

# Runtime Error Analysis

## - A Machine Learning Perspective

Praful Mangalath

University of Colorado, Boulder  
Center for Computational Language and Education Research (CLEAR)

April 29th, 2009

# Project Summary

- ▶ Runtime Error Analysis
- ▶ 5535 Deliverables
  - ▶ ~~developed~~ *developing*\* a bug finding toolkit for **C**
  - ▶ Benchmarks on Siemens Test Suite
- ▶ Applied machine learning techniques to detect runtime errors

# Outline of this talk

- ▶ Setup background and explain the problem
- ▶ Demo
- ▶ Details of Implementation
- ▶ Experimental data

# Finding Errors in Code - Static Properties

- ▶ Check for syntactic and static semantic rules
- ▶ Errors to Warnings ratio low
- ▶ Cheap and easy to use
- ▶ Tools : FindBugs, Splint

# Finding Errors in Code - Dynamic Properties

- ▶ Code verification with Abstract Interpretation.
- ▶ Without executing program investigate program behavior
- ▶ Derive dynamic properties from source code
- ▶ mature and sound mathematical basis
- ▶ Tools : BLAST, SLAM (Static Driver Verifier)

# Finding Errors in Code - Dynamic Properties

- ▶ Test driven code verification
- ▶ Identifies only symptoms not cause of error
- ▶ Tracing anomaly to root cause manual time-consuming process
- ▶ Effectiveness limited to test case coverage

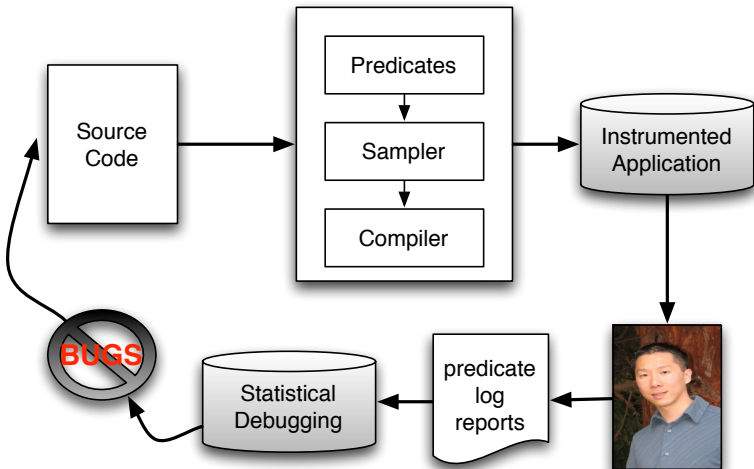
# Verifying Dynamic Properties - Analogy

- ▶ Goal - predict the trajectory of a projectile mid-air
- ▶ Abstract Interpretation
  - ▶ laws of physics (gravity, initial speed, air braking coeff)
  - ▶ transform problem into set of equations
  - ▶ solve by mathematical rules, formal or numeric
- ▶ Test driven
  - ▶ launch many projectiles and record observations
  - ▶ derive empirical laws of motion and error margins
  - ▶ estimate trajectory and report a confidence parameter
- ▶ Mathworks White Paper: *'Verifying Code When Software Reliability is Critical.'*, Paul Barnard, Marc Lalo, & Jim Tung. 2008

# Cooperative Bug Isolation (CBI) Project

- ▶ "Scalable Statistical Bug Isolation" Ben Liblit, Mayur Naik, Alice Zheng, Alex Aiken & Michael Jordan (PLDI 2005)
- ▶ bug-finding post-deployment
- ▶ application in the wild >> writing test cases
- ▶ "Interesting program behavior is expressible as a predicate on a state at a particular program point"
- ▶ Sample predicates from users running these applications  $\approx$   
Yields best test case coverage

# Cooperative Bug Isolation (CBI) Project - Architecture



# Modeling Program Behavior with Predicates

<CODE>

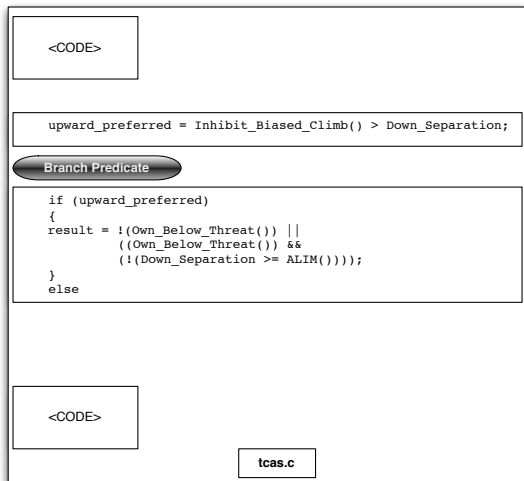
```
upward_preferred = Inhibit_Biased_Climb() > Down_Separation;
```

```
if (upward_preferred)
{
  result = !(Own_Below_Threat()) ||
           ((Own_Below_Threat()) &&
            !(Down_Separation >= ALIM()));
}
else
```

<CODE>

tcas.c

# Modeling Program Behavior with Predicates

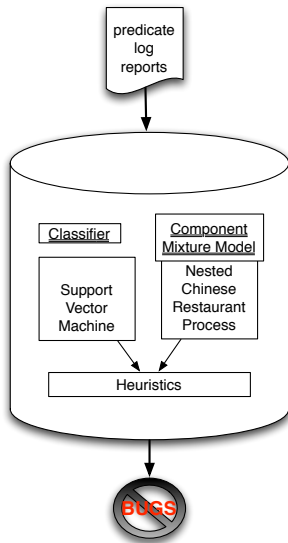


**Figure:** For each conditional, count how many times the branch predicate is false or true. Each branch induces one instrumentation point with a pair of counters.

# Modeling Program Behavior - Execution Profiles

- ▶ Instrumentation sites
  - ▶ branches - pair of counters ( $\text{branch}_{false}, \text{branch}_{true}$ )
  - ▶ bounds - at each assignment site we record max and min values
  - ▶ function-calls - count function entries
- ▶ Collect predicate values with some sampling period
- ▶ Collect execution profile
- ▶ A set of execution profiles (failed & successful runs) is the input to the machine learning component

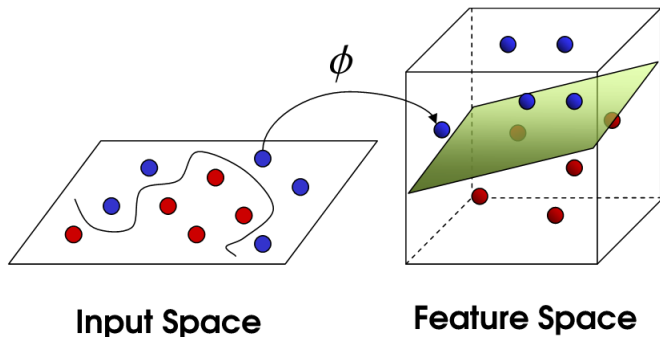
# Machine Learning Components



# Demo

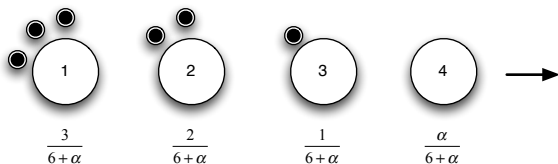
# Classifier Design - Support Vector Machine

- ▶ Goal to use predicates as features to determine failed/successful execution profiles
- ▶ Linear algorithm in feature space is equivalent to non-linear algorithm in input space
- ▶ Ranks predicate features that were significant in making fail/pass decision



# Hierarchical Mixture Model - Nested Chinese Restaurant Process

- ▶ Goal to enable predicates to share clusters
- ▶ Number of clusters varies for each report and needs to be inferred automatically
- ▶ For complex source code with library dependencies clusters could be hierarchical



# Data

- ▶ Siemens Test Suite
- ▶ 132 known expert induced bugs
- ▶ supporting test cases

# Conclusion

- ▶ Machine learning approach to runtime error analysis
- ▶ Tool requires no specialized annotation or expertise to tune/run
- ▶ More data  $\implies$  better performance in ML
- ▶ Instrument real-world application