Massachusetts Institute of Technology
6.042J/18.062J, Fall '05: Mathematics for Computer Science
Prof. Albert R. Meyer and Prof. Ronitt Rubinfeld

Course Notes, Week 7
October 21
revised October 21, 2005, 364 minutes

State Machines: Invariants and Termination

1 State machines

State machines are an abstract model of step-by-step processes, and accordingly, they come up in many areas of Computer Science. You may already have seen them in a digital logic course, a compiler course, or a a probability course.

1.1 Basic definitions

A state machine is really nothing more than a digraph, except that the vertices are called "states" and the edges are called "transitions." The transition (edge) from state p to state q will be written $p \longrightarrow q$.

A state machine also comes equipped with a designated *start state*.

State machines used in digital logic and compilers usually have only a finite number of states, but machines that model continuing computations typically have an infinite number of states. In many applications, the transitions in the state graphs are labelled with tokens indicating inputs or outputs, or numbers indicating costs, capacities or probabilities. In these Notes, we'll stick to unlabelled transitions.

1.2 Examples

Here are some simple examples of state machines.

Example 1.1. A bounded counter, which counts from 0 to 99 and overflows at 100. The state graph is shown in Figure 1, with start state zero.

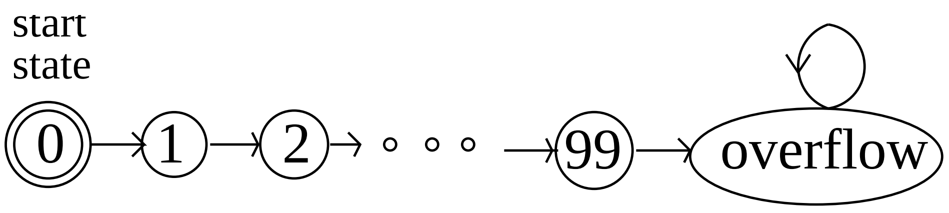

Figure 1: *The state graph of the 99-bounded counter.*

This machine isn't much use once it overflows, since it has no way to get out of its overflow state.

Example 1.2. An unbounded counter is similar, but has an infinite state set, yielding an infinite digraph. This is harder to draw `:-)`

Example 1.3. The Die Hard 3 scenario with a 3 and a 5 gallon water jug could be modelled as a state machine. For states we could use pairs, (b, l) of real numbers such that $0 \leq b \leq 5, 0 \leq l \leq 3$. We let b and l be arbitrary real numbers since Bruce could pour any amount of water into a bucket. The start state is $(0, 0)$, since both jugs start empty.

There are 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.

Problem 1. Which states of the Die Hard 3 machine have direct transitions to exactly two states?

1.3 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 the start state corresponds to a possible system behavior; such a path is called an *execution* of the state machine. A state is called *reachable* if there is a path to it starting from the start state, that is, if it appears in some execution.

We observed that the state (4,3) was reachable, reflecting the fact that Bruce and Samuel successfully disarmed the bomb in Die Hard 3.

2 Reachability and Invariants

A useful approach in analyzing state machine is to identify invariant properties of states.

Definition 2.1. 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 2.2 (Invariant Theorem). *Let P be an invariant predicate for a state machine. If P holds for the start state, 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.

2.1 Die Hard Once and For All

Now back to Die Hard Once and For All. This time there is a 9 gallon jug instead of the 5 gallon jug. We can model this with a state machine whose states and transitions are specified the same way as for the Die Hard 3 machine, with all occurrences of "5" replaced by "9."

Now reaching any state of the form $(4, l)$ is impossible. We prove this using the Invariant Theorem. Namely, we define the invariant predicate, $P((b.l))$, to be that b and l are nonnegative integer mulitples of 3. So P obviously holds for the state state $(0, 0)$.

The proof that P is an invariant, we assume $P((b, l)$ holds for some state $(b.l)$ and show that if $(b, l) \to (b', l')$, then $P(b', l')$. The proof divides into cases, according to which transition rule is used. For example, suppose the transition followed from the "fill the little jug" rule. This means $(b, l) \to (b, 3)$. But $P((b, l)$ impiles that b is an integer multiple of 3, and of course 3 is an integer multiple of 3, so P still holds for the new state $(b, 3)$. Another example is when the transition rule used is "pour from big jug into little jug" for the subcase that $b + l > 3$. Then state is $(b, l) \to (b - (3 - l), 3)$. But since b and l are integer multiples of 3, so is $b - (3 - l)$. So in this case too, P holds after the transition.

We won't bother to crank out the remaining cases, which can all be checked with equal ease. Now by the Invariant Theorem, we conclude that every reachable state satisifies P. But since no state of the form $(4, l)$ satisifies P, we have rigorously proved that Bruce dies once and for all!

2.2 The Robot

There is a robot. He walks around on a grid and at every step he moves one unit north or south *and* one 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 simulate some robot moves. For example, starting at (0,0) the robot could move northeast to (1,1), then southeast to (0,2), then southwest to (-1, 1), then southwest again to (-2, 0).

Let's model the problem as a state machine and then prove a suitable invariant. A state will be a pair of integers corresponding to the coordinates of the robots position. State (i, j) has transitions to four different states: $(i \pm 1, j \pm 1)$.

The problem is now to choose an appropriate invariant predicate, P, that is true for the start state $(0, 0)$ and false for $(1, 0)$. The Invariant Theorem then will imply that the robot can never reach $(1, 0)$. A direct attempt at an invariant is to let $P(q)$ be the predicate that $q \neq (1, 0)$.

Unfortunately, this is not going to work. Consider the state $(2, 1)$. Clearly $P((2, 1))$ holds because $(2, 1) \neq (1, 0)$. And of course $P((1, 0))$ does not hold. But $(2, 1) \to (1, 0)$, so this choice of P will not yield an invariant.

We need a stronger predicate. Looking at our example execution you might be able to guess a proper one, namely, that the sum of the coordinates is even! If we can prove that this is an invariant, then we have proven that the robot never reaches $(1, 0)$ because the sum $1 + 0$ of its coordinates is not an even number, but the sum $0 + 0$ of the coordinates of the start state is an even number.

Theorem 2.3. *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 the start state $(0, 0)$. But and $P((0, 0))$ is true because $0 + 0$ is even.

Next, we must show that P is an invariant. That is, we must show that for each transition $(i, j) \to (i', j')$, if $i+j$ is even, then $i'+j'$ is even. But $i' = i \pm 1$ and $j' = j \pm 1$ by definition of the transitions. Therefore, $i' + j'$ is equal to $i + j - 2$, $i + j$, or $i + j + 2$, all of which are even. $\square$

Corollary 2.4. *The robot cannot reach* $(1, 0)$.

Problem 2. A robot moves on the two-dimensional integer grid. It starts out at $(0, 0)$, and is allowed to move in any of these four ways:

1. (+2,-1) Right 2, down 1
2. (-2,+1) Left 2, up 1
3. (+1,+3)
4. (-1,-3)

Prove that this robot can never reach (1,1).

3 Sequential algorithm examples

The Invariant Theorem was formulated by Robert Floyd at Carnegie Tech in 1967[1]. Floyd was already famous for work on formal grammars that had wide influence in the design of programming language syntax and parsers; in fact, that was how he got to be a professor even though he never got a Ph.D.

[1]The following year, Carnegie Tech was renamed Carnegie-Mellon Univ.

In that same year, Albert R. Meyer was appointed Assistant Professor in the Carnegie Tech Computation Science department where he first met Floyd. Floyd and Meyer were the only theoreticians in the department, and they were both delighted to talk about their many shared interests. After just a few conversations, Floyd's new junior colleague decided that Floyd was the smartest person he had ever met.

Naturally, one of the first things Floyd wanted to tell Meyer about was his new, as yet unpublished, Invariant Theorem. Floyd explained the result to Meyer, and Meyer could not understand what Floyd was so excited about. In fact, Meyer wondered (privately) how someone as brilliant as Floyd could be excited by such a trivial observation. Floyd had to show Meyer a bunch of examples like the ones that follow in these notes before Meyer realized that Floyd's excitement was legitimate —not at the truth of the utterly obvious Invariant Theorem, but rather at the insight that such a simple theorem could be so widely and easily applied in verifying programs.

Floyd left for Stanford the following year. He won the Turing award —the "Nobel prize" of Computer Science— in the late 1970's, in recognition both of his work on grammars and on the foundations of program verification. He remained at Stanford from 1968 until his death in September, 2001. A eulogy describing Floyd's life and work by his closest colleague, Don Knuth, can be found at `http://www.acm.org/pubs/membernet/stories/floyd.pdf`.

We'll illustrate Floyd's idea by verifying the Euclidean Greatest Common Divisor (GCD) Algorithm. Another example apeears in a class problem.

3.1 Proving Correctness

It's generally useful to distinguish two aspects of state machine or process correctness:

1. The property that the final results, if any, of the process satisfy system requirements. This is called *partial correctness*.

   You might suppose that if a result was only partially correct, then it might also be partially incorrect, but that's not what's meant here. The word "partial" comes from viewing a process that might not terminate as computing a *partial function*. So partial correctness means that when there is a result, it is correct, but the process might not always produce a result, perhaps because it gets stuck in a loop.

2. The property that the process always finishes or is guaranteed to produce some desired output. This is called *termination*.

Partial correctness can commonly be proved using the Invariant Theorem.

3.2 The Euclidean Algorithm

We have already introduced the Euclidean algorithm to compute $\gcd(a, b)$. We can represent this algorithm as a state machine. A state will be a pair of integers (x, y) which we can think of as integer registers in a register program. The state transitions are defined by the rule

$$(x, y) \to (y, \text{remainder}(x, y)) \qquad\qquad \text{if } y \neq 0.$$

We want to prove:

1. starting from state with $x = a$ and $y = b$, if we ever finish, then we have the right answer. That is, at termination, $x = \gcd(a, b)$. This is a *partial correctness* claim.

2. we do actually finish. This is a process *termination* claim.

3.2.1 Partial Correctness of GCD

First let's prove that if GCD gives an answer, it is a correct answer. Specifically, let $d ::= \gcd(a, b)$. We want to prove that *if* the procedure finishes in a state (x, y), then $x = d$.

Proof. So define the state predicate

$$P((x, y)) ::= \quad [\gcd(x, y) = d \text{ and } (x > 0 \text{ or } y > 0)].$$

P holds for the start state (a, b), by definition of d and the requirement that at least one a and b is positive. Also, P is an invariant because

$$\gcd(x, y) = \gcd(y, \text{remainder}(x, y))$$

for all $x, y \in \mathbb{N}$ such that $y \neq 0$, as we observed in the Notes on Number Theory. So by the Invariant Theorem, P holds for all reachable states.

Since the only rule for termination is that $y = 0$, it follows that if (x, y) is a terminal state, then $y = 0$. If this terminal state is reachable, then the invariant holds for (x, y). This implies that $\gcd(x, 0) = d$ and that $x > 0$. We conclude that $x = \gcd(x, 0) = d$. □

3.2.2 Termination of GCD

Now we turn to the second property, that the procedure must terminate. To prove this, notice that y gets strictly smaller after any one transition. That's because the value of y after the transition is the remainder of x divided by y, and this remainder is smaller than y by definition. But the value of y is always a natural number, so by the Well Ordering Principle, it reaches a minimum value among all its values at reachable states. But there can't be a transition from a state where y has its minimum value, because the transition would decrease y still further. So the reachable state where y has its minimum value is a state at which no further step is possible, that is, at which the procedure terminates.

Note that this argument does not prove that the minimum value of y is zero, only that the minimum value occurs at termination. But we already noted that the only rule for termination is that $y = 0$, so it follows that the minimum value of y must indeed be zero.

3.3 The Extended Euclidean Algorithm

There is an extension of the Euclidean Algorithm that provides an alternative to the Pulverizer for expressing the gcd of two integers and an integer linear combination of them. In particular, given natural numbers x,y, with $y > 0$, we claim the following procedure[2] halts with integers s, t in registers S and T such that

$$sa + tb = \gcd(a, b).$$

Inputs: $x, y \in \mathbb{N}, y > 0$.

Registers: X,Y,S,T,U,V,Q.

Extended Euclidean Algorithm:

```
X := m; Y := n; S := 0; T := 1; U := 1; V := 0;
loop:
if Y|X, then halt
else
  Q := quotient(X,Y);
         ;;the following assignments in braces are SIMULTANEOUS
 {X := Y,
  Y := remainder(X,Y);
  U := S,
  V := T,
  S := U - Q * S,
  T := V - Q * T};
goto loop;
```

Note that X,Y behave exactly as in the Euclidean GCD algorithm in Section 3.2, except that this extended procedure stops one step sooner, ensuring that $\gcd(x, y)$ is in Y at the end. So for all inputs x, y, this procedure terminates for the same reason as the Euclidean algorithm: the contents, y, of register Y is a natural number-valued variable that strictly decreases each time around the loop.

We claim that invariant properties that can be used to prove partial correctness are:

$$\begin{aligned} \gcd(X,Y) &= \gcd(a,b), && (1)\\ Sa+Tb &= Y, \text{ and} && (2)\\ Ua+Vb &= X. && (3) \end{aligned}$$

To verify these invariants, note that invariant (1) is the same one we observed for the Euclidean algorithm. To check the other two invariants, let x, y, s, t, u, v be the contents of registers X,Y,S,T,U,V at the start of the loop and assume that all the invariants hold for these values. We must prove that (2) and (3) hold (we already know (1) does) for the new contents x', y', s', t', u', v' of these registers at the next time the loop is started.

Now according to the procedure, $u' = s, v' = t, x' = y$, so invariant (3) holds for u', v', x' because of invariant (2) for s, t, y. Also, $s' = u - qs, t' = v - qt, y' = x - qy$ where $q = \text{quotient}(x, y)$, so

$$s'a + t'b = (u - qs)a + (v - qt)b = ua + vb - q(sa + tb) = x - qy = y',$$

and therefore invariant (2) holds for s', t', y'.

[2]This procedure is adapted from Aho, Hopcroft, and Ullman's text on algorithms.

Also, it's easy to check that all three invariants are true just before the first time around the loop. Namely, at the start $X = a, Y = b, S = 0, T = 1$ so $Sa + Tb = 0a + 1b = b = Y$ so (b) holds; also $U = 1, V = 0$ and $Ua + Vb = 1a + 0b = a = X$ so (3) holds. So by the Invariant Theorem, they are true at termination. But at termination, the contents, Y, of register Y divides the contents, X, of register X, so invariants (1) and (2) imply

$$\gcd(a, b) = \gcd(X, Y) = Y = Sa + Tb.$$

So we have the gcd in register Y and the desired coefficients in S, T.

4 Derived Variables

The preceding termination proofs involved finding a natural-number-valued measure to assign to states. We might call this measure the "size" of the state. We then showed that the size of a state decreased with every state transition. By the Well Ordering Principle, the size can't decrease indefinitely, so when a minimum size state is reached, there can't be any transitions possible: the process has terminated.

More generally, the technique of assigning values to states —not necessarily natural numbers and not necessarily decreasing under transitions— is often useful in the analysis of algorithms. *Potential functions* play a similar role in physics. In the context of computational processes, such value assignments for states are called *derived variables*.

For example, for the Die Hard machines we could have introduced a derived variable, f : states $\to \mathbb{R}$, for the amount of water in both buckets, by setting $f((a, b)) ::= a + b$. Similarly, in the robot problem, the position of the robot along the x-axis would be given by the derived variable x-coord, where x-coord$((i, j)) ::= i$.

We can formulate our general termination method as follows:

Definition 4.1. A derived variable f : states $\to \mathbb{R}$ is *strictly decreasing* iff

$$q \to q' \text{ implies } f(q') < f(q).$$

Theorem 4.2. *If f is a strictly decreasing derived variable of a state machine that takes only nonnegative integer values, then the length of any execution starting at state q is at most $f(q)$.*

Of course we could prove Theorem 4.2 by induction on the value of $f(q)$. But think about what it says: "If you start counting down at some natural number $f(q)$, then you can't count down more than $f(q)$ times." Put this way, the theorem is so obvious that no one should feel deprived that we omit a proof.

Corollary 4.3. *If there exists a strictly decreasing natural-number-valued derived variable for some state machine, then every execution of that machine terminates.*

We now define some other useful flavors of derived variables taking values over partial ordered sets. It's useful to generalize the familiar notations $\leq$ and $<$ for ordering the real numbers: if $\preceq$ is a partial order on some set A, then define $\prec$ to be the corresponding *strict* partial order

$$a \prec a' \; ::= \quad a \preceq a' \wedge a \neq a'.$$

Definition 4.4. Let $\preceq$ be a partial order on a set, A. A derived variable $f : Q \to A$ is *strictly decreasing* with respect to $\preceq$ iff

$$q \to q' \text{ implies } f(q') \preceq f(q) \text{ and } f(q') \neq f(q).$$

It is *weakly decreasing* iff

$$q \to q' \text{ implies } f(q') \preceq f(q) \text{ or } f(q') = f(q).$$

Strictly increasing and *weakly increasing* derived variables are defined similarly.[3]

The existence of a natural-number-valued *weakly* decreasing derived variable does not guarantee that every execution terminates. That's because an infinite execution could proceed through states in which a weakly decreasing variable remained constant.

5 The Stable Marriage Problem

Okay, frequent public reference to derived variables may not help your mating prospects. But they can help with the analysis!

5.1 The Problem

Suppose that there are n boys and n girls. Each boy ranks all of the girls according to his preference, and each girl ranks all of the boys. For example, Bob might like Alice most, Carol second, Hildegard third, etc. There are no ties in anyone's rankings; Bob cannot like Carol and Hildegard equally. Furthermore, rankings are known at the start and stay fixed for all time.

The general goal is to marry off boys and girls so that everyone is happy; we'll be more precise in a moment. Every boy must marry exactly one girl and vice-versa —no polygamy.

If we want these marriages to last, then we want to avoid an unstable arrangement:

Definition 5.1. A set of marriages is *unstable* if there is a boy and a girl who prefer each other to their spouses.

For example, suppose that Bob is married to Carol, and Ted is married to Alice. Unfortunately, Carol likes Ted more than Bob *and* Ted likes Carol more than Alice. The situation is shown in Figure 2. So Carol and Ted would both be happier if they ran off together. We say that Carol and Ted are a *rogue couple*, because this is a situation which encourages roguish behavior.

Definition 5.2. A set of marriages is *stable* if there are no rogue couples or, equivalently, if the set of marriages is not unstable.

Now we can state the *Stable Marriage Problem* precisely: find spouses for everybody so that the resulting set of marriages is stable. It is not obvious that this goal is achievable! In fact, in the gender blind *Stable Buddy* version of the problem, where people of any gender can pair up as

[3]Weakly increasing variables are often also called *nondecreasing*. We will avoid this terminology to prevent confusion between nondecreasing variables and variables with the much weaker property of *not* being a decreasing variable.

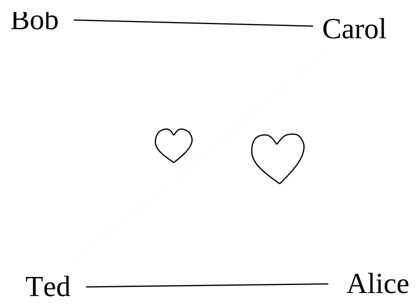

Figure 2: *Bob is married to Carol, and Ted is married to Alice, but Ted prefers Carol to his mate, and Carol prefers Ted to her mate. So Ted and Carol form a rogue couple, making their present marriages unstable.*

buddies, there may be no stable pairing! However, for the "boy-girl" marriage problem, a stable set of marriages does always exist.

Incidentally, although the classical "boy-girl-marriage" terminology for the problem makes some of the definitions easier to remember (we hope without offending anyone), solutions to the Stable Marriage Problem are really useful. The stable marriage algorithm we describe was first published in a paper by D. Gale and L.S. Shapley in 1962. At the time of publication, Gale and Shapley were unaware of perhaps the most impressive example of the algorithm. It was used to assign residents to hospitals by the National Resident Matching Program (NRMP) that actually predates their paper by ten years. Acting on behalf of a consortium of major hospitals (playing the role of the girls), the NRMP has, since the turn of the twentieth century, assigned each year's pool of medical school graduates (playing the role of boys) to hospital residencies (formerly called "internships"). Before the procedure ensuring stable matches was adopted, there were chronic disruptions and awkward countermeasures taken to preserve assignments of graduates to residencies. The stable marriage procedure was discovered and applied by the NRMP in 1952. It resolved the problem so successfully, that the procedure continued to be used essentially without change at least through 1989.[4]

Let's find a stable set of marriages in one possible situation, and then try to translate our method to a general algorithm. The table below shows the preferences of each girl and boy in decreasing order.

boys	*girls*
$1 : CBEAD$	$A : 35214$
$2 : ABECD$	$B : 52143$
$3 : DCBAE$	$C : 43512$
$4 : ACDBE$	$D : 12345$
$5 : ABDEC$	$E : 23415$

How about we try a "greedy" strategy?[5] We simply take each boy in turn and pack him off with

[4]Much more about the Stable Marriage Problem can be found in the very readable mathematical monograph by Dan Gusfield and Robert W. Irving, The Stable Marriage Problem: Structure and Algorithms, MIT Press, Cambridge, Massachusetts, 1989, 240 pp.

[5]"Greedy" is not any moral judgment. It refers to algorithms that work by always choosing the next state that makes the largest immediate progress.

his favorite among the girls still available. This gives the following assignment.

$$\begin{aligned} 1 &\to C \\ 2 &\to A \\ 3 &\to D \\ 4 &\to B \\ 5 &\to E \end{aligned}$$

To determine whether this set of marriages is stable, we have to check whether there are any rogue couples. Boys 1, 2, and 3 all got their top pick among the girls; none would even think of running off. Boy 4 may be a problem because he likes girl A better than his mate, but she ranks him dead last. However, boy 4 also likes girl C better than his mate, and she rates him above her own mate. Therefore, boy 4 and girl C form a rogue couple! The marriages are not stable. We could try to make ad hoc repairs, but we're really trying to develop a general strategy.

Another approach would be to use induction. Suppose we pair Boy 1 with his favorite girl, C, try to show that neither of these two will be involved in a rogue couple, and then solve the remaining problem by induction. Clearly Boy 1 will never leave his top pick, Girl C. But the problem with this approach is that we *can't* be sure that Girl C won't be in a rogue couple. Girl C might very well dump Boy 1 – she might even rate him last!

This turns out to be a tricky problem. The best approach is to use a mating ritual that is reputed to have been popular in some mythic past.

5.2 The Mating Algorithm

We'll describe the algorithm as a Mating Ritual that takes place over several days. The following events happen each day:

Morning: Each girl stands on her balcony. Each boy stands under the balcony of his favorite among the girls on his list, and he serenades her. If a boy has no girls left on his list, he stays home and does his 6.042 homework.

Afternoon: Each girl who has one or more suitors serenading her, says to her favorite suitor, "Maybe. . . , come back tomorrow." To the others, she says, "No. I will never marry you! Take a hike!"

Evening: Any boy who is told by a girl to take a hike, crosses that girl off his list.

Termination condition: When every girl has at most one suitor, the ritual ends with each girl marrying her suitor, if she has one.

There are a number of facts about this algorithm that we would like to prove:

- The algorithm terminates.
- Everybody ends up married.

- The resulting marriages are stable.

Furthermore, we would like to know if the algorithm is fair. Do both boys and girls end up equally happy?

5.3 The State Machine Model

Before we can prove anything, we should have clear mathematical definitions of what we're talking about. In this section we sketch how to define a rigorous state machine model of the Marriage Problem.

So let's begin by formally defining the problem.

Definition 5.3. A Marriage Problem consists of two disjoint sets of size $n \in \mathbb{N}$ called the-Boys and the-Girls. The members of the-Boys are called *boys*, and members of the-Girls are called *girls*. For each boy, B, there is a strict total order, $<_B$, on the-Girls, and for each girl, G, there is a strict total order, $<_G$, on the-Boys.

The idea is that $<_B$ is boy B's preference ranking of the girls. That is, $G_1 <_B G_2$ means that B prefers girl G_2 to girl G_1. Similarly, $<_G$ is girl G's preference ranking of the boys.

To model the Mating Algorithm with a state machine we make a key observation: to determine what happens on any day of the ritual, all we need to know is which girls are on which boys' lists that day. So we define a state to some mathematical data structure providing this information. Namely, define a state to be the "still-has-on-his-list" relation, R, between boys and girls. So $B\,R\,G$ means girl G has not been crossed off boy B's list.

We start the Mating Algorithm with no girls crossed off. So in the start state is the *complete bipartite* relation in which every boy is related to every girl.

According to the Mating ritual, on any given morning, a boy will serenade the girl he most prefers among those he has not as yet crossed out. Mathematically, the girl he is serenading is just the maximum girl under B's ordering $<_B$ among the girls on B's list. (If the list is empty, he's not serenading anybody.)

We could go on to define a girl's favorite to be the maximum under her preference ordering among the boys serenading her. We could go on similarly to specify the state transitions with precise mathematical rules, but we won't bother. The point is that if need be, we could define everything using basic mathematical data structures like sets, functions, and relations. But let's stick with the intuitive vocabulary of boys and girls and their preferences.

5.4 Termination

It's easy to see why the Mating Algorithm terminates: every day at least one boy will cross a girl off his list. If no girl can be crossed off any list, then the ritual has terminated. But initially there are n girls on each of the n boys' lists for a total of n^2 list entries. Since no girl ever gets added to a list, the total number of entries on the lists decreases every day that the ritual continues, and so the ritual can continue for at most n^2 days.

5.5 They All Live Happily Every After...

We still have to prove that the Mating Algorithm leaves everyone in a stable marriage. To do this, we note one very useful fact about the ritual: if a girl has a favorite boy suitor on some morning of the ritual, then that favorite suitor will still be serenading her the next morning —because his list won't have changed. So she is sure to have today's favorite boy among her suitors tomorrow. That means she will be able to choose a favorite suitor tomorrow who is at least as desirable to her as today's favorite. So day by day, her favorite suitor can stay the same or get better, never worse. In others words, a girl's favorite is a weakly increasing variable with respect to her preference order on the boys.

Now we can verify the Mating Algorithm using a simple invariant predicate, P, that captures what's going on:

> For every girl, G, and every boy, B, if G is crossed off B's list, then G has a favorite suitor and she prefers him over B.

Why is P invariant? Well, we know that G's favorite tomorrow will be at least as desirable to her as her favorite today, and since her favorite today is more desirable than B, tomorrow's favorite will be too.

Notice that P also holds on the first day, since every girl is on every list. So by the Invariant Theorem, we know that P holds on every day that the Mating ritual runs. Knowing the invariant holds when the Mating Algorithm terminates will let us complete the proofs.

Theorem 5.4. *Everyone is married by the Mating Algorithm.*

Proof. Suppose, for the sake of contradiction, that some boy is not married on the last day of the Mating ritual. So he can't be serenading anybody, that is, his list must be empty. So by invariant P, every girl has a favorite boy whom she prefers to that boy. In particular, every girl has a favorite boy that she marries on the last day. So all the girls are married. What's more there is no bigamy: a boy only serenades one girl, so no two girls have the same favorite.

But there are the same number of girls as boys, so all the boys must be married too. $\square$

Theorem 5.5. *The Mating Algorithm produces stable marriages.*

Proof. Let Bob be some boy and Carole be any girl that he is *not* married to on the last day of the Mating ritual. We will prove that Bob and Carole are not a rogue couple. Since Bob was an arbitrary boy, it follows that no boy is part of a rogue couple. Hence the marriages on the last day are stable.

To prove the claim, we consider two cases:

Case 1. Carole is not on Bob's list. Then since invariant P holds, we know that Carole prefers her husband to Bob. So she's not going to run off with Bob: the claim holds in this case.

Case 2. Otherwise, Carole is on Bob's list. But since Bob is not married to Carole, he must have chosen to serenade his wife instead of Carole, so he must prefer his wife. So he's not going to run off with Carole: the claim also holds in this case. $\square$

5.6 ...And the Boys Live Especially Happily

Who is favored by the Mating Algorithm, the boys or the girls? The girls seem to have all the power: they stand on their balconies choosing the finest among their suitors and spurning the rest. What's more, we know their suitors can only change for the better as the Algorithm progresses.

Similarly, a boy keeps serenading the girl he most prefers among those on his list until he must cross her off, at which point he serenades the next most preferred girl on his list. So from the boy's point of view, the girl he is serenading can only change for the worse. Sounds like a good deal for the girls.

But it's not! The fact is that from the beginning the boys are serenading their first choice girl, and the desirability of the girl being serenaded decreases only enough to give the boy his most desirable possible spouse. So the mating algorithm does as well as possible for all the boys and actually does the worst possible job for the girls.

To explain all this we need some definitions. Let's begin by observing that while the mating algorithm produces one set of stable marriages, there may be other ways to arrange stable marriages among the same set of boys and girls. For example, reversing the roles of boys and girls will often yield a different set of stable marriages among them.

Definition 5.6. If there is some stable set of marriages in which a girl, G, is married to a boy, B, then G is said to be a *possible spouse* for B, and likewise, B is a *possible spouse* for G.

This captures the idea that incompetent nerd Ben Bitdiddle has no chance of marrying pop star Britney Spears: she is not a possible spouse. No matter how the cookie crumbles, there will always be some guy that she likes better than Ben and who likes her more than his own mate. No marriage of Ben and Britney can be stable.

Note that since the mating algorithm always produces one stable set of marriages, the set of possible spouses for a person —even Ben— is never empty.

Definition 5.7. A person's *optimal* spouse is the possible spouse that person most prefers. A person's *pessimal* spouse is the possible spouse that person least prefers.

Here is the shocking truth about the Mating Algorithm:

Theorem 5.8. *The Mating Algorithm marries every boy to his optimal mate and every girl to her pessimal mate.*

Proof. The proof is in two parts. First, we show that every boy is married to his optimal mate. The proof is by contradiction.

Assume for the purpose of contradiction that some boy does not get his optimal girl. There must have been a day when he crossed off his optimal girl —otherwise he would still be serenading her or some even more desirable girl.

By the Well Ordering Principle, there must be a first day when a boy crosses off his optimal girl. Let B be one of these boys who first crosses off his optimal girl, and let G be B's optimal girl.

According to the rules of the ritual, B crosses off G because she has a favorite suitor, B', whom she prefers to B. So on the morning of the day that B crosses off G, her favorite B' has not crossed off his own optimal mate. This means that B' must like G more than any other possible spouse. (Of course, she may not be a possible spouse; she may dump him later.)

Since G is a possible spouse for B, there must be a stable set of marriages where B marries G, and B' marries someone else. But B' and G are a rogue couple in this set of marriages: G likes B' more than her mate, B, and B' likes G more than the possible spouse he is married to. This contradicts the assertion that the marriages were stable.

Now for the second part of the proof: showing that every girl is married to her pessimal mate. Again, the proof is by contradiction.

Suppose for the purpose of contradiction that there exists a stable set of marriages, $\mathcal{M}$, where there is a girl, G, who fares worse than in the Mating Algorithm. Let B be her spouse in the Mating Algorithm. By the preceding argument, G is B's optimal spouse. Let B' be her spouse in $\mathcal{M}$, a boy whom she likes even less than B. Then B and G form a rogue couple in $\mathcal{M}$: B prefers G to his mate, because she is optimal for him, and G prefers B to her mate, B', by assumption. This contradicts the assertion that $\mathcal{M}$ was a stable set of marriages. $\square$