SYNTHESIS OF TEMPORAL LOGIC TESTERS FOR AUTO-GRADING A CYBER-PHYSICAL SYSTEMS LAB

Alex Donze\textsuperscript{1}, Garvit Juniwal\textsuperscript{1}, Jeff Jensen\textsuperscript{2}, Sanjit A. Seshia\textsuperscript{1}

\textsuperscript{1}UC Berkeley
\textsuperscript{2}National Instruments
Overview

• **Cyber-Physical System (CPS):**
  
  Software + physical world. E.g. controller code + robot in an arena.

• **Auto-grading = verification + debugging.**

• Complex environment interactions $\Rightarrow$ *run-time verification*

• Simulation trace $\Rightarrow$ monitor *signal temporal logic* properties $\Rightarrow$ localize *faulty* behavior

• Failure to meet *goal* $\equiv$ Fault
Laboratory Setting (EECS149)

- **Robot Navigation Goals**: avoid obstacles (maintaining heading direction) and climb hills.
- **Controller Inputs**: bump/cliff sensor, 3-axes accelerometer data
- **Controller Outputs**: left/right wheel speed
Controller

```cpp
/* state transition - run region */

// state actions */

switch (state) {
    case INITIAL:
    case PAUSE_WAIT_BUTTON_RELEASE:
    case UNPAUSE_WAIT_BUTTON_PRESS:
    case UNPAUSE_WAIT_BUTTON_RELEASE:
        /* in pause mode, robot should be stopped */
        leftWheelSpeed = rightWheelSpeed = 0;
        break;

    case DRIVE:
        /* full speed ahead! */
        leftWheelSpeed = rightWheelSpeed = maxWheelSpeed;
        break;

    case TURN:
        leftWheelSpeed = maxWheelSpeed;
        rightWheelSpeed = -leftWheelSpeed;
        break;

    default:
        /* Unknown state */
        leftWheelSpeed = rightWheelSpeed = 0;
        break;
}
```
Simulator
Fault Detection

- **Environment**: Arena composed of obstacles and hills
- **Monitor**: Signal Temporal Logic formula that captures presence of fault in a trace
- **Test**: Environment + Monitor

A test is "triggered" by a controller if the fault property holds on the simulation trace in the environment.

\[ + \quad G_{[0,60]} \text{pos.}z < 0.4 \]
Are we done?

- Grading should be robust to variations in environment and student solutions.
  - Obstacle placement; hill incline & height
  - Different wheel speeds; strategies.

- Introduce parameters in environment and STL formula.
- Creating *temporal logic test benches* = solving a *parameter synthesis* problem.
Parameterization

- Generate a collection of tests (parameter space)
- Only a subset of this collection is indicative of the fault. (interesting subspace)
- If at least one test from this subspace is triggered, we label the controller as faulty.

Test Bench

\[ G_{[0, \tau]} \text{pos.} z < h \]

\[ [\theta \mapsto (-90, 90), \tau \mapsto (60, +\infty),
\quad h \mapsto (0, 0.4)] \]
Auto-grading Flow

Controller

Simulator

Environment

Fault Monitor

Monitoring Engine

Parameters: incline/obstacle placements, initial position/orientation

Timed sequence of values like wheel speeds, position, orientation, etc.

Breach

Faults, Feedback

Parametric Signal Temporal Logic (PSTL). Parameters: timeSCALE

Ni Robotics Simulator

Test bench
Synthesis of Test Benches

• Finding this subspace manually is tedious.
• Coming up with *reference solutions* is relatively easy.

• Synthesize test bench subspace from reference controllers.
• Include every test which
  • is triggered on *at least one* reference solution with the fault, but
  • is NOT triggered by *any* reference solution without the fault.
$$G_{[0, \tau]} \text{pos.z} < h$$

Controller with fault

Controller without fault
Use of Monotonicity

Monotonicity of parameters utilized for:

- Efficient algorithm to compute subspace
- Notion of a minimal adequate test sample: enough to check for a triggered test within this sample.
For hill climb
Strict Inequality Check Fault

- Fault property: turn in-place
- Parameters:
  - $tau$ – period of turning
  - $delta$ – allowed margin of movement while turning
For strict inequality check
Evaluation

• Evaluated using student solutions from an on-campus offering of the course.
• Synthesized test benches for 2 goal properties:
  • Get around obstacle and maintain heading direction
  • Climb hills
• And 4 specific kinds of faults:
  • No filter
  • No hill detection
  • Strict equality check
  • Avoiding obstacle in wrong direction
• High accuracy for every fault. For the ‘strict inequality check’ fault, we compared a sample of 200 manually graded solutions (6 of which had this fault) against the auto-grader. The auto-grader had zero false labels.
References

• www.leeseshia.org/lab

• PLOOC 2013 workshop talk: Sanjit Seshia and Jeff Jensen. *Meeting the MOOC challenge for Embedded Systems*

• CPS-ed 2013: Jeff C. Jensen, Edward A. Lee, and Sanjit A. Seshia. *Virtualizing Cyber-Physical Systems: Bringing CPS to Online Education*