Formal Reasoning With Verilog HDL

Gordon J. Pace[1]
University of Malta

Jifeng He[2]
University of Oxford

[1] `gpace@cs.um.edu.mt`
[2] `jifeng.he@comlab.ox.ac.uk`

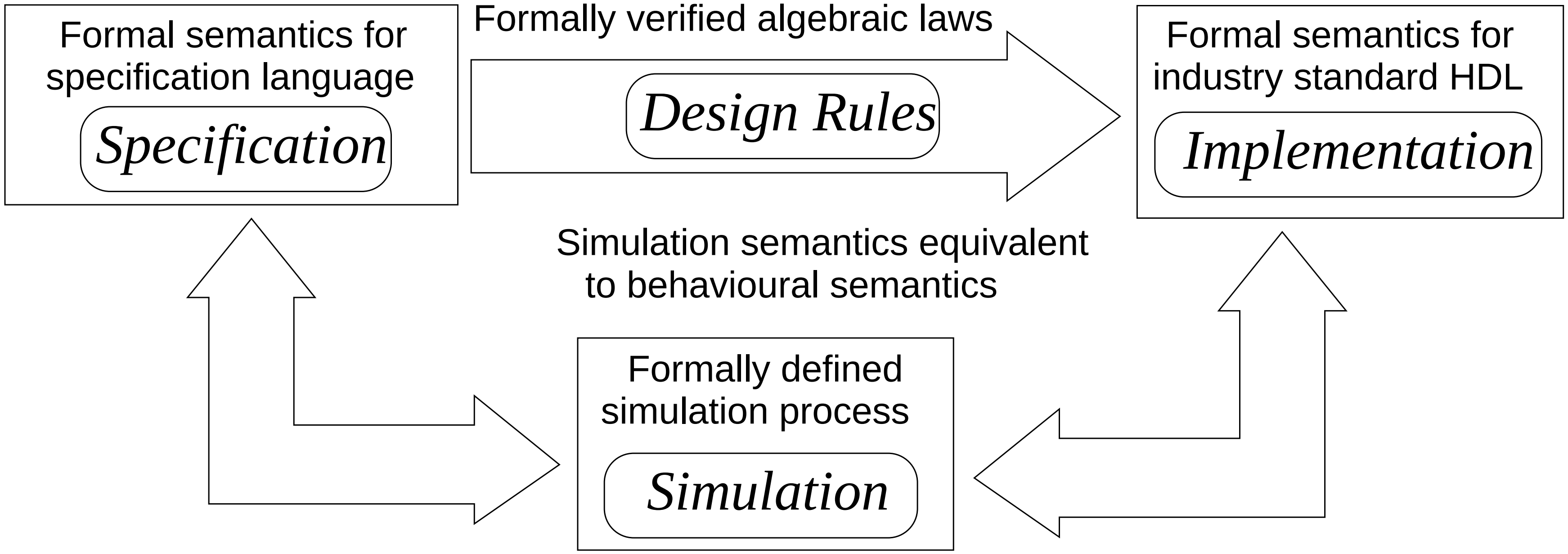
The Problem

There are two main approaches to hardware 'verification':

- **Industry:** Promotes (and uses) simulation
- **Academia:** Promotes formal verification

Can these two be amalgamated into one framework?

The Proposed Framework



Choosing an HDL

1. Classes of HDLs:
   - Develop tailormade HDL
   - Academically developed HDL
   - Industry standard HDL
2. Choosing an industry standard HDL: Verilog or VHDL?

Structural and Behavioural Specification

```
module NOT_beh (in, out);
input in; output out;
      always  @in out = ~in;
endmodule
```

```
module NOT_struc (in, out);
input in; output out; wire p,g;
     POWER  P1  (p);
      GROUND G1  (g);
      PTRANS PT1 (p,in,out);
      NTRANS NT1 (g,in,out);
endmodule
```

Verilog

Imperative programming-like instructions:

- Assignment: `v=e`
- Sequential Composition: `P; Q`
- Conditional, loops, etc

Time-control instructions:

- Time guards: `#n`
- Wait for signal level: `wait v`
- Wait for signal edge: `@v`, `@posedge v`, etc

Verilog: Top Level Modules

One-off execution: `initial P`

Repeated execution: `always P`

Modules can be put in parallel using the $\|$ operator.

Defining the Semantics

- Temporal logics: Duration Calculus
- Representation of events:

$$\begin{aligned} \mathit{TIME} &= \mathbb{R}^+ \\ \mathit{State} &: \mathit{TIME} \to \mathbb{B} \\ \mathit{Duration\ Formula} &: \mathit{INTERVAL} \to \mathbb{B} \end{aligned}$$

Duration Calculus

- $l = n$ means that the current time interval is of length n:
$$(l = n)[b, e] \stackrel{def}{=} (e - b = n)$$
- $\lceil P \rceil^*$ means that state P holds over the whole time interval:
$$\lceil P \rceil^*[b, e] \stackrel{def}{=} \int_b^e P\, dt = e - b$$
- $P; Q$ means that the interval can be split into two where P holds over the first and Q over the second:
$$(D; E)[b, e] \stackrel{def}{=} \exists m \in [b, e] \cdot D[b, m] \wedge E[m, e]$$

Relational Duration Calculus

Pre-value: $\overleftarrow{v}$ represents the value of v just before the current time interval.

Post-value: $\overrightarrow{v}$ represents the value of v just after the current time interval.

Relational Chop: $D \fatsemi E$ is similar to the chop operator except that it also enforces that the post-values in D are the same as the pre-values in E.

The Semantics

$$[\![\texttt{P}]\!]_W(D)$$

The semantics of program `P`, with output alphabet W, such that at the end of the execution of `P`, the system acts according to duration formula D.

Language restrictions:

No merge on output wires: Only one module may write to any variable.

Registers as state holders: Global variables are cannot be written to using immediate assignments.

No interference on shared variables: Global signals may not be written to and read from simultanuously.

Imperative Programming Constructs

Immediate assignments: act just like normal assignments …

$$[\![\mathtt{v=e}]\!]_W(D) \stackrel{def}{=} \left(\begin{array}{l} \overrightarrow{v} = \overleftarrow{e} \wedge l = 0 \\ \wedge \mathbf{Const}(W - \{v\}) \end{array} \right) ⨾ D$$

Sequential Composition: just like the normal ; operator:

$$[\![\mathtt{P;Q}]\!]_W(D) \stackrel{def}{=} [\![\mathtt{P}]\!]_W([\![\mathtt{Q}]\!]_W(D))$$

Time Related Instructions (I)

Guards:

$$[\![\texttt{guard}]\!]_W(D) \stackrel{def}{=} (\mathbf{Const}(W) \land \texttt{guard}_{nt}) \lor (\mathbf{Const}(W) \land \texttt{guard}_{ter}) \,;\, D$$

Wait for n time units:

$$\texttt{\#}n_{ter} \stackrel{def}{=} l = n$$

$$\texttt{\#}n_{nt} \stackrel{def}{=} l < n$$

Wait until v becomes true:

$$\texttt{wait}\ v_{ter} \stackrel{def}{=} \lceil \neg v \rceil^* \land \overrightarrow{v} = 1$$

$$\texttt{wait}\ v_{nt} \stackrel{def}{=} \lceil \neg v \rceil^* \land \overrightarrow{v} = 0$$

Time Related Instructions (II)

Guarded Assignments:

$$[\![\texttt{guard v=e}]\!]_W(D) \stackrel{def}{=} [\![\texttt{guard;v=e}]\!]_W(D)$$

$$[\![\texttt{v=guard e}]\!]_W(D) \stackrel{def}{=} \exists\texttt{tmp}\cdot[\![\texttt{tmp=e;guard;v=tmp}]\!]_{W\cup\{\texttt{tmp}\}}(D)$$

Top Level Modules

The semantics of a top-level module can be given in terms of those already given:

$$[\![\texttt{initial P}]\!] \stackrel{def}{=} [\![\texttt{P}]\!]_W(\mathbf{Const}(W))$$

$$[\![\texttt{P} \parallel \texttt{Q}]\!] \stackrel{def}{=} [\![\texttt{P}]\!] \wedge [\![\texttt{Q}]\!]$$

Case Study: Synchronising Handshake

$$\texttt{HS} \overset{def}{=} \texttt{x=\#1 1; wait y; x=\#1 0}$$

$$\begin{array}{lll} \texttt{COMM} & \overset{def}{=} & \texttt{initial x=0; P1; HS(x,y); P2} \\ & \| & \texttt{initial x=0; Q1; HS(y,x); Q2} \end{array}$$

Can we prove that `P1` and `Q2` are never executed at the same time?

Can they have a common execution point?

Do `P2` and `Q2` start at the same time?

Case Study: Synchronising Processes

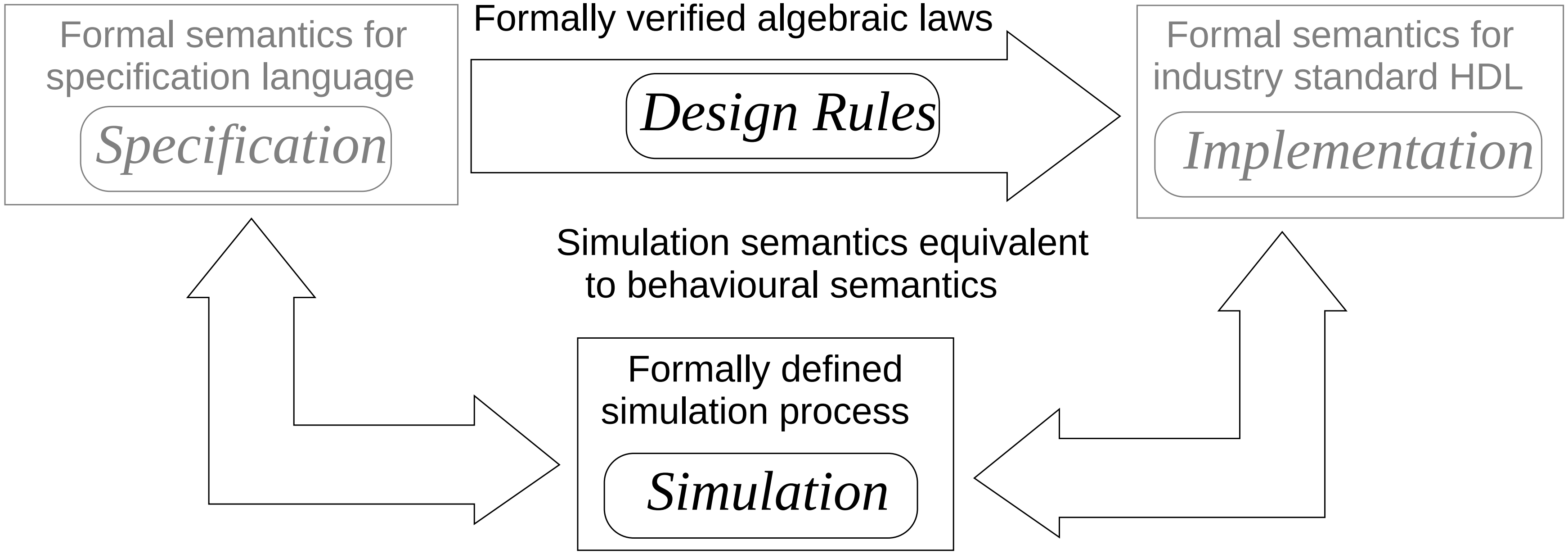
Given a queue handler (QH) and two clients (CL1 and CL2):

```
QH = u1, u2 = 0, 0;
     forever begin
       wait (g1 or g2);
       #1 u1, u2 = g1, g2 and not g1;
       if (u1) then wait (not g1);
               else wait (not g2);
       #1 u1, u2 = 0, 0;
     end;
```

```
CL1 = g1 = 0;
      P1;
      g1 = #1 1;
      wait (u1);
      Q1;
      g1 = #1 0;
```

Can we prove that Q1 and Q2 are mutually exclusive?

Conclusions and Further Work



Conclusions and Further Work

Plus:

Algebraic Laws: A number of algebraic laws allowing comparison of programs without resorting to the complex semantics have been derived.

Real-Time Case Study: Taking advantage of the use of DC, we have also used these semantics to reason from a (small) real-time specification down to an implementation in Verilog.

In the near future ...

- Study further the conversion from real-time specifications to Verilog programs.
- Mechanising some of the (boring) aspects of these proofs.