# Section 20

### Stefen Boyd

Since the only thing that can be randomized is a class, this whole section should be a subsection of 11. At least it should be just after 11. If added to section 11, some of the sections may need to be grouped as this would be a large subsection in contrast with lots of tiny ones.

It seems natural to group this with section 11 since the bnf for 11 will include most all of the rand stuff since it's part of the way classes are declared.

Response by Arturo: Constraints are indeed built on top of classes, but classes are perfectly useful without constraints. Constraints are declared inside classes, but constraints are not how classes are declared. I don't think there is a need to merge these two rather large sections into one. It may just confuse users. We could move the two sections so that they are next to one another.

# General

### Stefen Boyd

If we don't already have an issue open for the random algorithm, we should have one. Granted, constraint solving may not be able to be specified, but given no constraints, the same random members of a class should have the same sequence between different implementations. The random algorithm used for all randomization w/o constraints should be identified (randomize(), $urandom, $srandom, $urandom_range)

Response by David: This whole discussion was already covered in the previous review of this chapter. The decision we made then was that since we would not be standardizing the constraint solver there is no point in standardizing the random functions.

### Neil Korpusik

We discussed providing the ability of plugging other solvers to a particular implementation. Where is that covered?

Response by Arturo: That will have to be covered by some C-API and doesn't need to be part of the language LRM. We should open the issue with SV-CC.

### Stefen Boyd

All bnf for the constraints should go into one spot (in 20.4). Unless it's a problem, I'll have Stu make the necessary changes when I give him the BNF syntax boxes.

Response by Arturo: See CH-116

### Stefen Boyd

Several references to "scalar" that should be "singular"

Response by David: See CH-116.

## Section 20.2

### Neil Korpusik

Second to last paragraph. Sentence in green, why is that?

By default, **pre_randomize()** and **post_randomize()** call their overloaded parent class methods. When **pre_randomize()** or **post_randomize()** are overloaded, care must be taken to invoke the parent class' methods, unless the class is a base class (has no parent class).

Response by Arturo: See CH-116.

## Section 20.3

### Neil Korpusik

Third and fourth bullets: What happens to the previous dynamic array contents?

Example in page 201: Typo in example of constraint db (; and } swapped).

Last bullet item in page 201: "An object variable …". It's not clear. What's an object variable? What are the other class variables?

Response by Arturo: See CH-116

## Section 20.4

### Neil Korpusik

First paragraph: Is there a restriction on the class declaration order? Doesn't seem right.

Response by Arturo: See CH-116

## Section 20.5

### Stefen Boyd

Sections 20.5 through 20.12 should all be 20.4.1 through 20.4.x since they relate to constraints. 20.16 through 20.18 should also be moved to subsections of 20.4.

Response by Arturo: Will that make things clearer? 20.18 describes how to modify constraints, and it's at the end since it summarizes what was covered before.

## Section 20.6

### Stefen Boyd

Inheritance: The last bullet doesn't belong here. It should go in 20.13.2.

Response by Arturo: Section 20.13.2 describes the class methods associated with random constraints. Section 20.6 describes the rules for inheriting and overriding constraints, which are strictly declarative (not methods).

## Section 20.7

### Stefen Boyd

Paragraph beginning with "*value_range_list*" ends with statement that "The bound to the left of the colon MUST be less..." What if it isn't? Is that an error or do you get the empty list? Is this only a compile check? Is it possible to get variable values in ranges that would violate at runtime?

Response by Arturo: These questions are explained in the text. If the left bound is not less than the right then "the range is empty an contains no values". The implication is that the constraint will never be satisfied (i.e., an inside {} is always false). Clearly, this check can not just be a compile check (there is an example that shows the range as all variables.

### Neil Korpusik

The 2nd grammar rule "*expression* **inside** *array*;": What's array? Needs an example.

Response by Arturo: See CH-116

## Section 20.9

### Neil Korpusik

Example in page 206: Typo in constraint c: (; and } swapped).

Last paragraph of this section "It is important …" is appropriate for section 20.12.

Response by Arturo: See CH-116

## Section 20.10

### Neil Korpusik

Change 1st sentence from "constraint are" to "constraint is" or "constraints are".

The use of anonymous constraint may be misleading. Suggest unnamed instead.

Response by Arturo: See CH-116

## Section 20.12

### Neil Korpusik

Fifth bullet item (page 208). Circular dependencies: Run-time or compile-time check?

Response by Arturo: Some circular dependencies can be determined at compile time. But, in the general case, checking for circularity requires run-time checks because constraints may be turned on and off, so what may appear as circularity to the compiler may turn out to be OK at run-time. In any event, the language doesn't specify and leaves it up to the implementations.

## Section 20.13.1

### Neil Korpusik

Change the last sentence in the section ("Checking results …" ) from an advisory to a justification.

Response by Arturo: See CH-116

## Section 20.13.2

### Stefen Boyd

pre/post_randomize(): Super is being checked in if statement to see if it's true. Seems that we need consistency between object handles and the handle type. I like this shortcut for "super!=null" (that is what it means isn't it?), but if we can do it here, we should be able to do it with the handle data type.

Response by Arturo: This implicit comparison to null is allowed by object handles also. Section 3.7 defines null as having a value of zero, so this is all consistent.

### Neil Korpusik

Should move randomize restrictions to randomized section (20.13.1)?

Typo in third bullet item "is not be" => "is not".

Response by Arturo: See CH-116

## Section 20.15.1

### Stefen Boyd

All the rest of the testbench donation avoided creating reserved words "on" and "off" but here they are again with $constraint_mode and $rand_mode. Someone (Arturo) needs to come up with alternate syntax soon (I need it for the bnf).

Response by Arturo: The document should state that these are only conceptual. They have the values 0 and 1.

It would appear that these should simply be methods for use on either an object or it's members. Especially since they can't be used with multiple objects at the same time. How about rand_on() rand_off, constraint_on(), constraint_off()?

### Neil Korpusik

Shouldn't constraint_mode / rand_mode be methods?

Response by Arturo: These were not made into methods because they refer to the constraint. Without the object qualifier, the constraint is not visible from the context of the caller. Making into a method would require passing the constraint as a string and prevent the compiler from checking it, otherwise it can lead to ambiguity. For example:

```
task foo( Object p );
    int b;
    p.constraint_mode( 1, b ); // b may a constraint in p, but it's an int in foo's scope
    $constraint_mode( 1, p.b );          // here b is unambiguous and correct verilog
endtask
```

## Section 20.16.1

### Stefen Boyd

All the rest of the testbench donation avoided creating reserved words "on" and "off" but here they are gain with $constraint_mode and $rand_mode. Someone (Arturo) needs to come up with alternate syntax soon (I need it for the bnf).

It would appear that these should simply be methods for use on either an object or it's members. Especially since they can't be used with multiple objects at the same time. How about rand_on() rand_off, constraint_on(), constraint_off()?

Response by Arturo: See answers for 20.15.1 (same issue really).