 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

---

Specs constrain programs  
- like structured programming, types, etc.

---

Strauss output

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

```
start
X := \text{socket}()
```

```
Y := \text{accept}(\text{sock} = X)
write(\text{sock} = Y)
```

```
\text{close}(\text{sock} = X)
Close(\text{sock} = Y)
```

```
end
```

Spec. says what programs should do:
- What order of calls?
- What values in calls?

---

Strauss inputs

Interface (e.g., sockets)

Client programs  
- Library

Traces

```
straus: socket(domain = 2, type = 1, proto = 0, return = 7)
```

---

State transition model (STM)

```python
def socket():
    close(def use sock)
def accept(use sock)
def close(use sock)
def read(use sock)
def write(use sock)
```

---

Client programs  
- Library

Interface (e.g., sockets)

Traces

```
straus: socket(domain = 2, type = 1, proto = 0, return = 7)
```

```
straus: socket(domain = 2, type = 1, proto = 0, return = 7)
```

---

State transition model (STM)

```python
def socket():
    close(def use sock)
def accept(use sock)
def close(use sock)
def read(use sock)
def write(use sock)
```
### Strauss inputs

**Interface (e.g., sockets)**

- **Client programs**
- **Library**

#### State transition model (STM)

```
def socket()
  def close(so)
  def accept(so)
  def read(so)
  def write(so)
  def close()

X := socket()
Y := accept(so = X)
close(so = Y)
close(so = X)
```

**Seed pattern**

- **Strauss**

### 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

### 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`, `xpliots`, ...
  - Included serious races and performance bugs
  - By verifying dynamically (on traces)

### 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**

### 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 [Natalia Kohn]
  - Loop invariants [ESC/Java]
  - Earlier work
    - Arithmetic properties [Dakon]
    - Cliches [Programmer’s Apprentice]
    - Communicating processes [Verisoft]

### Overview of Strauss

- **Traces and STM**
- **Seed pattern**
- **Scenarios**
- **Classify and learn**
- **Specification**

---

**Note:** The diagram indicates the flow of information and the process of mining specifications using Strauss. The flowchart shows the interaction between client programs, the library, and the state transition model (STM) for socket operations. The summary highlights the contributions of mining temporal specifications on widely distributed programs, identifying bugs and races, and the related work in obtaining specifications.
Trace + STM = Trace program

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)
...

Trace + STM = Trace program

State transition model (STM)

```
def socket()
    malloc(size)
def accept(use sock)
    socket(domain)
    accept(sock = 7, addr = 0x00, addr_len = 0x50)
    write(sock = 8, buf = 0x100, len = 23)
    close(sock = 8)
```

Trace program

Vars: v7, v8

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

A different STM

State transition model (STM)

```
def socket()
    malloc()
def accept(use sock)
    socket(domain)
    accept(sock = 7, addr = 0x00, addr_len = 0x50)
    write(sock = 8, buf = 0x100, len = 23)
    close(sock = 8)
```

Trace program

Vars: v0x100

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

Overview of Strauss

Traces and STM

apply STM

extract scenarios

Scenarios

Classify and learn

Specification
Extracting scenarios from a TPDG
Pick a seed
Slice forwards
Slice backwards
Chop each pair

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)
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)

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)

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)

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)

Overview of Straus

- Traces and STM
- Seed pattern
- apply STM
- extract scenarios
- Trace programs (with PDGs)
- Classify and learn
- Specify
- Scenarios

Overview of learning

- Scenarios (dep. graphs)
- standardize
- Strings
- NFA
- Learn NFA
Standardizing scenarios

\[ v_7 := \text{socket}() \]
\[ v_8 := \text{accept}(\text{sock} = v_7) \]
\[ \text{write}(\text{sock} = v_8) \]
\[ \text{read}(\text{sock} = v_8) \]
\[ v_8 := \text{close}(\text{sock} = v_8) \]
\[ v_7 := \text{close}(\text{sock} = v_7) \]

Standardizing scenarios

\[ v_7 := \text{socket}() \]
\[ v_8 := \text{accept}(\text{sock} = v_7) \]
\[ v_3 := \text{accept}(\text{sock} = v_2) \]
\[ v_2 := \text{close}(\text{sock} = v_2) \]
\[ \text{read}(\text{sock} = v_3) \]
\[ \text{write}(\text{sock} = v_3) \]
\[ v_8 := \text{close}(\text{sock} = v_8) \]
\[ v_3 := \text{close}(\text{sock} = v_3) \]

Overview of learning

Raman’s PFSA algorithm

1. Build a weighted retrieval tree
2. Merge similar states
Raman’s PFSA algorithm

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

socket

95
accept
5

70
write
5

25
read

25
write

end

Overview of learning

Strings

standardize

Scenarios

(dep. graphs)

NFA

Learn NFA
Overview of learning

Classifying scenarios

Classifying scenarios

Concept analysis: mammals

Concept analysis: mammals
Concept analysis: mammals

- cats
- gibbons
- dogs

4-legged hairy

cats

dogs

Concept analysis: scenarios

<table>
<thead>
<tr>
<th>Scen.</th>
<th>Takes transition 0</th>
<th>Takes transition 1</th>
<th>Takes transition 2</th>
<th>Takes transition 3</th>
<th>Takes transition 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>yes</td>
<td>yes</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>yes</td>
<td>yes</td>
<td></td>
<td>yes</td>
</tr>
</tbody>
</table>

Concept analysis: scenarios

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

Takes transition 1

Takes transition 0

Takes transition 1
Where to find bugs?

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

How Strauss verifies a spec

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

Experimental results

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

<table>
<thead>
<tr>
<th>Rule</th>
<th>Scenarios</th>
<th>STM Biases</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSetSelectionOwner</td>
<td>92</td>
<td>22</td>
<td>16</td>
</tr>
<tr>
<td>XIinternalSelection</td>
<td>167</td>
<td>5</td>
<td>16</td>
</tr>
<tr>
<td>XGetSelection</td>
<td>52</td>
<td>52</td>
<td>3</td>
</tr>
<tr>
<td>XIinternalAtom (negative)</td>
<td>1719</td>
<td>13859</td>
<td>17</td>
</tr>
<tr>
<td>ParseTransTable</td>
<td>2410</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>RemoveTimeOut</td>
<td>803</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>XPutImage</td>
<td>2014</td>
<td>207</td>
<td>9</td>
</tr>
<tr>
<td>ParseAccelTable</td>
<td>39</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

Classification was useful

<table>
<thead>
<tr>
<th>Rule</th>
<th># concepts inspected</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSetSelectionOwner</td>
<td>3</td>
</tr>
<tr>
<td>XIinternalSelection</td>
<td>3</td>
</tr>
<tr>
<td>XGetSelection</td>
<td>2</td>
</tr>
<tr>
<td>XIinternalAtom (negative)</td>
<td>3</td>
</tr>
<tr>
<td>ParseTransTable</td>
<td>1</td>
</tr>
<tr>
<td>RemoveTimeOut</td>
<td>1</td>
</tr>
<tr>
<td>XPutImage</td>
<td>3</td>
</tr>
<tr>
<td>ParseAccelTable</td>
<td>7</td>
</tr>
</tbody>
</table>

XSetSelectionOwner

English: Pass XSetSelectionOwner the timestamp from the last event.

(event.time = X) := XNextEvent()
XFilterEvent(event.time = X)
XDispatchEvent(event.time = X)
(event.time = X) := XCheckWindowEvent()
ch_XtActionProc(event.time = X)
XSetSelectionOwner(time = X)

For all calls XSetSelectionOwner(time = X)
**XInternAtom**

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

\[
\begin{align*}
\text{cb}_\text{XtEventHandler}(\text{event.display} = X) := \text{XAppNextEvent}() \\
\text{cb}_\text{XtActionProc}(\text{event.display} = X) := \text{XNextEvent}() \\
\text{XFilterEvent}(\text{event.display} = X) := \text{XFilterEvent}() \\
\text{XtCallActionProc}(\text{event.display} = X) := \text{XCheckWindowEvent}() \\
\text{XtDispatchEvent}(\text{event.display} = X) := \text{XWindowEvent}
\end{align*}
\]

For all calls XInternAtom(display = X)

**XPutImage**

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

\[
\begin{align*}
Z := \text{XCreateGC}(\text{display} = X) \\
Y := \text{XCreateImage}(\text{display} = X) \\
Y := \text{XShmCreateImage}(\text{display} = X) \\
\text{XPutImage}(\text{display} = X, \text{image} = Y, \text{gc} = Z)
\end{align*}
\]

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

---

**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?

**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

**My publications**

- On Strauss.
  - Mining specifications. With Rastislav Bodik and James R. Larus. POPL02.
- Path profiling
  - Exploiting hardware performance counters with flow and context sensitive profiling. With Tom Ball and James R. Larus. PLDI97.

**End of talk**