



# Applying Concept Analysis to User-session-based Testing of Web Applications

Sreedevi Sampath<sup>1</sup>, Sara Sprenkle<sup>1</sup>, Emily Gibson<sup>1</sup>, Lori Pollock<sup>1</sup> and Amie Souter<sup>2</sup>

<sup>1</sup>University of Delaware, <sup>2</sup>Sarnoff Corporation



## User-session-based Testing



Access application

Beta Web Application (v.0.9) Deployment

### Example User Session

```
register.jsp?name=sree&pass=phd
shop.jsp?book_id=1&book_cat=java
```

Record field data

User Sessions

Test v.1.0 with a test suite of user sessions

Web Application (v.1.0)

Fault Detection Report

Coverage Evaluation Report

### Web Applications

Client (HTML)

Server (Java)

Database (MySQL)

Front End

Back End

### Problem:

Reduce cost of maintaining and executing large test suites of user sessions

### Approach:

- Cluster user sessions by concept analysis
- Select user sessions from clusters
- Reduce test suite on-the-fly
- Automate the testing process

## Test Suite Reduction

### Input: User session data

A user session is a sequence of URLs and name-value pairs.

user session (i.e., IP address) = <url1, url2, url3, url4, url5>

object

attribute

attributes (URLs)

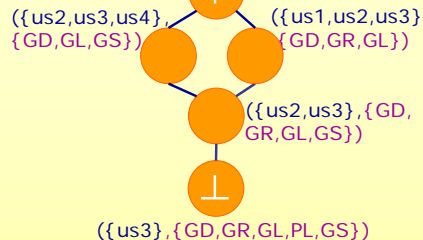
	GD	GR	GL	PL	GS
us1	•	•	•		
us2	•	•	•		•
us3	•	•	•	•	•
us4	•		•		•

### Relation Table:

objects (user sessions)

### Output: Concept Lattice

{{us1,us2,us3,us4},{GD,GL}}



Each node is a concept =  $(O_i, A_j)$  where  $O_i = \{us_1, \dots, us_i\}$  and  $A_j = \{a_1, \dots, a_j\}$  and all  $O_i$  have all  $A_j$  in common and vice versa

Lattice shows partial order of concepts

### Target Criterion

Cover all URLs of application while maintaining use case representation

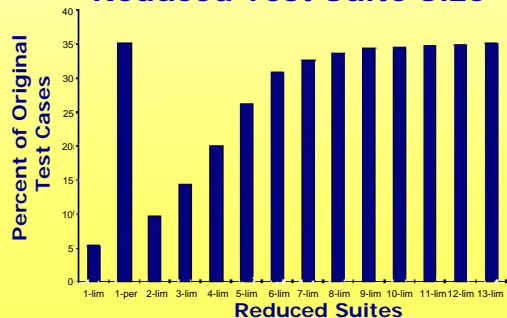
### Reduction Heuristics

Select user session from

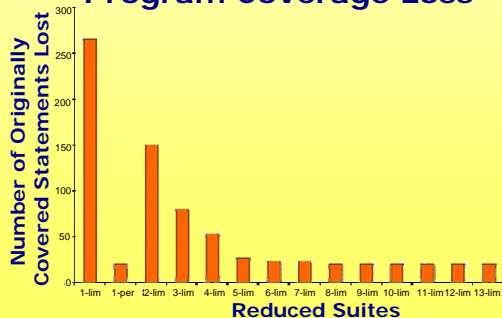
- 1-lim:** bottom node and next-to-bottom nodes. In this example: {us2, us3}
- 1-per:** each node from bottom to top. In this example: {us2, us3, us1, us4}
- k-lim:** each node from bottom to k-levels. In this example: k= 3, {us3, us2, us1, us4}

## Experimental Evaluation

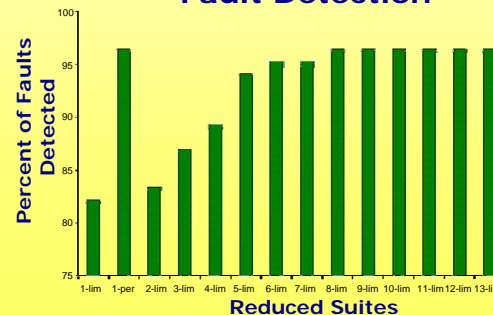
### Reduced Test Suite Size



### Program Coverage Loss



### Fault Detection



### Analysis Summary

- 1-lim** creates the smallest suite but loses code coverage and faults
- 2-lim** is more effective than **1-lim** with small increase in suite size