2024
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
MARCH 4-7, 2024

Four Problems with Policy-Based Constraints and How to Fix Them

Dillan Mills
Synopsys, Inc.

Chip Haldane
The Chip Abides, LLC

accellera
SYSTEMS INITIATIVE

Constraints and Policy Class Review

Constraints Review

- Random objects and constraints are the foundational building blocks of constrained random verification
- Embedded fixed constraints are simple but lack flexibility
- In-line constraints are marginally more flexible but their definitions are still fixed within the calling context
- In-line constraints must all be specified within a single call to `randomize()`

Policy Class Review

- Policy classes are a technique for applying constraints in a portable, reusable, and incremental manner
- Leverage an aspect of ”global constraints”, simultaneously solving constraints across a set of random objects
- Randomizing a class that contains policies also randomizes the policies
- The policies contain a reference back to the container
- Consequently, the policy container is constrained by the policies it contains

Policy Class Example: `policy_base`

```
class policy_base#(type ITEM=uvm_object);
    ITEM item;

    virtual function void set_item(ITEM item);
        this.item = item;
    endfunction
endclass
```

Policy Class Example: `policy_list`

```
class policy_list#(type ITEM=uvm_object) extends policy_base#(ITEM);
    rand policy_base#(ITEM) policy[$];

    function void add(policy_base#(ITEM) pcy);
        policy.push_back(pcy);
    endfunction

    function void set_item(ITEM item);
        foreach(policy[i]) policy[i].set_item(item);
    endfunction
endclass
```

`policy_base` and `policy_list` Summary

- These two base classes provide the core definitions for policies
- `policy_base` implements the hook back to the policy container
- `policy_list` organizes related policies into groups
- The parameterization requires a unique specialization for each policy-enabled container

Policy Class Example: Implementation

```
class addr_txn;
  rand addr_t addr;
  rand policy_base#(addr_txn) policy[$];

  function void pre_randomize;
    foreach(policy[i])
      policy[i].set_item(this);
  endfunction
endclass
```

```
class addr_policy extends policy_base#(addr_txn);
  rand addr_t addrs[$];

  function void add(addr_t addr);
    addrs.push_back(addr);
  endfunction

  constraint c_addr {
    item.addr inside {addrs};
  }
endclass
```

- `addr_txn.addr` will be constrained to one of the values added through `addr_policy`

Policy Class Example: Usage

```
class addr_constrained_txn extends addr_txn;
  function new;
    addr_policy addr_policy = new;
    policy_list#(addr_txn) pcy = new;
    addr_policy.add('h00000000);
    addr_policy.add('h10000000);
    pcy.add(addr_policy);
    this.policy = {pcy};
  endfunction
endclass
```

- Instantiate and randomize like normal with a call to `txn.randomize();`
- Each value added to the policy list will be used to constrain the `addr` field

Problem #1: Parameterized Policies

Problem #1: Parameterized Policies

- `policy_base` is parameterized to the class it constrains
- Different specializations cannot be grouped and indexed
- An extended class hierarchy with constrainable values in each level requires a unique policy type and policy list for each level
- Each list must be separately traversed and mapped back to the container during `pre_randomize`
- Users have to keep track of the different lists and which signals apply to each list

Parameterized Policies: Example

```
class addr_p_txn extends addr_txn;
  rand bit parity;
  rand policy_base#(addr_p_txn)
↪   p_policy[$];

  function void pre_randomize;
    foreach(p_policy[i])
      p_policy[i].set_item(this);
  endfunction
endclass
```

```
class addr_c_txn extends addr_p_txn;
  function new;
    policy_list#(addr_txn) pcy = new;
    policy_list#(addr_p_txn) p_pcy = new;
    pcy.add(/*addr policies*/);
    p_pcy.add(/*parity policies*/);
    this.policy = {pcy};
    this.p_policy = {p_pcy};
  endfunction
endclass
```

- This method will not scale—each additional subclass requires a new policy type and list

Parameterized Policies: Solution

- Replace the parameterized policy base with a non-parameterized base
- Move the parameterization to an extension class that implements the base

```
interface class policy;
  pure virtual function void set_item(uvm_object item);
endclass

virtual class policy_imp#(type ITEM=uvm_object) implements policy;
  protected rand ITEM m_item;

  virtual function void set_item(uvm_object item);
    if (!$cast(m_item, item)) /* cleanup */;
  endfunction
endclass

typedef policy policy_queue[$];
```

Policy Interface and Implementation Classes

- Non-parameterized base enables all policies targeting a particular class hierarchy to be stored in a single common `policy_queue`
- Parameterized `policy_imp` implements the base and provides core functionality required by all policies
- No strong typing means all implementing classes must share a common base class— `uvm_object` is a safe choice for UVM testbenches
- `set_item()` must handle the cases where an invalid type is passed in

Policy Definition Updates

- Policy definitions are mostly still the same as when using parameterized classes
- Policy classes should be updated to extend the new `policy_imp` class

```
class addr_policy extends policy_imp#(addr_txn);
```

- Underlying constraints should be written as implications to verify `item` is not missing

```
constraint c_addr {m_item != null -> m_item.addr inside {addrs};}
```

Policy Implementation and Usage Updates

- The base txn class needs to inherit from `uvm_object` to be type-compatible
- The `policies` list in the base txn is replaced with a `rand policy_queue policies` declaration
- subclasses of the base txn class no longer need their own policy lists or `pre_randomize()` functions
- The constrained txn can push all policies into the shared `policy_queue` in the base txn class

Policy Usage Example

```
class addr_txn extends uvm_object;
  rand policy_queue policies;
// ...
class addr_c_txn extends addr_p_txn;
  function new;
    // ...
    this.policies.push_back(/*addr_txn policies*/);
    this.policies.push_back(/*addr_p_txn policies*/);
  endfunction
endclass
```

Problem #2: Definition Location

Problem #2: Definition Location

- “Where should I define my policy classes?”
- Easy enough to stick them in a file close to the class they are constraining
- Better solution: directly embed policy definitions in the class they constrain
  - Eliminates all guesswork about where to define and discover policies
  - Embedded policies gain access to all members of their container class (including protected properties and methods)

Embedded Policy Example

```
class addr_txn extends uvm_object;
  class POLICIES;
    /* policy definitions */
  endclass
endclass

class addr_p_txn extends addr_txn;
  class POLICIES extends addr_txn::POLICIES;
    /* additional policy definitions */
  endclass
endclass
```

```
class addr_c_txn extends addr_p_txn;
  function new;
    addr_c_txn::POLICIES::addr_policy a_pcy = new(/*...*/);
    this.policies.push_back(a_pcy);
  endfunction
endclass
```

- Wrap the policies in a `POLICIES` class to optimize organization
- subclass `POLICIES` extend from parent `POLICIES`

Optimize Further

- Mark properties `protected` so they can only be manipulated with policies
- Add static functions to instantiate and initialize policies with a single call

```
class addr_txn extends uvm_object;
  protected rand a_t addr;
  class POLICIES;
    // addr_policy definition
    static function addr_policy FIXED_ADDR(a_t a);
      FIXED_ADDR = new(a);
    endfunction
  endclass
endclass

class addr_c_txn extends addr_p_txn;
  function new;
    this.policies.push_back(
      addr_c_txn::POLICIES::FIXED_ADDR('hFF00));
  endfunction
endclass
```

Problem #3: Boilerplate Overload

Problem #3: Boilerplate Overload

- Content

Problem #4: Unexpected Policy Reuse Behavior and Optimizing for Lightweight Policies

Problem #4: Unexpected Policy Reuse Behavior and Optimizing for Lightweight Policies

- Content

More Improvements to the Policy Package

More Improvements to the Policy Package

- Content

Conclusion

Conclusion

- Content

Questions?