Automating Exercise Generation: A Step towards Meeting the MOOC Challenge for Embedded Systems

Dorsa Sadigh, Sanjit Seshia and Mona Gupta
UC Berkeley
WESE 2012
MOTIVATION

• Recent trend towards massive open online courses (MOOCs)
  • Allow learning at different speeds
  • Access to standard curriculum
  • Enrollment of large number of students

• Challenges of moving online from traditional classroom
  • Generating problems and Solutions
  • Grading & Feedback
  • Monitoring
AUTOMATIC EXERCISE GENERATION

• Generating New Problems
  • Given existing problems, we want to generate “Similar” problems
    • Similar concepts are tested
    • Do not intend to completely remove human insight

• Automatic Solution Generation
  • Automatically solve a mathematical formulation

• Auto-grading
  • Checking whether a candidate solution is valid
RELATED WORK

- Related work in context of automatic problem generation has been explored in different domains.
  - Automatic problem generation of *algebra proof problems*. (Singh et. al. 2012)
  - Instantiating parameters of algebra problems with random constants. (Jurkovic 2001)

- We focus our work on problem generation for embedded systems.
This course is intended to introduce the design and analysis of computational systems that interact with physical processes.

http://leeseshia.org/
OUTLINE

• Model-based problems
  • Traffic-Light example
  • Solving it
  • Generate a Template
  • Generate New Problems
    • Modify Model, Trace, Specification

• Real-time scheduling problems
  • RM Example
  • Solving it
  • Generate a Template
  • Generate New Problems
MODEL-BASED PROBLEMS
TRAFFIC LIGHT EXAMPLE

This problem considers the FSM that models arrivals of pedestrians at a crosswalk. We assume that the traffic light at the crosswalk is controlled by the FSM shown in the figure. In all cases, assume that a time triggered model, where both the pedestrian model and the traffic light model react once per second. Assume further that the composition of the two models is synchronous composition.

inputs: sigR, sigG, sigY : pure
outputs: pedestrian : pure
true /

variable: count: \{0, \ldots, 60\}
inputs: pedestrian : pure
outputs: sigR, sigG, sigY : pure

\[
\text{count} < 60 / \text{count} := \text{count} + 1
\]

\[
\text{count} \geq 60 / \text{sigG} \quad \text{count} := 0
\]

\[
\text{pedestrian} \land \text{count} < 60 / \text{count} := \text{count} + 1
\]

\[
\text{count} \geq 5 / \text{sigR} \quad \text{count} := 0
\]

\[
\text{pedestrian} \land \text{count} \geq 60 / \text{sigY} \quad \text{count} := 0
\]

\[
\text{count} \geq 60 / \text{sigY} \quad \text{count} := 0
\]

\[
\text{count} := \text{count} + 1
\]
TRAFFIC LIGHT EXAMPLE

• For the given pedestrian model, find a trace whereby a pedestrian arrives (the machine transitions to waiting) but the pedestrian is never allowed to cross. That is, at no time after the pedestrian arrives is the traffic light in state red.

Solution:
Start from (Red, None), make a transition to (Green, Waiting), and now a pedestrian has arrived and is not allowed to cross.
Red -> Green (which emits sigG), and None -> Waiting (which emits pedestrian).
SOLUTION TO TRAFFIC LIGHT EXAMPLE

variable: count: \{0, \ldots, 60\}
inputs: pedestrian : pure
outputs: sigR, sigG, sigY : pure

\[
\phi = G(\text{pedestrian} \rightarrow (F \text{ SigR}))
\]

Composed Model

Counterexample Trace

\langle \psi \rangle

(\text{Red, None} \rightarrow (\text{Green, Waiting}))

Specification
GENERATED TEMPLATE

- Given a model $<M>$
- System Model $<S>$
  - Time triggered/Event triggered
- Environment Model $<E>$
  - Time triggered/Event triggered

- Given a specification $\langle \phi \rangle$
  - LTL/English

- Find trace $\langle \psi \rangle$
  - Violates / reaches the specification
PROPERTY, TRACE, MODEL

\[
\langle S \rangle
\]

\[
\langle M \rangle
\]

\[
\langle E \rangle
\]

\[
\langle \phi \rangle
\]

\[
\langle \psi \rangle
\]

Time / Event Triggered

Synch/ Asynch

Formal Des./ Code/ Timed Automaton

Property English/ LTL

Trace Witness/ Counterexample

University of California, Berkeley
# How to Find the Solutions

Techniques to Find Solutions for Model Based Problems

<table>
<thead>
<tr>
<th>Given</th>
<th>Find</th>
<th>Solution Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\langle \phi \rangle$</td>
<td>$\langle M \rangle$</td>
<td>Constrained Synthesis or Repair</td>
</tr>
<tr>
<td>$\langle M \rangle$</td>
<td>$\langle \psi \rangle$</td>
<td>Simulation of Model</td>
</tr>
<tr>
<td>$\langle M \rangle$</td>
<td>$\langle \phi \rangle$</td>
<td>Specification Mining</td>
</tr>
<tr>
<td>$\langle M \rangle &amp; \langle \psi \rangle$</td>
<td>$\langle \psi \rangle$</td>
<td>Simulation with Guidance</td>
</tr>
<tr>
<td>$\langle M \rangle &amp; \langle \phi \rangle$</td>
<td>$\langle \psi \rangle$</td>
<td>Model Checking</td>
</tr>
</tbody>
</table>
## Classification

<table>
<thead>
<tr>
<th>Given</th>
<th>Find</th>
<th>Variations</th>
<th>Exercise #</th>
</tr>
</thead>
</table>
| $\langle \phi \rangle$ | $\langle M \rangle$ | (i) $\phi \in$ English or LTL  
(ii) use hybrid systems for $\phi$  
(iii) Modify previous system  | 3.1, 3.2, 9.5 |
| $\langle M \rangle$ | $\langle \psi \rangle$ | (i)  
(ii)  
(iii)  | 9.5 |

More than 60% of Questions from relevant chapters match this template and can be generated automatically.

Classification of Model-Based problems in Lee & Seshia
HOW TO GENERATE NEW PROBLEMS

• Modifying the model \( \langle M \rangle \)

| \( \langle M \rangle \) | \( \langle \psi \rangle \) | Simulation of Model |
| \( \langle M \rangle \) | \( \langle \phi \rangle \) | Specification Mining |
| \( \langle M \rangle \) & \( \langle \psi \rangle \) | \( \langle \psi \rangle \) | Simulation with Guidance |
| \( \langle M \rangle \) & \( \langle \phi \rangle \) | \( \langle \psi \rangle \) | Model Checking |

• Modifying the trace \( \langle \psi \rangle \)

| \( \langle M \rangle \) & \( \langle \psi \rangle \) | \( \langle \psi \rangle \) | Simulation with Guidance |

• Modifying the specification \( \langle \phi \rangle \)

| \( \langle \phi \rangle \) | \( \langle M \rangle \) | Constrained Synthesis or Repair |
| \( \langle M \rangle \) & \( \langle \phi \rangle \) | \( \langle \psi \rangle \) | Model Checking |
MODIFYING MODEL <M>

- Mutation Operators:
  - Change the initial state.
  - Change the target state of a transition.
  - Create a new transition.
  - Increment or decrement the number of states by one.

- Create 32 new receptive models by combining the first three operators up to 5 mutations.
- About 10 of these models are plausible scenarios.
AN EXAMPLE OF A MUTANT

inputs: sigR, sigG, sigY : pure
outputs: pedestrian : pure

true /

true / pedestrian

Initial state is changed and self loop is added to waiting

Indicates the pedestrian can continue waiting even when the light turns green.
## MUTANTS FOR PEDESTRIAN MODEL

<table>
<thead>
<tr>
<th>Num. of Mutations</th>
<th>Num. of Models Created</th>
<th>Types of Mutations (number)</th>
<th>Num. of Realistic Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>Adding a new transition (4)</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>Combination of adding new transitions and changing initial state (4) Changing end points of two transitions (4)</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>New transition, changing initial state, changing end point (4) New transition, changing end points of two transitions (4)</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>New transition, changing initial state, changing end points of two transitions (4) New transition, changing end points of three transitions (4)</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>New transition, changing initial state, changing end points of three transitions (4)</td>
<td>0</td>
</tr>
<tr>
<td>6+</td>
<td>0</td>
<td>None</td>
<td>0</td>
</tr>
</tbody>
</table>
Automobiles today have the features below. Implement the features as an extended state machine.

• The dome light is turned on as soon as any doors is opened. It stays on for 30 seconds after all doors are shut.

We can write this as an LTL specification.

\[ G[(D_1 \text{Open} \lor D_2 \text{Open} \lor D_3 \text{Open} \lor D_4 \text{Open}) \rightarrow (\text{light}_{on} \land X\text{light}_{on} \land \ldots \land X^{30}\text{light}_{on})] \]

Then we can generalize:

\[ G[\phi \rightarrow (\bigwedge_{i=0}^{k} X^{k}\psi)] \]
MODIFYING
SPECIFICATION \( \langle \phi \rangle \)

\[
G[\phi \rightarrow (\bigwedge_{i=0}^{k} X^k \psi)] \quad \phi = (\text{Locked} \land \text{DoorOpenAttempt})
\]

\[
k = 600
\]

\[
\psi = \text{alarm}_{\text{on}}
\]

Take a new instance

This indicates the property that if the car is locked when someone attempts to open the door, the car alarm goes off and continues for 5 minutes.

\[
G[(\text{Locked} \land \text{DoorOpenAttempt}) \rightarrow (\bigwedge_{i=0}^{600} X^{600} \text{alarm}_{\text{on}})]
\]
Real-Time Scheduling Problems
This problem studies fixed-priority scheduling. Consider two tasks to be executed periodically on a single processor, where task1 has period $p_1=4$ and task2 has period $p_2=6$.

Let the execution time of task1 be $e_1=1$. Find the maximum value for the execution time $e_2$ of task2 such that the RM schedule is feasible.
SOLUTION TO SCHEDULING PROBLEM

- How do we solve it?

\[
[p_1 < p_2] \rightarrow \left\lfloor \frac{p_2}{p_1} \right\rfloor \ast e_1 + e_2 = p_2
\]

\[
[4 < 6] \rightarrow \left\lfloor \frac{6}{4} \right\rfloor \ast 1 + e_2 = 6 \implies e_2 = 4
\]
REAL-TIME SCHEDULING TEMPLATE

• Given $<n>$ tasks
  • Periodic/sporadic
  • Fixed/dynamic priority
  • Given their execution times, periods, deadlines
    \[ \langle e_1, \ldots, n \rangle \quad \langle p_1, \ldots, n \rangle \quad \langle d_1, \ldots, n \rangle \]

• Given $<S>$ a scheduling protocol
  • Rate Monotonic/Earliest Deadline First

• Find unknown variable
  • execution time/period
GENERALIZE THE EQUATION

- Formulate the problem based on the scheduling protocol
- Use an SMT solver to find the solution

Formulate the solution:

\[
[p_1 < p_2] \rightarrow \left[ \frac{p_2}{p_1} \right] \cdot e_1 + e_2 = p_2
\]

Generalize it for N tasks:

\[
[p_1 < \cdots < p_n] \rightarrow \left[ \frac{p_n}{p_1} \right] \cdot e_1 + \cdots + \left[ \frac{p_n}{p_{n-1}} \right] \cdot e_{n-1} + e_n = p_n
\]

University of California, Berkeley
PROBLEM GENERATION

- Given constraints on periods and execution times, generate random numbers for the variables of the template.
- Constraints for Rate Monotonic Schedule:

\[ p_1 < p_2, \quad e_1 < p_1, \quad p_1 \neq p_2 \]

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>11</td>
<td>5</td>
<td>19</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>19</td>
<td>20</td>
<td>6</td>
<td>7</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>14</td>
<td>6</td>
<td>15</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>19</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>17</td>
<td>4</td>
<td>6</td>
<td>17</td>
<td></td>
</tr>
</tbody>
</table>
Real-Time scheduling Problem Solver/Generator

Consider n tasks that are executed periodically on a single processor.

Choose the number of tasks, and the desired scheduling protocol to generate values for execution times and periods.

You can also provide the periods of the tasks p_1,...,p_n and execution times a_1,...,a_n and ask for the execution times.

Model Mutation through Ptolemy

Original Pedestrian Light Model

Mutation Parameters

<table>
<thead>
<tr>
<th>Number of Mutations</th>
<th>Generator</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Outputted Models: 8

Solution to RM Schedule

verifun.eecs.berkeley.edu/autogen/PtolemyDemo/modelMutation.php

verifun.eecs.berkeley.edu/autogen/SchedulingDemo/schedule.php
FUTURE WORK

• Use this technique for other groups of problems.
  • Code involving interrupts/threads
• Immediate customized feedback to students
  • Error localization
• Interactive exercise creation toolkit for instructors