Relating State Machines: Equivalence

Garlan Handout 9 8 October 1997

So far we have seen what a state machine is and how it can be used to model concepts like system behavior, input actions (or events) with arguments, output actions (or events) with results, and nondeterminism. We have even hinted at how a state machine would be an appropriate model of an abstract data type, and hence one of the bases for object-oriented programming.

We have also seen that given a state machine we can reason about some of its properties, most importantly, invariants.

In this and the next handout we consider the following question: "Given two state machines, how are they related?" This handout discusses the relationship of *equivalence* between two state machines. The next handout discusses the problem of whether one machine *satisfies* (in some sense) another. We won't be spending any time in this course defining different notions of equivalence or showing two state machines equivalent. Hence this handout is short and to be read for edification.

1 Why Would You Care About Equivalence?

Given two machines, M_1 and M_2, why would you care to know whether they are "equivalent"? The most compelling answer is that if you know M_2 is equivalent to M_1 then you know that you can *substitute* M_2 for M_1 without changing the overall system in which you do the substitution. This substitution principle is extremely important. Many areas in Computer Science rely on this principle. The most obvious example is in using a compiler. A compiler transforms a program into (we hope) a more efficient one. The source and target programs had better compute the same answer given the same inputs; in this sense the two programs are equivalent, and the target program is an efficient substitute for the source program. Similarly, software engineering relies on the notion of modularity where you can replace one component for another without affecting the rest of the system; functional programming, equational reasoning, and rewrite rule theory all rely on the principle of substituting equals for equals, sometimes called *referential transparency*; even caching protocols in a multiprocessor or distributed system aim at keeping replicas equivalent (the "cache consistency" problem) so that the existence of multiple versions is hidden from the user.

Efficiency is one reason you might want to substitute one machine for another. M_2 may have fewer states or fewer state variables or fewer state transitions than M_1. In the real world, there are other good reasons: cost, user-friendliness, maintainability, portability, or just esthetics.

2 What Does Equivalence Mean?

The hard question is "When are two machines equivalent?" The answer is dependent on the context in which you intend to do the substitution. Intuitively, we strive for some notion of *observational equivalence* such that I as an external observer (the context) "cannot tell the difference" between whether M_1 is being used or M_2. If I can't perceive a difference, then for all intents and purposes they are equivalent.

Answering this question is a subject of much debate and decades of research, mostly in the theoretical community on models of concurrent systems. Let me give you an idea of what people might debate about.

©1995 Jeannette M. Wing

At first, you might think it's easy to simply define equivalence in terms of input-output behavior. If I feed the two machines the same input do I get the same output? I ask this question for all possible inputs. If I always get the same output for the same input, they are equivalent; if not, they are different. This notion of equivalence takes a pretty narrow view of what a state machine is. It essentially says that a state machine represents a function and determining equivalence between two state machines boils down to the problem of determining whether they compute the same function.

However, a software system is not nearly as simple. For example, software systems, and hence their state machine models, often have intermediate observable effects on the environment (their context). It does not suffice to just consider the final states of the machines. Thus, many prefer to take the broader view that two machines are equivalent if their behaviors (i.e., the sets of traces) are the same. To determine whether two sets of traces are the same we're going to need a way to determine whether two traces are the same. To determine whether two traces are the same we're going to need a way to determine whether two actions (or states) are the same. To determine whether two actions (or states) are the same we may need to ignore some actions or states variables (e.g., internal ones) and not others. So, you can see that it's not so easy to decide whether two state machines are equivalent; each substructure of a state machine introduces another place for differing opinions.

For example, depending on whether you take an event-based or state-based view of what a trace is you could come up with different answers given two machines. Consider the Light example with *off* and *on* states and the *flick* action. A different Light example with three states, e.g., red, amber, and green, but with the same action set $\{flick\}$ (think of a lever with three positions rather than two) will have the same behavior if you take an event-based view of traces, but different behavior if you take a state-based view.

Even for the same view of traces, there are different notions of equivalence. Suppose in determining whether two behaviors are the same, we need to determine whether two state-based traces are the same. Would you view a trace with "stuttering" states as equivalent to one without? Consider the Register example that has *read* and *write* actions where after doing two *read*'s in a row, I remain in the same state:

$$\langle x = 0, write(1)/ok(), x = 1, read()/ok(1), x = 1, read()/ok(1), x = 1, write(5)/ok(), x = 5\rangle$$

A state-based trace of this execution is:

$$\langle x = 0, x = 1, x = 1, x = 1, x = 5\rangle$$

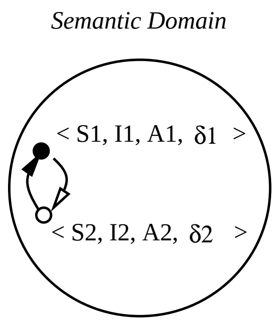
Is it equivalent to the following?

$$\langle x = 0, x = 1, x = 5\rangle$$

It may be even harder to answer this question if your model includes infinite traces where a trace might end with an infinite number of stuttering states.

3 Showing Equivalence

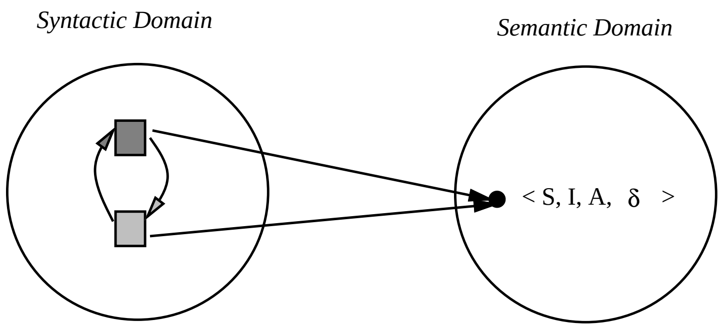
How do you show whether two machines are equivalent or not? There are two general approaches you can take. First, you could work within just the semantic domain



Show one quadruple (semantic element)
is the same as the other.

and show that the two semantic entities, in this case quadruples, are equivalent. If your notion of equivalence is not defined directly in terms of quadruples, but rather behavior sets then your semantic domain would be behavior sets and you'd have to show equivalence between two behavior sets, which might require showing equivalence between traces, etc..

Or, second, you could work within the syntactic domain



Show one description (syntactic element)
is the same as the other.

and show that the two syntactic descriptions, e.g., pre/post-condition specifications, Z specifications, or CSP programs, are equivalent in some sense. For example, for two different pre/post-condition specifications or two different Z specifications you might show that each of the predicates of one implies the corresponding predicate of the other and vice versa. For CSP programs, you might use properties of process algebras to show equivalence.[1] Since the two syntactic descriptions are "the same," then they denote the same semantic entity, in this a case a quadruple representing a state machine; thus, they must describe the same state machine.

Another way to categorize how we might show the equivalence between two machines is as follows:

- Show equivalence from first principles using mathematical logic, theories of sets and sequences, induction, etc. Since equality on sets, sequences, integers, booleans, and other primitive types is mathematically well-defined, if you beat everything down to these primitive types, then you have a way of showing equivalence. This technique could be used if you wanted to show the equivalence of two quadruples.

- Show one "simulates" the other and vice versa. Anything I can do in one machine has some equivalent action or sequence of actions in the other. For example, if you wanted to show that one behavior set is the same as the other, then you might use this approach.

[1] You are not expected to understand this statement until later in the course.

- Transform one into the other. For example, if you wanted to show that one state machine description is “the same” as the other, you might use this approach. Showing that compiled code is “correct” amounts to showing that the transformed code is “the same” as the original code.