The Test Case Reasoning Assistant

Dana P. Leonard, Jason O. Hallstrom, Murali Sitaraman
School of Computing
Clemson University

This work is supported in part through grants from the National Science Foundation (DUE-0633506, CNS-0745846, DMS-0701187, CCF-0811748).

Course Module

Target

Any course where interface contracts can be introduced

Objectives

- Reading and interpreting formal specifications
- Understanding abstract models of SW behavior
- Using components based on contract understanding

Approach

Teach students to read, interpret, and apply interface specifications using test point creation exercises

Pilot Studies

CpSc 215:
Software Development Foundations

CpSc 372:
Software Engineering Foundations

CLEMSON UNIVERSITY

CS 349:
Software Engineering

DENISON UNIVERSITY

Exercise #1

```
Function Mystery(preserves x : Integer) : Integer
requires
  x >= 0
ensures
  (result * result <= x) and
  ((result + 1) * (result + 1)) > x
```

Question

What does the mystery function compute?

Solution Strategy

Develop test cases that satisfy the specification

#x	result	check ?
0	0	(0*0 ≤ 0) ∧ (1*1 > 0)
1	1	(1*1 ≤ 1) ∧ (2*2 > 1)
4	3	**(3*3 ≤ 4) ∧ (4*4 > 4)**
18	4	(4*4 ≤ 18) ∧ (5*5 > 18)

Exercise #2

```
Function Mystery(preserves x : Integer) : Boolean
requires
  x > 2
ensures
  Mystery() =
     there exists k, m : Integer
     s.t. (k >= 2) and (m >= 2) and (x = k * m)
```

#x	result	check ?
4	TRUE	∃ k,m : (k ≥ 2) ∧ (m ≥ 2) ∧ (4 = k*m)
7	FALSE	∃ k,m : (k ≥ 2) ∧ (m ≥ 2) ∧ (7 = k*m)
13	TRUE	**∃ k,m : (k ≥ 2) ∧ (m ≥ 2) ∧ (13 = k*m)**
22	TRUE	∃ k,m : (k ≥ 2) ∧ (m ≥ 2) ∧ (22 = k*m)

Question

What does the mystery function compute?

Abstract Models

Later, students are taught to reason about component behavior using abstract mathematical models

Theory

```
Str(Integer) : mathematical string of integers
   examples : <>, <1>, <2, 1>, <3, 1, 2>, <4, 1, 2, 3>
   concatenation : <> o <1> = <1>, <1, 2> o <5, 6> = <1, 2, 5, 6>
   length : |<>| = 0, |<1, 3, 2>| = 3
```

Specification

```
Stack is modeled by Str(Entry)

Operation Push(updates s : Stack, preserves x : Entry)
ensures s = <x> o #s

Operation Pop(updates s : Stack, produces x : Entry)
ensures #s = <x> o s

Operation Length(preserves s : Stack)
ensures result = |s|
```

Exercise #3

```
Operation Mystery(updates s : Stack, preserves x : Integer)
requires
  x <= |self|
ensures
  there exists l, r : Str(Entry)
  s.t. (#s = l o r) and (|l| = x) and (s = r o l)
```

#s	#x	s	check ?
<>	0	<>	∃ l,r : (<> = l o r) ∧ (|l| = 0) ∧ (<> = r o l)
<1>	1	<1>	∃ l,r : (<1> = l o r) ∧ (|l| = 1) ∧ (<1> = r o l)
<1,2,3>	2	<2,3,1>	**∃ l,r : (<1,2,3> = l o r) ∧ (|l| = 2) ∧ (<2,3,1> = r o l)**
<1,2,3,4,5>	3	<4,5,1,2,3>	∃ l,r : (<1,2,3,4,5> = l o r) ∧ (|l| = 3) ∧ (<4,5,1,2,3> = r o l)

The Reasoning Workbench

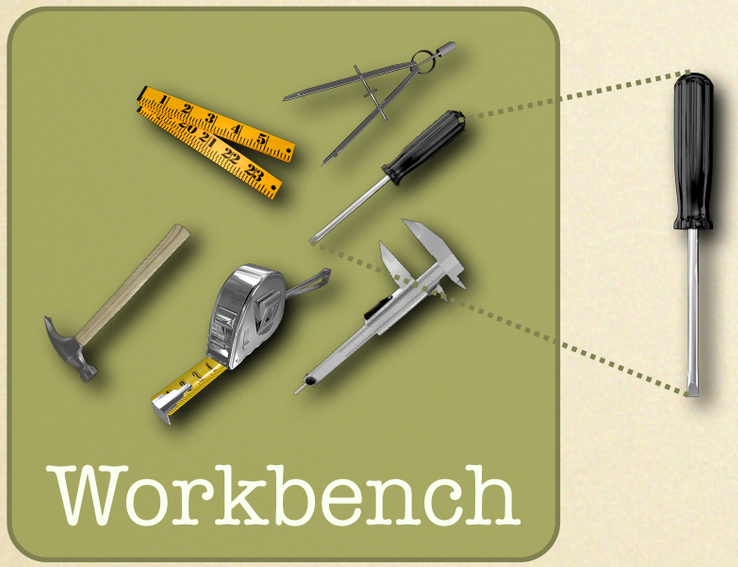
Smart classrooms enable new teaching tools and exercises that enhance the learning process:

- Excite and engage students
- Provide real-time performance feedback
- Monitor student learning and identify problem areas

The Reasoning Workbench

A collection of **reasoning assistants** that guide students through reasoning exercises and provide real-time feedback as they work

Test Case Reasoning Assistant



Test Case Reasoning Assistant (TCRA)

Purpose

Assist students in developing test cases that measure and reinforce their understanding of abstract models and interface specifications

TCRA Overview

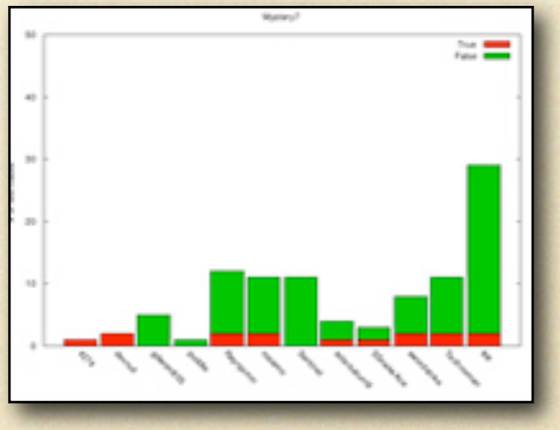
Student Interface

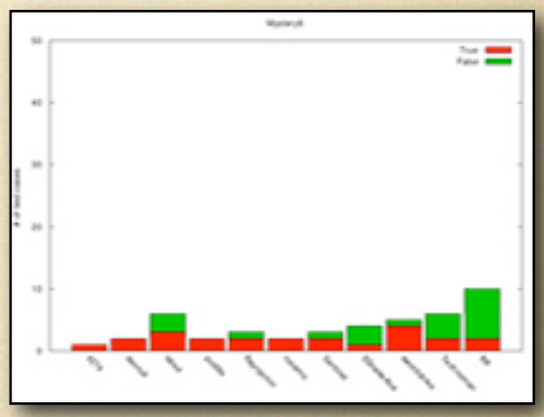
Exercise Repository

```
public class StackExercise
implements TCRAExercise {

  presentation | checking

}
```

performance data

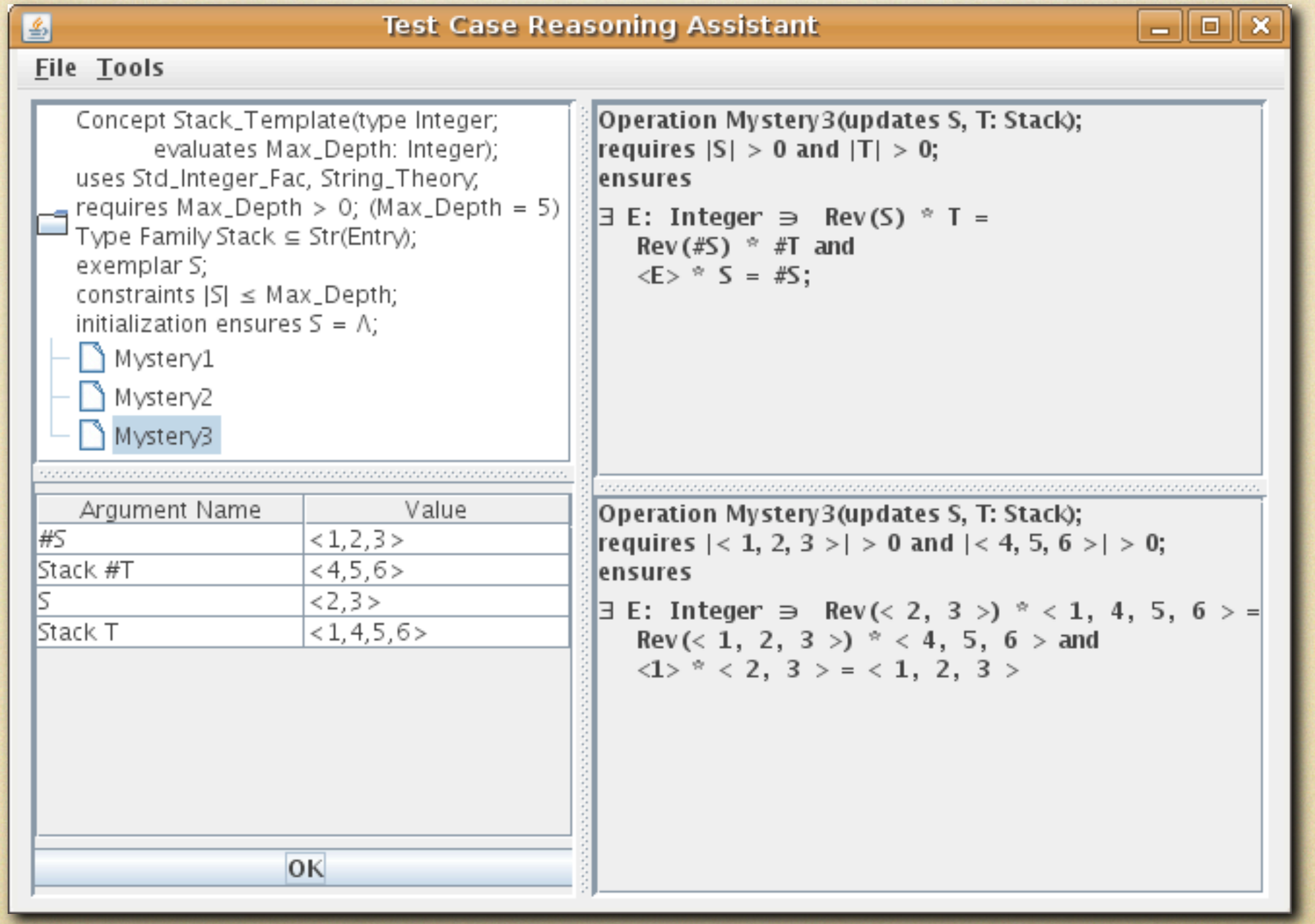
Instructor Interface



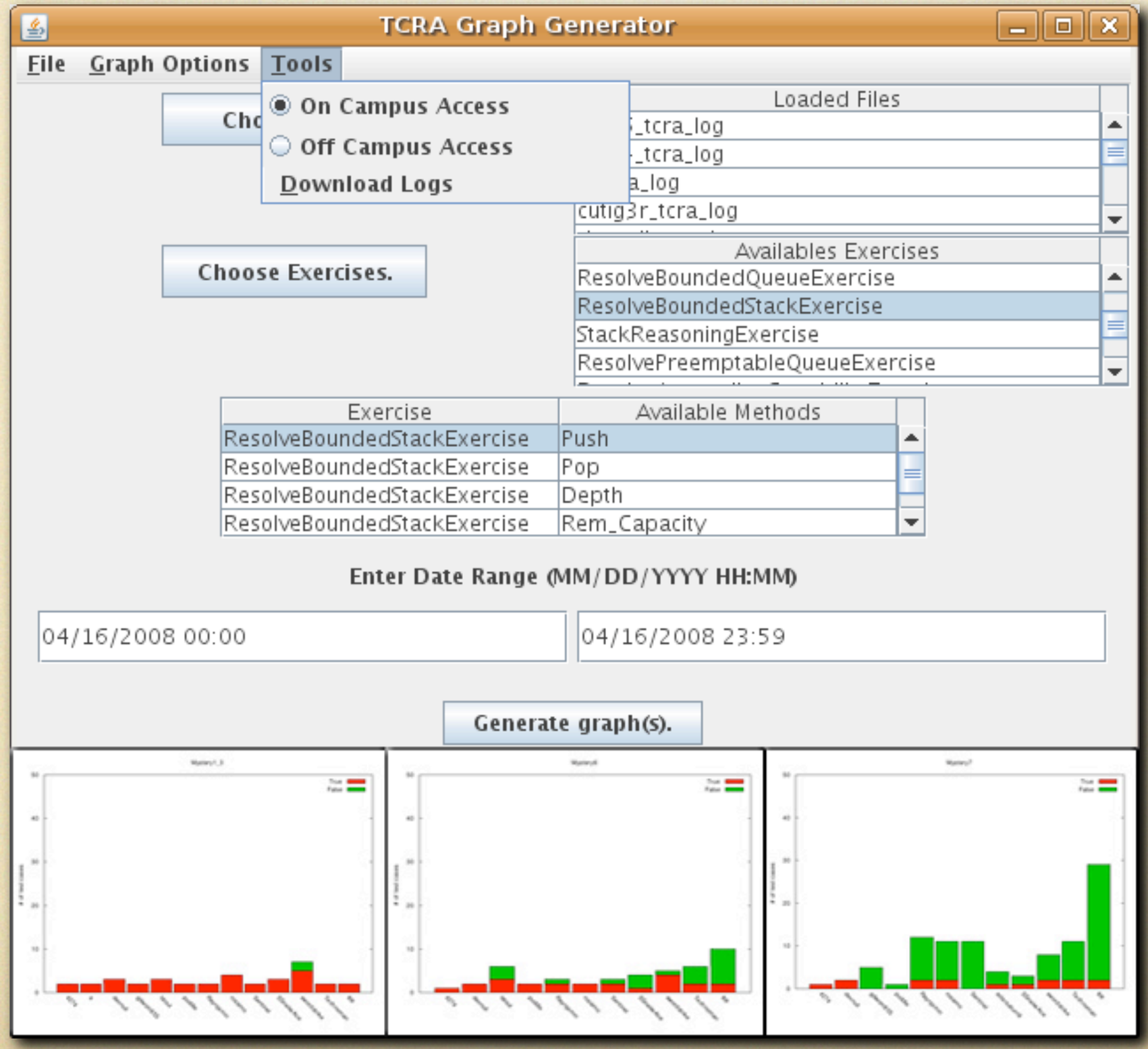


individual / aggregate performance summaries

TCRA Student Interface



TCRA Instructor Interface



Additional Modules

More advanced modules require students to rely on specifications as part of their development work

Contract Development in Teams

- Medium-scale project subdivided into components
- Students work independently using formal contracts
- Systems composed from selected implementations

Tool-Assisted Program Verification

- Students derive simple verification conditions (VCs)
- Derivation process is reinforced using VC generator
- Students prove generated VCs using proof assistant

Questions?

The Test Case Reasoning Assistant

Dana P. Leonard, Jason O. Hallstrom, Murali Sitaraman

School of Computing

Clemson University

This work is supported in part through grants from the National Science Foundation (DUE-0633506, CNS-0745846, DMS-0701187, CCF-0811748).