Earlier we talked about about finding equivalent FSMs with fewer states.

Now we'll develop an approach for finding such FSMs by looking for two states that that can be merged into a single state without changing the behavior of the FSM in any externally distinguishable manner.

Two states are equivalent if they meet the following two criteria.

First, the states must have identical outputs.

This makes sense: the outputs are visible to the outside, so if their values differed between the two states, that difference would clearly be externally distinguishable!

Second, for each combination of input values, the two states transition to equivalent states.

Our strategy for deriving an equivalent machine with fewer states will be to start with our original FSM, find pairs of equivalent states and merge those states.

We'll keep repeating the process until we can't find any more equivalent states.

Let's try this on our ant FSM.

First we need to find a pair of states that have the same outputs.

As it turns out, there's only one such pair: WALL1 and CORNER, both of which assert the turn-right and forward outputs.

Okay, so let's assume that WALL1 and CORNER are equivalent and ask if they transition to equivalent states for each applicable combination of input values.

For these two states, all the transitions depend only on the value of the R input, so we just have to check two cases.

If R is 0, both states transition to CORNER.

If R is 1, both states transition to WALL2.

So both equivalence criteria are satisfied and we can conclude that the WALL1 and CORNER states are equivalent and can be merged.

This gives us the four-state FSM shown here, where we've called the single merged state WALL1.

This smaller, equivalent FSM behaves exactly as the previous 5-state FSM.

The implementation of the 5-state machine requires 3 state bits; the implementation of the 4-state machine only requires 2 state bits.

Reducing the number of state bits by 1 is huge since it reduces the size of the required ROM by half!

Just as we were able to achieve considerable hardware savings by minimizing Boolean equations, we can do the same in sequential logic by merging equivalent states.

Roboant customers are looking forward to the price cut!

Let's look at what we'd need to do if we wanted to implement the FSM using logic gates instead a ROM for the combinational logic.

First we have to build the truth table, entering all the transitions in the state transition diagram.

We'll start with the LOST state.

So if the FSM is in this state, the F output should be 1.

If both antenna inputs are 0, the next state is also LOST.

Assigning the LOST state the encoding 00, we've captured this information in the first row of the table.

If either antenna is touching, the FSM should transition from LOST to the rotate-counterclockwise state.

We've given this an encoding of 01.

There are three combinations of L and R values that match this transition, so we've added three rows to the truth table.

This takes care of all the transitions from the LOST state.

Now we can tackle the transitions from the rotate-counterclockwise state.

If either antenna is touching, the next state is again rotate-counterclockwise.

So we've identified the matching values for the inputs and added the appropriate three rows to the transition table.

We can continue in a similar manner to encode the transitions one-by-one.

Here's the final table, where we've used don't cares to reduce the number of rows for presentation.

Next we want to come up with Boolean equations for each of the outputs of the combinational logic, i.e., the two next-state bits and the three motion-control outputs.

Here are the Karnaugh maps for the two next-state bits.

Using our K-map skills from Chapter 4, we'll find a cover of the prime implicants for S1-prime and write down the corresponding product terms in a minimal sum-of-products equation.

And then do the same for the other next-state bit.

We can follow a similar process to derive minimal sum-of-products expressions for the motion-control outputs.

Implementing each sum-of-products in a straight-forward fashion with AND and OR gates, we get the following schematic for the ant brain.

Pretty neat!

Who knew that maze following behavior could be implemented with a couple of D registers and a handful of logic gates?

There are many complex behaviors that can be created with surprisingly simple FSMs.

Early on, the computer graphics folks learned that group behaviors like swarming, flocking and schooling can be modeled by equipping each participant with a simple FSM.

So next time you see the massive battle scene from the Lord of the Rings movie, think of many FSMs running in parallel!

Physical behaviors that arise from simple interactions between component molecules can sometimes be more easily modeled using cellular automata - arrays of communicating FSMS - than by trying to solve the partial differential equations that model the constraints on the molecules' behavior.

And here's an idea: what if we allowed the FSM to modify its own transition table?

Hmm.

Maybe that's a plausible model for evolution!

FSMs are everywhere!

You'll see FSMs for the rest of your life…