Model-based Synthesis of Reactive Planning Online Testers for Non-deterministic Embedded Systems

Jüri Vain
Dept. of Computer Science
Tallinn University of Technology

J.Vain "...Reactive Planning Testing..."
Lyngby, June 14, 2010
Outline

- Preliminaries
  - Model-Based Testing
  - Online testing
- Reactive Planning Tester (RPT)
- Constructing the RPT
- Performance of the approach
- Demo
Context: Model-Based Testing

- **Given**
  - a specification model and
  - an Implementation Under Test (IUT),
  - Test goal

- **Find**
  - If the IUT conforms to the specification in terms expressed in test goal.
Model-Based Testing

- The specification and test goal need to be formalised.
- We assume specs are given as
  - Extended Finite State Machines
  - UPTA
  - LSC
  - etc.

J.Vain "...Reactive Planning Testing..."
Lyngby, June 14, 2010
Online testing

- Denotes test generation and execution algorithms that
  - *compute successive stimuli at runtime* directed by
    - the test purpose and
    - the observed outputs of the IUT

J.Vain "...Reactive Planning Testing..."
Lyngby, June 14, 2010
Online testing

- Advantages:
  - The *state-space explosion* problem is reduced because only a limited part of the state-space needs to be kept track of at any point in time.

- Drawbacks:
  - *Exhaustive planning is difficult* due to the limitations of the available computational resources at the time of test execution.

J.Vain "...Reactive Planning Testing..."  
Lyngby, June 14, 2010
Online testing:
Spectrum of planning methods

- Random walk (RW): select test stimuli in random
  - inefficient - based on random exploration of the state space
  - leads to test cases that are unreasonably long
  - may leave the test purpose unachieved

- RW with reinforcement learning (anti-ant)
  - the exploration is guided by some reward function

- Exploration with exhaustive planning
  - MC provides *possibly an optimal* witness trace
  - the *size of the model is critical* in explicit state MC
  - state explosion in "combination lock" or deep loop models
Online testing:
Spectrum of planning methods

- Random walk (RW): select test stimuli in random
  - inefficient - based on random exploration of the state space
  - leads to test cases that are unreasonably long
  - may leave the test purpose unachieved

- RW with reinforcement learning (anti-ant)
  - the exploration is guided by some reward function

- Planning with limited horizon!

- Exploration with exhaustive planning
  - MC provides possibly an optimal witness trace
  - the size of the model is critical in explicit state MC
  - state explosion in "combination lock" or deep loop models
Reactive Planning

- Instead of a complete plan, only a set of *decision rules* is derived.
- The rules direct the system when executed towards the planning goal.
- Based on current situation evaluation just one subsequent input is computed at a time.
- Planning horizon can be adjustable.
Reactive Planning
[Brian C. Williams and P. Pandurang Nayak, 96 and 97]

- A Reactive Planning works in 3 phases:
  - Mode identification (MI)
  - Mode reconfiguration (MR)
  - Model-based reactive planning (MRP)
- MI and MR set up the planning problem identifying initial and target states
- MRP generates a plan
Reactive Planning Tester (RPT)

- MI – Where are we?
  Observe the output of the IUT to determine the current mode (state of the model)

- MR – Where do we want to go?
  Determined by still unsatisfied test (sub)goals

- MRP – How do we get there?
  Gain function guides the exploration of the model (choose the transition with the shortest path to the next subgoal)
RPT: Key Assumptions

- Testing is guided by the (EFSM) model of the tester and the test goal.
- Stimulae to the IUT are tester outputs generated by model execution.
- Responses from the IUT are inputs to the tester model.
- Decision rules of reactive planning are encoded in the guards of the transitions of the tester model.
- The rules are constructed by offline analysis based on the given IUT model and the test purpose.

J.Vain “...Reactive Planning Testing…”
Aalborg, April 8, 2010
RPT: The Model

- The IUT model is presented as an output observable nondeterministic EFSM in which all paths are feasible.

- Algorithm of making EFSM feasible:
Example: Nondeterministic FSM

$i_0$ and $i_3$ are output observable nondeterministic inputs
Encoding the Test Purpose in IUT Model

- Trap - a boolean variable assignment attached to the transitions of the IUT model
- A trap variable is initially set to false.
- The trap update functions are executed (set to true) when the transition is visited.
Add Test Purpose

bool t₀ = false;
...
bool t₇ = false;
Model of the tester

- Generated from the IUT model decorated with test purpose
- Transition guards encode the rules of online planning
- 2 types of tester states:
  - active – tester controls the next move
  - passive – IUT controls the next move
- 2 types of transitions:
  - Observable – source state is a passive state (guard ≡ true),
  - Controllable – source state is an active state (guard ≡ p_S /\ p_T where p_S – guard of the IUT transition; p_T – gain guard)

The gain guard (defined on trap variables) must ensure that only the outgoing edges with maximum gain are enabled in the given state.
Construction of the Tester Sceleton

States: 
- active
- passive

Transitions: 
- observable
- controllable
Add IO and Gain Guards
Constructing the gain guards (GG): intuition

- GG must guarantee that
  - each transition enabled by GG is a prefix of some locally optimal (w.r.t. test purpose) path;
  - tester should terminate after the test goal is reached or all unvisited traps are unreachable from the current state;
  - to have a quantitative measure of the gain of executing any transition $e$ we define a gain function $g_e$ that returns a distance weighted sum of unsatisfied traps that are reachable along $e$. 
Recall lessons from nature:
Ants’ Collective Hunting Strategies

Pheromone Guided Hunting:
• Maximizing prey localization
• Minimizing prey catching effort

Path selection criteria:
strength of pheromone trail - the analog to gain function
Constructing Gain Guards: intuition

Planning cones to be covered for decision making

Decision point

Alternative choices
Constructing Gain Guards: intuition

Gain functions

Decision point

Alternative choices

$g_e(tr_i, T)$

$g_e(tr_j, T)$

$g_e(tr_k, T)$
Constructing the gain guards: the gain function

- $g_e = 0$, if it is useless to fire the transition $e$ from the current state with the current variable bindings;
- $g_e > 0$, if firing the transition $e$ from the current state with the current variable bindings visits or leads closer to at least one unvisited trap;
- $g_{e_i} > g_{e_j}$ for transitions $e_i$ and $e_j$ with the same source state, if taking the transition $e_i$ leads to unvisited traps with smaller distance than taking the transition $e_j$;
- Having gain function $g_e$ with given properties define GG:

$$p_T \equiv (g_e = \max_k g_{e_k}) \land g_e > 0$$
Constructing the Gain Functions:

*shortest path trees*

- Reachability problem of trap labelled transitions can be reduced to *single-source shortest path problem*.
- Arguments of the gain function $g_e$ are
  - Shortest path tree $TR_e$ with root node $e$
  - $V_T$ – vector of trap variables
- To construct $TR_e$ we create a dual graph $G = (V_D, E_D)$ of the tester control graph $M_T$ where
  - the vertices $V_D$ of $G$ correspond to the transitions of the $M_T$,
  - the edges $E_D$ of $G$ represent the pairs of consecutive transitions sharing a state in $M_T$ (2-switches)
Constructing the Gain Guards: 

*shortest path tree (example)*

The dual graph of the tester model

The shortest-paths tree (left) and the reduced shortest-paths tree (right) from the transition $e^c_{01}$
Constructing the gain guards: gain function (1)

- Represent the reduced tree \( TR(e_i, G) \) as a set of elementary sub-trees each specified by the production \( \nu_i \leftarrow \bigcup_{j \in \{1,..n\}} \nu_j \)

- Rewrite the right-hand sides of the productions as arithmetic terms:

\[
\nu_i \rightarrow (-t_i)^\uparrow \cdot \frac{c}{d(\nu_0, \nu_i) + 1} + \max_{j=1,k}(\nu_j),
\]

- \( t_i^\uparrow \) - trap variable \( t_i \) lifted to type \( \mathbb{N} \),
- \( c \) - constant for rescaling the numerical value of the gain function,
- \( d(\nu_0, \nu_i) \) the distance between vertices \( \nu_0 \) and \( \nu_i \) where

\[
d(\nu_0, \nu_i) = l + \sum_{j=1}^{l} w_j
\]

- \( l \) - the number of hyper-edges on the path between \( \nu_0 \) and \( \nu_i \)
- \( w_j \) - weight of \( j \)-th hyperedge
Constructing the gain guards: 

*gain function* (2)

- For each symbol $\nu_i$ denoting a leaf vertex in $TR(e,G)$ define a production rule

  $$
  \nu_i \rightarrow (-t_i)^\uparrow \cdot \frac{c}{d(\nu_0, \nu_i) + 1}
  $$

- Apply the production rules (3) and (4) starting from the root symbol $\nu_0$ of $TR(e,G)$ until all non-terminal symbols $\nu_i$ are substituted with the terms that include only terminal symbols $t^\uparrow_i$ and $d(\nu_0, \nu_i)$
Example: Gain Functions

<table>
<thead>
<tr>
<th>Transition</th>
<th>Gain function for the transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_{01}^c$</td>
<td>$g_{e_{01}}(T) = c \cdot \max(-t_0/2,$ $-t_1/2 + \max(-t_2/4, -t_3/4, -t_4/4 + \max(-t_5/6, -t_6/6, -t_7/6)))$</td>
</tr>
<tr>
<td>$e_2^c$</td>
<td>$g_{e_2}(T) = c \cdot (-t_2/2 + \max(-t_0/4,$ $-t_1/4 + \max(-t_3/6, -t_4/6 + \max(-t_5/8, -t_6/8, -t_7/8))))$</td>
</tr>
<tr>
<td>$e_{34}^c$</td>
<td>$g_{e_{34}}(T) = c \cdot \max(-t_3/2 + -t_2/4 + \max(-t_0/6, -t_1/6),$ $-t_4/2 + \max(-t_5/4, -t_6/4, -t_7/4))$</td>
</tr>
</tbody>
</table>
### Example: Gain Guards

<table>
<thead>
<tr>
<th>Transition</th>
<th>Gain guard formula for the transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_{01}^c$</td>
<td>$p_{01}^c(T) \equiv$</td>
</tr>
<tr>
<td></td>
<td>$g_{e_{01}^c}(T) = \max(g_{e_{01}^c}(T'))$</td>
</tr>
<tr>
<td></td>
<td>$\land g_{e_{01}^c}(T') &gt; 0$</td>
</tr>
<tr>
<td>$e_{2}^c$</td>
<td>$p_{2}^c(T) \equiv$</td>
</tr>
<tr>
<td></td>
<td>$g_{e_{2}^c}(T) = \max(g_{e_{2}^c}(T'), g_{e_{34}^c}(T'))$</td>
</tr>
<tr>
<td></td>
<td>$\land g_{e_{2}^c}(T') &gt; 0$</td>
</tr>
<tr>
<td>$e_{34}^c$</td>
<td>$p_{34}^c(T) \equiv$</td>
</tr>
<tr>
<td></td>
<td>$g_{e_{34}^c}(T) = \max(g_{e_{2}^c}(T'), g_{e_{34}^c}(T'))$</td>
</tr>
<tr>
<td></td>
<td>$\land g_{e_{34}^c}(T') &gt; 0$</td>
</tr>
</tbody>
</table>
The complexity of the synthesis of the reactive planning tester is determined by the complexity of constructing the gain functions.

For each gain function the cost of finding the $TR_e$ by breadth-first-search is $O(|V_D| + |E_D|)$ [Cormen], where
- $|V_D| = |E_T|$ - number of transitions of $M_T$
- $|E_D|$ - number of transition pairs of $M_T$ (is bounded by $|E_S|^2$)

For all controllable transitions of the $M_T$ the upper bound of the complexity of the computations of the gain functions is $O(|E_S|^3)$.

At runtime each choice by the tester takes $O(|E_S|^2)$ arithmetic operations to evaluate the gain functions.
Experimental results:
Test Goal: All Transitions

<table>
<thead>
<tr>
<th>Algorithm of the tester</th>
<th>Model 1 (8 trans.)</th>
<th>Model 2 (16 trans.)</th>
<th>Model 3 (32 trans.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random choice</td>
<td>56 ± 36</td>
<td>295 ± 130</td>
<td>1597 ± 1000</td>
</tr>
<tr>
<td>Anti-ant</td>
<td>21 ± 4</td>
<td>53 ± 13</td>
<td>218 ± 81</td>
</tr>
<tr>
<td>Reactive planner</td>
<td>17 ± 3</td>
<td>37 ± 6</td>
<td>80 ± 10</td>
</tr>
</tbody>
</table>
**Experimental Results:**

**Test Goal: Selected Transition**

<table>
<thead>
<tr>
<th>Algorithm of the tester</th>
<th>Model 1 (8 trans.)</th>
<th>Model 2 (16 trans.)</th>
<th>Model 3 (32 trans.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random choice</td>
<td>34 ± 35</td>
<td>120 ± 114</td>
<td>699 ± 719</td>
</tr>
<tr>
<td>Anti-ant</td>
<td>14 ± 7</td>
<td>36 ± 19</td>
<td>140 ± 70</td>
</tr>
<tr>
<td>Reactive planner</td>
<td>5 ± 2</td>
<td>8 ± 3</td>
<td>11 ± 3</td>
</tr>
</tbody>
</table>
Demo: "combination lock"

- Comparison of methods
  - Random search
  - Anti-ant
  - Reactive planning tester
Case study: Feeder Box Control Unit (FBCU) of the street lighting subsystem
Test environment of the FBCU

- MOTES Test Generator
- Poseidon for UML CASE tool
- TTCN-3 test code
- System Adapter
- FBCU

Diagram showing the flow of information from MOTES to TTCN-3 test code, then to MessageMagic TTCN-3 Test Tool, and finally to the FBCU.
Shaping RPT planning cones (i)

Gain functions

$g_e(tr_i, T)$  $g_e(tr_j, T)$  $g_e(tr_k, T)$

Decision point

Alternative choices

In online-testing decision time strictly bounded!
Shaping RPT planning cones (ii)

Gain functions

Decision point

Alternative choices

Decision time strictly bounded!

Prune the cone!

Horizon

(Different ways to define $h$, and visibility)

Doctoral course ‘Advanced topics in Embedded Systems’. Lyngby’09
### Average lengths of test sequences in the experiments

<table>
<thead>
<tr>
<th>Horizon</th>
<th>All transitions test coverage</th>
<th>Single transition test coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Planning with horizon</td>
<td>Planning with horizon</td>
</tr>
<tr>
<td></td>
<td>anti-anti</td>
<td>anti-ant</td>
</tr>
<tr>
<td></td>
<td>random choice</td>
<td>random choice</td>
</tr>
<tr>
<td>0</td>
<td>18345 ± 5311</td>
<td>44595 ± 19550</td>
</tr>
<tr>
<td></td>
<td>2199 ± 991</td>
<td>4928 ± 4455</td>
</tr>
<tr>
<td>1</td>
<td>18417 ± 4003</td>
<td>19725 ± 7017</td>
</tr>
<tr>
<td></td>
<td>2156 ± 1154</td>
<td>6656 ± 5447</td>
</tr>
<tr>
<td>2</td>
<td>5120 ± 1678</td>
<td>4935 ± 1875</td>
</tr>
<tr>
<td></td>
<td>1276 ± 531</td>
<td>2516 ± 2263</td>
</tr>
<tr>
<td>3</td>
<td>4187 ± 978</td>
<td>3610 ± 2538</td>
</tr>
<tr>
<td></td>
<td>746 ± 503</td>
<td>1632 ± 1745</td>
</tr>
<tr>
<td>4</td>
<td>2504 ± 815</td>
<td>2077 ± 552</td>
</tr>
<tr>
<td></td>
<td>821 ± 421</td>
<td>1617 ± 1442</td>
</tr>
<tr>
<td>5</td>
<td>2261 ± 612</td>
<td>1276 ± 426</td>
</tr>
<tr>
<td></td>
<td>319 ± 233</td>
<td>618 ± 512</td>
</tr>
<tr>
<td>6</td>
<td>2288 ± 491</td>
<td>1172 ± 387</td>
</tr>
<tr>
<td></td>
<td>182 ± 116</td>
<td>272 ± 188</td>
</tr>
<tr>
<td>7</td>
<td>1374 ± 346</td>
<td>762 ± 177</td>
</tr>
<tr>
<td></td>
<td>139 ± 74</td>
<td>147 ± 125</td>
</tr>
<tr>
<td>8</td>
<td>851 ± 304</td>
<td>548 ± 165</td>
</tr>
<tr>
<td></td>
<td>112 ± 75</td>
<td>171 ± 114</td>
</tr>
<tr>
<td>9</td>
<td>701 ± 240</td>
<td>395 ± 86</td>
</tr>
<tr>
<td></td>
<td>72 ± 25</td>
<td>119 ± 129</td>
</tr>
<tr>
<td>10</td>
<td>406 ± 102</td>
<td>329 ± 57</td>
</tr>
<tr>
<td></td>
<td>73 ± 29</td>
<td>146 ± 194</td>
</tr>
<tr>
<td>11</td>
<td>337 ± 72</td>
<td>311 ± 58</td>
</tr>
<tr>
<td></td>
<td>79 ± 30</td>
<td>86 ± 59</td>
</tr>
<tr>
<td>12</td>
<td>323 ± 61</td>
<td>284 ± 38</td>
</tr>
<tr>
<td></td>
<td>41 ± 15</td>
<td>74 ± 51</td>
</tr>
<tr>
<td>13</td>
<td>326 ± 64</td>
<td>298 ± 44</td>
</tr>
<tr>
<td></td>
<td>34 ± 8</td>
<td>48 ± 31</td>
</tr>
<tr>
<td>14</td>
<td>335 ± 64</td>
<td>295 ± 40</td>
</tr>
<tr>
<td></td>
<td>34 ± 9</td>
<td>40 ± 23</td>
</tr>
<tr>
<td>15</td>
<td>324 ± 59</td>
<td>295 ± 42</td>
</tr>
<tr>
<td></td>
<td>25 ± 4</td>
<td>26 ± 5</td>
</tr>
<tr>
<td>16</td>
<td>332 ± 51</td>
<td>291 ± 52</td>
</tr>
<tr>
<td></td>
<td>23 ± 2</td>
<td>24 ± 3</td>
</tr>
<tr>
<td>17</td>
<td>324 ± 59</td>
<td>284 ± 32</td>
</tr>
<tr>
<td></td>
<td>22 ± 2</td>
<td>21 ± 1</td>
</tr>
<tr>
<td>18</td>
<td>326 ± 66</td>
<td>307 ± 47</td>
</tr>
<tr>
<td></td>
<td>21 ± 1</td>
<td>21 ± 1</td>
</tr>
<tr>
<td>19</td>
<td>319 ± 55</td>
<td>287 ± 29</td>
</tr>
<tr>
<td></td>
<td>21 ± 1</td>
<td>21 ± 1</td>
</tr>
<tr>
<td>20</td>
<td>319 ± 68</td>
<td>305 ± 43</td>
</tr>
<tr>
<td></td>
<td>21 ± 1</td>
<td>21 ± 1</td>
</tr>
</tbody>
</table>

Doctoral course ‘Advanced topics in Embedded Systems’. Lyngby'09
Average test sequence lengths of the test sequences

- **Test goal: all transitions**

- **Test goal: single transition**
Average time spent for online planning of the next step

![Graph showing the average time spent for online planning of the next step. The x-axis represents the horizon, and the y-axis represents the msec. Two lines are present: one for all transitions and another for single transition.](image)
How to derive data constraints?

- For all transitions \( t(s_i,.) \) of state \( s_i \) generate reduced reachability tree \( RRT_i \) s.t.
  - transition \( t(s_i,.) \) is a root and the trap labeled transitions the terminal nodes of the \( RRT_i \).
- Compute data constraint for each path \( \pi_j \) of \( RRT_i \)
  - use \( wp \)-algorithm (starting from trap node) for pairs of neighbour traps of \( \pi_j \)
  - unfold loops using \( gfp \) for termination
  - for constructing the gain function of \( \pi_j \) record (when traversing \( \pi_j \)):
    - traps remaining on the path \( \pi_j \) and
    - The lengths of inter-trap paths
  - construct the gain function for full path \( \pi_j \) using trap-to-trap distances on that path and the vector of trap variables.
  - Global data constraint for the path is a conjunction of data constraints pairwise traps of \( \pi_j \).
Online computation of data constraints

1: for paths $\pi_j \in \Pi(s_i)$ departing from $s_i$
   evaluate the gain vector $\Gamma$

2: IF $\exists \pi_j \in \Pi(s_i). unchecked(\pi_j)$ THEN choose
   the path with highest gain ELSE STOP

3: Solve the data constraint $C(\pi_j)$ for $\pi_j$

4: If $|C(\pi_j)| = \emptyset$ THEN
   $unchecked(\pi_j)$=true; GOTO 2
   ELSE
   execute $t_i, t^1_i \in \pi_j$
Summary

- RP always drives the execution towards still unsatisfied subgoals.

- Efficiency of planning:
  - Number of rules that have to be evaluated at each step is relatively small (i.e., = the number of outgoing transitions of current state)
  - The execution of decision rules is significantly faster than looking through all potential alternatives at runtime.
  - Provides test sequences that are lengthwise close to optimal.
Thank You!

Questions?