EE224 Handout
Formal Verification: basic concepts and tools

Madhav P. Desai

April 21, 2018

1 Formal verification of combinational systems

Verification is the process of confirming that an implementation is consistent with its specification. For example, suppose we are asked to design an adder, and we implement it using a carry-lookahead scheme, then we must verify that our implementation does in fact function like an adder, as specified.

To make the problem more concrete, we may assume that $x_1, x_2, \ldots, x_n$ are inputs to the the system, which computes outputs $y_1, y_2, \ldots y_m$. The specification must be reduced to a set of Boolean functions

$$\begin{aligned} y_1 &= f_1^{Spec}(x_1, x_2, \ldots x_n) \\ y_2 &= f_2^{Spec}(x_1, x_2, \ldots x_n) \\ \ldots & \\ y_m &= f_m^{Spec}(x_1, x_2, \ldots x_n) \end{aligned}$$

Define the *characteristic* function:

$$\chi_{spec}(x_1, x_2, \ldots x_n, y_1, y_2, \ldots y_m) \;=\; \Pi_{i=1}^{m} \overline{(y_i \oplus f_i^{spec}(x_1, x_2, \ldots, x_n))}$$

the specification equations are satisfied if and only if

$$\chi_{spec} \;=\; 1$$

For example: if our specification is to build a two bit full adder, the inputs are a, b, c_{in} and the outputs are sum, c_{out}. The Boolean functions representing the specification could be

$$\begin{aligned} sum &= a \oplus b \oplus c_{in} \\ c_o ut &= (a.b) + ((a+b).c_{in}) \end{aligned}$$

Thus the specification is captured by the single Boolean function

$$\chi_{spec}(a, b, c_{in}, sum, c_{out}) = \overline{(sum \oplus (a \oplus b \oplus c_{in}))}.\overline{(c_{out} \oplus ((a.b) + ((a+b).c_i n)))}$$

and we must have

$$\chi_{spec} = 1$$

for the specification equations to be satisfied.

The implementation is another set of Boolean functions

$$\begin{aligned} y_1 &= f_1^{Impl}(x_1, x_2, \ldots x_n) \\ y_2 &= f_2^{Impl}(x_1, x_2, \ldots x_n) \\ \ldots & \\ y_m &= f_m^{Impl}(x_1, x_2, \ldots x_n) \end{aligned}$$

Define the characteristic function of the implementation to be

$$\chi_{impl}(x_1, x_2, \ldots x_n, y_1, y_2, \ldots y_m) = \Pi_{i=1}^{m} \overline{y_i \oplus f_i^{spec}(x_1, x_2, \ldots, x_n)}$$

Clearly, the implementation is equivalent to the specification if and only if

$$\chi_{spec} \oplus \chi_{impl}$$

can *never* evaluate to 1.

For example, if the implementation of the full adder is as shown in Figure 1, then the equations which represent the implementation are

$$\begin{aligned} u &= a.b \\ v &= a \oplus b \\ w &= v.c_{in} \\ sum &= c_{in} \oplus v \\ c_{out} &= u + v \end{aligned}$$

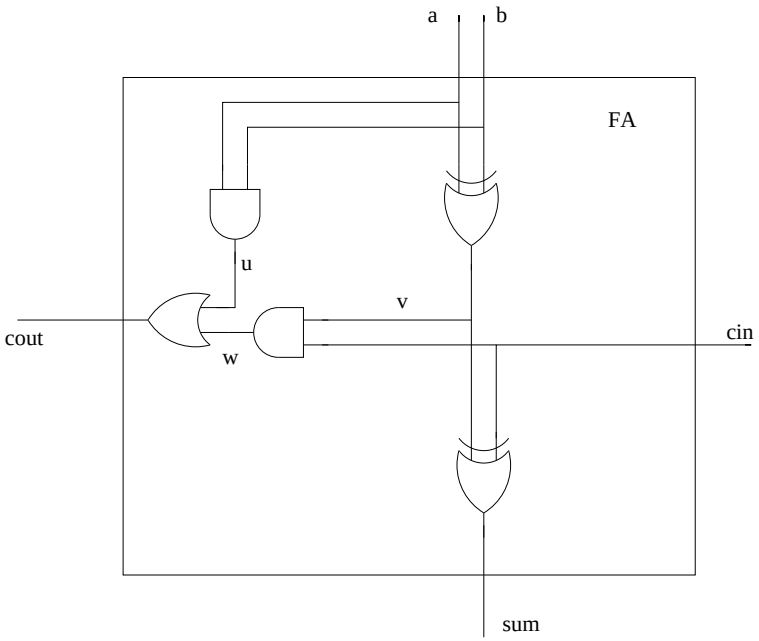

Figure 1: Full Adder

These can be reduced to a single function as follows

$$\begin{aligned}\chi_{impl} &= \overline{(u \oplus (a.b))} \\ &\quad .\overline{(v \oplus (a \oplus b))} \\ &\quad .\overline{(w \oplus (v.c_{in}))} \\ &\quad .\overline{(sum \oplus (c_{in} \oplus v))} \\ &\quad .\overline{(c_{out} \oplus (u + v))}\end{aligned}$$

We must confirm that

$$\Xi = \chi_{spec} \oplus \chi_{impl}$$

can never evaluate to 1.

Thus the simplest way to verify that two combinational circuits are equivalent is to construct the function Ξ and show that it is not satisfiable, that is, there is no input combination for which the function evaluates to 1.

1.1 What about don't cares

Sometimes one is given a set of don't cares in a specification. Suppose that the don't cares are defined by

$$D(x_1, x_2, \ldots, x_n) \;=\; 1$$

How would our verification change? Basically, if D evaluates to 1, then this is a don't care combination so that it doesn't matter if the specification and implementation are different. On the other hand if D evaluates to 0, then χ_{spec} and χ_{impl} must agree. Thus all we need to do is check that the function

$$\overline{D} \Rightarrow \overline{\chi_{spec} \oplus \chi_{impl}}$$

can never take the value 0. Recall that $a \Rightarrow b = \overline{a} + b$. Taking complements we obtain the function

$$D.(\chi_{spec} \oplus \chi_{impl})$$

which should never evaluate to 1 (should not be satisfiable).

2 Image and pre-image computation

Suppose you are given a function $f : \mathbf{B}^n \rightarrow \mathbf{B}^m$. Often we will need to compute the image $f(S)$ of a set $S \in \mathbf{B}^n$, or a pre-image $f^{-1}(T)$ of a set $T \in \mathbf{B}^m$. This is very useful in verifying sequential machines as we shall see later.

The direct mechanism which allows us to do this is as follows. Let f be described as a collection of simple Boolean function $f_i : \mathbf{B}^n \rightarrow \mathbf{B}$ as

$$\begin{aligned} y_1 &= f_1(x_1, x_2, \ldots x_n) \\ y_2 &= f_2(x_1, x_2, \ldots x_n) \\ \ldots & \\ y_m &= f_m(x_1, x_2, \ldots x_n) \end{aligned}$$

Define the *characteristic* function:

$$\chi(x_1, x_2, \ldots x_n, y_1, y_2, \ldots y_m) \;=\; \Pi_{i=1}^{m} \overline{y_i \oplus f_i(x_1, x_2, \ldots, x_n)}$$

Let $S \in \mathbf{B}^n$ be defined as the set of points on which a function $f_S : \mathbf{B}^n \rightarrow \mathbf{B}^m$ evaluates to 1. Then the image of S consists of those points $(y_1, y_2, \ldots, y_m)$ for which there exists some $(x_1, x_2, \ldots, x_n) \in S$. That is, we are interested in $\mathbf{y} = (y_1, y_2, \ldots y_n)$ such that

$$\text{there exists } \mathbf{x} = (x_1, x_2, \ldots x_n) \text{ such that } (f_S(\mathbf{x}).\chi(\mathbf{x}, \mathbf{y})) = 1$$

Note that if $g(x_1, x_2)$ is a function of two variables then the existential quantification of g with respect to x_1 is defined as

$$\begin{aligned} \text{there exists } x_1 \text{ such that } g(x_1, x_2) = 1 &= g(0, x_2) + g(1, x_2) \\ &= g_{\overline{x_1}} + g_{x_1} \end{aligned}$$

which is a function of x_2 alone. Further if $g(x_1, x_2, x_3)$ is a function of three variables then we may be interested in calculating

$$h(x_3) \;=\; \text{there exists } x_1, x_2 \text{ such that } g(x_1, x_2, x_3) = 1$$

Now $h(x_3)$ may be obtained by first calculating

$$p(x_2, x_3) = \text{there exists } x_1 \text{ such that } g(x_1, x_2, x_3) = 1$$

and then

$$h(x_3) = \text{there exists } x_2 \text{ such that } p(x_2, x_3) = 1$$

Using this mechanism, of quantification on the variables $x_1, x_2, \ldots x_n$, the image set $f(S)$ can be computed.

The computation of pre-images is similar and you can figure it out yourself...

2.1 An example

Let $g(x_1, x_2, x_3) = x_1.x_2 + x_2.x_3 + x_1.x_3$. Suppose S is the set described by the function $x_1 + x_2 = 1$. Then the function

$$f_S.\chi = (x_1 + x_2).\overline{((x_1.x_2 + x_2.x_3 + x_1.x_3) \oplus y)}$$

will be 1 if and only if x_1, x_2, x_3 are in the S and y is related to them through g.

Existential quantification on x_1 alone gives us

$$x_2.\overline{(x_2.x_3 \oplus y)} + \overline{(x_2 + x_3) \oplus y)}$$

Continuing, and existentially quantifying on x_2 gives us

$$\overline{(x_3 \oplus y)} + \overline{(x_3 \oplus y)} + y$$

Continuing, and existentially quantifying on x_3 gives us

$$1$$

Thus the image of S is the entire range of the function g.

3 Verification of Sequential Circuits

How does one verify that a sequential circuit correctly implements a specification? In general the specification is available as a finite state machine. The implementation is typically a logic circuit with logic gates and flip flops. It is also easy to derive the state machine corresponding to a logic circuit, as was done in class[1]

Once we have state machine descriptions for the specification and for the implementation, we can construct the state machine shown in Figure 2, and try to prove that the output of this state machine can never be 1. To do this, we will first construct the product machine and minimize it by identifying equivalent states. At the end of this process of minimization, we should be left with a single state and confirm that the output is always 0.

4 Sequential circuits: calculating the set of reachable states

Using the image computation idea shown earlier, we can compute the state of reachable states of an FSM.

Suppose that the FSM is defined by the Boolean relations

$$\begin{aligned} y(k) &= \lambda(x(k), q(k) \\ q(k+1) &= \delta(x(k), q(k) \end{aligned}$$

[1]If there are n flip-flops, then the state machine will have 2^n states. The reset state will be the initial state. The next-state function can be found by analysing the combinational logic network obtained by removing the flip-flops from the circuit.

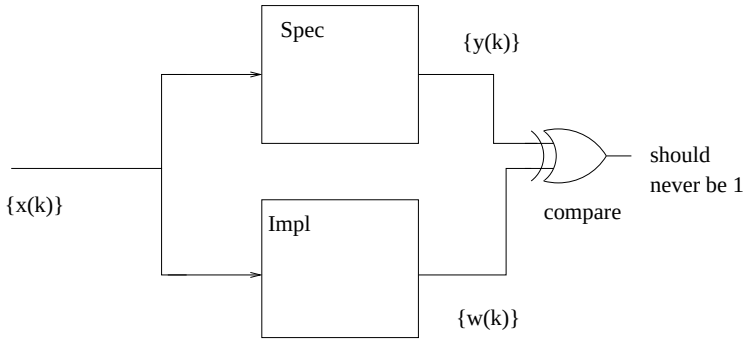

Figure 2: Product FSM

We construct the characteristic function χ of the FSM as

$$\chi(x(k), q(k), q(k+1)) \;=\; (y(k) == \lambda(x(k), q(k))).(q(k+1) == \delta(x(k), q(k)))$$

Now we compute the set of reachable states as follows (we use Boolean functions to represent sets).

1. Assume that at $k = 0$, the set of reachable states is given by a Boolean function $Q_0(q)$. The set of states reachable in 0 steps is also given by R_0. We compute Q_1 by doing the quantification

   there exists $x(0), q(0)$ such that $(Q_0(q(0)).\chi(x(0), q(0), q(1)) = 1)$

   This gives a function of $q(1)$ which is precisely the set of states Q_1 reachable in one step starting at $k = 0$. The set of states reachable in up to 1 step starting from $k = 0$ is then $R_1 = Q_1 + Q_0$ (note: $+$ is union).

2. Now continue from $k = 1$, and obtain $Q_2, Q_3, \ldots$ and $R_2, R_3, \ldots$.

3. Stop when $R_{k+1} = R_k$ for some k (this will always happen eventually).

5 Property checking

What we have seen so far is comparison between a specification and a reference. But how do we check the specification itself? The obvious way is to confirm that the specification satisfies certain properties. For example, in

case of the adder specification, we could specify the property that $a + a = 2a$ always holds. Sometimes these properties are also termed *invariants*.

So the strategy is: convert the properties to be checked into Boolean functions (say $P(\mathbf{x}, \mathbf{y})$). Confirm that for all $\mathbf{x}$ meeting the specification, the property P is always satisfied. That is, the function

$$\chi(\mathbf{x}, \mathbf{y}) \Rightarrow P(\mathbf{x}, \mathbf{y})$$

always evaluates to 1. In case of sequential circuits, the property will need to be expressed in terms of $x(0), x(1), \ldots x(k)$, $y(1), y(2), \ldots y(k)$ and $q(1), q(2), \ldots q(k)$ and the characteristic function may also need to be extended to be a function of $x(j), y(j), q(j)$ for $j \leq k$ etc..

To summarize, there is a lot of formal checking which we can perform by using Boolean analysis. Of course when it is impossible to check formally, we need to simulate, but thats another story.