CS 292C Computer-Aided Reasoning for Software

Lecture 2: Solver-Aided Programming I

Inspired by CSE507 from Emina Torlak

Yu Feng

Fall 2019

Summary of previous lecture

- Introducing the cast
- Ideas about final project
- Course structure

Ideas for final projects

- Detect vulnerable smart contracts without specs (MSR)
- Verify complex properties (LTL) in smart contracts (MSR)
- Detect malware in third-party libraries for Android (Google security)
- Interactive data visualization synthesis (UW, UT)
- Synthesize DoS attacks for Solidity compilers (Solidity)
- Super-optimization via reinforcement learning (Google Brain)
- Type-directed synthesis for polymorphism
- ……

Outline of this lecture

- The classical way for using solvers
- Solver-aided programming
- Rosette constructs

A classical way to use solvers

```
foo (int a) {
 x = 10;
 y = 5;
}
```

$$x = 10 \wedge y = 5$$

```
foo (int a) {
 if (a > 0)
  x = 10;
 else
  y = 5;
}
```

$$a > 0 \implies x = 10 \wedge a <= 0 \implies y = 5$$

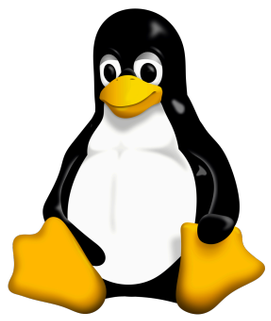
```
foo (int a) {
 if (a > 0)
  x = 10;
 else
  y = 5;
 assert y > 4
}
```

$$a > 0 \implies x = 10 \wedge a <= 0 \implies y = 5$$

$$\implies y > 4$$

A classical way to use solvers

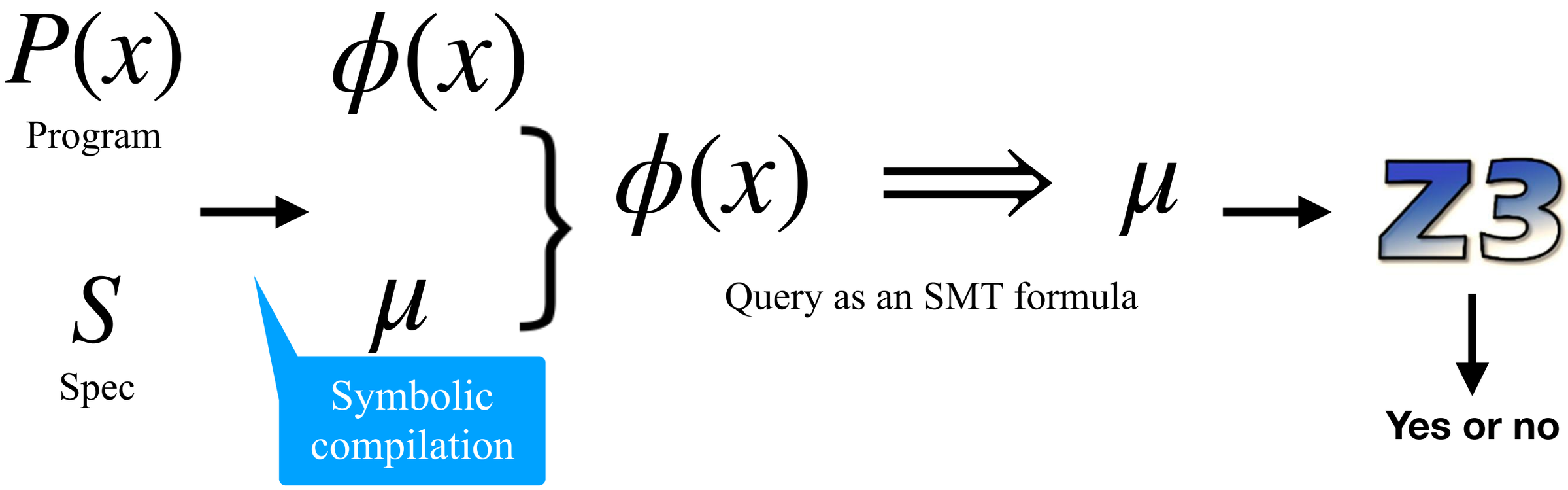
??



??

How to deal with complex systems?

A classical way to use solvers



Symbolic compilation can take years of effort!

A programming model that integrates solvers into the language, providing constructs for program verification, synthesis, and debugging.

Solver-aided programming

```
p(x) {
  v = 12
```

```
p(x) {
  v = ??
  …
}
assert safe(x, p(x))
```

Find an input on which the program fails.

Localize bad parts of the program.

Find values that repair the failing run.

Find code that repairs the program.

Solver-aided applications

Systems

SOSP'19, OSDI'18,
SOSP'17, OSDI'16

Blockchain

Browser engines

Biology

POPL'14

Education

Data science

PLDI'18, PLDI'17

Robotics

HPC

ASPLOS'16, OSDI'18

Gaming

Malware

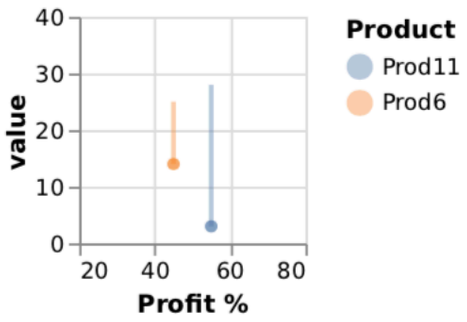
NDSS'17

Visualization

Solver-aided applications

Prod	Base	Inc	Profit
Prod1	61	70	132
Prod2	128	22	47
…	…	…	…

13x4



Line(Prod6,45,11,45,25)
Line(Prod11,55,25,55,28)
point(Prod6,45,11)
point(Prod11,55,25)

```
MultiLayer(
  Line(x=Profit,y=value,
       color=Product),
  Scatter(x=Profit,y=value,
       color=Product))
```

```
P1:
t=mutate(T,Top,Base+Inc)
t=gather(t,key=[Prod,Profit],
         value=[Base,Top])
t=select(t,[Profit,value,Product])
P2:
t=gather(T,[Prod,Profit],[Base])
t=select(t,[Profit,value,Product])
```

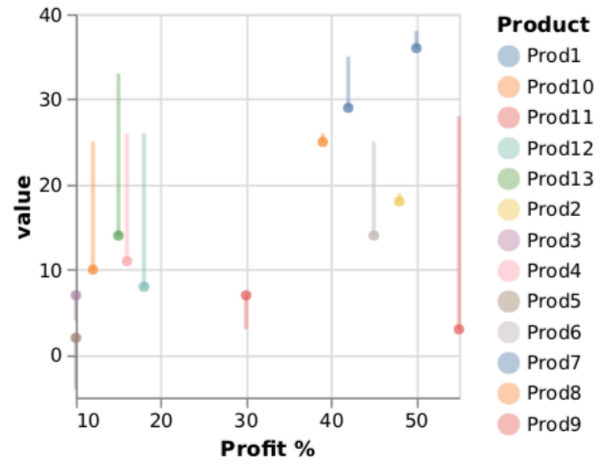
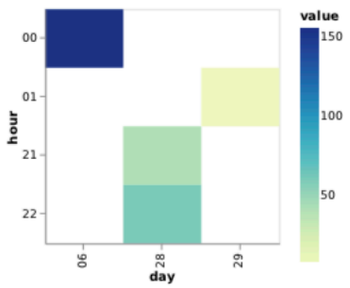


Fig. 16. Illustration of task #2.

DayHour	Site	Type	Value
06_00	e90e50	PC	22
06_00	e90e50fx	PC	8
06_01	e90e50	Mobile	8
…	…	…	…

1986x4



point(06,00,155)
point(28,21,42)
point(28,22,60)
point(29,01,8)

```
Scatter[rect](
    x=Day,y=Hour,
    color=sumValue)
```

```
t=group_by(T,[DayHour])
t=summarize(t,sum,Value)
t=separate(t,DayHour)
t=select(t,[Day,Hour,sumValue])
```

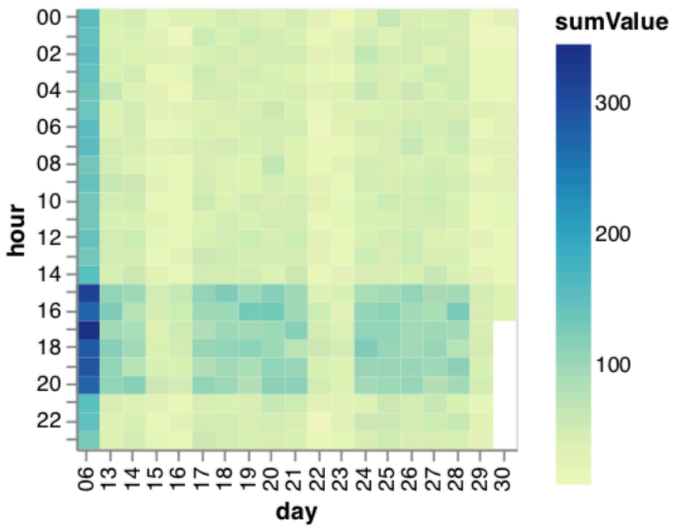
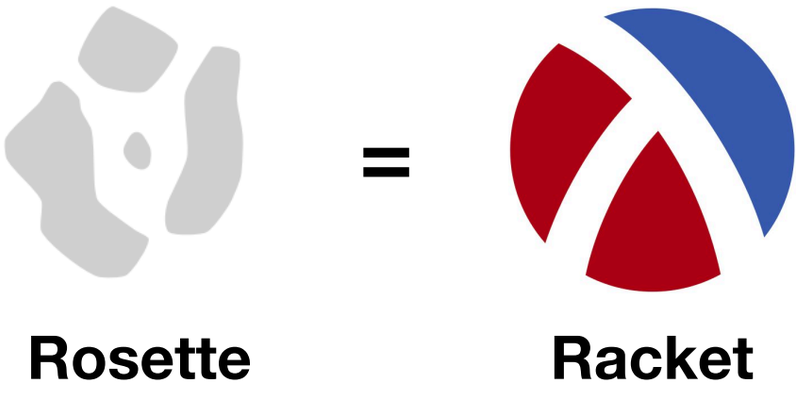


Fig. 17. Illustration of task #3.

Rosette constructs




```
(define-symbolic id type)
(define-symbolic* id type)
```

symbolic values

```
(assert expr)
```

assertions

```
(verify expr)
(debug [type ...+] expr)
(solve expr)
(synthesize
  #:forall expr
  #:guarantee expr)
```

queries

A Lightweight Symbolic Virtual Machine for Solver-Aided Host Languages. Emina Torlak and Rastislav Bodik. PLDI'14

Rosette constructs: verify

Search for a binding of symbolic constants to concrete values that violates at least one of the assertions

```
(define-symbolic id type)
(define-symbolic* id type)
```

symbolic values

```
(assert expr)
```

assertions

```
(verify expr)
(debug [type ...+] expr)
(solve expr)
(synthesize
  #:forall expr
  #:guarantee expr)
```

queries

```
(define (poly x)
 (+ (* x x x x) (* 6 x x x) (* 11 x x) (* 6 x)))

(define (factored x)
 (* x (+ x 1) (+ x 2) (+ x 2)))

(define (same p f x)
 (assert (= (p x) (f x))))

(define-symbolic i integer?)

(define cex (verify (same poly factored i)))
(evaluate i cex)
```

Rosette constructs: debugging

Searches for a minimal set of expressions that are responsible for the observed failure

```
(define-symbolic id type)
(define-symbolic* id type)
```

symbolic values

```
(assert expr)
```

assertions

```
(verify expr)
(debug [type ...+] expr)
(solve expr)
(synthesize
  #:forall expr
  #:guarantee expr)
```

queries

```
(define (poly x)
 (+ (* x x x x) (* 6 x x x) (* 11 x x) (* 6 x)))

(define (factored x)
 (* (+ x (??)) (+ x 1) (+ x (??)) (+ x (??))))

(define (same p f x)
 (assert (= (p x) (f x))))

(define-symbolic i integer?)

(define binding
    (synthesize #:forall (list i)
                #:guarantee (same poly factored i)))
```

Rosette constructs: synthesis

Search for a binding of symbolic constants to concrete values that satisfy the assertions

```
(define-symbolic id type)
(define-symbolic* id type)
```

symbolic values

```
(assert expr)
```

assertions

```
(verify expr)
(debug [type ...+] expr)
(solve expr)
(synthesize
  #:forall expr
  #:guarantee expr)
```

queries

```
(define (poly x)
 (+ (* x x x x) (* 6 x x x) (* 11 x x) (* 6 x)))

(define (factored x)
 (* (+ x (??)) (+ x 1) (+ x (??)) (+ x (??))))

(define (same p f x)
 (assert (= (p x) (f x))))

(define-symbolic i integer?)

(define binding
    (synthesize #:forall (list i)
                #:guarantee (same poly factored i)))
```

Rosette constructs: angelic execution

Searches for a minimal set of expressions that are responsible for the observed failure

```
(define-symbolic id type)
(define-symbolic* id type)
```

symbolic values

```
(assert expr)
```

assertions

```
(verify expr)
(debug [type ...+] expr)
(solve expr)
(synthesize
  #:forall expr
  #:guarantee expr)
```

queries

```
(define-symbolic x y integer?)

(define (poly x)
 (+ (* x x x x) (* 6 x x x) (* 11 x x) (* 6 x)))

(define sol
    (solve (begin (assert (not (= x y)))
                  (assert (< (abs x) 10))
                  (assert (< (abs y) 10))
                  (assert (not (= (poly x) 0)))
                  (assert (= (poly x) (poly y))))))
```

TODOs by next lecture

- Paper reading list is out
- Install Rosette and Neo
  - Install Rosette: https://docs.racket-lang.org/rosette-guide/ch_getting-started.html
  - Install Neo: https://github.com/fredfeng/Trinity
- Start to look for partners for your final project!