Carving Differential Unit Test Cases from System Test Cases

Sebastian Elbaum, Hui Nee Chin, Matthew B. Dwyer, Jonathan Dokulil

ESQuaReD

University of Nebraska - Lincoln, USA

This work was partially supported by the U.S. National Science Foundation and IBM

Web Application
System test $St$

- Start & Load
- Start
- Send Inputs

http request

html response

- Check

Web server

Web Application

SQL server

Carving for $m$ as exercised by $St$

- Start & Load
- Start
- Send Inputs

http request

html response

- Check

Web server

Web Application

SQL server
Carve program state $S$ before executing $m$:

- Start & Load
- Start
- Send Inputs

Web server $\rightarrow$ Web Application $\rightarrow$ SQL server

http request $\rightarrow$ html response

- Check $S_{pre}$

Carve program state $S$ after executing $m$:

- Start & Load
- Start
- Send Inputs

Web server $\rightarrow$ Web Application $\rightarrow$ SQL server

http request $\rightarrow$ html response

- Check $S_{post}$

Carved Test Case
Scenario: $m$ evolved to $m'$

3 Reasons to Carve DUTs

- Lack good unit test suite
- Have strong system suite
  - Evolved beyond unit test suite
  - Exercises interesting interactions
- Have inefficient system suite
  - Too cumbersome for fault isolation
  - Too coarse for regression testing
  - Too expensive to set up and execute
• Start & Load
• Start
• Send Inputs

http request

Web server
Web Application
SQL server

html response

• Check

• Start & Load
• Start
• Send Inputs

http request

Web server
Web Application
SQL server

html response

Purchase.Sum

P1 → P2 → P3

Request.Process

Purchase

P1

P2

P3

...
Projections

- **Insight:** not all parts of *Spre* are necessary!
- **Projections:** preserve selected parts of *Spre*
  - K-bounded reachable projections
  - Interface reachable projections
  - May-reference objects in the heap
  - Trace-based projections
  - ...
Using Projections

- Reduce size of DUTs

\[ S_{pre} \xrightarrow{\pi} S_{pre} \]
Store relevant \( S_{pre} \)

- Reduce number of DUTs

DUT \( i \)
\[ S_{pre} \xrightarrow{=} S_{pre} \]

DUT \( j \)
\[ S_{pre} \xrightarrow{=} S_{pre} \]
Keep \( i \) or \( j \)

HTTP request
Web server
HTML response
Web Application
SQL server

*Start & Load*
*Start*
*Send Inputs*
*Check*

\[ \text{Request.Process} \rightarrow \text{Purchase.Sum} \rightarrow \text{Spost} \]

*Purchase.Sum*
*P1 → P2 → P3*

\[ \text{...} \]
Sensitivity Adjustment through \textit{diff} Functions

- **Insight:** not all $S_{post}$ differences are relevant
- Differencing functions on $S_{post}$ and $S_{post}'$
  - Return values or ‘this’ object instance
  - Reachable projections
  - Spectra
  - Heap shape

\[
\begin{align*}
S_{post} & \rightarrow S_{post}' \\
S_{post} & \neq S_{post}'
\end{align*}
\]

Framework

Capture

Program

\[S_{pre} \rightarrow S_{post}\]

Carving

Load - Capture

\[S_{post} \rightarrow S_{post}'\]

Replay
Framework

Load - Capture  
Program

DUTs Filter

Projections
State-based  Action-based

Carving

DUT Reduction

Efficiency  Robustness

Load - Capture  
Program

Spre
Spost'

Replay

Load - Capture

Spost

Spre

DUTs Filter

Projections
State-based  Action-based

Carving

DUT Reduction

Efficiency  Robustness
Framework

Load - Capture
Program
DUTs
Filter
DUT Reduction
Projections
State-based
Action-based
Efficiency
Robustness
Carving

Load - Capture
Program
m
Filter
DUTs
Spre
Spost
Spost'
Differencing Functions
State-based
Action-based
Sensitivity
Post-states
Replay

Instantiation of Framework: State-based

Program
m
ContextFactory
ContextBounding
XStream
Filter
pre/post
m:s
Filter
m':s
projection
CustomLoader
bcel
Carver

Bounding Analysis
Side effect

ContextFactory
ContextBounding
XStream
CustomLoader
bcel
Replay

Dut

m':s
m':s
post
m
Options
m
m'
Options

Dif
function

m
m
m
m
Study

- Goal: compare St versus DUTs
- Questions: cost, fault detection, robustness
- Context: regression testing
- Artifact: Sienna -- http://sir.unl.edu
  - Multiple versions
  - Seeded faults
  - System test suite
Study Setup and Design

1. Generate DUTs
   – Carved DUTs on V0
2. Selected tests per version
   – Selected tests exercising changes components in Vi
3. Run/replay test suites
   – Fault free versions of Siena -- oracle
   – Faulty versions of Siena
4. Compare outcome of test suites
   – Execution time on fault free versions
   – Fault detection

Results: Carving Efficiency

- Fully automated carving
  +100 methods, +500 system tests
  – Generated over 20K DUTs
  – Carved complete program state: ~160 minutes, 2GB
- Observations
  – Simple filters were extremely effective
  – Heap structure greatly influences effectiveness of projections to reduce Spre
  – Compression reduced space requirements by 2 orders of magnitude
Results: Replay Efficiency

![Graph showing replay efficiency across different versions](image)

DUTs took 20% of S-selection

Results: Effectiveness

- **DUT** suite detected all faults found by **Sts**

- An instance: passing **Sts** had failing **DUTs**
  - **DUTs** detected a fault not found by corresponding **St**
  - Error did not propagate to system level

- Multiple instances: failing **Sts** without failing **DUT**
  - **DUTs** were not replayable
  - Multitude of **DUTs** compensated for limitations of individual ones
Distilled Findings

- Feasibility and Efficiency
  - Carving is fully automated - no tester participation
  - +5 times faster to execute than system tests
- Effectiveness
  - DUTs suites detected faults found by system tests
    - Exceptions: lower k-depth, structural/interface changes
  - DUTs detected differences missed by system tests
- Sensitivity
  - DUTs detected differences corresponding to legal changes
  - Differencing functions mitigated false positives

Future Work

- Synergy between DUTs/Junit/St tests
- Selective re-carving
- DUTs, Compound-DUTs, From-To-DUTs
- Reading and using DUTs