Synthesis and Verification for All

Emina Torlak

University of Washington
emina@cs.washington.edu
homes.cs.washington.edu/~emina/
Hello
my name is

Emina
a programmable programming language
Racket

a programmable programming language
a programmable programming language
a solver-aided programming language
a little programming for everyone
A little programming for everyone

Every knowledge worker wants to program …
A little programming for everyone

Every knowledge worker wants to program …

› spreadsheet data manipulation
A little programming for everyone

Every knowledge worker wants to program …

- spreadsheet data manipulation
- models of cell fates
A little programming for everyone

Every knowledge worker wants to program ...

- spreadsheet data manipulation
- models of cell fates
- cache coherence protocols
- memory models
A little programming for everyone

Every knowledge worker wants to program …

- spreadsheet data manipulation
- models of cell fates
- cache coherence protocols
- memory models
A little programming for everyone

Every knowledge worker wants to program …

- spreadsheet data manipulation [Flashfill, POPL’11]
- models of cell fates [SBL, POPL’13]
- cache coherence protocols [Transit, PLDI’13]
- memory models [MemSAT, PLDI’10]
A little programming for everyone

We all want to build programs …

› spreadsheet data manipulation
› models of cell fates
› cache coherence protocols
› memory models

less time

less expertise

solver-aided languages

hardware designer

biologist

social scientist
solver-aided tools
solver-aided tools, languages
solver-aided tools, languages, and applications
solver-aided tools
Programming ...

specification

P(x) {
 ... 
 ... 
}

12
Programming ...

test case

P(x) {
  ...
  ...
}
assert safe(P(2))
Programming with a solver-aided tool

P(x) {
    ...
    ...
} 
assert safe(P(2))
Programming with a solver-aided tool

Find an input on which the program fails.

∃x . ¬ safe(P(x))

SAT/SMT solver

CBMC [Kroening et al., DAC’03]
Dafny [Leino, LPAR’10]
Miniatur [Vaziri et al., FSE’07]
Klee [Cadar et al., OSDI’08]
Programming with a solver-aided tool

Find an input on which the program fails.
Localize bad parts of the program.

P(x) {
  v = x + 2
  ...
} assert safe(P(x))

x = 42 ∧ safe(P(x))

BugAssist [Jose & Majumdar, PLDI'11]
Programming with a solver-aided tool

Find an input on which the program fails. 
Localize bad parts of the program. 
Find values that repair the failing run.

P(x) {
    v = choose()
    ...
} 
assert safe(P(x))

∃v . safe(P(42, v))

Kaplan [Koksal et al, POPL’12]
PBNJ [Samimi et al., ECOOP’10]
Squander [Milicevic et al., ICSE’11]
Programming with a solver-aided tool

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.

x - 2

P(x) {
  v = ??
  ...
} assert safe(P(x))

∃e . ∀x . safe(P_e(x))

Sketch [Solar-Lezama et al., ASPLOS’06]
Comfusy [Kuncak et al., CAV’10]
The standard (hard) way to build a tool

```plaintext
P(x) {
    ...
    ...
}
assert safe(P(x))
```

expertise in PL, FM, SE

translator

SAT/SMT solver
A new, easy way to build tools

P(x) {
    ...
    ...
}  
assert safe(P(x))
A new, easy way to build tools

Implement a language for an application domain, get the tools for free!

P(x) {
  ...
  ...
}  
assert safe(P(x))

verify  
debug  
solve  
synth

an interpreter or a library
A new, easy way to build tools

Implement a language for an application domain, get the tools for free!

P(x) {
  ...
  ...
} assert safe(P(x))
A new, easy way to build tools

```latex
P(x) \{
  ...
  ...
\}
assert safe(P(x))
```

Implement a language for an application domain, get the tools for free!

Hard technical challenge: how to efficiently translate a program and its interpreter?

[Torlak & Bodik, PLDI'14, Onward'13]
solver-aided languages
Layers of languages

- **domain-specific language (DSL)**

  - library
  - interpreter

- **host language**

  A formal language that is specialized to a particular application domain and often limited in capability.

  A high-level language for implementing DSLs, usually with meta-programming features.
Layers of languages

- **domain-specific language (DSL)**
  - library
  - interpreter

- **host language**

- **artificial intelligence**
  - Church, BLOG

- **databases**
  - SQL, Datalog

- **hardware design**
  - Bluespec, Chisel, Verilog, VHDL

- **math and statistics**
  - Eigen, Matlab, R

- **layout and visualization**
  - LaTex, dot, dygraphs, D3

- **Scala, Racket, JavaScript**
Layers of languages

domain-specific language (DSL)

library

interpreter

host language

C = A * B

[associativity]

C / Java

for (i = 0; i < n; i++)
    for (j = 0; j < m; j++)
        for (k = 0; k < p; k++)
            C[i][k] += A[i][j] * B[j][k]
Layers of solver-aided languages

- **solver-aided domain-specific language (SDSL)**
  - library
  - interpreter
- **solver-aided host language**
- **symbolic virtual machine**
Layers of solver-aided languages

- solver-aided domain-specific language (SDSL)
- library
- interpreter
- solver-aided host language
- symbolic virtual machine

[Torlak & Bodik, Onward’13, PLDI’14]
Layers of solver-aided languages

solver-aided domain-specific language (SDSL)

library interpreter

solver-aided host language

symbolic virtual machine

spatial programming
  Chlorophyll
intelligent tutoring
  RuleSynth
memory models
  MemSynth
optimal synthesis
  Synapse
radiotherapy controllers
  Neutrons
BGP router configurations
  BagPipe

[Rorlak & Bodik, Onward’13, PLDI’14]
Anatomy of a solver-aided host language

Racket
Anatomy of a solver-aided host language

(define-symbolic id type)
(assert expr)
(verify expr)
(debug [expr] expr)
(solve expr)
(synthesize [expr] expr)
A tiny example SDSL

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

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

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

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

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

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 SDSL:  \texttt{ROSETTE}

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

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

A tiny example SDSL:

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

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

```
(define bvmax
  `((2 bvge 0 1)
    (3 bvneg 2)
    (4 bvxor 0 2)
    (5 bvand 3 4)
    (6 bvxor 1 5)))
```
def bvmax(r0, r1):
    r2 = bvge(r0, r1)
    r3 = bvneg(r2)
    r4 = bvxor(r0, r2)
    r5 = bvand(r3, r4)
    r6 = bvxor(r1, r5)
    return r6

> bvmax(-2, -1)  

A tiny example SDSL:

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

(out opcode in ...)
```
A tiny example SDSL:

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

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

interpreted:

```scheme
(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 SDSL:

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

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

interpreted:

```scheme
(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 SDSL:

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

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

interpreted:

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

> bvmax(-2, -1)
A tiny example SDSL:

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

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

```scheme
(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)))
```

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

```scheme
0 -2
1 -1
2
3
4
5
6
```
def bvmax(r0, r1):
    r2 = bvge(r0, r1)
    r3 = bvneg(r2)
    r4 = bvxor(r0, r2)
    r5 = bvand(r3, r4)
    r6 = bvxor(r1, r5)
    return r6

> bvmax(-2, -1)

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

0
1
2
3
4
5
6

(load (last))

RÔSETTE

A tiny example SDSL:
A tiny example SDSL:

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

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

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

A tiny example SDSL:
def bvmax(r0, r1):
    r2 = bvge(r0, r1)
    r3 = bvneg(r2)
    r4 = bvxor(r0, r2)
    r5 = bvand(r3, r4)
    r6 = bvxor(r1, r5)
    return r6

> bvmax(-2, -1)
def bvmax(r0, r1):
    r2 = bvge(r0, r1)
    r3 = bvneg(r2)
    r4 = bvxor(r0, r2)
    r5 = bvand(r3, r4)
    r6 = bvxor(r1, r5)
    return r6

> bvmax(-2, -1)

(define bvmax
  `(((2 bvge 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 SDSL:

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

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

```Scheme
(define bvmax
  (define (interpret prog inputs)
    (define (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)))

(define (interpret prog inputs)
  (define (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)))
```

22
A tiny example SDSL:

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

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

**ROSETTE**

```scheme
(define bvmax
  `((2 bvge 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)))

(pattern matching
  dynamic evaluation
  first-class & higher-order procedures
  side effects)
```
A tiny example SDSL: \textbf{ROSETTE}

\begin{verbatim}
def bvm\text{ax}(r0, r1):
    r2 = bvge(r0, r1)
    r3 = bv\text{neg}(r2)
    r4 = bv\text{xor}(r0, r2)
    r5 = bv\text{and}(r3, r4)
    r6 = bv\text{xor}(r1, r5)
    return r6

> verify(bv\text{max}, \text{max})
\end{verbatim}
A tiny example SDSL:

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

> verify(bvmax, max)
```

Creates two fresh symbolic constants of type number and binds them to variables n0 and n1.

```
(define-symbolic n0 n1 integer?)
(define inputs (list n0 n1))
(verify
  (assert (= (interpret bvmax inputs)
             (interpret max inputs))))
```
A tiny example SDSL:

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

> verify(bvmax, max)
```

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

```scheme
(define-symbolic n0 n1 integer?)
(define inputs (list n0 n1))
(verify
 (assert (= (interpret bvmax inputs) (interpret max inputs)))))
```
A tiny example SDSL:

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

> verify(bvmax, max)
(0, -2)
```

(verify expr) searches for a concrete interpretation of symbolic constants that causes expr to fail.
A tiny example SDSL:

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

> verify(bvmax, max)
(0, -2)

> bvmax(0, -2)
-1
```

(\text{define-symbolic} \, n0 \, n1 \, \text{integer}?)
(\text{define} \, \text{inputs} \, (\text{list} \, n0 \, n1))
(\text{verify}
  \begin{align*}
    & (\text{assert} \, (= \, (\text{interpret} \, \text{bvmax} \, \text{inputs}) \, (\text{interpret} \, \text{max} \, \text{inputs})))
  \end{align*}
)
A tiny example SDSL:

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

```scheme
(define inputs (list 0 -2))
(debug [input-register?]
  (assert (= (interpret bvmax inputs)
             (interpret max inputs))))
```
def bvmax(r0, r1):
    r2 = bvge(r0, r1)
    r3 = bvneg(r2)
    r4 = bvxor(r0, r2)
    r5 = bvand(r3, r4)
    r6 = bvxor(r1, r5)
    return r6

> debug(bvmax, max, (0, -2))

(define inputs (list 0 -2))
(debug [input-register?]
    (assert (= (interpret bvmax inputs)
               (interpret max inputs)))))
A tiny example SDSL:

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

> synthesize(bvmax, max)
```

(\texttt{define-symbolic} n0 n1 integer?)
(\texttt{define} inputs (list n0 n1))
(\texttt{synthesize} [inputs]
  (\texttt{assert} (= (interpret bvmax inputs)
                     (interpret max inputs))))
A tiny example SDSL:

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

```
> synthesize(bvmax, max)
```

```
(define-symbolic n0 n1 integer?)
(define inputs (list n0 n1))
(synthesize [inputs]
  (assert (= (interpret bvmax inputs)
             (interpret max inputs)))))
```
symbolic virtual machine (SVM)
How it all works: a big picture view

[Torlak & Bodik, Onward'13]

[Torlak & Bodik, PLDI'14]
How it all works: a big picture view

- **ROSSETTE**
- **program** → **SDSL**
- **symbolic virtual machine** → **solver Z3**

[Torlak & Bodik, Onward’13]

[Torlak & Bodik, PLDI’14]
How it all works: a big picture view

- pattern matching
- dynamic evaluation
- first-class procedures
- higher-order procedures
- side effects
- macros

theories of bitvectors, integers, reals, and uninterpreted functions

[Torlak & Bodik, Onward’13]

[Torlak & Bodik, PLDI’14]
Translation to constraints by example

\[ \text{solve: } ps = () \]

\[ \text{for } v \in vs: \]
\[ \text{if } v > 0: \]
\[ ps = \text{insert}(v, ps) \]

\[ \text{assert len(ps) == len(vs)} \]

reverse and filter, keeping only positive numbers

\[ \text{vs} \]

\[ (3, 1, -2) \]

\[ \text{ps} \]

\[ (1, 3) \]
Translation to constraints by example

vs
(3, 1, -2)

ps = ()
for v in vs:
    if v > 0:
        ps = insert(v, ps)

ps
(1, 3)
Translation to constraints by example

solve:
    ps = ()
    for v in vs:
        if v > 0:
            ps = insert(v, ps)
    assert len(ps) == len(vs)
Translation to constraints by example

solve:
    ps = ()
    for v in vs:
        if v > 0:
            ps = insert(v, ps)
    assert len(ps) == len(vs)
Translation to constraints by example

solve:
  ps = ()
  for v in vs:
    if v > 0:
      ps = insert(v, ps)
  assert len(ps) == len(vs)

vs
(a, b)

constraints
a>0 ∧ b>0
Design space of precise symbolic encodings

solve:
    ps = ()
    for v in vs:
        if v > 0:
            ps = insert(v, ps)
    assert len(ps) == len(vs)

symbolic execution

bounded model checking
Design space of precise symbolic encodings

solve:
ps = ()
for v in vs:
    if v > 0:
        ps = insert(v, ps)
assert len(ps) == len(vs)
Design space of precise symbolic encodings

solve:
  ps = ()
  for v in vs:
    if v > 0:
      ps = insert(v, ps)
  assert len(ps) == len(vs)
Design space of precise symbolic encodings

solve:
ps = ()
for v in vs:
    if v > 0:
        ps = insert(v, ps)
assert len(ps) == len(vs)

bounded model checking

vs ↦ (a, b)
ps ↦ ()
a ≤ 0
ps ↦ ()
b ≤ 0
ps ↦ (b)
b > 0
ps ↦ (b, a)
ps0 = ite(a > 0, (a), ( ))

symbolic execution

vs ↦ (a, b)
ps ↦ ()
a ≤ 0
ps ↦ ()

a ≤ 0
b ≤ 0
false

a ≤ 0
b > 0

a > 0
b ≤ 0
false

a > 0
b > 0
false

assert len(ps) == len(vs)
Design space of precise symbolic encodings

**solve:**

\[
\begin{align*}
\text{ps} &= () \\
\text{for } v \text{ in } \text{vs}: \\
\quad \text{if } v > 0: \\
\quad \quad \text{ps} &= \text{insert}(v, \text{ps}) \\
\text{assert } \text{len(ps)} &= \text{len(vs)}
\end{align*}
\]

**symbolic execution**

\[
\begin{align*}
\text{vs} \mapsto (a, b) \\
\text{ps} \mapsto ()
\end{align*}
\]

\[
\begin{align*}
\text{a} \leq 0 & \rightarrow \text{ps} \mapsto () \\
\text{ps} \mapsto () & \rightarrow \text{ps} \mapsto (a) \\
\text{b} \leq 0 & \rightarrow \text{ps} \mapsto (b) \\
\text{ps} \mapsto (b) & \rightarrow \text{ps} \mapsto (a) \\
\text{b} > 0 & \rightarrow \text{ps} \mapsto (b, a)
\end{align*}
\]

\[
\begin{align*}
\{a \leq 0\} \land \{b \leq 0\} \land \{\text{false}\} & \rightarrow \{a \leq 0\} \land \{b > 0\} \land \{\text{false}\} \land \{\text{false}\} \\
\{a \leq 0\} \land \{b > 0\} \land \{\text{false}\} \land \{\text{false}\} \land \{a > 0\} \land \{b \leq 0\} \land \{\text{false}\} & \rightarrow \{a > 0\} \land \{b > 0\} \land \{a > 0\} \land \{b > 0\} \land \{\text{false}\} \land \{\text{true}\}
\end{align*}
\]

\[
\begin{align*}
\text{bounded model checking}
\end{align*}
\]

\[
\begin{align*}
\text{vs} \mapsto (a, b) \\
\text{ps} \mapsto () \\
\text{a} \leq 0 & \rightarrow \text{ps} \mapsto (a) \\
\text{ps} \mapsto (a) & \rightarrow \text{ps} \mapsto ps_0 \\
\text{b} > 0 & \rightarrow \text{ps} \mapsto ps_1
\end{align*}
\]

\[
\begin{align*}
ps_0 &= \text{ite}(a > 0, (a), ( )) \\
ps_1 &= \text{insert}(b, ps_0)
\end{align*}
\]
Design space of precise symbolic encodings

solve:
ps = ()
for v in vs:
    if v > 0:
        ps = insert(v, ps)
assert len(ps) == len(vs)

symbolic execution

bounded model checking

ps₀ = ite(a > 0, (a), ( ))
ps₁ = insert(b, ps₀)
ps₂ = ite(b > 0, ps₀, ps₁)
assert len(ps₂) = 2
A new design: type-driven state merging

solve:
    ps = ()
    for v in vs:
        if v > 0:
            ps = insert(v, ps)
    assert len(ps) == len(vs)
solve:
    ps = ()
    for v in vs:
        if v > 0:
            ps = insert(v, ps)
    assert len(ps) == len(vs)

Merge values of
  ‣ primitive types: symbolically
  ‣ immutable types: structurally
  ‣ all other types: via unions

\{ a > 0, b > 0, true \}
A new design: type-driven state merging

solve:
ps = ()
for v in vs:
    if v > 0:
        ps = insert(v, ps)
assert len(ps) == len(vs)

Merge values of
- primitive types: symbolically
- immutable types: structurally
- all other types: via unions
A new design: type-driven state merging

solve:
    ps = ()
    for v in vs:
        if v > 0:
            ps = insert(v, ps)
    assert len(ps) == len(vs)

Merge values of
    • primitive types: symbolically
    • immutable types: structurally
    • all other types: via unions
A new design: type-driven state merging

```
solve:
    ps = ()
    for v in vs:
        if v > 0:
            ps = insert(v, ps)
    assert len(ps) == len(vs)
```

Merge values of
- primitive types: symbolically
- immutable types: structurally
- all other types: via unions
A new design: type-driven state merging

solve:
  ps = ()
  for v in vs:
    if v > 0:
      ps = insert(v, ps)
  assert len(ps) == len(vs)
A new design: type-driven state merging

solve:
   ps = ()
   for v in vs:
      if v > 0:
         ps = insert(v, ps)
   assert len(ps) == len(vs)
solve:
ps = ()
for v in vs:
    if v > 0:
        ps = insert(v, ps)
assert len(ps) == len(vs)

Symbolic union: a set of guarded values, with disjoint guards.

g₀ = a > 0

symbolic virtual machine

vs ⊩ (a, b)
ps ⊩ ()
¬g₀ ⊩ ()
g₀ ⊩ ()

ps ⊩ (a)

ps ⊩ { g₀ ⊩ (a),
         ¬g₀ ⊩ () }

ps ⊩ ()
A new design: type-driven state merging

solve:

```python
ps = ()
for v in vs:
    if v > 0:
        ps = insert(v, ps)
assert len(ps) == len(vs)
```

Execute `insert` concretely on all lists in the union.

```latex
\begin{align*}
g_0 &= a > 0 \\
g_1 &= b > 0
\end{align*}
```

symbolic virtual machine

- \( vs \mapsto (a, b) \)
- \( ps \mapsto () \)
- \( \neg g_0 \mapsto () \)
- \( g_0 \mapsto (a) \)
- \( ps \mapsto \{ g_0 \vdash (a), \neg g_0 \vdash ( ) \} \)
- \( g_1 \mapsto ( ) \)
- \( ps \mapsto \{ g_0 \vdash (b, a), \neg g_0 \vdash (b) \} \)
A new design: type-driven state merging

solve:

```python
ps = ()
for v in vs:
    if v > 0:
        ps = insert(v, ps)
assert len(ps) == len(vs)
```

g₀ = a > 0
g₁ = b > 0
A new design: type-driven state merging

solve:
   ps = ()
   for v in vs:
       if v > 0:
           ps = insert(v, ps)
   assert len(ps) == len(vs)

\[ g_0 = a > 0 \]
\[ g_1 = b > 0 \]
\[ g_2 = g_0 \land g_1 \]
\[ g_3 = \neg (g_0 \Leftrightarrow g_1) \]
\[ g_4 = \neg g_0 \land \neg g_1 \]
\[ c = \text{ite}(g_1, b, a) \]
A new design: type-driven state merging

solve:
ps = ()
for v in vs:
    if v > 0:
        ps = insert(v, ps)
assert len(ps) == len(vs)

Evaluate \( \text{len} \) concretely on all lists in the union; assertion true only on the list guarded by \( g_2 \).

\[
\begin{align*}
g_0 &= a > 0 \quad & \text{assert } g_2

g_1 &= b > 0

g_2 &= g_0 \land g_1

g_3 &= \neg(g_0 \iff g_1)

g_4 &= \neg g_0 \land \neg g_1
\end{align*}
\]
A new design: type-driven state merging

solve:
ps = ()
for v in vs:
    if v > 0:
        ps = insert(v, ps)
assert len(ps) == len(vs)

g₀ = a > 0
g₁ = b > 0
g₂ = g₀ ∧ g₁
g₃ = ¬(g₀ ⇔ g₁)
g₄ = ¬g₀ ∧ ¬g₁
c = ite(g₁, b, a)
assert g₂
solver-aided programming for everyone
Recent applications of ROSETTE

**Synthesizing Memory Models from Litmus Tests**
James Bornholt and Emina Torlak (under submission)

**Synthesizing Custom Tutoring Rules for Introductory Algebra**
Eric Butler, Emina Torlak, and Zoran Popovic (under submission)

**Scalable Verification of BGP Configurations (OOPSLA’16)**
Konstantin Weitz, Doug Woos, Emina Torlak, Michael D. Ernst, Arvind Krishnamurthy, and Zachary Tatlock

**Investigating Safety of a Radiotherapy Machine (CAV’16)**
Stuart Pernsteiner, Calvin Loncaric, Emina Torlak, Zachary Tatlock, Xi Wang, Michael Ernst, and Jon Jacky

Eric Butler, Emina Torlak, and Zoran Popovic

**Specifying and Checking File System Crash-Consistency Models (ASPLOS’16)**
James Bornholt, Antoine Kaufmann, Jialin Li, Arvind Krishnamurthy, Emina Torlak, and Xi Wang.

**Scaling up Superoptimization (ASPLOS’16)**
Phitchaya Mangpo Phothilimthana, Aditya Thakur, Rastislav Bodík, and Dinakar Dhurjati.

**Synapse: Optimizing Synthesis with Metasketches (POPL’16)**
James Bornholt, Emina Torlak, Dan Grossman, and Luis Ceze.

...
Recent applications of ROSETTE

Investigating Safety of a Radiotherapy Machine (CAV’16)
Stuart Pernsteiner, Calvin Loncaric, Emina Torlak, Zachary Tatlock, Xi Wang, Michael Ernst, and Jon Jacky

Scaling up Superoptimization (ASPLOS’16)
Phitchaya Mangpo Phothilimthana, Aditya Thakur, Rastislav Bodík, and Dinakar Dhurjati.
Recent applications of ROSETTE: Neutrons

Investigating Safety of a Radiotherapy Machine (CAV’16)
Stuart Pernsteiner, Calvin Loncaric, Emina Torlak, Zachary Tatlock, Xi Wang, Michael Ernst, and Jon Jacky

Clinical Neutron Therapy System (CNTS) at UW

• Used Rosette to build a verifier for the EPICS DSL.
• Found safety-critical defects in a pre-release version of the therapy control software.
• Used by CNTS staff to verify changes to the control software.

Scaling up Superoptimization (ASPLOS’16)
Phitchaya Mangpo Phothilimthana, Aditya Thakur, Rastislav Bodík, and Dinakar Dhurjati.
Recent applications of Rosette: Enlearn

Investigating Safety of a Radiotherapy Machine (CAV’16)
Stuart Pernsteiner, Calvin Loncaric, Emina Torlak, Zachary Tatlock, Xi Wang, Michael Ernst, and Jon Jacky

Scaling up Superoptimization (ASPLOS’16)
Phitchaya Mangpo Phothilimthana, Aditya Thakur, Rastislav Bodík, and Dinakar Dhurjati.

- Education technology company founded by Zoran Popovic.
- Uses Rosette to develop educational math games, released to thousands (and eventually millions) of students.
- Building a Rosette-based DSL to enable educators to develop their own educational tools.
Recent applications of ROSETTE: Greenthumb

Investigating Safety of a Radiotherapy Machine (CAV’16)
Stuart Pernsteiner, Calvin Loncaric, Emina Torlak, Zachary Tatlock, Xi Wang, Michael Ernst, and Jon Jacky

Scaling up Superoptimization (ASPLOS’16)
Phitchaya Mangpo Phothilimthana, Aditya Thakur, Rastislav Bodík, and Dinakar Dhurjati.

- A Rosette-based framework for creating efficient superoptimizers for new ISAs.
- Requires only an emulator for the ISA and a few ISA-specific search utility functions!
- Hosts superoptimizers for ARM and GreenArrays (GA) ISA.
- Up to 82% speedup over gcc -O3 for ARM; within 19% of hand optimized code for GA.
thanks
thanks

ROSETTE

your SDSL

synthesize
design
debuge
solve
verify

symbolic virtual machine