Synthesis and Verification for All

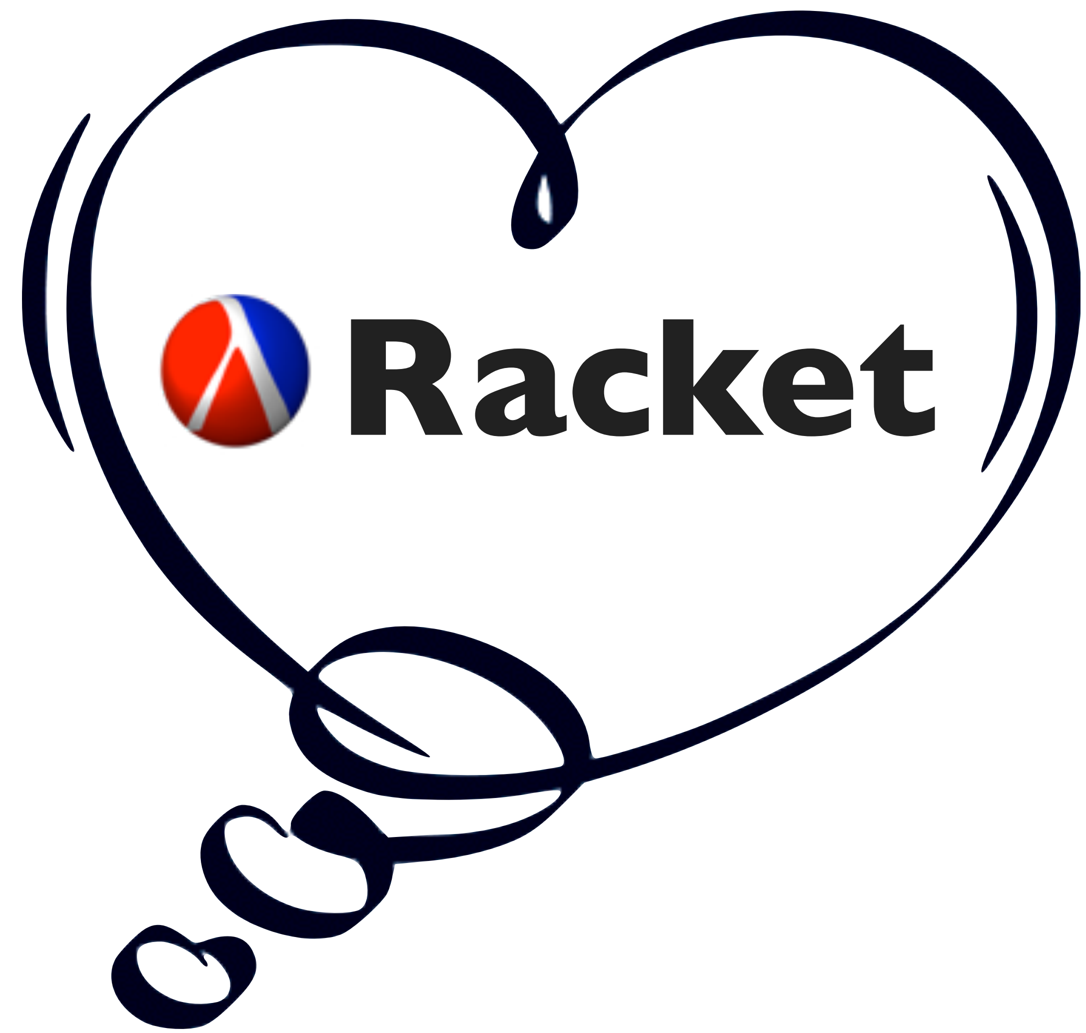
Emina Torlak

University of Washington

emina@cs.washington.edu

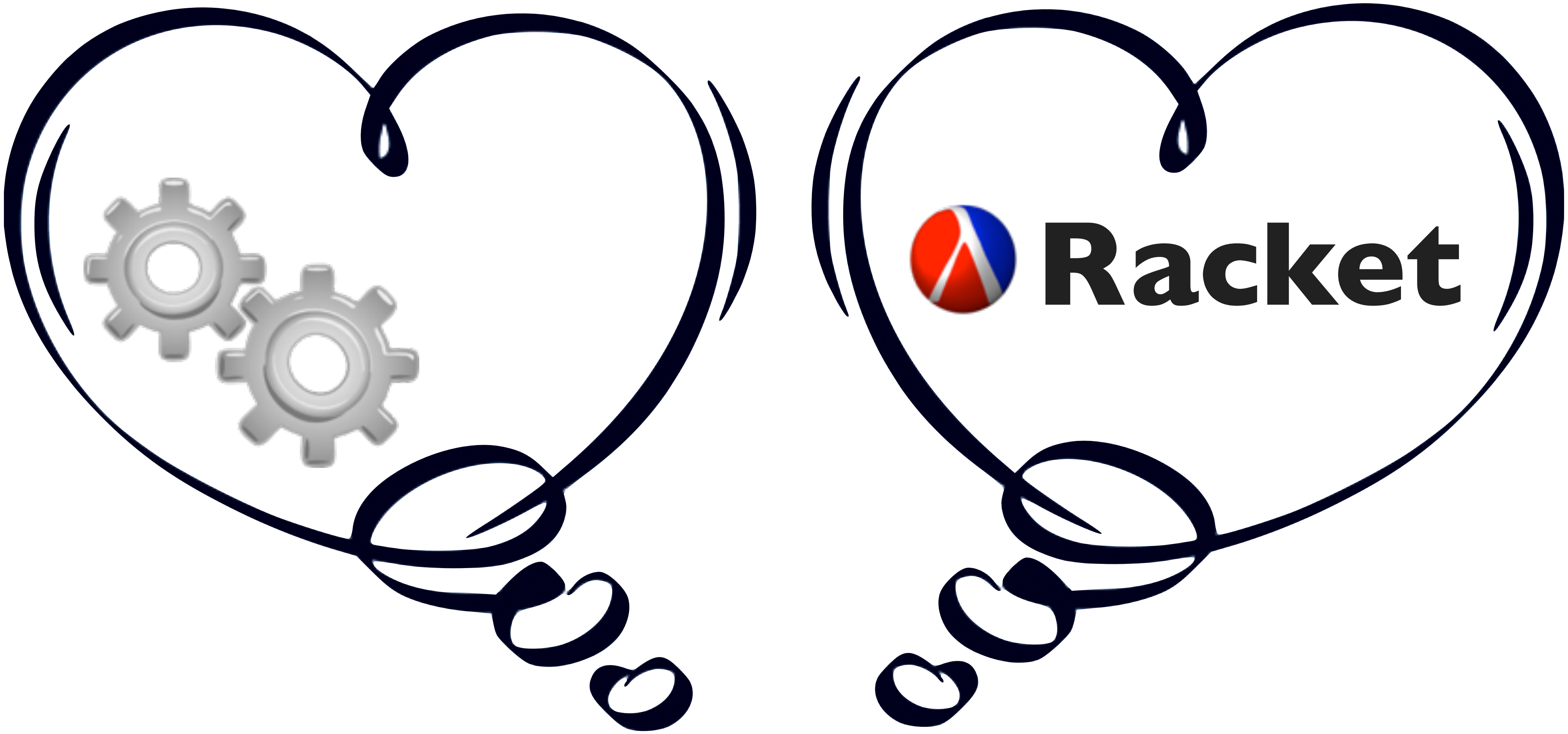
homes.cs.washington.edu/~emina/


Hello
my name is
Emina

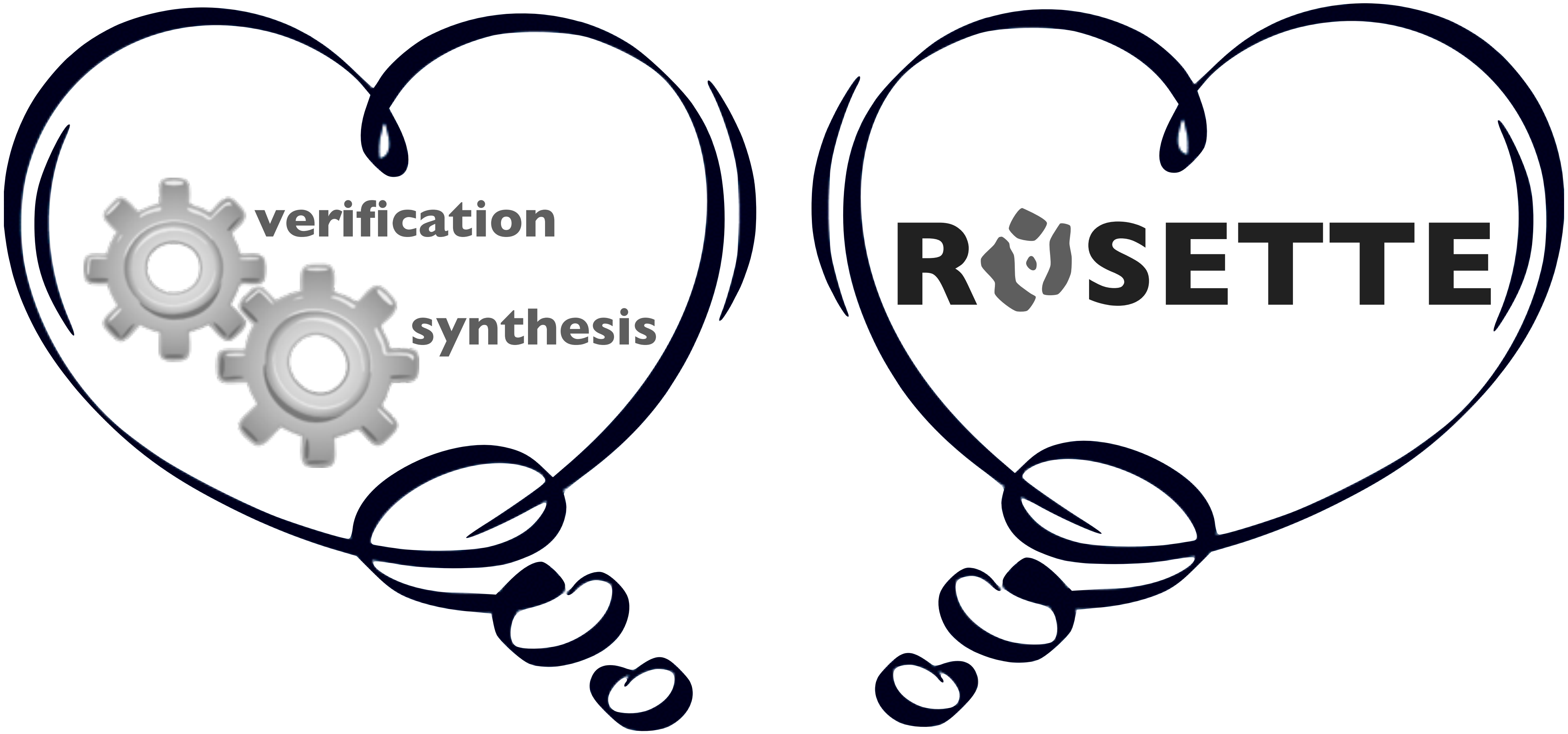

Racket

Racket

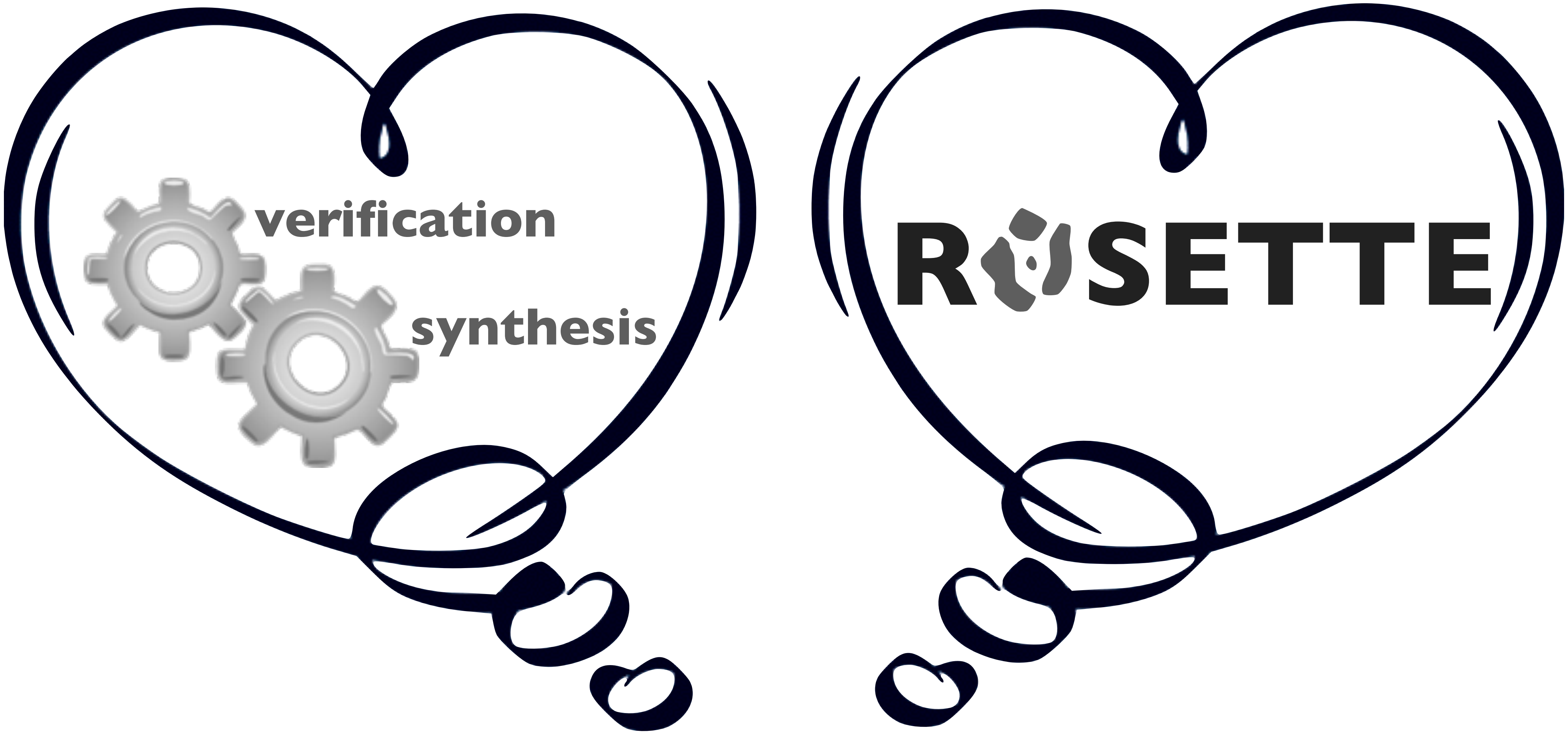
a programmable programming language



a programmable programming language



a programmable programming language



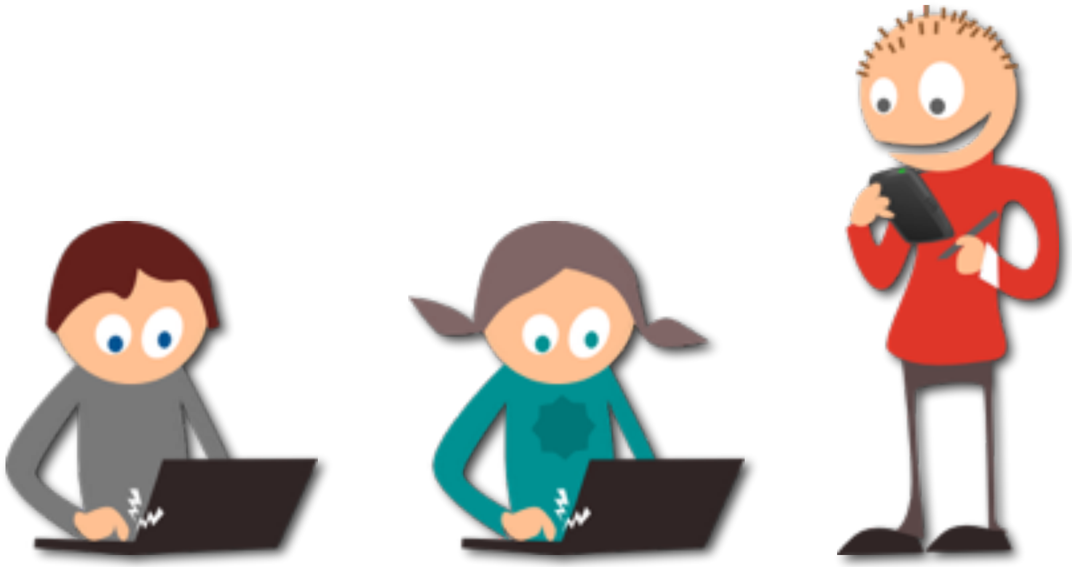
a solver-aided programming language

vision

a little programming for everyone

A little programming for everyone

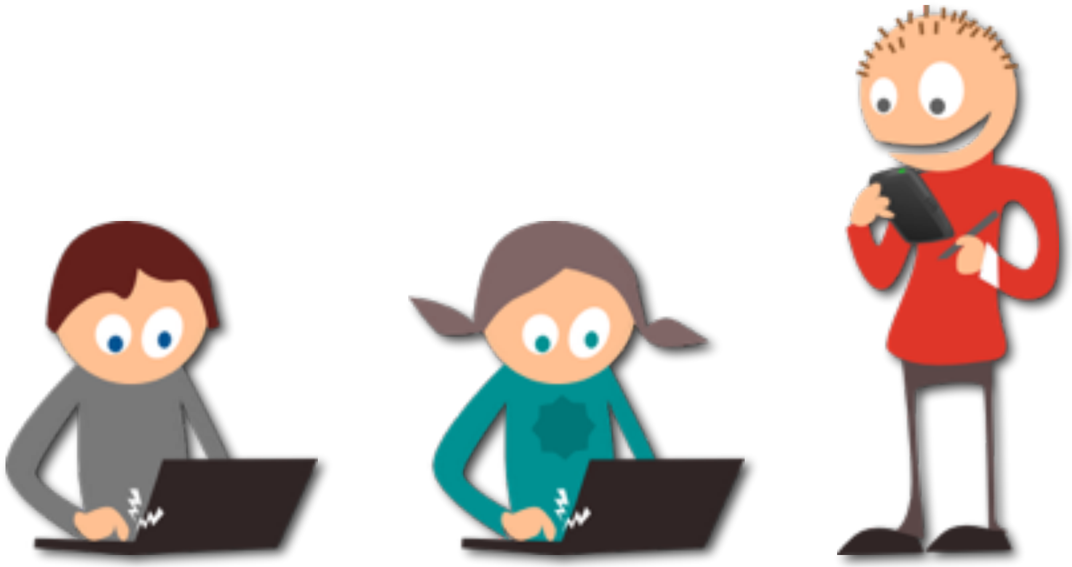
Every knowledge worker wants to program …



A little programming for everyone

Every knowledge worker wants to program …

- spreadsheet data manipulation

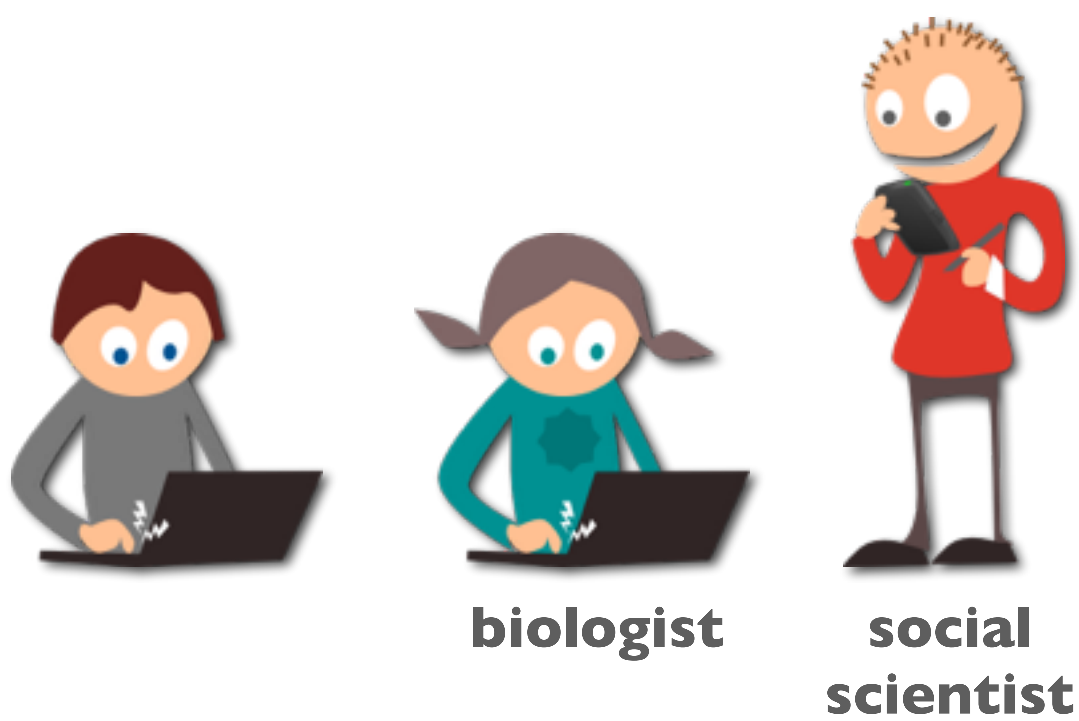


social
scientist

A little programming for everyone

Every knowledge worker wants to program …

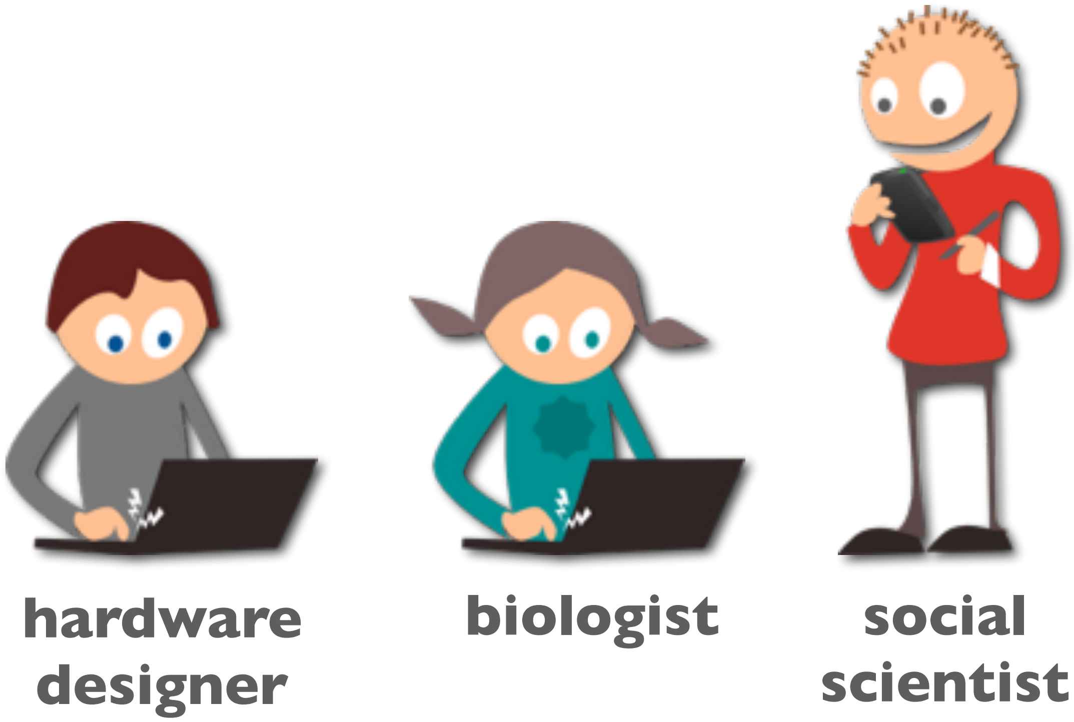
- spreadsheet data manipulation
- models of cell fates



A little programming for everyone

Every knowledge worker wants to program …

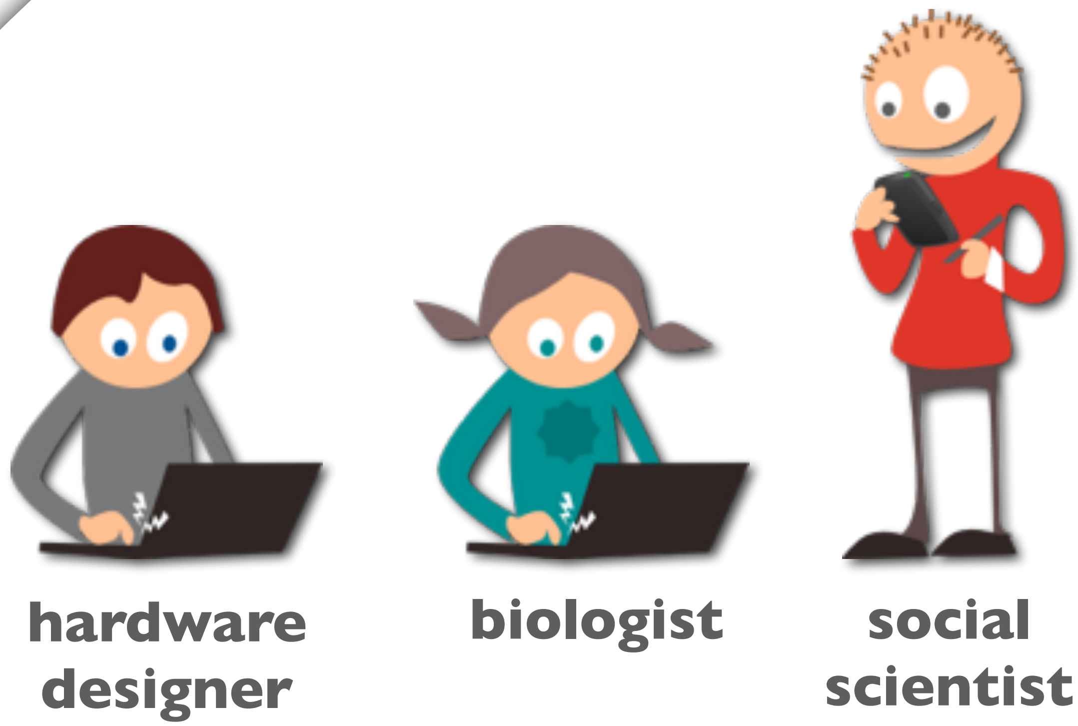
- spreadsheet data manipulation
- models of cell fates
- cache coherence protocols
- memory models



A little programming for everyone

Every knowledge worker wants to program …

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



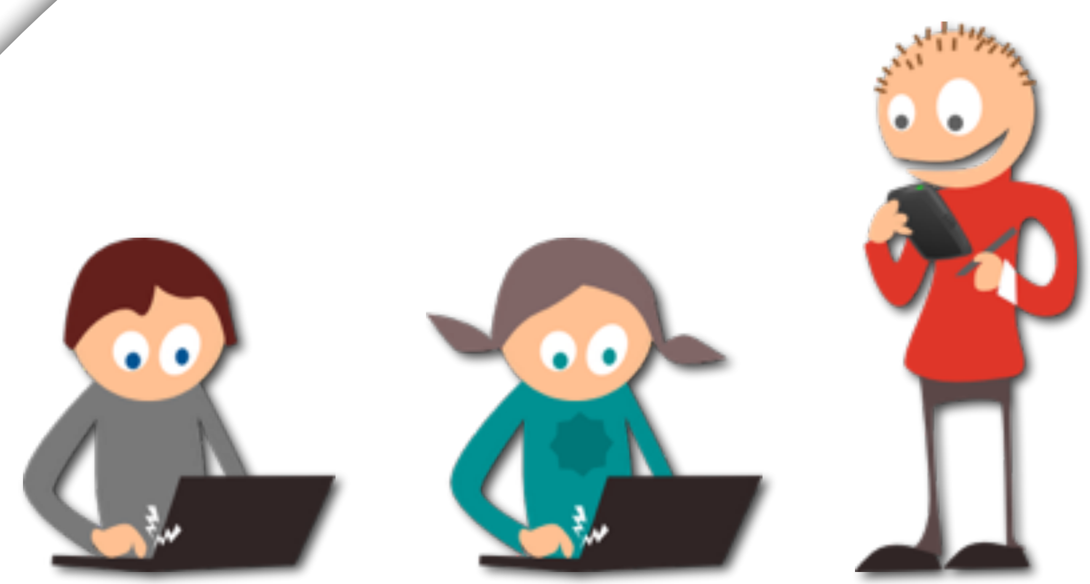
A little programming for **everyone**

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**]

time

expertise

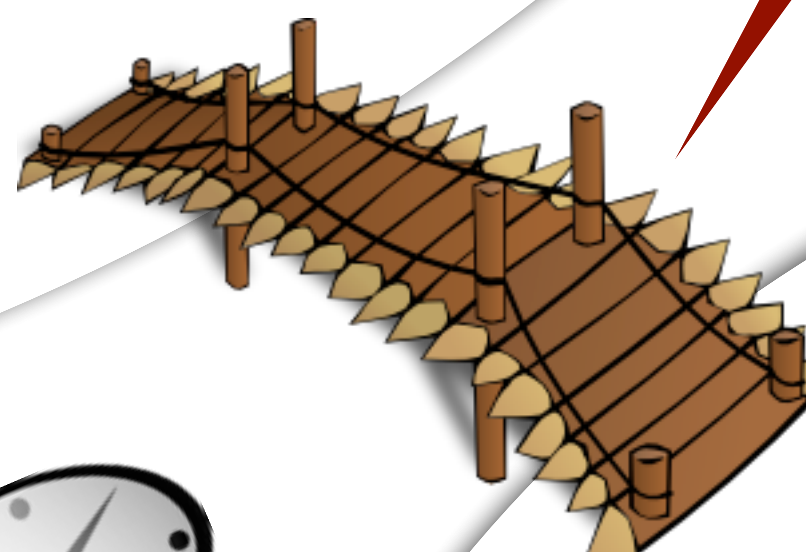


hardware designer **biologist** **social scientist**

A **little** programming for everyone

We all want to build programs …

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



solver-aided languages

less time

less expertise

hardware designer

biologist

social scientist

outline

outline

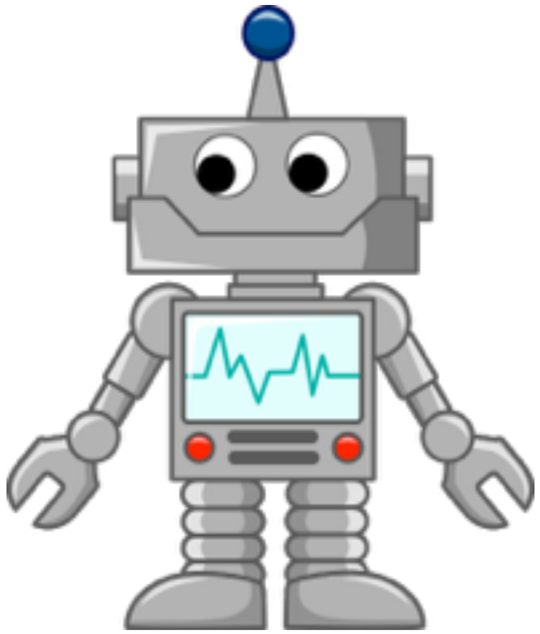
solver-aided tools

outline

solver-aided tools, languages

outline

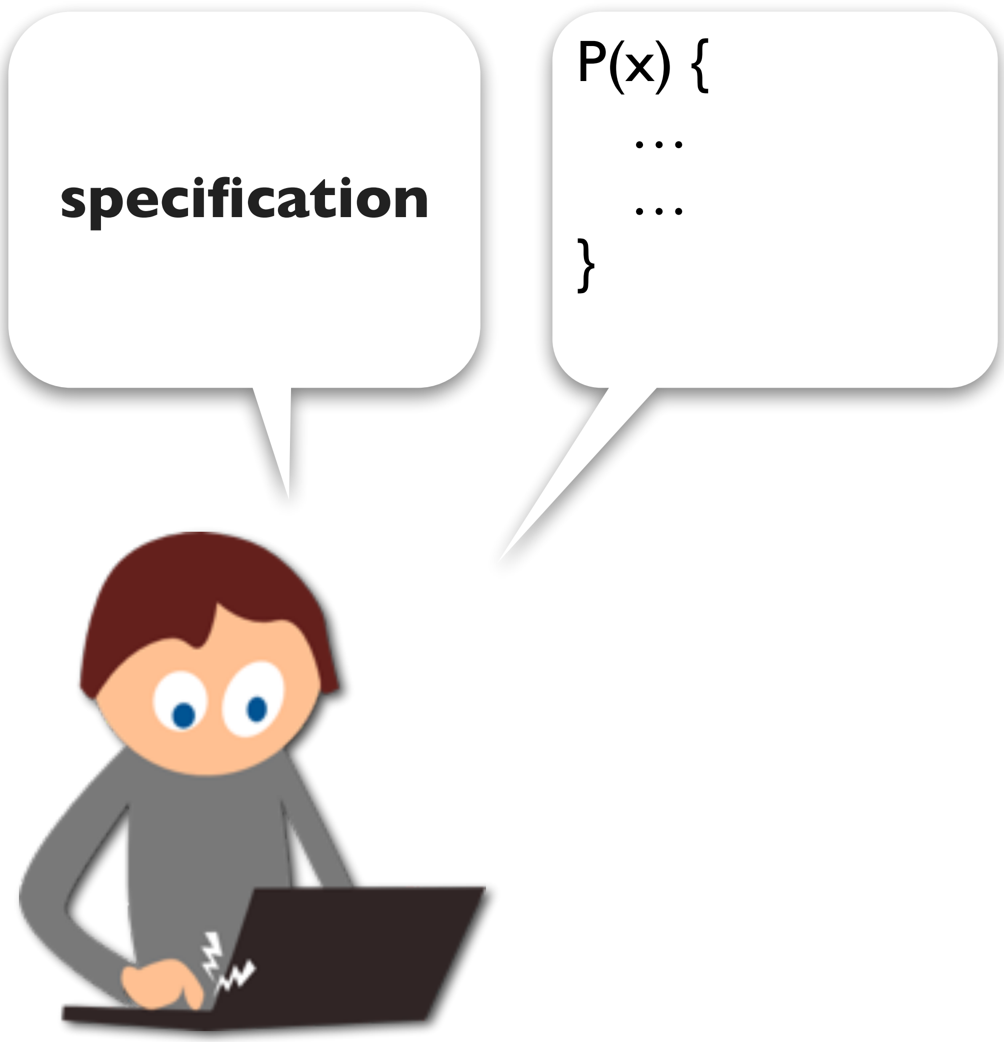
solver-aided tools, languages, and applications



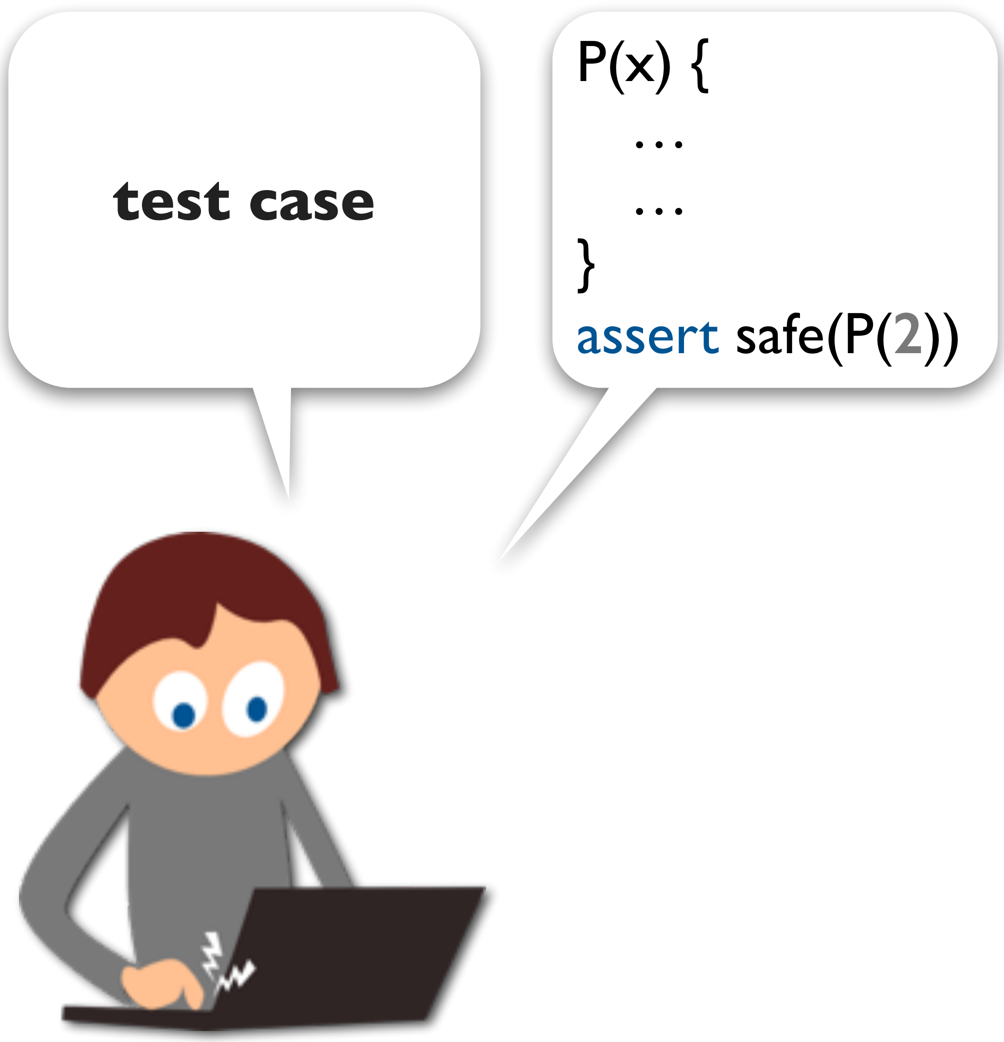
tools

solver-aided tools

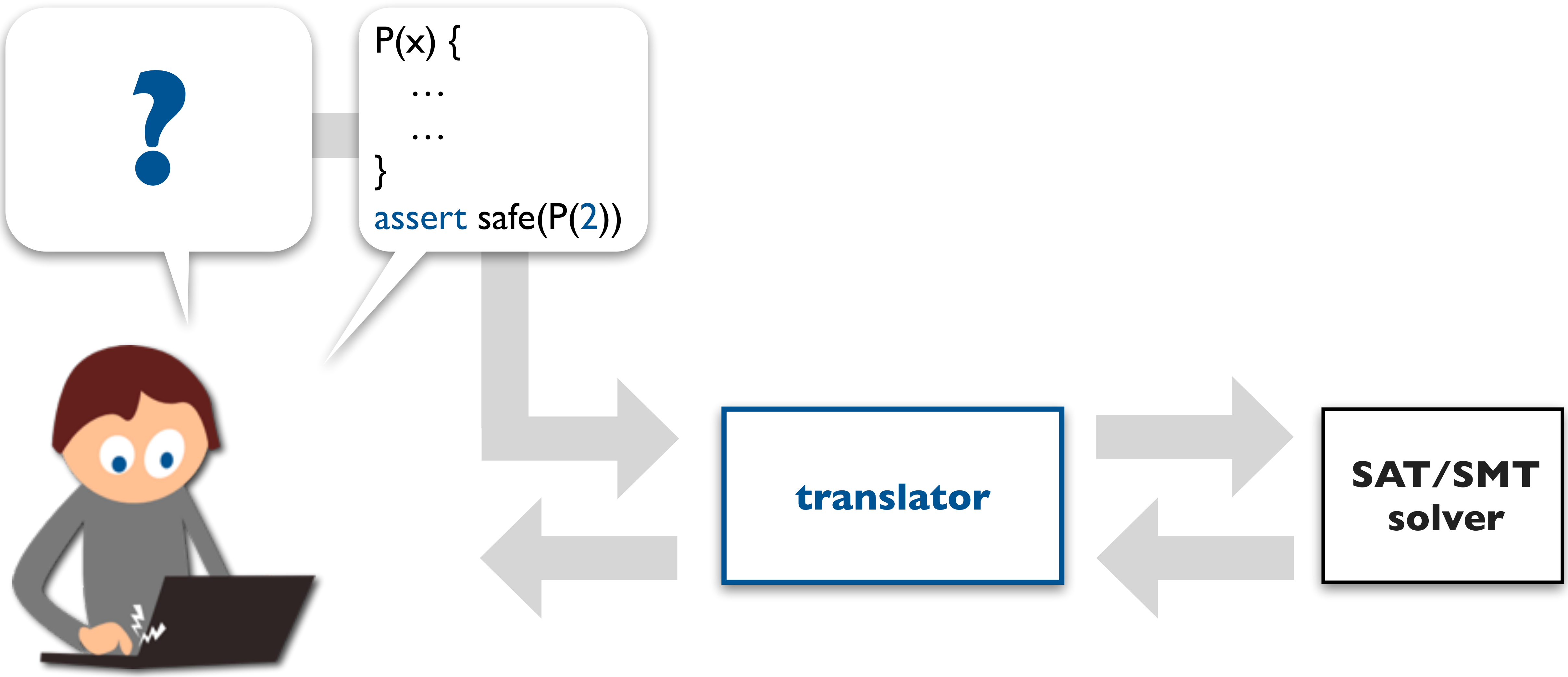
Programming ...



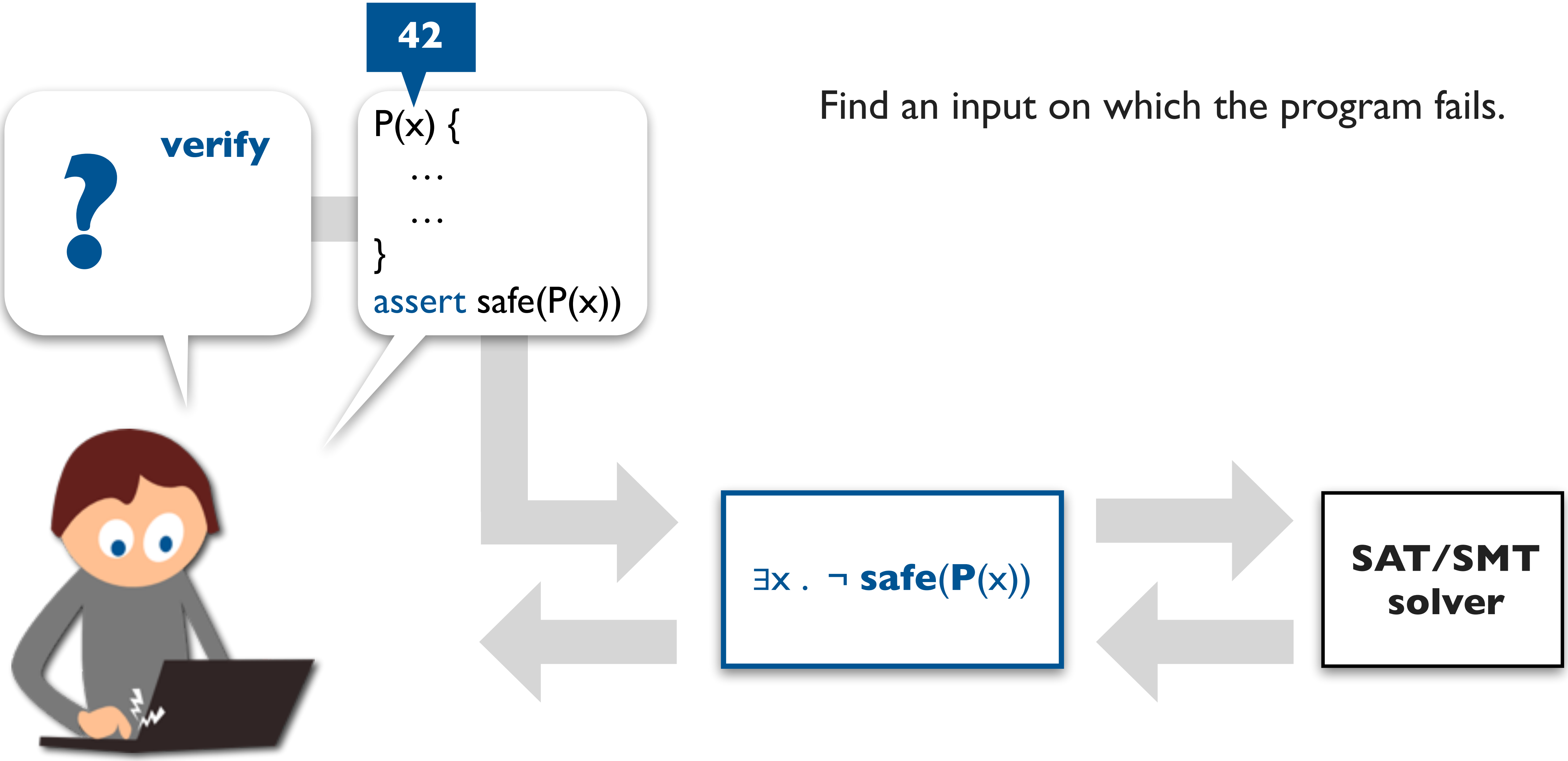
Programming ...



Programming with a solver-aided tool



Programming with a solver-aided tool



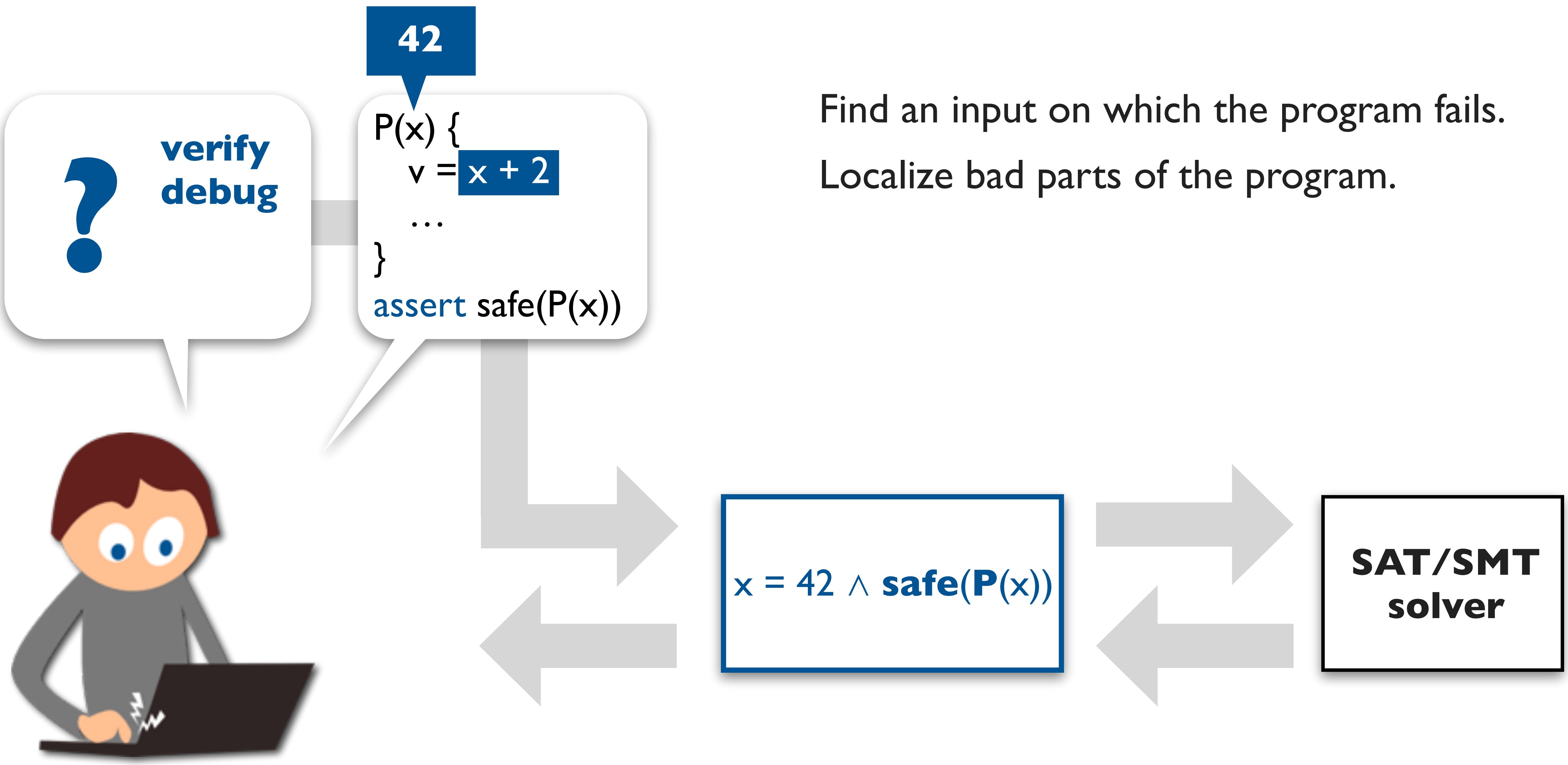
Find an input on which the program fails.

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

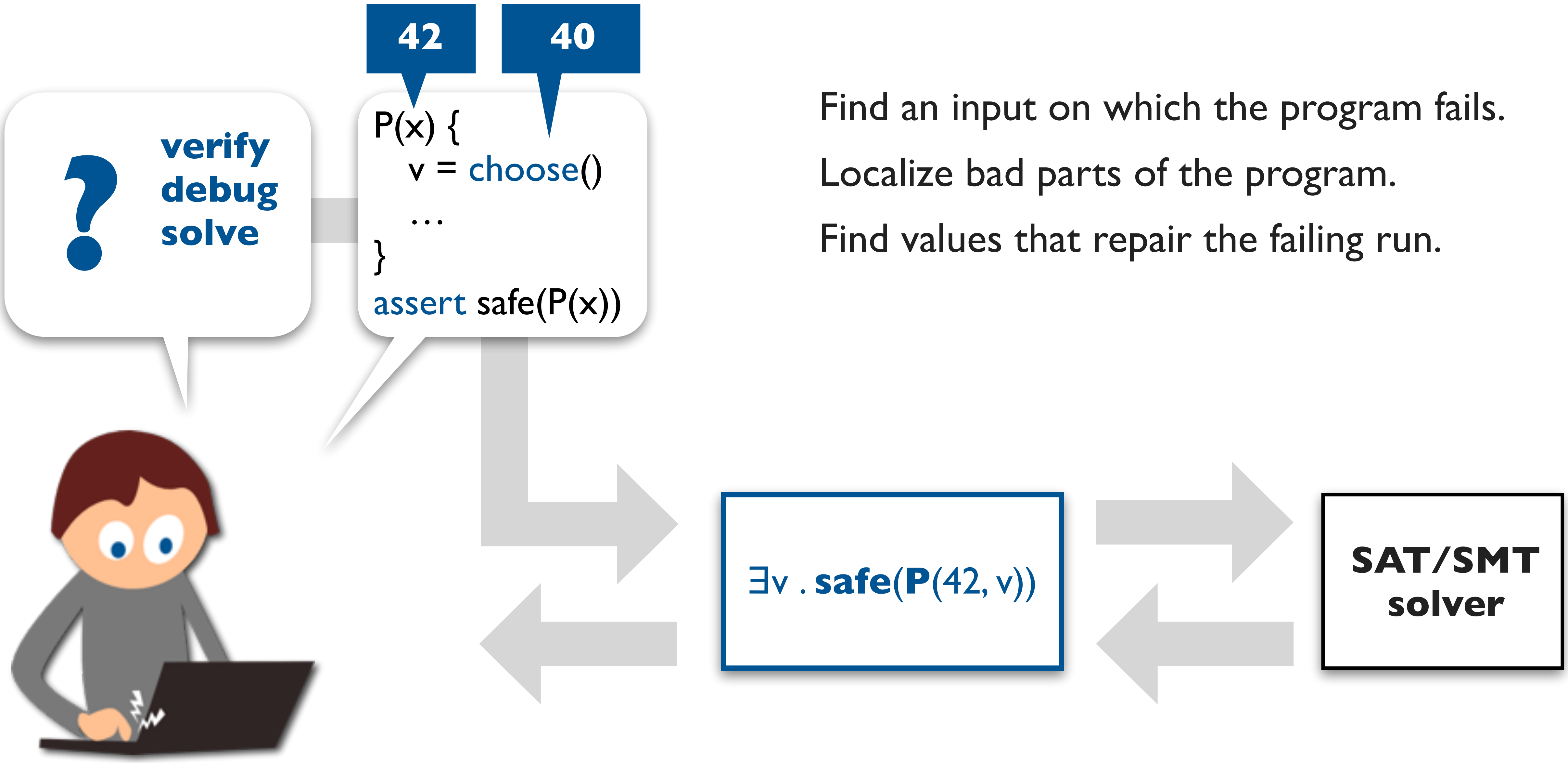
Find an input on which the program fails.

Localize bad parts of the program.



BugAssist [Jose & Majumdar, PLDI'11]

Programming with a solver-aided tool



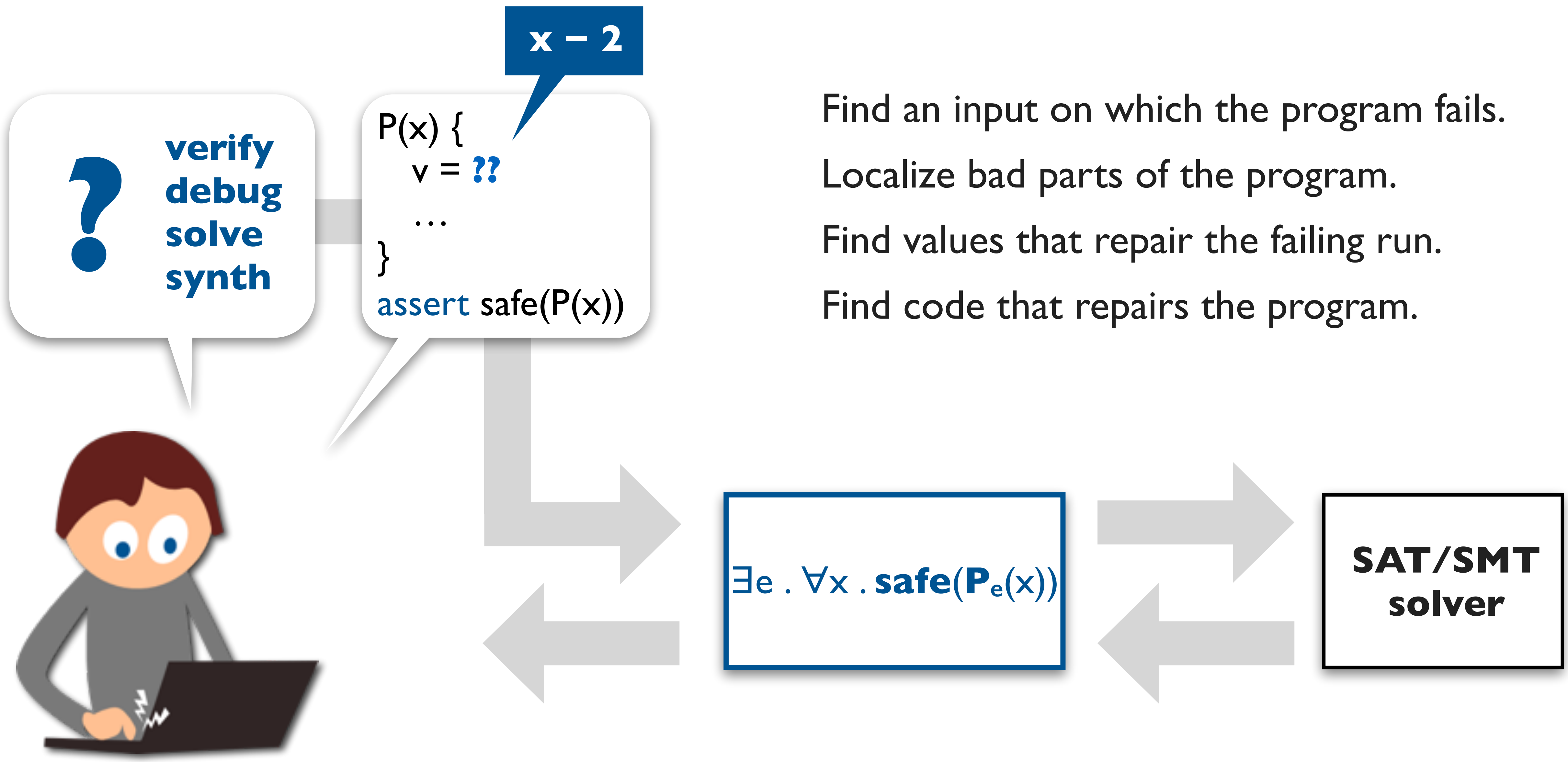
Find an input on which the program fails.

Localize bad parts of the program.

Find values that repair the failing run.

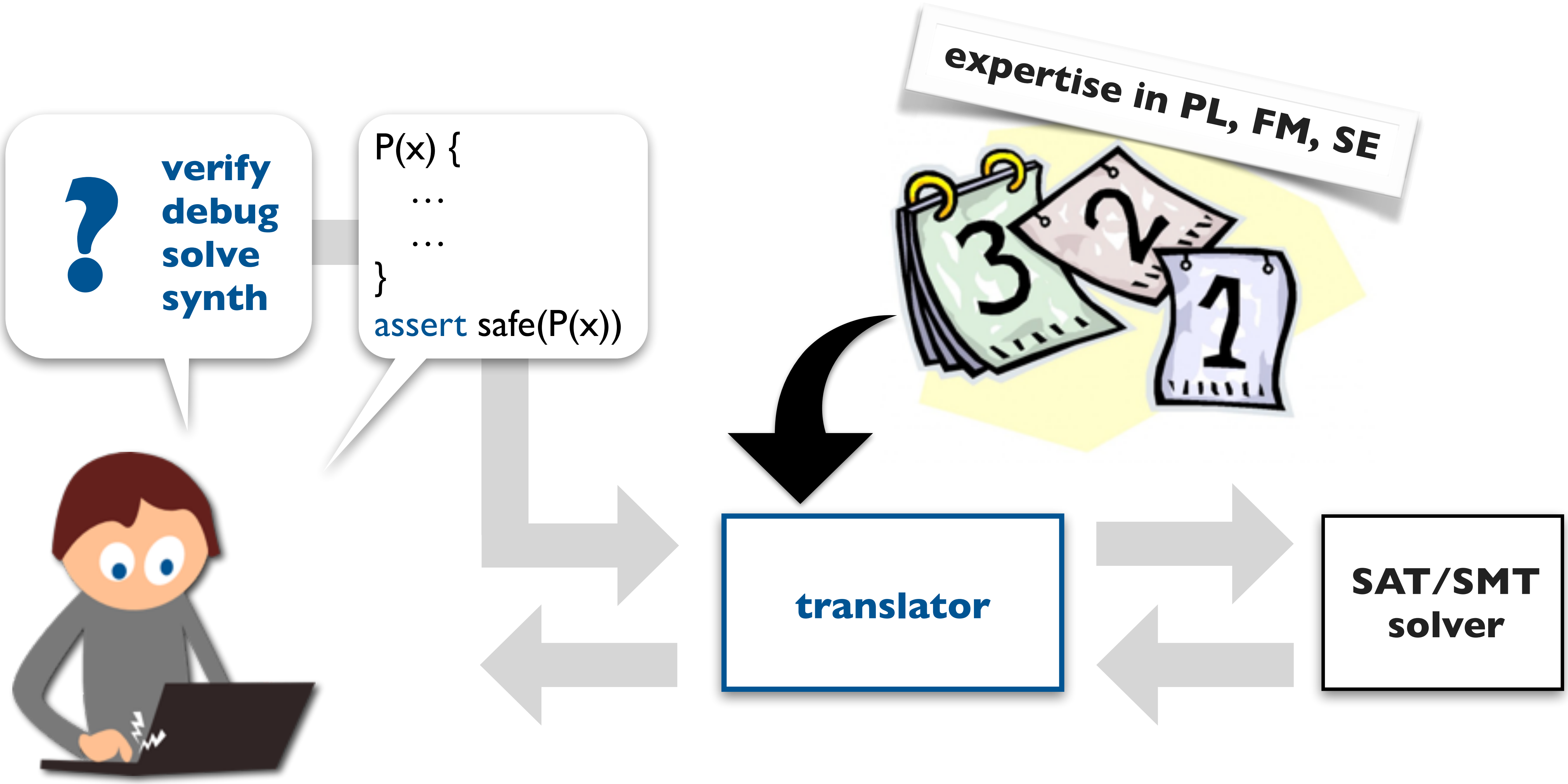
Kaplan [Koksal et al, POPL'12]
PBnJ [Samimi et al., ECOOP'10]
Squander [Milicevic et al., ICSE'11]

Programming with a solver-aided tool

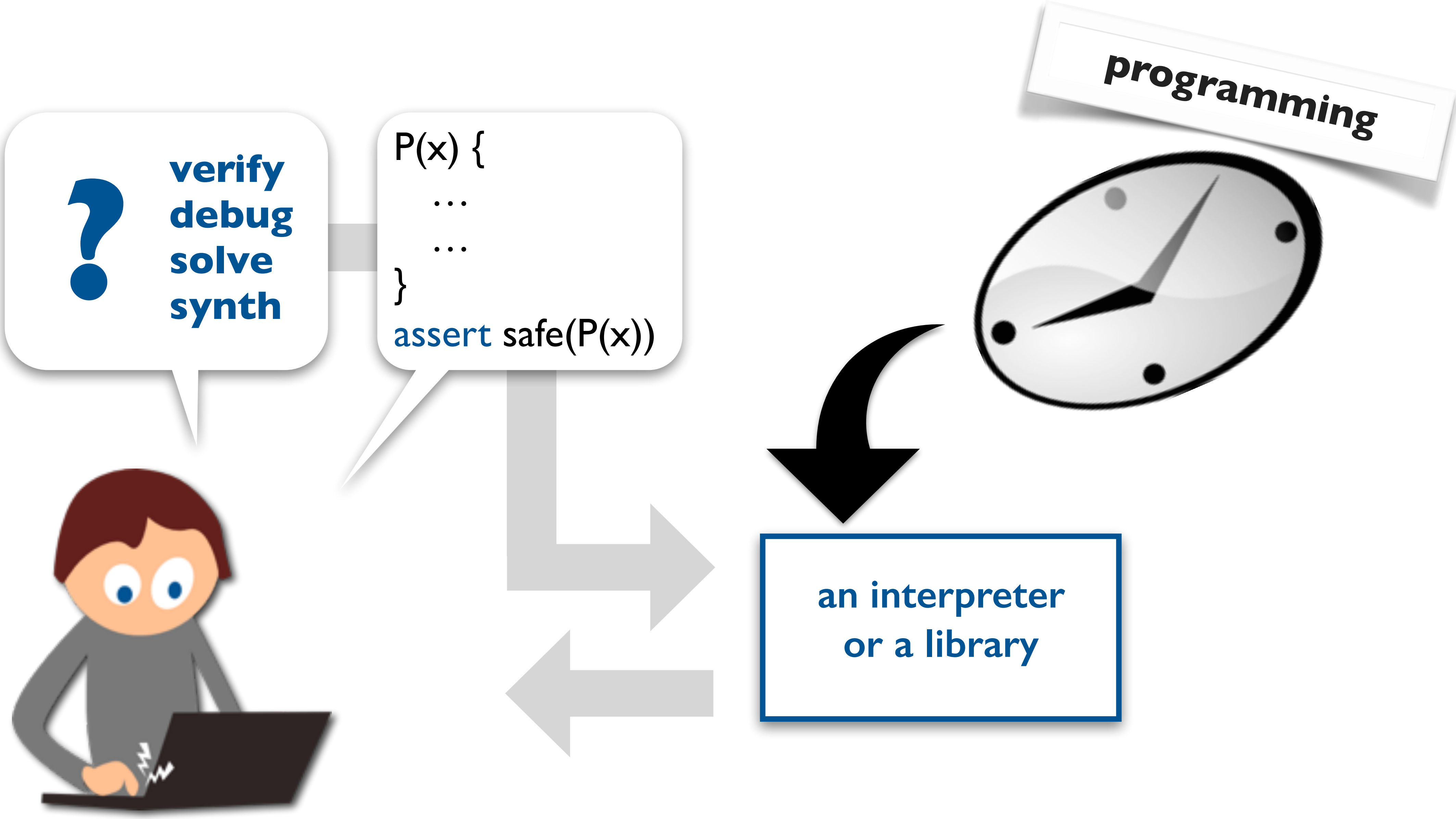


Sketch [Solar-Lezama et al., ASPLOS'06]
Comfusy [Kuncak et al., CAV'10]

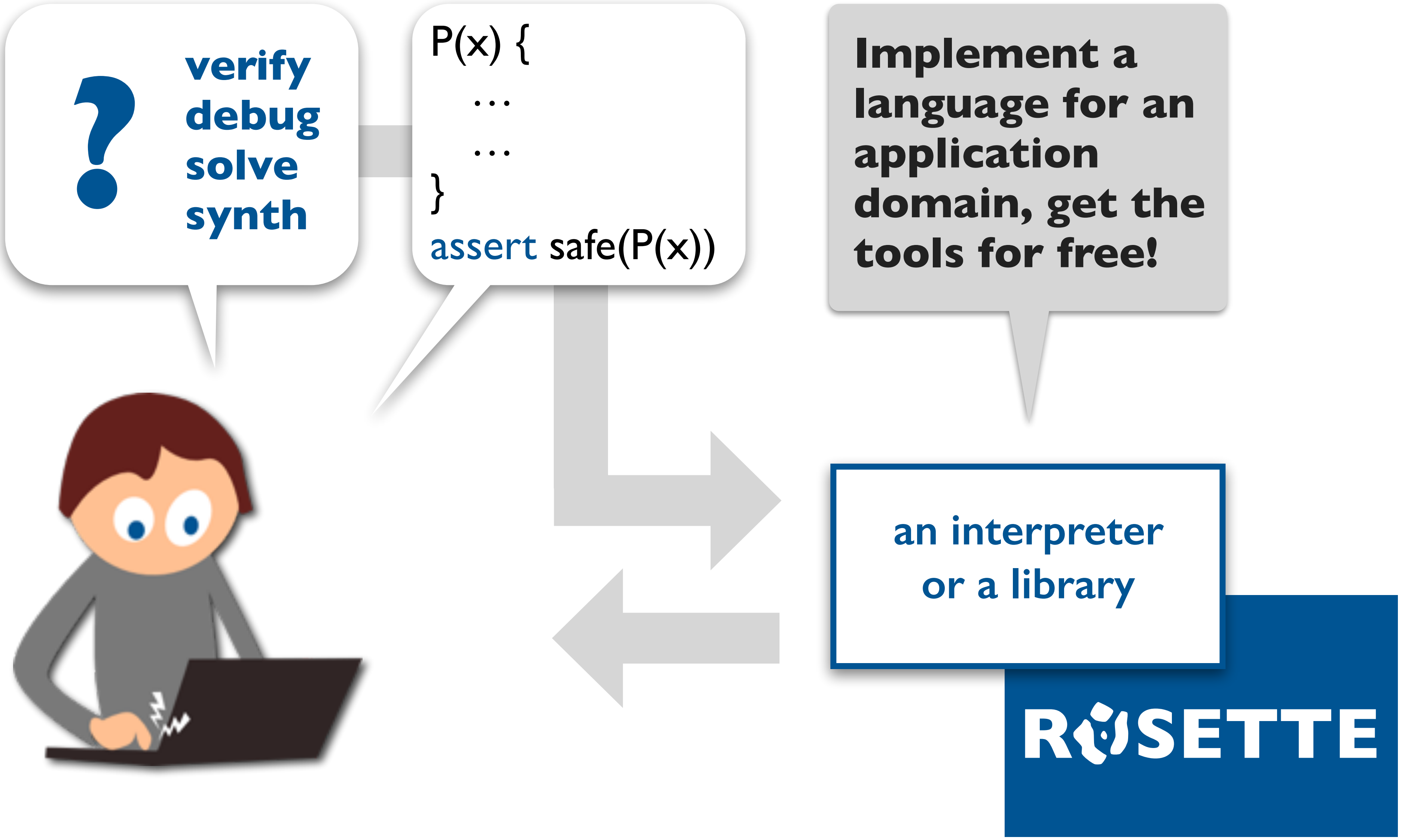
The standard (hard) way to build a tool



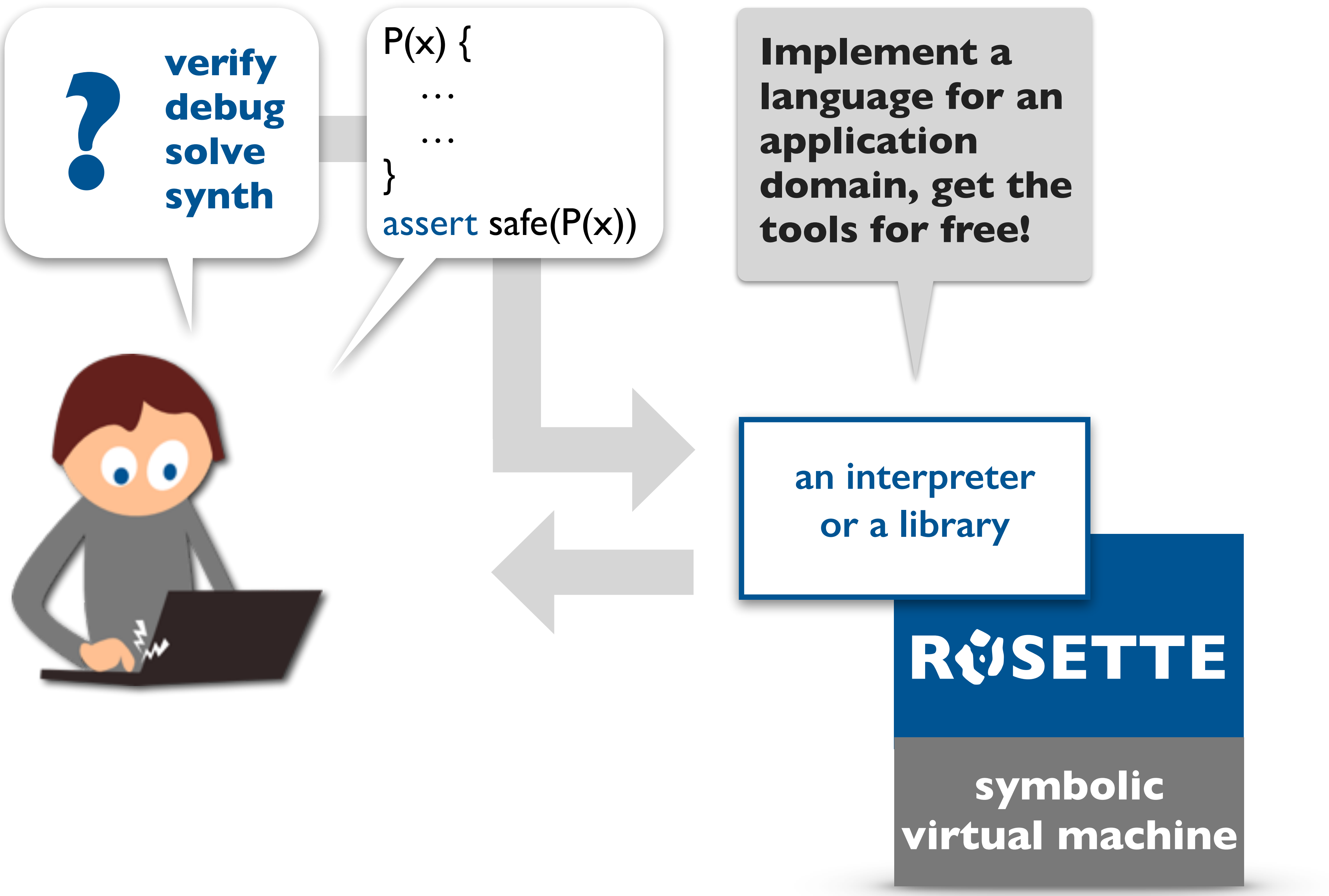
A new, easy way to build tools



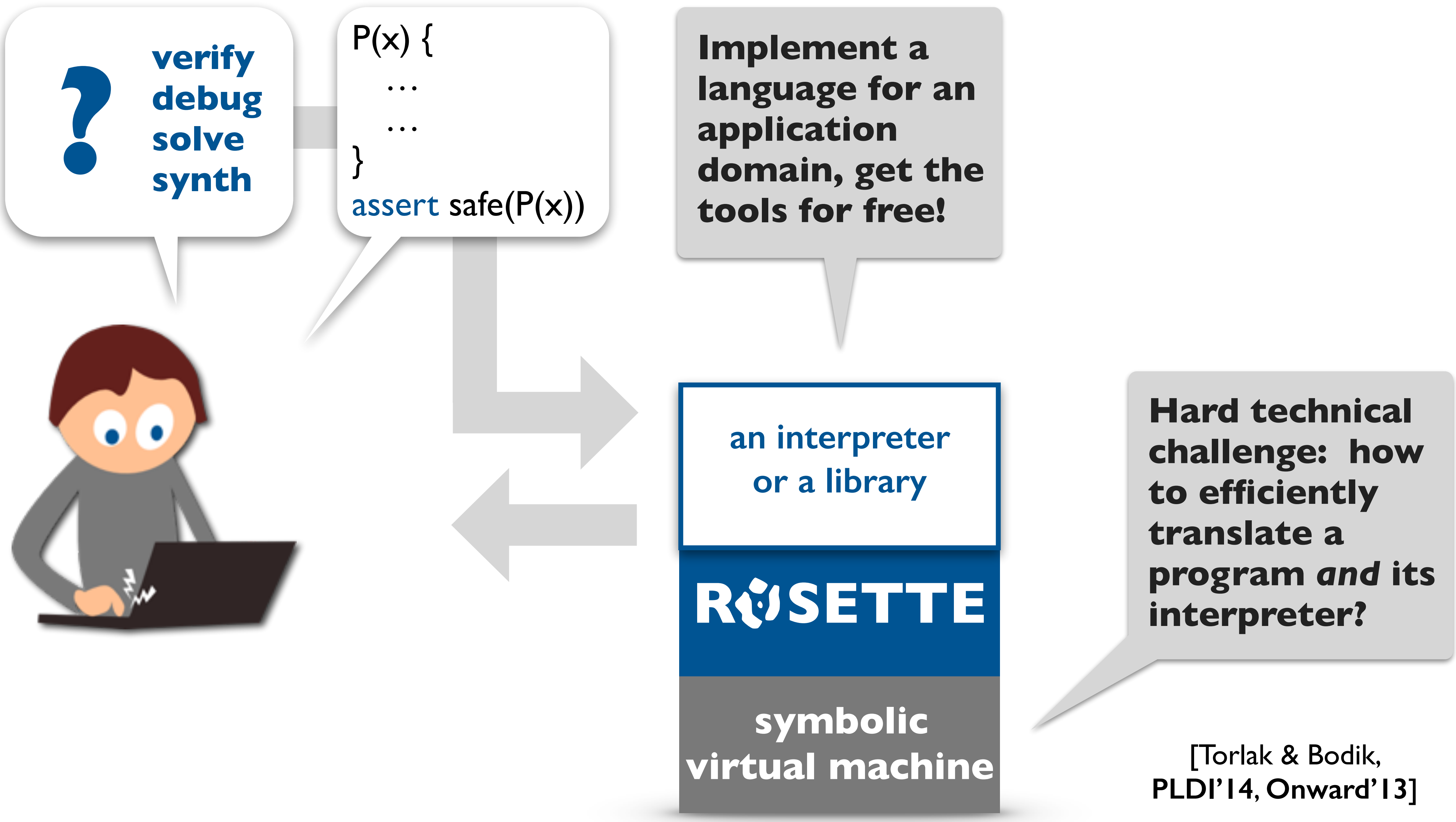
A new, easy way to build tools



A new, easy way to build tools



A new, easy way to build tools



[Torlak & Bodik, PLDI'14, Onward'13]

design

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

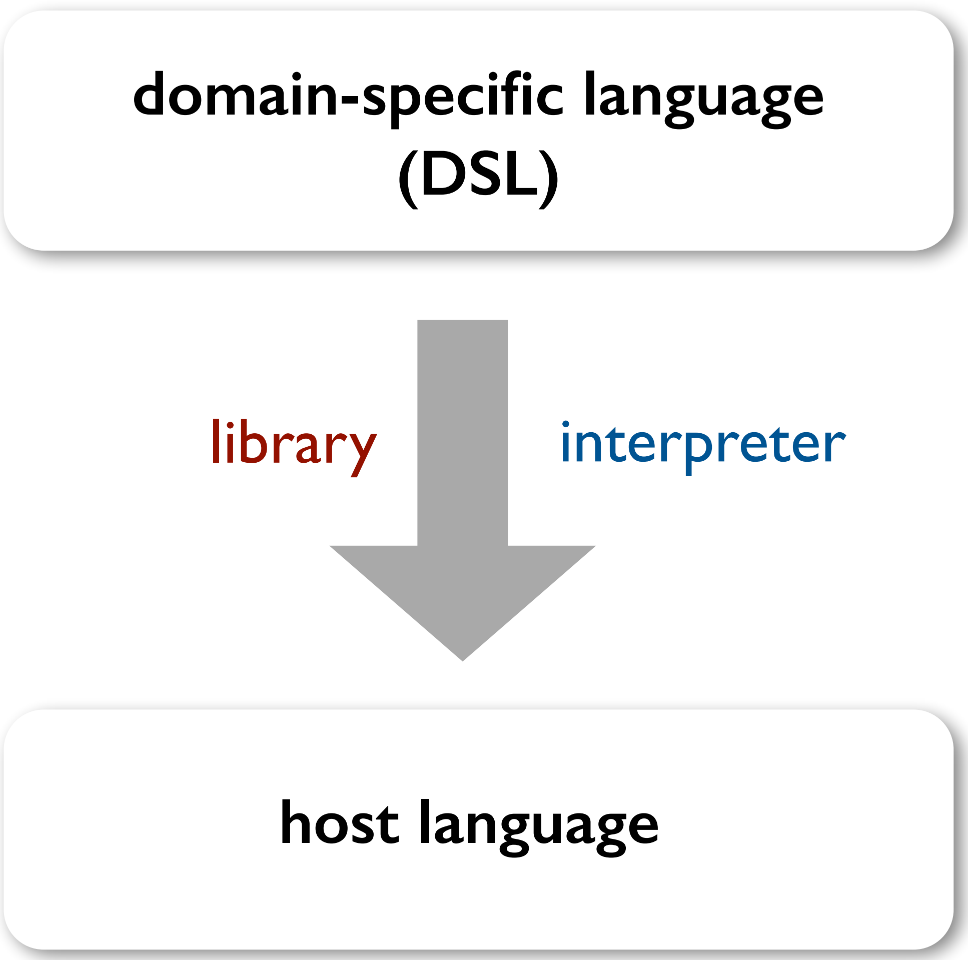
hardware design
Bluespec, Chisel, Verilog, VHDL

math and statistics
Eigen, Matlab, R

layout and visualization
LaTex, dot, dygraphs, D3

Scala, Racket, JavaScript

Layers of languages



Eigen / Matlab

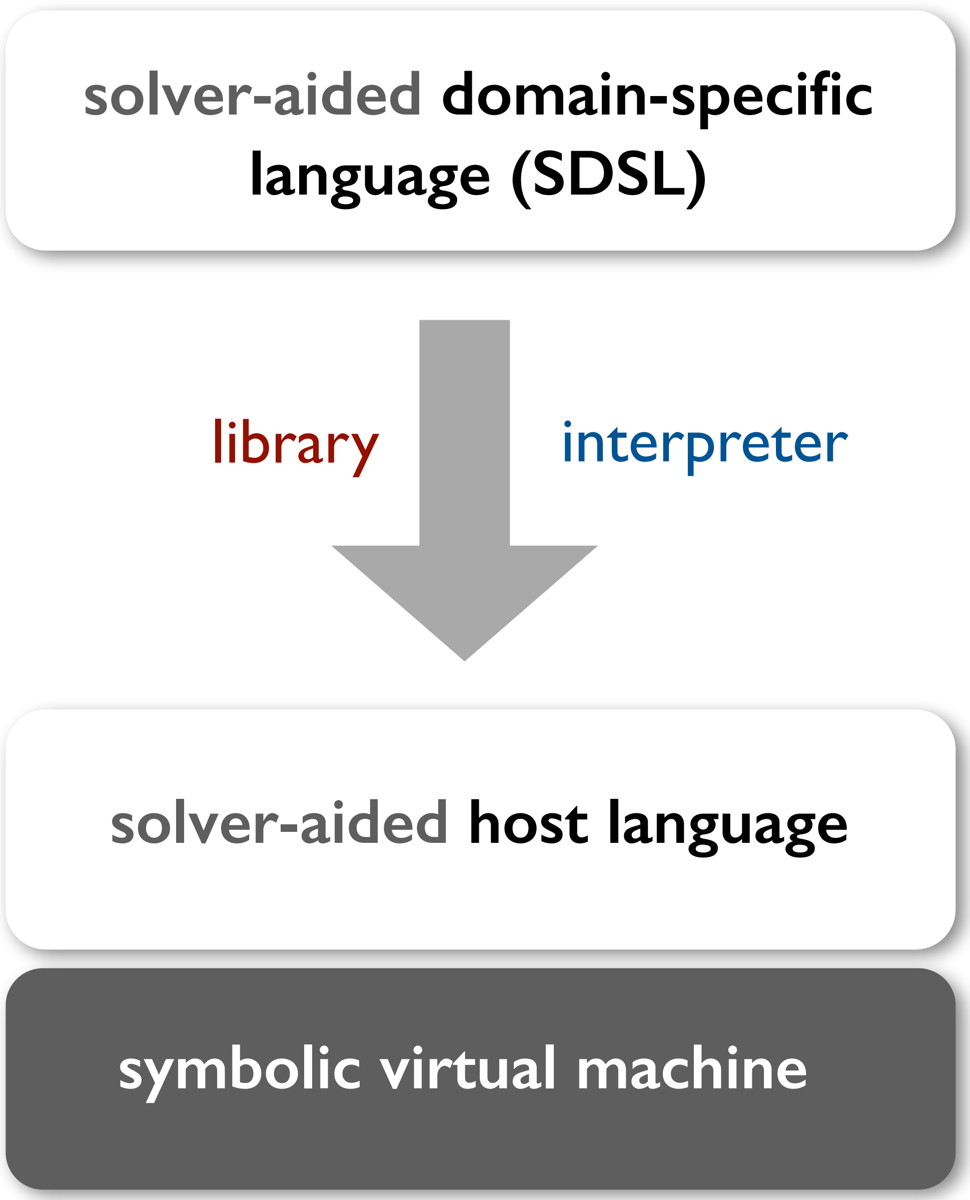
```
C = A * B
```

[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



Layers of solver-aided languages

solver-aided **domain-specific language (SDSL)**

library interpreter

solver-aided **host language**

symbolic virtual machine

ROSETTE

[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

ROSETTE

[Torlak & 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)
```

ROSETTE

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

```
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
```

test debug
verify synth

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
```

test **debug**
verify **synth**

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: ROSETTE

```
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: ROSETTE

```
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)
```

parse

```
(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: ROSETTE

```
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)
```

parse

```
(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: ROSETTE

```
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)
```

interpret

```
(define bvmax
 `((2 bvge 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 SDSL: ROSETTE

```
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)
```

interpret

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

0	-2
1	-1
2	
3	
4	
5	
6	

```
(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: ROSETTE

```
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	-2
1	-1
2	
3	
4	
5	
6	

interpret

```
(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: ROSETTE

```
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	-2
1	-1
2	
3	
4	
5	
6	

interpret

```
(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: ROSETTE

```
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
```

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

0	-2
1	-1
2	
3	
4	
5	
6	

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

interpret

```
(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: ROSETTE

```
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	-2
1	-1
2	0
3	
4	
5	
6	

interpret

```
(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: ROSETTE

```
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	−2
1	−1
2	0
3	0
4	−2
5	0
6	−1

interpret

```
(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: ROSETTE

```
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
```

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

0	-2
1	-1
2	0
3	0
4	-2
5	0
6	-1

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

interpret

```
(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: ROSETTE

```
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
```

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

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

```
(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: ROSETTE

```
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)
```

query

```
(define-symbolic n0 n1 integer?)
(define inputs (list n0 n1))
(verify
  (assert (= (interpret bvmax inputs)
             (interpret max inputs))))
```

A tiny example SDSL: ROSETTE

```
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)
```

query

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: ROSETTE

```
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)
```

query

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

```
(define-symbolic n0 n1 integer?)
(define inputs (list n0 n1))
(verify
  (assert (= (interpret bvmax inputs)
             (interpret max inputs))))
```

A tiny example SDSL: ROSETTE

```
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)
```

query

```
(define-symbolic n0 n1 integer?)
(define inputs (list n0 n1))
(verify
  (assert (= (interpret bvmax inputs)
             (interpret max inputs))))
```

(verify *expr*) searches for a concrete interpretation of symbolic constants that causes *expr* to fail.

A tiny example SDSL: ROSETTE

```
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
```

query

```
(define-symbolic n0 n1 integer?)
(define inputs (list n0 n1))
(verify
  (assert (= (interpret bvmax inputs)
             (interpret max inputs))))
```

A tiny example SDSL: ROSETTE

```
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))
```

query

```
(define inputs (list 0 -2))
(debug [input-register?]
  (assert (= (interpret bvmax inputs)
             (interpret max inputs))))
```

A tiny example SDSL: ROSETTE

```
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))
```

query

```
(define inputs (list 0 -2))
(debug [input-register?]
  (assert (= (interpret bvmax inputs)
             (interpret max inputs))))
```

A tiny example SDSL: ROSETTE

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

> synthesize(bvmax, max)
```

query

```
(define-symbolic n0 n1 integer?)
(define inputs (list n0 n1))
(synthesize [inputs]
  (assert (= (interpret bvmax inputs)
             (interpret max inputs)))))
```

A tiny example SDSL: ROSETTE

```
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)
```

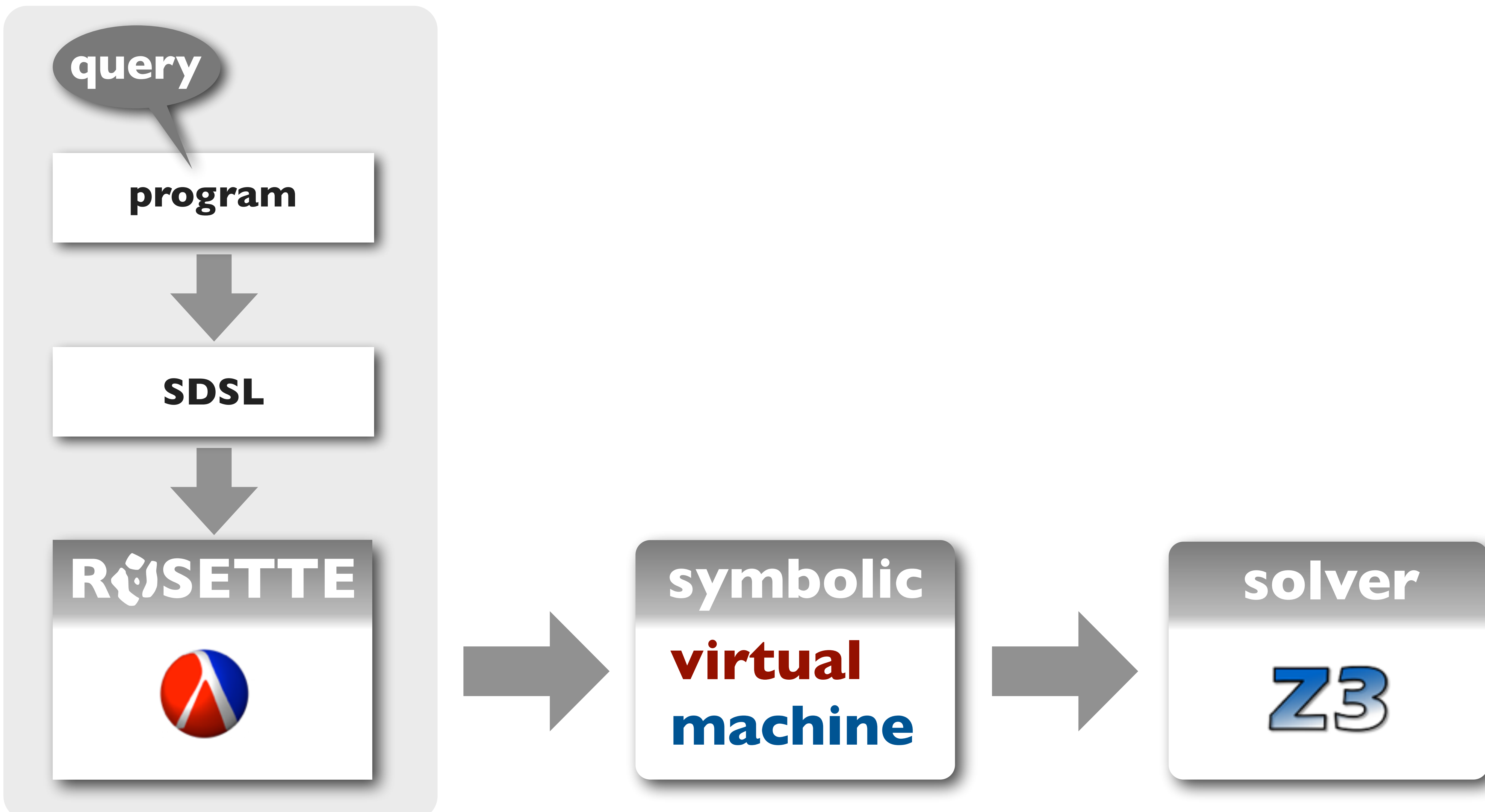
query

```
(define-symbolic n0 n1 integer?)
(define inputs (list n0 n1))
(synthesize [inputs]
  (assert (= (interpret bvmax inputs)
             (interpret max inputs))))
```

tech

symbolic virtual machine (SVM)

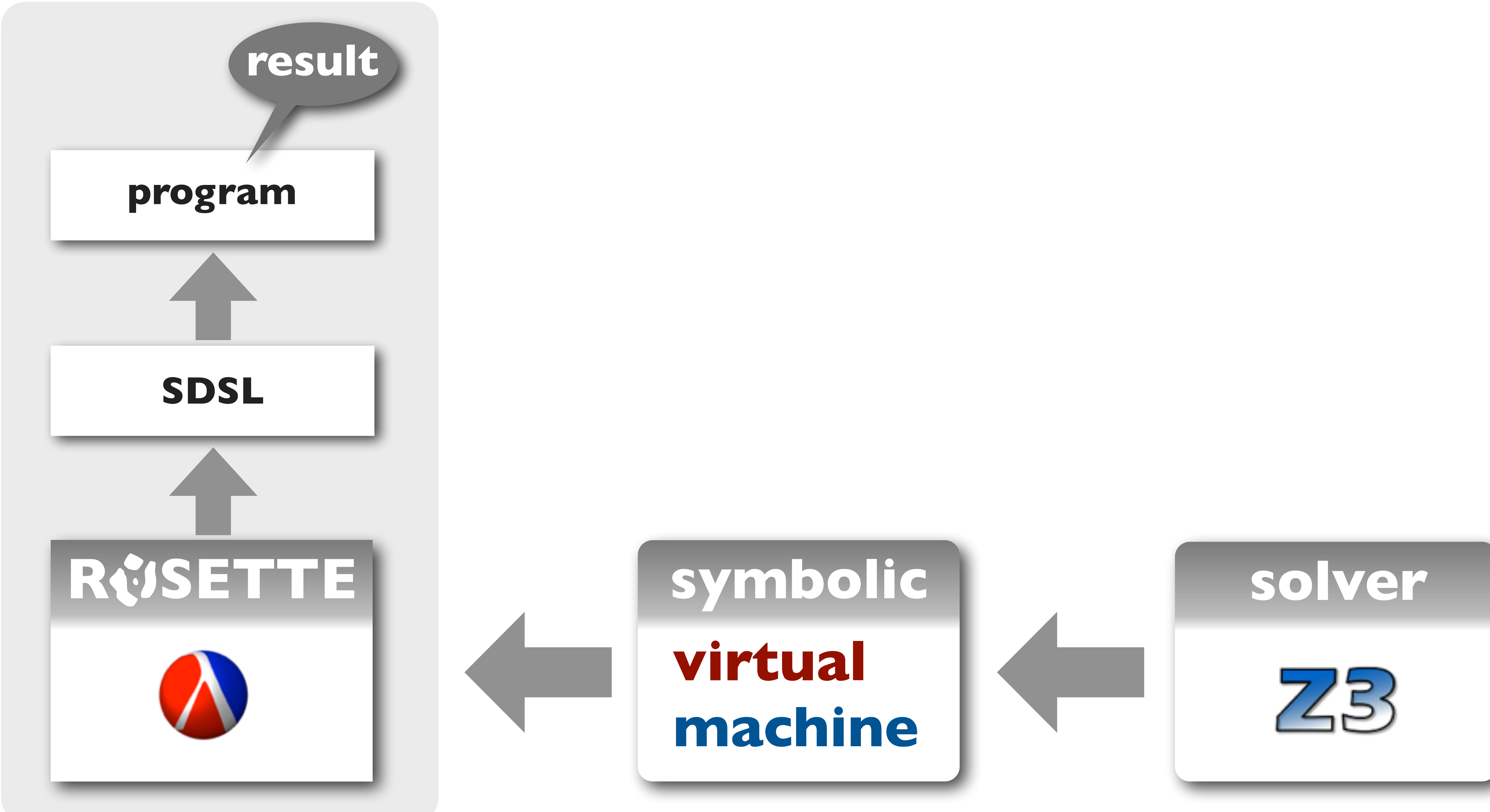
How it all works: a big picture view



[Torlak & Bodik, **Onward'13**]

[Torlak & Bodik, **PLDI'14**]

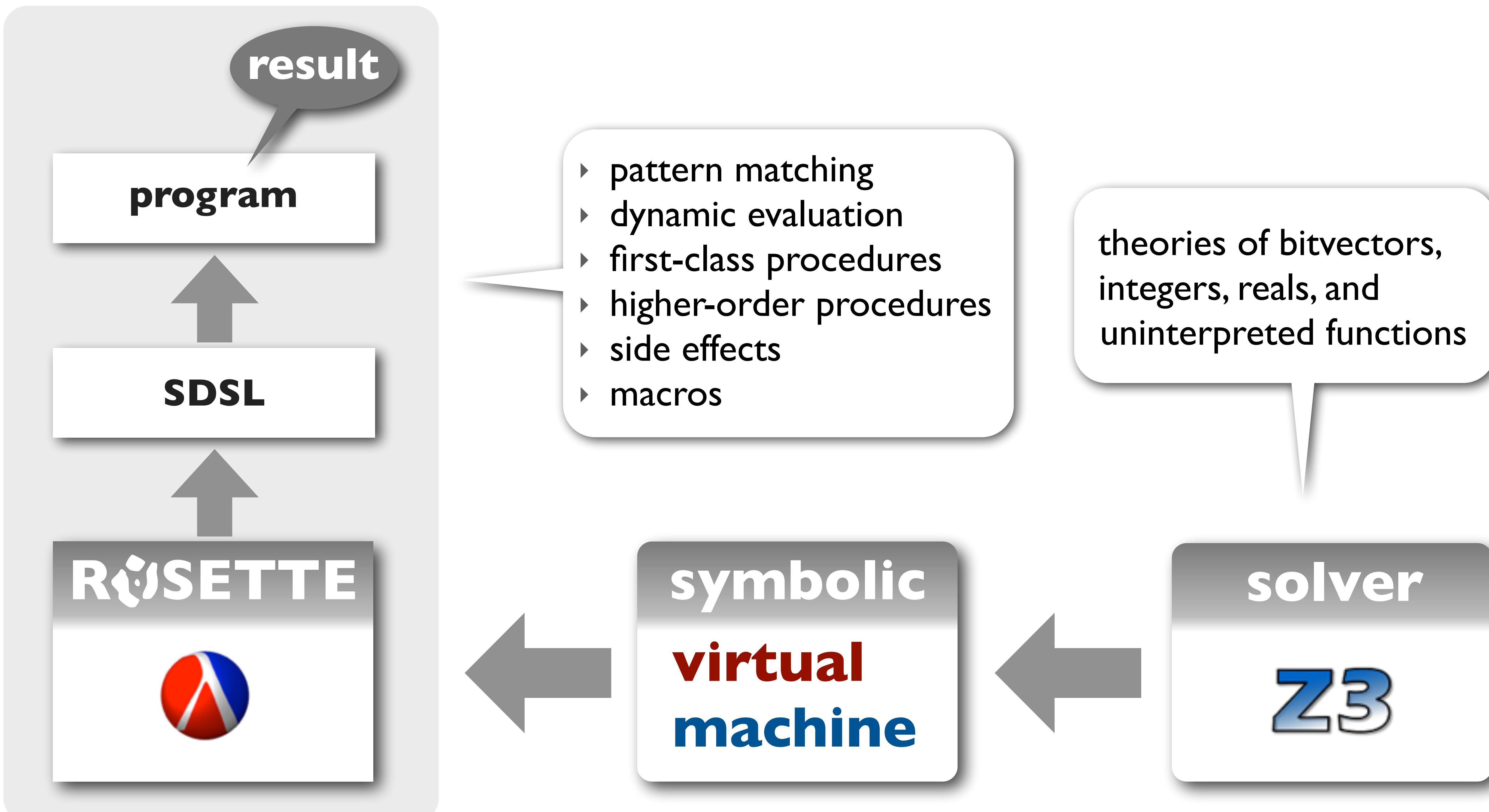
How it all works: a big picture view



[Torlak & Bodik, **Onward'13**]

[Torlak & Bodik, **PLDI'14**]

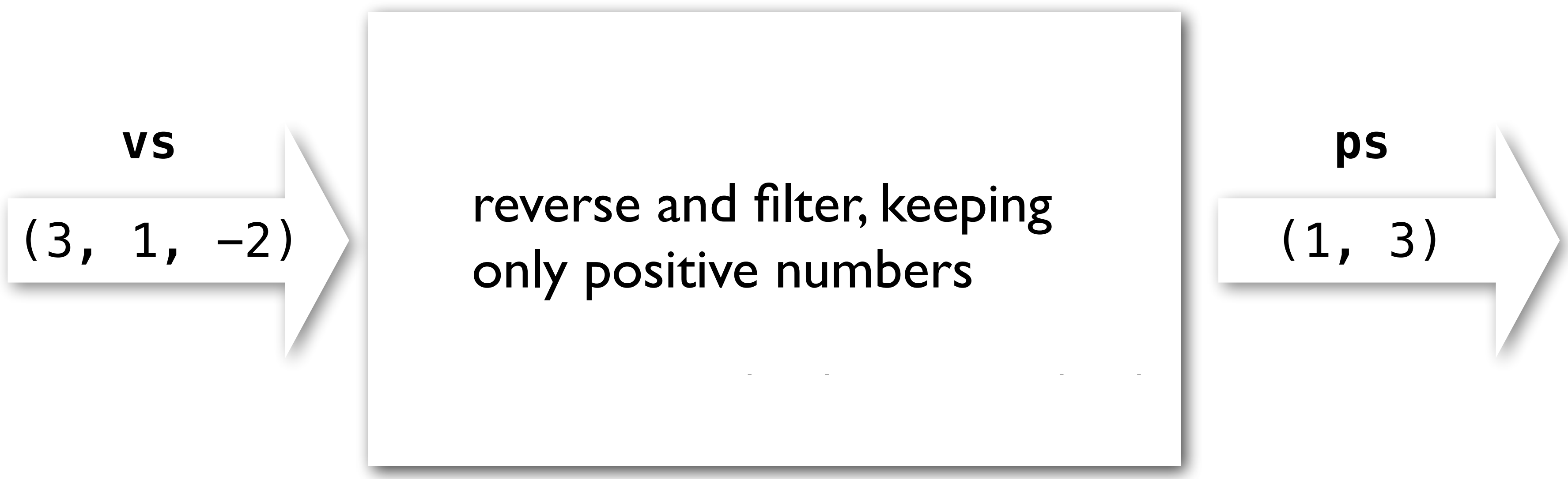
How it all works: a big picture view



[Torlak & Bodik, **Onward'13**]

[Torlak & Bodik, **PLDI'14**]

Translation to constraints by example



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

vs

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

constraints

Translation to constraints by example

vs

(a, b)

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

constraints

Translation to constraints by example

vs

(a, b)

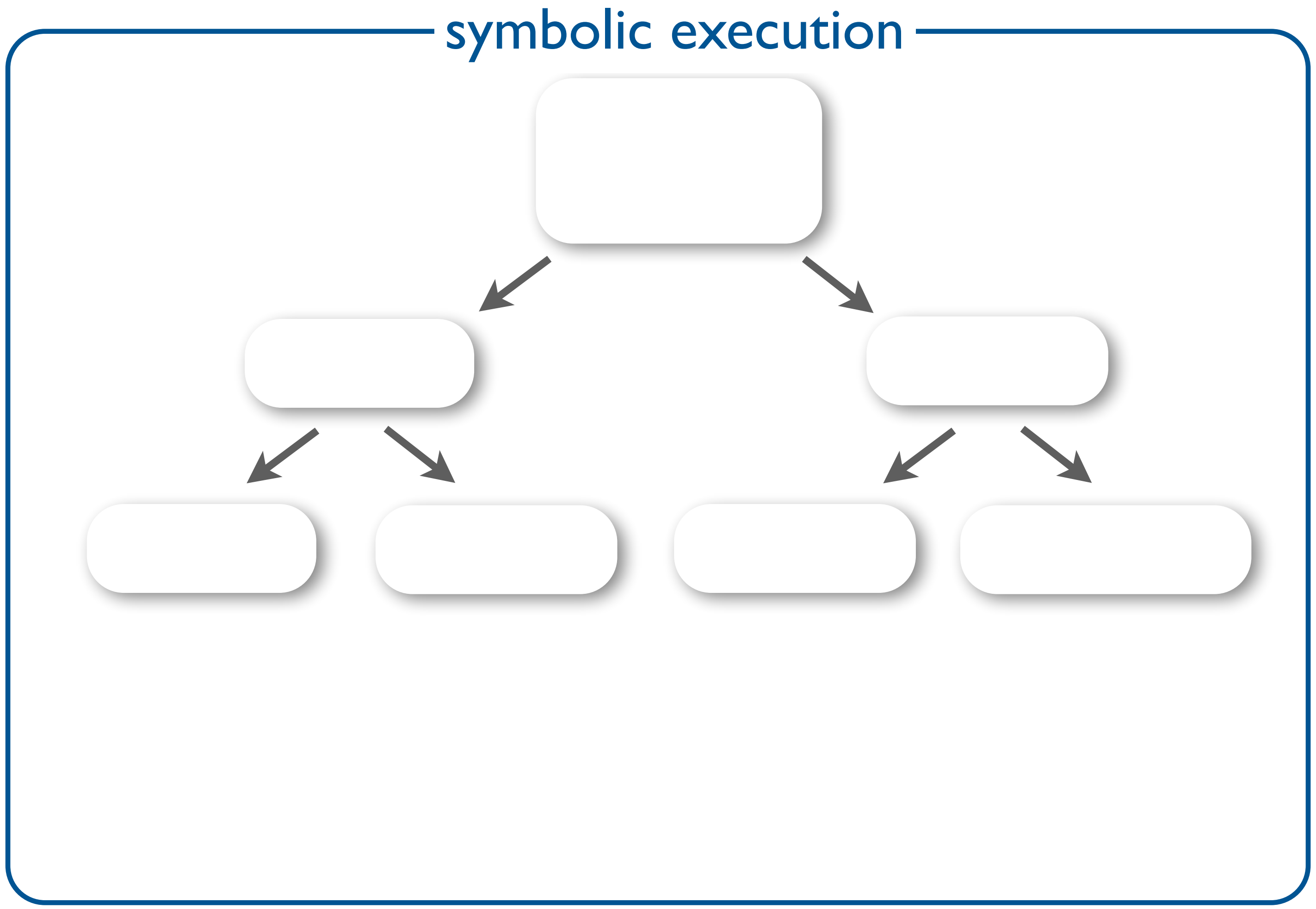
```
solve:
  ps = ()
  for v in vs:
    if v > 0:
      ps = insert(v, ps)
  assert len(ps) == len(vs)
```

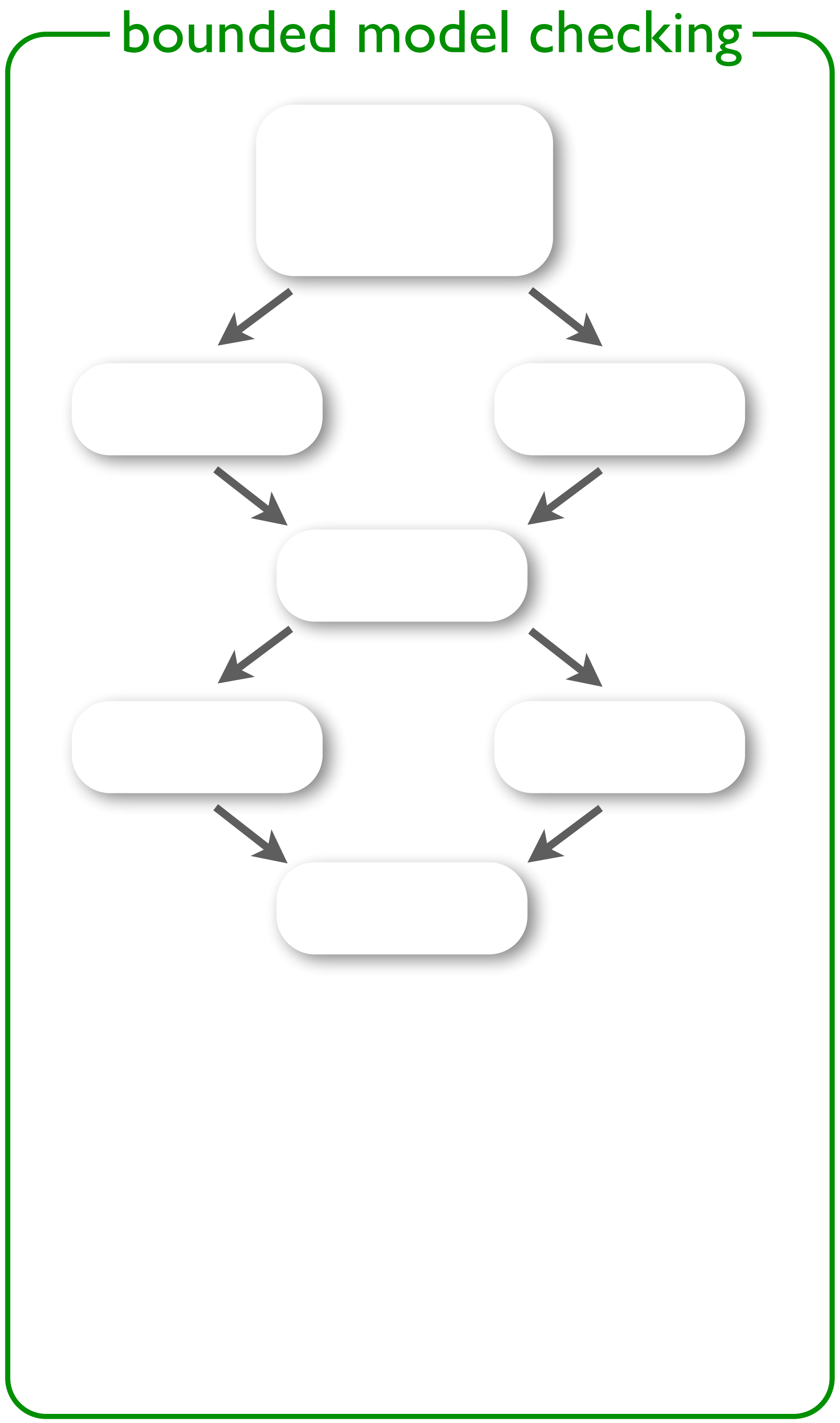
constraints

a>0 ∧ b>0

Design space of precise symbolic encodings

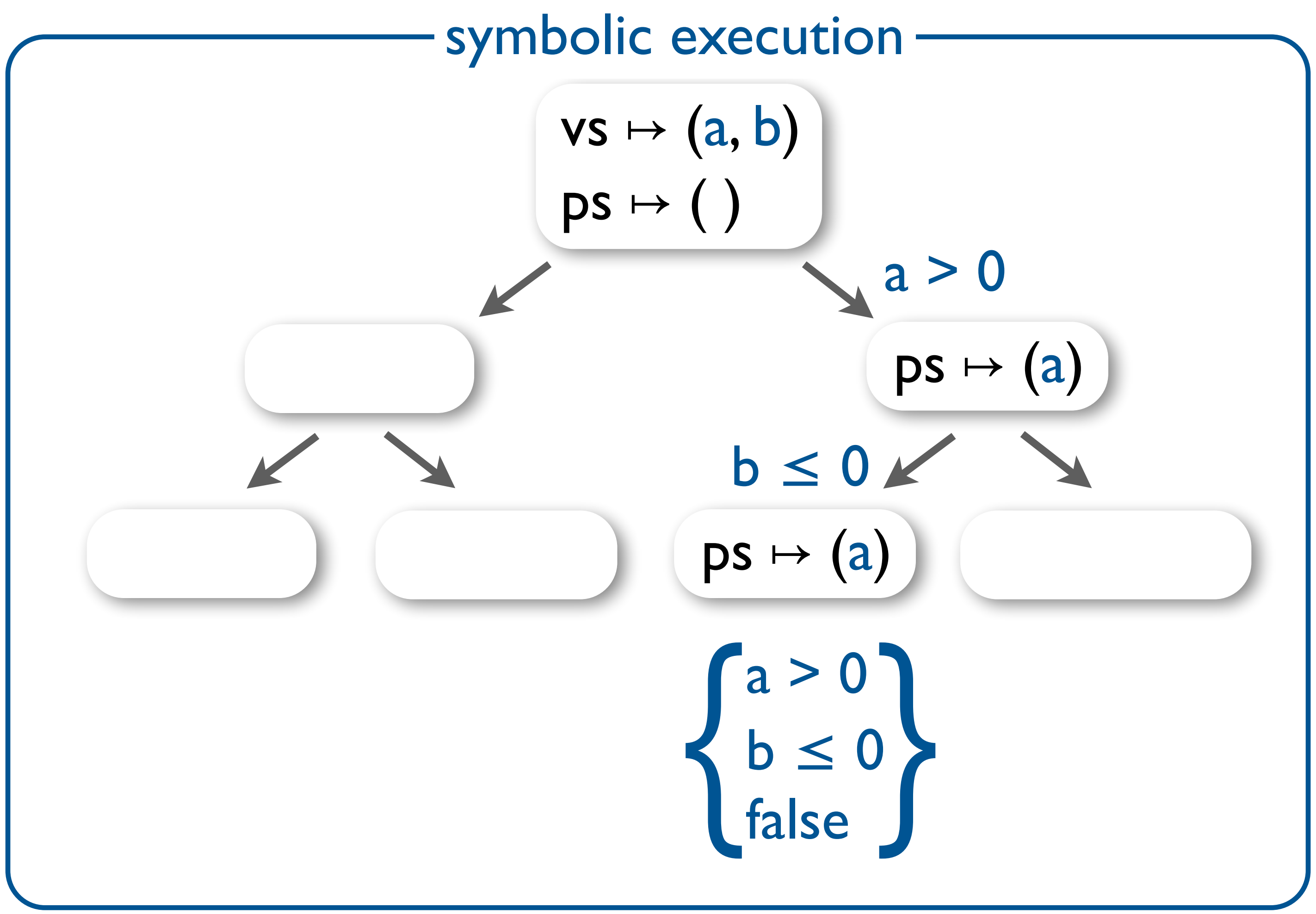
```
solve:
  ps = ()
  for v in vs:
    if v > 0:
      ps = insert(v, ps)
  assert len(ps) == len(vs)
```

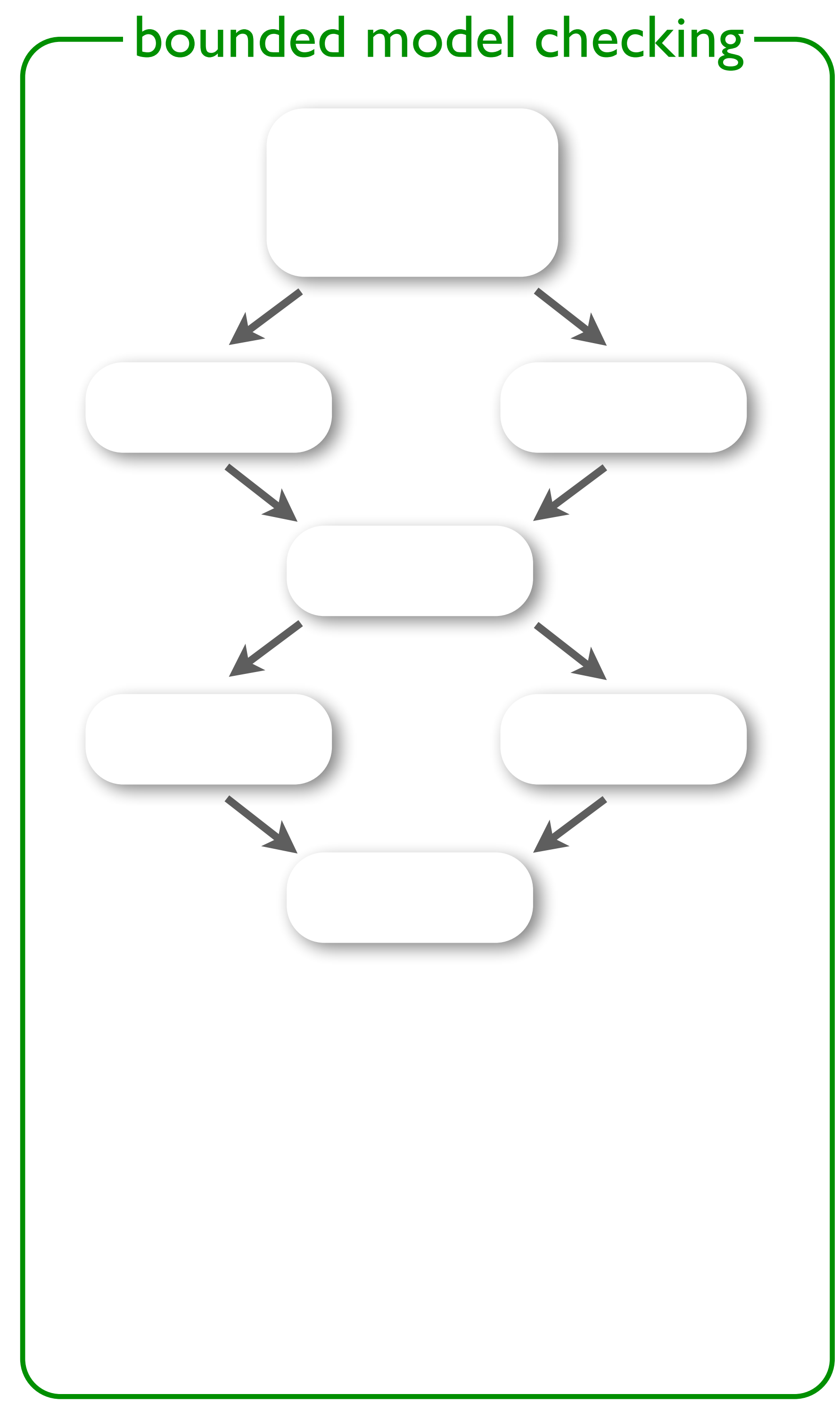




Design space of precise symbolic encodings

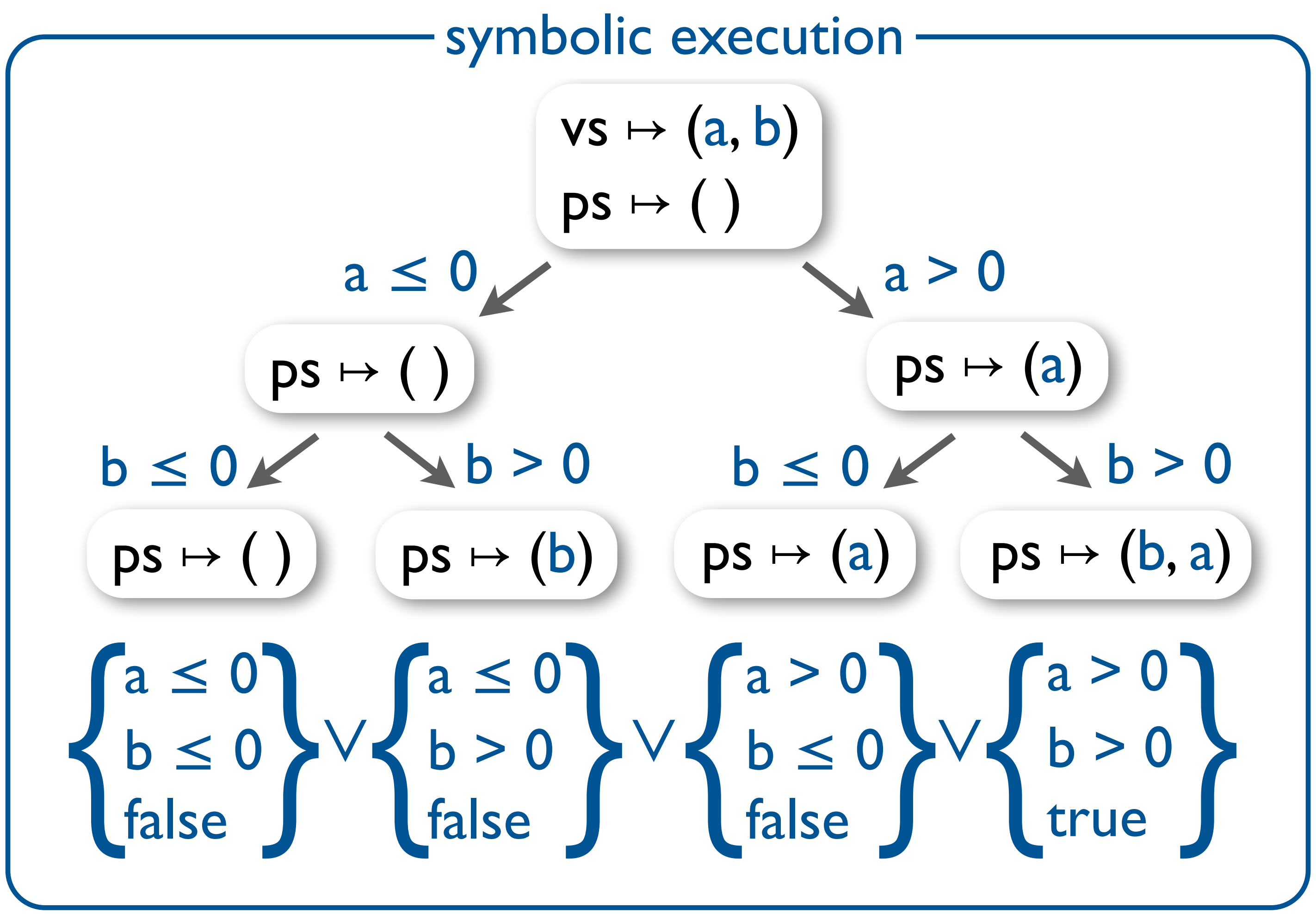
```
solve:
  ps = ()
  for v in vs:
    if v > 0:
      ps = insert(v, ps)
  assert len(ps) == len(vs)
```

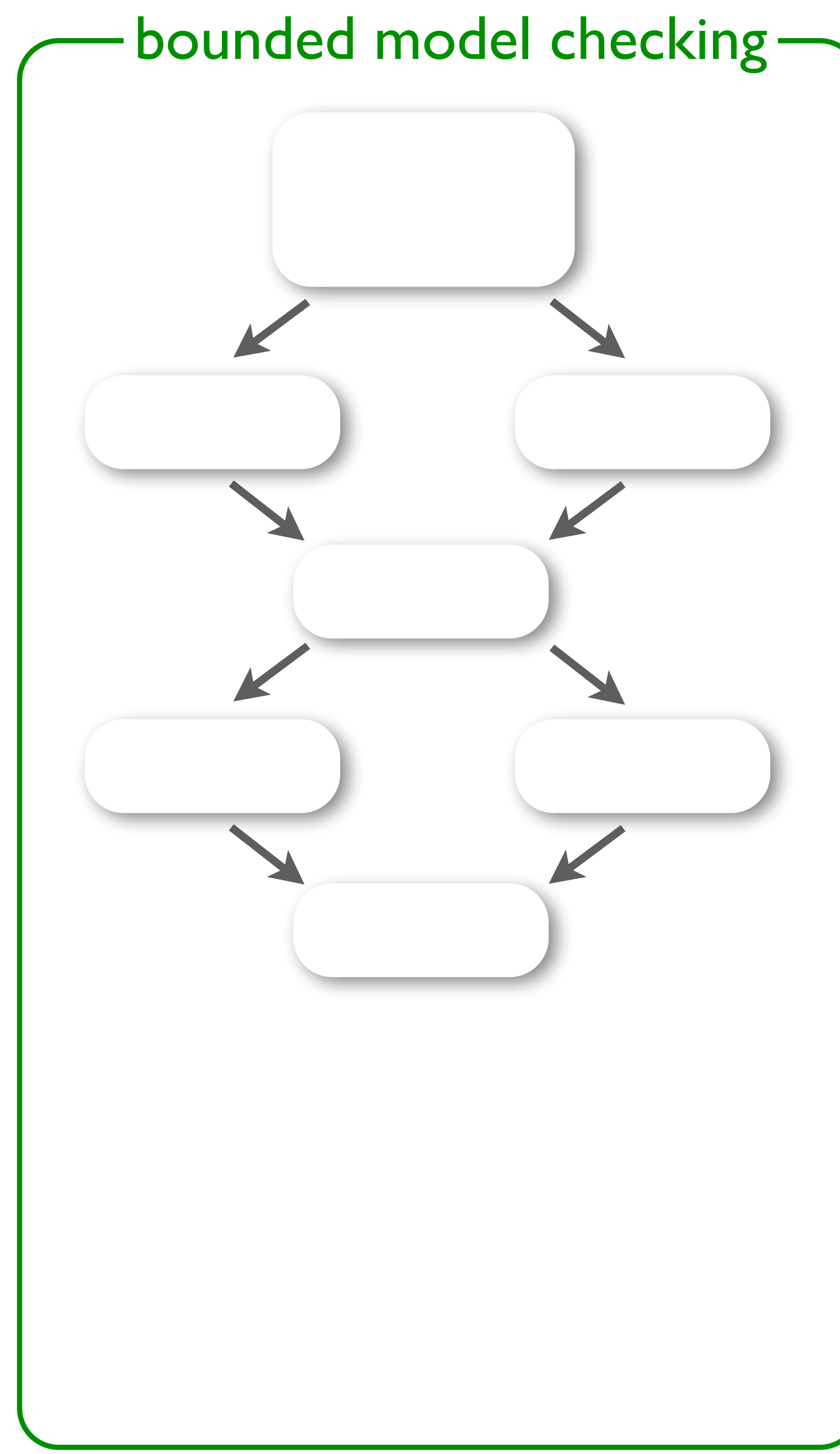




Design space of precise symbolic encodings

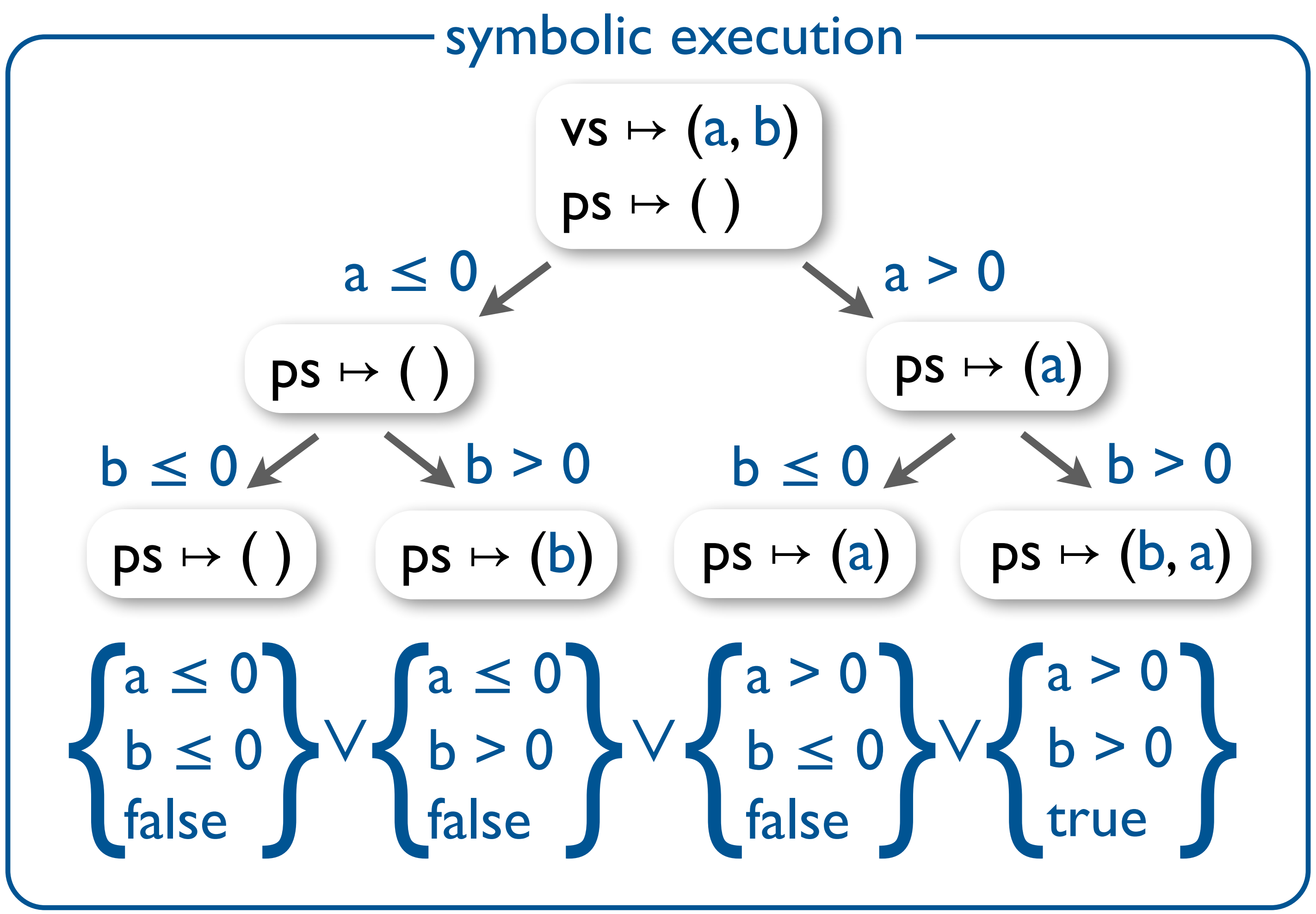
```
solve:
  ps = ()
  for v in vs:
    if v > 0:
      ps = insert(v, ps)
  assert len(ps) == len(vs)
```

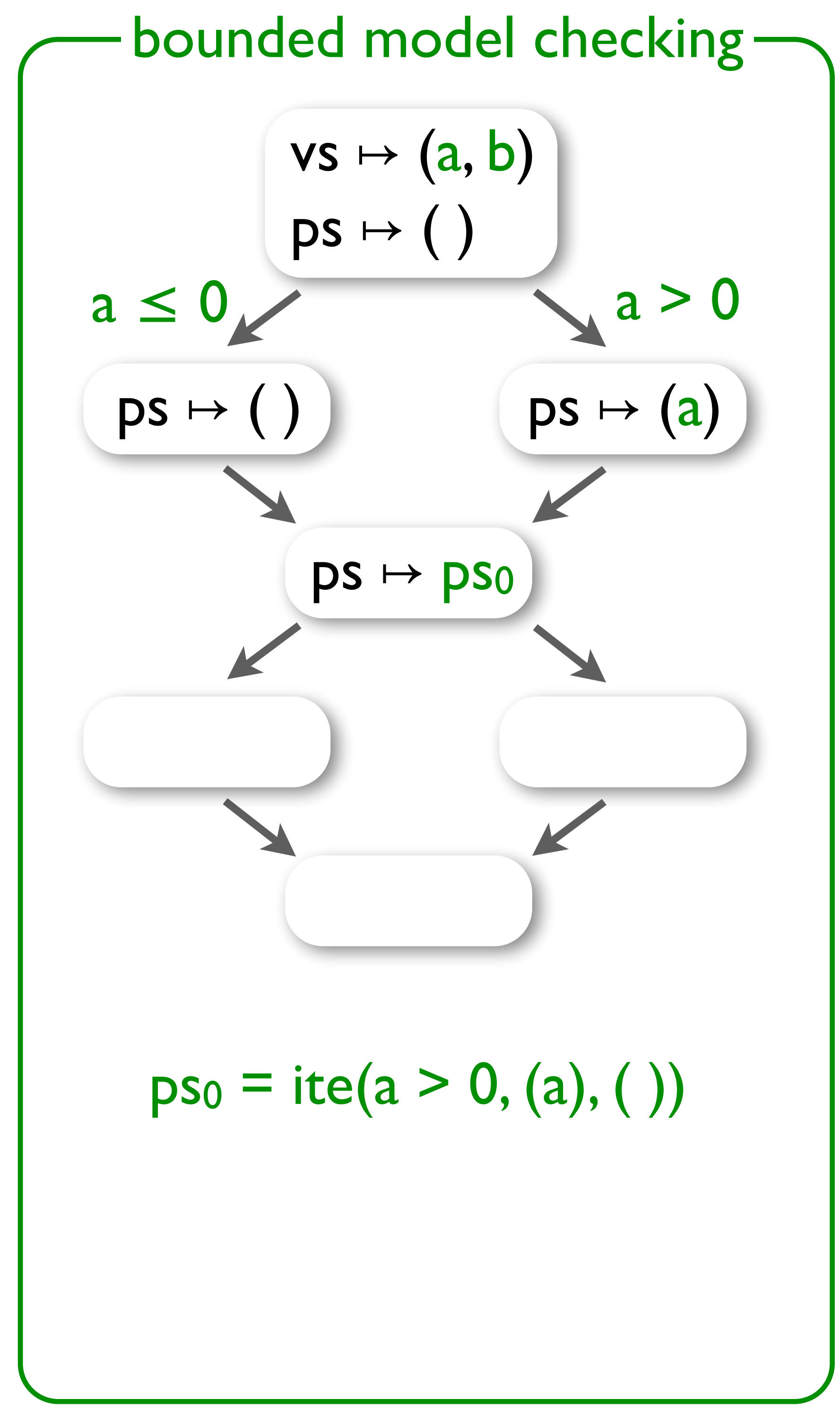




Design space of precise symbolic encodings

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

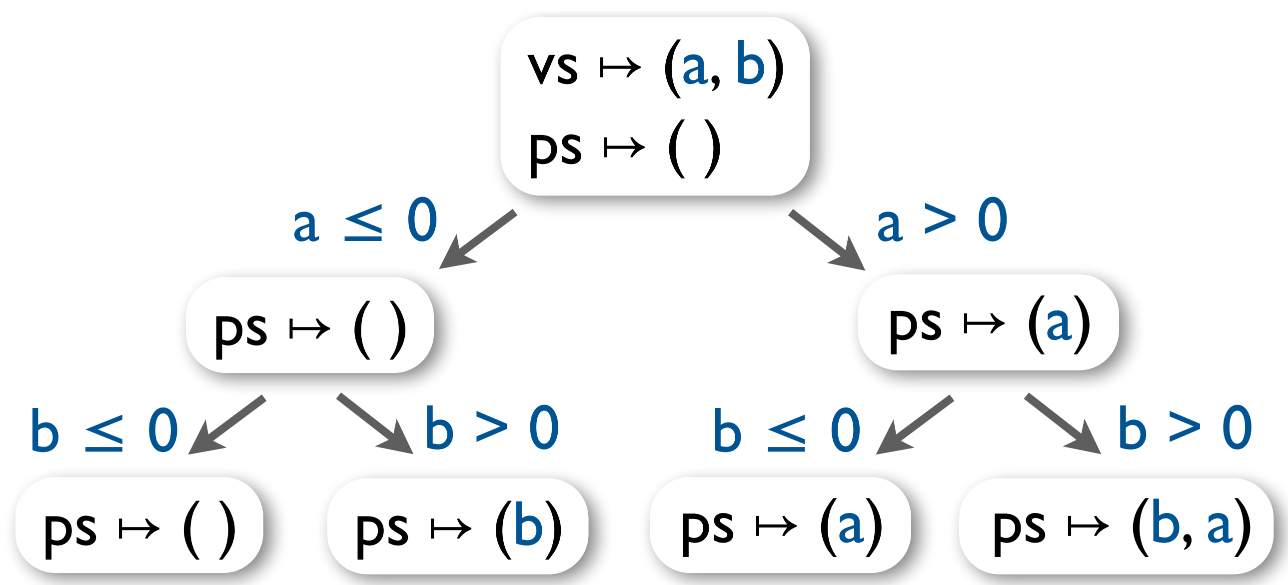




Design space of precise symbolic encodings

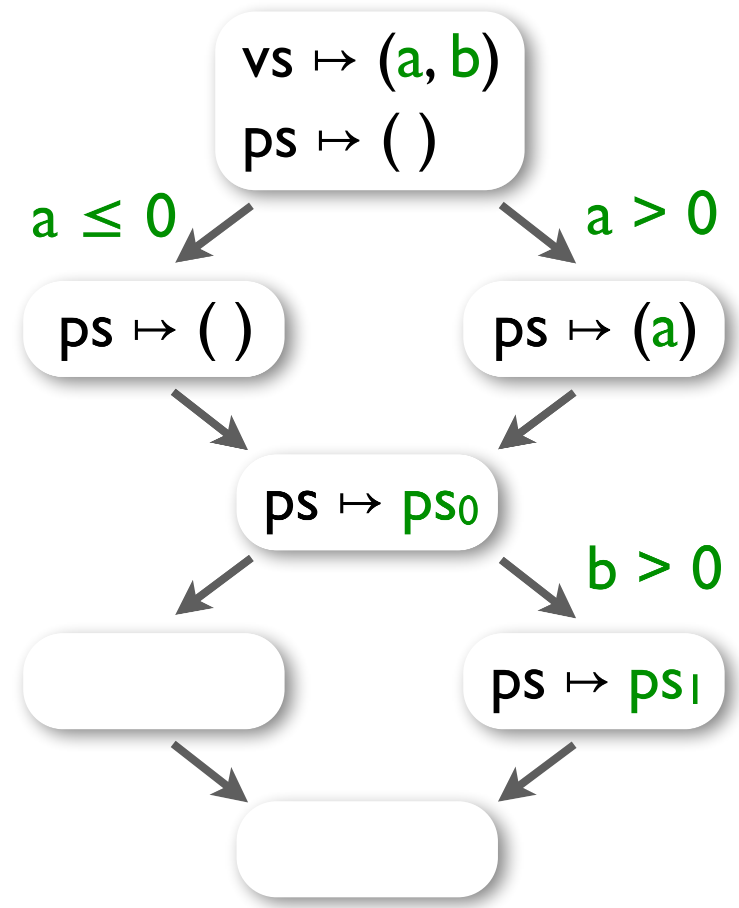
```
solve:
  ps = ()
  for v in vs:
    if v > 0:
      ps = insert(v, ps)
  assert len(ps) == len(vs)
```

symbolic execution



$$\left\{\begin{array}{l} a \leq 0 \\ b \leq 0 \\ \text{false} \end{array}\right\} \vee \left\{\begin{array}{l} a \leq 0 \\ b > 0 \\ \text{false} \end{array}\right\} \vee \left\{\begin{array}{l} a > 0 \\ b \leq 0 \\ \text{false} \end{array}\right\} \vee \left\{\begin{array}{l} a > 0 \\ b > 0 \\ \text{true} \end{array}\right\}$$

bounded model checking

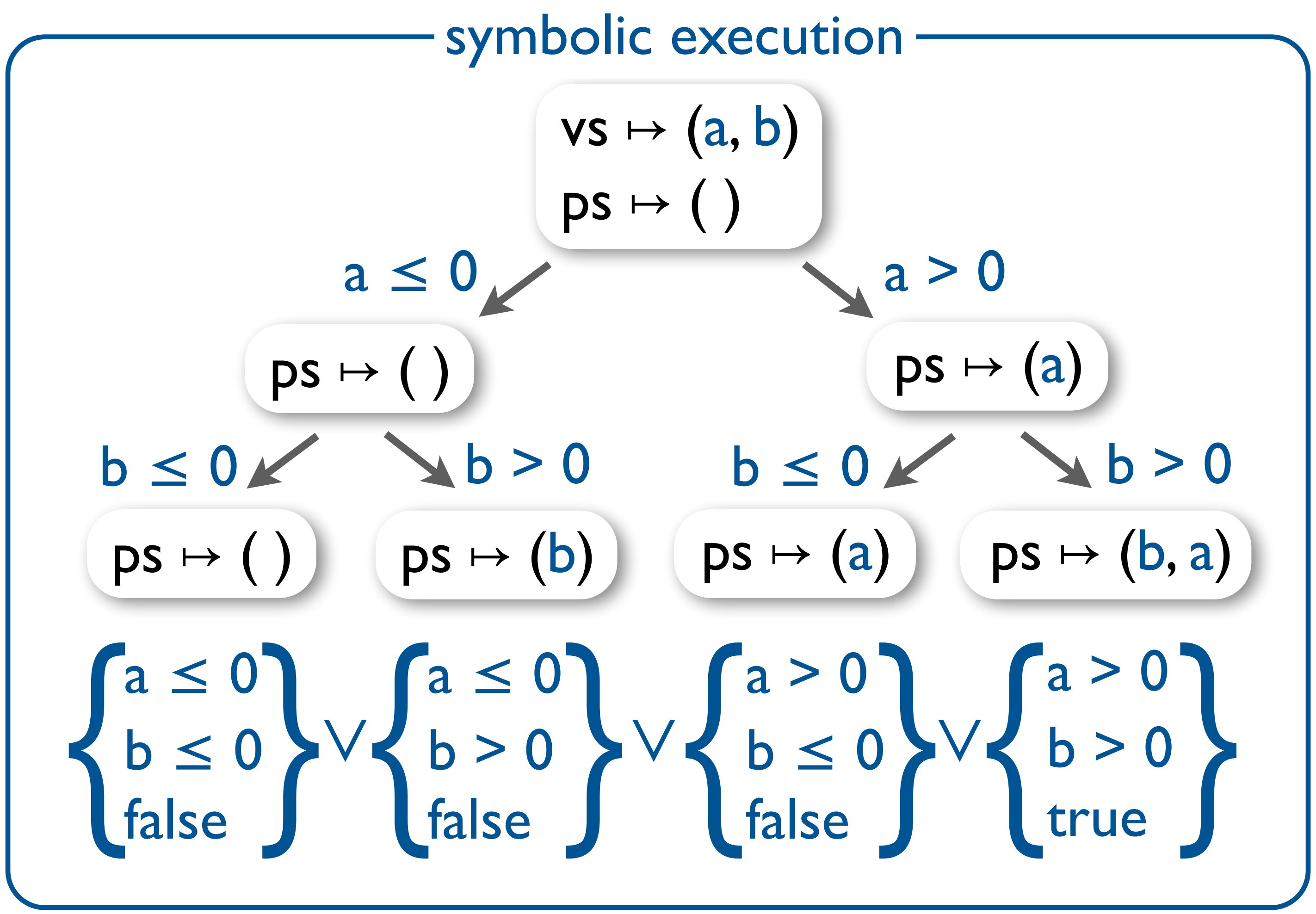


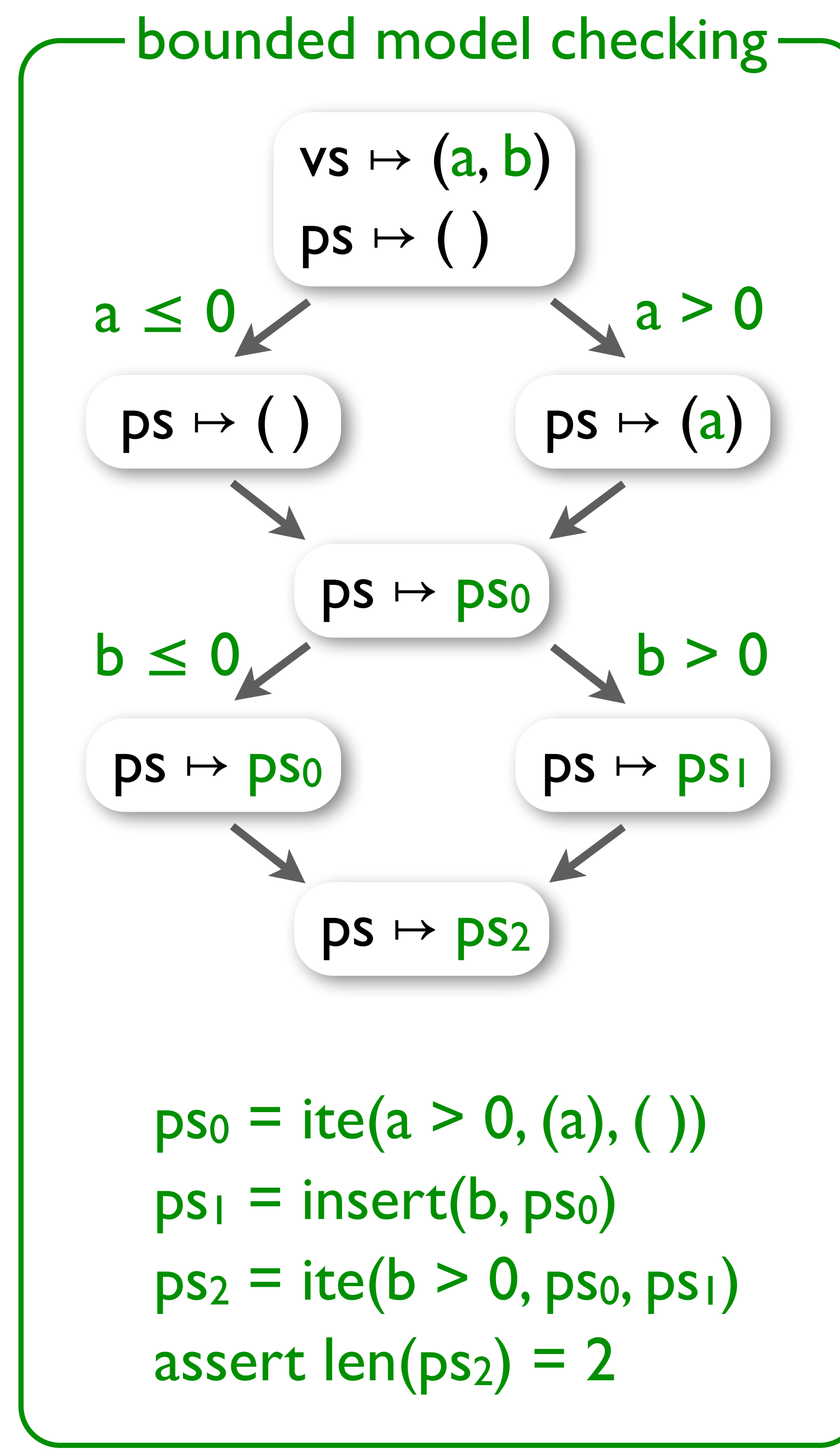
$ps_0 = \text{ite}(a > 0, (a), (\,))$

$ps_1 = \text{insert}(b, ps_0)$

Design space of precise symbolic encodings

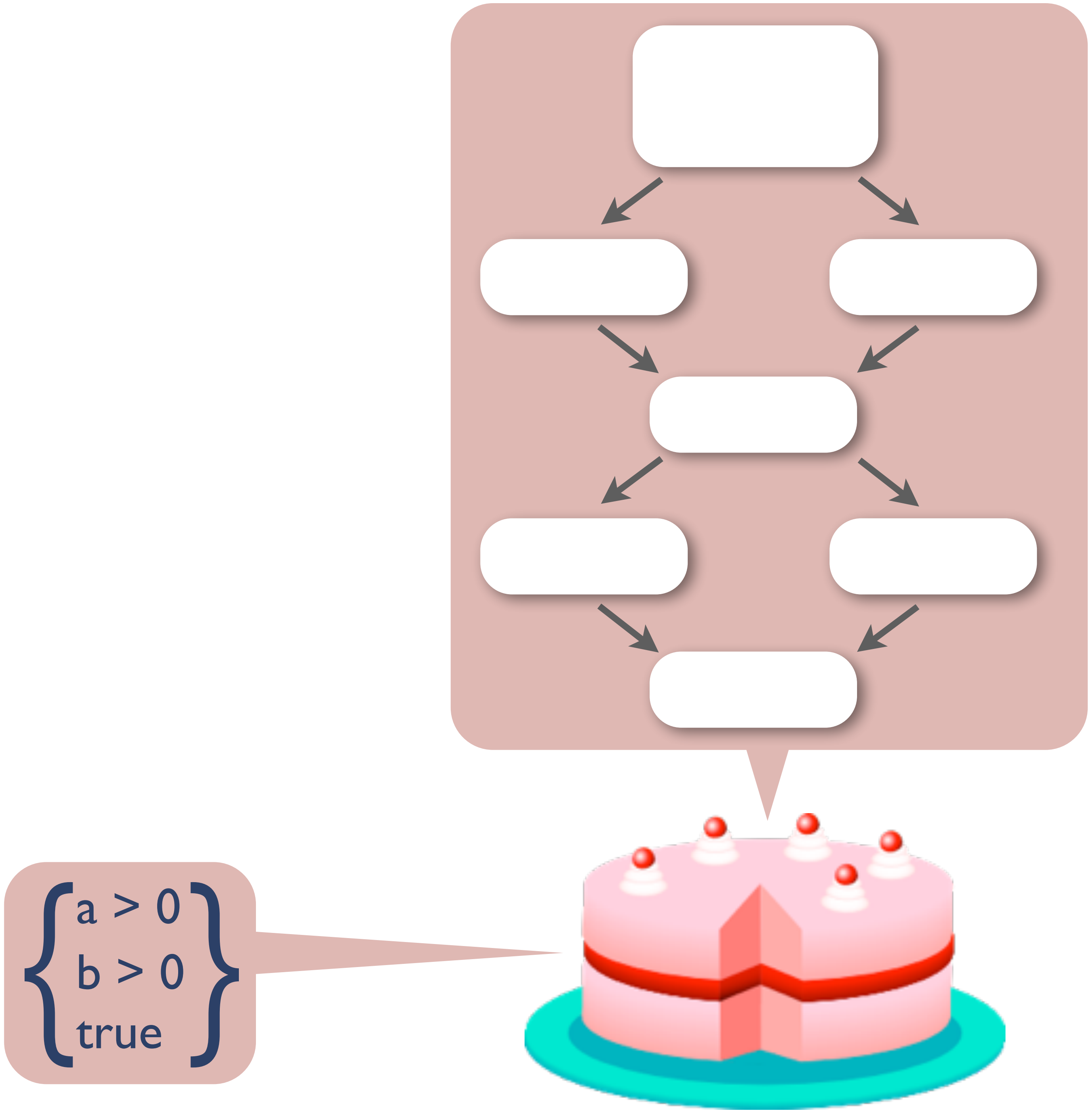
```
solve:
  ps = ()
  for v in vs:
    if v > 0:
      ps = insert(v, ps)
  assert len(ps) == len(vs)
```





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)
```

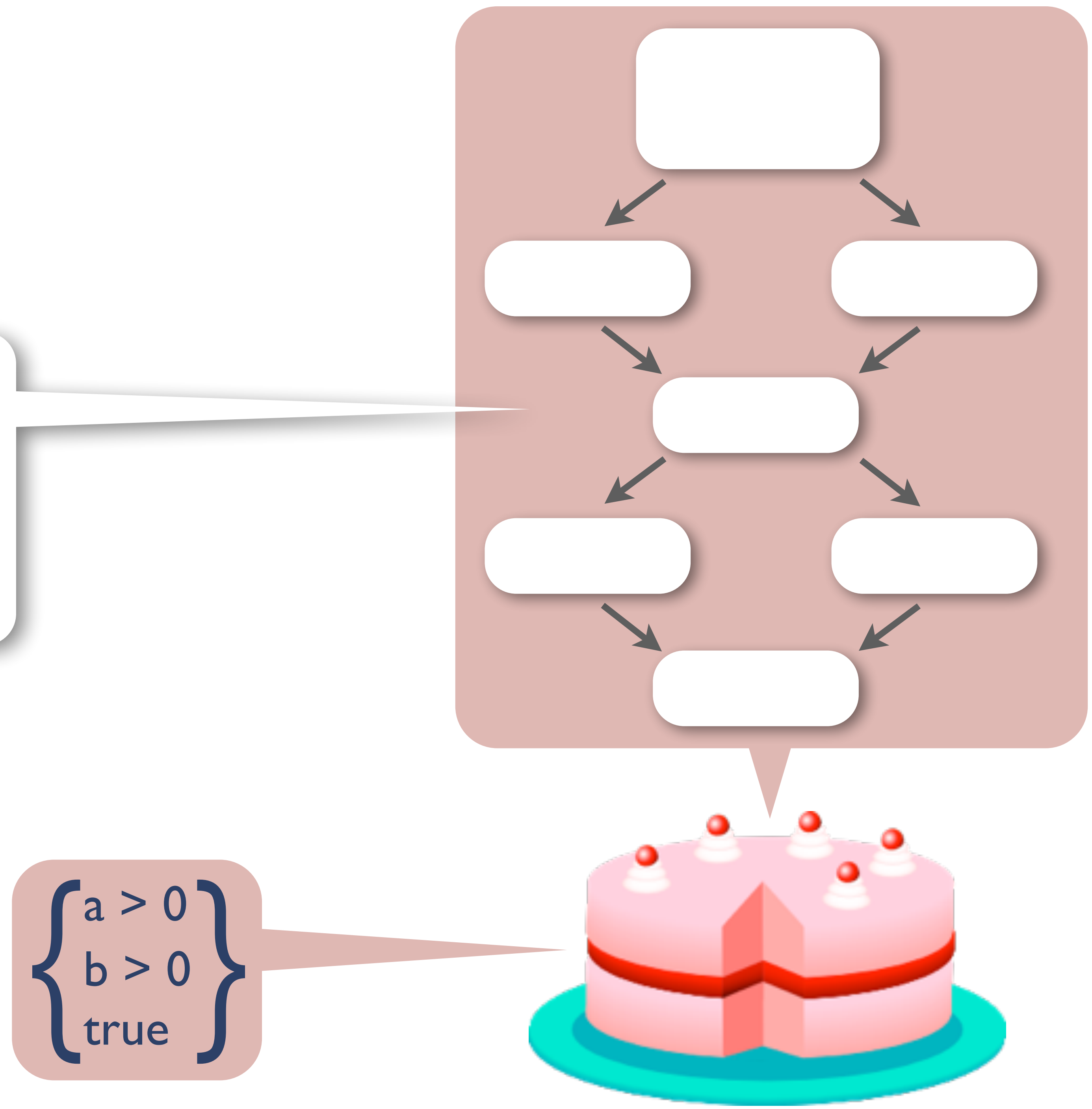


A new design: type-driven state merging

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

Merge values of

- primitive types: symbolically
- immutable types: structurally
- all other types: via unions

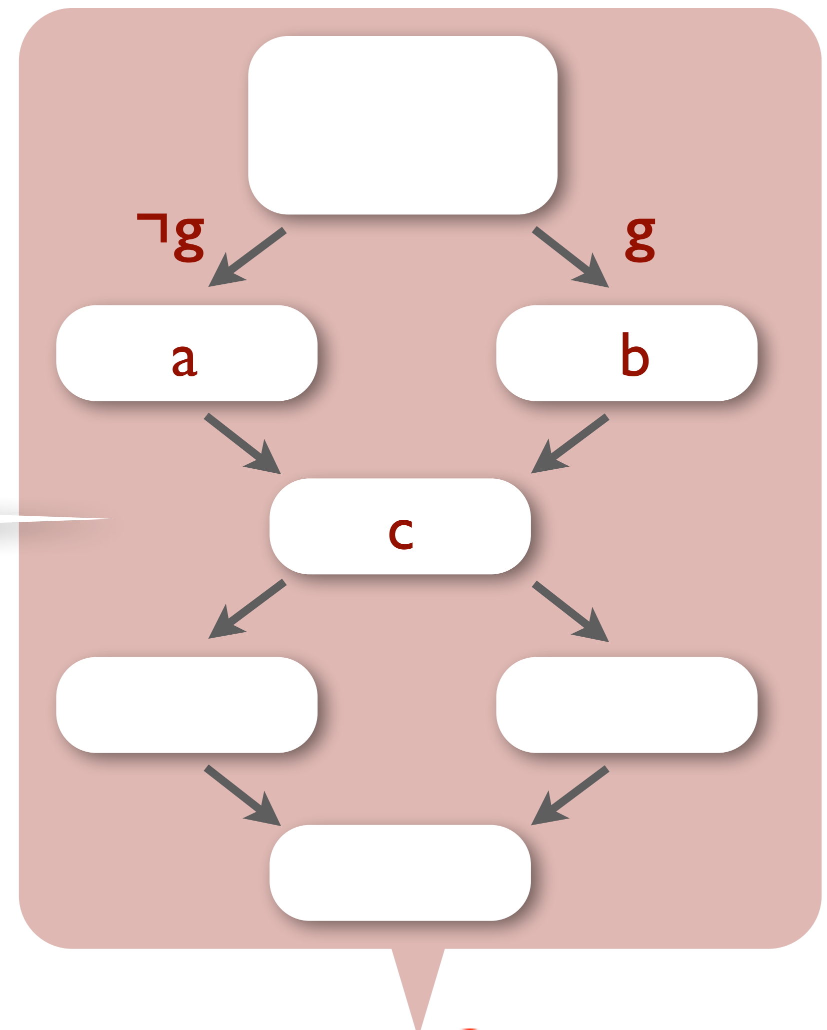


A new design: type-driven state merging

```
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



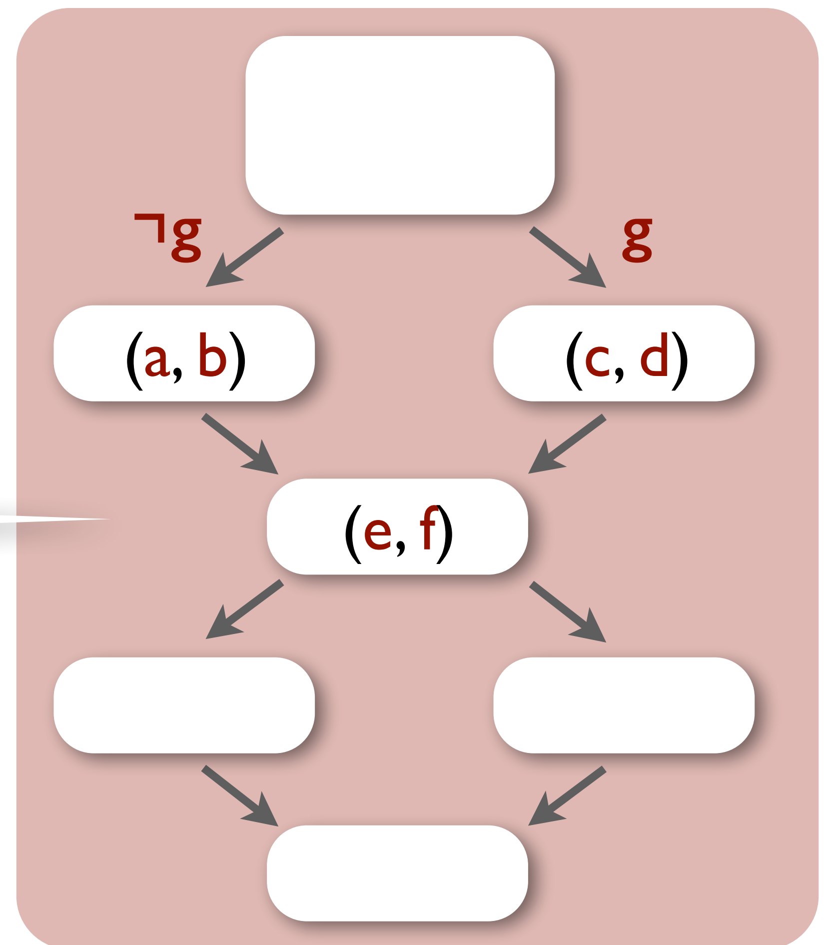
$$\left\{\begin{array}{l} a > 0 \\ b > 0 \\ \text{true} \end{array}\right\}$$

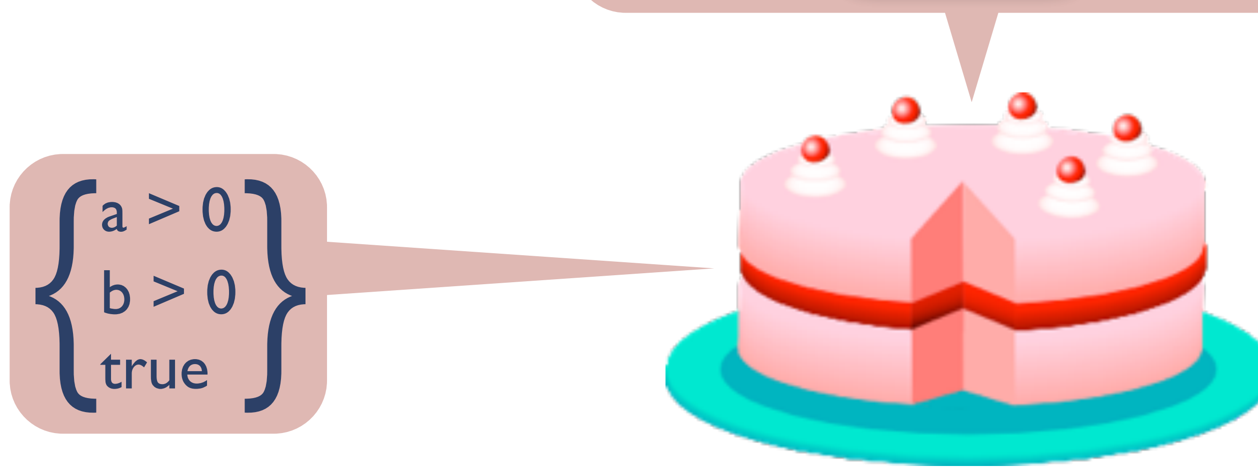
A new design: type-driven state merging

```
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



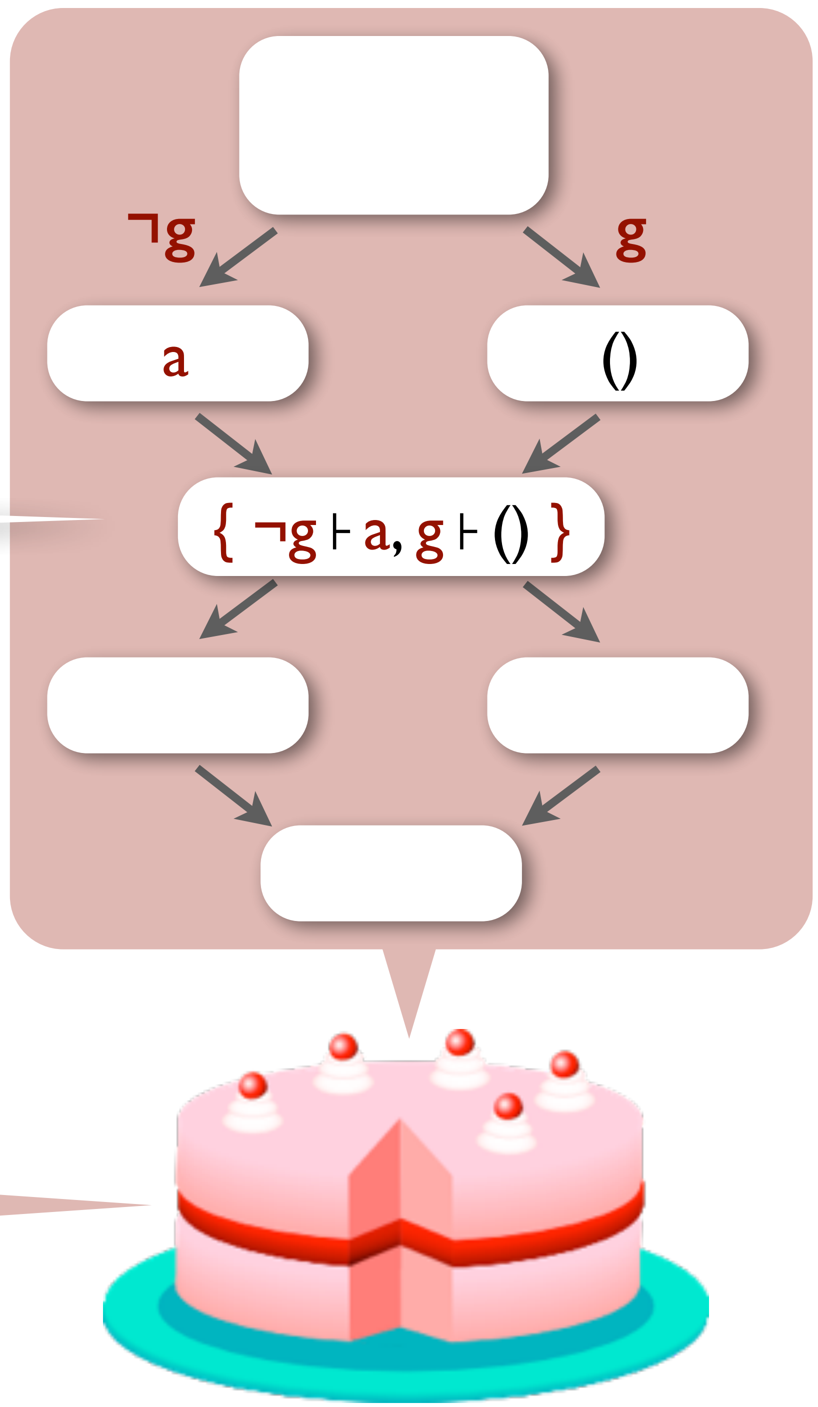


A new design: type-driven state merging

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

Merge values of

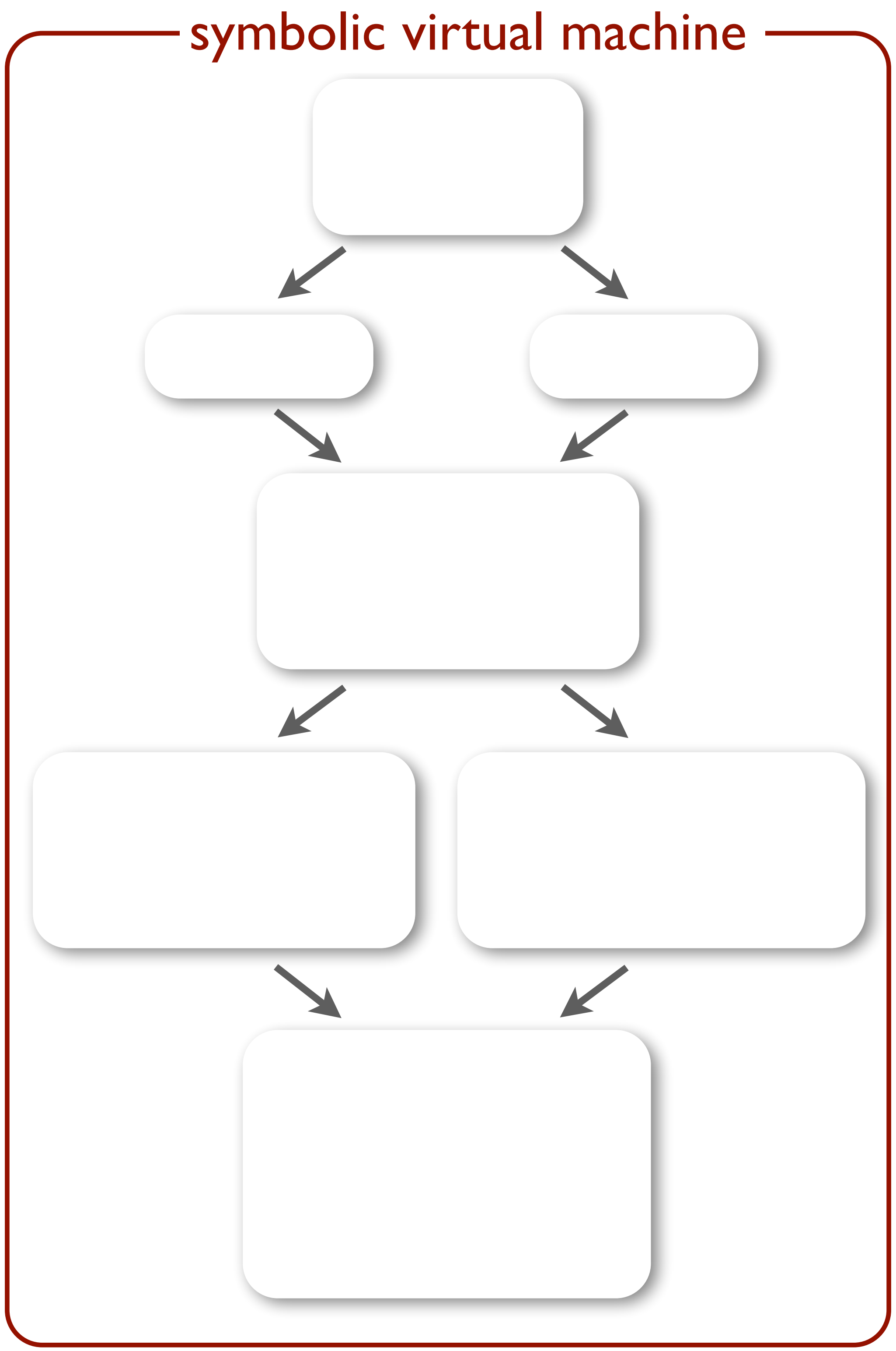
- primitive types: symbolically
- immutable types: structurally
- all other types: via unions



$\left\{\begin{matrix} a > 0 \\ b > 0 \\ true \end{matrix}\right\}$

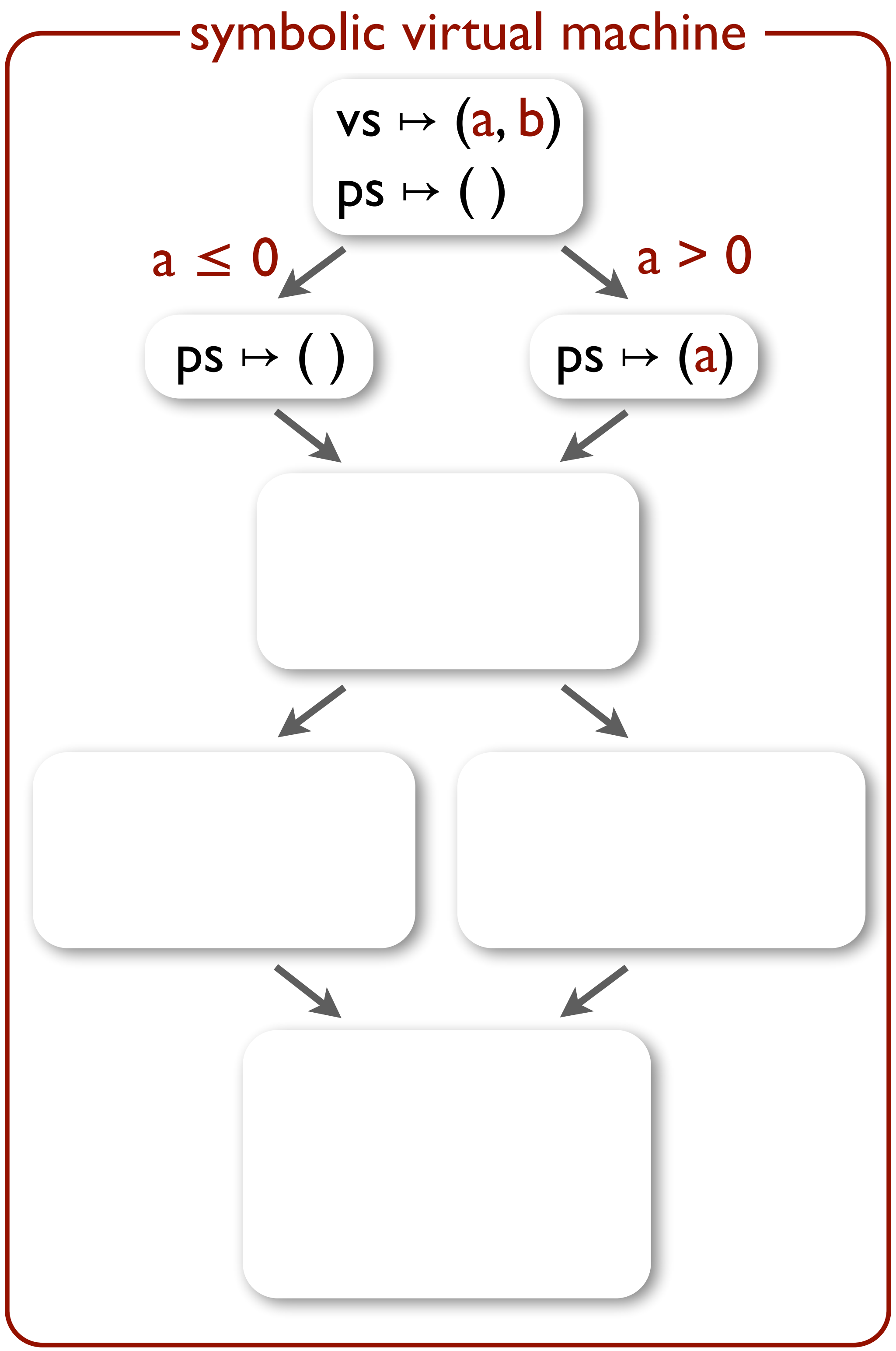
A new design: type-driven state merging

```
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)
```

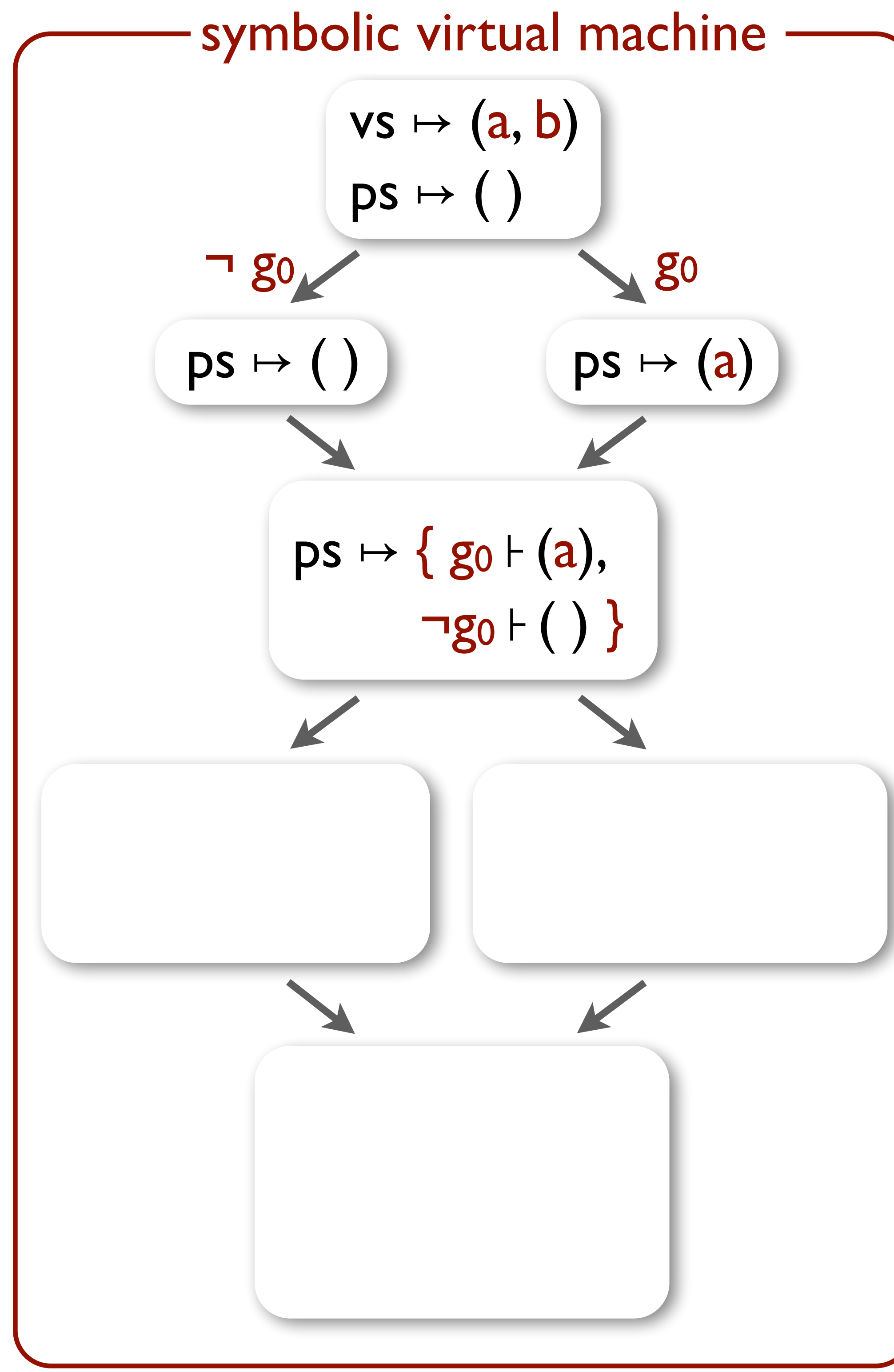


A new design: type-driven state merging

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

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

$g_0 = a > 0$

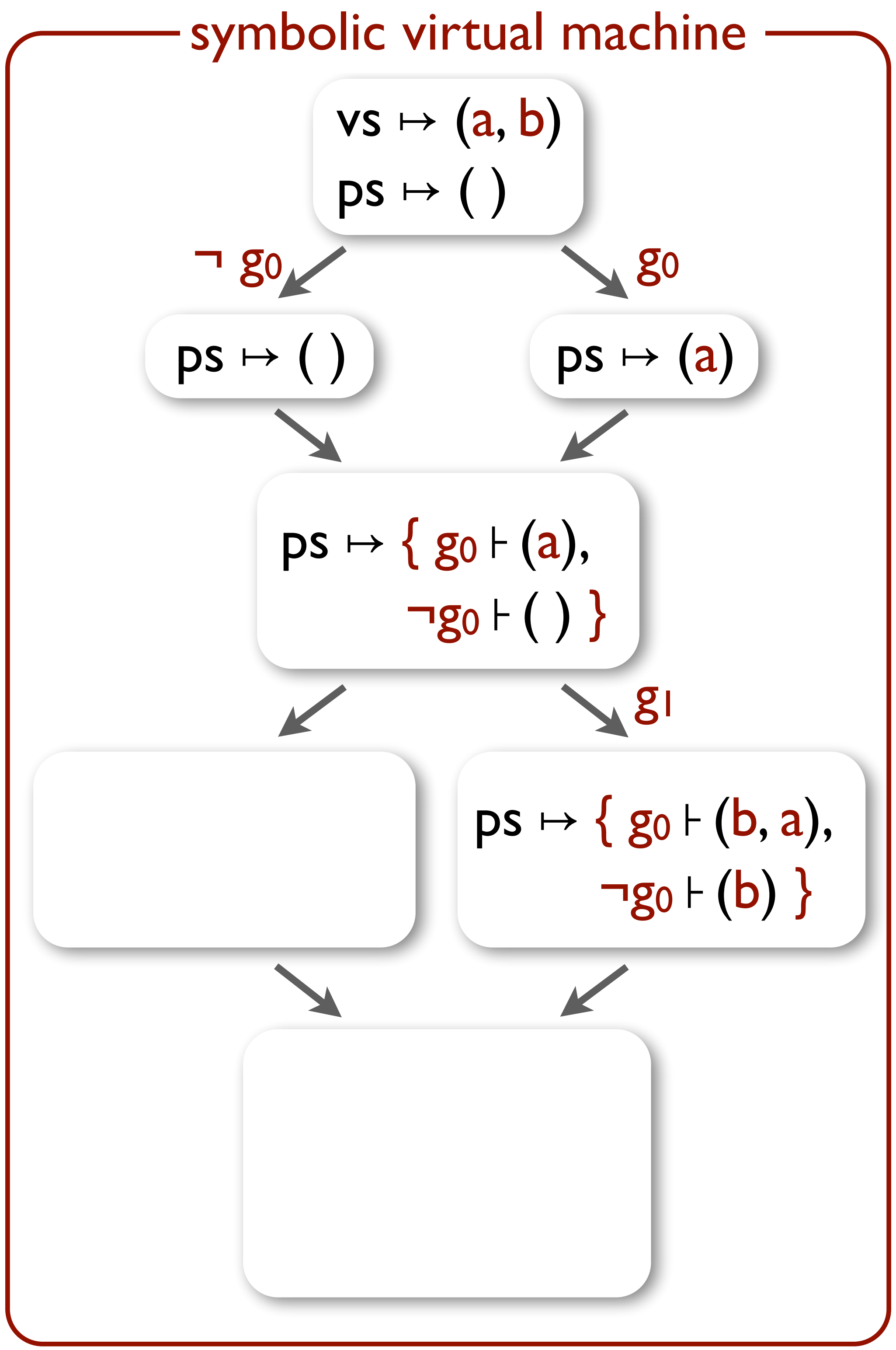


A new design: type-driven state merging

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

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

$g_0 = a > 0$
$g_1 = b > 0$

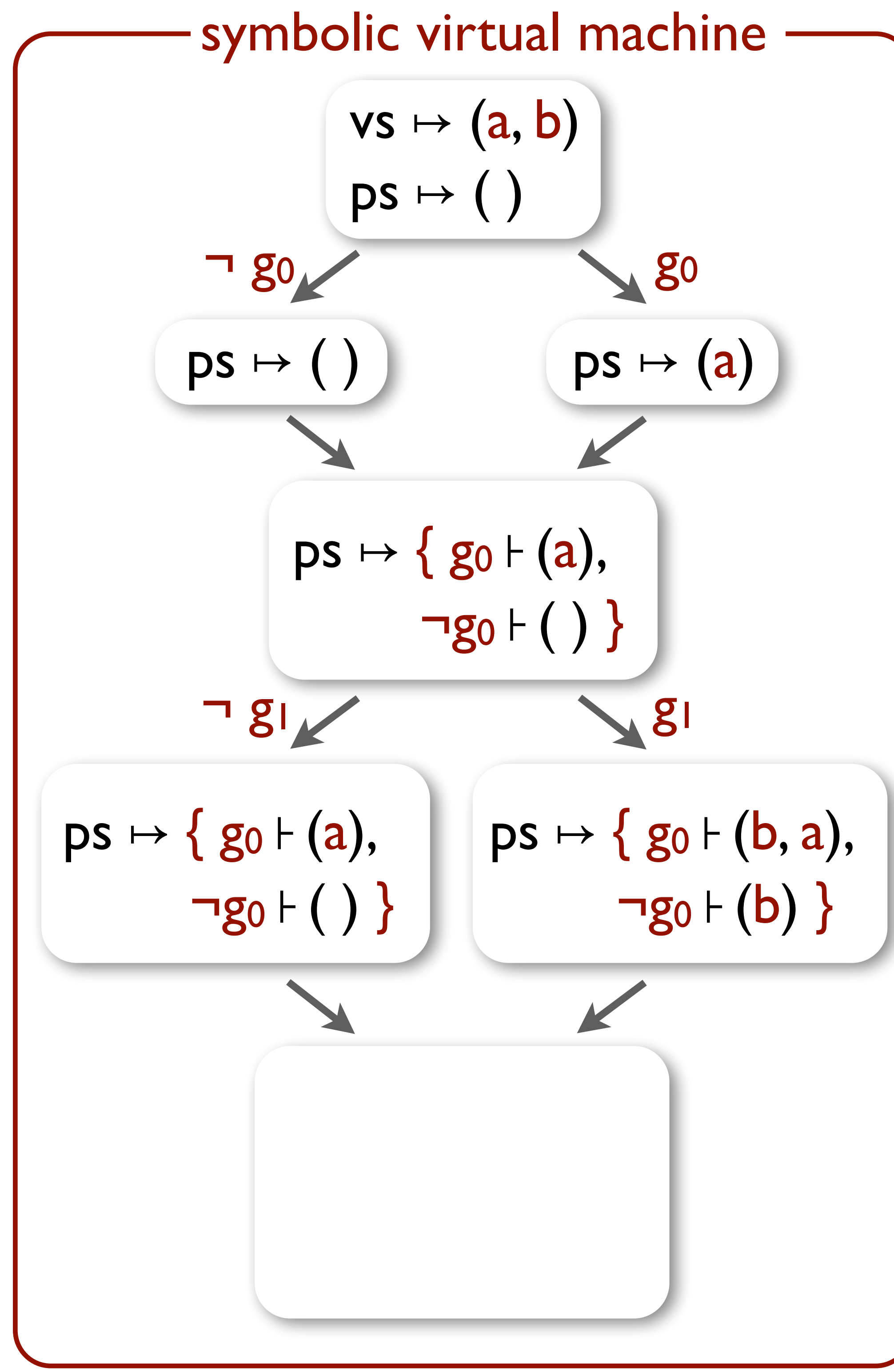


A new design: type-driven state merging

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

$g_0 = a > 0$

$g_1 = b > 0$



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)
```

$g_0 = a > 0$

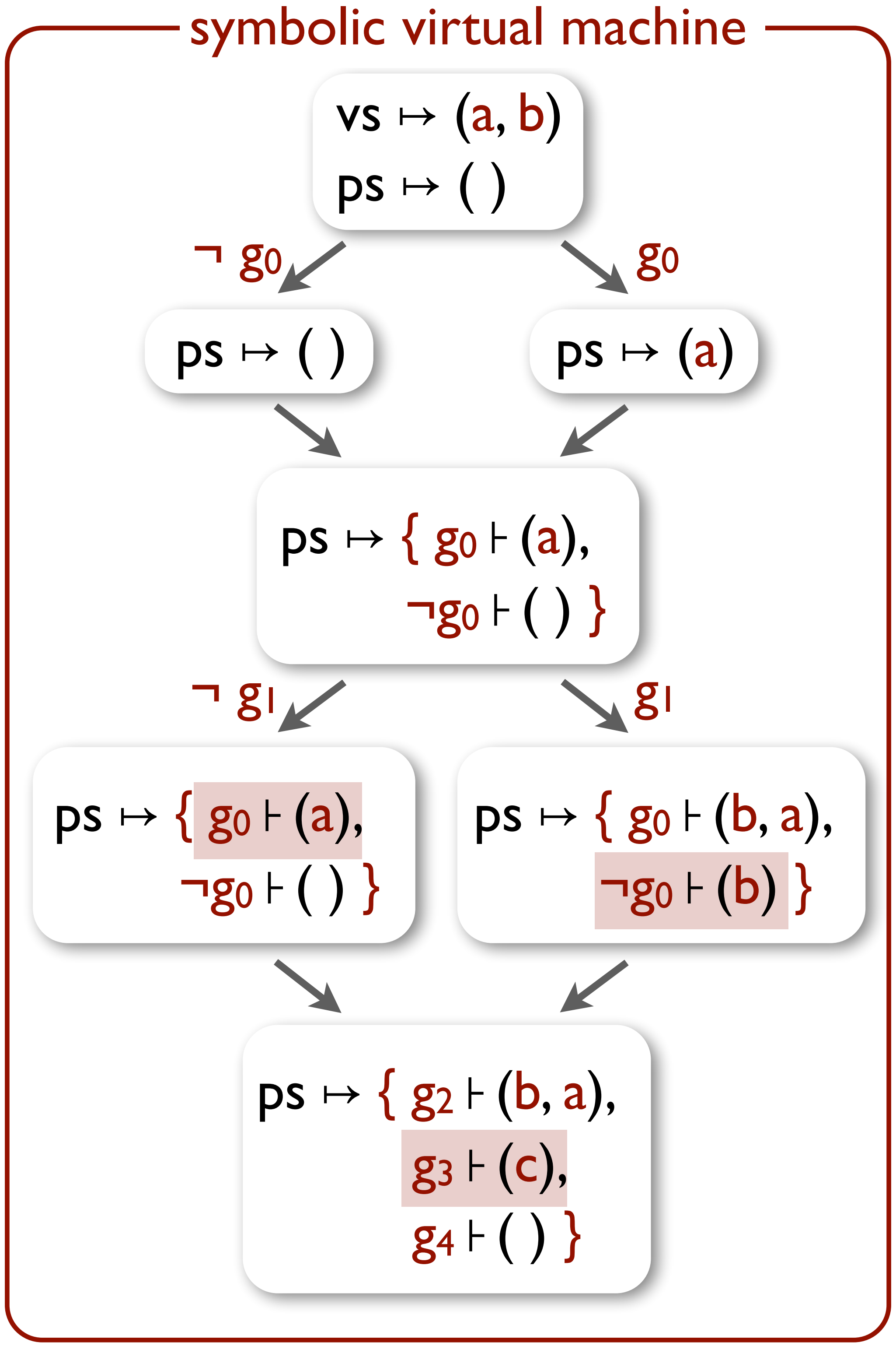
$g_1 = b > 0$

$g_2 = g_0 \wedge g_1$

$g_3 = \neg(g_0 \Leftrightarrow g_1)$

$g_4 = \neg g_0 \wedge \neg g_1$

$c = \mathrm{ite}(g_1, b, a)$

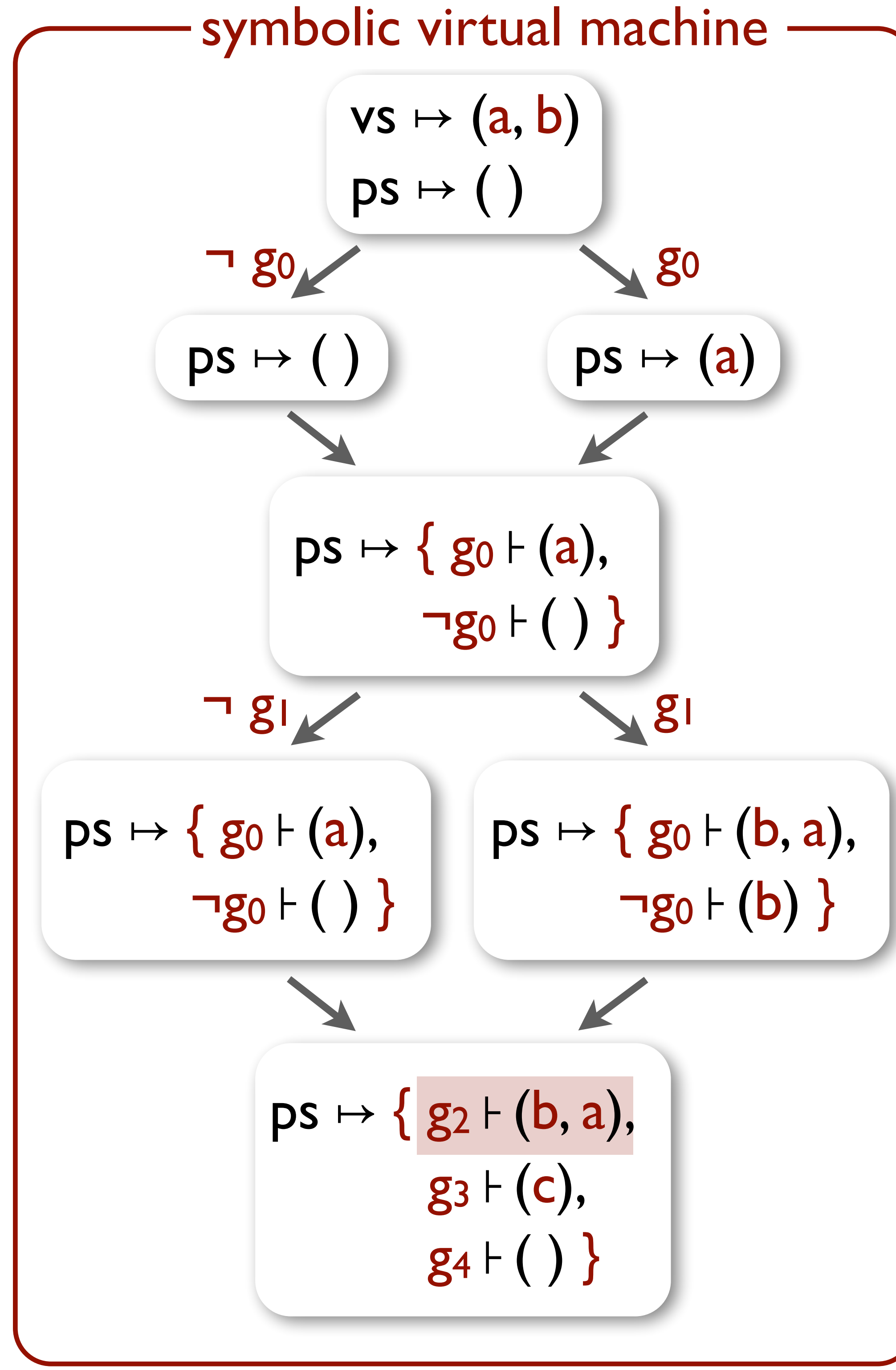


A new design: type-driven state merging

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

Evaluate `len` concretely on all lists in the union; assertion true only on the list guarded by g_2.

$g_0 = a > 0$
$g_1 = b > 0$
$g_2 = g_0 \wedge g_1$
$g_3 = \neg(g_0 \Leftrightarrow g_1)$
$g_4 = \neg g_0 \wedge \neg g_1$
$c = \mathrm{ite}(g_1, b, a)$
assert g_2



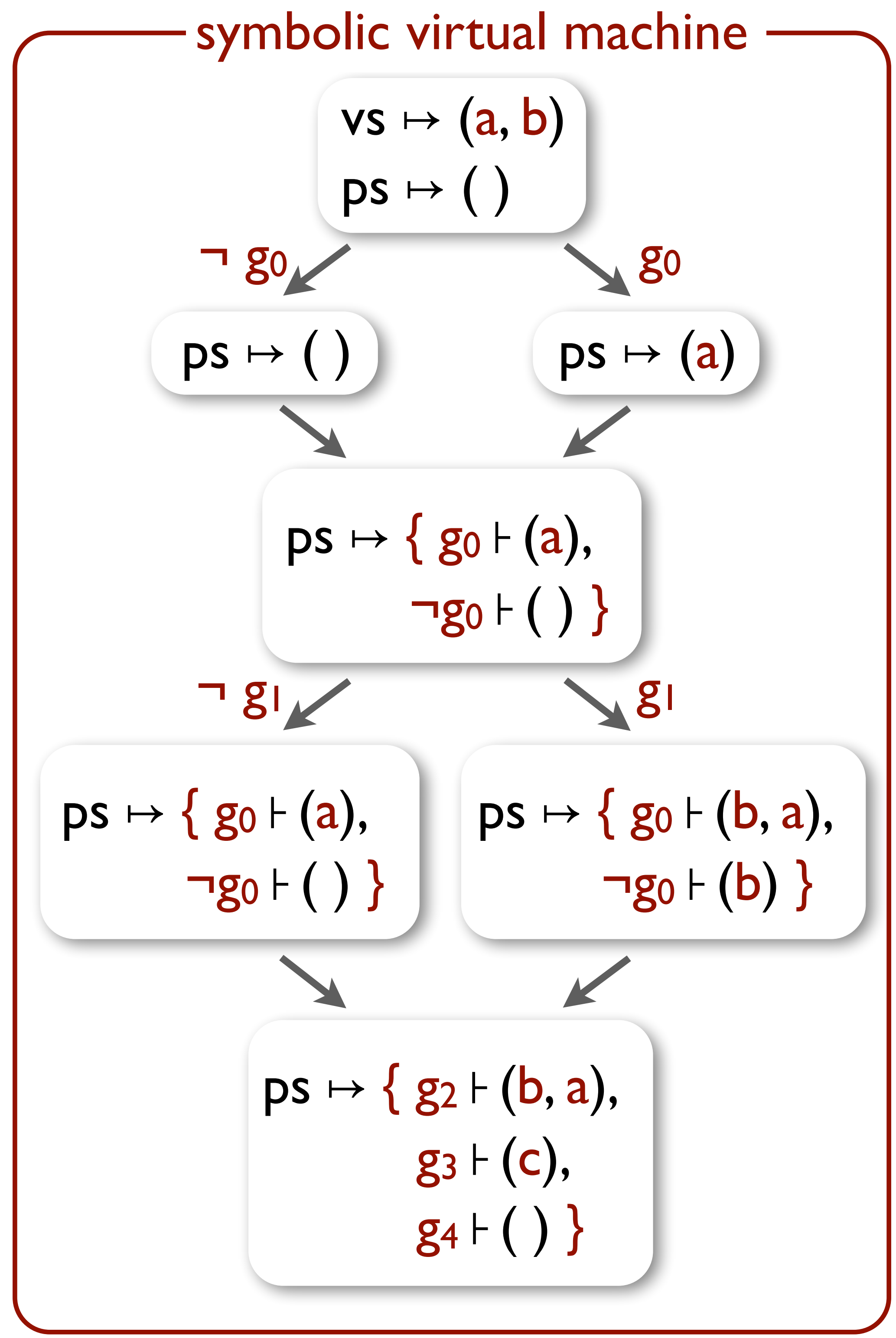
A new design: type-driven state merging

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

polynomial encoding

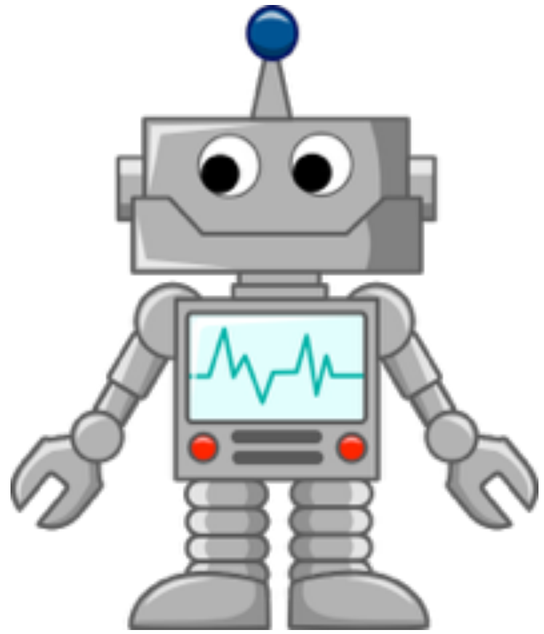
concrete evaluation

$g_0 = a > 0$
$g_1 = b > 0$
$g_2 = g_0 \wedge g_1$
$g_3 = \neg(g_0 \Leftrightarrow g_1)$
$g_4 = \neg g_0 \wedge \neg g_1$
$c = \mathrm{ite}(g_1, b, a)$
assert g_2



apps

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

A Framework for Synthesis and Parameterized Design of Rule Systems Applied to Algebra (ITS'16)
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.

...

Enlearn

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

Enlearn

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

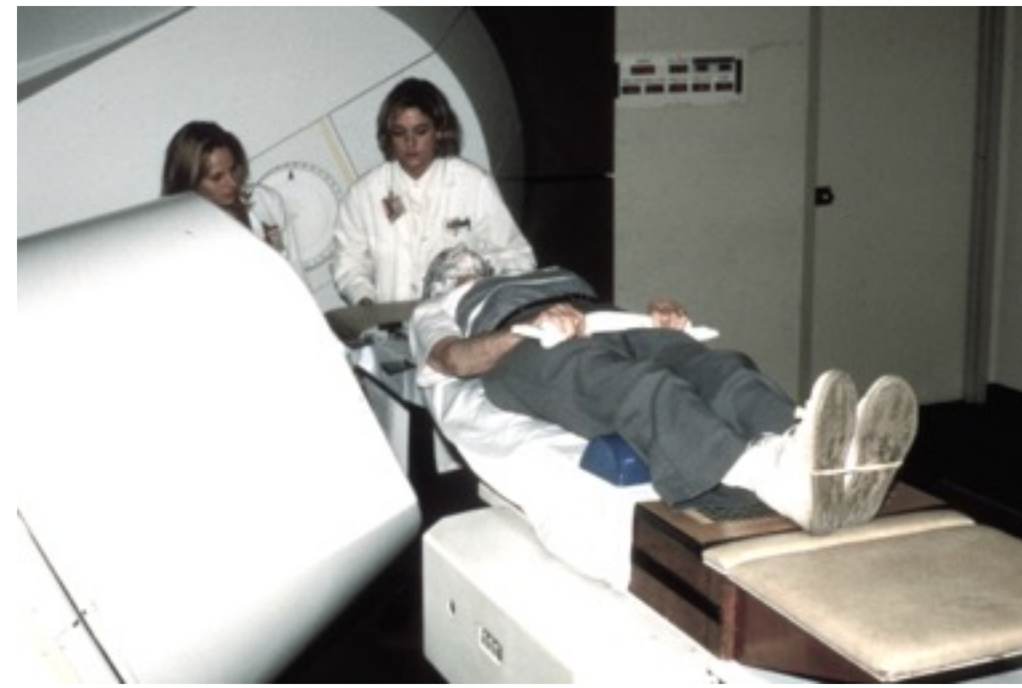
Recent applications of ROSETTE: Neutrons

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

Enlearn

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

Clinical Neutron Therapy System (CNTS) at UW



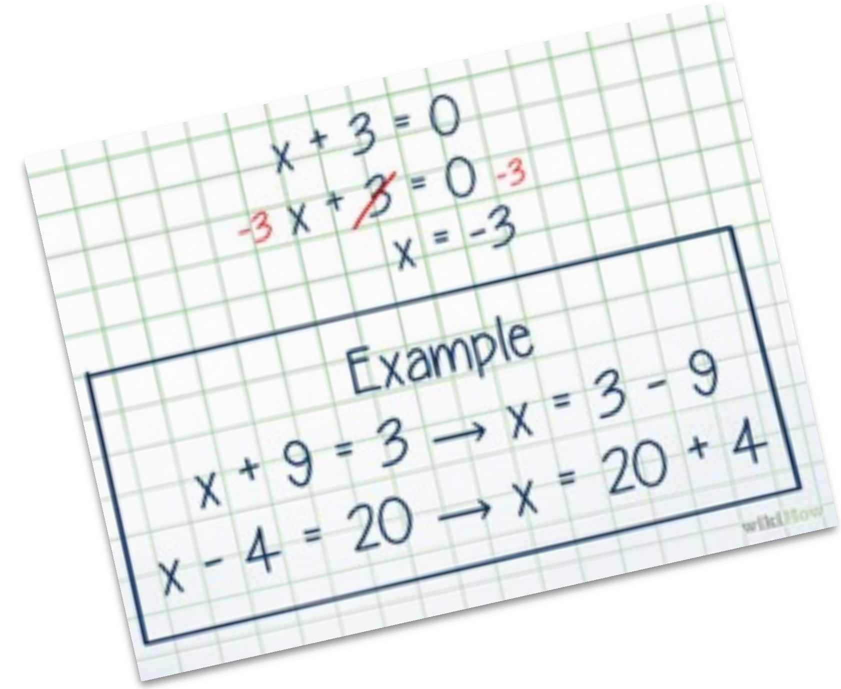
- Used Rosette to build a verifier for the EPICS DSL.
- Found safety-critical defects in a pre-release version of the therapy control software.
- Used by CNTS staff to verify changes to the control software.

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

Enlearn

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

verify

debug

your SDSL

solve

synthesize

ROSETTE

symbolic virtual machine