A Specification-based Test Generation Framework for RESTful Web Applications

Sujit Kumar Chakrabarti
Software Engineering and Analysis Lab
`https://sealiiitb.github.io/`

IIIT-B

November 17, 2025

Overview

1 Introduction

2 Test the REST (TTR)
- Testing RESTful APIs

3 Variations and Extensions
- Mocking
- Specification Generation

4 Wrap-Up

1 Introduction

2 Test the REST (TTR)
- Testing RESTful APIs

3 Variations and Extensions
- Mocking
- Specification Generation

4 Wrap-Up

Before we start ...

Wep Applications

Webapps are everywhere!

Banking

Healthcare

eCommerce

ERP

eGovernance

Social
Networking

Wep Applications

Characteristics of Webapps

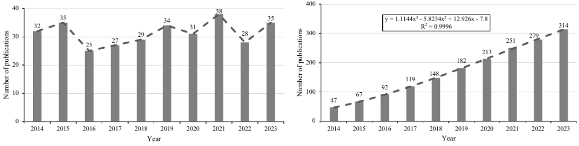
- Important
- Complex
- Distributed
- Dynamic

Desirable Properties

- **Functional properties:** Correct, complete, consistent
- **Non-function properties:** Performance (response time), scalable, available, reliable, secure, compliant, fair, inclusive, sustainable ...

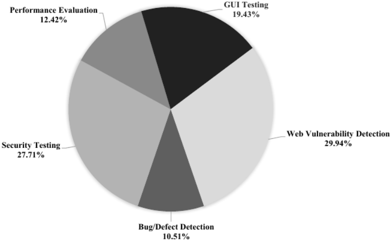
Wep Applications

Testing of Webapps[1]



(a) Number of publications per year.

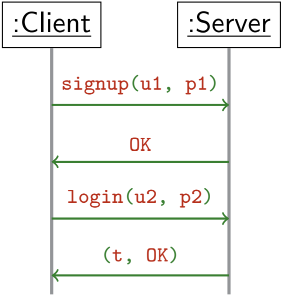
(b) Cumulative number of publications per year.



[1] A Survey on Web Application Testing: A Decade of Evolution - *Tao Li, Rubing Huang, Chenhui Cui, Dave Towey, Lei Ma, Yuan-Fang Li, Wen Xia*

Business Logic Testing

A Web Application



:Client

:Server

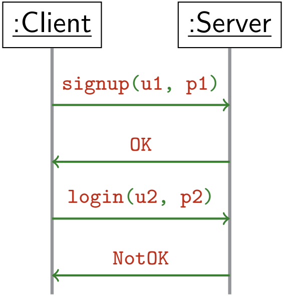
`signup(u1, p1)`

`OK`

`login(u2, p2)`

`(t, OK)`

Signup ✓ Login ✓



:Client

:Server

`signup(u1, p1)`

`OK`

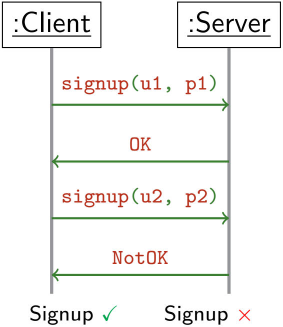
`login(u2, p2)`

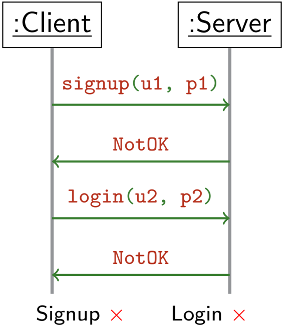
`NotOK`

Signup ✓ Login ×

Business Logic Testing

A Web Application





Business Logic Testing

A Web Application

Characteristics of business logic:

- Involves multiple client-server interactions
- Data dependency between interactions
- Dependency on continually evolving application state
- Too complex! Automation is essential.

Business Logic Testing

A Web Application

Characteristics of business logic:

- Involves multiple client-server interactions
- Data dependency between interactions
- Dependency on continually evolving application state
- Too complex! Automation is essential.

Our approach:

- Formally specify the system
- Automatically generate test cases from system + test specifications

1 Introduction

2 Test the REST (TTR)
- Testing RESTful APIs

3 Variations and Extensions
- Mocking
- Specification Generation

4 Wrap-Up

1 Introduction

2 Test the REST (TTR)

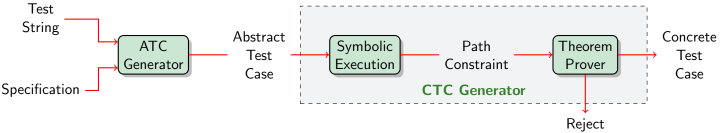
- Testing RESTful APIs

3 Variations and Extensions

- Mocking
- Specification Generation

4 Wrap-Up

Specification based Testing



Formal specification

Signup

- Before signup is called, the DB must not contain the username being currently used.
- A call to signup, if it succeeds, will return with an HTTP OK response code.
- After a successful signup, the DB will now have a record corresponding to the username that's been used for signing up.

Formal specification

Signup

- Before signup is called, the DB must not contain the username being currently used.
- A call to signup, if it succeeds, will return with an HTTP OK response code.
- After a successful signup, the DB will now have a record corresponding to the username that's been used for signing up.

Formal specification

SignupOK	
Precondition	$u \notin dom(U)$
API	$\texttt{signup}(u, p) \rightarrow HttpOK$
Postcondition	$U' = U[u \mapsto p]$

Formal specification

Globals

$U : (string, string) map$
$T : (token, string) map$

Init

$U = \{\}$
$T = \{\}$

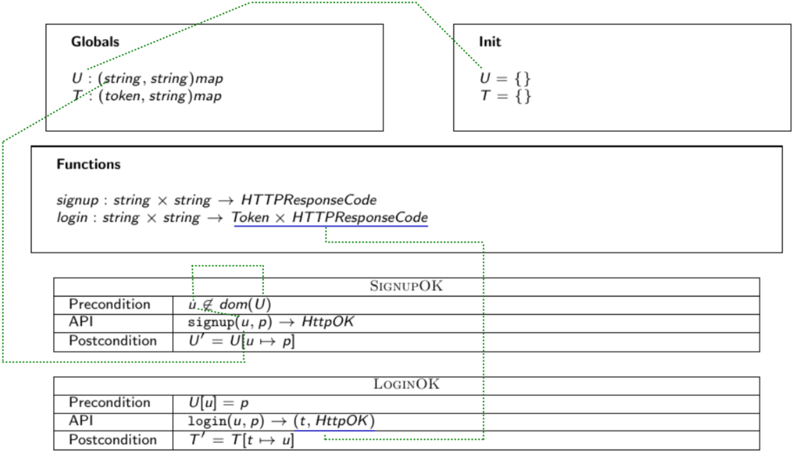
Functions

$signup : string \times string \rightarrow HTTPResponseCode$
$login : string \times string \rightarrow Token \times HTTPResponseCode$

SignupOK	
Precondition	$u \notin dom(U)$
API	$\texttt{signup}(u, p) \rightarrow HttpOK$
Postcondition	$U' = U[u \mapsto p]$

LoginOK	
Precondition	$U[u] = p$
API	$\texttt{login}(u, p) \rightarrow (t, HttpOK)$
Postcondition	$T' = T[t \mapsto u]$

Formal specification



Globals

$U : (string, string)map$
$T : (token, string)map$

Init

$U = \{\}$
$T = \{\}$

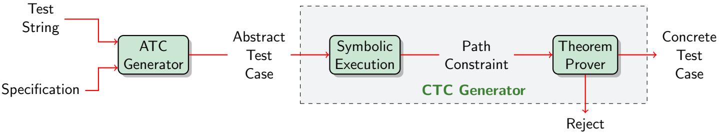
Functions

$signup : string \times string \rightarrow HTTPResponseCode$
$login : string \times string \rightarrow \underline{Token \times HTTPResponseCode}$

	SignupOK
Precondition	$u \notin dom(U)$
API	$\texttt{signup}(u, p) \rightarrow HttpOK$
Postcondition	$U' = U[u \mapsto p]$

	LoginOK
Precondition	$U[u] = p$
API	$\texttt{login}(u, p) \rightarrow \underline{(t, HttpOK)}$
Postcondition	$T' = T[t \mapsto u]$

Specification based Testing



Abstract and Concrete Test Cases

Abstract and Concrete Test Cases

Abstract test case

```
let u1 := input<string>()
let p1 := input<String>()
let u2 := input<String>()
let p2 := input<String>()
let U = new Map<String, String>()
let T = new Map<Token, String>()

assume(u1 ∉  dom(U))
let r1 := signup(u1, p1)
assert(U[u1] = p1)

assume(u2 ∈ dom(U))
let (r2, t) := login(u2, p2)
assert(T[t] = u2)
```

Concrete test case

```
let u1 := "xyz"
let p1 := "abc"
let u2 := "xyz"
let p2 := "abc"
let U = new Map<String, String>()
let T = new Map<Token, String>()


let r1 := signup(u1, p1)
assert(U[u1] = p1)


let (r2, t) := login(u2, p2)
assert(T[t] = u2)
```

The GETATC Algorithm

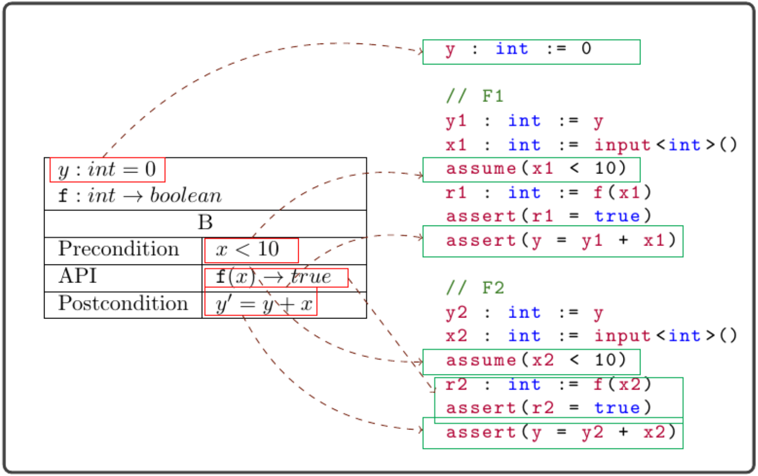
Test string: `f f`

Formal Specification:

$y : \mathit{int} = 0$ f : $\mathit{int} \to \mathit{boolean}$	
Precondition	$x < 10$
API	f$(x) \to \mathit{true}$
Postcondition	$y' = y + x$

```
y : int := 0

// F1
y1 : int := y
x1 : int := input<int>()
assume(x1 < 10)
r1 : int := f(x1)
assert(r1 = true)
assert(y = y1 + x1)

// F2
y2 : int := y
x2 : int := input<int>()
assume(x2 < 10)
r2 : int := f(x2)
assert(r2 = true)
assert(y = y2 + x2)
```

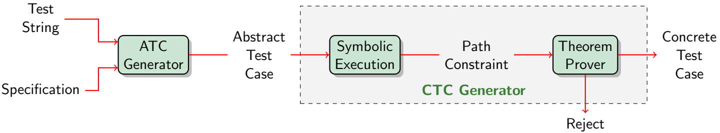
The genATC Algorithm



$y : int = 0$ $\mathtt{f} : int \to boolean$	
B	
Precondition	$x < 10$
API	$\mathtt{f}(x) \to true$
Postcondition	$y' = y + x$

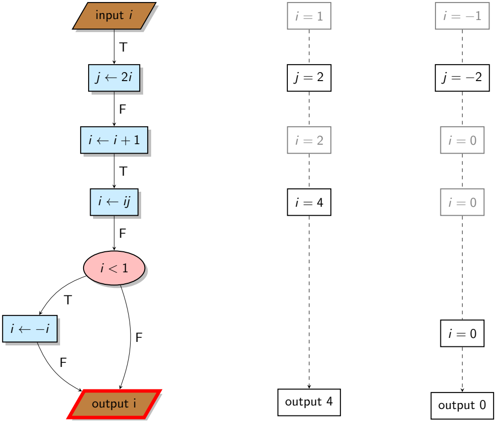
```
y : int := 0

// F1
y1 : int := y
x1 : int := input<int>()
assume(x1 < 10)
r1 : int := f(x1)
assert(r1 = true)
assert(y = y1 + x1)

// F2
y2 : int := y
x2 : int := input<int>()
assume(x2 < 10)
r2 : int := f(x2)
assert(r2 = true)
assert(y = y2 + x2)
```

Specification based Testing



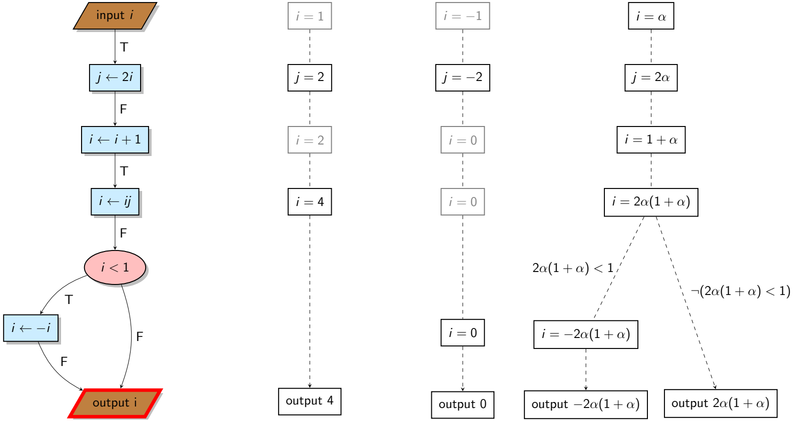
Symbolic Execution

Concrete Execution – Example



Symbolic Execution

Symbolic Execution – Example



Generating Concrete Test Case

```
function $\textsc{genCTC}(t, L, \sigma)$
    if $\neg\textsc{isAbstract}(t)$ then
        return $t$
    else
        $t' \leftarrow \textsc{rewriteATC}(t, L)$
        $L' \leftarrow \textsc{symex}(t', \sigma)$
        return $\textsc{getCTC}(t', L', \sigma)$
```

- $\textsc{isAbstract}(t)$: Returns true if the testcase t has atleast one input command.
- σ: Type environment

Rewriting Abstract Test Case

Considering the abstract test case as a sequence of statements:
$t = [s_1; s_2; ...; s_n]$ and $L = [v_1; v_2; ...; v_m]$

```
function REWRITEATC($t$, $L$)
    if $|t| = 0 \wedge |L| \neq 0$ then raise Error
    match $s_1$ with
    |    case $Input(x) \Rightarrow$
             $s_1' \leftarrow Assign(x, v_1)$
             return $s_1'$ :: REWRITEATC($[s_2; ...; s_n]$ $[v_2; ...; v_m]$)
    |    _ $\Rightarrow$ return $s_1$ :: REWRITEATC($[s_2; ...; s_n]$ $[v_1; ...; v_m]$)
```

Here :: is list construction operator. For example: $1 :: [2; 3] = [1; 2; 3]$

Rewriting Abstract Test Case

Example

t	L	t'
`x := input()`	1	`x := 1`
`y := input()`	2	`x := 2`
`z := input()`		`z := input()`
...		

Symbolic Execution

Considering the abstract test case as a sequence of statements:
$t = [s_1; s_2; \dots; s_n]$

```
  function symex($[s_1, s_2, \dots, s_n]$, $\sigma$)
      $C \leftarrow []$
      for $i = 1$ to $n$ do
          if isReady($s_i$) then
              symexInstr($s_i$, $\sigma$, $C$)
          else
              break
      $pc \leftarrow$ computePathConstraint($C$)
      return solve($pc$)
```

Symbolic Execution

What do we mean by Ready?

$f(x, y)$, $\sigma = [x \mapsto 1, y \mapsto 2]$	✓
$f(x, y)$, $\sigma = [x \mapsto Add(1, X_1), y \mapsto 2]$	×

Symbolic Execution

isReady Function

```
function $\textsc{isReady}(s, \sigma)$
    match $s$ with
    |   case $Assign(x, e) \Rightarrow$ return $\textsc{isReady}(e, \sigma)$
    |   ...
```

```
function $\textsc{isReady}(e, \sigma)$
    match $e$ with
    |   case $Var(x) \Rightarrow$ return $\neg\ \textsc{isSymbolic}(\sigma[x])$
    |   case $Num(n) \Rightarrow$ return $true$
    |   case $FunCall(f, [a_1; \ldots; a_n])$ where $\textsc{isAPI}(f) \Rightarrow$ return $\bigwedge_{i=1}^{n} \textsc{isReady}(a_i)$
    |   case $FunCall(f, [a_1; \ldots; a_n])$ where $\neg\textsc{isAPI}(f) \Rightarrow$ return $true$
    |   ...
```

Symbolic Execution

isSymbolic Function

```
function $\textsc{isSymbolic}(e)$
    match $e$ with
    |    case $Var(\_) \Rightarrow$ return $true$
    |    case $Num(n) \Rightarrow$ return $false$
    |    case $Add(e_1, e_2) \Rightarrow$ return $\textsc{isSymbolic}(e_1) \lor \textsc{isSymbolic}(e_2)$
    |    ...
```

Symbolic evaluation

Instruction

```
function SYMEXINSTR($s$, $\sigma$, $C$)
    match $s$ with
    |    case $Assign(x, e) \Rightarrow \sigma \leftarrow \sigma[x \mapsto \text{SYMEVAL}(e, \sigma)]$
    |    case $Assume(c) \Rightarrow C \leftarrow C \frown \text{SYMEVAL}(c, \sigma)$
    |    $\ldots$
```

Here, $\frown$ stands for addition of an element to a list:

$[1; 2; 3] \frown 4 = [1; 2; 3; 4]$

Symbolic Execution

Computing the path constraint

Expression

```
function SYMEVAL($e$, $\sigma$)
    match $e$ with
    |   case $Var(x) \Rightarrow$ return $\sigma[x]$
    |   case $Num(n) \Rightarrow$ return $\sigma[x]$
    |   case $Add(e_1, e_2) \Rightarrow$ return $Add($SYMEVAL$(e_1, \sigma[x])$, SYMEVAL$(e_2, \sigma[x]))$
    |   ...
```

Computing Path Constraint

```
function COMPUTEPATHCONSTRAINT($[c_1, c_2, ..., c_n]$)
    return $c_1 \wedge c_2 \wedge ... \wedge c_n$
```

The TestGen Algorithm

Symbolic Execution – Run 1

```
y : int := 0

// F1
y1 : int := y
x1 : int := input<int>()
assume(x1 < 10)
r1 : int := f(x1)
assert(r1 = true)
assert(y = y1 + x1)

// F2
y2 : int := y
x2 : int := input<int>()
assume(x2 < 10)
r2 : int := f(x2)
assert(r2 = true)
assert(y = y2 + x2)
```

1. $y = 1$
2. $y1 = 1$
3. $x1 = X_1$
4. $\mathrm{assume}(X_1 < 10)$

The TestGen Algorithm

Symbolic Execution – Run 1

```
y : int := 0

// F1
y1 : int := y
x1 : int := input<int>()
assume(x1 < 10)
r1 : int := f(x1)
assert(r1 = true)
assert(y = y1 + x1)

// F2
y2 : int := y
x2 : int := input<int>()
assume(x2 < 10)
r2 : int := f(x2)
assert(r2 = true)
assert(y = y2 + x2)
```

1. $y = 1$
2. $y1 = 1$
3. $x1 = X_1$
4. assume$(X_1 < 10)$

f needs a concrete value of $x1$!

The TestGen Algorithm

Symbolic Execution – Run 1

```
y : int := 0

// F1
y1 : int := y
x1 : int := input<int>()
assume(x1 < 10)
r1 : int := f(x1)
assert(r1 = true)
assert(y = y1 + x1)

// F2
y2 : int := y
x2 : int := input<int>()
assume(x2 < 10)
r2 : int := f(x2)
assert(r2 = true)
assert(y = y2 + x2)
```

1. $y = 1$
2. $y1 = 1$
3. $x1 = X_1$
4. $\text{assume}(X_1 < 10)$

f needs a concrete value of $x1$!

Solution:

1. Compute path constraint so far: $X_1 < 10$.
2. Give to SMT solver to solve. SAT. Example value: $X_1 = 5$.

The TestGen Algorithm

Abstract Test Case (version 2)

```
y : int := 0
y1 : int := 0
x1 : int := 5
r1 : int := f(x1)
...
...
```

The TestGen Algorithm

Symbolic Execution – Run 2

1. $y = 1$
2. $y1 = 1$
3. $x1 = 5$
4. $f(x1) \rightarrow v_1$
5. $r1 = v_1$
6. assert($r1 = \mathit{true}$)
7. assert($y = y1 + 5$)
8. $y2 = y$
9. $x2 = X_2$
10. assume($X_2 < 10$)

The TestGen Algorithm

Symbolic Execution – Run 2

1. $y = 1$
2. $y1 = 1$
3. $x1 = 5$
4. $f(x1) \rightarrow v_1$
5. $r1 = v_1$
6. assert$(r1 = \mathit{true})$
7. assert$(y = y1 + 5)$
8. $y2 = y$
9. $x2 = X_2$
10. assume$(X_2 < 10)$

f needs a concrete value of $x2$!

The TestGen Algorithm

Symbolic Execution – Run 2

1. $y = 1$
2. $y1 = 1$
3. $x1 = 5$
4. $f(x1) \to v_1$
5. $r1 = v_1$
6. assert$(r1 = \mathit{true})$
7. assert$(y = y1 + 5)$
8. $y2 = y$
9. $x2 = X_2$
10. assume$(X_2 < 10)$

f needs a concrete value of $x2$!

Solution:

1. Compute path constraint so far: $X_2 < 10$.
2. Give to SMT solver to solve. SAT. Example value: $X_2 = 2$.

The TestGen Algorithm

Abstract Test Case (version 3)

```
y : int := 0
y1 : int := 0
x1 : int := 5
r1 : int := f(x1)
assert(r1 = true)
assert(y = y1 + x1)
y2 : int := y
x2 : int := 2
r2 : int := f(x2)
assert(r2 = true)
assert(y = y2 + x2)
```

Implementation and Experiments

The Team

Research Scholars

Aira Jain

Pranita Ganguly

The Team

Research Scholars

Aira Jain

Pranita Ganguly

Implementation and Experiments

1. Prototype: Implemented using C++
2. SMT solver: Z3
3. Case studies: Student and Institute projects done by people outside the team
4. Test strings generated manually
5. Result: Successfully generated 100s of integration test cases
6. Fault injection
7. Our tests detect these faults while state-of-the-art REST testing tool can't.

1 Introduction

2 Test the REST (TTR)
- Testing RESTful APIs

3 Variations and Extensions
- Mocking
- Specification Generation

4 Wrap-Up

1 Introduction

2 Test the REST (TTR)

- Testing RESTful APIs

3 Variations and Extensions

- Mocking
- Specification Generation

4 Wrap-Up

Mock Generation using Contracts

Mocking – Problem

- f is the function under test (client side).
- f calls another function g (server side).
- g is a potentially side-effectful function that requires some prerequisite code to run before it becomes valid to call g.
- Disadvantages:
  1. Complex to prepare unit test for f (because it has to prepare the server for the test).
  2. Makes tests run slower.
  3. Sometimes, may not be feasible due to unavailability of server.

Mock Generation using Contracts

Motivation - Mocking

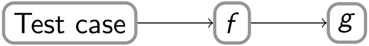
- Mocking allows us to simplify the problem of writing unit tests for f.
- Instead of using g, we use g', a mock of g.
- g' has the same function signature as g.
- g' is much simpler than g.
- g' returns a correct value that allows f's test to proceed even without our having to run the prerequisite code for g.

Mock Generation using Contracts

Motivation - Mocking

- Mocking allows us to simplify the problem of writing unit tests for f.
- Instead of using g, we use g', a mock of g.
- g' has the same function signature as g.
- g' is much simpler than g.
- g' returns a correct value that allows f's test to proceed even without our having to run the prerequisite code for g.
- g' runs on the client.

Testing without mocking



Test case

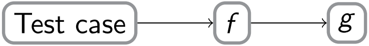
Prerequisite code for g

call to g

Mock Generation using Contracts

Motivation - Mocking

Testing without mocking



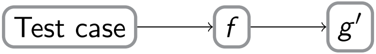
Test case → f → g

Test case

Prerequisite code for g

call to g

Testing with mocking



Test case → f → g'

Test case

No Prerequisite code for g

call to g'

Mock Generation using Contracts

Mocking - Challenges

1. Ensuring valid precondition
2. Ensuring valid postcondition

Mock Generation using Contracts

Mocking - Challenges

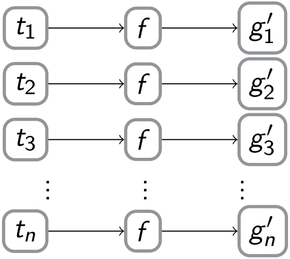
1. Ensuring valid precondition
2. Ensuring valid postcondition
3.

$$\begin{tikzcd}
t_1 \arrow[r] & f \arrow[r] & g'_1 \\
t_2 \arrow[r] & f \arrow[r] & g'_2 \\
t_3 \arrow[r] & f \arrow[r] & g'_3 \\
\vdots & \vdots & \vdots \\
t_n \arrow[r] & f \arrow[r] & g'_n
\end{tikzcd}$$

Mock Generation using Contracts

Mocking - Challenges

1. Ensuring valid precondition
2. Ensuring valid postcondition
3.



A DIFFERENT g' FOR EACH TEST CASE!

Mock Generation using Contracts

Mocking - Challenges

1. Ensuring valid precondition
2. Ensuring valid postcondition
3.

$$\begin{tikzcd}
\boxed{t_1} \arrow[r] & \boxed{f} \arrow[r] & \boxed{g'_1} \\
\boxed{t_2} \arrow[r] & \boxed{f} \arrow[r] & \boxed{g'_2} \\
\boxed{t_3} \arrow[r] & \boxed{f} \arrow[r] & \boxed{g'_3} \\
\vdots & \vdots & \vdots \\
\boxed{t_n} \arrow[r] & \boxed{f} \arrow[r] & \boxed{g'_n}
\end{tikzcd}$$

A DIFFERENT g' FOR EACH TEST CASE! – Automation necessary.

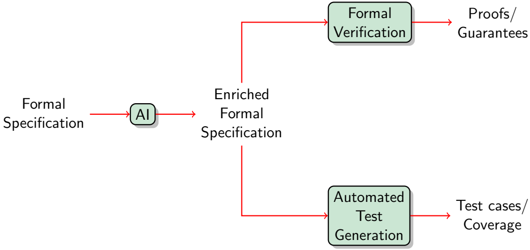
1 Introduction

2 Test the REST (TTR)
   - Testing RESTful APIs

3 Variations and Extensions
   - Mocking
   - Specification Generation

4 Wrap-Up

Specification Generation using LLMs



Formal Specification

AI

Enriched Formal Specification

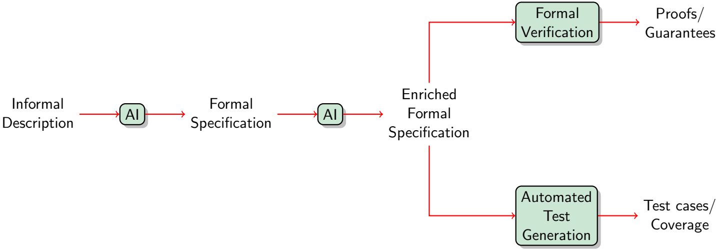
Formal Verification

Proofs/ Guarantees

Automated Test Generation

Test cases/ Coverage

Specification Generation using LLMs



1. Introduction

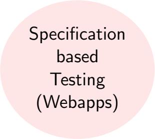
2. Test the REST (TTR)
   - Testing RESTful APIs

3. Variations and Extensions
   - Mocking
   - Specification Generation

4. Wrap-Up

The TestGen Algorithm

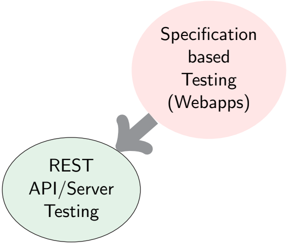
A Roadmap



Specification based Testing (Webapps)

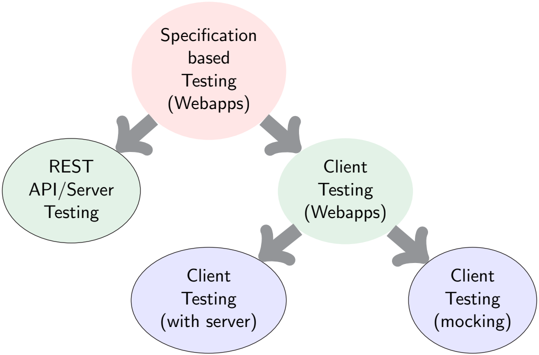
The TestGen Algorithm

A Roadmap



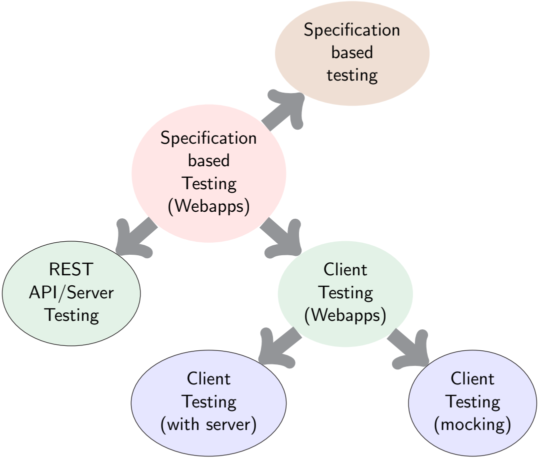
The TestGen Algorithm

A Roadmap



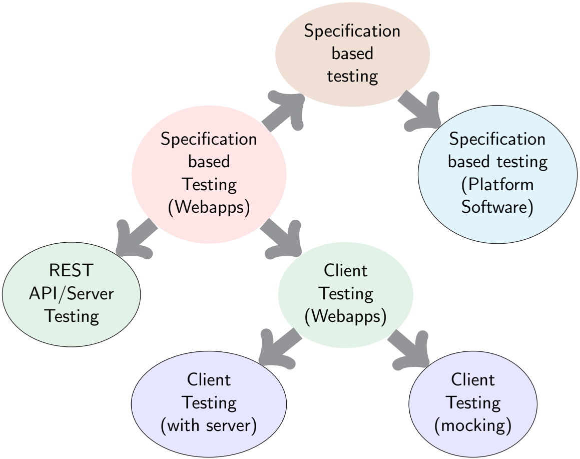
The TestGen Algorithm

A Roadmap


Specification based testing
Specification based Testing (Webapps)
REST API/Server Testing
Client Testing (Webapps)
Client Testing (with server)
Client Testing (mocking)

The TestGen Algorithm

A Roadmap



Specification based testing

Specification based Testing (Webapps)

Specification based testing (Platform Software)

REST API/Server Testing

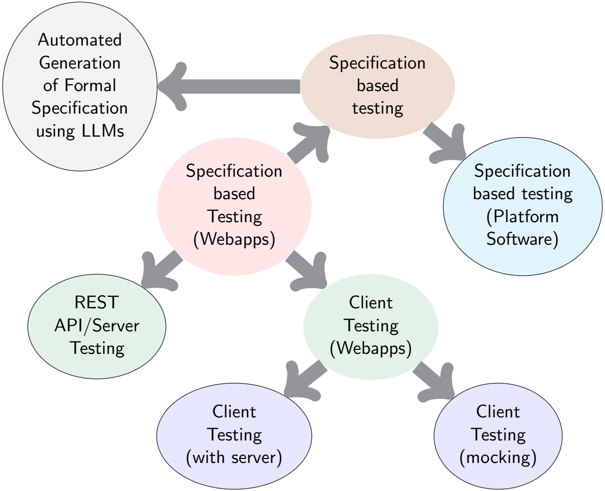
Client Testing (Webapps)

Client Testing (with server)

Client Testing (mocking)

The TestGen Algorithm

A Roadmap


Automated Generation of Formal Specification using LLMs
Specification based testing
Specification based Testing (Webapps)
Specification based testing (Platform Software)
REST API/Server Testing
Client Testing (Webapps)
Client Testing (with server)
Client Testing (mocking)

The Team

Project Students

Sai Kaushik

Srinivasan

Shishir Shahi

Vihan Vashisht

Vineet Priyedarshi

Acknowledgement

CTRI

Digital Governance

Center for Technology Research and Innovation

CTRI-DG

Royal Academy
of Engineering

**Royal Academy
of Engineering, UK**

Thank You!