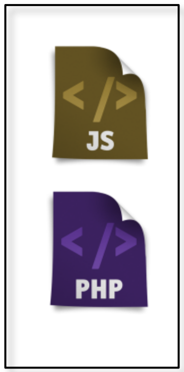
UCSB Verification Lab
Director: Tevfik Bultan

- Research areas
  - *automated verification, program analysis, formal methods, software engineering, computer security*
- Recent research results
  - String analysis for web application vulnerability detection and repair [FMSD,IJFCS,ISSTA'14,ICST'14,ICSE'12,ISSTA'12,ICSE'11,SPIN'11]
  - Data model verification for MVC based web applications [TOSEM, ICSE'15,ASE'15,ICSE'14,ISSTA'13,ASE'12,ISSTA'11]
  - Analyzing message-based interactions in distributed systems [IEEE TSC,ASE'14,FACS'13,POPL'12,VMCAI'12,WWW'11]
  - Automata based model counting constraint solver [CAV'15]
  - Path complexity analysis for programs [ESEC/FSE'15]
- Recent awards
  - ACM SIGSOFT Distinguished Paper Award in ASE'14
  - Best paper and best paper runner-up awards at UCSB GSWC'14
  - ACM SIGSOFT 2015 Outstanding Dissertation Award

Web App Vulnerability Detection & Repair

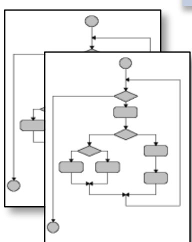
GOAL: To automatically detect and repair vulnerabilities that are caused by input validation and sanitization errors (such as XSS and SQL Injection)



Web App

(1) Sanitizer Extraction

Static and dynamic program analysis to extract input validation and sanitization operations



Sanitizer functions

(2) String Analysis

Automata based string analysis using symbolic fixpoint computations

Vulnerability signatures

(3) Bug Detection and Repair

Differential or policy directed (using attack patterns) bug detection and repair

Bug reports (attack strings) and code patches

Web App Data Model Verification & Repair

GOAL: To automatically detect and repair data model errors in web apps written using MVC based frameworks (such as Ruby on Rails)

Rails code

Formal data model

Formal data model + properties

User specified properties

FOL or Boolean formulas

Bug reports (property violating instances)

Code patches

(1) Model Extraction

Static analysis + instrumented execution

(2) Property Inference

Search for property patterns in data model schema

(3) Logic Translation

Encoding in First Order Logic (unbounded) or Boolean logic (bounded)

(4) Verification

Verification via automated theorem provers or SAT solvers

(5) Data Model Repair

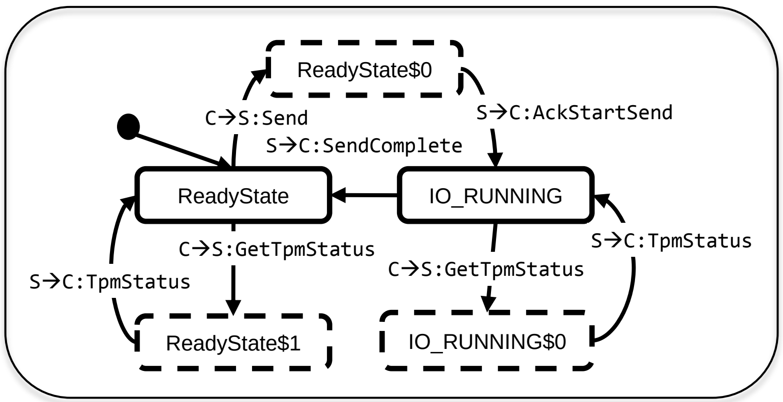
Automated repair based on property patters

Analyzing Message-based Interactions

GOAL: Automated analysis of distributed systems which use message-based communication

APPLICATIONS: Deadlock detection in web services, Erlang programs, Singularity OS processes

Input communication protocol



Message-based communication:

- asynchronous

(using FIFO message buffers)

- synchronous

(rendezvous communication)

Realizability check:
Is the protocol implementable in a distributed manner without deadlocks?

Synchronizability check:
Does the protocol behavior change with synchronous vs. asynchronous communication

Results:

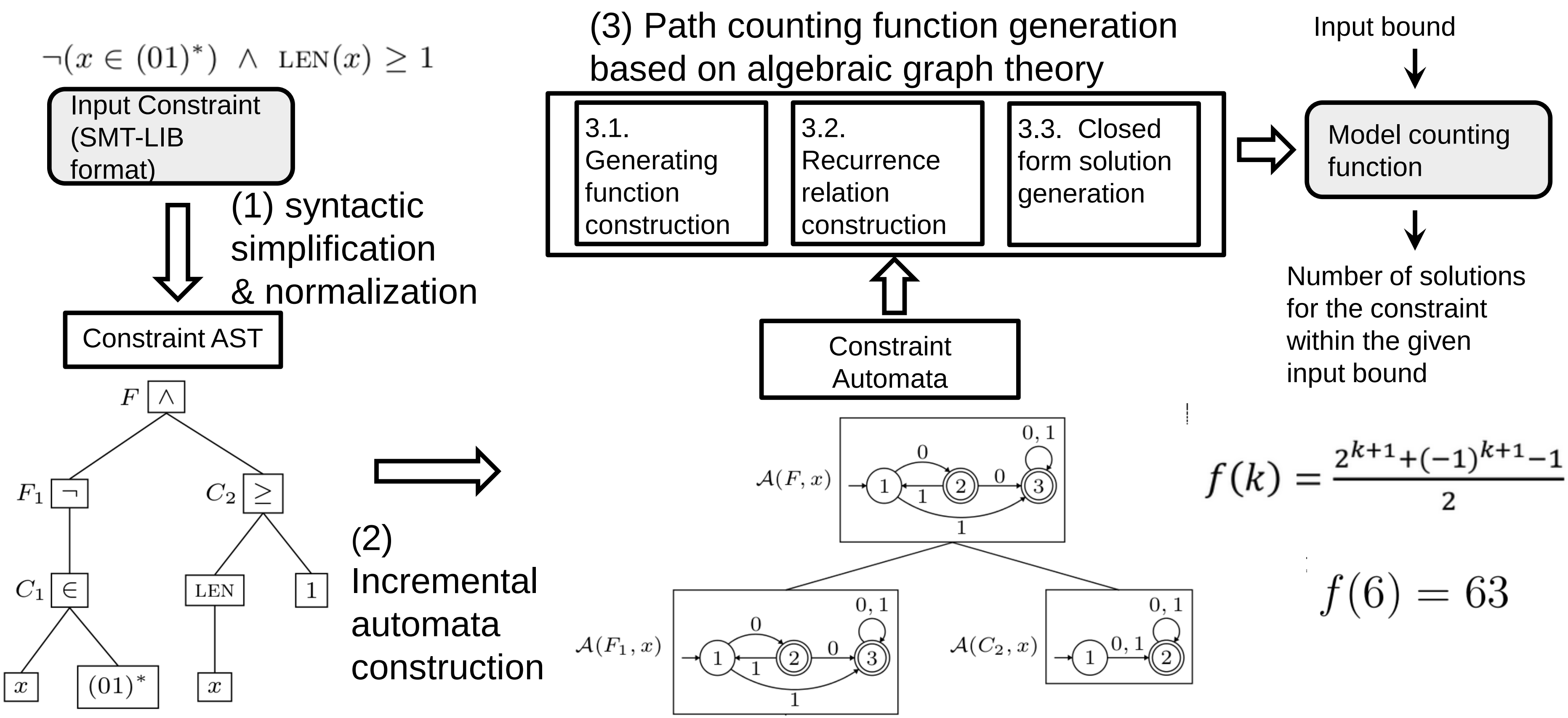
- Realizability and synchronizability checks are decidable
- Identified a subclass of asynchronously communicating systems which can be verified automatically
- Identified a flaw in Singularity OS protocol verification framework

Model Counting Constraint Solver

GOAL: Given a constraint, generate a model counting function that returns the number of solutions within a given bound

APPLICATIONS: Quantitative information flow, probabilistic verification

APPROACH: Construct an automaton that accepts all the solutions to the given constraint, which reduces the model counting to path counting



Computing Path Complexity of Programs

GOAL: Given a program, generate a path complexity function that returns the number of paths in the program within a given depth

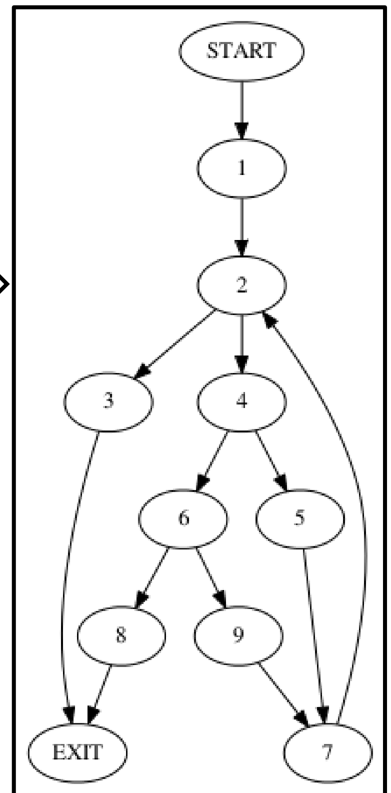
APPLICATIONS: Determining difficulty of path coverage, guidance for verification and testing heuristics

APPROACH: Path counting function generation on the control flow graph

(1) Input Java code

```
private static int binarySearch0(long[] a,
    int fromIndex, int toIndex, long key) {
  int low = fromIndex;
  int high = toIndex - 1;
  while (low <= high) {
    int mid = (low + high) >>> 1;
    long midVal = a[mid];
    if (midVal < key)
      low = mid + 1;
    else if (midVal > key)
      high = mid - 1;
    else
      return mid; // key found
  }
  return -(low + 1);  // key not found.
}
```

(2) Control Flow Graph



(3) Path complexity function

$$path(n) = 6.86\,(1.17)^n + 0.22\,(1.09)^n + 0.13\,(0.84)^n + 2$$

(4) Asymptotic path complexity

$$path(n) = \Theta(1.17^n)$$