6.1200J/18.062J *Mathematics for Computer Science* Thursday 15th February, 2024
Massachusetts Institute of Technology, Spring 2024
Z. Abel, B. Chapman, E. Demaine
revised Monday 4th March, 2024

Lecture 04: State machines

Today we're going to introduce a new abstraction called a *state machine*, that lets us model how algorithms work. We'll show how to use induction to prove properties of state machines. But first, an example.

1 The 8 Puzzle

Suppose we have 3×3 board with 8 tiles and one empty space, arranged as follows.

1	2	3
4	5	6
8	7	$*$

We're allowed to slide a tile horizontally or vertically into the empty space. Using moves only of this type, is it possible to convert the above configuration to the configuration below?

1	2	3
4	5	6
7	8	$*$

This is called the 8 puzzle. A version on a 4×4 grid with 15 tiles, called the 15 puzzle, was a national sensation in America in the 1880s, with large cash prizes being offered for anyone who could find a solution. We do not have that much cash, but we have candy. Would anyone like to try to solve the 8 puzzle in *two minutes* in return for some candy?

2 State machines

This was a trick question. It turns out the 8 puzzle is impossible to solve, and we can prove it! We will do so by introducing the framework of *state machines*, which are an abstraction that is very useful for modeling algorithms.

Definition 1. *A state machine is defined by a collection of* states, *a specified* initial state, *and for each state a (possibly empty) list of possible* transitions *to other states. If a state has multiple possible transitions, we say the state machine is* non-deterministic*; otherwise, it is* deterministic.

An *execution* of a state machine is just a sequence of states, starting from the initial state, that follows the transition rules.

How do we model the 8 puzzle as a state machine?

- The *states* are board configurations. We will write these concisely by listing out the cells in left-to-right, top-to-bottom order.
- The *initial state* is the board configuration 12345687∗.

The question we would like to answer is one of *reachability*.

Definition 2. *A state is* reachable *if there is some execution that reaches that state.*

Theorem 1 (What we want to show)**.** *The state* 12345678∗ *is not reachable.*

3 Reachability and invariants

There is a powerful technique to prove *unreachability* of states, called the *invariant principle*. It is an application of mathematical induction to state machines. In the abstract, the idea is extremely simple.

Definition 3. *A* preserved predicate *is a predicate $P(\cdot)$ defined on states, such that if $P(s)$ is true for a state s, and there exists a transition $s \mapsto t$ from s to t, then $P(t)$ is true as well.*

Definition 4. *An* invariant *is a property that is true for all reachable states.*

Theorem 2 (The Invariant Principle)**.** *Suppose $P(\cdot)$ is true for the initial state and is a preserved predicate. Then $P(\cdot)$ is an invariant.*

Proof. Let s_0 be the initial state of our state machine. Let $Q(n)$ be the following predicate: for all executions $s_0, s_1, \ldots, s_n$ of $n+1$ states, it holds that $P(s_0) \wedge P(s_1) \wedge \ldots P(s_n)$ is true. We are going to prove that $Q(n)$ is true for all $n \geq 0$ by induction.

Base case: For $n = 0$ the only execution is s_0, and we know by definition that $P(s_0)$ is true. **Inductive step:** Let $s_0, s_1, \ldots, s_{n+1}$ be an execution. Then $s_0, \ldots, s_n$ is an execution of length $n+1$, so by the inductive hypothesis, $P(s_n)$ is true. Since $P(\cdot)$ is an invariant, it is a preserved predicate as well, so $P(s_{n+1})$ also holds.

Thus, by induction, we have shown that $Q(n)$ holds for all n. □

Corollary 3. *If $P(s)$ is false and P is an invariant, then s in unreachable.*

Proof. By contradiction. □

The basic idea is simple, but choosing the invariant $P(\cdot)$ is not always so simple! We'll show you an invariant for the 8 puzzle, but in general this is a bit of an art. In particular, this week's PEST has a problem where the invariant is extremely difficult to guess if you're not experienced with these sorts of proofs. As instructed on the problem set, we encourage you to *ask for a hint* for this problem—after you've thought about it a bit on your own!

Counting inversions Let $a_1, a_2, \ldots, a_n$ be a sequence of natural numbers. The number of *inversions* is the number of pairs of indices $i < j$ such that $a_i > a_j$. For example the sequence 1234 has no inversions. The sequence 1324 has 1 inversion ($i, j = 2, 3$). The sequence 4132 has 4 inversions.

Let's now think about board configurations of the 8 puzzle, and map each one to a sequence of numbers from 1 to 8 by just deleting the $*$. What happens to the number of inversions in this sequence as we slide tiles around?

- **Horizontal moves:** it's easy to see that a horizontal move doesn't change the sequence at all. This is because a horizontal move just swaps $*$ with one of its neighbors to the left or the right. But when we generate the sequence, we delete $*$, so the sequence is the same. If the sequence doesn't change, the number of inversions doesn't change either. So could this be our invariant?

- **Vertical moves:** Now we have to be careful. Vertical moves can change the sequence!

  $$\begin{array}{|c|c|c|}\hline 1&2&3\\\hline 4&5&6\\\hline 8&7&*\\\hline\end{array} \mapsto \begin{array}{|c|c|c|}\hline 1&2&3\\\hline 4&5&*\\\hline 8&7&6\\\hline\end{array}$$

  The sequence changed from 12345687 to 12345876. The first has 1 inversion. The second has 3 inversions! So the number of inversions changed.

  Let's try another example. Let's do a horizontal move and then another vertical move.

  $$\begin{array}{|c|c|c|}\hline 1&2&3\\\hline 4&5&*\\\hline 8&7&6\\\hline\end{array} \mapsto \begin{array}{|c|c|c|}\hline 1&2&3\\\hline 4&*&5\\\hline 8&7&6\\\hline\end{array} \mapsto \begin{array}{|c|c|c|}\hline 1&*&3\\\hline 4&2&5\\\hline 8&7&6\\\hline\end{array}$$

  Now the sequence we end up with 13425876. This has 5 inversions! So the number of inversions changes. It seems to change by 2 each time! In fact, a vertical move always changes the sequence by shuffling three adjacent elements like this:

  $$a_i, a_{i+1}, a_{i+2} \mapsto a_{i+2}, a_i, a_{i+1}.$$

  This changes the number of inversions as follows. The pair of indices $i, i+1$ is an inversion afterwards iff $i, i+2$ was *not* an inversion before. The pair $i, i+2$ is an inversion afterwards iff $i+1, i+2$ was *not* an inversion before. The pair $i+1, i+2$ is an inversion iff the pair $i, i+1$ *was* an inversion before.

  This seems pretty complicated! The number of inversions could even remain constant under this: e.g. $132 \mapsto 213$. Maybe the number of inversions is the wrong thing to look at. Indeed, let's look at something a bit more coarse-grained: the *parity* of the number of inversions. Basically, in this process, we *toggle* exactly two of the pairs of indices. So the number of inversions must always change by an *even* number. This means the parity of the number of inversions is unchanged by vertical moves! (Even plus odd is odd, and even plus even is even!)

So define the predicate $P(s)$ to be true if the number of inversions in s is odd. Then P is a preserved predicate by the above. Moreover, in the initial state 12345687$*$, P is true because the number of inversions is 1. Hence, by the invariant principle, P is an invariant! Now, to conclude, observe that 12345678$*$ has 0 inversions—an even number! Therefore, it is unreachable.

4 Termination

So far we've seen how to use state machines and invariants to argue about reachability. Another important property of state machines is termination.

Definition 5. *A* final state *is one with no possible transitions.*

Definition 6. *A state machine* terminates *if there are no infinite executions.*

Intuitively, this says that if you try to run the state machine, then no matter what choices you make for each transition, you end up at a final state.

There is a powerful technique to prove termination called the method of *derived variables*, also known as *potential functions*.

Definition 7. *A derived variable (aka potential function) is a function mapping states to real numbers.*

Definition 8. *A derived variable f is* strictly decreasing *if, for any transition $s \mapsto t$, $f(t) < f(s)$. It's* weakly decreasing *if $f(t) \leq f(s)$.*

Theorem 4. *Suppose f is a derived variable with* natural number values *that is* strictly decreasing. *Then the state machine terminates.*

The idea is that if the state machine did not terminate, then there would be an infinite execution. But the sequence of derived variables on the states of this execution would have to be an infinite sequence of natural numbers that is strictly decreasing—this is impossible!

We will show how to use this idea to prove that a very simple sorting algorithm works.

Simple sort Suppose we have a sequence of n letters (not necessarily distinct)

$$a_1, a_2, \ldots, a_n.$$

Let's consider the following algorithm to put them in sorted (alphabetical) order. At each step, choose an index i such that a_i, a_{i+1} are out of order, and swap them. If no such i exists, then terminate.

What is the state machine for this?

- States: all permutations (reorderings) of the input sequence.

- Initial state: the given sequence.
- Transitions: Clear from the description of the algorithm.

What about final states? These are precisely the states such that for all i, a_i, a_{i+1} are in sorted order.

Claim 5. *Every final state is in sorted order.*

Exercise: prove this using induction!

So clearly the algorithm *works* if it happens to hit a final state (this is called *partial correctness*). What if it never hits one though?

Theorem 6. *The simple sort state machine terminates.*

Proof. Of course, we're going to use a potential function for this. What shall we choose? Let's try the number of inversions! At least this is natural number valued.

It remains to show that it is strictly decreasing. Indeed, this is easy:

$$\cdots a_{i-1}a_ia_{i+1}\cdots \mapsto \cdots a_{i-1}a_{i+1}a_i\cdots .$$

The only pair of indices that can possibly change its inversion status is $i, i+1$. And by the choice of i, we know this pair always goes from being an inversion to being a *non*-inversion. Hence the number of inversions always goes down by exactly 1. Hence, the state machine terminates! □

In fact, we get something a little stronger: a bound on *how many steps* until termination. We can see that no matter what choice of transitions we make, it is always equal to the number of inversions in the input sequence. This is always at most $n(n-1)/2$, the total number of pairs of indices.

Physics inspiration The term " potential function" should remind you of potential energy from physics. Indeed, both the invariant principle and the method of potential functions are reminiscent of the role energy plays in physics. Total energy is like an *invariant*: it is a function of the state, that cannot change under transitions. We know that a state that has different energy than the initial state is impossible to reach. The "potential function" is maybe not so aptly named: it behaves more like the mechanical energy, which can only be reduced by dissipative forces, but can never go up in a closed system.