Lecture 16 - State Machines and Invariants
6.042 - April 10, 2003

1 State machines

State machines show up all over computer science: inside microchips, in the guts of compilers, in the analysis of protocols, and in the study of the mathematical theory of computation.

1.1 Basic definitions

Mathematically speaking, a state machine is a set of states together with a binary relation that describes how the machine moves from one state to another. In addition, there are some start states, which the machine may be in when it begins executing.

Definition 1 *A* state machine *has three parts:*

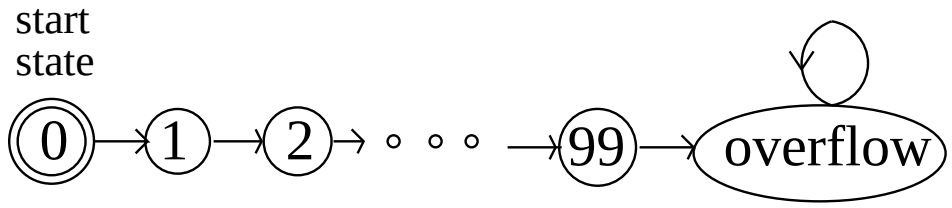
1. *A nonempty set, Q, whose elements are called* states.
2. *A nonempty subset $Q_0 \subseteq Q$, called the set of* start states.
3. *A binary relation, δ, on Q, called the* transition relation.

You may have seen variations of this definition in a course on digital logic, compilers, or the theory of computation. In those courses, the machines usually have only a *finite* number of states. Also, the edges in the state graphs are usually labelled with input or output tokens. We won't need state machines take take input or produce output for the applications that we'll consider here.

Another view is that a state machine is really nothing more than a digraph whose vertices are the states and whose edges are determined by the transition relation. Reflecting this view, we often write $q \to q'$ as alternative notation for the assertion that $(q, q') \in \delta$. When we draw a state machine as a digraph, we'll double-circle the vertices corresponding to start states.

1.2 Example: A Counter

Here is a simple example of a state machine, depicted as a directed graph.



Formally, this corresponds to the following state machine:

- $Q = \{0, 1, 2, \ldots, 99, \texttt{overflow}\}$
- $Q_0 = \{0\}$
- $\delta = \{(0,1), (1,2), (2,3), \ldots, (99, \texttt{overflow}), (\texttt{overflow}, \texttt{overflow})\}$. Note that the overflow state has a self-loop, a transition to itself.

Unlike graphs, which just sit there, state machines *execute*. We'll make the notion of execution formal in a minute. For now, imagine a ticking clock. At every moment, the state machine has a *current state*. Initially, this is one of the start states. With each tick of the clock, the machine transitions to a new state. In particular, if the machine is currently in state $q \in Q$, then it next transitions to some state $q' \in Q$ such that $q \to q'$ is a valid transition; that is, $(q, q') \in \delta$.

For example, the machine above begins in state zero. With each tick of the clock, it moves from state n to state $n + 1$. When it reaches the overflow state, it remains there forever. Thus, this is a simple, bounded counter.

Our formal definition of the execution of a state machine will make no mention of clocks and current states. However, it is often convenient easier to think about state machines in those terms.

One can construct an unbounded counter similar to the 0 to 99 counter above. But that would have an infinite state set, yielding an infinite digraph. This is harder to draw.

1.3 Example: Die Hard

In the movie Die Hard 3, Bruce Willis and Samuel Jackson are coerced by a homicidal maniac into trying to disarm a bomb on a weight-sensitive platform near a fountain. To disarm the bomb, they need to quickly measure out exactly four gallons of water and place it on the platform. They have two empty jugs, one that holds three gallons and one that holds five gallons, and an unlimited supply of water from the fountain. Their only options are to fill a jug to the top, empty a jug completely, or pour water from one jug to the other until one is empty or the other is full. They do succeed in measuring out the four gallons while carefully obeying these rules. You can figure out how (or go see the movie or).

The Die Hard 3 situation can be formalized as a state machine as well.

- $Q = \{(b, l) \in \mathbb{R}^2 \mid 0 \leq b \leq 5, 0 \leq l \leq 3\}$. Note that b and l are arbitrary real numbers, not necessarily integers. After all, Bruce could scoop any unmeasured amount of water into a bucket.
- $Q_0 = \{(0, 0)\}$ (because both jugs start empty)
- δ has several kinds of transitions:
  1. Fill the little jug: $(b, l) \to (b, 3)$ for $l < 3$.
  2. Fill the big jug: $(b, l) \to (5, l)$ for $b < 5$.
  3. Empty the little jug: $(b, l) \to (b, 0)$ for $l > 0$.
  4. Empty the big jug: $(b, l) \to (0, l)$ for $b > 0$.
  5. Pour from the little jug into the big jug: for $l > 0$,
  $$(b, l) \to \begin{cases} (b + l, 0) & \text{if } b + l \leq 5, \\ (5, l - (5 - b)) & \text{otherwise.} \end{cases}$$
  6. Pour from big jug into little jug: for $b > 0$,
  $$(b, l) \to \begin{cases} (0, b + l) & \text{if } b + l \leq 3, \\ (b - (3 - l), 3) & \text{otherwise.} \end{cases}$$

Note that in contrast to the 99-counter state machine, there is more than one possible transition out of states in the Die Hard machine. Thus, the state of the machine after n ticks of the clock is not completely determined.

A machine is called *deterministic* if its execution behavior is uniquely determined: there is only one start state and there is at most one transition out of every state. Otherwise, it is *nondeterministic*. In these terms, the Die Hard machine is nondeterministic, and the counter machines are deterministic.

1.4 Executions of State Machines

The Die Hard 3 machine models every possible way of pouring water among the jugs according to the rules. Die Hard properties that we want to verify can now be expressed and proved using the state machine model. For example, Bruce will disarm the bomb if he can *reach* some state of the form $(4, l)$.

In graph language, a (possibly infinite) path through the state machine graph beginning at a start state corresponds to a possible system behavior or process execution. A state is reachable if there is a path to it starting from one of the start states.

Definition 2 *An* execution *is a (possibly infinite) sequence $q_0, q_1, \ldots$ such that $q_0 \in Q_0$, and $\forall i \geq 0\, (q_i, q_{i+1}) \in \delta$. A state is* reachable *if appears in some execution.*

For example, we said that Bruce and Samuel successfully disarmed the bomb in Die Hard 3. In particular, the state (4,3) is reachable; in particular, it appears in the execution described below.

action	state
start	(0,0),
fill the big jug	(5,0),
pour from big to little	(2,3),
empty the little	(2,0),
pour from big into little	(0,2),
fill the big jug,	(5,2),
pour from big into little	(4,3).

1.5 Die Hard Once and For All

Bruce is getting burned out on dying hard, and according to rumor, is contemplating a sequel called *Die Once and For All*. In this film, he will face a more devious maniac who provides him with the same three gallon jug, but with a *nine* gallon jug instead of the five gallon one. The water-pouring rules are the same. He must quickly measure out exactly four gallons or the bomb will go off.

This time the task is impossible— whether done quickly or slowly. We can prove this without much difficulty. Namely, we'll prove that it is impossible, by any sequence of moves, to get exactly four gallons of water into the large jug.

A sequence of moves is constructed one move at a time. This suggests a general approach for proofs about sequential processes: to show that some condition always holds during the executions of a process, use induction on the number, n, of steps or operations in the executions. For Die Hard, we can let n be the number of times water is poured.

All will be well if we can prove that neither jug contains four gallons after n steps for all $n \geq 0$. This is already a statement about n, and so it could potentially serve as an induction hypothesis. Let's try lunging into a proof with it:

Theorem 1 *Bruce dies once and for all.*

Let $P(n)$ be the predicate that neither jug contains four gallons of water after n steps. We'll try to prove $\forall n\, P(n)$ using induction hypothesis $P(n)$.

In the base case, $P(0)$ holds because both jugs are initially empty. In the inductive step, we assume that neither jug has four gallons after n steps and try to prove that neither jug has four gallons after $n + 1$ steps.

Now we are stuck; the proof cannot be completed. The fact that neither jug contains four gallons of water after n steps is not sufficient to prove that neither jug can contain four gallons after $n+1$ steps. For example, after n steps each jug might hold two gallons of water. Pouring all water in the three-gallon jug into the nine-gallon jug would produce four gallons on the $(n+1)$-st step.

What to do? We use the familiar strategy of strengthening the induction hypothesis. Some experimentation suggests strengthening $P(n)$ to be the predicate that after n steps, the number of gallons of water in each jug *is a multiple of three*. This is a stronger predicate: if the number of gallons of water in each jug is a multiple of three, then neither jug contains four gallons of water. This strengthened induction hypothesis does lead to a correct proof of the theorem.

However, to be precise about this proof, we need to use the state machine model. The states and transition relation are the same as for the Die Hard 3 machine, with all occurrences of "5" replaced by "9." We could do an induction proof, but here we will use a proof pattern that corresponds to the Invariant Theorem.

2 Reachability and Invariants

An induction proof about the Once and For All machine would follow a general proof pattern that could be used to any analyze state machine behavior. Namely, we show that the integer-multiple-of-3 property holds at the start state and remained *invariant* under state transitions. So it must hold at all reachable states. In particular, since no state of the form $(4, l)$ satisfies the invariant, no such a state can be reachable.

Definition 3 *An* invariant *for a state machine is a predicate, P, on states, such that whenever $P(q)$ is true of a state, q, and $q \to r$ for some state, r, then $P(r)$ holds.*

Now we can reformulate the Induction Axiom specially for state machines:

Theorem 2 (Invariant Theorem) *Let P be an invariant predicate for a state machine. If P holds for all start states, then P holds for all reachable states.*

The truth of the Invariant Theorem is as obvious as the truth of the Induction Axiom. We could prove it, of course, by induction on the length of finite executions, but we won't bother.

Now, we will prove Theorem 1 using the predicate $P(q)$ corresponding to the number of gallons in each jug in state q is an integer multiple of 3. We will first prove that P is an invariant.

Lemma 3 *P is an invariant.*

Proof. Assume $P(b, l)$ is true, i.e., b and l are integer multiples of 3. We have to show that for every state reachable by a single transition from $P(b, l)$, the amounts in the jugs are still integer multiples of 3.

The proof is by cases, according to which transition rule is used in the next step. For example, using the "fill the little jug" rule for the $(n+1)$-st transition, we arrive at state $(b, 3)$. We already know that b is an integer multiple of 3, and of course 3 is an integer multiple of 3, so the new state $(b, 3)$ has integer multiples of 3 gallons in each jug, as required. Another example is when the transition rule used is "pour from big jug into little jug" for the subcase that $b + l > 3$. Then the $(n + 1)$-st state is $(b - (3 - l), 3)$. But since b and l are integer multiples of 3, so is $b-(3-l)$. So in this case too, both jugs will contain an integer multiple of 3 gallons.

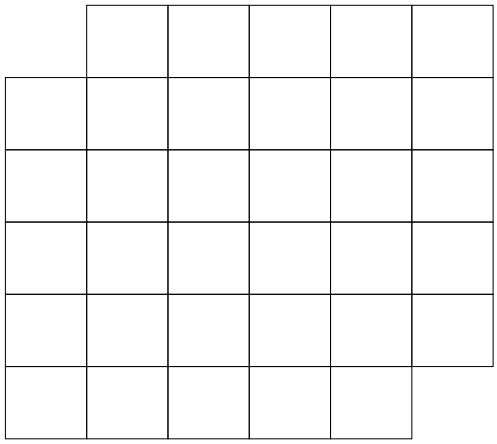
We won't bother to crank out the remaining cases, which can all be checked with equal ease. This means that P is an invariant. □

Now, here's the short (!) proof for Theorem 1.

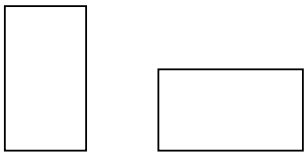
Proof. P is an invariant. $P(0, 0)$ is true since 0 is an integer multiple of 3. (It can be expressed as $0 \cdot 3$.) By the Invariant Theorem, state $(4, l)$ cannot be a reachable state since $P(4, l)$ is false. Therefore, Bruce dies! □

3 Mutilated Grids

Suppose we have a 6×6 "mutilated" grid shown below. Can we tile it with Dominoes oriented in two ways?



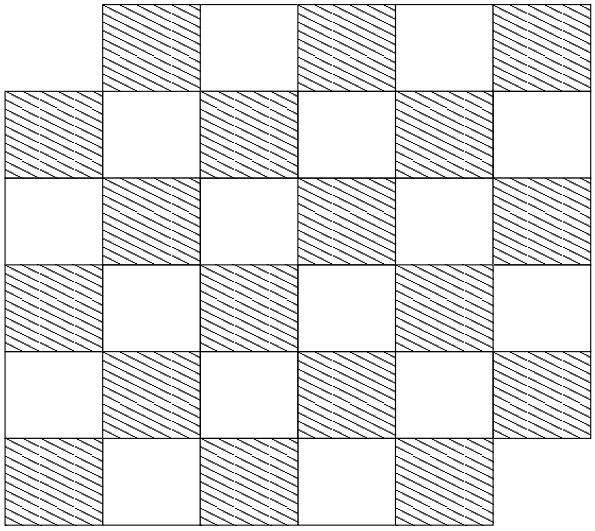
(a)



(b)

It turns out we cannot. How can we prove this?

Look at the mutilated grid shown below, with squares alternately colored black and white. It is immediately clear that each domino covers both a black and a white square. We can use this observation to prove that we cannot tile the mutilated 6×6 grid. In fact, we can prove a more general result.



First, note that a $n \times n$ ordinary grid or a $m \times m$ mutilated grid cannot be tiled with Dominoes when m is odd for the simple reason that there are an odd number of squares in these grids, since m^2 and $m^2 - 2$ are both odd when m is odd. Let us consider a $2n \times 2n$ mutilated grid. Our grid has $n = 3$.

We will model the tiling process as a state machine. The states correspond to partially-tiled grids. The state machine for the tiling process is given below.

- $Q = \{(x, y) \in \mathbb{N}^2 \mid 0 \leq x \leq 2n^2 - 2, 0 \leq y \leq 2n^2\}$, where x is the number of white squares tiled (i.e., covered) by a Domino and y is the number of tiled black squares.
- $Q_0 = \{(0, 0)\}$ (because no squares are tiled initially).
- δ has a very simple structure: $(x, y) \rightarrow (x+1, y+1)$, for $0 \leq x < 2n^2 - 2$, $0 \leq y < 2n^2$.

Let P be the predicate that the number of white squares covered is equal to the number of black squares covered. We will prove that P is an invariant.

Lemma 4 *P is an invariant.*

Proof. Let x be the number of black squares tiled and y be the number of white squares tiled. Clearly, if $x = y$, then $x + 1$ equals $y + 1$, so P is an invariant. □

Theorem 5 *The $2n \times 2n$ mutilated grid cannot be tiled.*

Proof. Use P as the invariant. Clearly, for starting state $(0, 0)$, the invariant holds. Therefore, by the Invariant theorem, the state $(2n^2 - 2, 2n^2)$ cannot be reached, since it does not satisfy P. This means that we cannot tile the grid. □