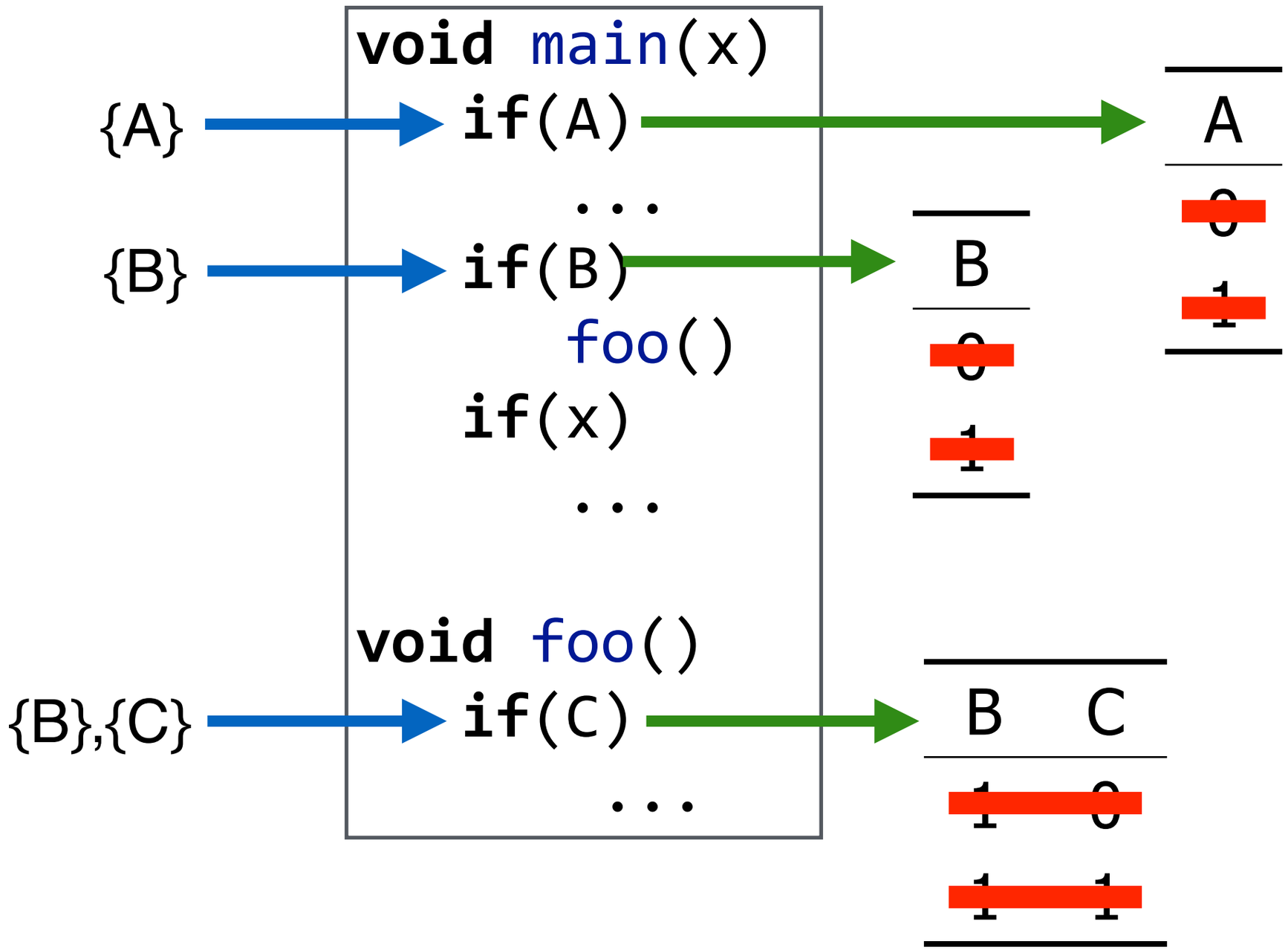
```
void main(x)
    if(A) // A=0
        ...
    if(B) // B=1
        foo()
    if(x)
        ...

void foo()
    if(C) // C=1
        ...
```

Dynamic Taint Analysis

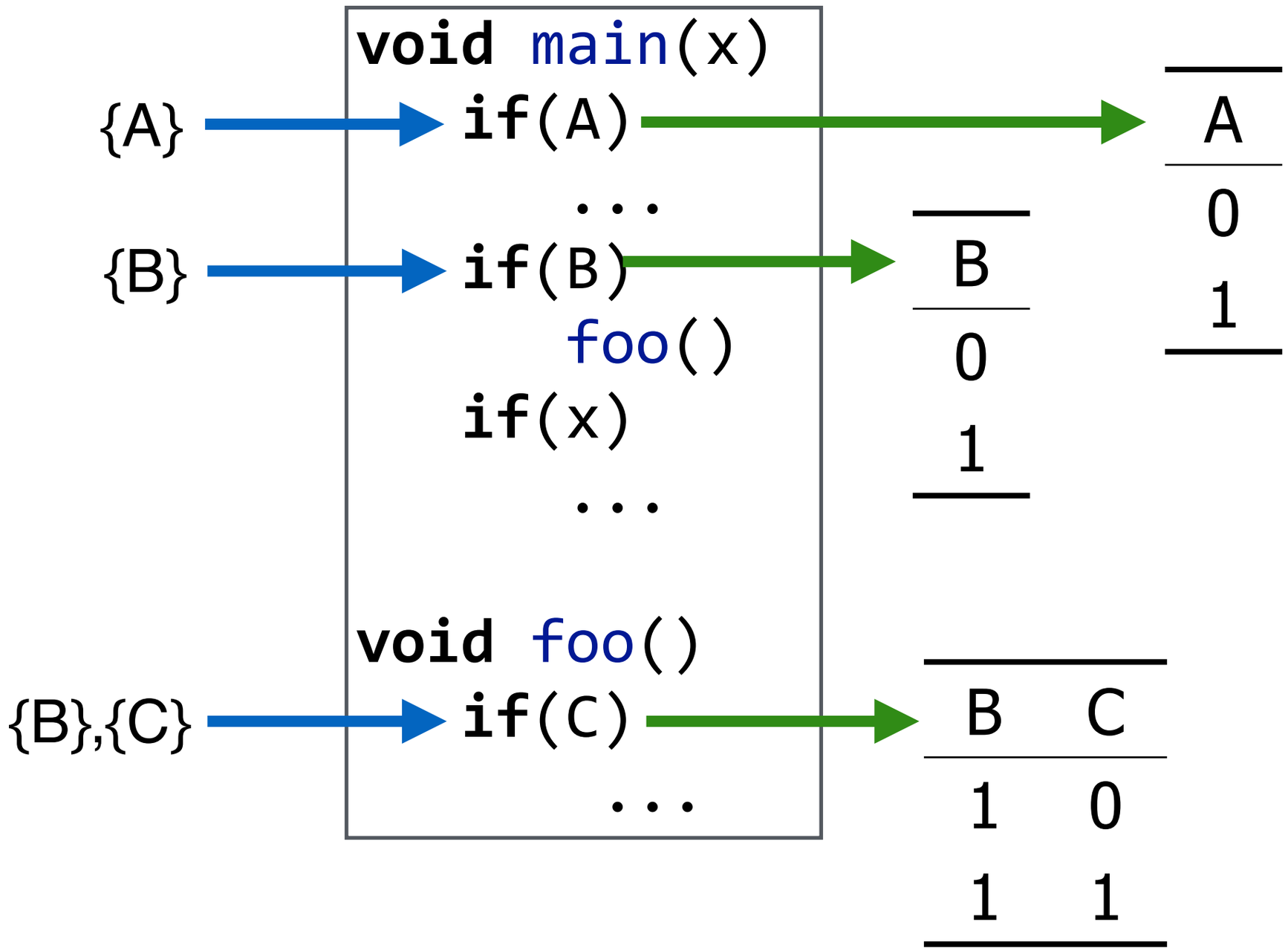
A	B	C
0	1	1



Pick Next Configuration to Execute



Find Configurations to Efficiently Explore Decisions



Limitations of Dynamic Taint Analysis

Hard to track implicit flows with unsound analysis

Tracking Implicit Flows

```
void main()
    x = 0
    if(A)
        x = 1
    if(x)
        ...
```

Tracking Implicit Flows

A
0

```
void main()
    x = 0
    if(A) // A=0
        x = 1
    if(x)
        ...
```

Tracking Implicit Flows

A
0

{A}

```
void main()
    x = 0
    if(A) // A=0
        x = 1
    if(x)
        ...
```

Tracking Implicit Flows

A
0

{A}

```
void main()
    x = 0
    if(A) // A=0
        x = 1
    if(x)
        ...
```

Missed information

Limitations of Dynamic Taint Analysis

Hard to track implicit flows with unsound analysis

Tooling overhead

Summary

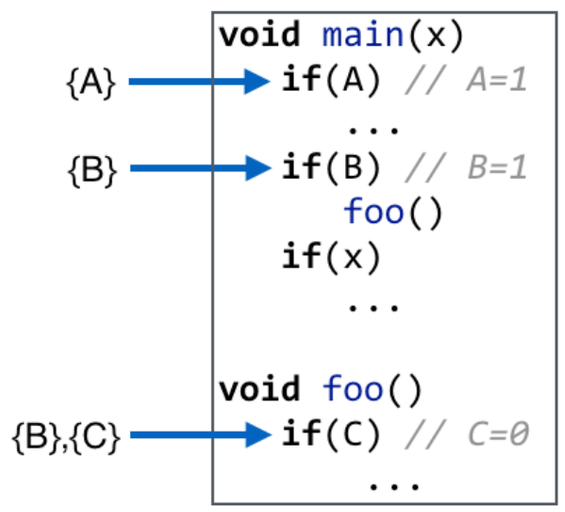
Dynamic taint analysis

Find how options influence control-flow decisions

Find configurations to efficiently explore decisions

Dynamic Taint Analysis

A	B	C
1	1	0



Find Configurations to Efficiently Explore Decisions