

# Informatics 1

Computation and Logic



Sets of States: Venn Diagrams and Truth Tables

Michael Fourman  
@mp4man

1

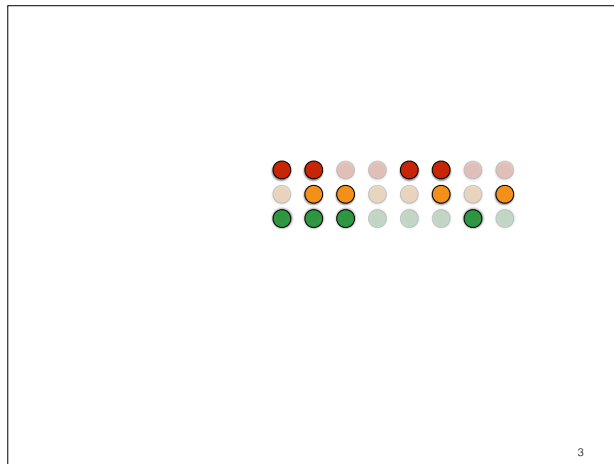
This course provides a first glimpse of the deep connections between computation and logic. We will focus primarily on the simplest non-trivial examples of logic and computation: propositional logic and finite-state machines.

In this lecture we 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.

properties & sets  
boolean circuits  
boolean formulæ  
boolean functions

2

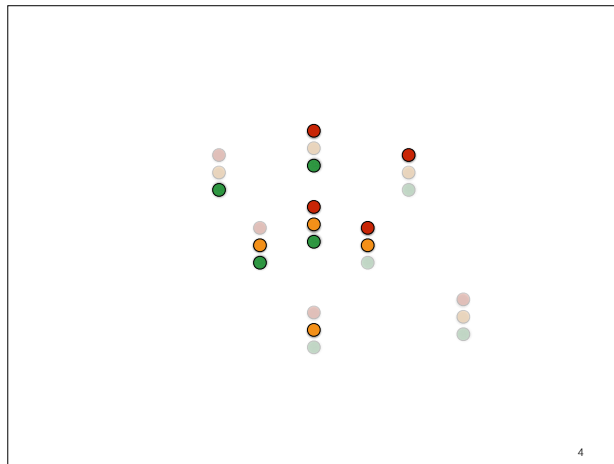
Properties and sets are the things we want to talk about.  
Once we have chosen a language we consider two things that  
have the same properties to be identical.



The possible states of the signal correspond to all possible combinations of lights, even though only four of them should occur in practice.

So there are 8 states.

Our simple language with three variable will allow us to describe any set of states - all 256 of them

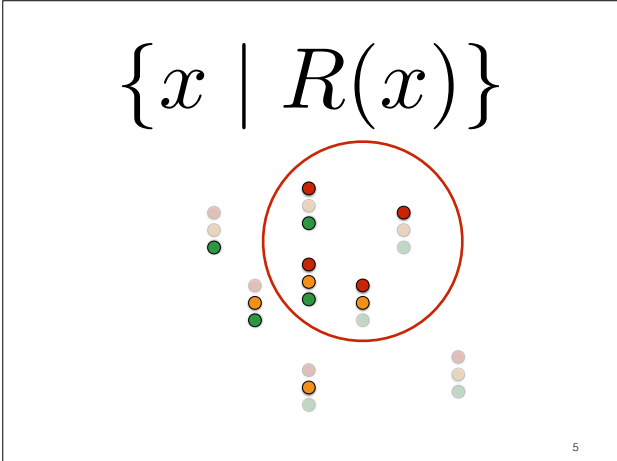


In general, we consider a system with some finite number,  $n$ , of Boolean variables. Here we have three variables RAG reach represented by a light.

The possible states of the system correspond to Boolean valuations of the variables: these are assignments giving a Boolean value for each variable.

Here each light may be on (true) or off (false); we have 8 possible valuations. In general, there will be  $2^n$  valuations.

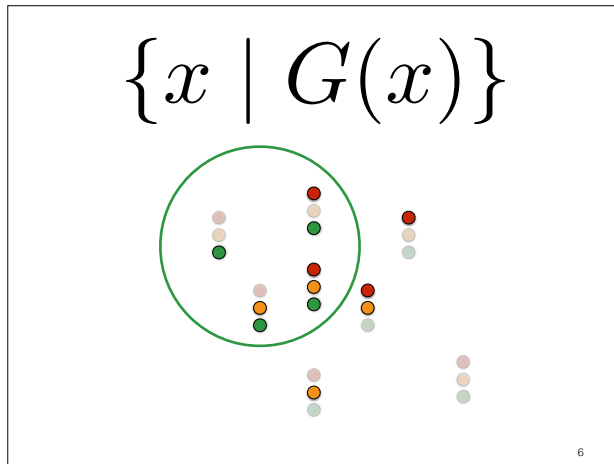
Each of our Boolean variables corresponds to a property of states.



This notation for *set comprehension* will be useful.

Here  $x$  ranges over states.  $R(x)$  is the property that the red light is on.

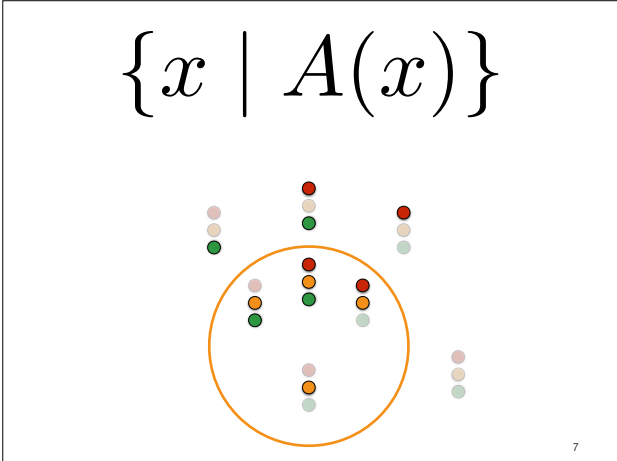
Similarly we have properties  $G(x)$  and  $A(x)$ .



This notation for *set comprehension* will be useful.

Here  $x$  ranges over states.  $R(x)$  is the property that the red light is on.

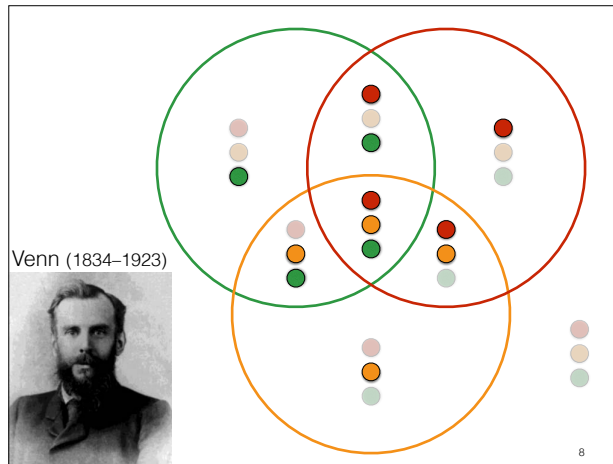
Similarly we have properties  $G(x)$  and  $A(x)$ .



This notation for *set comprehension* will be useful.

Here  $x$  ranges over states.  $R(x)$  is the property that the red light is on.

Similarly we have properties  $G(x)$  and  $A(x)$ .



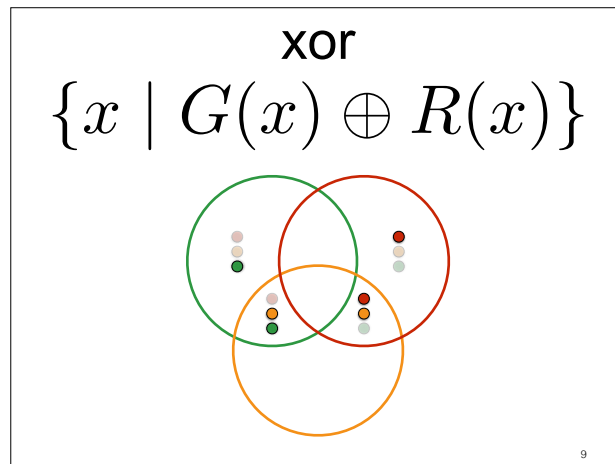
We can place the states in a Venn Diagram.

This includes all eight possible combinations of values for the three Boolean state variables.

For  $n$  larger than 3, the Venn diagram needs more dimensions than most of us can easily visualise.


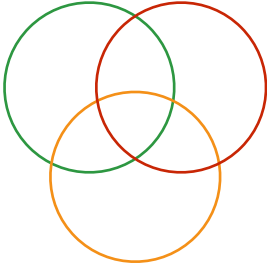
However, the notation of set comprehension can be used for any number of dimensions.





We can compute the set of states corresponding to any expression

**xor**

$$\{x \mid G(x) \oplus (R(x) \oplus A(x))\}$$


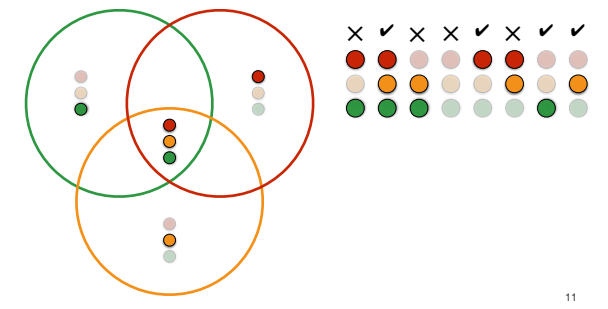
??

10

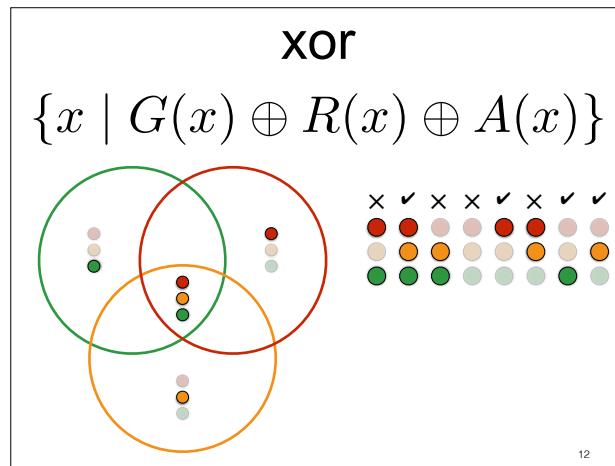
To try in class

**xor**

$$\{x \mid G(x) \oplus (R(x) \oplus A(x))\}$$



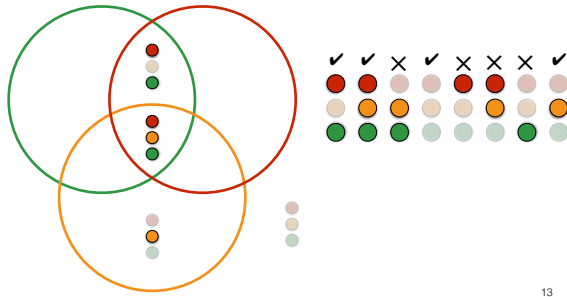
We find that the solution is symmetric, so *xor* is associative.




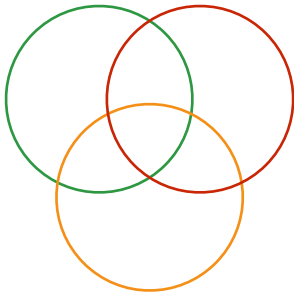
We find that the solution is symmetric, so *xor* is associative.

iff

$$\{x \mid G(x) \leftrightarrow R(x)\}$$



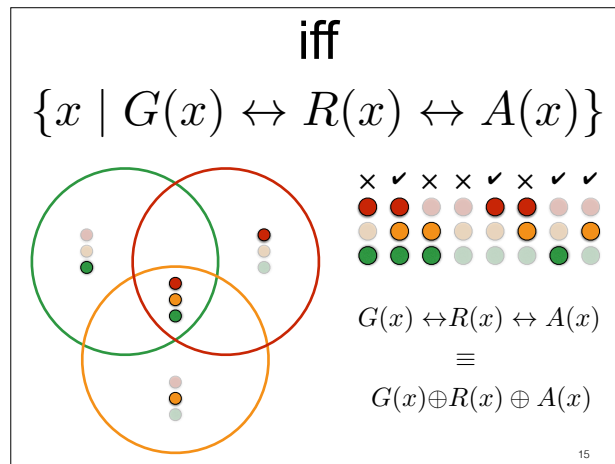
**iff**

$$\{x \mid G(x) \leftrightarrow (R(x) \leftrightarrow A(x))\}$$


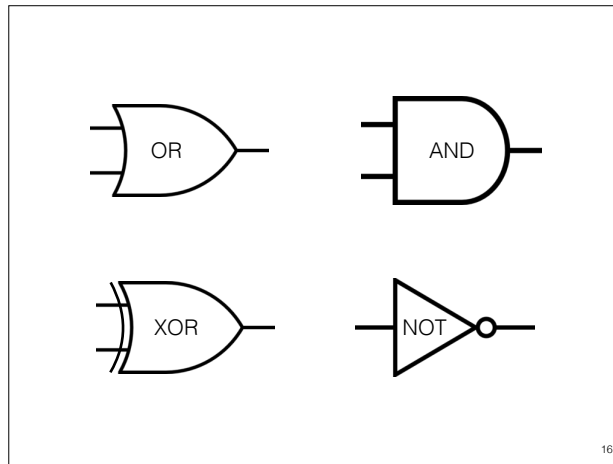
??

14

To do in class



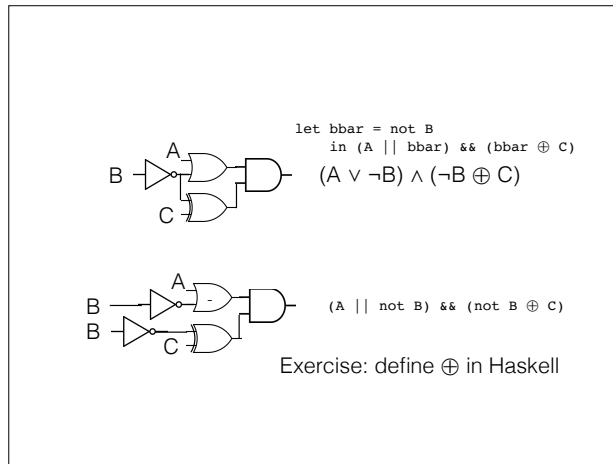
To determine whether two expressions are equivalent, we can check whether they give the same values for all  $2^n$  states of the system. Venn diagram is just a presentation of truth table for two or three variables.



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.



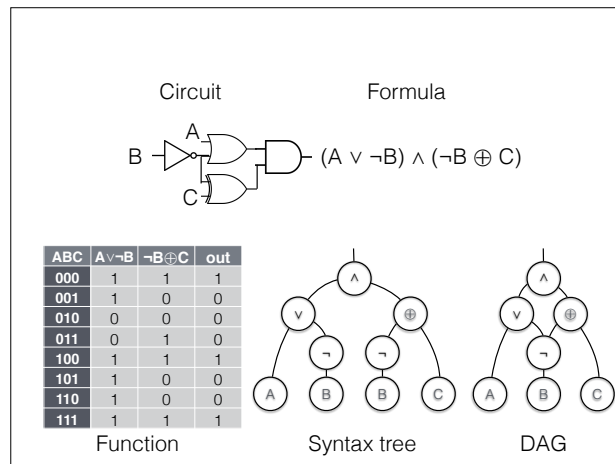


The computation of the next state can be implemented by some basic *logic gates*. These are circuits that take signals representing binary values as inputs (on the left of each gate in our diagram) and produce a signal representing the output value specified by the relevant truth table.

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

The sharing, or reuse, of a computed value corresponds to the `let ... in ...` pattern in Haskell.

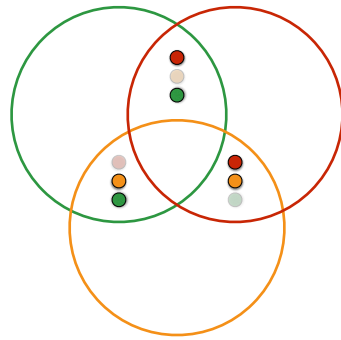
Write a Haskell function to print out the truth table for a boolean function.



Two Boolean circuits or formulæ are equivalent if they compute the same Boolean function. That is, they have the same truth table, or equivalently, they are represented by the same set of valuations. A circuit can express the re-use of a subcomputation, in a way that an expression cannot.

A formula is represented abstractly by a syntax tree. A circuit can be represented abstractly by a Directed Acyclic Graph (DAG)

Find a proposition



??

# Basic Boolean operations

**1, T**

$\vee$

$\wedge$

$\neg$

**0, ⊥**



Boole (1815 – 1864)

true, top  
disjunction, or  
conjunction, and  
negation, not  
false, bottom

## The algebra of sets

$$\mathcal{P}(S) = \{X \mid X \subseteq S\}$$

$$X \vee Y = X \cup Y \quad \text{union}$$

$$X \wedge Y = X \cap Y \quad \text{intersection}$$

$$\neg X = S \setminus X \quad \text{complement}$$

$$0 = \emptyset \quad \text{empty set}$$

$$1 = S \quad \text{entire set}$$