

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 first lecture we look at an 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.

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

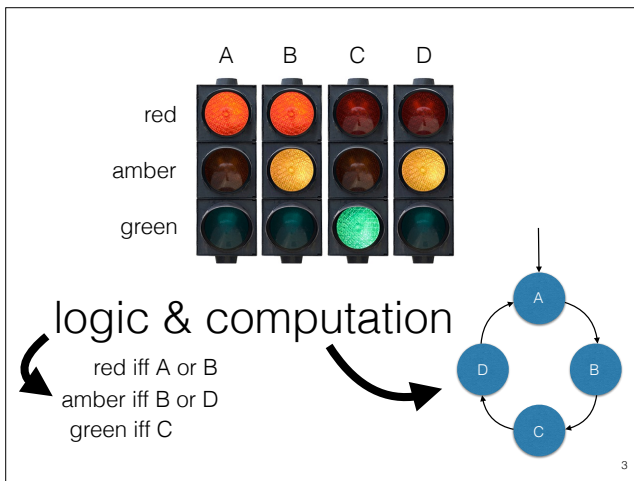
2

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.

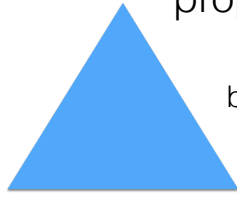
We can describe this simple example as a machine that cycles through four states, with a logical equation for each light that describes the set of states in which that light is on.

“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.

Propositional Logic concerns
properties of things



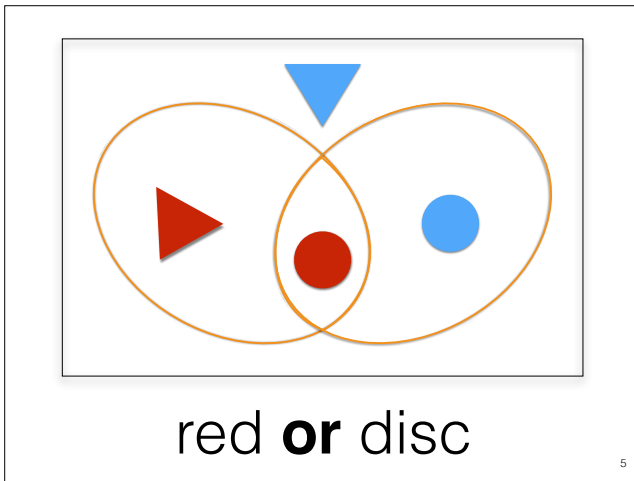
big blue triangle



small red disc

4

For the first part of this lecture, we consider a very simple 'world', where everything is either red or blue, either big or small, and either a triangle or a disc. Moreover, there is one, and only one thing of each type: only one big blue triangle, only one small red disc, and so on ...

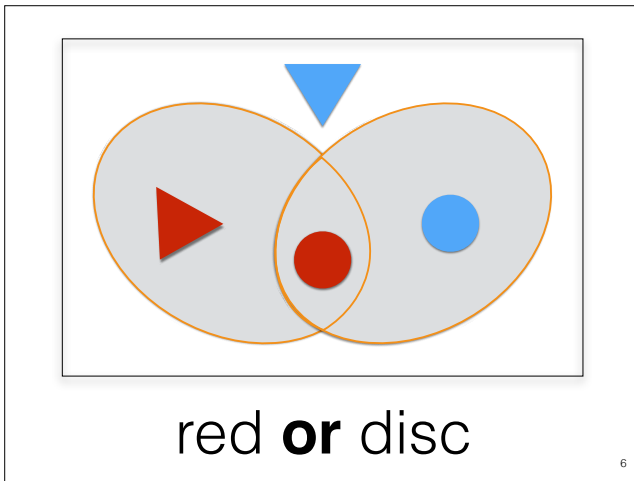


There are only four small things, all shown in this diagram.

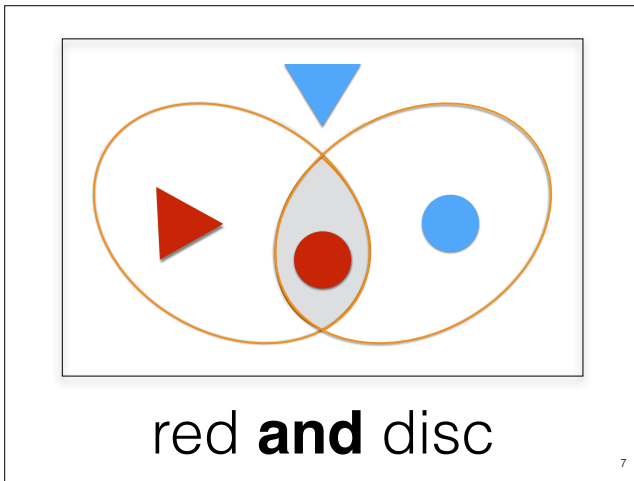
The diagram also includes two circles, representing sets of things.

Each of these sets is defined by a property.

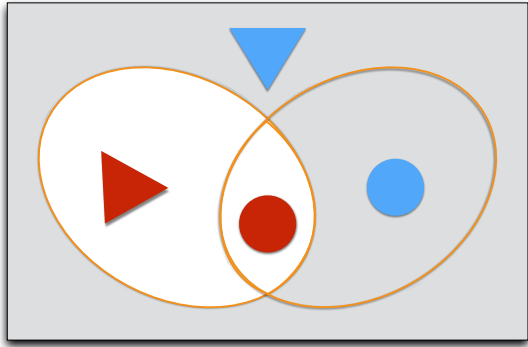
One represents the set of small red things, the other represents the set of small discs.



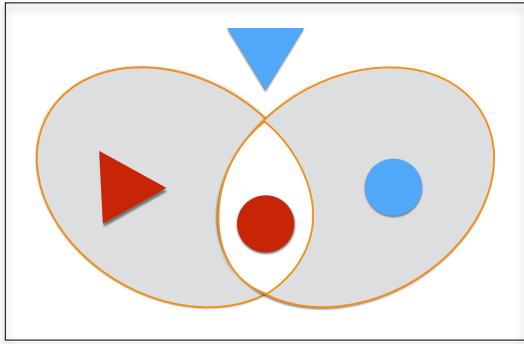
Regions of the diagram correspond to logical combinations of properties.



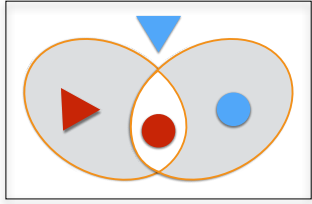
Regions of the diagram
correspond to logical
combinations of properties



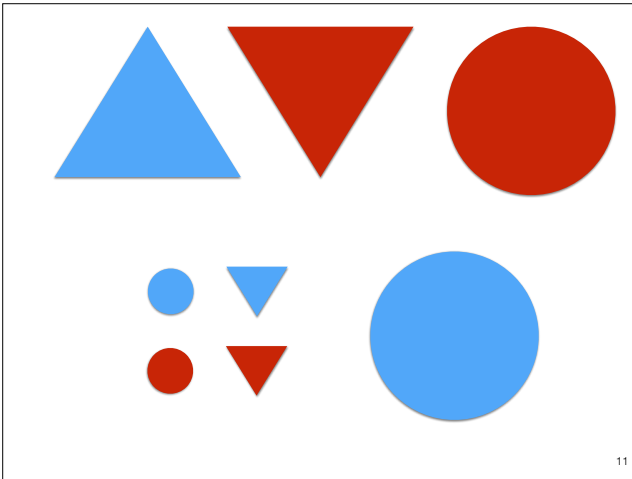
not red



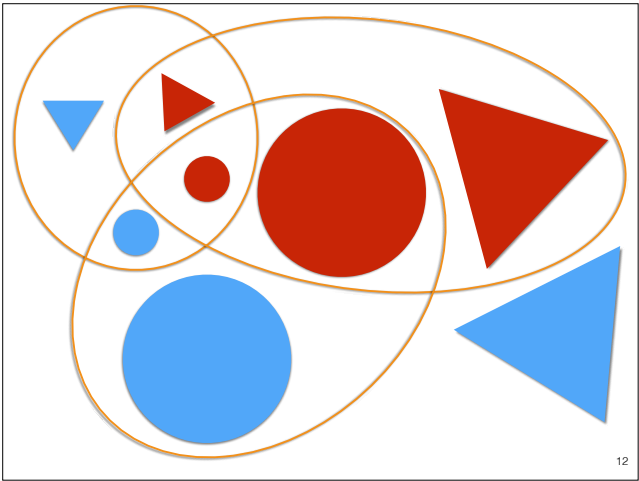
red **xor** disc



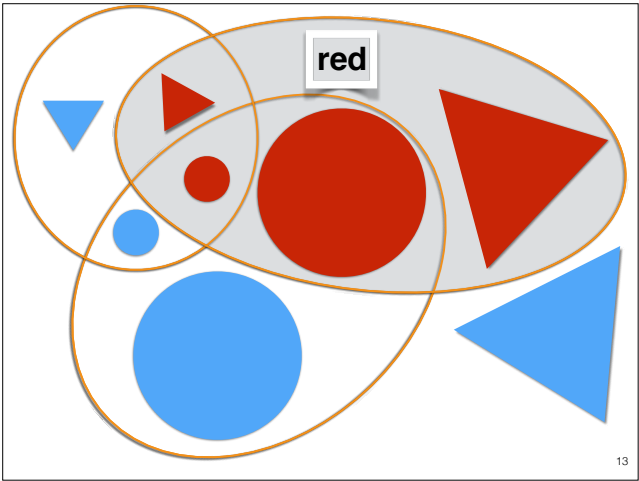
(red **or** disc) **and**
not (red **and** disc)
=
red **xor** disc



If everything is
either red or blue (not red)
and
either small or big (not small)
and
either disc or triangle (not
disc)
then we have $8 = 2 \times 2 \times 2$
possible combinations of
three Boolean properties.



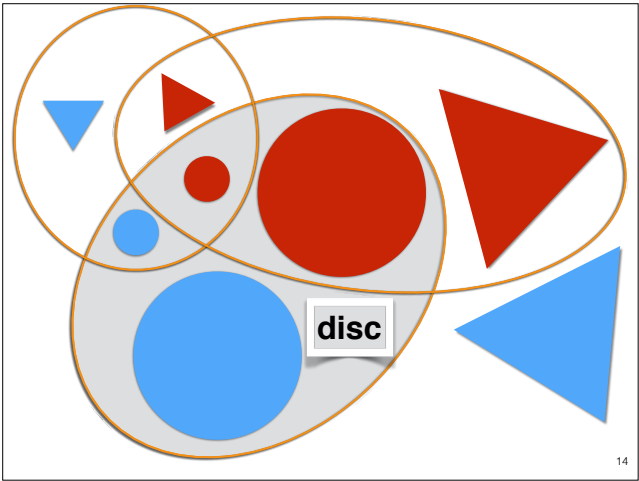
The corresponding Venn diagram has eight regions.



Each circle corresponds to a basic proposition.

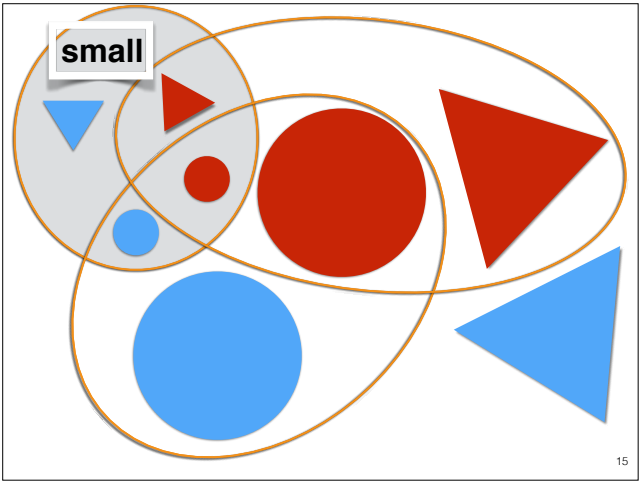
red

Each circle includes four of the eight regions



Each circle corresponds to a
basic proposition.

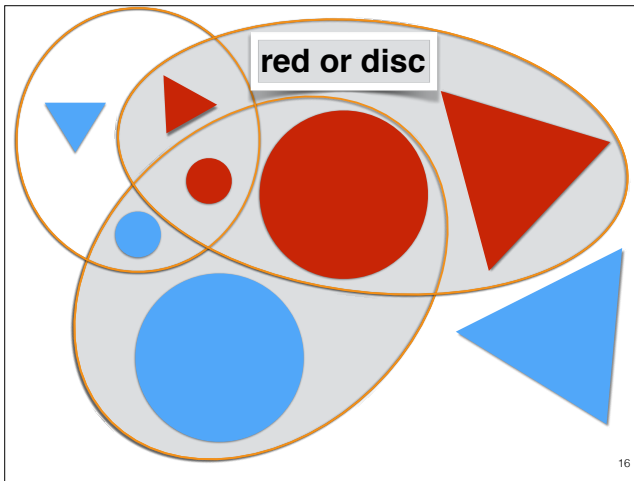
disc



Each circle corresponds to a basic proposition.

small

Each circle includes

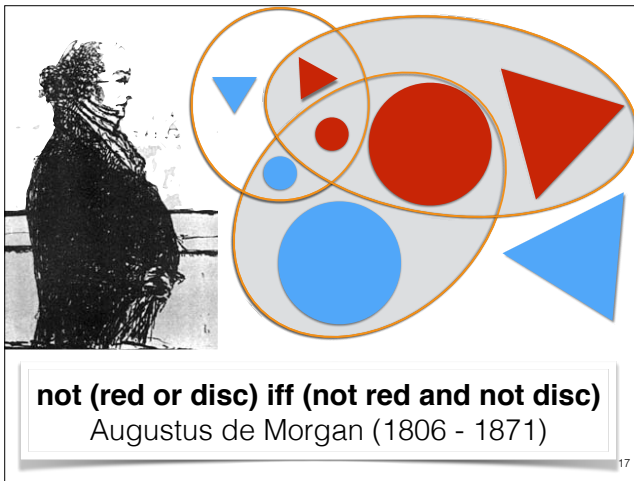


A complex proposition corresponds to a set of regions.

red or disc

This example includes six of the eight regions

The blue triangles, which are not red and not disc, are excluded.



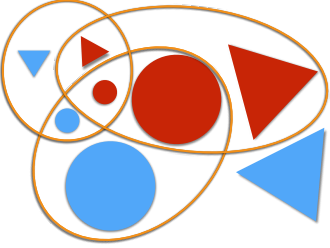
A complex proposition corresponds to a set of regions.

red or disc

This example includes six of the eight regions

The blue triangles, which are not red and not disc, are excluded.

Exercise 1.1



There are 8 regions in the diagram. How many subsets of this set of 8 regions are there?

Given any subset of the eight regions can you write a complex proposition to which it corresponds (using **and**, **or**, and **not** as connectives)?

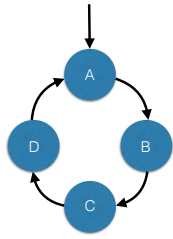
18

A complex proposition corresponds to a set of regions.

red or disc

This example includes six of the eight regions

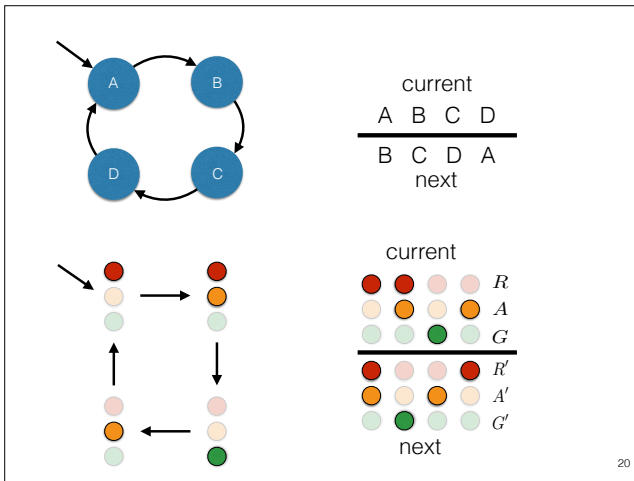
The blue triangles, which are not red and not disc, are excluded.



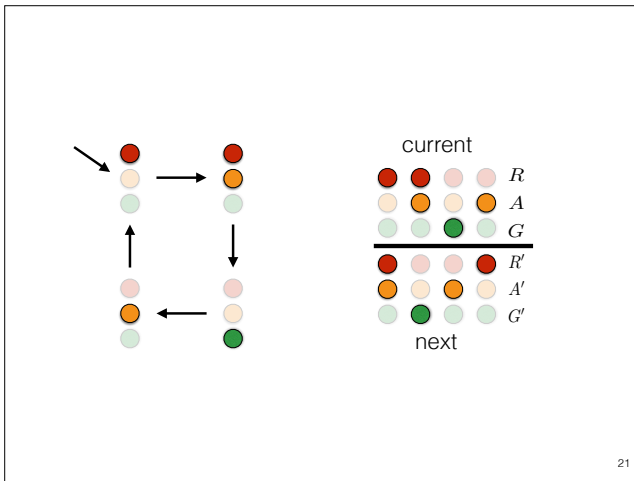
current			
A	B	C	D
B	C	D	A
next			

19

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.



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.



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

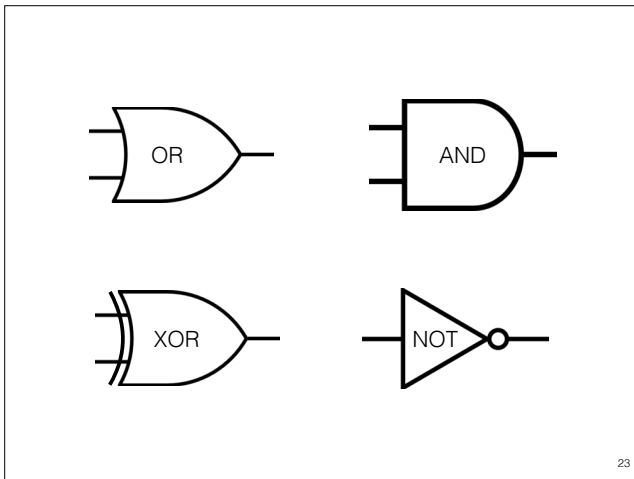
next

A	$\neg A$
0	1
1	0

22

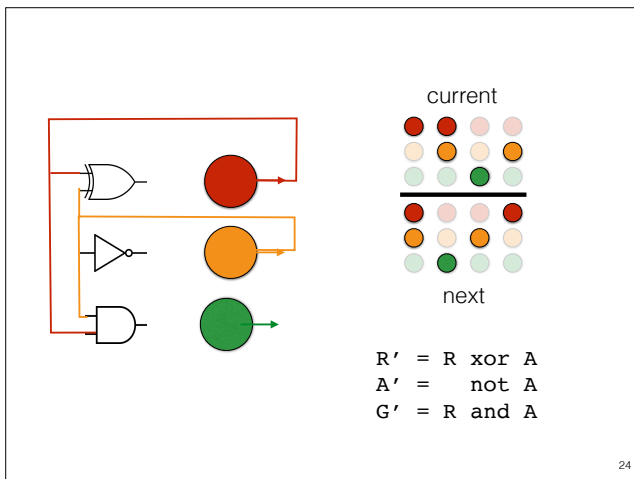
We can describe the next state of the lights in terms of the current state. The state is described by saying which lights are on and which are off.

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.

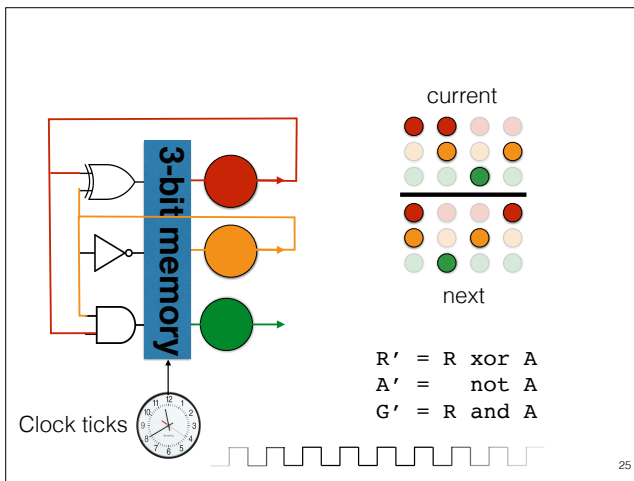
The symbols are idealisations the actual circuits may have other connections, for example, to provide power.



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.

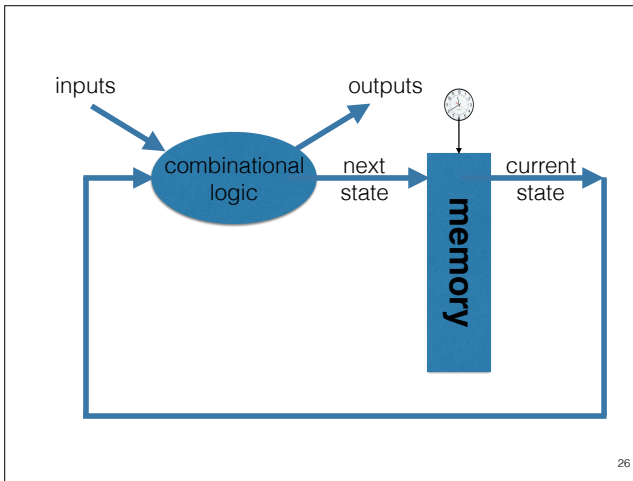
We need memory. One simple form of memory is a *latch*, a special kind of circuit with two inputs, *data* and *clock*. When the clock ticks the current input data value is loaded and stored. The stored value is output, and does not change until the next tick of the clock.



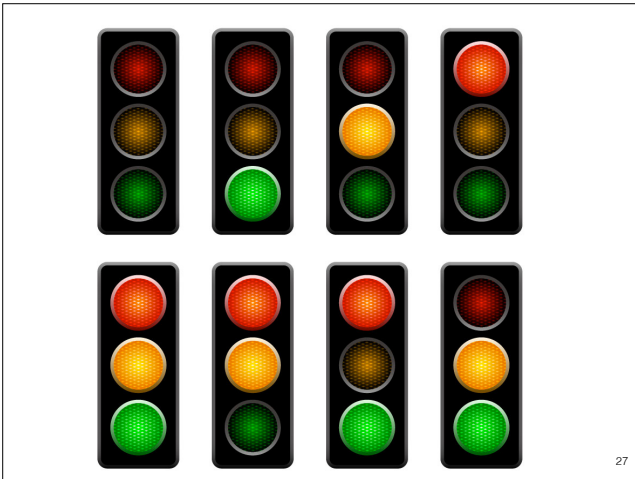
The next-state logic for sequencing our traffic lights can be implemented using three different gates. Many different technologies can be used to implement logic gates, some may use high and low voltages to represent binary values, others might use currents, but this logical description of our circuit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



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.

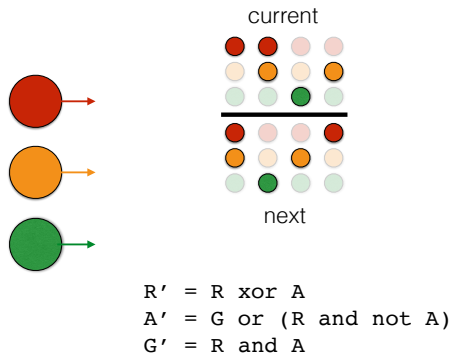


Since there are three lights, there are actually eight possible states for the signal. If we look back at our logic design, we see that only the values of R and A are used to compute the next state.



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



Slide 25 (lecture 1) 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 1.3

The image shows 16 2x2 truth tables arranged in a grid. Five are labeled with boolean expressions:

- $A \vee B$: $\begin{matrix} 01 & 11 \\ 11 & 10 \end{matrix}$
- $A \rightarrow B$: $\begin{matrix} 10 & 11 \\ 11 & 01 \end{matrix}$
- $\neg A$: $\begin{matrix} 11 & 00 \\ 00 & 11 \end{matrix}$
- $A \wedge B$: $\begin{matrix} 10 & 00 \\ 00 & 01 \end{matrix}$
- B : $\begin{matrix} 10 & 01 \\ 10 & 01 \end{matrix}$

The remaining 11 tables are unlabeled and represent other possible binary operations on two variables.

Each of the 16 2x2 tables above represents the truth table of a binary boolean operation.

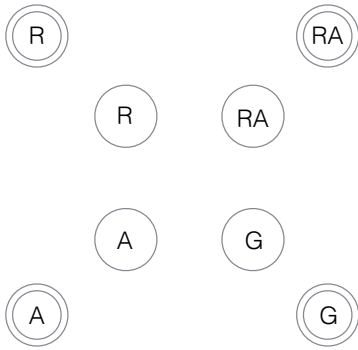
Label each table with a boolean expression for which it is the truth table (five tables are already labelled – begin by checking whether these are correct).

How many of the binary operations actually depend on both variables?

How many depend on only one variable?

How many depend on no variables?

Exercise 1.4



31

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