
NCS
NATIONAL CYBER SUMMIT
June 5-7, 2018 | Huntsville, Alabama

U Mad? Binary Analysis with the Angr Framework

Ben Denton, PhD. | DESE Research, Inc.
bdenton@dese.com | @b_denton

Intro

- What is binary analysis?
- What is angr? (An unapologetic oversimplification)
- Demos!

What is Binary Analysis?

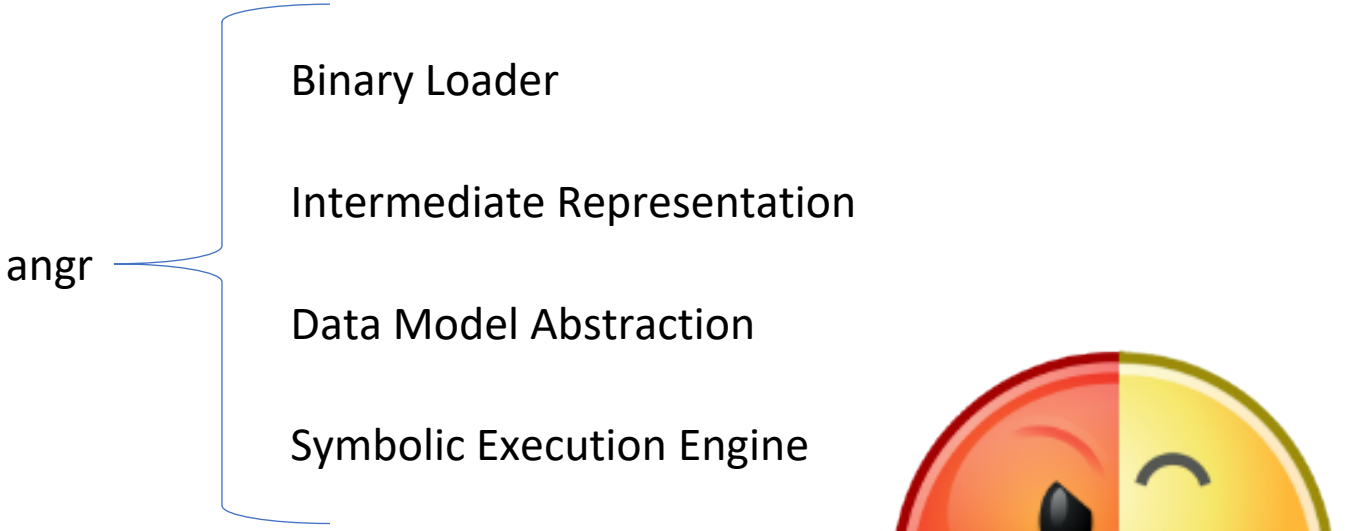
- Software bugs have taken down spaceships[1], caused nuclear centrifuges to spin out of control [2], and forced the recall of 100,000s of vehicles resulting in billions of dollars in damages [3].
- How can you find these bugs when source code is unavailable?

- **Reverse Engineering**, **Vulnerability Assessment,** and **Binary Analysis**
- Process: Disassemble, Triage, Understand, Analyze, Symbolize.

[1] *Ariane 5: Who Dunnit?* https://ieeexplore.ieee.org/document/589224/
[2] *Lessons from Stuxnet* https://ieeexplore.ieee.org/document/5742014/
[3] *A Case Study of Toyota Unintended Acceleration and Software Safety* https://users.ece.cmu.edu/~koopman/pubs/koopman14_toyota_ua_slides.pdf

What is angr?

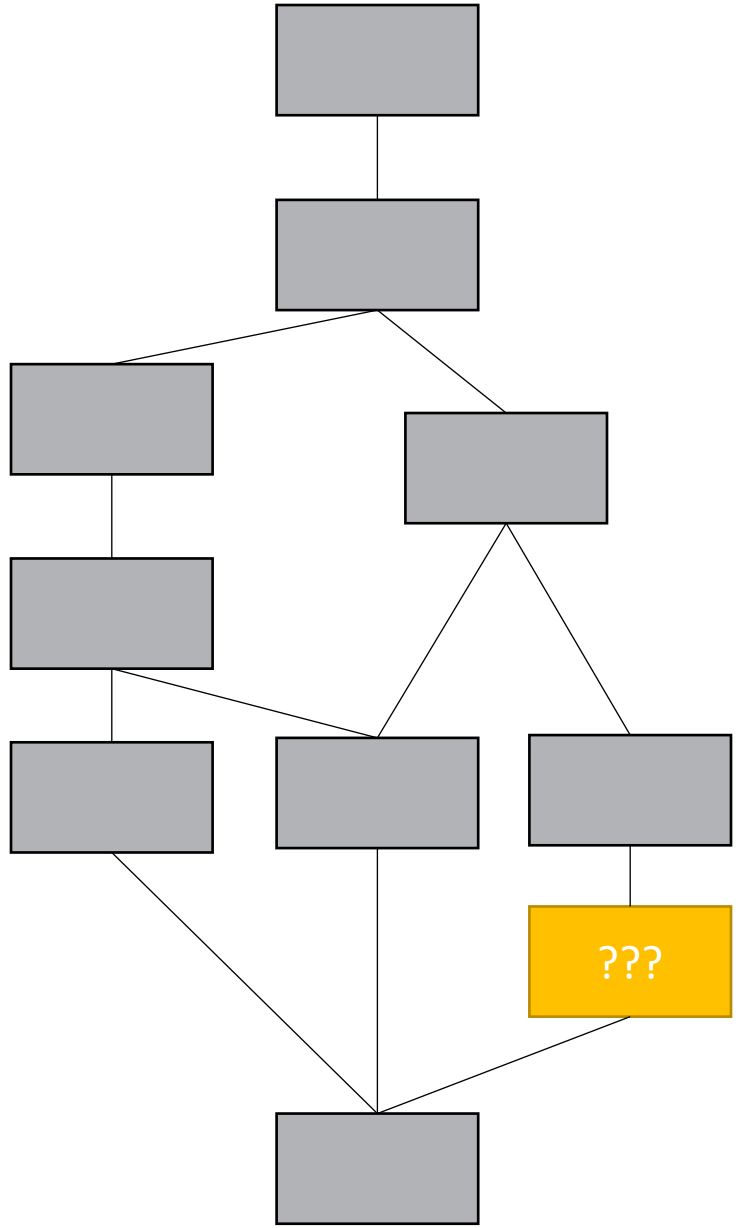
- Binary Analysis Framework developed by the University of California Santa Barbara since 2013.
- Features:
  - iPython accessible
  - Powerful analyses
  - Versatile
  - Open and expandable
  - Architecture “independent”



Software Analysis

"How do I trigger path x or condition y?"

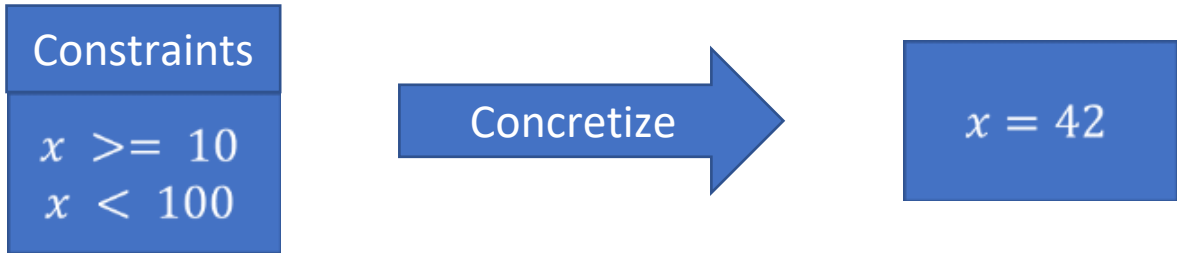
- Dynamic analysis
  - Input a? No. Input b? No. Input c? …
  - Based on concrete inputs to application
- Static analysis
  - "You can't"
  - "You might be able to…"
  - "IDK"
  - Based on various static techniques.



Symbolic Execution

"How do I trigger path x or condition y?"

1. Interpret the application.
2. Track "constraints" on variables.
3. When the required condition is triggered, "concretize" to obtain a possible input.



Symbolic Execution Example

```
def f (x, y):
  if (x > y):
    x = x + y
    y = x - y
    x = x - y
    if (x - y > 0):
      call g()
  return (x, y)
```

This function swaps the values of x and y when x > y.

The x – y > 0 statement is always false so the call is unreachable.

Source code is here but our techniques allow for the same analysis without source code.

Symbolic Execution Example

```
def f (x, y):
  if (x > y):
    x = x + y
    y = x - y
    x = x - y
    if (x - y > 0):
      call g()
  return (x, y)
```

Execute the program on *symbolic values*.

Symbolic Execution Example

```
def f (x, y):
  if (x > y):
    x = x + y
    y = x - y
    x = x - y
    if (x - y > 0):
      call g()
  return (x, y)
```

Execute the program on *symbolic values*.

Symbolic state maps variables to symbolic values.

x → A

y → B

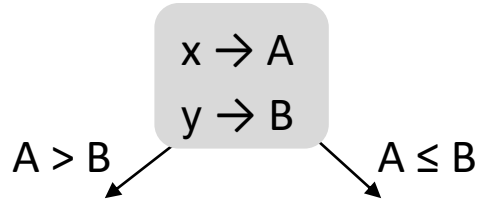
Symbolic Execution Example

```
def f (x, y):
  if (x > y):
    x = x + y
    y = x – y
    x = x – y
    if (x – y > 0):
      call g()
  return (x, y)
```

Execute the program on *symbolic values*.

Symbolic state maps variables to symbolic values.

Path condition is a quantifier-free formula over the symbolic inputs that encodes all branch decisions take so far.



Symbolic Execution Example

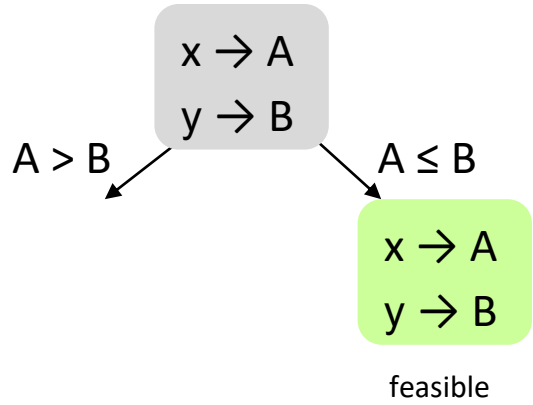
```
def f (x, y):
  if (x > y):
    x = x + y
    y = x – y
    x = x – y
    if (x – y > 0):
      call g()
  return (x, y)
```

Execute the program on *symbolic values*.

Symbolic state maps variables to symbolic values.

Path condition is a quantifier-free formula over the symbolic inputs that encodes all branch decisions take so far.

All paths in the program form its *execution tree*, in which some paths are *feasible* and some are *infeasible*.



Symbolic Execution Example

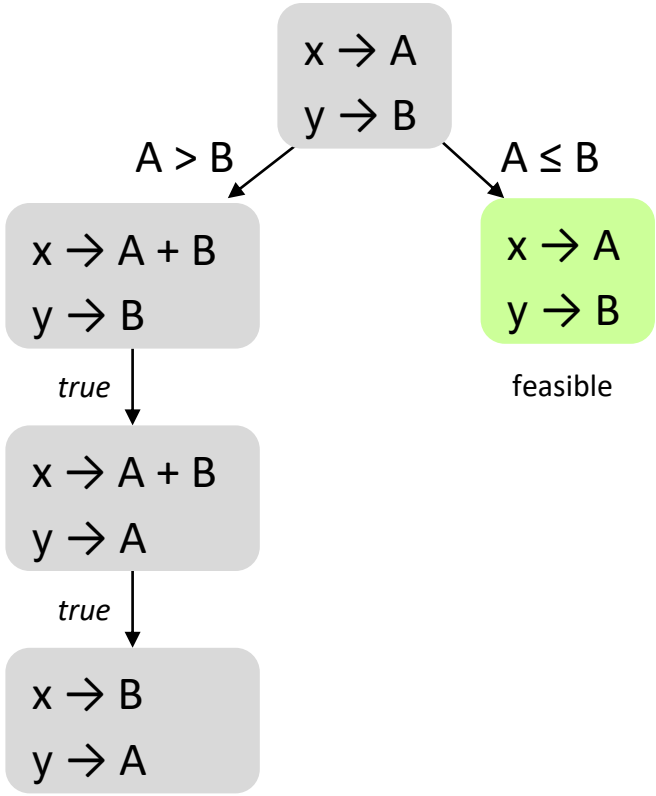
```
def f (x, y):
  if (x > y):
    x = x + y
    y = x - y
    x = x - y
    if (x - y > 0):
      call g()
  return (x, y)
```

Execute the program on *symbolic values*.

Symbolic state maps variables to symbolic values.

Path condition is a quantifier-free formula over the symbolic inputs that encodes all branch decisions take so far.

All paths in the program form its *execution tree*, in which some paths are *feasible* and some are *infeasible*.



Symbolic Execution Example

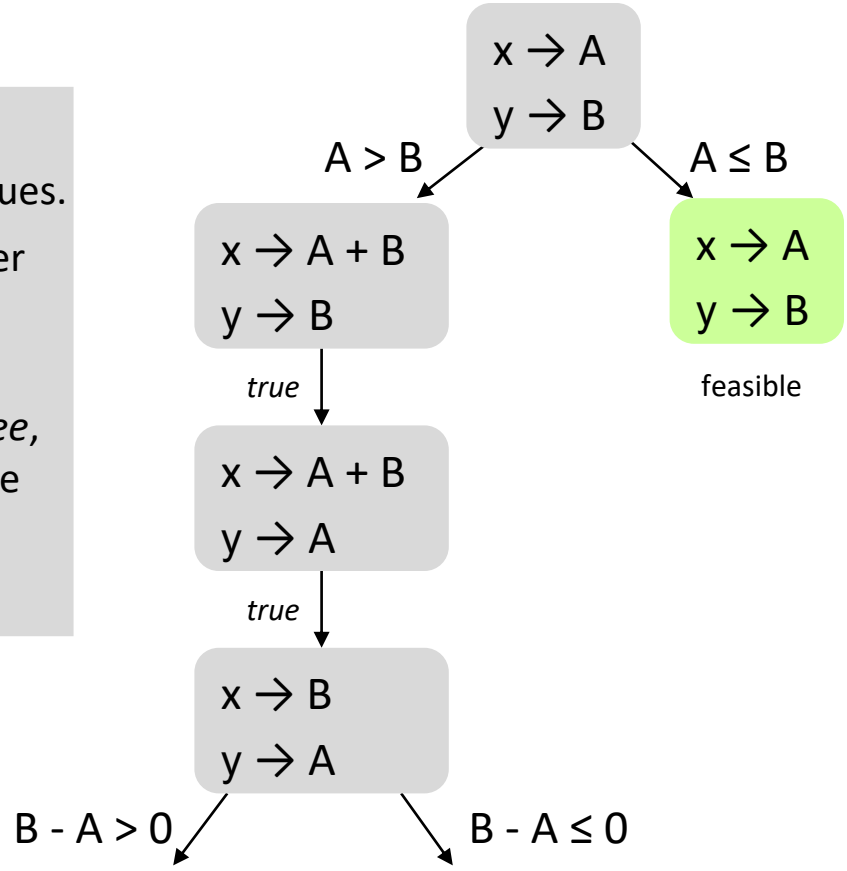
```
def f (x, y):
  if (x > y):
    x = x + y
    y = x – y
    x = x – y
    if (x – y > 0):
      call g()
  return (x, y)
```

Execute the program on *symbolic values*.

Symbolic state maps variables to symbolic values.

Path condition is a quantifier-free formula over the symbolic inputs that encodes all branch decisions take so far.

All paths in the program form its *execution tree*, in which some paths are *feasible* and some are *infeasible*.



Symbolic Execution Example

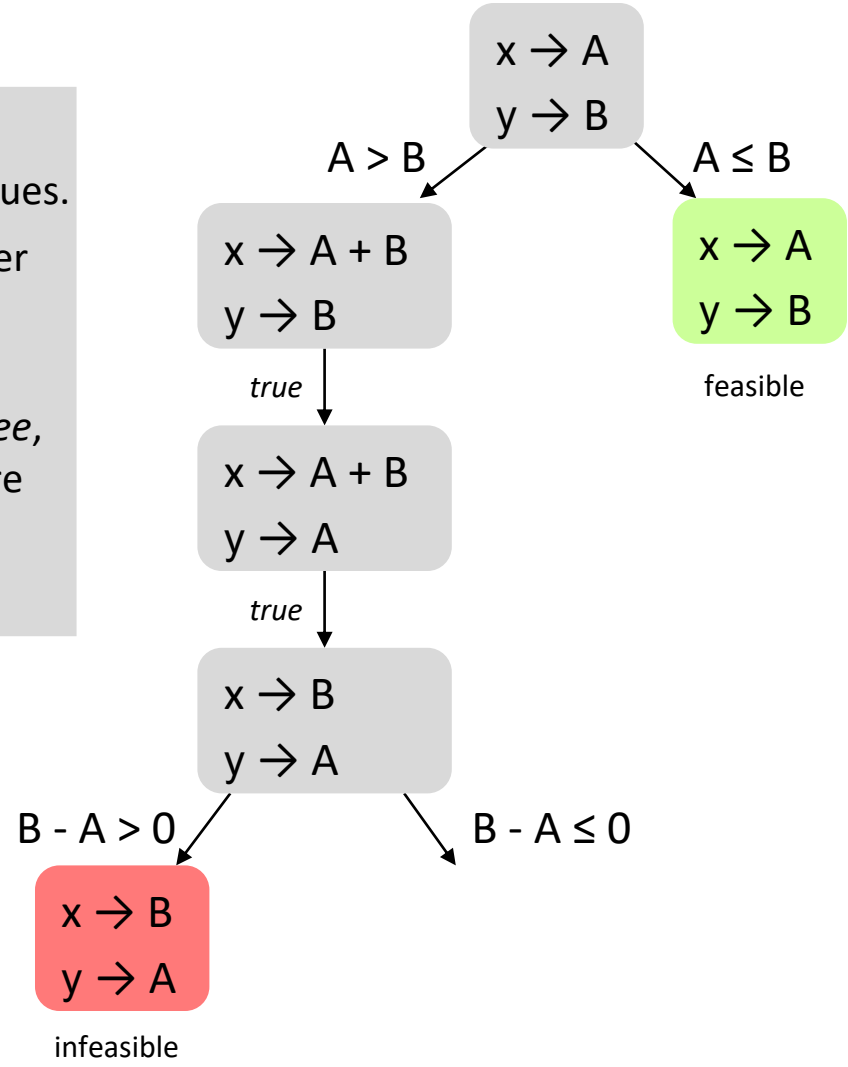
```
def f (x, y):
  if (x > y):
    x = x + y
    y = x – y
    x = x – y
    if (x – y > 0):
      call g()
  return (x, y)
```

Execute the program on *symbolic values*.

Symbolic state maps variables to symbolic values.

Path condition is a quantifier-free formula over the symbolic inputs that encodes all branch decisions take so far.

All paths in the program form its *execution tree*, in which some paths are *feasible* and some are *infeasible*.



Symbolic Execution Example

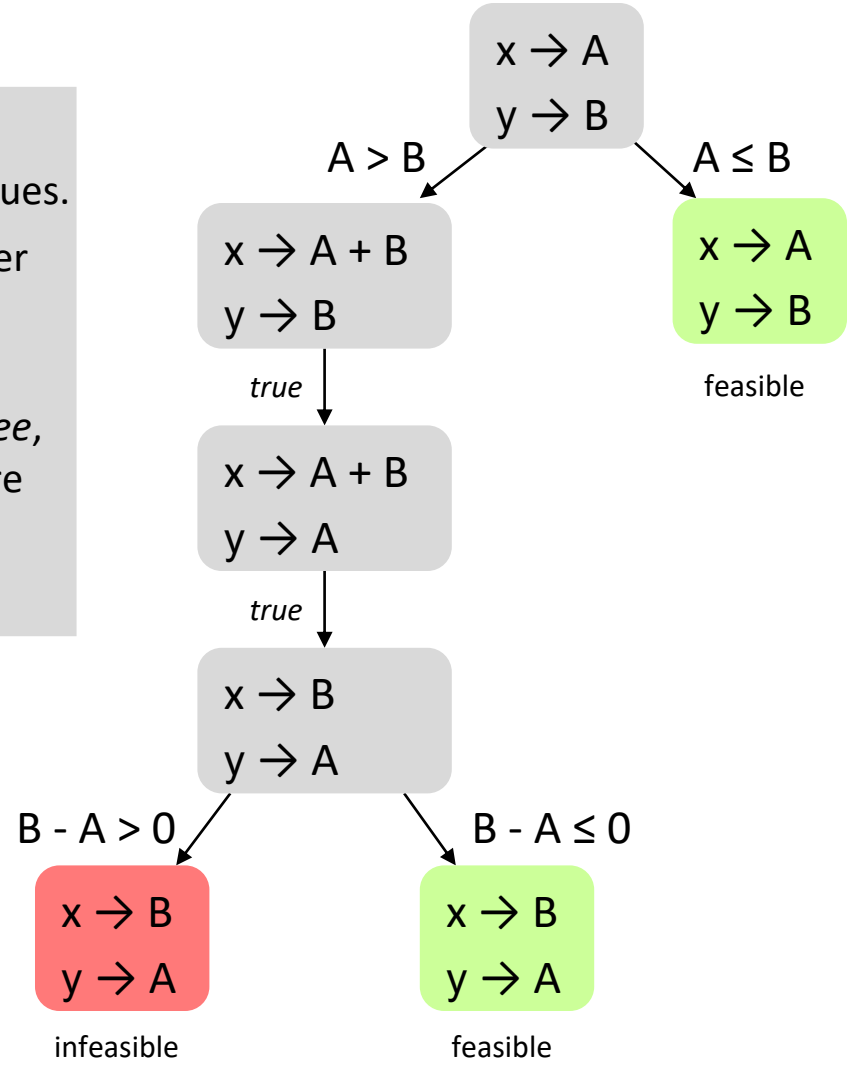
```
def f (x, y):
  if (x > y):
    x = x + y
    y = x - y
    x = x - y
    if (x - y > 0):
      call g()
  return (x, y)
```

Execute the program on *symbolic values*.

Symbolic state maps variables to symbolic values.

Path condition is a quantifier-free formula over the symbolic inputs that encodes all branch decisions take so far.

All paths in the program form its *execution tree*, in which some paths are *feasible* and some are *infeasible*.



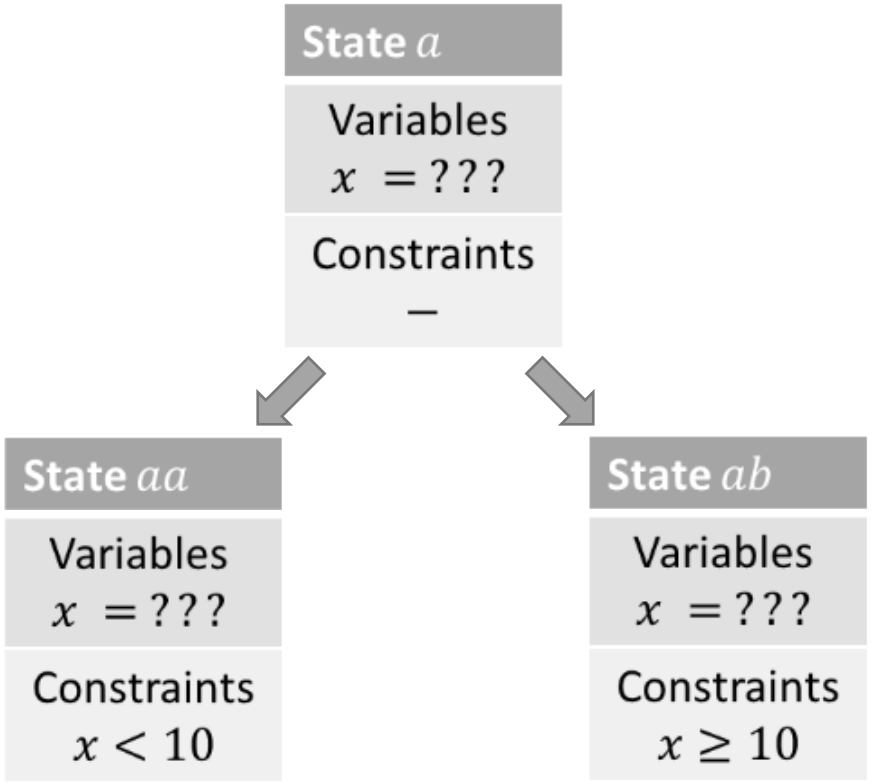
Symbolic Execution Example

```
x = int (input())
if x ≤ 10:
  if x < 100:
      print “Two!”
  else:
      print “Lots!”
else:
  print “One!”
```

State a
Variables $x = ???$
Constraints –

Symbolic Execution Example

```
x = int (input())
if x ≤ 10:
  if x < 100:
      print “Two!”
  else:
      print “Lots!”
else:
  print “One!”
```



Symbolic Execution Example

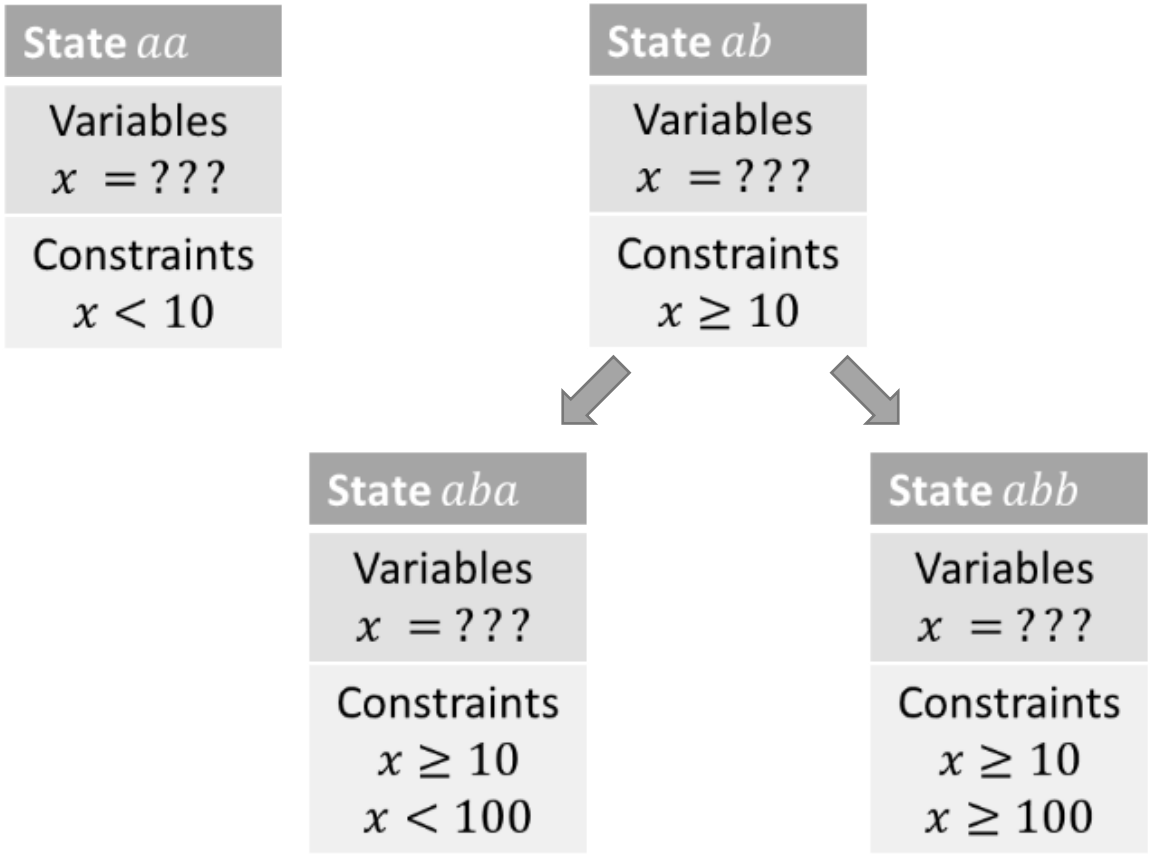
```
x = int (input())
if x ≤ 10:
  if x < 100:
      print “Two!”
  else:
      print “Lots!”
else:
  print “One!”
```

State aa
Variables $x = ???$
Constraints $x < 10$

State ab
Variables $x = ???$
Constraints $x \geq 10$

Symbolic Execution Example

```
x = int (input())
if x ≤ 10:
  if x < 100:
      print “Two!”
  else:
      print “Lots!”
else:
  print “One!”
```



Symbolic Execution Example

```
x = int (input())
if x ≤ 10:
  if x < 100:
      print “Two!”
  else:
      print “Lots!”
else:
  print “One!”
```

State aba

Variables

$x = ???$

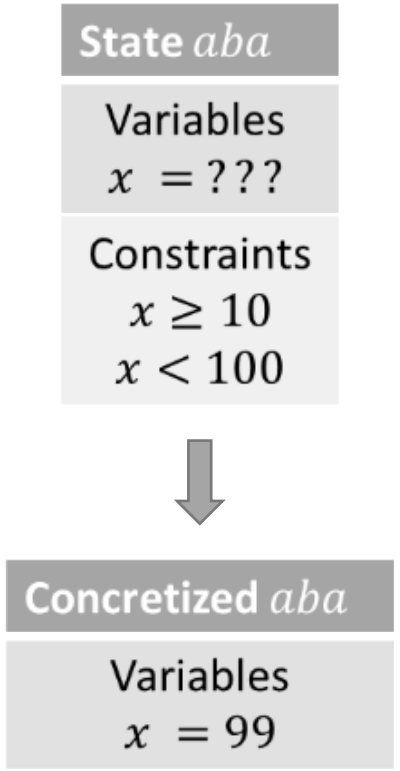
Constraints

$x \geq 10$

$x < 100$

Symbolic Execution Example

```
x = int (input())
if x ≤ 10:
  if x < 100:
      print “Two!”
  else:
      print “Lots!”
else:
  print “One!”
```



Demo: crackme

- Available at https://github.com/bendenton/2018_NCS

Demo: crackme2

- Available at https://github.com/bendenton/2018_NCS

Demo: crackme3

- Available at https://github.com/bendenton/2018_NCS

Questions?

bdenton@dese.com

https://github.com/bendenton/2018_NCS

https://www.linkedin.com/in/ben-denton/