



Computation and Logic

Traffic Lights

Michael Fourman





@mp4man



1

This course provides a first glimpse of the deep connections between computation and logic. We will focus primarily on the simplest non-trivial examples of logic and computation: propositional logic and finite-state machines.

In this lecture we briefly look again at the Wolf Goose and Corn example. We will then look at another example that introduces some ideas that we will explore further in later lectures, and introduce some notation which should become more familiar in due course.

	West		East
	WW	WB	WE
	CW	CB	CE
	GW	GB	GE
	FW	FB	FE

A farmer has to get a wolf, a goose, and a sack of corn across a river.

She has a boat, which can only carry her and one other thing.

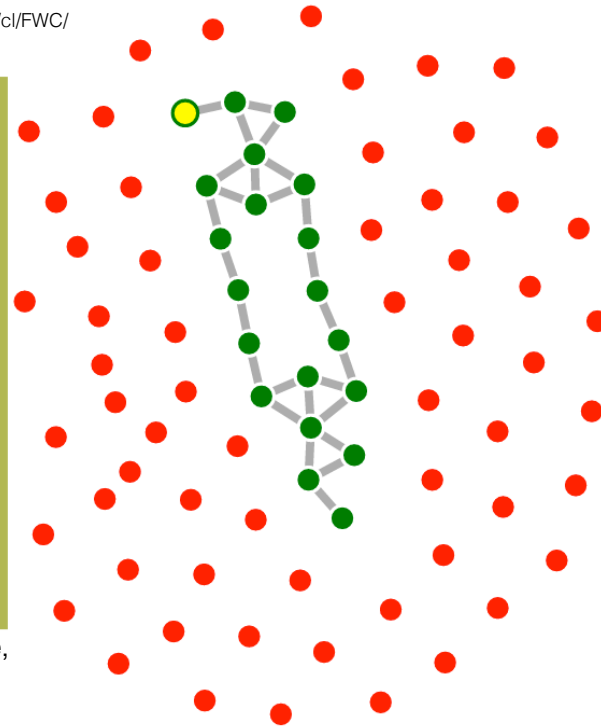
If the wolf and the goose are left together, the wolf will eat the goose.

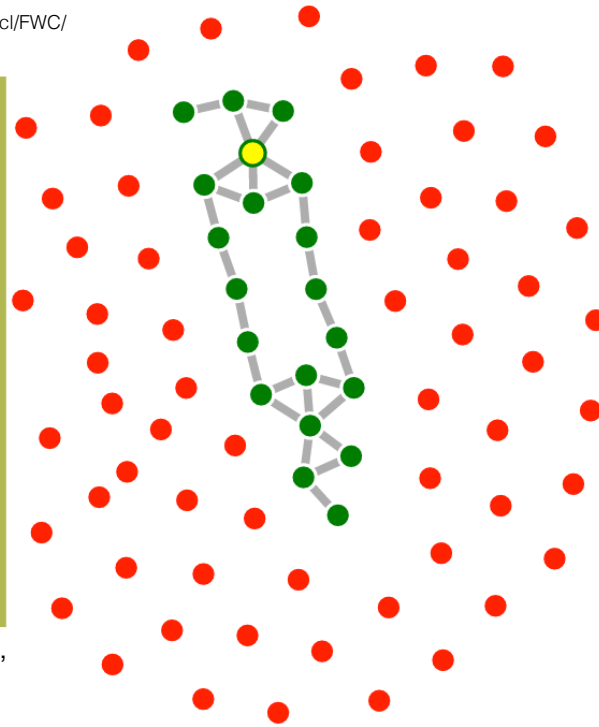
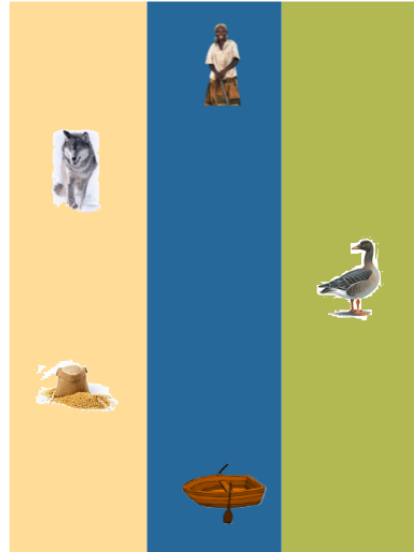
If the goose and the corn are left together, the chicken will eat the corn.



A farmer has to get a wolf, a goose, and a sack of corn across a river.

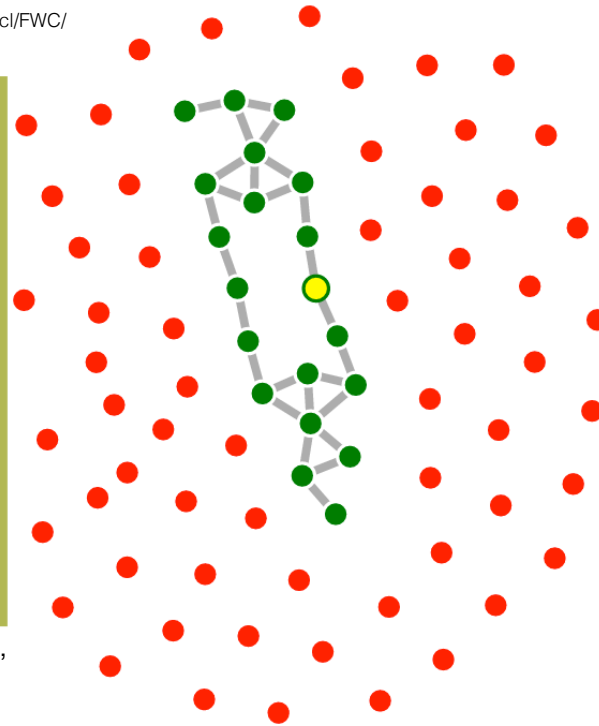
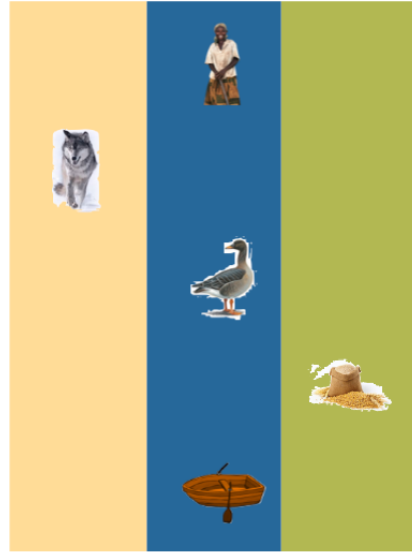
How can we use logic to specify the transitions?





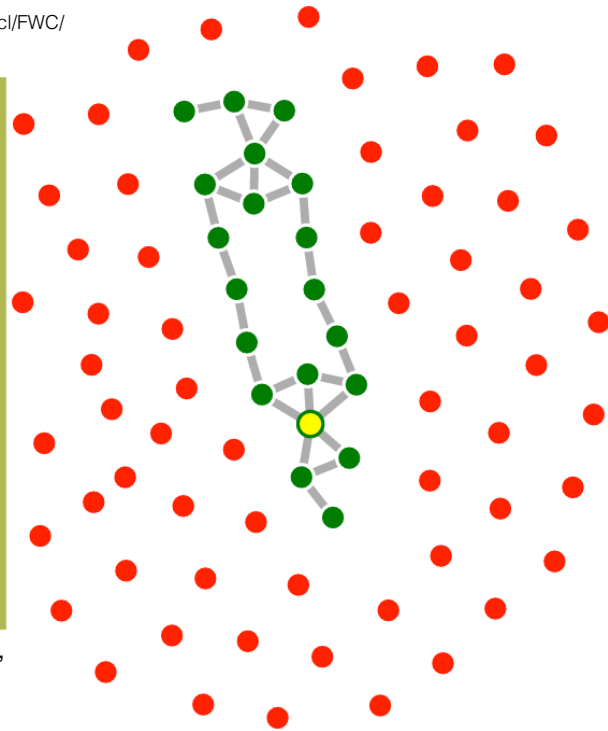
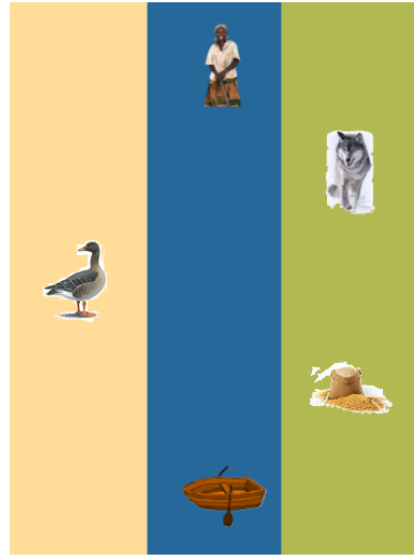
A farmer has to get a wolf, a goose, and a sack of corn across a river.

How can we use logic to specify the transitions?



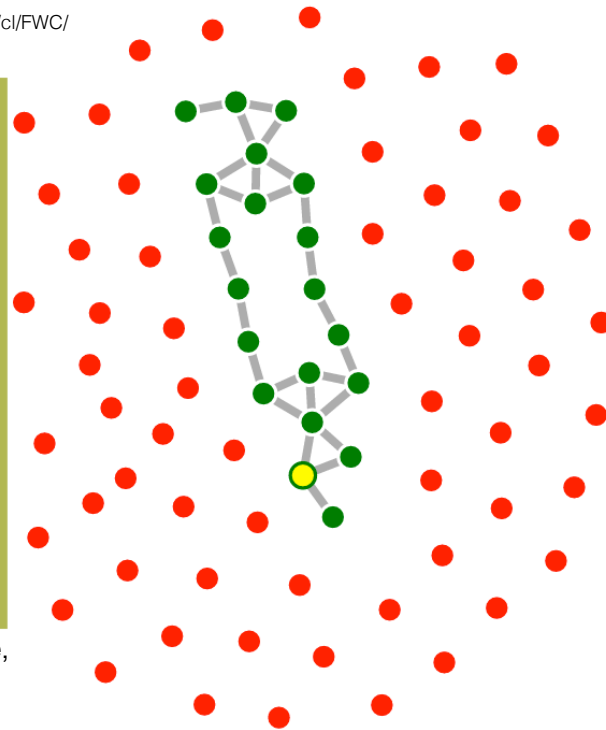
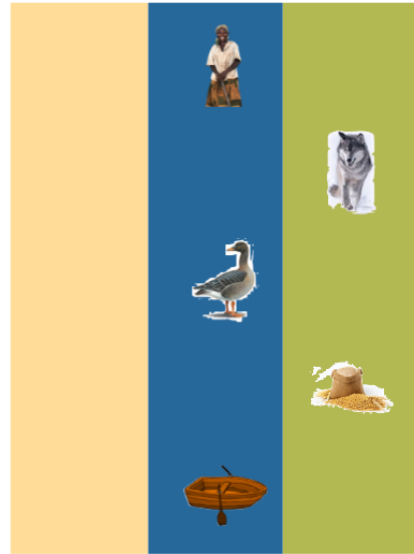
A farmer has to get a wolf, a goose,
and a sack of corn across a river.

How can we use logic to specify the transitions?







A farmer has to get a wolf, a goose, and a sack of corn across a river.

How can we use logic to specify the transitions?








A farmer has to get a wolf, a goose,
and a sack of corn across a river.






How can we use logic to specify the transitions?

	West		East
	WW	WB	WE
	CW	CB	CE
	GW	GB	GE
	FW	FB	FE

one place $(WW \oplus WB \oplus WE) \wedge \neg(WW \wedge WB \wedge WE)$
 not solo $GB \rightarrow FB$
 no conflict $GW \wedge (WW \vee CW) \rightarrow FW$
 no overload $\neg(GB \wedge CB) \wedge \neg(GB \wedge WB) \wedge \neg(WB \wedge CB)$

	West		East
	WW	WB	WE
	CW	CB	CE
	GW	GB	GE
	FW	FB	FE

- one place $(WW \oplus WB \oplus WE) \wedge \neg(WW \wedge WB \wedge WE)$ x4
- not solo $GB \rightarrow FB$ x3
- no conflict $GW \wedge (WW \vee CW) \rightarrow FW$ x2
- no overload $\neg(GB \wedge CB) \wedge \neg(GB \wedge WB) \wedge \neg(WB \wedge CB)$ x1

	West		East
	WW	WB	WE
	CW	CB	CE
	GW	GB	GE
	FW	FB	FE





$$WB \leftrightarrow (\neg WW \wedge \neg WE)$$

⊕	⊥	T
⊥	⊥	T
T	T	⊥

≠	⊥	T
⊥	⊥	T
T	T	⊥

↔	⊥	T
⊥	T	⊥
T	⊥	T

=	⊥	T
⊥	T	⊥
T	⊥	T

	West	East
	$\neg WW$	WE
	$\neg GW$	GE
	GW	GE
	FW	FE

We can use different atoms to model the system.

We introduce 8 atoms whose meanings are given as the negations of the east and west propositions we used earlier.

e.g $WW \leftrightarrow \neg WW$

We can define the old propositions in terms of the new ones:

$$WB \leftrightarrow (WW \wedge WE) \quad \times 4$$

$$WW \leftrightarrow \neg WW \quad \times 8$$

This encoding uses only 8 propositional atoms – 256 states.

The oneplace axiom is now simpler. It eliminates one quarter of the 256 states, leaving 192.

one place	$\neg WW \vee WE$	$\times 4$
not solo		$\times 3$
no conflict		$\times 2$
no overload		$\times 1$

Give the other axioms in terms of the new atoms.

Traffic Light Signals



RED means 'Stop'. Wait behind the stop line on the carriageway



RED AND AMBER also means 'Stop'. Do not pass through or start until GREEN shows



GREEN means you may go on if the way is clear. Take special care if you intend to turn left or right and give way to pedestrians who are crossing



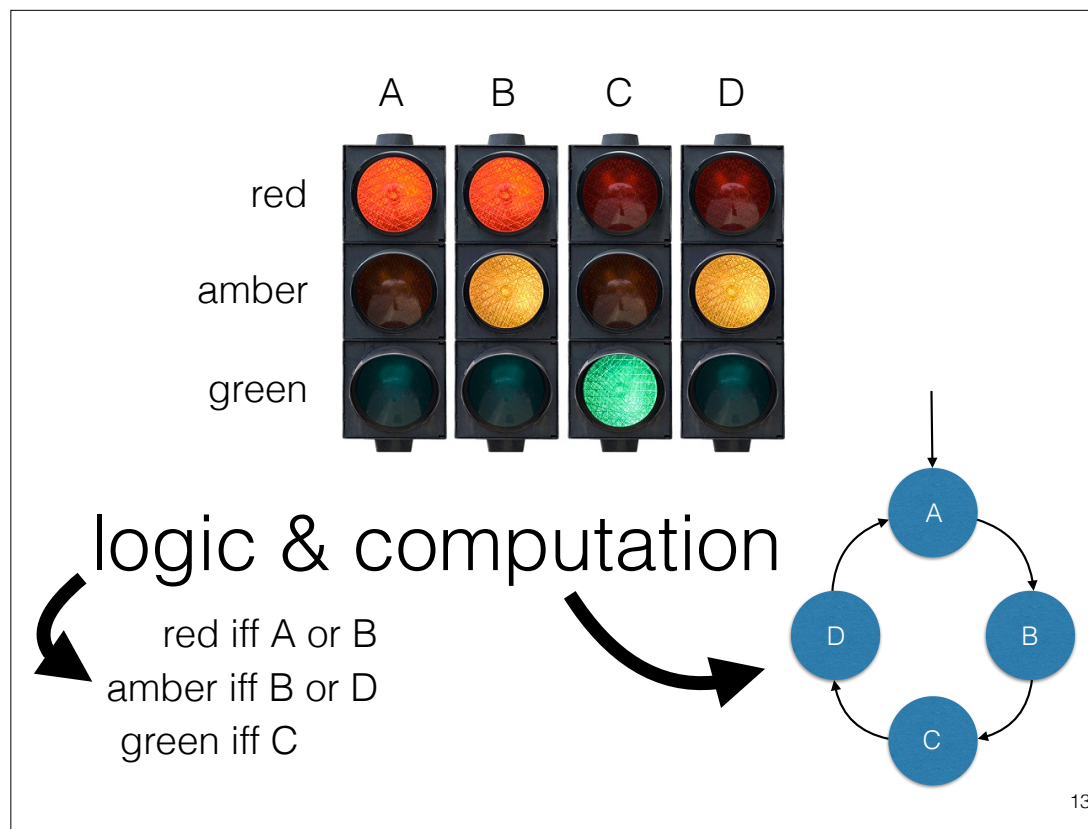
AMBER means 'Stop' at the stop line. You may go on only if the AMBER appears after you have crossed the stop line or are so close to it that to pull up might cause an accident

12

Our example is a traffic light controller, which generates the cyclic sequence of lights stipulated in the Highway Code:

red – red-amber – green – amber

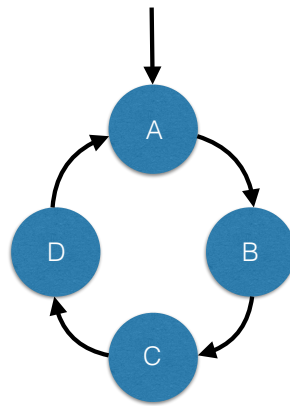
Cars are permitted to proceed when the green light shows; in all other cases they must stop before the white line, if it is safe to do so.



In this course we will introduce the tools required to specify and analyse more complicated examples of such systems. We can describe this simple example as a machine that cycles through four states, with a logical equation for each light that describes the set of states in which that light is on.

“iff” means “if and only if”.

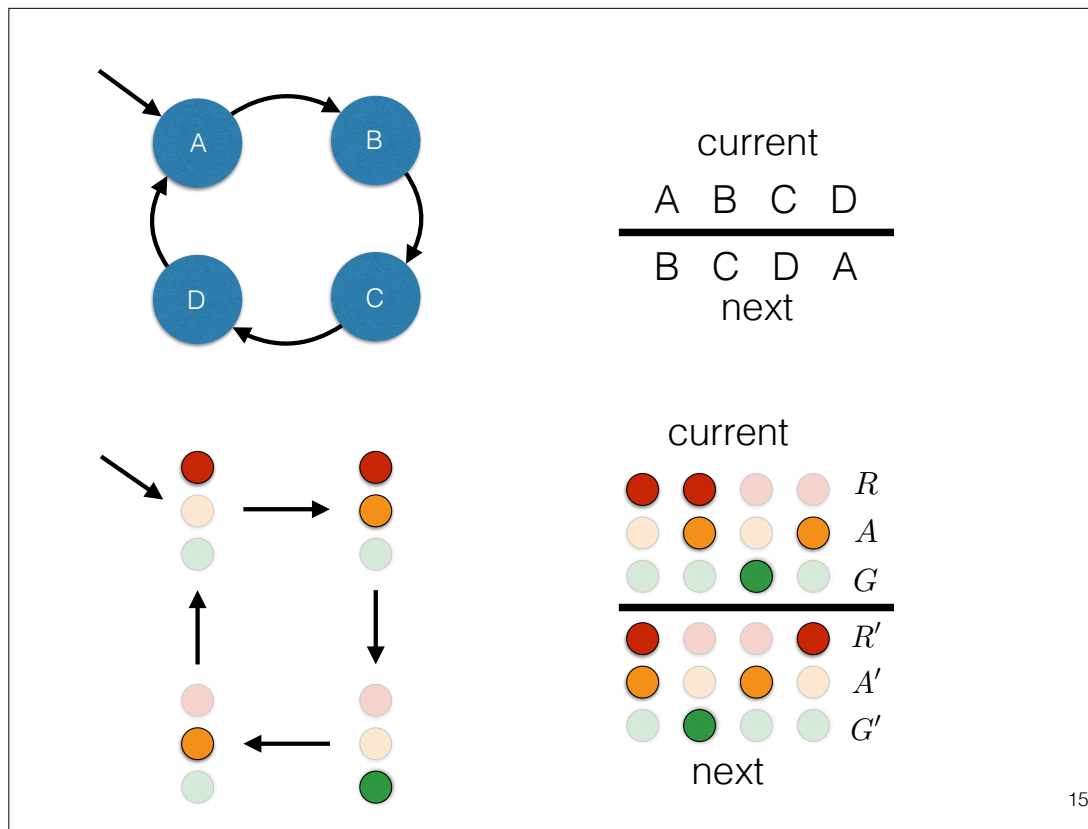
The machine describes a simple ‘computation’: start in state A and cycle through the four states. The logical formulae describe the logic.



	current			
	A	B	C	D
	B	C	D	A
	next			

14

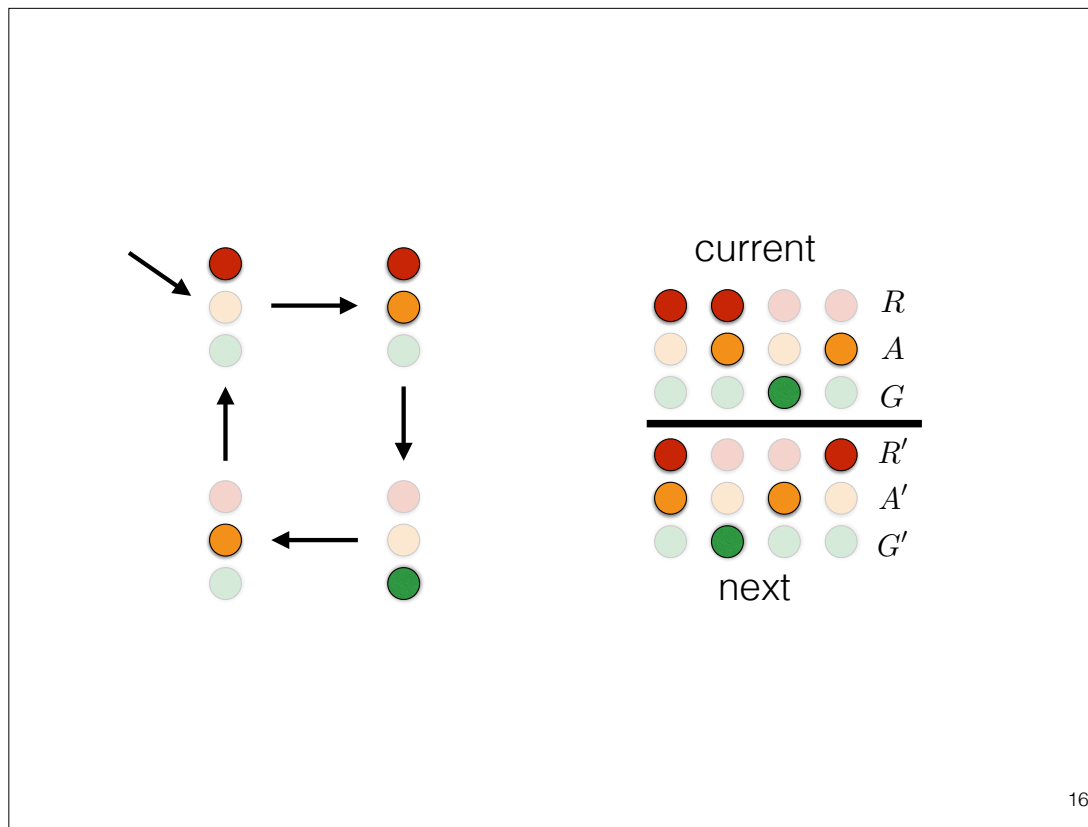
We can draw a *state-transition diagram* (shown to the left of this slide) to describe the permitted sequence of states, or give a next-state table (shown on the right) showing which next state corresponds to each current state.



15

We can draw a *state-transition diagram* (shown to the left of this slide) to describe the permitted sequence of states, or give a next-state table (shown on the right) showing which next state corresponds to each current state.

This system is deterministic. The next state is completely determined by the current state.



16

We can draw a *state-transition diagram* (shown to the left of this slide) to describe the permitted sequence of states, or give a next-state table (shown on the right) showing which next state corresponds to each current state.

$$R' = R \text{ xor } A = R \oplus A$$

$$A' = \text{not } A = \neg A$$

$$G' = R \text{ and } A = R \wedge A$$

R	A	$R \wedge A$	$R \oplus A$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

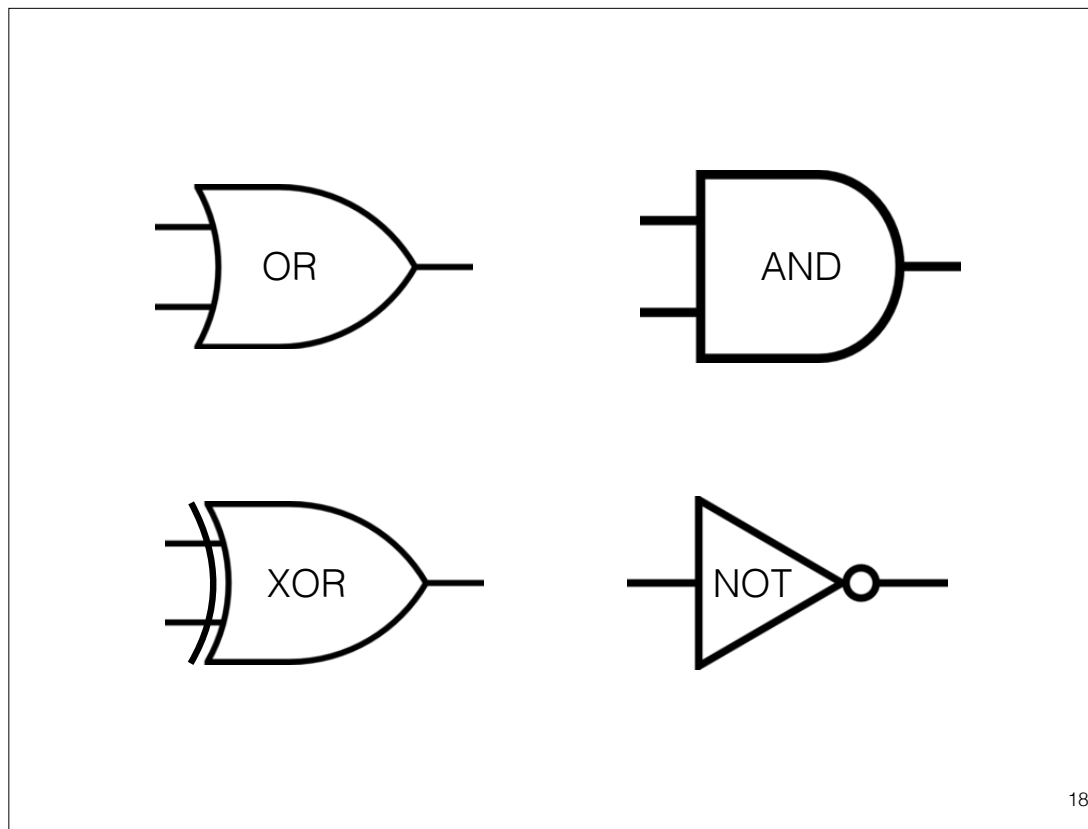
current

next

A	$\neg A$
0	1
1	0

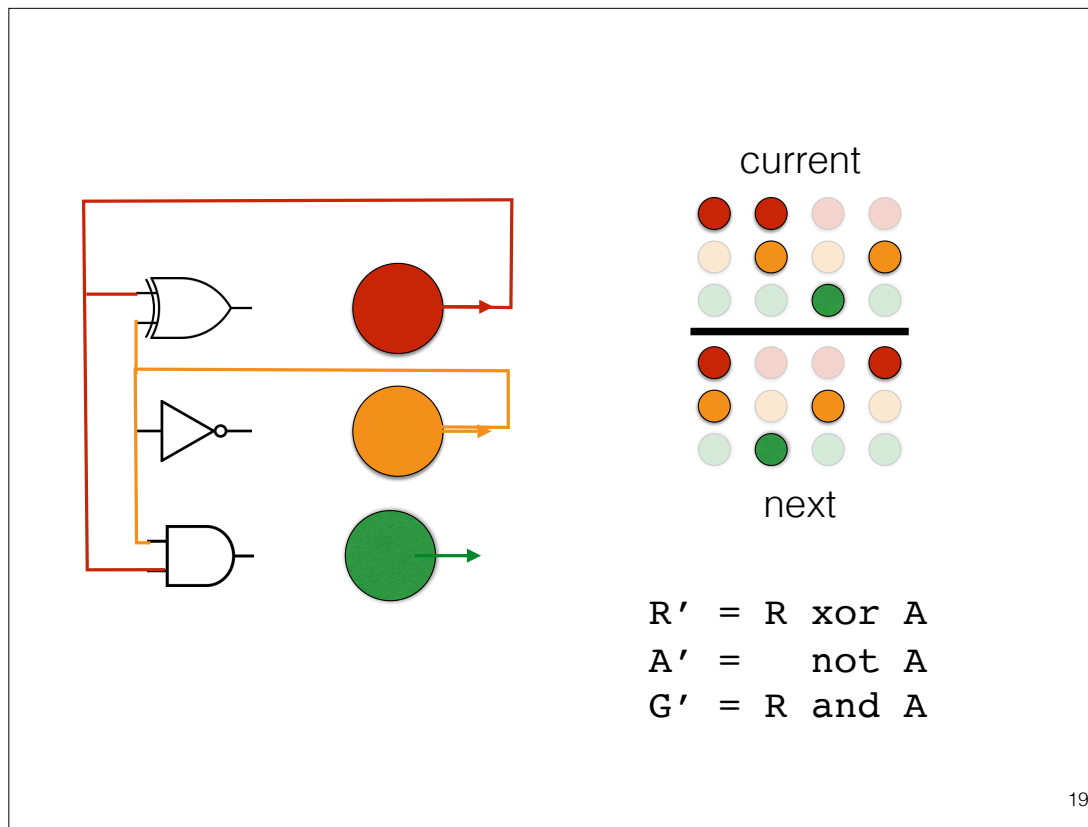
We can describe the next state of the lights in terms of the current state. The state is described by saying which lights are on and which are off.

Let $R A G$ be binary variables, each taking a 0-1 value (zero or one), corresponding to the red, amber and green lights. A value of zero indicates that the corresponding light is off; a value of one indicates that it is on. We write $R' A' G'$ for the next-state variables. Then, for example, the amber light is on in the next state if and only if (iff) it is off in the current state. We can write this as an equation, $A' = \text{not } A$, where *not* is the operation defined by the truth table: $\text{not } 1 = 0$; $\text{not } 0 = 1$.



The computation of the next state can be implemented by some basic *logic gates*. These are circuits that take signals representing binary values as inputs (on the left of each gate in our diagram) and produce a signal representing the output value specified by the relevant truth table.

The symbols are idealisations the actual circuits may have other connections, for example, to provide power.

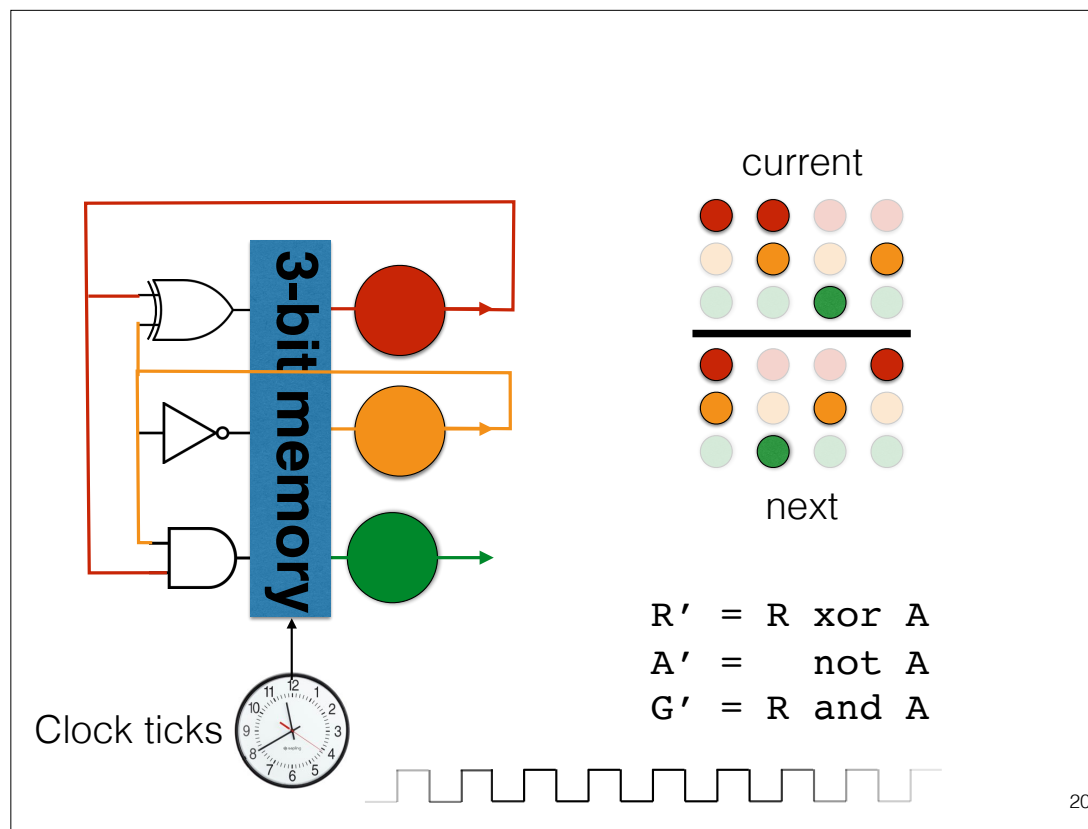


19

The next-state logic for sequencing our traffic lights can be implemented using three different gates. Many different technologies can be used to implement logic gates, some may use high and low voltages to represent binary values, others might use currents, but this logical description of our circuit provides a common abstract level of design.

In our diagram, the current state is stored in the three coloured discs. The outputs of the three gates represent the next state. To make the state transition we need to replace the current state by the next state.

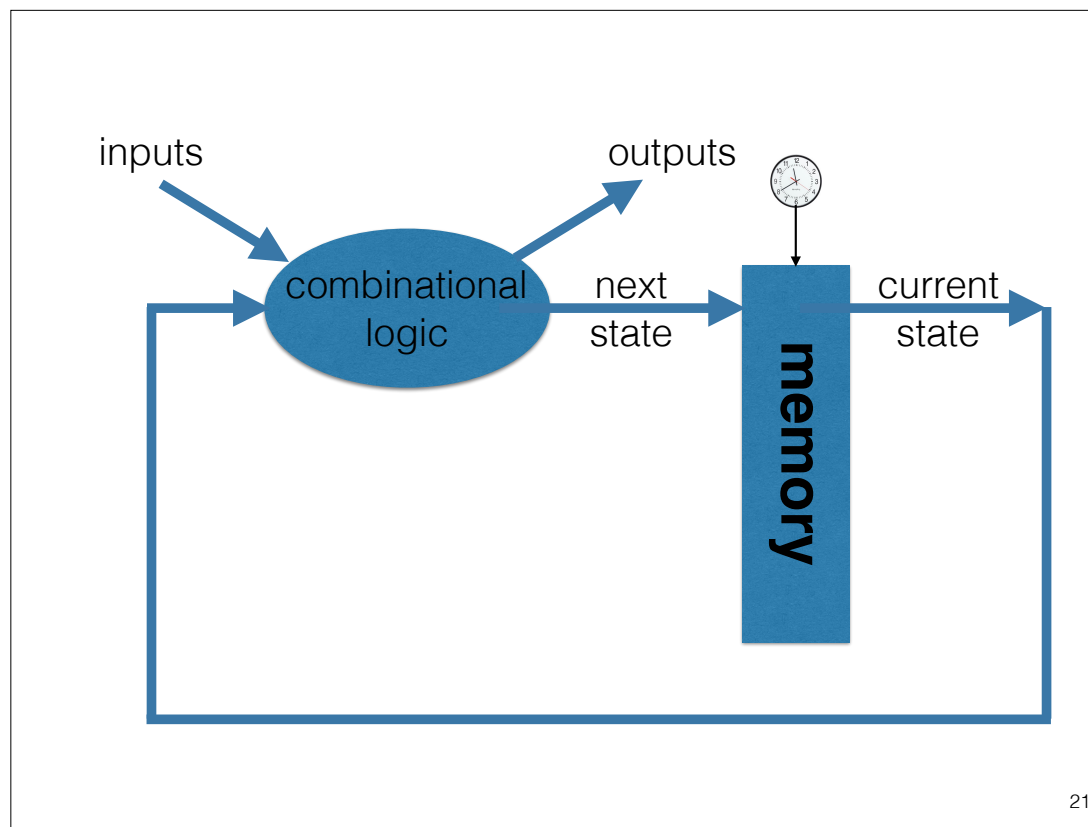
We need memory. One simple form of memory is a *latch*, a special kind of circuit with two inputs, *data* and *clock*. When the clock ticks the current input data value is loaded and stored. The stored value is output, and does not change until the next tick of the clock.



The next-state logic for sequencing our traffic lights can be implemented using three different gates. Many different technologies can be used to implement logic gates, some may use high and low voltages to represent binary values, others might use currents, but this logical description of our circuit provides a common abstract level of design.

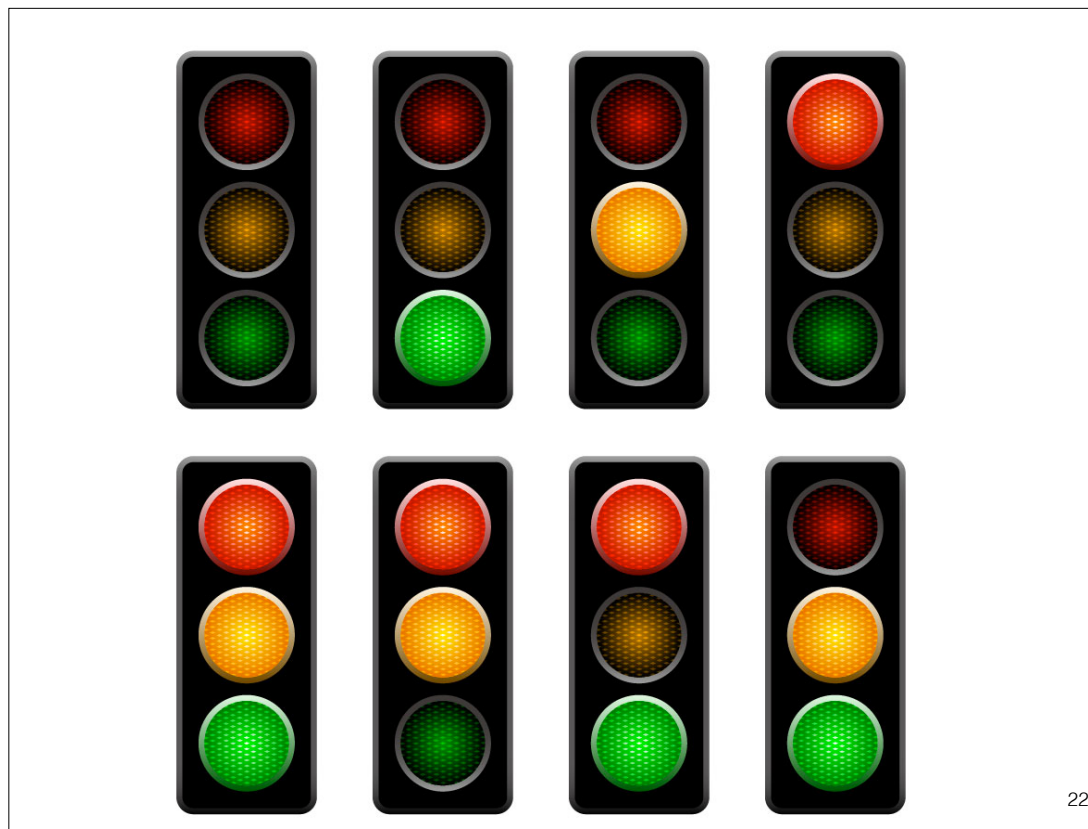
In our diagram, the current state is stored in the three coloured discs. The outputs of the three gates represent the next state. To make the state transition we need to replace the current state by the next state.

We need memory. One simple form of memory is a *latch*, a special kind of circuit with two inputs, *data* and *clock*. When the clock ticks the current input data value is loaded and stored. The stored value is output, and does not change until the next tick of the clock.



21

This gives us one basic architecture for implementing a *finite-state machine*. This is a clocked circuit. Our clock is digital: it issues a discrete series of ticks. A memory stores the current state. At each tick of the clock, the next state is loaded into memory, and becomes the current state. A combinational logic circuit computes the next state and outputs from the current state and inputs. It takes some time for the next state to be computed. The loading of the memory must be completed before this happens, to avoid conflict and confusion. Furthermore, the next clock tick should only come after the computation is completed. So, some delay in the combinational logic is essential, to allow time for the memory to be loaded before the new values occur, and the timing of the next tick of the clock must allow ample time for this delay.

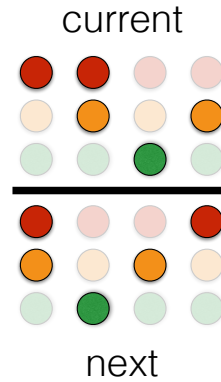
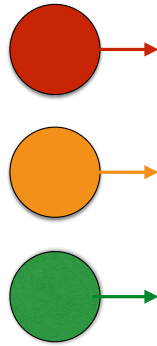


Since there are three lights, there are actually eight possible states for the signal. If we look back at our logic design, we see that only the values of R and A are used to compute the next state.



In real life, things get much more complex. One of the things we will start to discuss later in this course is how to describe and analyse more complex machines.

Exercise 1.2



$$\begin{aligned}R' &= R \text{ xor } A \\A' &= G \text{ or } (R \text{ and not } A) \\G' &= R \text{ and } A\end{aligned}$$

24

Slide 19 shows an implementation of the traffic light controller.

We could have designed our logic differently.

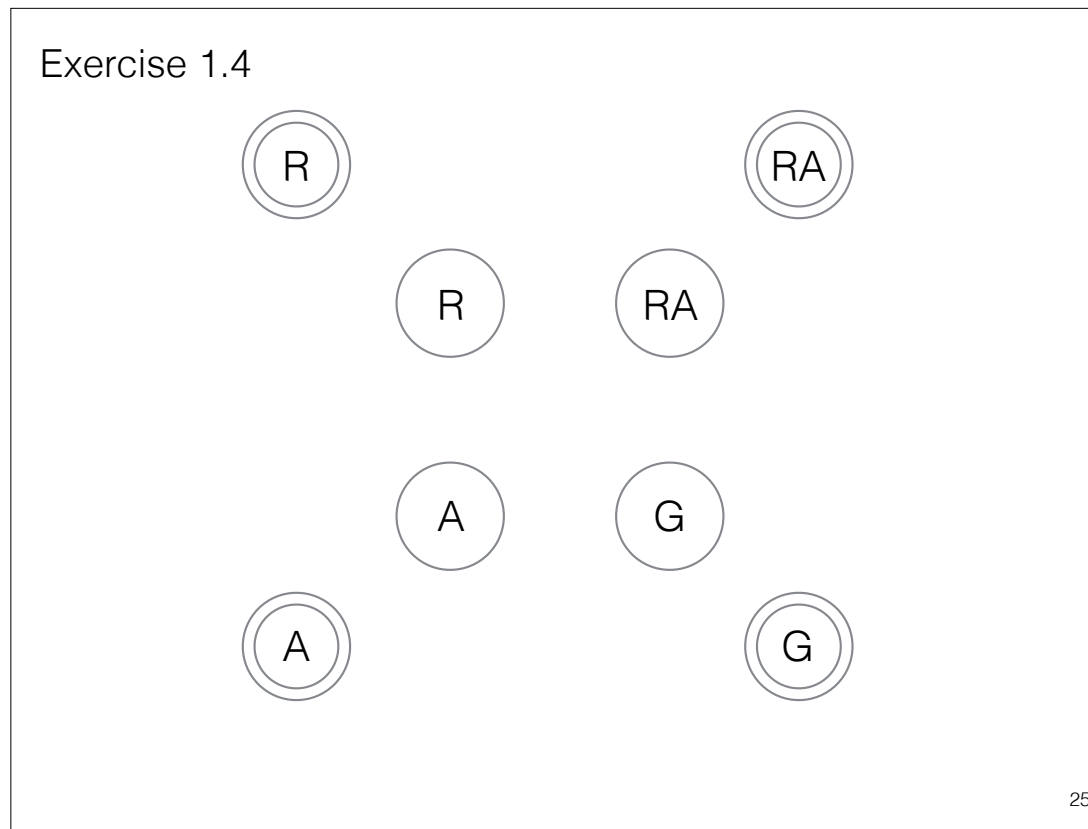
For example, letting

$$A' = G \text{ or } (R \text{ and not } A).$$

Draw the circuit for this implementation.

Is this a correct implementation of the controller? Explain your answer.

Exercise 1.4



As discussed in the lecture, the diagram represents the beginnings of a refinement of our description of the traffic light controller. We model a sensor that detects a car ready to pass the light. For each state of the lights, (R, RA, G, A) we have two states, one (with a double circle) where there is a car, and the other, without a car, as before.

Draw arrows to indicate state changes that still obey the correct sequence for the lights, but also respect the following two rules.

1. A car can only pass the light if it is green.
2. The light only changes from red to red-amber when a car is detected

Optional: You may also design the logic for the controller.

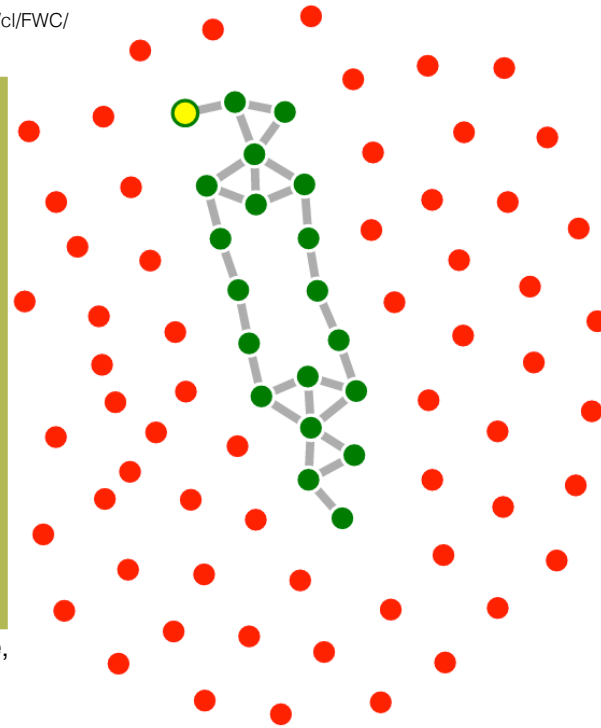
Use a new boolean variable C to represent the presence of a car, and give equations for R', A' and G'.

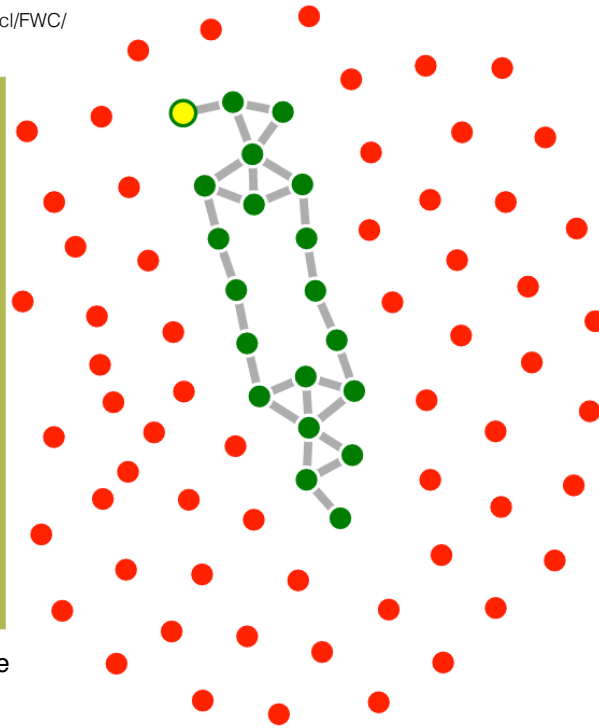
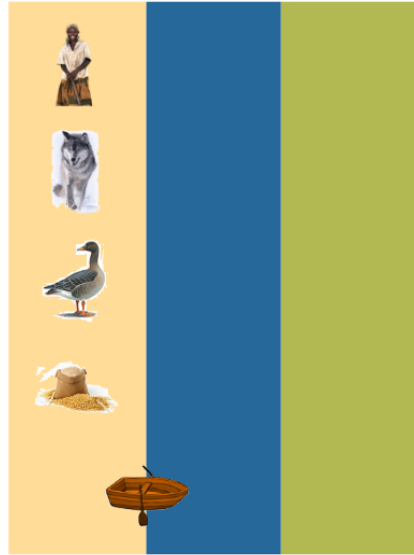
Should we also give an equation for C' ?



A farmer has to get a wolf, a goose, and a sack of corn across a river.

How can we use logic to specify the transitions?





How can we use logic to specify the transitions?

This is a **non-deterministic** system. We define a next state **relation**.



How can we use logic to specify the transitions?

This is a **non-deterministic** system. We define a next state **relation**.

Again we introduce next state variables WW' etc.

Here we have
 $FW \wedge WW \wedge GW \wedge CW$

Is it possible that WE' ?



How can we use logic to specify the transitions?

This is a **non-deterministic** system. We define a next state **relation**.

Again we introduce next state variables WW' etc.

Here we have
 $FW \wedge WW \wedge GW \wedge CW$

Is it possible that WE' ? **NO**

One thing true in our model is that
 $WE' \rightarrow WE \vee WB$

What else do we need to say to give a complete description ?

What does it mean for a description to be complete?