Lecture Notes

The topic for the next few weeks will be the application of induction and other proof techniques to the design and analysis of algorithms and systems.

This week we will focus on the problem of proving that an algorithm is correct; that is, that the behavior really matches some specification. Proving the correctness of software is becoming increasingly important as critical tasks such a flying airplanes are delegated to computers.

We'll introduce many of the tools needed for proofs of correctness while working through one example problem, and then we'll apply these tools to a second example.

1 Die Hard

In the movie *Die Hard 3*, Bruce Willis and Samuel Jackson need exactly 4 gallons of water to disarm a bomb. They have a reservoir of water and two empty jugs, one that holds 3 gallons and one that holds 5 gallons. 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. Following their script carefully, they succeed!

However, according to rumor, there will be a sequel, *Die Hard 4: Die Once and For All*. In this film, Bruce will face a even more devious villain. This time he will have a 3-gallon jug and a 6-gallon jug. The water-pouring rules are the same. Bruce must produce 4 gallons of water in 30 seconds or the bomb goes off.

A little experimentation suggests that the task is impossible. But we'll all rest a little easier if we can mathematically prove that there can be no *Die Hard 5*.

1.1 Attempted Proof by Induction

Following *our* script carefully, let's try a proof by induction. The first difficulty is that induction is appropriate for proving that a statement holds for all values of a natural variable, but no such variable conveniently appears in the problem statement.

In problems where we want to show that some condition always holds, a natural choice is to let n be a number of steps or operations. In this case, we can let n be the number of times water is poured. All will be well if we can prove that neither jug contains 4 gallons after n steps for all $n \geq 0$. This statement contains a natural-valued variable, and so can serve as an induction hypothesis.

The condition that we want to prove always holds, in this case "neither jug contains 4 gallons", is called an invariant. This term will be defined formally below. Let's just try lunging into a proof.

Theorem 1.1 Bruce dies.

Proof. (first attempt) The proof is by induction. Let $P(n)$ be the predicate that neither jug contains 4 gallons of water after n steps. In the base case, $P(0)$ holds because both jugs are initially empty. In the inductive step, assume that neither jug has 4 gallons after n steps to prove that neither jug has 4 gallons after $n+1$ steps...

Now we are stuck; the proof can not be completed. The fact that neither jug contains 4 gallons of water after n steps is not sufficient to prove that neither jug can contain 4 gallons after $n+1$ steps. For example, after n steps each jug might hold 2 gallons of water. Pouring all water in the 3-gallon jug into the 6-gallon jug would produce 4 gallons after $n+1$ steps.

Our script advises that we strengthen the induction hypothesis. A "stronger" induction hypothesis is one that implies the old hypothesis.

Some more experimentation suggests letting $P(n)$ be the predicate that after n steps, the number of gallons of water in each jug is a multiple of 3. This is a stronger predicate; if the number of gallons of water in each jug is a multiple of 3, then neither jug contains 4 gallons of water. With this new predicate the invariant is, "the number of gallons of water in each jug is a multiple of 3".

Though this induction hypothesis leads to a proof of the theorem, we'll first introduce some formal machinery that will be useful for this and other problems.

1.2 State Machines

Definition 1.2 A *state machine* is a triple (Q, Q_0, δ) where:

1. Q is a non-empty set of *states* (possibly infinite)
2. $Q_0 \subseteq Q$ is a non-empty set of *start states* (possibly infinite)
3. $\delta \subseteq Q \times Q$ is a set of allowed transitions between states

Often we will use the notation $q \rightarrow q'$ as a shorthand for the transition (q, q'). A good way to think about a state machine is as a directed graph where the states Q are vertices, and the allowed transitions δ are directed edges.

(This is similar to the definition of a finite state machine that you will encounter in theory of computation and hardware classes. There are at least two important differences, however. First, we do not require that the number of states be finite. Second, here there is no notion of the "input" or "output" of a state machine.)

As an example, a state machine (Q, Q_0, δ) can be used to model a counter that starts at 0 and overflows at 100:

$$\begin{aligned}
Q &= \{0, 1, 2, \ldots, 99, overflow\} \\
Q_0 &= \{0\} \\
\delta &= \{0 \rightarrow 1, 1 \rightarrow 2, \ldots, 98 \rightarrow 99, 99 \rightarrow overflow, overflow \rightarrow overflow\}
\end{aligned}$$

Thinking about this state machine as a directed graph gives Figure 1.

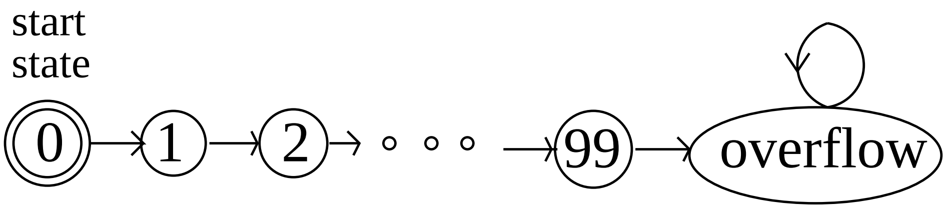

Figure 1: *This is a representation of the counter state machine as a directed graph.*

If the counter had no overflow or limit, then there would be an infinite number of states, one for every natural number. In this case, the associated directed graph would have an infinite number of vertices.

The reason for introducing state machines is to help model every possible behavior of some procedure. This, in turn, gives a formal basis for proving things about the procedure. The notion of a "possible behavior" of a procedure is captured by the following definition.

Definition 1.3 An *execution* of a state machine is a sequence (possibly infinite) $q_0, q_1, \ldots$ such that $q_0 \in Q_0$ and for all $i \geq 0$, $(q_i, q_{i+1}) \in \delta$.

In some state machines, there are many transitions out of each state. This is not true in the counter example; in this case, there is only one outgoing transition for each state. This makes the counter a deterministic state machine. More formally:

Definition 1.4 A state machine is *deterministic* if there is a single start state ($|Q_0| = 1$) and for all states q, there exists at most one state q' such that $(q, q') \in \delta$.

A state machine that is not deterministic is called *nondeterministic*. Though non-deterministic state machines are usually more interesting, even the analysis of a deterministic state machine can be tricky. We'll see an example of this next lecture.

1.3 Die Harder

The Die Hard problem can be modeled with a nondeterministic state machine. To do this, we must define the three components of a state machine, Q, Q_0, and δ.

The first task is to define the set of states, Q. Every state should be associated with the amount of water in the jugs:

$$Q = \{(a, b) \mid 0 \leq a \leq 3,\ 0 \leq b \leq 6,\ a, b \in \mathcal{R}\}$$

Here $\mathcal{R}$ denotes the set of real numbers. The variable a is the amount of water in the 3-gallon jug, and b is the amount of water in the 6-gallon jug. Each pair (a, b) is a state. Of course, this defines an enormous set Q of states! Note that we have not even assumed that jugs always contain an integer number of gallons of water. This fact may seem obvious from the choice of start state and the rules for pouring water, but there is no reason to assume it in advance. After all, a jug can contain a half gallon of water.

The second job is to define the set of start states, Q_0. This is easy; both jugs start empty and so $Q_0 = \{(0,0)\}$.

The most complicated task is to define the set of allowed transitions δ. Recall that Bruce had three basic options: 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. We can further break these down into six operations. For each operation, we can define a set of transitions as shown in the table below where $0 \leq a \leq 3$ and $0 \leq b \leq 6$. The set δ is the union of the six sets defined below.

1. Fill the 3-gallon jug: $(a,b) \rightarrow (3,b)$
2. Fill the 6-gallon jug: $(a,b) \rightarrow (a,6)$
3. Empty the 3-gallon jug: $(a,b) \rightarrow (0,b))$
4. Empty the 6-gallon jug: $(a,b) \rightarrow (a,0)$
5. Pour 3-gal into 6-gal: $(a,b) \rightarrow (max(0, a+b-6), min(a+b, 6))$
6. Pour 6-gal into 3-gal: $(a,b) \rightarrow (min(a+b, 3), max(0, a+b-3))$

Now we have defined Q, Q_0, and δ, so we have a state machine. The state machine is nondeterministic, since there are 6 allowed transitions from each state, including self-loops.

1.4 Reachabilty and Invariants

Recall the purpose for building a state machine: to help model every possible behavior of a procedure as a basis for proving useful facts about the procedure. For the Die Hard example, we have now successfully modeled every possible way of pouring water between jugs. What remains is see how to prove a useful facts, such as "Bruce is toast".

Once a procedure is modeled with a state machine, a natural question is which states can be reached through allowable transitions from a start state. For example, in the Die Hard problem Bruce lives if he can reach any state of the form $(a, 4)$. More formally:

Definition 1.5 A state is *reachable* if it is contained in a finite-length execution.

For example, in the Die Hard example, state $(3, 0)$ can be reached from the start state by filling the 3-gallon jug. On the other hand, state $(a, 4)$ probably can not be reached from the start state for any a.

Often finding exactly which states are reachable is unnecessary; it may be sufficient just to prove that some statement is true for every reachable state. Such a statement is called an invariant.

Definition 1.6 An *invariant* is a predicate $P(q)$ that is true for all reachable states q.

In the Die Hard example, we can let $P((a, b))$ be the predicate that a and b are multiples of 3. (The double parenthesis in $P((a, b))$ just reflect that in this example each state is a pair of numbers in parentheses.) Our goal is then to prove that this predicate is an invariant. That will show that in every reachable state, the number of gallons of water in each jug is a multiple of 3. Since 4 is not a multiple of 3, Bruce dies. In general, proving invariants of state machines is a central task in proving the correctness of algorithms.

Here is the strategy for proving that a predicate is an invariant of a state machine. First, we show that the invariant holds for all of the start states. Next, we show that if the predicate holds for a particular state, then it holds for every state than can be reached in one transition. This strategy has the flavor of an induction proof. In fact, the theorem that justifies this strategy is proven by a induction on the length of an execution.

Theorem 1.7 (Invariant Theorem) Let $\mathcal{M} = (Q, Q_0, \delta)$ be a state machine and let P be a predicate on Q. If $P(q)$ is true for all $q \in Q_0$ and $P(q) \Rightarrow P(q')$ for all $(q, q') \in \delta$, then P is an invariant for $\mathcal{M}$.

The proof of this theorem is confusing because there is a predicate about predicates. In particular, $\hat{P}$ is the induction hypothesis, which is a statement about P, the predicate mentioned in the theorem.

Proof. The proof is by induction on the length of an execution. Let $\hat{P}(n)$ be the predicate that $P(q)$ is true for every state q contained in an execution of length n.

In the base case, $\hat{P}(1)$ is true because all executions of length 1 consist only of a start state, and $P(q)$ is true for all $q \in Q_0$.

In the inductive step, assume that $P(q)$ is true for every state q contained in an execution of length n in order to prove that $P(q')$ is true for every state q' contained in an execution of length $n + 1$. Consider a state q' contained in an execution of length $n + 1$. If q' is the first state in the execution, then q' is a start state and so $P(q')$ holds. Otherwise, let q be the state preceding q' in the execution. Then q is contained in an execution of length n, and so $P(q)$ is true by induction. Since $(q, q') \in \delta$, $P(q)$ implies that $P(q')$ is true.

We have shown that $P(q)$ is true for every state q contained in a finite length-execution, which is exactly the set of reachable states. Therefore, $P(q)$ is an invariant for $\mathcal{M}$. ■

1.5 Die Hard Once and For All

We are at last ready to prove the theorem.

Theorem 1.8 Let $P((a, b))$ be the predicate that a and b are multiples of 3. $P((a, b))$ is an invariant for Die Hard.

Proof. The proof uses the Invariant Theorem.

First, we must show that the predicate holds for all start states. $P((0,0))$ is true because 0 is a multiple of 3.

Next, we must show that $P((a,b)) \Rightarrow P((a',b'))$ for all $(a,b) \rightarrow (a',b') \in \delta$. That is, for each of the six types of transition in δ, we assume a and b are multiples of 3 to prove that a' and b' are multiples of 3. This gives six cases:

1. Transitions of the form $(a,b) \rightarrow (3,b)$ arising from filling the 3-gallon jug. Both 3 and b are multiples of 3.
2. Transitions of the form $(a,b) \rightarrow (a,6)$ arising from filling the 6-gallon jug. Both a and 6 are multiples of 3.
3. Transitions of the form $(a,b) \rightarrow (0,b)$ arising from emptying the 3-gallon jug. Both 0 and b are multiples of 3.
4. Transitions of the form $(a,b) \rightarrow (a,0)$ arising from emptying the 6-gallon jug. Both a and 0 are multiples of 3.
5. Transitions of the form $(a,b) \rightarrow (max(0,a+b-6), min(a+b,6))$ arising from pouring the 3-gallon jug into the 6-gallon jug. All of 0, $a+b-6$, $a+b$, and 6 are multiples of 3.
6. Transitions of the form $(a,b) \rightarrow (min(a+b,3), max(0,a+b-3))$ arising from pouring the 6-gallon jug into the 3-gallon jug. All of $a+b$, 3, 0, and $a+b-3$ are multiples of 3.

This proves that the predicate $P((a,b))$ is an invariant for Die Hard. ■

Corollary 1.9 Bruce dies!

(Whew! Maybe Bruce gets the last laugh; sure, he is blown to pieces by a mad bomber, but we're going to have to prove another invariant in the next section.)

Reducing the problem to a state machine and invariant question makes the problem so concrete that we might even hope to automatically check the claim that Bruce must die. This is another motivation for formal methods– checking the correctness of procedures and protocols by computer.

As a final note, what if we had tried to prove Bruce's demise with a different invariant? Let $P((a,b))$ the predicate that $b \neq 4$. This predicate is an invariant; the theorem implies that the predicate is true for all reachable states, and this is the definition of an invariant. But this invariant could not easily be proven with the Invariant Theorem. The problem is that this predicate may hold for a state, but not for a state only one transition away. For example, the predicate holds for (2, 2), but not for (0, 4). This is a roadblock, since the Invariant Theorem requires that we prove $P(q) \Rightarrow P(q')$ for all $(q,q') \in \delta$. The situation is analogous to the problem that we encountered in first trying to write an inductive proof that Bruce dies. Then the induction hypothesis was not strong enough. Here also, we had to strengthen the predicate to prove that it was an invariant.

2 The Robot

There is a robot. He walks around on a grid and at every step he moves 1 unit north or south *and* 1 unit east or west. (Read the last sentence again; if you do not have the robot's motion straight, you will be lost!) The robot starts at position $(0,0)$. Can the robot reach position $(1,0)$?

To get some intuition, we can build such a robot from soup cans and pipe cleaners and then observe its behavior over time. This experiment quickly suggests that the answer to the question is "no", but can we prove it?

Let's try to model the problem as a state machine and then prove a suitable invariant. Recall that a state machine has three parts, a set of states Q, a set of start states Q_0, and a set of transitions δ. To model the robot problem, we could define these variables as follows:

$$\begin{aligned} Q &= \{(i,j) \mid i,j \in \mathcal{Z}\} \\ Q_0 &= \{(0,0)\} \\ \delta &= \{(i,j) \rightarrow (i',j') \mid i' = i \pm 1, j' = j \pm 1\} \end{aligned}$$

The set $\mathcal{Z}$ is the integers. The problem is now to choose an appropriate predicate $P((i,j))$ and prove that it is an invariant. If this predicate is false for $(1,0)$, then we will have proven the robot can never reach $(1,0)$. A natural choice is to let $P((i,j))$ be the predicate that $(i,j) \neq (1,0)$.

Unfortunately, this is not going to work. The Invariant Theorem requires that we prove $P((i,j)) \Rightarrow P((i',j'))$ for all $(i,j) \rightarrow (i',j') \in \delta$. But this just is not true for our choice of predicate. For example, $(2,1) \rightarrow (1,0) \in \delta$, but $P((2,1))$ does not imply $P((1,0))$, because the former is true and the latter is false.

We need a stronger predicate. Consulting a list of consecutive positions of our robot gives some insight:

$$(0,0) \Rightarrow (-1,1) \Rightarrow (-2,0) \Rightarrow (-1,1) \Rightarrow (0,2) \Rightarrow (1,1) \Rightarrow \ldots$$

The sum of the coordinates is always even! If we can prove that this is an invariant, then we have proven that the robot never reaches $(1,0)$ because the sum of the coordinates at that point is odd.

Theorem 2.1 The sum of the robot's coordinates is always even.

Proof. The proof uses the Invariant Theorem.

Let $P((i,j))$ be the predicate that $i+j$ is even.

First, we must show that the predicate holds for all start states. $P((0,0))$ is true because $0+0$ is even.

Next, we must show that $P((i,j)) \Rightarrow P((i',j'))$ for all $(i,j) \rightarrow (i',j') \in \delta$. That is, for each transition $(i,j) \rightarrow (i',j') \in \delta$, assume $i+j$ is even to prove $i'+j'$ is even. By the definition of δ, $i' = i \pm 1$ and $j' = j \pm 1$. Therefore, $i'+j'$ is equal to $i+j-2$, $i+j$, or $i+j+2$, all of which are even. ∎

Corollary 2.2 $(1, 0)$ is not reachable.

This is true because $1 + 0$ is odd.