Improving Constrained Random Testing by Achieving Simulation Verification Goals through Objective Functions, Rewinding and Dynamic Seed Manipulation

Eldon Nelson M.S. P.E. (eldon_nelson@ieee.org) Intel Corporation
Outline

• Inspiration from Nintendo Playing AI Paper
  – Objective Function
• Application to Simulation
  – Seeds and Dynamic Re-Seeding
  – Objective Function in Verification
  – Checkpointing Feedback Loop
• Scaling
• Results
Full Code Implementation

https://github.com/tenthousandfailures/improving-constrained-random

Implementation of a proposed method to improve constrained random simulation http://tenthousandfailures.com
Tom Murphy VII

Computer program that learns to play classic NES games
The First Level of Super Mario Bros. is Easy with Lexicographic Orderings and Time Travel ... after that it gets a little tricky.

Dr. Tom Murphy VII Ph.D.*

1 April 2013

Abstract

This paper presents a simple, generic method for automating the play of Nintendo Entertainment System games.

Keywords: computational super mario brothers, memory inspection, lexicographic induction, networked entertainment systems, pit-jumping, ...

paper is mainly as a careful record of the current status for repeatability and further development on this important research subject. A short video version of this paper is available for those that hate reading, at http://tom7.org/mario, and is the more fun way to consume the results. This page also contains audiovisual material that makes this work more entertaining (for example, its output) and source code.

The basic idea is to deduce an objective function from

1.1 The NES hardware and emulation

The NES is based around an 8-bit processor running at 1.79 MHz, the Ricoh 2A03. 8 bits is really small. You can see them all right here: 00001111. It’s no coincidence that each controller also has 8 buttons: Up, Down, Left, Right, Select, Start, B and A. It has only 2048 bytes of general purpose RAM. (There is also some special purpose RAM for graphics, which we ignore in this work.) 2048 bytes is really small. You can see them all in Figure 1. As a result, NES programs are written to use memory efficiently and straightforwardly; usually there are fixed memory locations used for all the

1
Novel Alternative Solution

Computer program that learns to play classic NES games
# RAM Locations

<table>
<thead>
<tr>
<th>Address</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Value 3</th>
<th>Value 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>0x35</td>
<td>0x21</td>
<td>0xff</td>
<td></td>
</tr>
<tr>
<td>0x53</td>
<td>0x41</td>
<td>0x42</td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>0x5e</td>
<td>0x32</td>
<td>0x20</td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>0x32</td>
<td>0x3ee</td>
<td>0x3fe</td>
<td>0x00</td>
<td></td>
</tr>
</tbody>
</table>
10 Objective Functions
Proposal

- Faster Automated Coverage Closure
- Efficient Final Stimulus Solution
- Proposed Higher Quality of Coverage
Outline

• Inspiration from Nintendo Playing AI Paper
  – Objective Function

• Application to Simulation
  – Seeds and Dynamic Re-Seeding
  – Objective Function in Verification
  – Checkpointing Feedback Loop

• Scaling

• Results
Interval seed **did** increase the Objective Function

Interval seed **did not** increase the Objective Function

Rewind to last Interval and try new seed
Simulation Time Intervals

Values of Two Bit Vector in Sequence
Dynamic Re-Seeds

• SystemVerilog RNG (Random Number Generator)
  – .randomize()
Dynamic Re-Seeds

• SystemVerilog RNG (Random Number Generator)
  – .randomize()
  – .srandom(int) or UVM reseed()
Objective Function

• Measure of progress
• Covergroup as a simple “Objective Function”
  – Convenient covergroup percentage never decreases during simulation
Objective Function

• Measure of progress
• Covergroup as a simple “Objective Function”
  – Convenient covergroup percentage never decreases during simulation

```python
real coverage_value = 0;
coverage_value = dut.objective.match.get_coverage();
```
Checkpointing

• Saving a simulation point in time with all state
  – performance penalty
  – disabling some optimizations
  – vendor specific, but universal capability
### Murphy VII / Nelson Translations

<table>
<thead>
<tr>
<th>Murphy VII</th>
<th>Nelson</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective Function</td>
<td>Objective Function</td>
</tr>
<tr>
<td>Steps / Iterations</td>
<td>Interval</td>
</tr>
<tr>
<td>Motifs</td>
<td>Constrained Sequences</td>
</tr>
<tr>
<td>Backtracking</td>
<td>Rewinding</td>
</tr>
</tbody>
</table>
Outline

• Inspiration from Nintendo Playing AI Paper
  – Objective Function

• Application to Simulation
  – Seeds and Dynamic Re-Seeding
  – Objective Function in Verification
  – Checkpointing Feedback Loop

• Scaling

• Results
Design Under Test

```vhdl
library work;
use work.std;

entity test_dut is
  port(
    clk, a, b, c : in StdLogic;
    width-1:0 : in Integer;
  );
end test_dut;

architecture test_arch of test_dut is
begin
  process(clk)
  begin
    if rising_edge(clk) then
      c <= a;
    end if;
  end process;
end test_arch;
```
Design Under Test

dut #(parameter width)

width-1:0

a

c
assign c = ( a == b );

b

clk

Digital Comparator
Design Under Test

```vhdl
dut #(parameter width)

width-1:0

a

covergroup objective_cg
coverpoint match;
endgroup

b

clk

c

clk

Digital Comparator
```
Getting the Optimal Solution

• width = 2; about 4% chance of all seeds

\[
\frac{4!}{(2^4)^4} = \frac{4!}{16^4}
\]

• width = 5; probable no seed would solve optimally

\[
\frac{32!}{(2^{10})^{32}} = \frac{32!}{1024^{32}}
\]
Sim Log Example

------------------------ START eval_loop
DEBUG current simulation time is ctime : 27 ns
INFO STATUS : TCL : LOCAL REJECTED seed: 3552075441 at time: 17 ns
INFO STATUS : TCL : 27 ns : NO PROGRESS : false: 0.000000 > 0.000000
REWINDING TO CHECKPOINT {2} at 17 ns
All the Checkpoints created after checkpoint 2 are removed...
------------------------ END eval_loop
START eval_loop
DEBUG current simulation time is ctime: 27 ns
INFO STATUS: TCL: LOCAL REJECTED seed: 3552075441 at time: 17 ns
INFO STATUS: TCL: 27 ns: NO PROGRESS: false: 0.000000 > 0.000000
REWINDING TO CHECKPOINT {2} at 17 ns
All the Checkpoints created after checkpoint 2 are removed...
END eval_loop
Sim Log Example

------------------------- START eval_loop

DEBUG current simulation time is ctime: 27 ns

INFO STATUS : TCL : LOCAL REJECTED seed: 3552075441 at time: 17 ns

INFO STATUS : TCL : 27 ns : NO PROGRESS : false: 0.000000 > 0.000000

REWINDING TO CHECKPOINT {2} at 17 ns

All the Checkpoints created after checkpoint 2 are removed...

------------------------- END eval_loop
Sim Log Example

UVM_INFO sv/dut.sv(14) @ 20: reporter [dut_if] AFTER drive regs a: 0 b: 3
----------------------------- START eval_loop
DEBUG current simulation time is ctime: 27 ns
INFO STATUS : TCL : LOCAL REJECTED seed: 3552075441 at time: 17 ns
INFO STATUS : TCL: 27 ns : NO PROGRESS : false : 0.000000 > 0.000000 REWINDING TO CHECKPOINT {2} at 17 ns
All the Checkpoints created after checkpoint 2 are removed...
----------------------------- END eval_loop
UVM_INFO sv/dut.sv(14) @ 20: reporter [dut_if] AFTER drive regs a: 1 b: 3
----------------------------- START eval_loop
DEBUG current simulation time is ctime: 27 ns
INFO STATUS : TCL : LOCAL REJECTED seed: 3981500775 at time: 17 ns
INFO STATUS : TCL: 27 ns : NO PROGRESS : false : 0.000000 > 0.000000 REWINDING TO CHECKPOINT {2} at 17 ns
All the Checkpoints created after checkpoint 2 are removed...
----------------------------- END eval_loop
UVM_INFO sv/dut.sv(14) @ 20: reporter [dut_if] AFTER drive regs a: 0 b: 0
----------------------------- START eval_loop
DEBUG current simulation time is ctime: 27 ns
INFO STATUS : TCL : LOCAL ACCEPTED seed: 1493068099 at time: 17 ns
INFO STATUS : TCL: 27 ns : GOOD : 25.000000 > 0.000000
----------------------------- END eval_loop
UVM_INFO sv/dut.sv(14) @ 30: reporter [dut_if] AFTER drive regs a: 1 b: 3
Sim Log Example

START eval_loop
DEBUG current simulation time is ctime : 27 ns
INFO STATUS : TCL : LOCAL ACCEPTED seed: 1493068099 at time: 17 ns
INFO STATUS : TCL : 27 ns : GOOD : 25.000000 > 0.000000
END eval_loop

Sim Log Example

---------------------------------------  START eval_loop
DEBUG current simulation time is ctime : 27 ns
INFO STATUS : TCL : LOCAL ACCEPTED seed: 1493068099 at time: 17 ns
INFO STATUS : TCL : 27 ns : GOOD : 25.000000 > 0.000000
---------------------------------------  END eval_loop
replicate file

0 ns : -1 -> 0.000000 : seed 2
17 ns : 0.000000 -> 25.000000 : seed 1493068099
27 ns : 25.000000 -> 50.000000 : seed 765542315
37 ns : 50.000000 -> 75.000000 : seed 1532361113
47 ns : 75.000000 -> 100.000000 : seed 893445949
Normalized Probability
Histogram width=5

![Normalized Probability Histogram](image)

```vhdl
dut #(parameter width)
  a
c
  b
  assign c = ( a == b );
```
Cumulative Distribution
width=5
Outline

• Inspiration from Nintendo Playing AI Paper
  – Objective Function

• Application to Simulation
  – Seeds and Dynamic Re-Seeding
  – Objective Function in Verification
  – Checkpointing Feedback Loop

• Scaling

• Results
Scaling

• Imagine many simulations each trying seeds independently
  – all results sent to a central Server
Scaling

• Imagine many simulations each trying seeds independently
  – all results sent to a central Server

TCL Server

x20 and beyond simulators
TCL Server
TCL Server

REJECT

1%

7

20ns
Outline

• Inspiration from Nintendo Playing AI Paper
  – Objective Function

• Application to Simulation
  – Seeds and Dynamic Re-Seeding
  – Objective Function in Verification
  – Checkpointing Feedback Loop

• Scaling

• Results
Iterations Required

width=5
Iterations Required width=5
Parallel Potential

![Graph showing number of parallel SystemVerilog simulations vs. median attempts per unit time completed.](image-url)
• **Bernard Lowe**: It's the code you added, sir. It has some, uh...
Robert Ford: "Mistakes" is the word you're too embarrassed to use. You ought not to be. You're a product of a trillion of them. Evolution forged the entirety of sentient life on this planet using only one tool: the mistake.

- Westworld, Season 1 ep. 3, HBO, 2016
Results

- ✔ Faster Automated Coverage Closure
- ✔ Efficient Final Stimulus Solution
- TBD Proposed Higher Quality of Coverage
Compare to Formal

• Formal can’t use classes
  – No UVM
  – No concept of seed
  – Verification IP and Environments Restricted to Synthesizable
Compare to Graph

• Graph Based Stimulus

• Objective Function versus Graph
  – Defined Stimulus (Graph)
  – Defined Objective (Proposal)

• Different approaches to generating desired stimulus
  – Both can be complementary
make

> make help
  clean          Cleans up work area
  help           Help Text
  server         Starts up a TCL branching server
  shutdown       Shutdown the the TCL server
  status         Status from the TCL server
  synopsys       Runs a Synopsys Build and does ...
  synopsys_reload Builds and Reloads a simulation from file

Examples
  > make synopsys WIDTH=3
  > make -j5 sim_synopsys_parallel SERVER=127.0.0.1 PARALLEL_SIMS=5
  > make status
  > make sim_synopsys_reload
Limitations

• Improve Probability not Solve Probability

• Simulator Checkpointing Must Be Comprehensive
  – DPI Calls, external C Programs

• Log File is Jumbled on First Run
  – Fixed by running resultant “replicate” file
Limitations

Choice of Interval
+interval_time=x

Choice of Start Time
+start_time=x

Objective Max Target
+max_objective=x

Max Attempts
+max_attempts=x
• Porting the TCL to other Simulators

• Dynamic Intervals
  – Simulated Annealing
  – Murphy VII has ideas beyond the simple set Interval time used here

• Configuration Space Problem
Conclusion

• Inspiration from Tom Murphy VII paper
  – Nintendo AI

• Goals
  – ✔️ Faster Automated Coverage Closure
  – ✔️ Efficient Final Stimulus Solution
  – TBD Proposed Higher Quality of Coverage
Questions
Shy Audience Questions

- Compare this method with the current method of gathering coverage.
- How can you gather configuration coverage with this method?
- Is your paper better than this presentation because ... well, you know.
Backup Slides
Dynamic Re-Seeds

• SystemVerilog RNG (Random Number Generator)
  – .randomize()
  – .srandom(int)

```plaintext
static function void set_seed(int unsigned s);
  this.srandom(s);
endfunction
```
function void pre_randomize();

    if (ms_enable) begin
        ms_run();
    end

endfunction
function void ms_run();

...

type_id = get_type_name();
type_id2 = {uvm_instance_scope(), type_id};

if (uvm_pkg::uvm_random_seed_table_lookup.exists(inst_id))

...

reseed();

endfunction
Start

Request:
- Seed
- Simulation Time
- Objective Function Value

Does a Solution for this Simulation Time Exist?

Yes: Response: EXISTING

No: Does Seed at Simulation Time Improve Objective Function?

No: Response: REJECTED

Yes: Adopt Seed at Simulation Time

Yes: Response: ACCEPTED
Calls `srandom` on the object to reseed the object using the UVM seeding mechanism, which sets the seed based on type name and instance name instead of based on instance position in a thread.