CMU 17-651 Models of Software Systems Fall 2016

Reasoning About State Machines

Garlan **Handout 6** 25 September 1995

Given a state machine model of a system, we can do some formal reasoning about properties of the model. It's important to remember that we are proving some property about the *model of the system*, not the *system* itself. If the model is "incorrect" then we may not be able to prove anything useful about the system. Worse, if the model is "incorrect" then we may be able to prove something that has no correspondence to the real system. However, we hope that we've modeled our system properly so that whatever we prove about our model is true of the system being modeled.

But then, why not just reason about the system itself directly? One reason is that it's often impossible to reason about the system itself because it's too large, too complex, or too unwieldy. Another is that we may be interested in one aspect of the system and want to abstract from the irrelevant aspects. Another is that we may not actually have a real system; our model could simply be a high-level design of a system we might build and we want to do some reasoning about our design before spending the dollars building the real thing. Another is that it may be impossible to get our hands on the system (maybe it's proprietary). Another is that it may be impossible for us to run the system to check for the property because of its safety-critical side-effects (like setting off a bomb). So, in some cases, the best we can do is reason about a model of the system, and not the system itself.

In this handout I'll discuss a few kinds of properties we might want to reason about a state machine model.[1] The most important of these is *invariant properties*, properties that are true of every reachable state in the system.

1 Invariants

An *invariant* is a predicate that is true in all states. In the context of state machines, we usually care that an invariant is true in all *reachable* states. The statement of an invariant, θ, in full generality looks like:

$$\forall e : executions \bullet \forall s : S \bullet s \ \mathbf{in}\ e \Rightarrow \theta(s)$$

where θ is a predicate over variables in s and **in** is a predicate that says whether a state is in an execution. Normally the universal quantification over all executions and the condition that s be in e is omitted (it is understood):

$$\forall s : S \bullet \theta(s)$$

Sometimes we also omit the universal quantification over s as well because it's also understood:

$$\theta(s)$$

For example, here is an invariant for the Counter example given in State Machine Variations handout:

[1]In this handout, I'm going to be even less precise in my notation and use of formalism for the sake of clarity of presentation.

$$s(x) \geq 0$$

which says that in all states, x's value is greater than or equal to 0. We know this is true because initially x's value is 0 and because the *inc* action always increases x's value by 1. Since *inc* is Counter's only action, there's no other way to change x.

1.1 Proving an Invariant

How do we show that a predicate is an invariant? There are lots of techniques. If the state machine is finite, you can do an exhaustive case analysis and show that it holds for every state. This technique is fine if there are a small number of states or if you have a tool called a *model checker* handy (you will in your Analysis core course next semester).

If your state machine is infinite, you must resort to something else. I'm going to sketch out three techniques. They can all also be used if your state machine is finite. Technique C is the one most often used in practice.

A. Use induction over states in executions.

When you have to reason about an infinite domain, the technique that should spring to mind is *induction*. Induction is especially appropriate when there is a natural, often recursive, structure to your domain. Since what we want to prove is that a property is true of every state of every execution of the state machine, then I can induct over the states in the sequence of states of every execution. Recall that an execution looks like:

$$\langle s_0, a_1, s_1, a_2, \ldots, s_{i-1}, a_i, s_i, \ldots \rangle$$

Then to prove a property, θ, is invariant requires that for every execution:

1. Base Case: I show it holds in the initial state s_0, and
2. Inductive Step: I assume it holds in state s_{i-1} and show that it holds in state s_i.

B. Show that your state space predicate implies the invariant.

A second technique is that if you're lucky the predicate that you've used to define your state space is stronger than the invariant you're trying to prove. So regardless of whether a state is reachable or not, you can prove the invariant holds:

$$P \Rightarrow \theta$$

where P is the predicate describing your set of states. If it's true of every state, then certainly it's true of every reachable state.

For example, recall in the Counter example, the predicate P is simply $s(x) \geq 0$. Hence we can trivially show the invariant property holds:

$$s(x) \geq 0 \Rightarrow s(x) \geq 0$$

C. Use a proof rule using pre-/post-condition specifications.

Technique A requires that we reason in terms of first principles— using structural induction (over states in executions)—which can sometimes be cumbersome. And, usually, of course, we are not so lucky as in Technique B. Technique C is an alternative inductive proof strategy that is usually more manageable than Technique A: we do a case analysis of all actions, which indirectly proves the same thing as in Technique A.

At the bottom of p. 1 when I argued that the Counter's invariant holds, we need to show that the invariant holds in each initial state and then for each action[2] show that if the invariant holds in its pre-state, it holds in the post-state. In general, we have a proof rule that looks like:

[2]Here again, is another good reason to have only a *finite* set of actions.

$$\frac{\begin{array}{c}\forall s : I \bullet \theta(s) \\ \forall a : A \bullet \forall s, s' : S \bullet (s, a, s') \in \delta \Rightarrow ((\Phi(s) \wedge \theta(s) \wedge \Psi(s, s')) \Rightarrow \theta(s'))\end{array}}{\forall s : S \bullet \theta(s)}$$

where I is the set of initial states and A is the set of actions. Φ and Ψ are the pre- and post-conditions of a, respectively. Or, said in English:

1. Show that θ is true for each initial state.
2. For each action,
   - assume
     - the pre-condition Φ holds in the pre-state,
     - the invariant θ holds in the pre-state, and
     - the post-condition Ψ holds in the pre- and post-states, then
   - show
     - θ holds in the post-state.
3. Conclude that θ is an invariant.

The two main proof steps are sometimes called (1) *establishing the invariant* (true in initial states) and (2) *preserving the invariant* (assuming it's true in a pre-state and showing that each action's the post-state preserves it).

What's the rationale for this proof rule? First, notice I care only about reachable states (that's why the δ appears above). Second, consider any execution of my state machine:

$$\langle s_0, a_1, s_1, a_2, \ldots, s_{i-1}, a_i, s_i, \ldots \rangle$$

As in Technique A I need to make sure that θ holds in s_0 (establishing the invariant). Also, for any pair of successive states, (s_{i-1}, s_i), in the execution, if I assume θ holds in s_{i-1} then I need to show it holds in s_i. Since the only way I can get from any s_{i-1} to the next state s_i is by one of the actions a in A, then if I show the invariant if preserved for each a, I've shown for each reachable state the invariant holds.

1.2 OddCounter

Let OddCounter be the state machine:

More precisely,

OddCounter = (
$\{s : \{x\} \rightarrow int\}$,
$\{s : \{x\} \rightarrow int | s(x) = 1\}$,
$\{inc(i : int)\}$,
$\delta =$

```
inc(i: int)
    pre $i$ is even
    post $x' = x + i$
```

).

The invariant I want to prove is that OddCounter's state always holds an odd integer:

$$\theta \equiv x \text{ is odd.}$$

Intuitively we see this is true because

1. It's true of the initial state (1 is an odd integer).
2. For the action *inc*:
   - I assume:
     - i is even (i.e., the pre-condition holds in the pre-state),
     - x is odd (i.e., the invariant holds in the pre-state), and
     - $x' = x + i$ (i.e., the post-condition holds in the pre- and post-states)
   - I need to show that x' is odd.
     - This is true since from facts about numbers I know an odd number (x) plus an even number (i) is always odd.

1.3 Fat Sets

This example has two purposes. One is to give another example of an invariant. The other is to hint at how state machines are appropriate models of abstract data types, as you might implement in your favorite object-oriented language.

Here is a description of a FatSet abstract data type modeled as a state machine:

- $S = \{s : \{t\} \rightarrow set[int]\}$. The state variable, t, maps to a set of integers.
- $I = \{s : \{t\} \rightarrow set[int] | \#s(t) = 1\}$. The set of initial states is the set of states that map t to a singleton set. Notice that there are an infinite number of initial states.
- A = { *union(u: set[int])/ok(), card()/ok(int)* }
- $\delta =$

```
union(u: set[int])/ok()
        pre $u \neq \emptyset$
        post $t' = t \cup u$

card()/ok(int)
        pre $true$
        post $result = \#t \wedge t' = t$
```