My action items from the August 1, 2002 meeting

Cindy Eisner

IBM Haifa Research Laboratory
eisner@il.ibm.com

1 Formal semantics of the `until!` operator in English and in math

– In English:

An `until!` property holds in a given cycle iff:
- the property that is the right operand holds at the current cycle or at some future cycle; and
- the property that is the left operand holds at all cycles up to but not necessarily including that cycle.

– In math:

$\omega \models f_1$ *until*! $f_2 \iff$ there exists $k \in [0..|\omega|)$ such that $\omega^k \models f_2$, and for every $j \in [0..k)$, $\omega^j \models f_1$

2 How to define semantics independently of a particular implementation

Currently, the semantics of sequential assertions in SystemVerilog 3.0 are defined in terms of a particular implementation, namely spawning a process. The disadvantages of this are that:

1. It is impossible to know what an assertion means for tools, such as formal verification tools, which do not spawn processes.
2. It prevents optimizations which are equivalent to the intended semantics, but do not spawn processes. For instance, checking for multiple instances of the same sequence offset in time can be done using a single process by determinizing a non-deterministic state machine.

Rather, I suggest defining the semantics independently of a particular implementation. For instance, the semantics of a sequential assertion might be defined in two parts. The first part separates out the assertion from the rest of the code, resulting in modified Verilog code plus a standalone assertion. The second part provides semantics to the standalone assertion. For instance[1]:

```
always @(...)
  if (p == q)
    assert (a;b;c) @@ (posedge clk);
```

would be (automatically) translated to the following modified Verilog code:

```
always @(...)
begin
  trigger = 0;
  if (p == q)
  begin
    assert a;
    trigger = 1;
  end
end
```

plus the following standalone assertion:

```
assert {trigger} |=> {{b;c}@(posedge clk)};
```

I have used Sugar syntax for the standalone assertion, which simply states that whenever `trigger` is asserted, `b` should be asserted at the following positive edge of `clk`, and `c` at the one following that. The first item in the sequence is left as a simple boolean assertion. This ensures that the intended (possibly transitional) value of `a` is seen, in both combinational always blocks as well as sequential ones. The remainder of the sequence is pulled into a standalone assertion to which formal semantics can be given.

Now, a simulation tool which wishes to implement the standalone assertion using spawned processes is free to spawn a process at every assertion (positive edge for an event-based tool) of `trigger`, as described in the existing SystemVerilog 3.0 specification. A formal tool, or a simulation tool which wishes to be more efficient, can implement the standalone assertion using determinization of a non-deterministic state machine.

[1] For simplicity, I have used the simple sequence `(a;b;c)`. The solution is extendable to more complicated cases, for instance, to the sequence `(a*[0:3];b;c)`. Although such a solution is beyond the scope of this short document, I will happily supply it if there is interest.