CS 162 Programming languages

Lecture 13: Solver-Aided Programming II

Slides are based on Emina Torlak's tutorial at CAV'19

Yu Feng

Winter 2020

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

Solver-aided programming

```
p(x) {
  v = 12
}
assert safe(x, p(x))
```

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

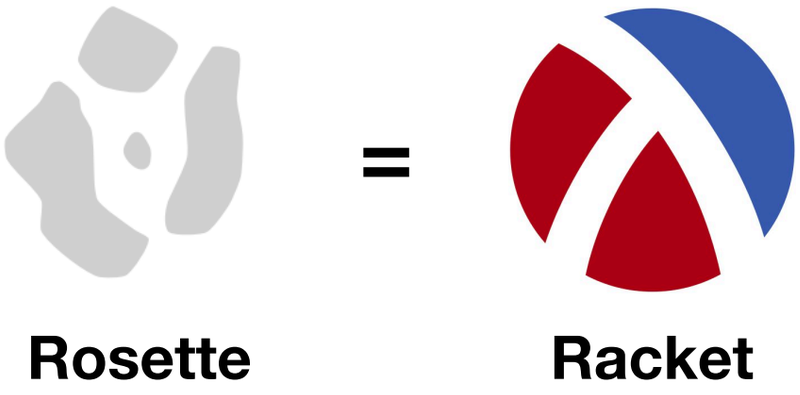
Find an input on which the program fails.

Localize bad parts of the program.

Find values that repair the failing run.

Find code that repairs the program.

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

A tiny example DSL

```
def bvmax(r0, r1) :
    r2 = bvsge(r0, r1)
    r3 = bvneg(r2)
    r4 = bvxor(r0, r2)
    r5 = bvand(r3, r4)
    r6 = bvxor(r1, r5)
return r6
```

We want to test, verify, debug, and synthesize programs in the BV SDSL.

BV: A tiny assembly-like language for writing fast, low-level library functions.

1. interpreter [10 LOC]
2. verifier [free]
3. debugger [free]
4. synthesizer [free]

A tiny example DSL

RISETTE

```
def bvmax(r0, r1) :
   r2 = bvsge(r0, r1)
   r3 = bvneg(r2)
   r4 = bvxor(r0, r2)
   r5 = bvand(r3, r4)
   r6 = bvxor(r1, r5)
return r6
```

parse

```
(define bvmax
   `((2 bvsge 0 1)
    (3 bvneg 2)
    (4 bvxor 0 2)
    (5 bvand 3 4)
    (6 bvxor 1 5)))
```

`(out opcode in ...)`

A tiny example DSL

```
def bvmax(r0, r1) :
   r2 = bvsge(r0, r1)
   r3 = bvneg(r2)
   r4 = bvxor(r0, r2)
   r5 = bvand(r3, r4)
   r6 = bvxor(r1, r5)
   return r6

> bvmax(-2,-1)
```

interpret

RⓄSETTE

```
(define bvmax
    `((2 bvsge 0 1)
      (3 bvneg 2)
      (4 bvxor 0 2)
      (5 bvand 3 4)
      (6 bvxor 1 5)))
```

```
(-2,-1)
```

```
(define (interpret prog inputs)
  (make-registers prog inputs)
  (for ([stmt prog])
    (match stmt
      [(list out opcode in ...)
       (define op (eval opcode))
       (define args (map load in))
       (store out (apply op args))]))
  (load (last)))
```

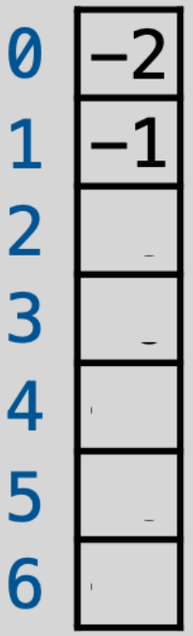
A tiny example DSL

```
def bvmax(r0, r1) :
   r2 = bvsge(r0, r1)
   r3 = bvneg(r2)
   r4 = bvxor(r0, r2)
   r5 = bvand(r3, r4)
   r6 = bvxor(r1, r5)
   return r6

> bvmax(-2,-1)
```

interpret

ROSETTE

```
(define bvmax
    `((2 bvsge 0 1)
      (3 bvneg 2)
      (4 bvxor 0 2)
      (5 bvand 3 4)
      (6 bvxor 1 5)))
```




```
(define (interpret prog inputs)
  (make-registers prog inputs)
  (for ([stmt prog])
    (match stmt
      [(list out opcode in ...)
       (define op (eval opcode))
       (define args (map load in))
       (store out (apply op args))]))
  (load (last)))
```

A tiny example DSL

```
def bvmax(r0, r1) :
   r2 = bvsge(r0, r1)
   r3 = bvneg(r2)
   r4 = bvxor(r0, r2)
   r5 = bvand(r3, r4)
   r6 = bvxor(r1, r5)
   return r6

(define (max x y)
   (if (bvsge x y) x y))

> verify(bvmax,max)
```

query

ROSETTE

Creates two fresh symbolic values of type 32-bit integer and binds them to the variables x and y.

```
(define-symbolic x y int32?)
(define in (list x y))
(verify
  (assert (equal?
    (interpret bvmax in)
    (interpret max in))))
```

(verify expr) searches for a concrete interpretation of symbolic values that causes

Symbolic values can be used just like concrete values of the same type.

A tiny example DSL

```
def bvmax(r0, r1) :
  r2 = bvsge(r0, r1)
  r3 = bvneg(r2)
  r4 = bvxor(r0, r2)
  r5 = bvand(r3, r4)
  r6 = bvxor(r1, r5)
  return r6

(define (max x y)
  (if (bvsge x y) x y))

> verify(bvmax,max)
[0,-2]
> bvmax(0,-2)
-1
```

query

ROSETTE

```
(define-symbolic x y int32?)
(define in (list x y))
(verify
  (assert (equal?
    (interpret bvmax in)
    (interpret max in))))
```

A tiny example DSL

```
def bvmax(r0, r1) :
   r2 = bvsge(r0, r1)
   r3 = bvneg(r2)
   r4 = bvxor(r0, r2)
   r5 = bvand(r3, r4)
   r6 = bvxor(r2, r5)
   return r6

> debug(bvmax,max,[0,-2])
```

query

RⓄSETTE

```
(define in (list (int32 0) (int32 -2)))
(debug [register?]
  (assert (equal?
           (interpret bvmax in)
           (interpret max in))))
```

A tiny example DSL

```
def bvmax(r0, r1) :
   r2 = bvsge(r0, r1)
   r3 = bvneg(r2)
   r4 = bvxor(r0, r2)
   r5 = bvand(r3, r4)
   r6 = bvxor(r2, r5)
   return r6

> debug(bvmax,max,[0,-2])
```

query

```
(define in (list (int32 0) (int32 -2)))
(debug [register?]
  (assert (equal?
           (interpret bvmax in)
           (interpret max in))))
```

A tiny example DSL

```
def bvmax(r0, r1) :
   r2 = bvsge(r0, r1)
   r3 = bvneg(r2)
   r4 = bvxor(??, ??)
   r5 = bvand(r3, ??)
   r6 = bvxor(??, ??)
   return r6

> synthesize(bvmax,max)
```

query

ROSETTE

```
(define-symbolic x y int32?)
(define in (list x y))
(synthesize
  #:forall in
  #:guarantee
  (assert (equal?
    (interpret bvmax in)
    (interpret max in)))))
```

A tiny example DSL

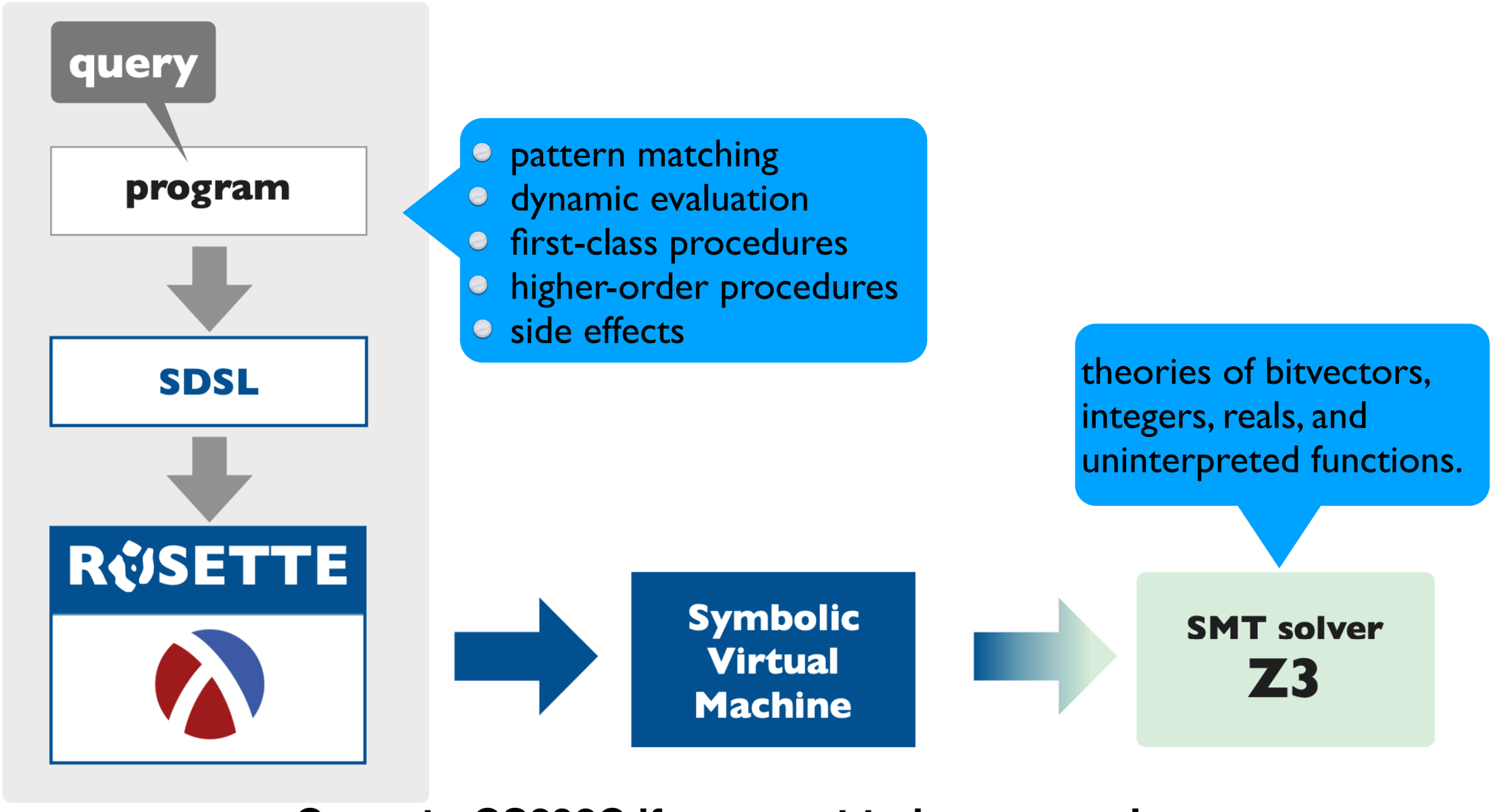
```
def bvmax(r0, r1) :
   r2 = bvsge(r0, r1)
   r3 = bvneg(r2)
   r4 = bvxor(r0, r1)
   r5 = bvand(r3, r4)
   r6 = bvxor(r1, r5)
   return r6

> synthesize(bvmax,max)
```

query

ROSETTE

```
(define-symbolic x y int32?)
(define in (list x y))
(synthesize
  #:forall in
  #:guarantee
  (assert (equal?
    (interpret bvmax in)
    (interpret max in)))))
```

How does it work?



Come to CS292C if you want to know more!