PROPOSAL: *pure* and *extern* keywords for constraint prototypes

Relates to: svdb-2514, ballot comment #182. Applies to: IEEE P1800 ballot draft 8.

Version 1, 27 April 2009

Version 1a, 29 April 2009: friendly amendment from Tom Alsop

Version 2, 05 May 2009: removed mention of "obligation" in 18.5.1

Version 3, 22 June 2009: pure constraint can replace an inherited constraint (response to Champions feedback)

REPLACE the whole of subclauses 18.5.1 (External constraint blocks) and 18.5.2 (Inheritance) with the following two subclauses:

18.5.1 External constraint blocks

Constraint blocks can be declared outside their enclosing class declaration if a *constraint prototype* appears in the enclosing class declaration. A constraint prototype specifies that the class shall have a constraint of the specified name, but does not specify a constraint block to implement that constraint. A constraint prototype can take either of two forms, as shown in the example below:

```
class C;
  rand int x;
  constraint proto1;  // implicit form
  extern constraint proto2; // explicit form
endclass
```

For both forms the constraint can be completed by providing an *external constraint block* using the class scope resolution operator, as in the following example:

```
constraint C::proto1 { x inside {-4, 5, 7}; }
constraint C::proto2 { x >= 0; }
```

An external constraint block shall appear in the same scope as the corresponding class declaration, and shall appear after the class declaration in that scope. If the explicit form of constraint prototype is used, it shall be an error if no corresponding external constraint block is provided. If the implicit form of prototype is used and there is no corresponding external constraint block, the constraint shall be treated as an empty constraint and a warning may be issued. An empty constraint is one that has no effect on randomization, equivalent to a constraint block containing the constant expression `1`.

For either form, it shall be an error if more than one external constraint block is provided for any given prototype, and it shall be an error if a constraint block of the same name as a prototype appears in the same class declaration.

18.5.2 Constraint inheritance

Constraints follow the same general rules for inheritance as other class members. The `randomize()` method is virtual and therefore honors constraints of the object on which it was called, regardless of the data type of the object handle through which the method was called.

A derived class shall inherit all constraints from its superclass. Any constraint in a derived class having the same name as a constraint in its superclass shall replace the inherited constraint of that name. Any constraint in a derived class that does not have the same name as a constraint in the superclass shall be an additional constraint.

If a derived class has a constraint prototype with the same name as a constraint in its superclass, that constraint prototype shall replace the inherited constraint. Completion of the derived class's constraint prototype shall then follow the rules described in **18.5.1** above.

An abstract class (*i.e.* a class declared using the syntax `virtual class`, as described in **8.20**) may contain *pure constraints*. A pure constraint is syntactically similar to a constraint prototype but uses the `pure` keyword, as in the example below:

```
virtual class D;
  pure constraint Test;
endclass
```

A pure constraint represents an obligation on any non-abstract derived class (*i*.e. a derived class that is not `virtual`) to provide a constraint of the same name. It shall be an error if a non-abstract class does not have an implementation of every pure constraint that it inherits. It shall be an error to declare a pure constraint in a non-abstract class.

It shall be an error if a class containing a pure constraint also has a constraint block, constraint prototype or external constraint block of the same name. However, any class (whether abstract or not) may contain a constraint block or constraint prototype of the same name as a pure constraint that the class inherits; such a constraint shall override the pure constraint, and shall be a non-pure constraint for the class and any class derived from it.

An abstract class that inherits a constraint from its superclass may have a pure constraint of the same name. In this case, the pure constraint in the derived virtual class shall replace the inherited constraint.

A constraint that overrides a pure constraint may be declared using a constraint block in the body of the overriding class, or may be declared using a constraint prototype and external constraint as described in **18.5.1**.

ADD the following text at the end of 18.5.10 (Static constraint blocks)

When a constraint is declared using a constraint prototype and an external constraint block, the `static` keyword shall be applied to both the constraint prototype and the external constraint block, or to neither. It shall be an error if one but not the other is qualified `static`. Similarly, a pure constraint may be qualified `static` but any overriding constraint must match the pure constraint's qualification or absence thereof.

REPLACE syntax production *constraint_prototype* in A.1.10

~~constraint_prototype ::= [**static**] **constraint** constraint_identifier **;**~~

WITH new productions:

constraint_prototype ::= [constraint_prototype_qualifier] [**static**] **constraint** constraint_identifier **;**

constraint_prototype_qualifier ::= **extern** | **pure**

REPLACE syntax production *constraint_prototype* in Syntax 18-2

~~constraint_prototype ::= [**static**] **constraint** constraint_identifier **;**~~

WITH new productions:

constraint_prototype ::= [constraint_prototype_qualifier] [**static**] **constraint** constraint_identifier **;**

constraint_prototype_qualifier ::= **extern** | **pure**