7.10 Symbolic FSM State Traversal

Many problems in the formal verification of hardware are based on such reachable states computations for FSMs (Finite State Machines). A reachable state is just one that is reachable for some tape (input sequence) from a given set of initial states. This type of computation uses a "symbolic" breadth first search approach to reach all reachable states by shortest paths.

In the context of FSMs, reachable states computations are based on implicit traversal of the STG (State Transition Graph) of the underlying FST. The key step in reachable states computations (and a host of other related computations of formal verification) is the computation of the image, $\text{Im}(\delta(s, z), C(s))$, of a given set of points $C$ in the domain of a specified transition function $\delta(s, z)$. Image computation plays a key role in formal verification, especially in FSM verification based on symbolic traversal of the STG. Preimage computation plays a similar, perhaps even more important role.

In symbolic traversal one computes sets of states, which are reachable in one FSM transition from a specified set of states [71, 73, 60, 262, 74, 79]. The breadth first search method of Procedure BFS_FSM allows one to deal naturally with multiple states simultaneously and has thus become the method of choice for the traversal of large machines[^1]. The full power of this approach is realized when BDDs are used to represent the characteristic functions of these sets (See Section 6.1). This process is sometimes called symbolic simulation of $\lambda$.

7.10.1 Transition Relations and Symbolic Image Computation

Let $\delta_i(s, z)$, $i = 1, \ldots, n$ be the $i^{th}$ encoded next state transition function of a given encoded FSM, and let $s$ and $z$ the coding vectors for states and inputs. A given symbolic state set $C(s)$ (characteristic function) is mapped by $\delta(s, z)$ into a state set $I \subseteq \{0, 1\}^n$ in the range, (or co-domain) of the functional vector $\delta$. The set of such co-domain points is called the image of $s$ under $C(s)$, the definitions following Page 84. In the symbolic approach, the image is typically computed using transition relations.

[^1]: It is also possible to extend the depth-first method to deal with groups of states simultaneously [7], yet in a less general and satisfactory way.

### Definition 7.10.1
Given a deterministic transition function $\delta(s, z)$, the corresponding transition relation $T(s, z, t)$ is defined[^2] by:

$$T(s, z, t) = \prod_{i=1}^{n} (t_i \equiv \delta_i(s, z)).$$

The equation $T(s, z, t) = 1$, denotes a set of encoded triples $s, z, t$ of state $s$, input $z$, and $z$-successor $t$ of $s$, each representing a transition in the FST of the given FSM.

Given the transition relation it is straightforward to compute the image by Boolean manipulations, but for this we need to define a new Boolean operation called existential abstraction.

### Definition 7.10.2
Given an $m$-variable Boolean function $f(x_1, \ldots, x_m)$, the existential abstraction[^3] of $f$ with respect to $x_i$ is:

$$\exists x_i f = f_x + f_{\neg x}. \quad \text{(Smoothing)}$$

Here $f_x = f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_m)$ stands for the positive cofactor of $f$ with respect to $x_i$ and $f_{\neg x} = f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_m)$ stands for the corresponding negative cofactor.

The name derives from the following property, which can be easily verified.

$$\exists x_i f(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = 1 \iff \exists x_i f^+(x_1, \ldots, x_n) = 1.$$  

It can be shown that $f_x$ is the smallest (fewest minterms) function that contains all minterms of $f$, and is independent of $x_i$.

We illustrate this definition by computing $\exists x f$ for the following function.

$$f = z' y' z + z z' + z y.$$  

The two cofactors are:

$$f_x = z' y' + z y' \quad \text{and} \quad f_{\neg x} = z.$$  

Hence, $\exists x f = z + y'$.

Given $f(s, z) = f(s_1, \ldots, s_n, x_1, \ldots, x_m)$, the existential and universal abstractions with respect to a set of variables are easily defined.

$$\exists x f(s, z) = \exists x_i (\exists x_{m-1} (\exists x_m f(s, z))).$$

That is, we first abstract $x_m$ from the original function $f$. Then we abstract $x_{m-1}$ from the result of this abstraction. From this second result we abstract $x_{m-2}$, and so on. The order in which we do these abstractions is immaterial. The final result $g(s) = \exists x f(s, z)$ is the smallest (fewest minterms) function that contains all minterms of $f(s, z)$, and is independent of $x$.

[^2]: Recall that $(s \equiv b) = (ab + \overline{ab})$.

[^3]: There is a corresponding dual operation, called universal abstraction which takes the conjunction rather than the disjunction of the cofactors. $f_{\exists x} = f_x \cdot f_{\neg x} \quad \text{(Consensus)}$
Given the above definition we can easily compute the image of the set $C(s)$ as

$$I(t) = \text{Img}(T, C) = \exists_x \exists_x' C(s) \cdot T(s, x, t).$$

(7.2)

In words, the image computation proceeds as follows. First compute the transition relation $T(s, x, t)$, and then compute the conjunction of this function and the function $C(s)$ and call the result $f(s, x, t)$. Then existentially abstract all the $x$-variables and all the $x'$-variables to obtain the result $I(t)$. The result is the smallest function independent of $s$ and $x$ which contains all the triples in $f(s, x, t)$.

Like this method of image computation, all the other steps of Procedure BFS.FSM can be converted into symbolic, BDD-based procedures.

We conclude our treatment of the symbolic approach with a simple example. Consider the two FSMs of Figure 7.45.

Example:

![Figure 7.45: Two non-equivalent FSMs](image)

These two machines are not equivalent. Their product is illustrated in Figure 7.46. Note in Figure 7.46 and other computer generated graphics, in the sequel, the optional state indicates the initial state of the FST.

Here the product states are given by the left-machine state (0 or 1) followed by an underscore and the right-machine state (0, 1, or 2). The error states of the product, $\{0, 0.1, 1.2\}$, are distinguished by box shapes rather than ellipses. Note the strings $x^1 = (10, 11, 10, 00)$ and $x^2 = (10, 11, 10, 01)$ produce the corresponding runs $\lambda^1 = \lambda^2 = (0.1, 1.1, 0.2, 1.0, 1.0)$, which have the output strings $\sigma^1 = \sigma^2 = (1, 1, 1, 0)$. Thus these two strings are distinguishing sequences for the pair of initial states of the two machines. Examination of the product FSM shows that these are the shortest error traces possible.

We now show how the above error traces can be obtained with symbolic, BDD-based computation. We illustrate only the first image computation in the application of Procedure BFS.FSM of Page 303. We first treat the 2-state FSM on the left of the figure, using the natural encoding with code bit $s_1$ to encode the two states, and the natural encoding for the four input symbols, which leads, as discussed in Section 7.1 of Page 255, to

$$t_1 = \delta^1_1 = \exists_x \exists_x' \lambda^1 \cdot [s_1 \cdot \lambda^1 + \lambda^2]$$

$$\lambda^1 = \delta^1_1.$$

We then treat the 3-state FSM on the right. We begin by encoding the states — we use the natural binary encoding of the digits, so that states 0, 1, 2 of the left FSM are encoded as $(s_1, s_2) = (00, 01, 10)$. Since two code bits are required we must have two latches, that is, one next state transition function for each code bit. We then construct a truth table (not shown) to realize the three functions $\delta^1_1(s_1, s_2, x_1, x_2)$, $\delta^2_1(s_2, s_3, x_1, x_2)$, and $\lambda^1(s_2, s_3, x_1, x_2)$.

By collecting the terms of the truth table where functions evaluate to 1, we get, after some simplification, the following expressions for $\delta^1_1(s_1, x)$, $\delta^2_1(s_1, x)$ and $\lambda(s_1)$.

$$t_1 = \delta^1_1 = s_3 \cdot \lambda^1 + \lambda^2$$

$$t_2 = \delta^2_1 = \exists_x \exists_x' \lambda^1 + \lambda^2$$

$$\lambda = s_3 \cdot \lambda^1 + \lambda^2.$$

We then form the product machine transition relation as the conjunction of the transition relations for the two submachines, leading to

$$T(s, x, t) = (t_1 \equiv \delta^1_1) \cdot (t_2 \equiv \delta^2_1) \cdot (t_3 \equiv \delta^3_1).$$

The first From set is the $C(s) = \exists_x \exists_x' \exists_x''$, which is the characteristic function of the set consisting of only the initial state of the product machine. We now wish to compute the image of $C(s)$, using Equation 7.2. First we observe that

$$T(s, x, t) \cdot C(s) = T(s, x, t) \cdot \exists_x \exists_x' \exists_x'' =$$

$$t_1 \equiv \exists_x \exists_x' \lambda^1 \cdot (t_2 \equiv 0) \cdot (t_3 \equiv \exists_x \lambda^2).$$

Since this conjunction evaluates to 1 for just one s-minterm $(\lambda^1 \lambda^2)$, it should be clear that

$$\exists_x(T(s, x, t) \cdot C(s)) = (t_1 \equiv \lambda^1) \cdot (t_2 \equiv 0) \cdot (t_3 \equiv \lambda^2) \cdot (t_4 \equiv \lambda^2).$$

![Figure 7.46: Product of the Two FSMs of Figure 7.45](image)
Chapter 7. Models of Sequential Systems

7.12 Summary

In this chapter we have briefly reviewed some important models that have been used to characterize sequential digital systems. These models were illustrated in Figure 7.32:

- **Finite State Transition Structures**, which we have called FSTs — In a given state, FSTs receive an input symbol, and make a transition to a new state;
- **Finite Automata**, which we shall call FAs — FAs are FSTs which also take notice when a favorable state (called *accepting*) is entered;
- **Finite State Machines**, which we shall call FSMs — FSMs are FSTs which emit a specified output symbol when they make a transition.

The treatment has emphasized the role of the underlying FSTs upon which automata and state machines are based.

We have presented in some detail the subject of state equivalence, beginning by defining the concept of "<em>k</em>-equivalence". Two states <em>s</em> and <em>t</em> of an FSM are <em>k</em>-equivalent if there exists no distinguishing input sequence of length <em>k</em>. If <em>s</em> and <em>t</em> are not <em>k</em>-equivalent, they are <em>k</em>-distinguishable. This means that when the length-<em>k</em> distinguishing sequence is applied, the <em>k</em>th output obtained when the machine is started in state <em>s</em>, is different than it would be if started in state <em>t</em>.

If there are <em>n</em> states in an FSM, two states were shown to be equivalent if and only if they are <em>n</em> − 1 equivalent. This has led to the partition/refinement algorithm for state minimization of an FSM, which first identifies all pairs of equivalent states, and then collapses sets of pairwise equivalent states into single states which are "representatives" of their equivalence classes.

By the definition of product machine, the concept of state equivalence was then used to define the behavioral equivalence of two FSMs. The bottom line is (roughly speaking) that two FSMs are equivalent if their initial states are equivalent.

We concluded by describing the revolutionary effect that BDDs have had on the nascent field of FSM equivalence checking in particular, and on Formal Verification in general.

7.13 Problems

1. Give the flow table which corresponds to the STG of Figure 7.11.

   **Solution.**

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>E/0</td>
</tr>
<tr>
<td>B</td>
<td>D/0</td>
</tr>
<tr>
<td>C</td>
<td>E/0</td>
</tr>
<tr>
<td>D</td>
<td>B/0</td>
</tr>
<tr>
<td>E</td>
<td>C/0</td>
</tr>
<tr>
<td>F</td>
<td>B/0</td>
</tr>
</tbody>
</table>