Finding Code That Explodes Under Symbolic Evaluation

James Bornholt
Emina Torlak

University of Washington

unsat.org

Automated reasoning tools help us solve hard programming problems

Automated reasoning tools help us solve hard programming problems

👷 Does my program still work after the file system crashes? [ASPLOS'16]

Verification

Automated reasoning tools help us solve hard programming problems

Does my program still work after the file system crashes? [ASPLOS'16]

Verification

Synthesis

How do I compile code for this weird new architecture? [PLDI'14]

Automated reasoning tools help us solve hard programming problems

Does my program still work after the file system crashes? [ASPLOS'16]

Verification

Synthesis

How do I compile code for this weird new architecture? [PLDI'14]

How do I teach kids the rules of algebra effectively? [VMCAI'18]

"Programs"

Symbolic evaluators

Does my program still work after the file system crashes? [ASPLOS'16]

How do I compile code for this weird new architecture? [PLDI'14]

Symbolic evaluators

Does my program still work after the file system crashes? [ASPLOS’16]

Interpreter for file system operations

How do I compile code for this weird new architecture? [PLDI’14]

Interpreter for new architecture instructions

Symbolic evaluators

Does my program still work after the file system crashes? [ASPLOS'16]

How do I compile code for this weird new architecture? [PLDI'14]

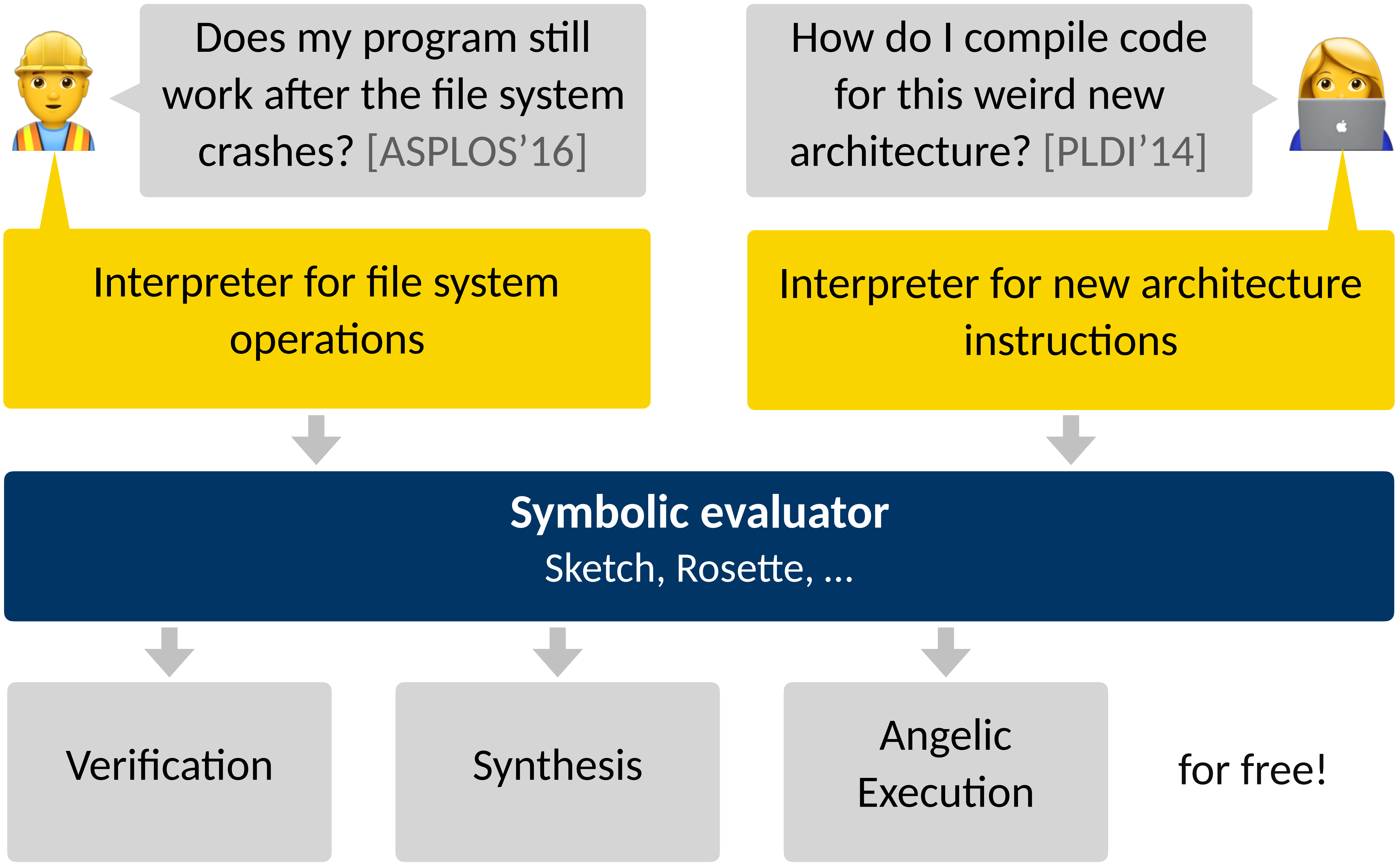
Interpreter for file system operations

Interpreter for new architecture instructions

Symbolic evaluator

Sketch, Rosette, …

Symbolic evaluators


Does my program still work after the file system crashes? [ASPLOS'16]
How do I compile code for this weird new architecture? [PLDI'14]
Interpreter for file system operations
Interpreter for new architecture instructions
Symbolic evaluator
Sketch, Rosette, …
Verification
Synthesis
Angelic Execution
for free!

Symbolic evaluators: no free lunch

Does my program still work after the file system crashes? [ASPLOS'16]

Interpreter for file system operations

Symbolic evaluator

Sketch, Rosette, …

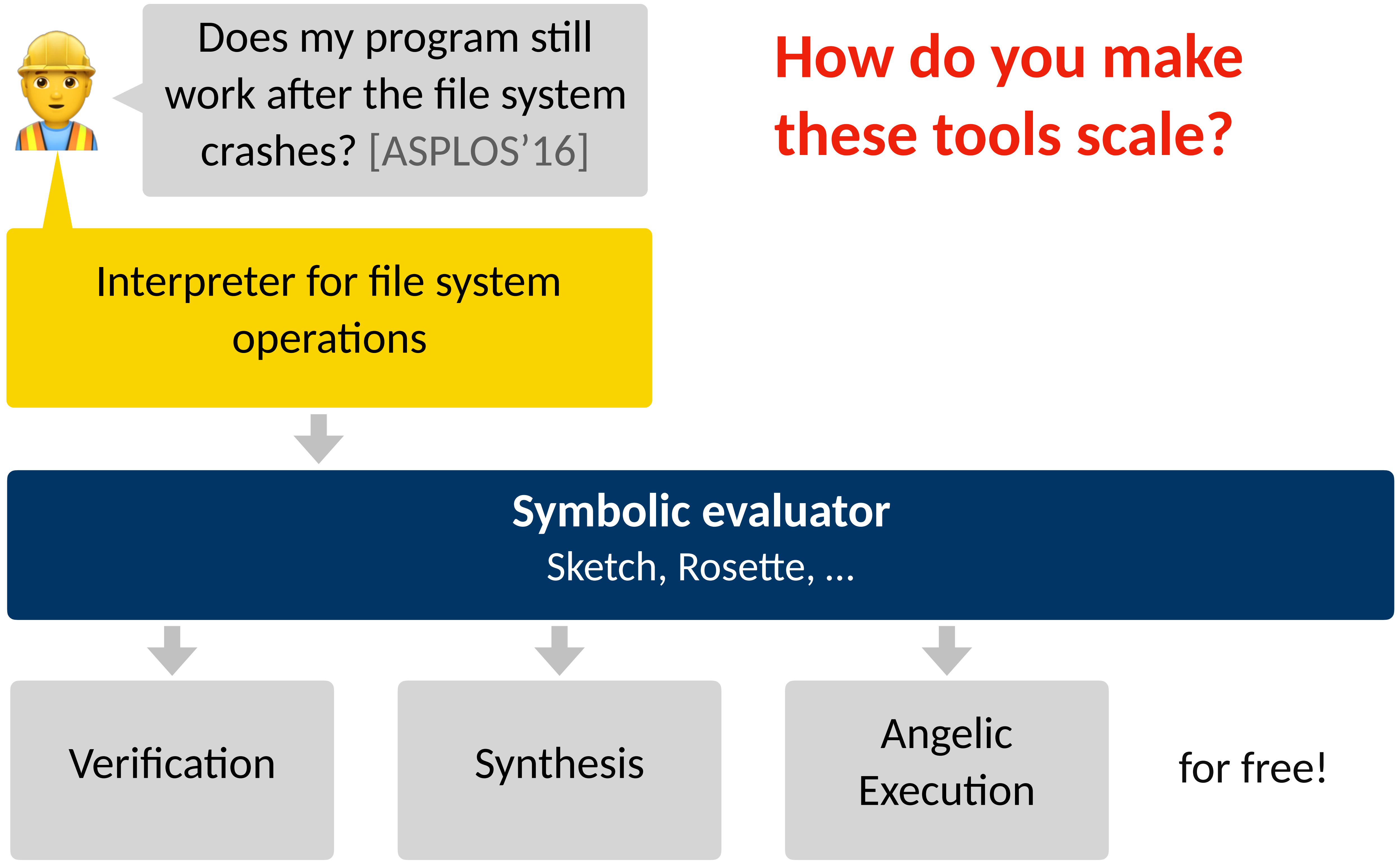
Verification

Synthesis

Angelic Execution

for free!

Symbolic evaluators: no free lunch


Does my program still work after the file system crashes? [ASPLOS'16]
Interpreter for file system operations
Symbolic evaluator
Sketch, Rosette, …
Verification
Synthesis
Angelic Execution
for free!

How do you make these tools scale?

Symbolic evaluators: no free lunch

Does my program still work after the file system crashes? [ASPLOS'16]

How do you make these tools scale?

Interpreter for file system operations

Searching *all paths* through the interpreter

Symbolic evaluator

Sketch, Rosette, …

Verification

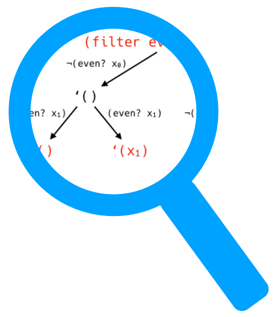
Synthesis

Angelic Execution

for free!

Symbolic profiling identifies performance issues in symbolic evaluation

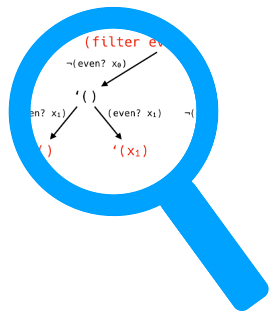
Symbolic profiling identifies performance issues in symbolic evaluation



Symbolic profiling

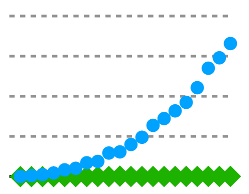
Data structures and analyses

Symbolic profiling identifies performance issues in symbolic evaluation



Symbolic profiling

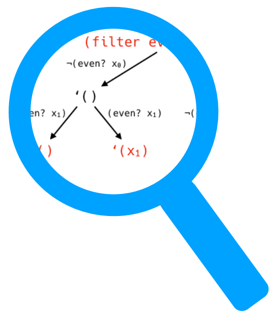
Data structures and analyses



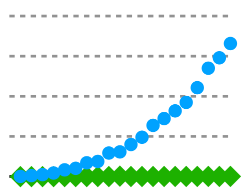
Symbolic evaluation anti-patterns

Common issues and source-level repairs

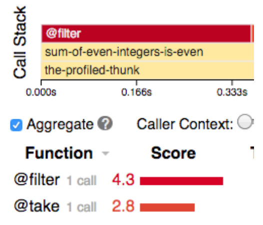
Symbolic profiling identifies performance issues in symbolic evaluation



Symbolic profiling
Data structures and analyses

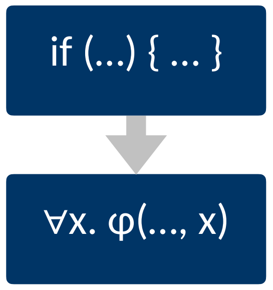


Symbolic evaluation anti-patterns
Common issues and source-level repairs



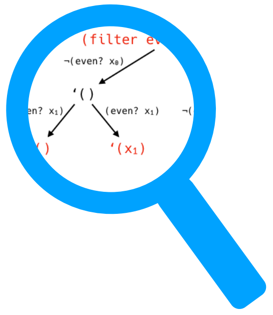
Empirical results
300× speedup on real-world tools

Symbolic profiling identifies performance issues in symbolic evaluation



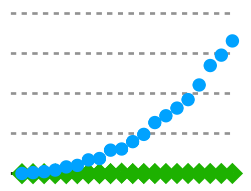
Symbolic evaluation

All-paths execution of programs



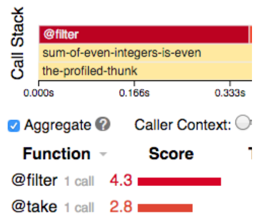
Symbolic profiling

Data structures and analyses



Symbolic evaluation anti-patterns

Common issues and source-level repairs



Call Stack

@filter
sum-of-even-integers-is-even
the-profiled-thunk

0.000s 0.166s 0.333s

Aggregate ☑ Caller Context:

Function	Score
@filter 1 call	4.3
@take 1 call	2.8

Empirical results

300× speedup on real-world tools

Symbolic evaluation

All-paths execution of programs

Symbolic evaluation executes *all* paths through a program

```
#lang rosette

(define (first-k-even lst k)
  (define xs (filter even? lst))
  (take xs k))
```

Symbolic evaluation executes *all* paths through a program

```
#lang rosette

(define (first-k-even lst k)
  (define xs (filter even? lst))
  (take xs k))
```

Inputs are *unknown* (trying to find values that violate spec)

Symbolic evaluation executes *all* paths through a program

```
#lang rosette

(define (first-k-even lst k)
  (define xs (filter even? lst))
  (take xs k))
```

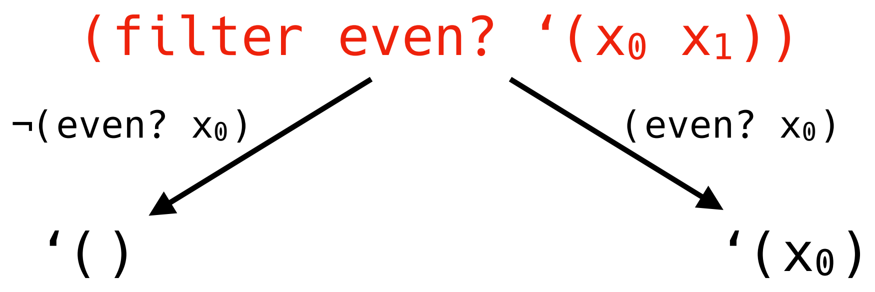
Inputs are *unknown* (trying to find values that violate spec)

```
(filter even? ‘(x₀ x₁))
```

Symbolic evaluation executes *all* paths through a program

```
#lang rosette

(define (first-k-even lst k)
  (define xs (filter even? lst))
  (take xs k))
```

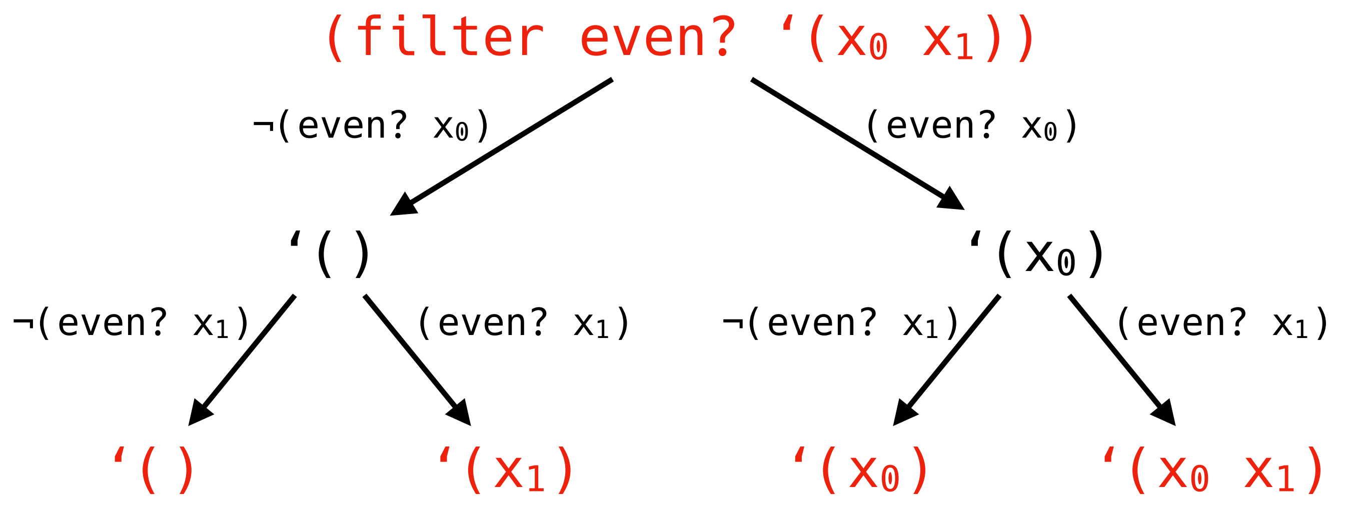
Inputs are *unknown*
(trying to find values that violate spec)



Symbolic evaluation executes *all* paths through a program

```
#lang rosette

(define (first-k-even lst k)
  (define xs (filter even? lst))
  (take xs k))
```

Inputs are *unknown* (trying to find values that violate spec)



Symbolic evaluation executes *all* paths through a program

```
#lang rosette

(define (first-k-even lst k)
  (define xs (filter even? lst))
  (take xs k))
```

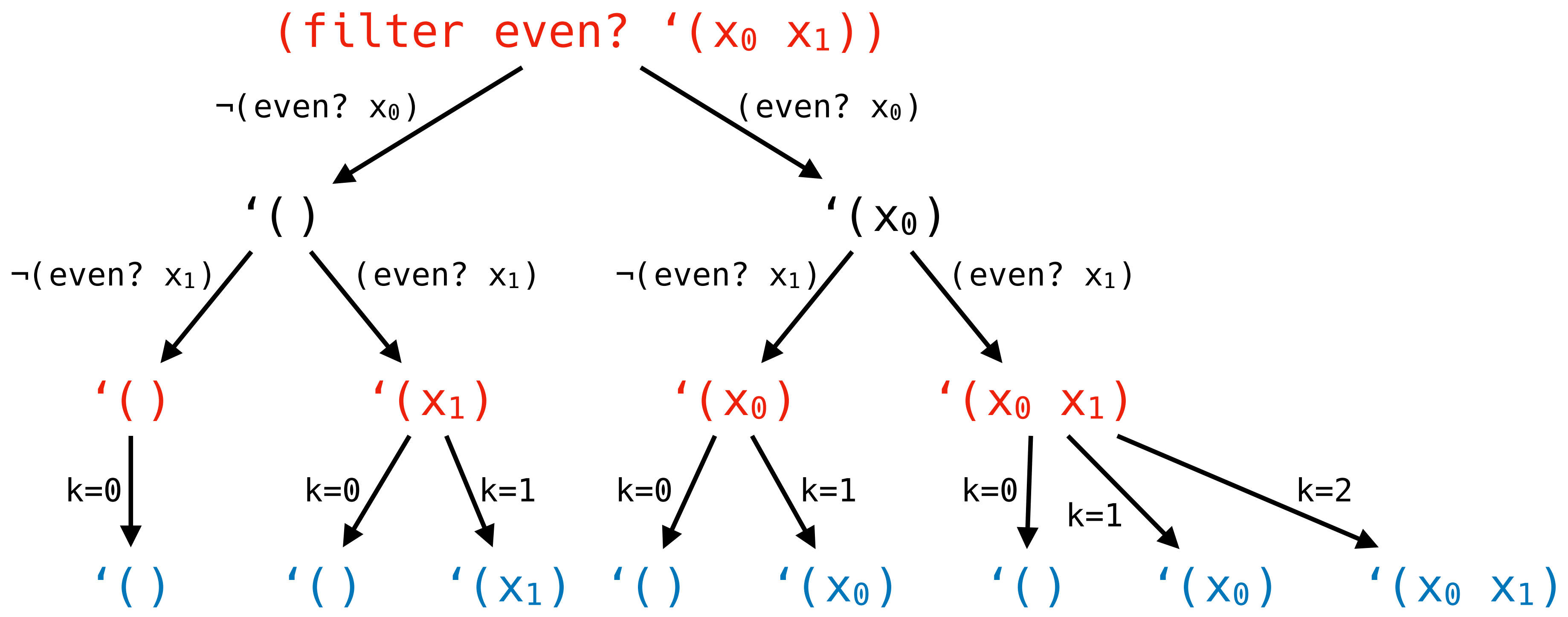
Inputs are *unknown* (trying to find values that violate spec)

$$\begin{tikzcd}
 & & & \texttt{(filter even? `(}x_0\ x_1\texttt{))} \arrow[dll, "\neg\texttt{(even? }x_0\texttt{)}"'] \arrow[drr, "\texttt{(even? }x_0\texttt{)}"] & & & \\
 & \texttt{`()} \arrow[dl, "\neg\texttt{(even? }x_1\texttt{)}"'] \arrow[dr, "\texttt{(even? }x_1\texttt{)}"] & & & & \texttt{`(}x_0\texttt{)} \arrow[dl, "\neg\texttt{(even? }x_1\texttt{)}"'] \arrow[dr, "\texttt{(even? }x_1\texttt{)}"] & \\
\texttt{`()} & & \texttt{`(}x_1\texttt{)} & & \texttt{`(}x_0\texttt{)} & & \texttt{`(}x_0\ x_1\texttt{)}
\end{tikzcd}$$

Symbolic evaluation executes *all* paths through a program

```
#lang rosette

(define (first-k-even lst k)
  (define xs (filter even? lst))
  (take xs k))
```

Inputs are *unknown* (trying to find values that violate spec)

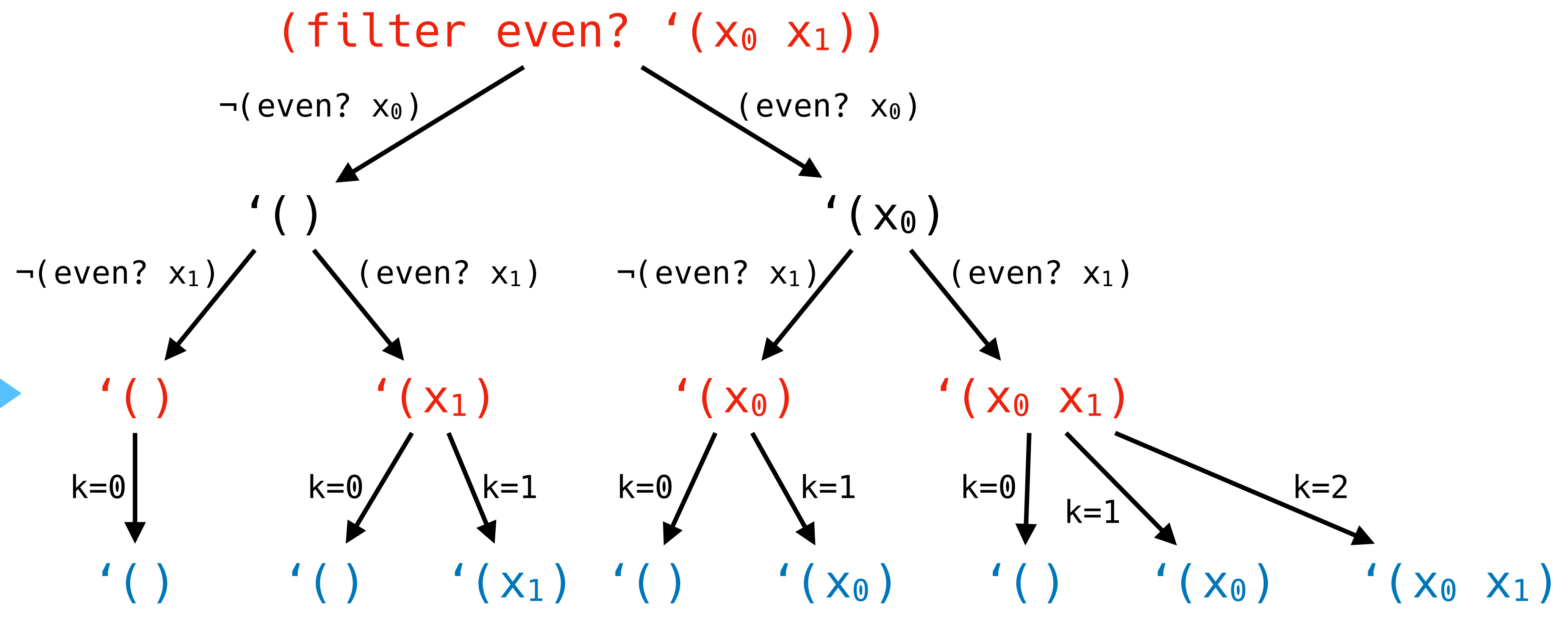


Symbolic evaluation executes *all* paths through a program

```
#lang rosette

(define (first-k-even lst k)
  (define xs (filter even? lst))
  (take xs k))
```

Inputs are *unknown* (trying to find values that violate spec)

take runs 2^2 times



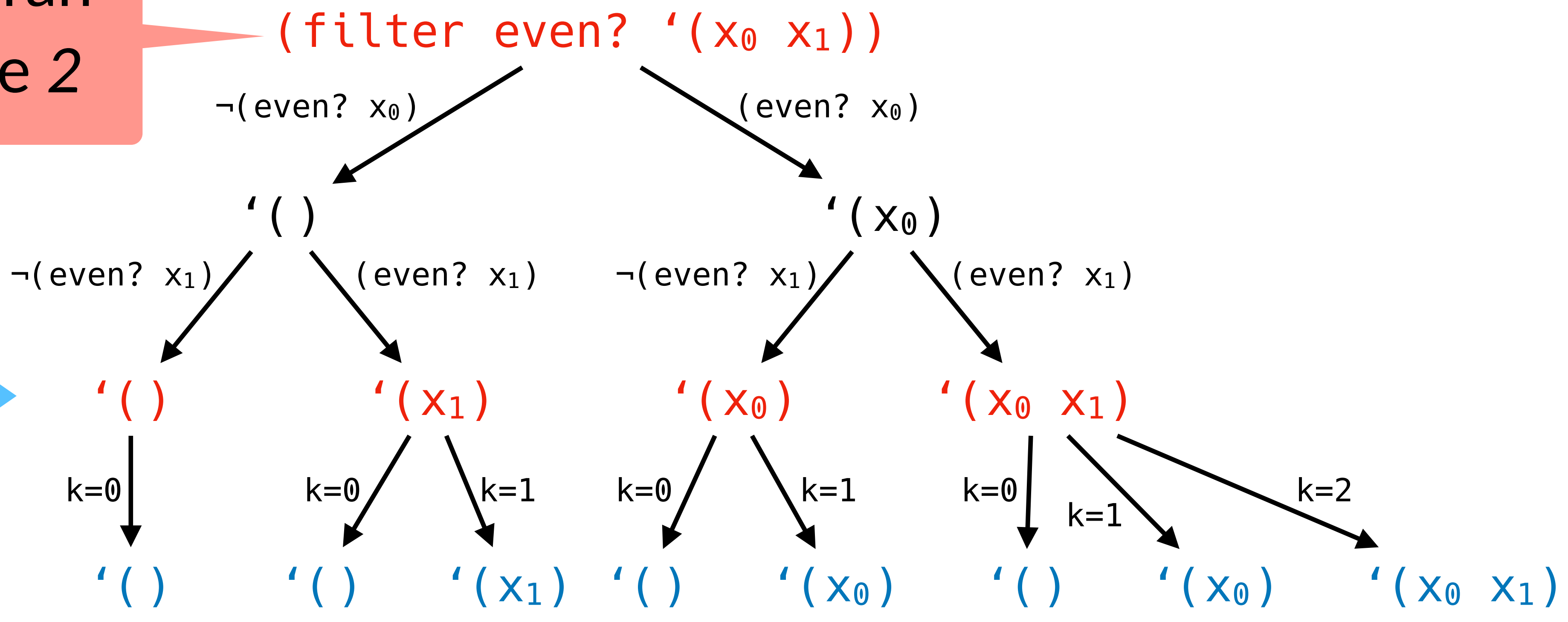
Symbolic evaluation executes *all* paths through a program

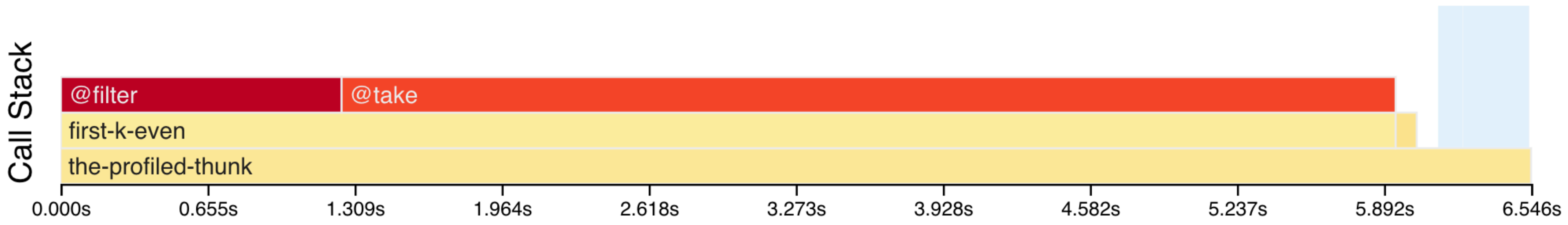
```
#lang rosette

(define (first-k-even lst k)
  (define xs (filter even? lst))
  (take xs k))
```

Inputs are *unknown* (trying to find values that violate spec)

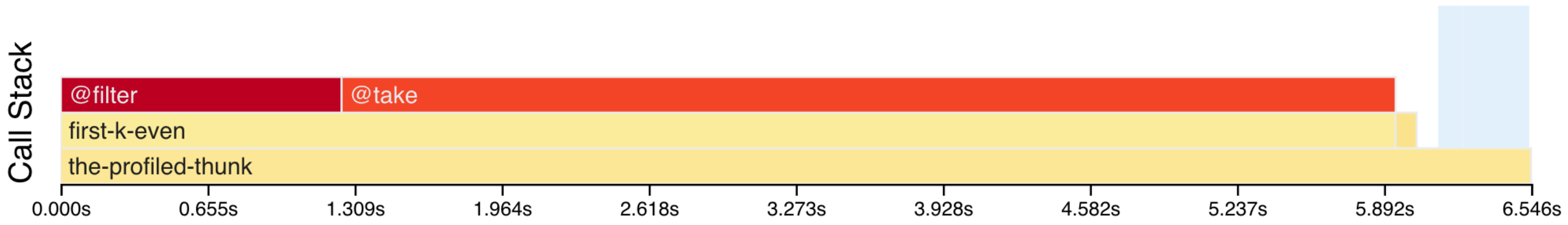
because filter ran on a list of size *2*

take runs 2^2 times





Function	Score	Time (ms)	Term Count	Unused Terms	Union Size	Merge Cases
filter 1 call	4.3	1249	137408	131164	4288	93664
take 1 call	2.8	4692	50312	49986	2209	49986
andmap 1 call	0.3	94	14180	14180	0	4097
the-profiled-thunk	0.1	511	66	0	0	0



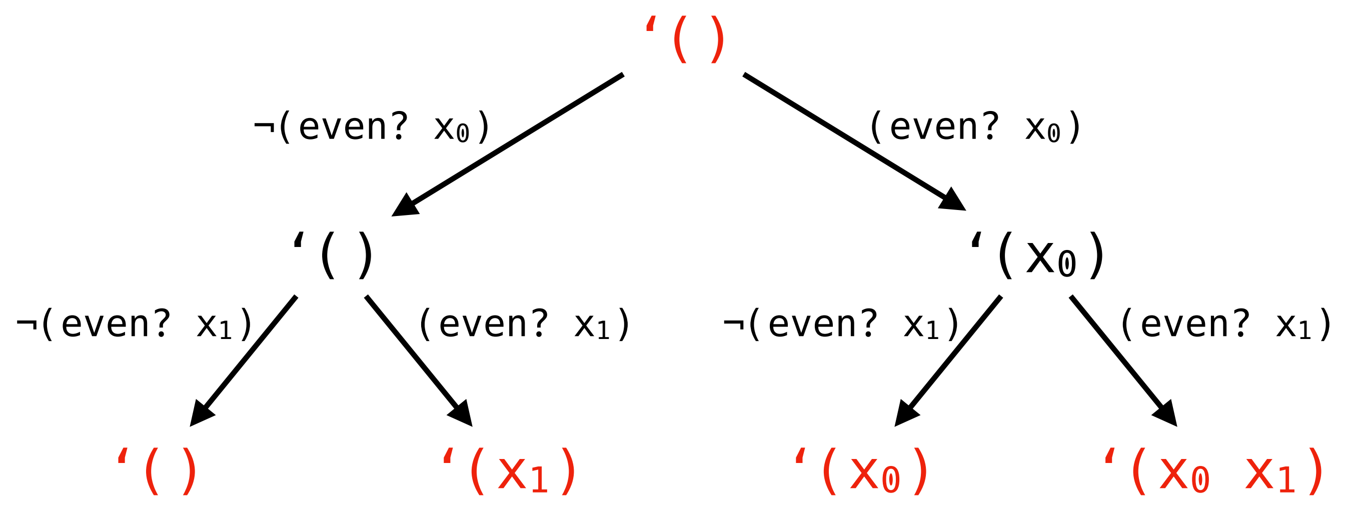
Function	Score	Time (ms)	Term Count	Unused Terms	Union Size	Merge Cases
filter 1 call	4.3	1249	137408	131164	4288	93664
take 1 call	2.8	4692	50312	49986	2209	49986
andmap 1 call	0.3	94	14180	14180	0	4097
the-profiled-thunk	0.1	511	66	0	0	0

Blaming filter even though it’s not the slowest

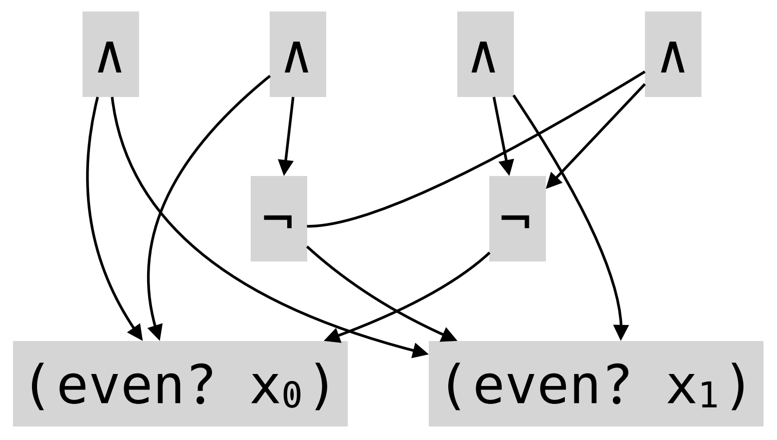
Symbolic profiling

Data structures and metrics

Two data structures to summarize symbolic evaluation



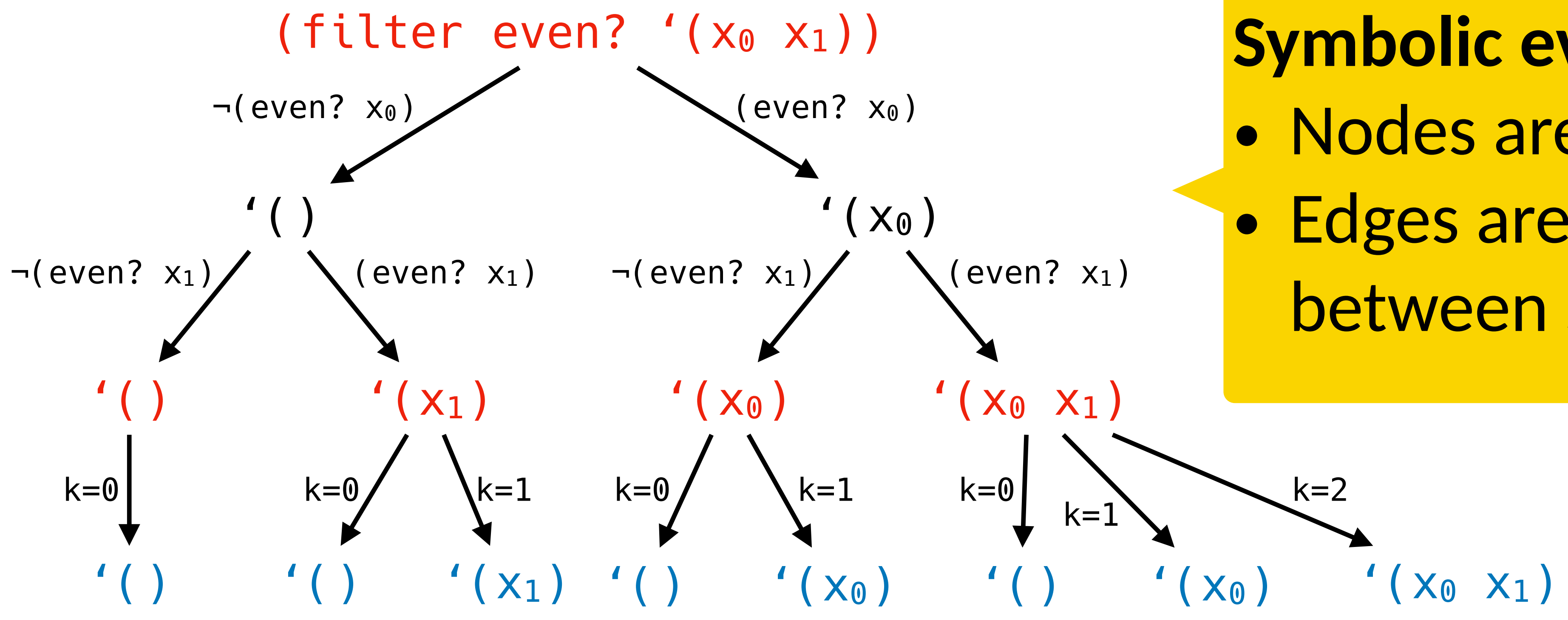
Symbolic evaluation graph
Reflects the evaluator's strategy
for all-paths execution of the program



Symbolic heap
Shape of all symbolic values
created by the program

Any symbolic evaluation technique can be summarized by these two data structures

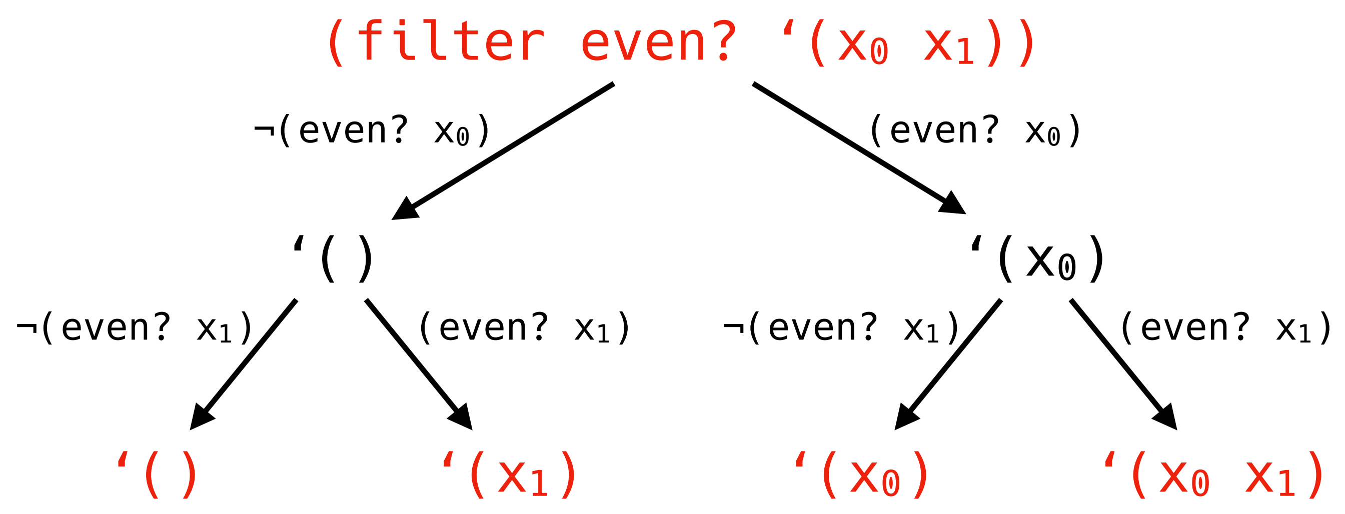
The *symbolic evaluation graph* summarizes branching and merging



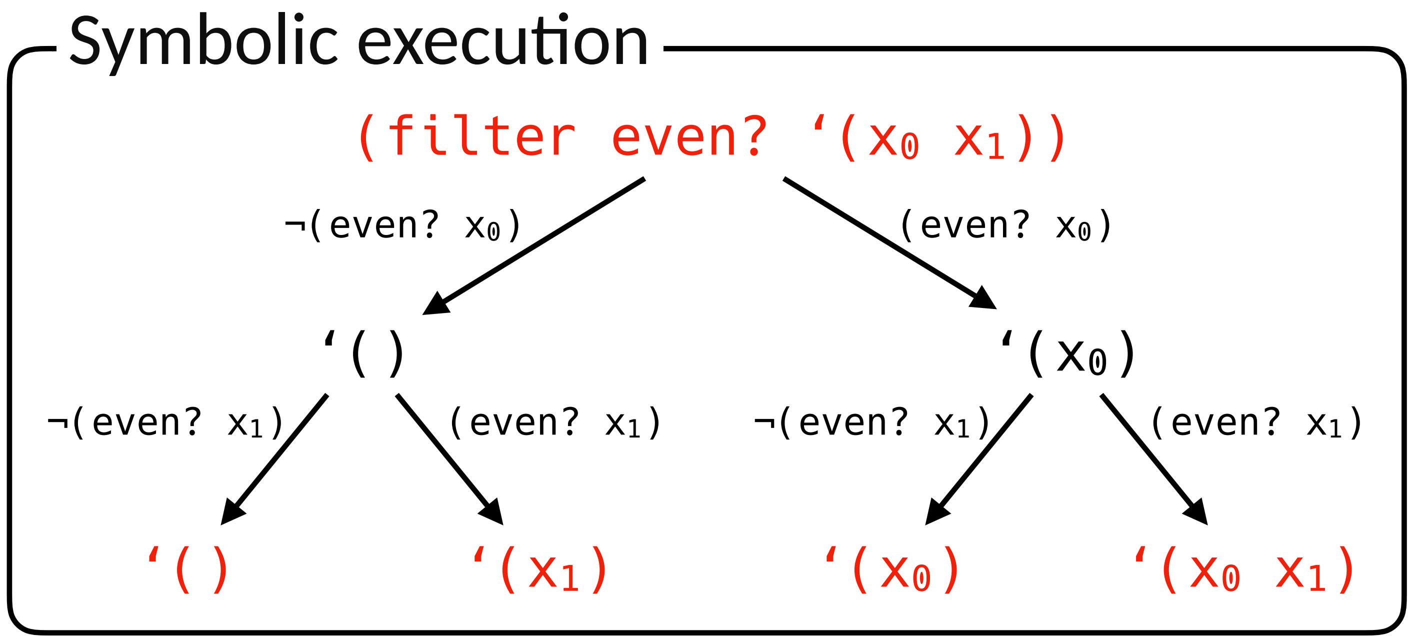
Symbolic evaluation graph

- Nodes are program states
- Edges are transitions between states

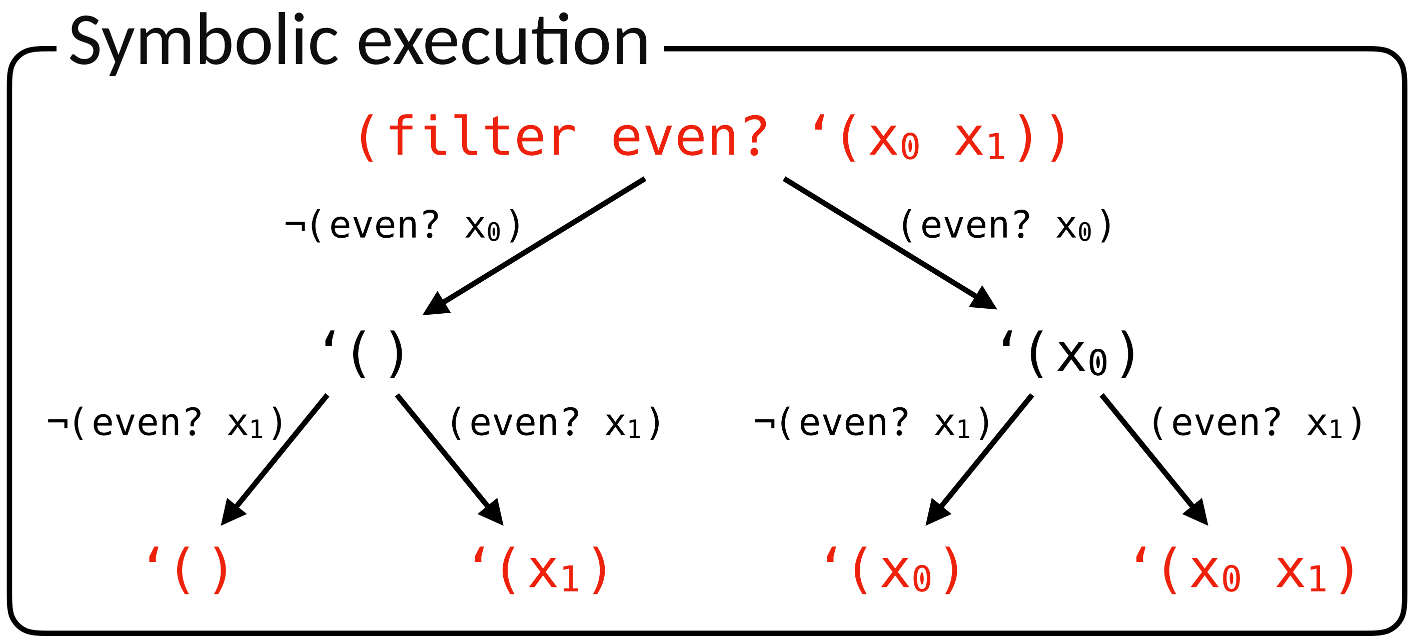
The *symbolic evaluation graph* summarizes branching and merging

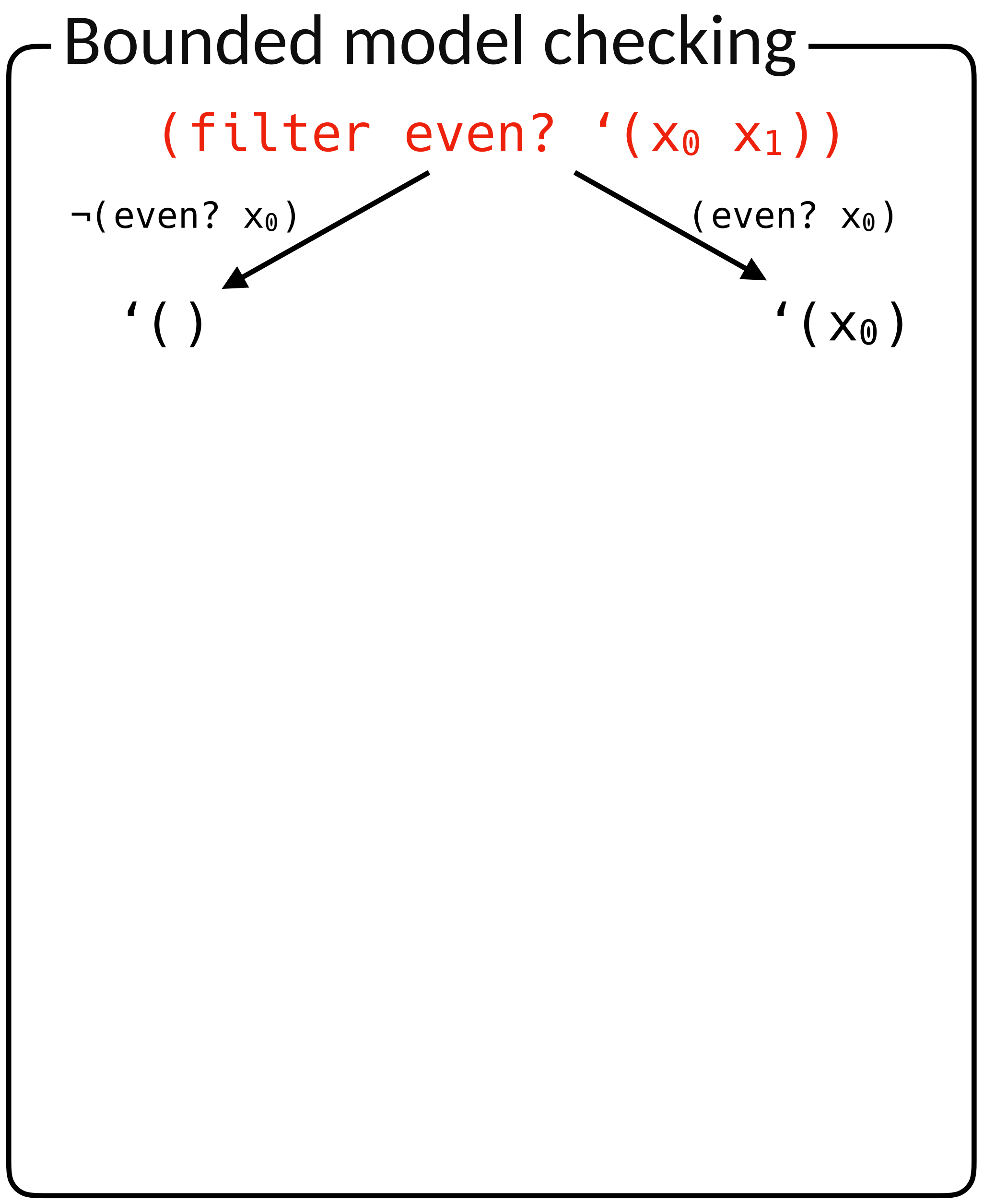

$$\begin{tikzcd}
 & & & \texttt{(filter even? `(}x_0\ x_1\texttt{))} \arrow[dll, "\neg\texttt{(even? }x_0\texttt{)}"'] \arrow[drr, "\texttt{(even? }x_0\texttt{)}"] & & & \\
 & \texttt{`()} \arrow[dl, "\neg\texttt{(even? }x_1\texttt{)}"'] \arrow[dr, "\texttt{(even? }x_1\texttt{)}"] & & & & \texttt{`(}x_0\texttt{)} \arrow[dl, "\neg\texttt{(even? }x_1\texttt{)}"'] \arrow[dr, "\texttt{(even? }x_1\texttt{)}"] & \\
\texttt{`()} & & \texttt{`(}x_1\texttt{)} & & \texttt{`(}x_0\texttt{)} & & \texttt{`(}x_0\ x_1\texttt{)}
\end{tikzcd}$$

The *symbolic evaluation graph* summarizes branching and merging

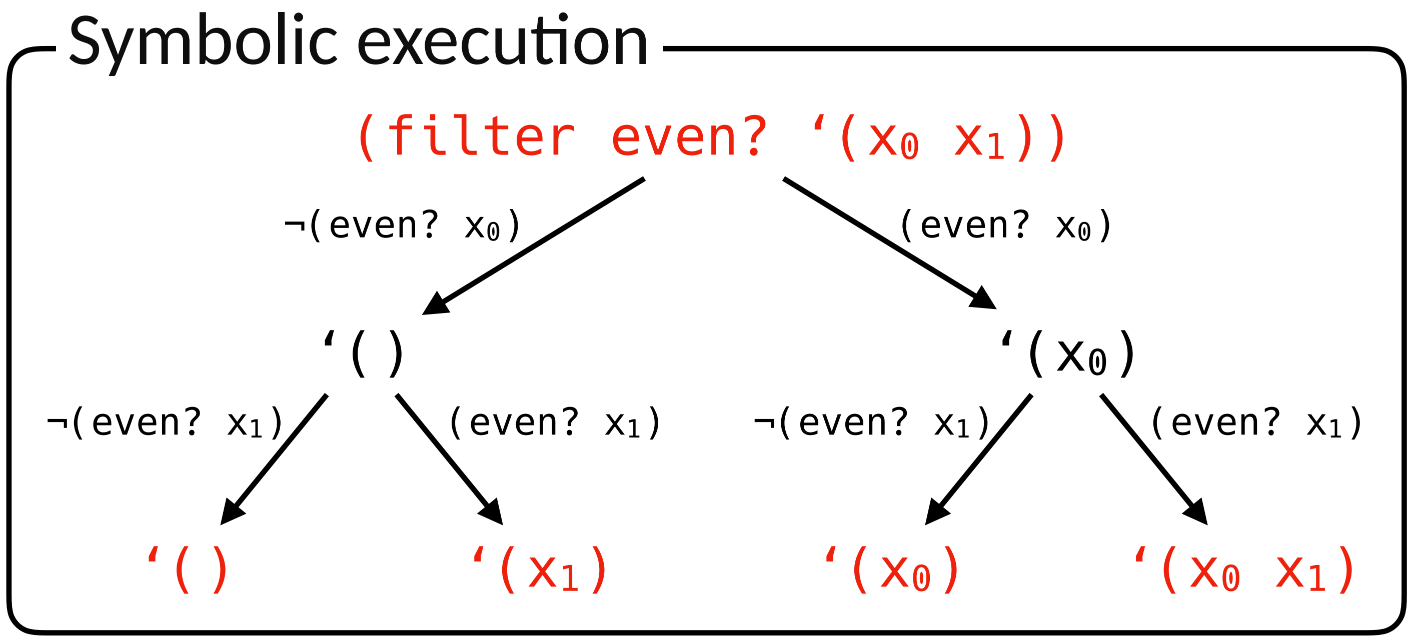


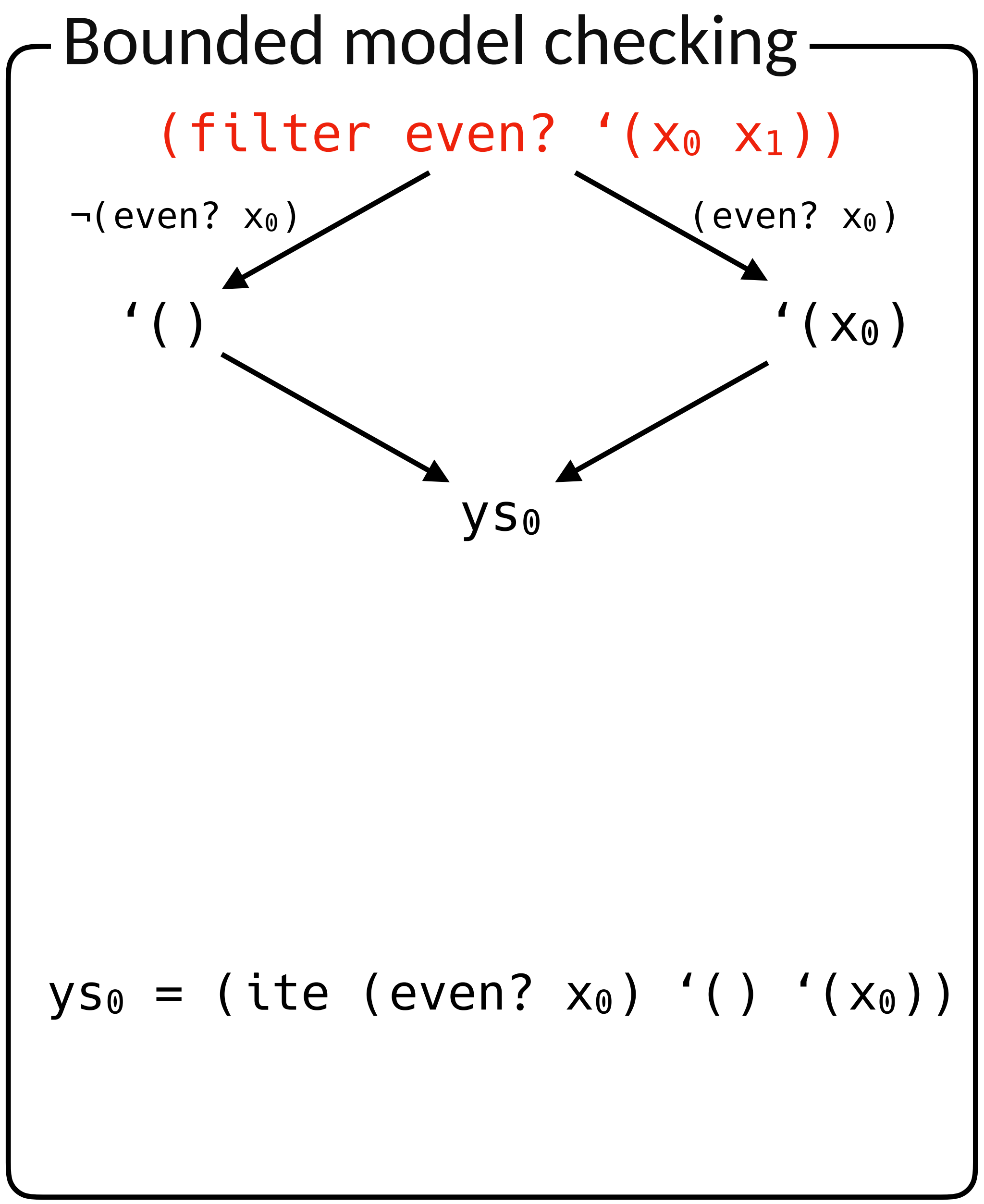
The *symbolic evaluation graph* summarizes branching and merging





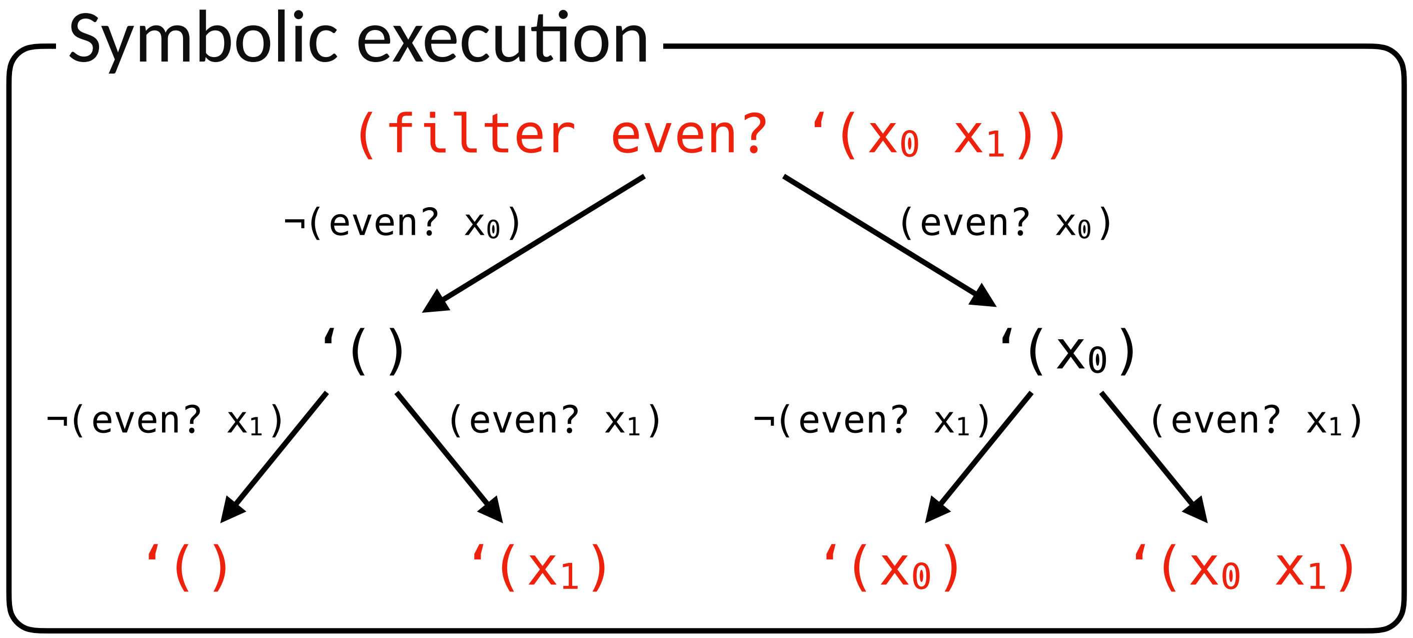
The *symbolic evaluation graph* summarizes branching and merging

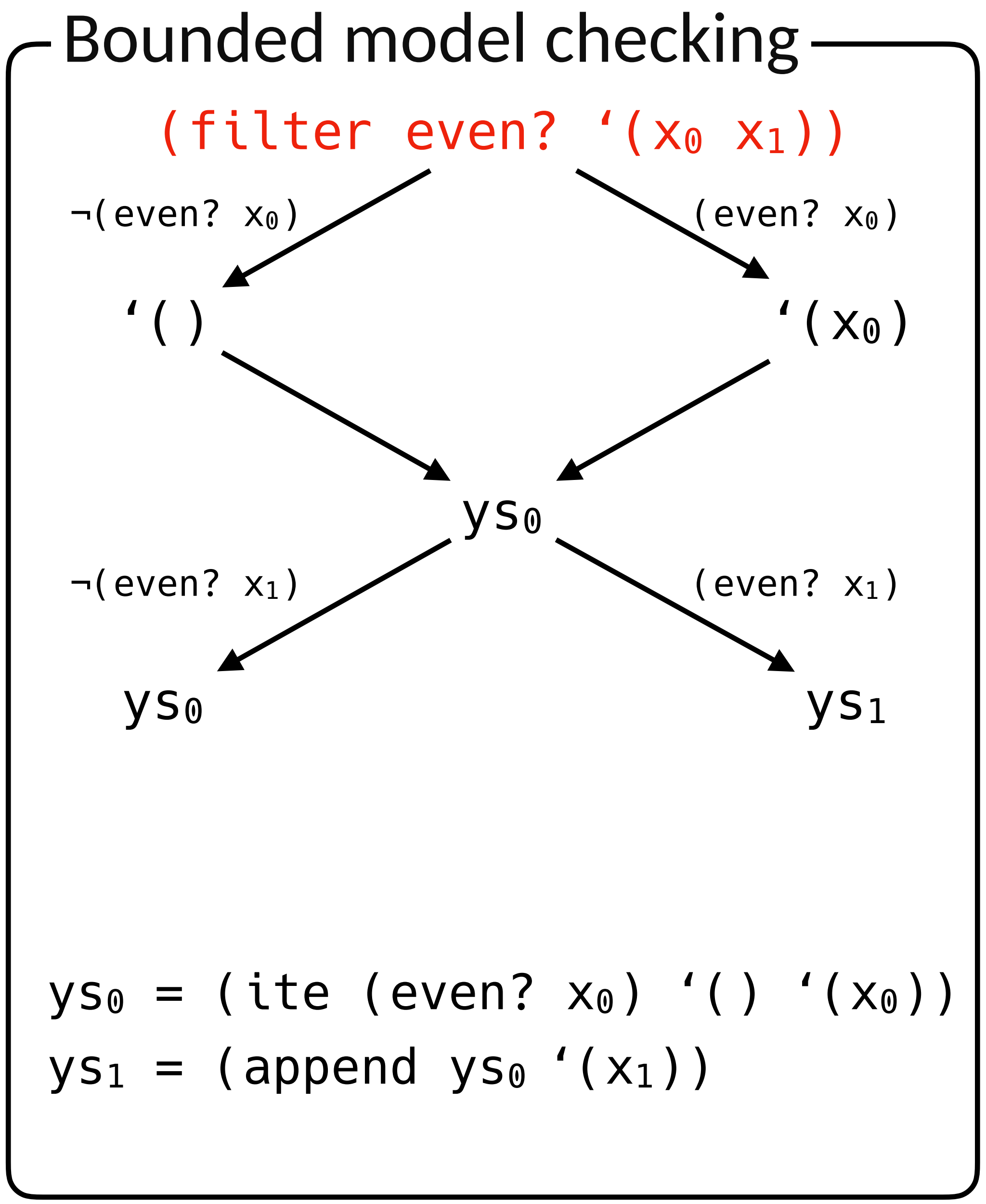




```
ys₀ = (ite (even? x₀) ‘() ‘(x₀))
```

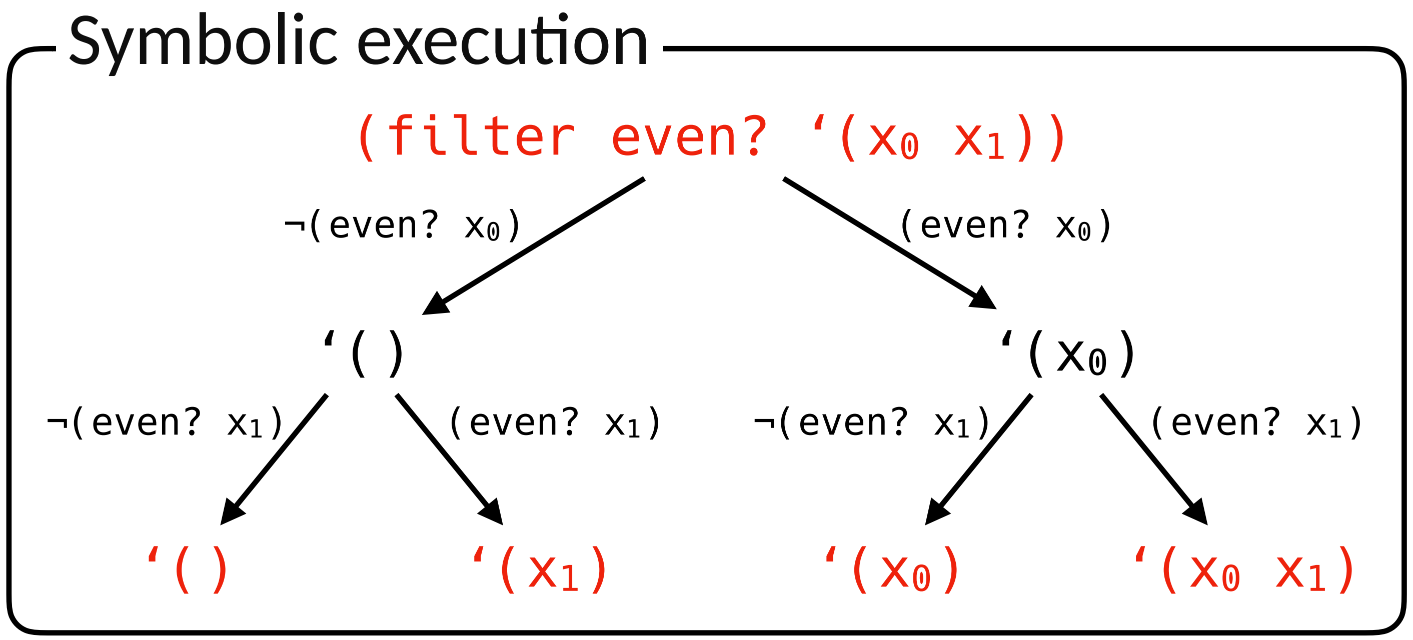
The *symbolic evaluation graph* summarizes branching and merging

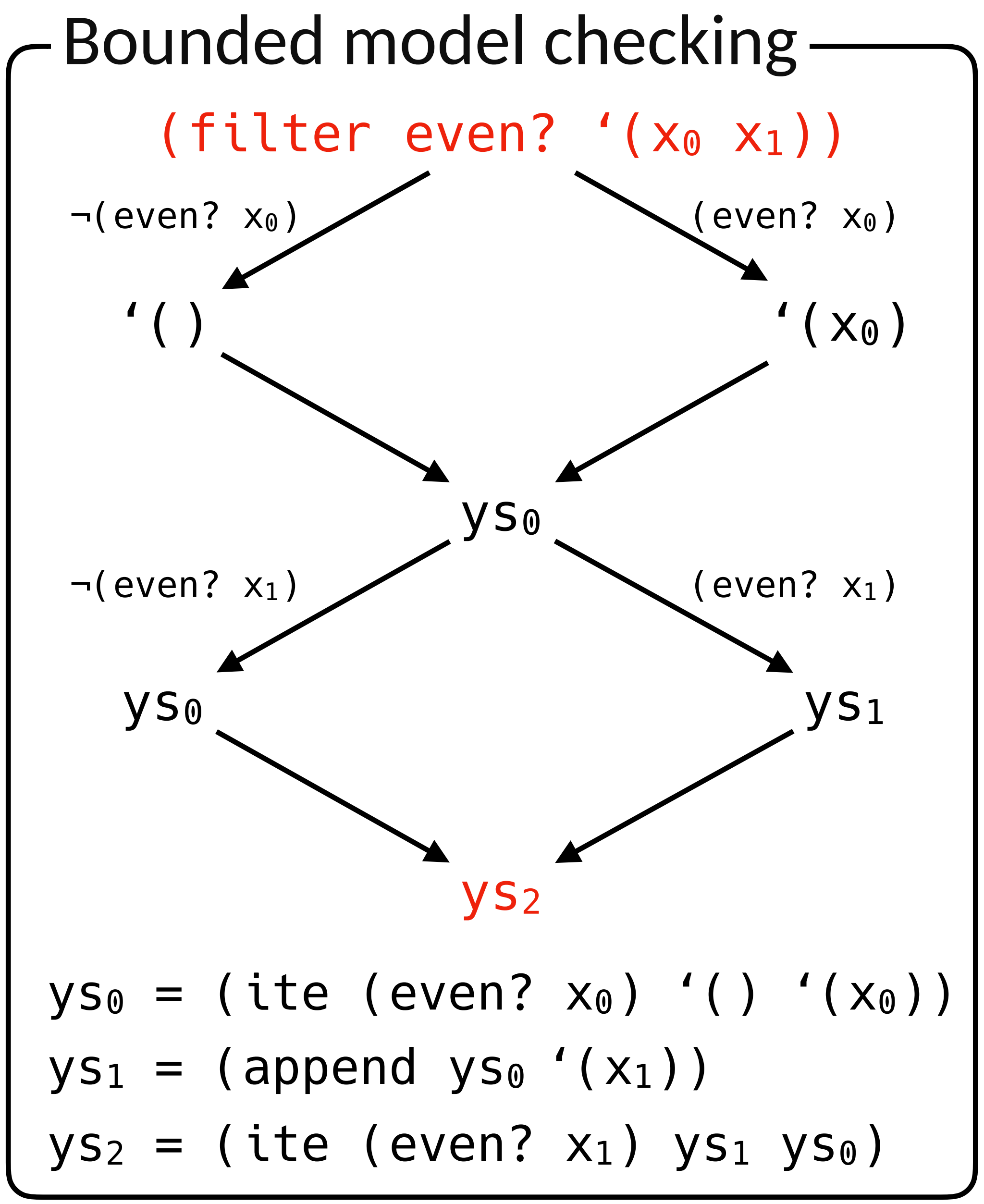




```
ys₀ = (ite (even? x₀) ‘() ‘(x₀))
ys₁ = (append ys₀ ‘(x₁))
```

The *symbolic evaluation graph* summarizes branching and merging

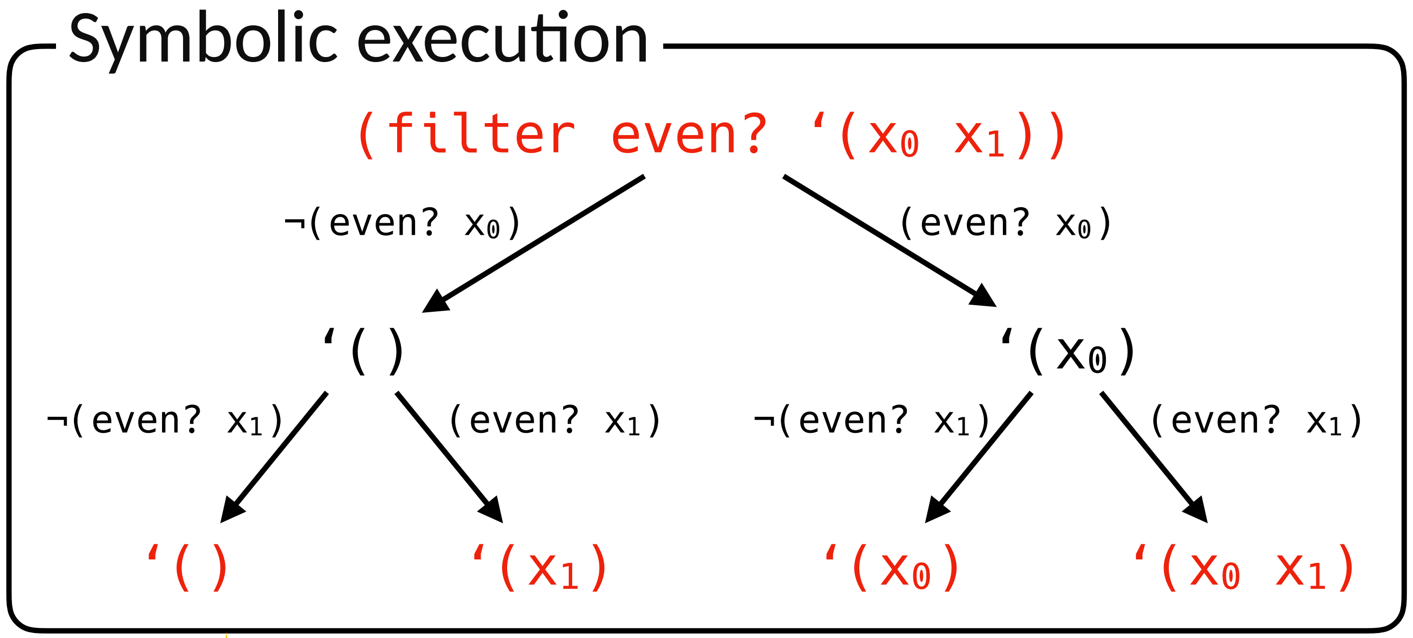




```
ys0 = (ite (even? x0) ‘() ‘(x0))
ys1 = (append ys0 ‘(x1))
ys2 = (ite (even? x1) ys1 ys0)
```

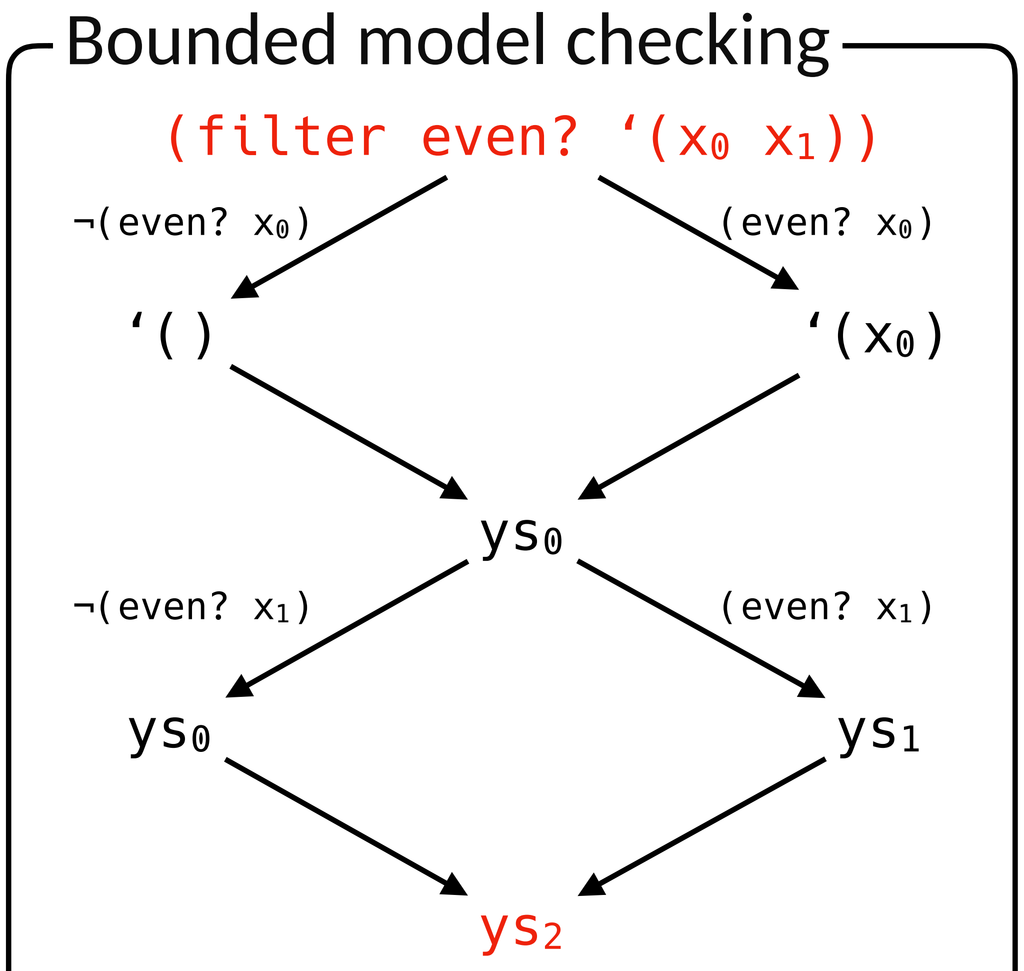
The *symbolic evaluation graph* summarizes branching and merging

Symbolic execution



More states, but more concrete

Bounded model checking

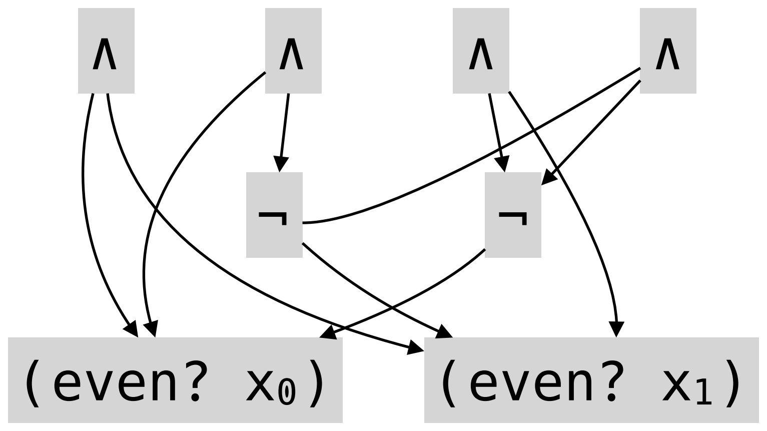


```
ys₀ = (ite (even? x₀) '() '(x₀))
ys₁ = (append ys₀ '(x₁))
ys₂ = (ite (even? x₁) ys₁ ys₀)
```

Fewer states but less concrete

The *symbolic heap* shows how symbolic values are used

Symbolic execution

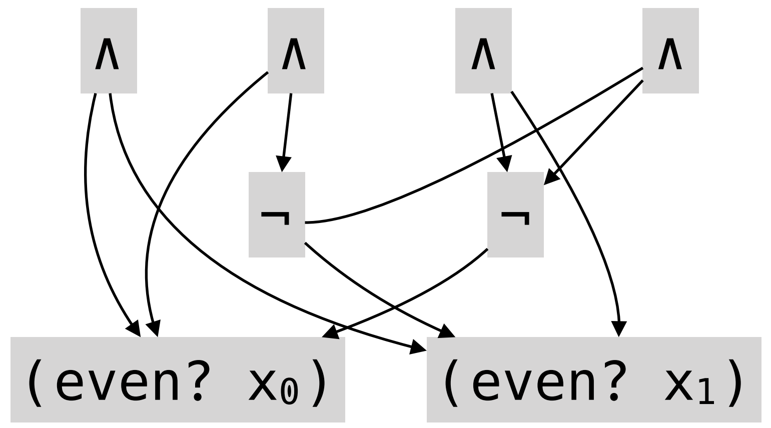


Symbolic heap

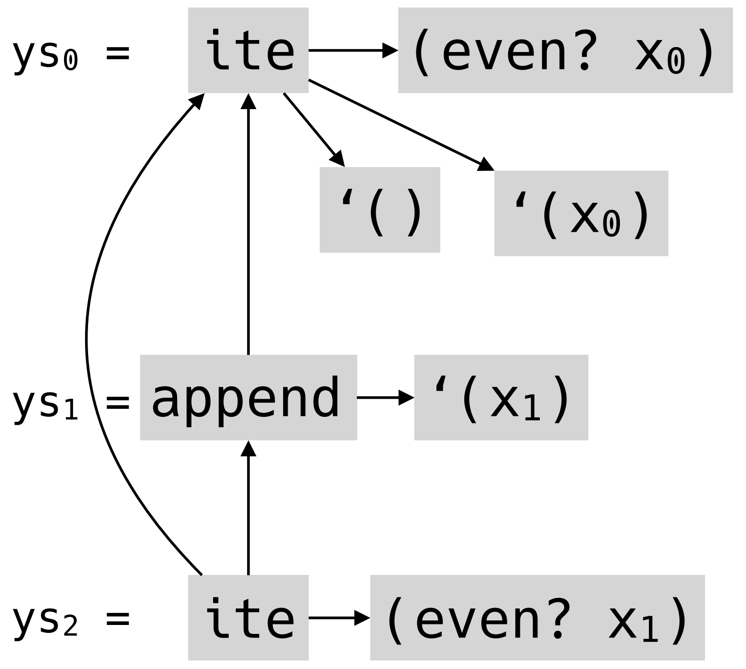
- Nodes are symbolic terms
- Edges are sub-terms

The *symbolic heap* shows how symbolic values are used

Symbolic execution



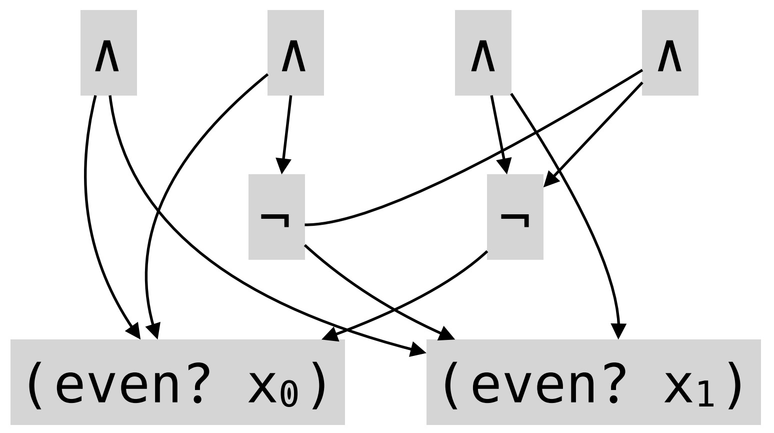
Bounded model checking



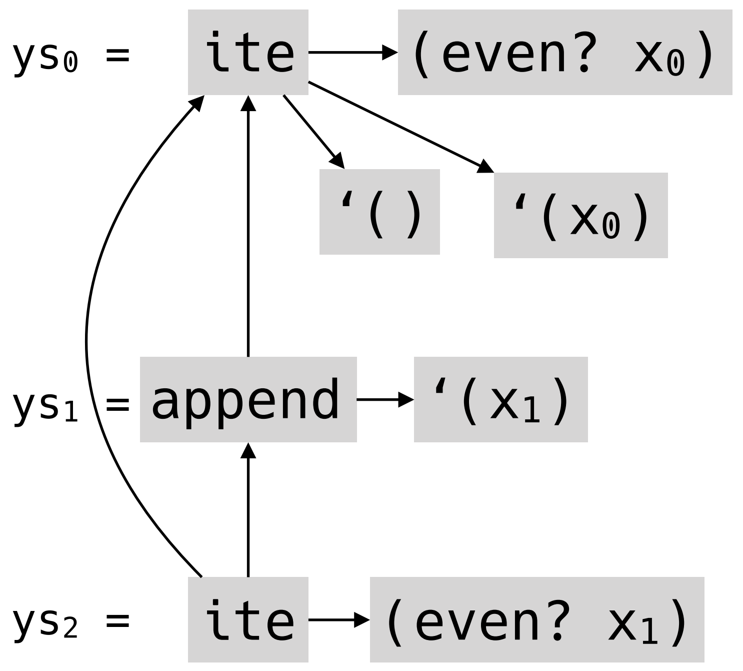
```
ys0 = (ite (even? x0) `() `(x0))
ys1 = (append ys0 `(x1))
ys2 = (ite (even? x1) ys1 ys0)
```

The *symbolic heap* shows how symbolic values are used

Symbolic execution



Bounded model checking

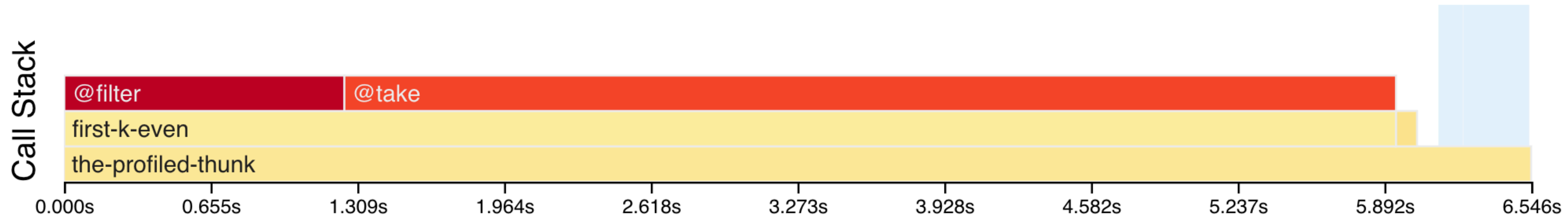


```
ys0 = (ite (even? x0) ‘() ‘(x0))
ys1 = (append ys0 ‘(x1))
ys2 = (ite (even? x1) ys1 ys0)
```

Only conditions in the heap

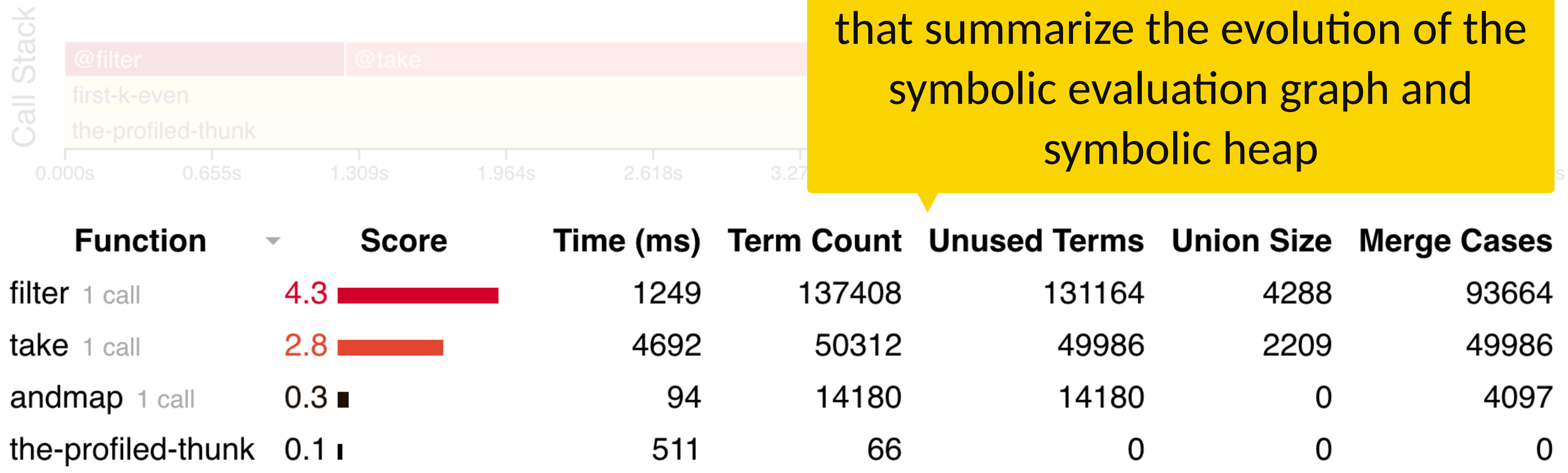
Conditions and values (lists etc.) in the heap

Analyzing symbolic data structures



Function	Score	Time (ms)	Term Count	Unused Terms	Union Size	Merge Cases
filter 1 call	4.3	1249	137408	131164	4288	93664
take 1 call	2.8	4692	50312	49986	2209	49986
andmap 1 call	0.3	94	14180	14180	0	4097
the-profiled-thunk	0.1	511	66	0	0	0

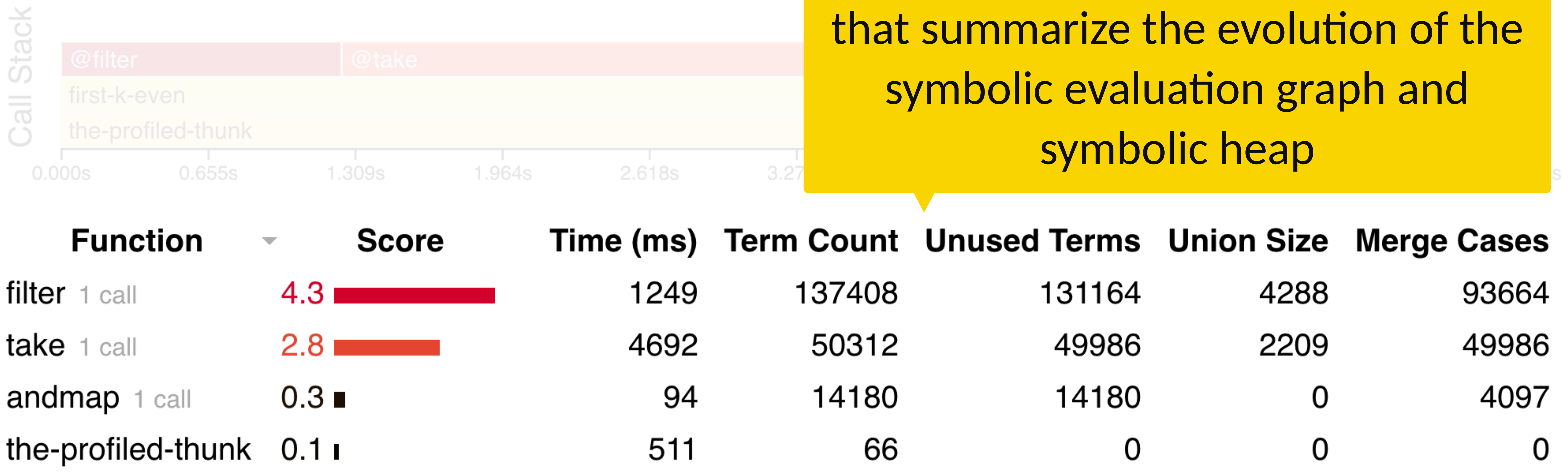
Analyzing symbolic data structures



For each procedure, measure metrics that summarize the evolution of the symbolic evaluation graph and symbolic heap

Function	Score	Time (ms)	Term Count	Unused Terms	Union Size	Merge Cases
filter 1 call	4.3	1249	137408	131164	4288	93664
take 1 call	2.8	4692	50312	49986	2209	49986
andmap 1 call	0.3	94	14180	14180	0	4097
the-profiled-thunk	0.1	511	66	0	0	0

Analyzing symbolic data structures



For each procedure, measure metrics that summarize the evolution of the symbolic evaluation graph and symbolic heap

Function	Score	Time (ms)	Term Count	Unused Terms	Union Size	Merge Cases
filter 1 call	4.3	1249	137408	131164	4288	93664
take 1 call	2.8	4692	50312	49986	2209	49986
andmap 1 call	0.3	94	14180	14180	0	4097
the-profiled-thunk	0.1	511	66	0	0	0

Summarize metrics as a score to rank procedures in the program

Symbolic evaluation anti-patterns

Common issues and repairs

Common anti-patterns and repairs in symbolic evaluation

- Algorithmic mismatch
  - Algorithms or optimizations poorly suited to symbolic evaluation

Common anti-patterns and repairs in symbolic evaluation

Algorithmic mismatch

- Algorithms or optimizations poorly suited to symbolic evaluation

```
(define (list-set lst idx val)
  (match lst
    [(cons x xs)
     (if (= idx 0)
         (cons val xs)
         (cons x (list-set xs (- idx 1) val)))]
    [_ lst]))
```

Terminates early once *idx* is found

Common anti-patterns and repairs in symbolic evaluation

Algorithmic mismatch

Algorithms or optimizations poorly suited to symbolic evaluation

```
(define (list-set lst idx val)
  (match lst
    [(cons x xs)
     (if (= idx 0)
         (cons val xs)
         (cons x (list-set xs (- idx 1) val)))]
    [_ lst]))
```

Terminates early once *idx* is found

Common anti-patterns and repairs in symbolic evaluation

Algorithmic mismatch

- Algorithms or optimizations poorly suited to symbolic evaluation

```
(define (list-set lst idx val)
  (match lst
    [(cons x xs)
     (cons (if (= idx 0) val x)
           (list-set xs (- idx 1) val))]

    [_ lst]))
```

Common anti-patterns and repairs in symbolic evaluation

Algorithmic mismatch

- Algorithms or optimizations poorly suited to symbolic evaluation

```
(define (list-set lst idx val)
  (match lst
    [(cons x xs)
     (cons (if (= idx 0) val x)
           (list-set xs (- idx 1) val))]

    [_ lst]))
```

Always recurse to the end of *lst*

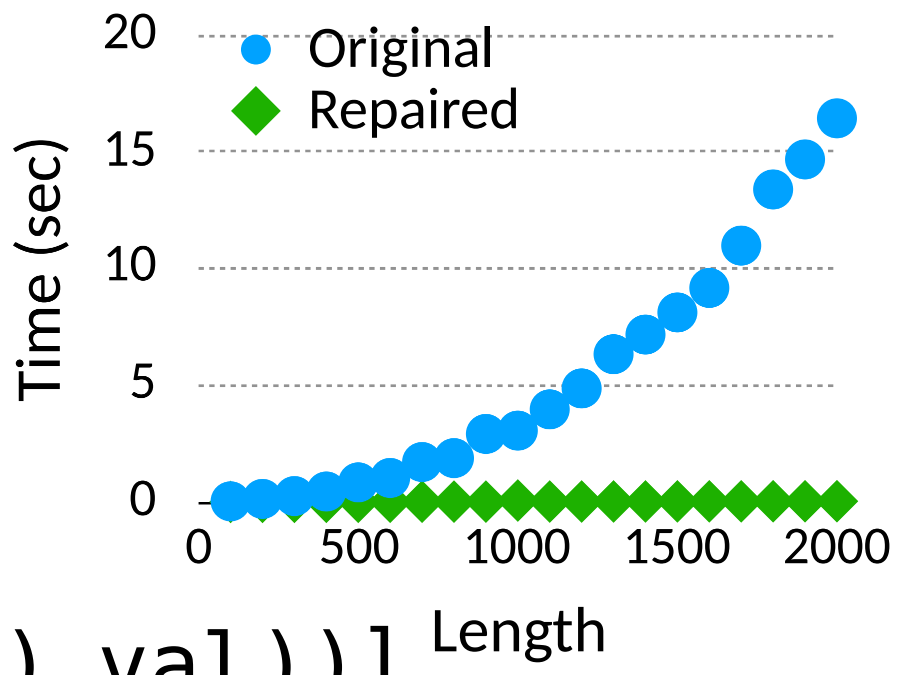
Common anti-patterns and repairs in symbolic evaluation

Algorithmic mismatch

- Algorithms or optimizations poorly suited to symbolic evaluation

```
(define (list-set lst idx val)
  (match lst
    [(cons x xs)
     (cons (if (= idx 0) val x)
           (list-set xs (- idx 1) val))]

    [_ lst]))
```

Always recurse to the end of *lst*



Common anti-patterns and repairs in symbolic evaluation

Algorithmic mismatch

- Algorithms or optimizations poorly suited to symbolic evaluation

Irregular representation

- Data structures of different shapes create different paths

Missed concretization

- Lost opportunities to exploit concrete values

Empirical results

Case studies and evaluation

Three symbolic profilers

We developed two implementations:

- The **Rosette** solver-aided language (Racket)
- The **Jalangi** dynamic analysis framework (JavaScript)

Since publication, based on our work:

- The **Crucible** symbolic simulation library (C, Java, ...) by Galois

Three symbolic profilers

We developed two implementations:

- The **Rosette** solver-aided language (Racket) Today
- The **Jalangi** dynamic analysis framework (JavaScript)

Since publication, based on our work:

- The **Crucible** symbolic simulation library (C, Java, …) by Galois

Actionable: real-world bugs

Case studies on published Rosette-based tools

Tool	**Speedup**
Type system soundness checker [POPL'18]	1.35×
Refinement type checker for Ruby [VMCAI'18]	6×
File-system crash consistency verifier [ASPLOS'16]	24×
Cryptographic protocol verifier [FM'18]	29×
SQL query verifier [CIDR'17]	75×
Safety-critical radiotherapy system verifier [CAV'16]	290×

Multiple patches accepted by developers

Actionable: real-world bugs

Case studies on published Rosette-based tools

Tool	**Speedup**
Type system soundness checker [POPL’18]	1.35×
Refinement type checker for Ruby [VMCAI’18]	6×
File-system crash consistency verifier [ASPLOS’16]	24×
Cryptographic protocol verifier [FM’	29×
SQL query verifier [CIDR’17]	75×
Safety-critical radiotherapy system verifier [CAV’16]	290×

Used in production at the UW Medical Center

Multiple patches accepted by developers

Explainable: study real users

Small user study: 8 Rosette users, asked to find known performance bug in 4 programs

Users solved every task more quickly when they had access to symbolic profiling

- 6 failures without symbolic profiling, none with

Qualitative feedback:

- “gave insight into what Rosette is doing”
- “even more useful on my own code”

Symbolic profiling identifies performance issues in symbolic evaluation

Does my program work on all inputs?

Is there a program that does what I want?

Verification

Synthesis

```
raco symprofile file.rkt
```

https://unsat.org

UNSAT PLSE W