

Strauss: A Specification Miner

Glenn Ammons
Department of Computer Sciences
University of Wisconsin-Madison

How should programs behave?

- Strauss mines temporal specifications
 - for example, always call free after malloc
- Why?
 - To debug
 - To verify
 - To test
 - To modify
 - To understand

2

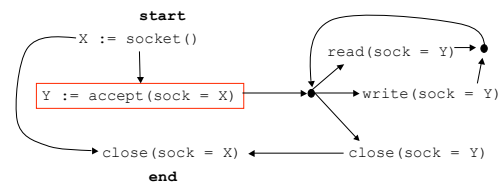
How should programs behave?

- Strauss mines temporal specifications
 - for example, always call free after malloc
- Why?
 - To debug
 - To verify
 - To test
 - To modify
 - To understand
- Specs constrain programs
 - like structured programming, types, etc.

3

Strauss output

For all calls $Y := \text{accept}(\text{sock} = X)$:

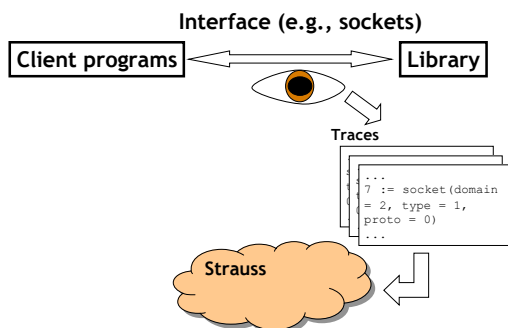


Spec. says what programs should do:

- What order of calls?
- What values in calls?

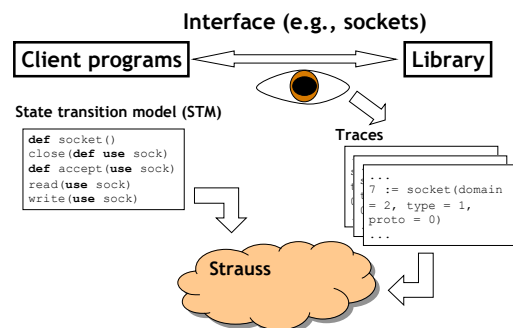
4

Strauss inputs



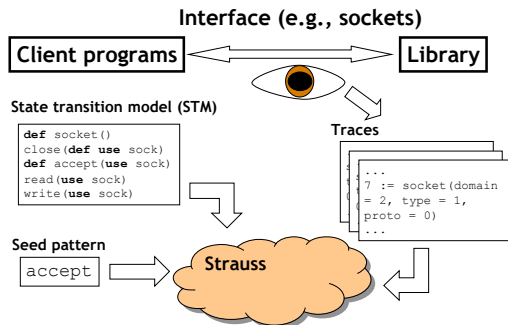
5

Strauss inputs



6

Strauss inputs



7

My contributions

- A dynamic approach to mining
 - analyze communication, not source code
- A tool for mining specifications
 - practical: scalable and partially automatic
 - temporal specs may refer to multiple data values
 - requires little info about code
- Heuristics to tell good code from bad code
 - Birds of a feather flock together
- Found specifications and bugs
 - two goals: summarize and predict

8

Summary of results

- Mined 8 specs. for X11
 - example: XInternAtom should only be called during initialization, not in the event loop.
- Found 61 bugs
 - In widely distributed programs
 - wish (Tk), xpdf, xpilot, ...
 - Included serious races and performance bugs
 - By verifying dynamically (on traces)

9

Related work: obtaining specs.

- By hand
 - From programmers
 - Types
 - Specification languages and annotations
 - From analysts, testers
 - Abstraction tools (Bandera [Dwyer et al.])
- With specification mining

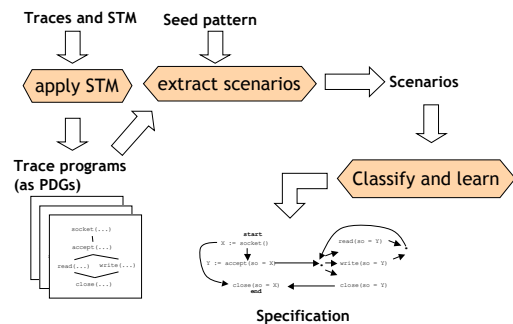
10

Comparing spec. miners

- Strauss
 - Specs: temporal, multiple objects, arbitrary NFAs
 - Mining: from traces, local, no code necessary
 - Goal: specs. for interfaces
- Other work
 - Concurrent work
 - FSMs for Java objects [Whaley, Martin, and Lam]
 - Simple templates [Metacompilation]
 - Loop invariants [ESC/Java]
 - Earlier work
 - Arithmetic properties [Daikon]
 - Cliches [Programmer's Apprentice]
 - Communicating processes [Verisoft]

11

Overview of Strauss



12

Trace + STM = Trace program

```

1 7 := socket(domain = 2,
              type = 1,
              proto = 0)
2 0x100 := malloc(size = 23)
3 8 := accept(sock = 7,
              addr = 0x40,
              addr_len = 0x50)
4 23 := write(sock = 8,
             buf = 0x100,
             len = 23)
5 0 := close(sock = 8)
...

```

State transition model (STM)

```

def socket()
close(def use sock)
def accept(use sock)
read(use sock)
write(use sock)

```

Trace program

Vars: v7, v8

```

1 v7 := socket()
2 skip
3 v8 := accept(sock = v7)
4 write(sock = v8)
5 v8 := close(sock = v8)
...

```

13

Trace + STM = Trace program

```

1 7 := socket(domain = 2,
              type = 1,
              proto = 0)
2 0x100 := malloc(size = 23)
3 8 := accept(sock = 7,
              addr = 0x40,
              addr_len = 0x50)
4 23 := write(sock = 8,
             buf = 0x100,
             len = 23)
5 0 := close(sock = 8)
...

```

State transition model (STM)

```

def socket()
close(def use sock)
def accept(use sock)
read(use sock)
write(use sock)

```

Trace program

Vars: v7, v8

```

1 v7 := socket()
2 skip
3 v8 := accept(sock = v7)
4 write(sock = v8)
5 v8 := close(sock = v8)
...

```

14

Trace + STM = Trace program

```

1 7 := socket(domain = 2,
              type = 1,
              proto = 0)
2 0x100 := malloc(size = 23)
3 8 := accept(sock = 7,
              addr = 0x40,
              addr_len = 0x50)
4 23 := write(sock = 8,
             buf = 0x100,
             len = 23)
5 0 := close(sock = 8)
...

```

State transition model (STM)

```

def socket()
close(def use sock)
def accept(use sock)
read(use sock)
write(use sock)

```

Trace program

Vars: v7, v8

```

1 v7 := socket()
2 skip
3 v8 := accept(sock = v7)
4 write(sock = v8)
5 v8 := close(sock = v8)
...

```

15

A different STM

```

1 7 := socket(domain = 2,
              type = 1,
              proto = 0)
2 0x100 := malloc(size = 23)
3 8 := accept(sock = 7,
              addr = 0x40,
              addr_len = 0x50)
4 23 := write(sock = 8,
             buf = 0x100,
             len = 23)
5 0 := close(sock = 8)
...

```

State transition model (STM)

```

def malloc()
free(def p)
read(use buf)
write(use buf)

```

Trace program

Var: v0x100

```

1 skip
2 v0x100 := malloc()
3 skip
4 write(buf = v0x100)
5 skip
...

```

16

A different STM

```

1 7 := socket(domain = 2,
              type = 1,
              proto = 0)
2 0x100 := malloc(size = 23)
3 8 := accept(sock = 7,
              addr = 0x40,
              addr_len = 0x50)
4 23 := write(sock = 8,
             buf = 0x100,
             len = 23)
5 0 := close(sock = 8)
...

```

State transition model (STM)

```

def malloc()
free(def p)
read(use buf)
write(use buf)

```

Trace program

Var: v0x100

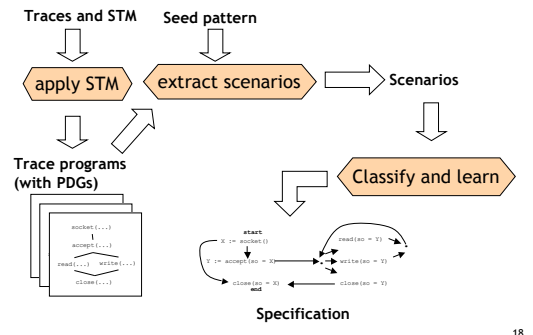
```

1 skip
2 v0x100 := malloc()
3 skip
4 write(buf = v0x100)
5 skip
...

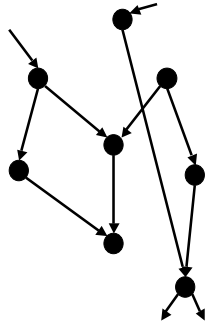
```

17

Overview of Strauss



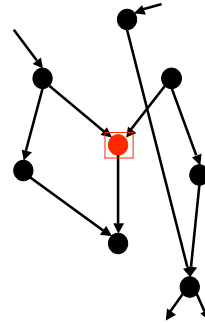
Extracting scenarios from a TPDG



Pick a seed
Slice forwards
Slice backwards
Chop each pair

19

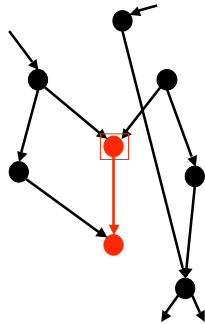
Extracting scenarios from a TPDG



Pick a seed
Slice forwards
Slice backwards
Chop each pair

20

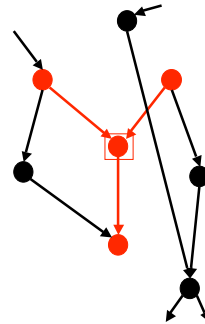
Extracting scenarios from a TPDG



Pick a seed
Slice forwards
Slice backwards
Chop each pair

21

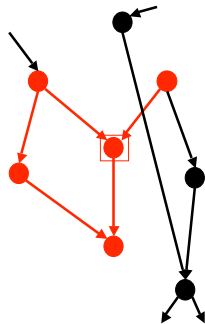
Extracting scenarios from a TPDG



Pick a seed
Slice forwards
Slice backwards
Chop each pair

22

Extracting scenarios from a TPDG



Pick a seed
Slice forwards
Slice backwards
Chop each pair

23

Extracting scenarios: example

Vars: v7, v8

```

1 v7 := socket
2 v8 := accept(sock = v7)
3 write(sock = v8)
4 v8 := close(sock = v8)
5 v7 := close(sock = v7)
    
```

24

Extracting scenarios: example

Vars: v7, v8

Pick a seed

```

1 v7 := socket
2 v8 := accept(sock = v7)
3 write(sock = v8)
4 v8 := close(sock = v8)
5 v7 := close(sock = v7)
    
```

25

Extracting scenarios: example

Vars: v7, v8

Pick a seed
Slice forwards

```

1 v7 := socket
2 v8 := accept(sock = v7)
3 write(sock = v8)
4 v8 := close(sock = v8)
5 v7 := close(sock = v7)
    
```

26

Extracting scenarios: example

Vars: v7, v8

Pick a seed
Slice forwards
Slice backwards

```

1 v7 := socket
2 v8 := accept(sock = v7)
3 write(sock = v8)
4 v8 := close(sock = v8)
5 v7 := close(sock = v7)
    
```

27

Extracting scenarios: example

Vars: v7, v8

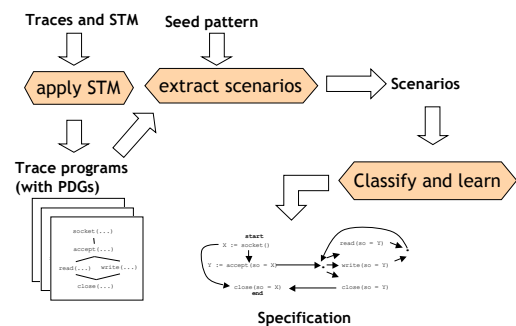
Pick a seed
Slice forwards
Slice backwards
Chop each pair

```

1 v7 := socket
2 v8 := accept(sock = v7)
3 write(sock = v8)
4 v8 := close(sock = v8)
5 v7 := close(sock = v7)
    
```

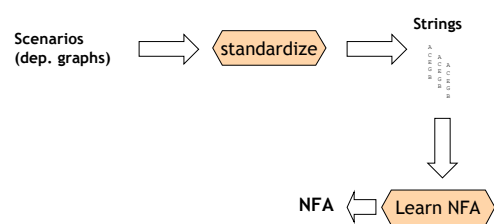
28

Overview of Strauss



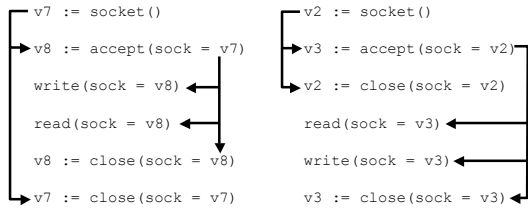
29

Overview of learning



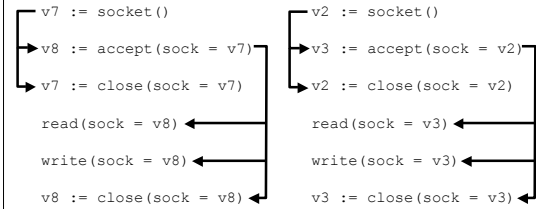
30

Standardizing scenarios



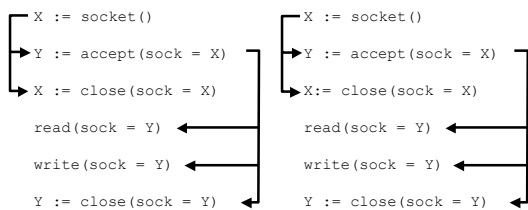
31

Standardizing scenarios



32

Standardizing scenarios



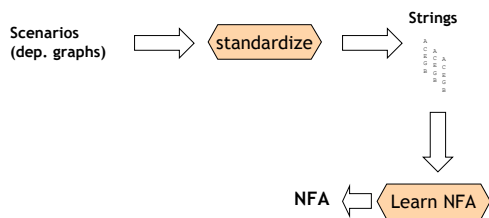
33

Standard scenario = string

X := socket()	A	X := socket()
Y := accept(sock = X)	B	Y := accept(sock = X)
X := close(sock = X)	C	X := close(sock = X)
read(sock = Y)	D	read(sock = Y)
write(sock = Y)	E	write(sock = Y)
Y := close(sock = Y)	F	Y := close(sock = Y)

34

Overview of learning



35

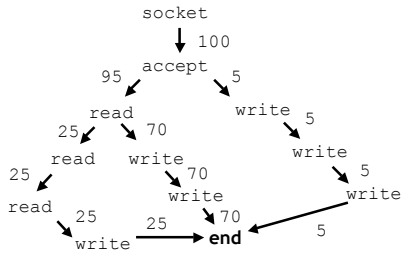
Raman's PFSA algorithm

1. Build a weighted retrieval tree
2. Merge similar states

36

Raman's PFSA algorithm

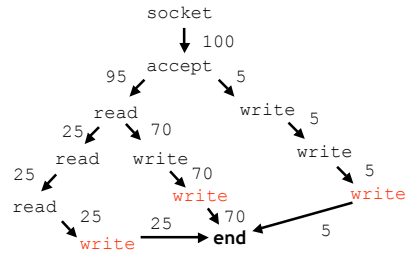
1. Build a weighted retrieval tree
2. Merge similar states



37

Raman's PFSA algorithm

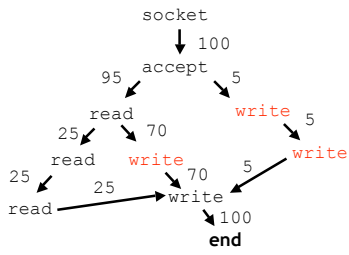
1. Build a weighted retrieval tree
2. Merge similar states



38

Raman's PFSA algorithm

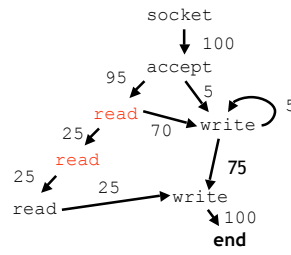
1. Build a weighted retrieval tree
2. Merge similar states



39

Raman's PFSA algorithm

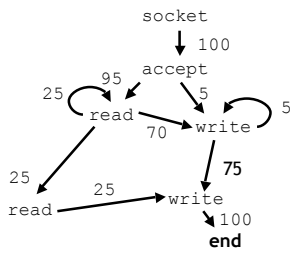
1. Build a weighted retrieval tree
2. Merge similar states



40

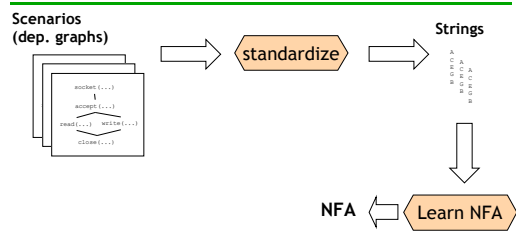
Raman's PFSA algorithm

1. Build a weighted retrieval tree
2. Merge similar states



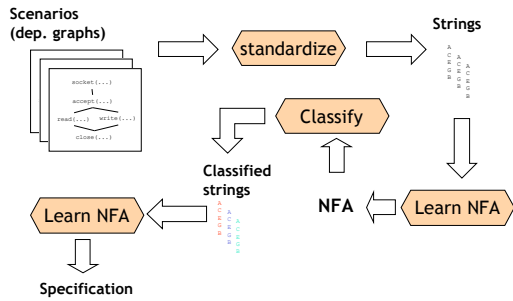
41

Overview of learning



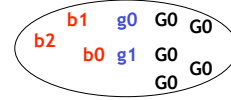
42

Overview of learning



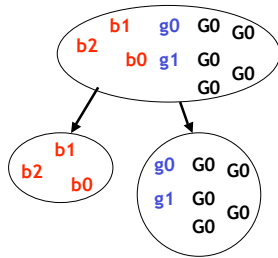
43

Classifying scenarios



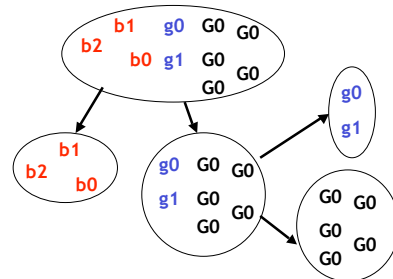
44

Classifying scenarios



45

Classifying scenarios



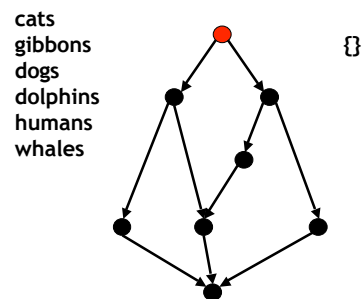
46

Concept analysis: mammals

	4-legged	hairy	smart	marine	thumbed
cats	yes	yes			
dogs	yes	yes			
dolphins			yes	yes	
gibbons		yes	yes		yes
humans			yes		yes
whales			yes	yes	

47

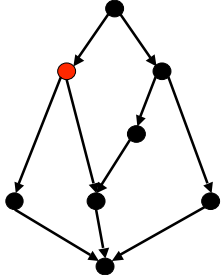
Concept analysis: mammals



48

Concept analysis: mammals

cats
gibbons
dogs

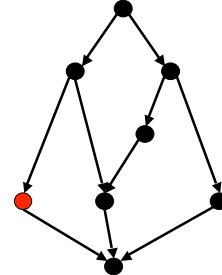


hairy

49

Concept analysis: mammals

cats
dogs



4-legged
hairy

50

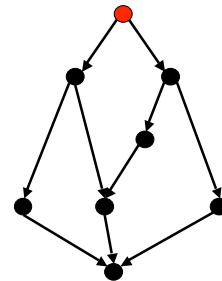
Concept analysis: scenarios

	Takes transition 0	Takes transition 1	Takes transition 2	Takes transition 3	Takes transition 4
Scen. 0	yes	yes			
Scen. 1	yes	yes			
Scen. 2			yes	yes	
Scen. 3		yes	yes		yes
Scen. 4			yes		yes
Scen. 5			yes	yes	

51

Concept analysis: scenarios

scen. 0
scen. 1
scen. 2
scen. 3
scen. 4
scen. 5

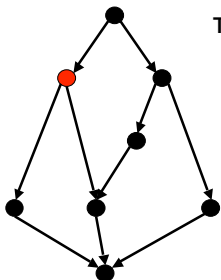


{}

52

Concept analysis: scenarios

scen. 0
scen. 1
scen. 2

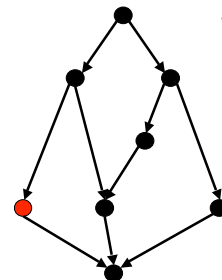


Takes transition 1

53

Concept analysis: scenarios

scen. 0
scen. 2



Takes transition 0
Takes transition 1

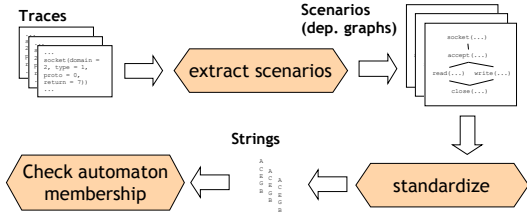
54

Where to find bugs?

- in programs (static verification)?
- or in traces (dynamic verification)?

55

How Strauss verifies a spec



56

Experimental results

Mined and verified 7 published X11 specs and one new spec. (90 traces from 72 programs)

Rule	Scenarios		STM	States	Edges
	Good	Bad			
XSetSelectionOwner	92	22	16	5	8
XtOwnSelection	167	5	16	5	9
XGetSelection	52	52	3	5	4
XInternAtom (negative)	1719	13959	17	7	14
ParseTransTable	2410	0	4	4	4
RemoveTimeOut	803	0	3	5	5
XPutImage	2614	207	9	7	8
ParseAccelTable	39	4	9	21	21

57

Bugs found

1. X11 selection protocol specs.:
 - Found 17 bugs total.
 - English spec is buggy!
2. XInternAtom should be called during initialization, not in the event loop
 - 42 buggy programs (out of 72); degree varied
- XPutImage spec. (more on this later)
 - 2 different bugs! But, also 2 false positives.
- ParseAccelTable
 - 1 false positive

58

Classification was useful

Rule	# concepts inspected
XSetSelectionOwner	3
XtOwnSelection	3
XGetSelection	2
XInternAtom (negative)	3
ParseTransTable	1
RemoveTimeOut	1
XPutImage	3
ParseAccelTable	7

59

XSetSelectionOwner

English: Pass XSetSelectionOwner the timestamp from the last event.

```

(event.time = X) := XNextEvent()
XFilterEvent(event.time = X)
XtDispatchEvent(event.time = X)
(event.time = X) := XCheckWindowEvent()
cb_XtActionProc(event.time = X)
XSetSelectionOwner(time = X)
  
```

For all calls XSetSelectionOwner(time = X)

60

XInternAtom

English: Don't call XInternAtom in the event loop.

```
cb_XtEventHandler(event.display = X)
cb_XtActionProc(event.display = X)
(event.display = X) := XtAppNextEvent()
(event.display = X) := XNextEvent()
XFilterEvent(event.display = X)
XtCallActionProc(event.display = X)
(event.display = X) := XWindowEvent
XtDispatchEvent(event.display = X)
(event.display = X) := XCheckWindowEvent()
XInternAtom(display = X)
```

For all calls XInternAtom(display = X)

61

XPutImage

English: The image and GC passed to XPutImage must have been created on the same display.

```
Z := XCreateGC(display = X)
↓
Y := XCreateImage(display = X)
Y := XShmCreateImage(display = X)
XPutImage(display = X, image = Y, gc = Z)
```

For all calls
XPutImage(display = X, image = Y, gc = Z)

62

Conclusions

- **Strauss is**
 - local
 - scalable
 - dynamic
- **Strauss finds**
 - real specs.
 - real bugs
- **Does Strauss**
 - generalize well?
 - work for others?
 - do too much or too little?

63

Suggestions for future work

- **Mining state transition models**
- **Beyond mining from traces**
 - on-line
 - profile-based
 - static
- **Language support**
 - for example, ESC/Java and loop invariants

64

My publications

- **On Strauss.**
 - Mining specifications. With Rastislav Bodik and James R. Larus. POPL02.
- **Path profiling**
 - Improving data-flow analysis with path profiles. With James R. Larus. PLDI98. Selected for *20 Years of PLDI (1979-1999)*.
 - Exploiting hardware performance counters with flow and context sensitive profiling. With Tom Ball and James R. Larus. PLDI97.

65

End of talk

66