

Data, Math and Methods

Week 6, Math Repeat I



Today

- Repeating what we learned last classes:
 - Tasks with logic formulas
 - And with state machines
- As a programming task we will get back to the State Machines project
 - Programming a demo together in class – then this will be opened for plugging in any state machine model (your task will be to edit it and add yours)

Recap: Logic Formulas

- Tables
- How do we calculate these on a simple example
- Tasks to do – first one together (in Excel/Sheets), rest on your own in groups (trying out Blackboard functionality) – everyone draws their own and then you check with each other in the group – finally you will send me the working version for checking.

Recap: basic logic tables

- Cheat sheet of all the basic logic operators:

not, and, or, implies, equals

a	b	$\neg a$	$a \wedge b$	$a \vee b$	$a \Rightarrow b$	$a \Leftrightarrow b$
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

exclusive or, not and, not or

$a \text{ XOR } b$	$a \text{ NAND } b$	$a \text{ NOR } b$
0	1	1
1	1	0
1	1	0
0	0	0

Implies: statement “if a is true then b is also true”

Logic Formulas – Task 0

- Check if $A \equiv B$ on all its logical evaluations
- \equiv means: logical equivalence (aka their tables are the same!)

A: $a \text{ XOR } b$

B: $(a \vee b) \wedge \neg(a \wedge b)$

Logic Formulas – Task 1

- Check if $A \equiv B$ on all evaluations

1.) A : \top (true on all possible a 's and b 's)

B : $\neg a \vee (a \wedge (a \vee b))$

2.) A : $a \wedge b \wedge c$

B : $a \vee b \vee c$

Logic Formulas – Task 2

- Does “or” \vee distribute? $c \vee (a \wedge b) \equiv (c \vee a) \wedge (c \vee b)$
 - Basically can we sort of multiply with the “ \vee ” operator like this? Please check and prove it with a logical table.

Basically boils down to checking if $A \equiv B$ for:

A: $c \vee (a \wedge b)$

B: $(c \vee a) \wedge (c \vee b)$

PS: the blue color shows how we “multiplied” it with OR

Logic Formulas – Task 3

- We have a long (potentially very long) row with “and” \wedge between many items. The trick to speed up the evaluation is to find one item in the row which would evaluate to False. Then even if everything else is True, the one False will bring the whole formula to False:
 - $\text{True} \wedge \text{True} \wedge \text{True} \wedge \text{True} \wedge \text{True} \wedge \text{False} \wedge \text{True} \wedge \text{True} \wedge \text{True} \wedge \text{True} \equiv \text{False}$

PS: In this case **False** works like 0 in multiplication. In algebra we call it an **absorbing element**.

Logic Formulas – Task 3

- We have a long (potentially very long) row with “and” \wedge between many items. The trick to speed up the evaluation is to find one item in the row which would evaluate to False. Then even if everything else is True, the one False will bring the whole formula to False:
 - $\text{True} \wedge \text{True} \wedge \text{True} \wedge \text{True} \wedge \text{True} \wedge \text{False} \wedge \text{True} \wedge \text{True} \wedge \text{True} \wedge \text{True} \equiv \text{False}$

PS: In this case **False** works like 0 in multiplication. In algebra we call it an **absorbing element**.

Task: find one element which always evaluates as False in this long formula:

$$(a \vee b) \wedge (\neg a \vee b) \wedge (a \wedge \neg a) \wedge (b) \wedge (b \wedge a)$$

Logic Formulas – Task 4

- We have a long (potentially very long) row with “or” \vee between many items. The trick to speed this one up is on the contrary to find a True statement as one of the items. Then any number of False items wouldn't change the evaluation as True:
 - $\text{False} \vee \text{False} \vee \text{False} \vee \text{False} \vee \text{False} \vee \text{False} \vee \text{False} \vee \text{False} \vee \text{True} \vee \text{False} \vee \text{False} \equiv \text{True}$

PS: In this case **True** works like 0 in multiplication. In algebra we call it an **absorbing element**.

Logic Formulas – Task 4

- We have a long (potentially very long) row with “or” \vee between many items. The trick to speed this one up is on the contrary to find a True statement as one of the items. Then any number of False items wouldn't change the evaluation as True:
 - $\text{False} \vee \text{False} \vee \text{False} \vee \text{False} \vee \text{False} \vee \text{False} \vee \text{False} \vee \text{False} \vee \text{True} \vee \text{False} \vee \text{False} \equiv \text{True}$

PS: In this case **True** works like 0 in multiplication. In algebra we call it an **absorbing element**.

Task: find one element which always evaluates as True in this long formula:

$$(a) \vee (b) \vee (\neg a) \vee (a) \vee (\neg a \wedge a) \vee (\neg a \vee a)$$

Recap: State Machines

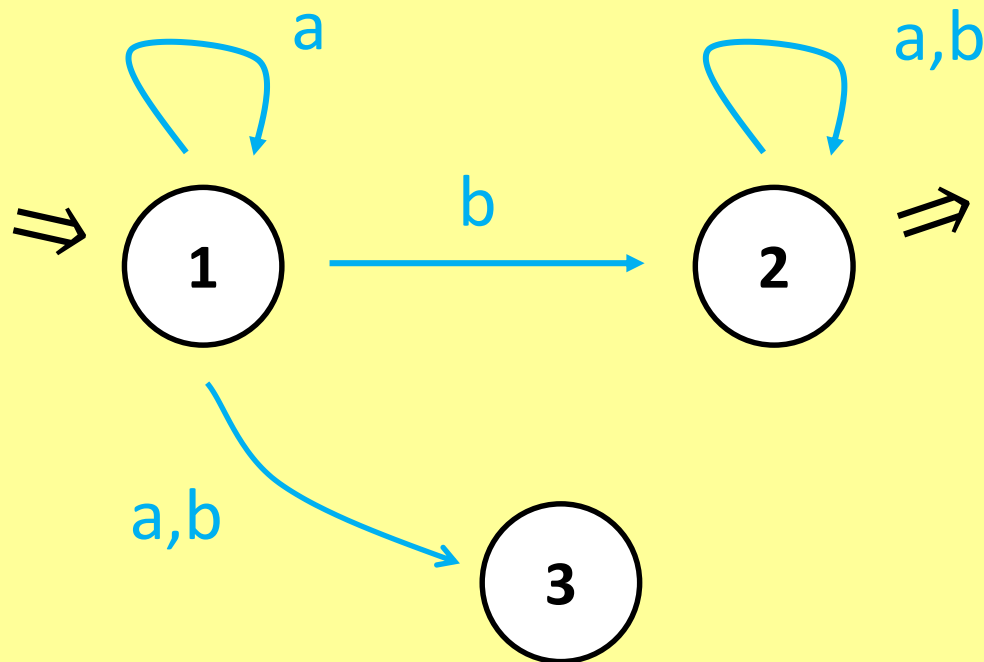
- What they are made of
- Reading a simple example of SM
- Tasks: We do first one together again (in Inkscape), then the rest in the groups – together make sure it's working – finally show me the result

State Machines

- Are made of:
 - States
 - One starting state
 - Few accepting states
 - Transitions
 - Incoming alphabet
- State Machine “accepts” a word if it ends up in an accepting state after reading this word.
- If we can choose where to go with a read letter from the word (this happens in non-deterministic State Machines), then we have to try all the possible paths. If we end in accepting state with at least one of them, then the word is accepted.

State Machines example

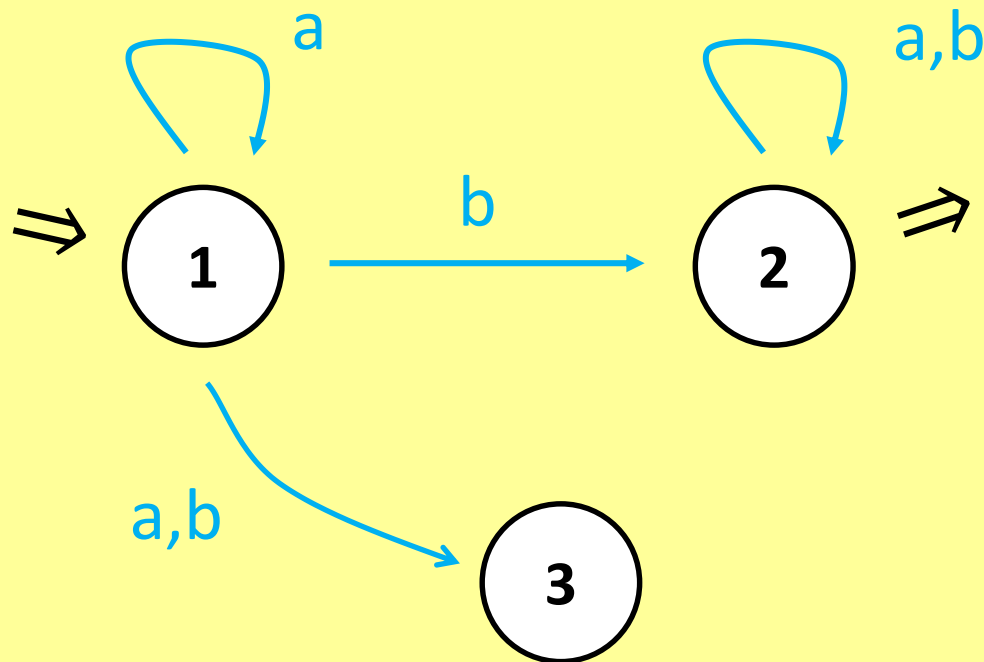
What would this machine do?



What does it do?

State Machines example

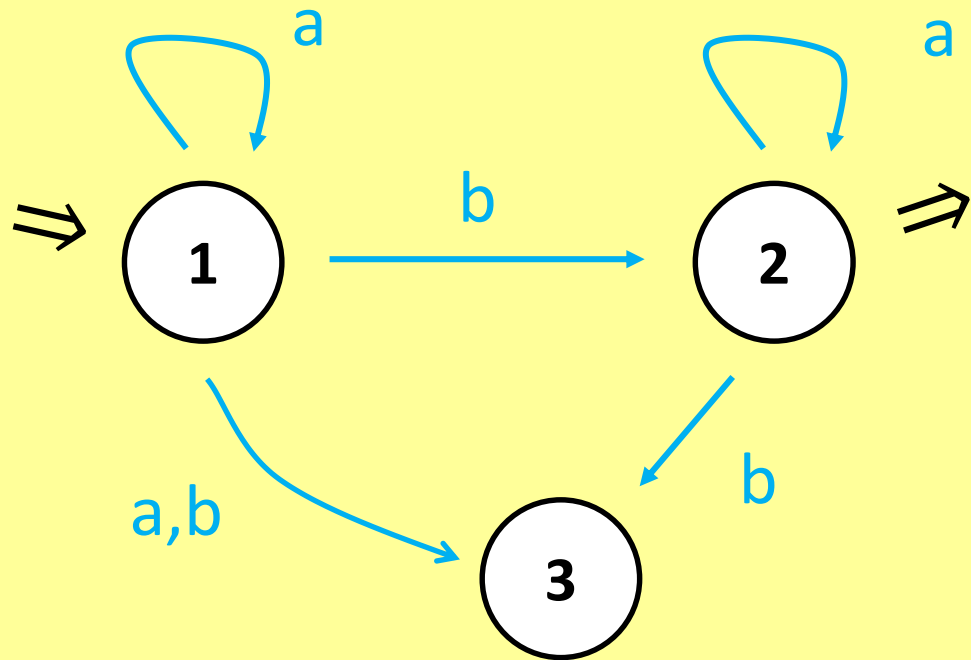
What would this machine do?



Words which contain at least one “b”!

State Machines example

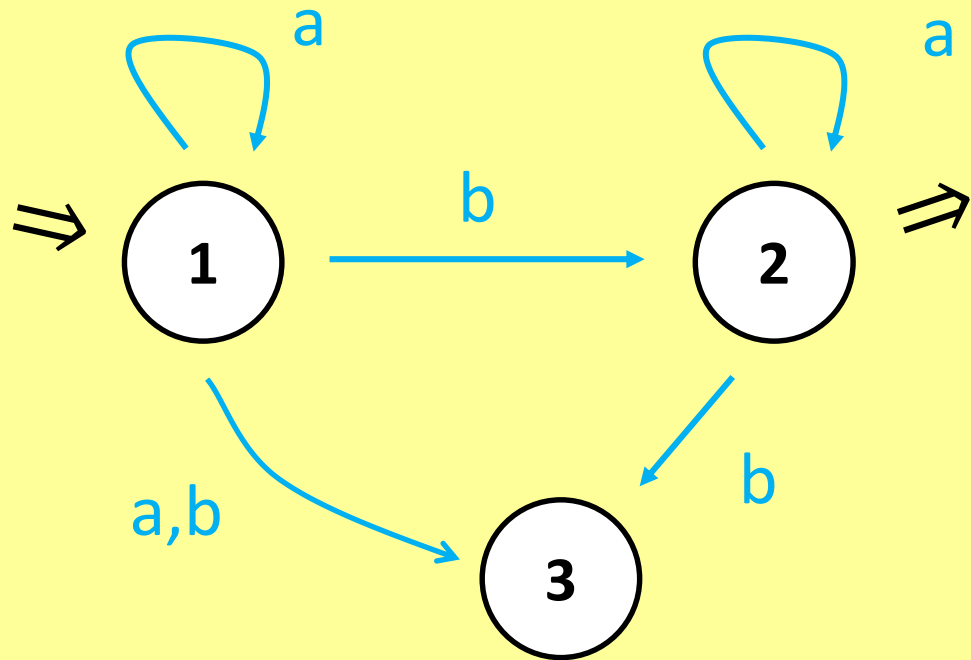
What would this machine do?



And what about this one?

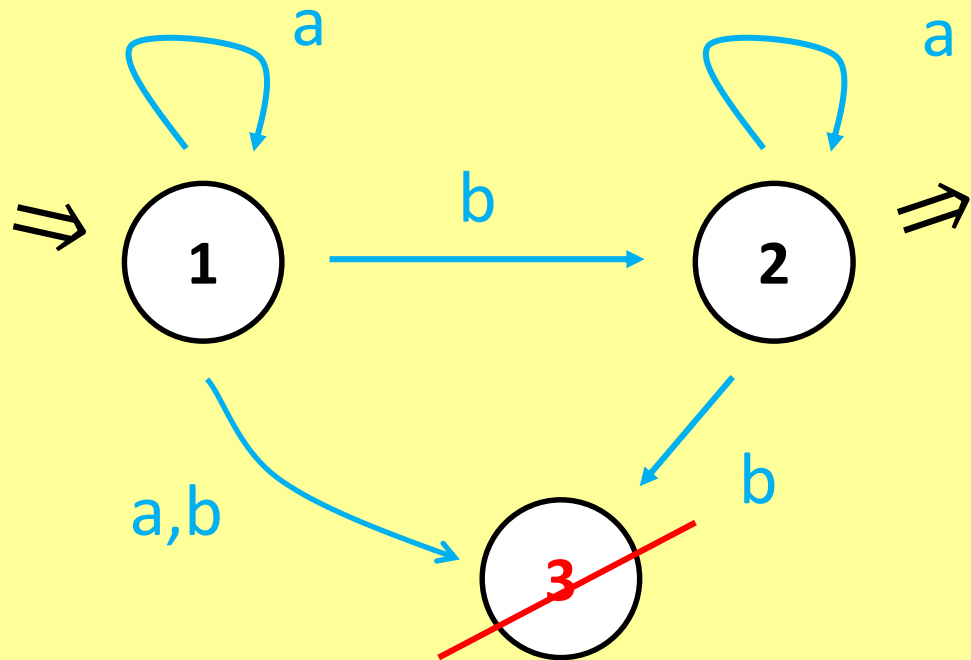
State Machines example

What would this machine do?



Words which contain at exactly one “b”!

Agreement:



Let's agree: When designing your State Machines, you don't have to include all the blind paths. (So **state 3** wasn't necessary)

But feel free to draw them if they help :).

State Machines – Tasks

Unless written differently, the **alphabet** is $[a, b]$.

- Task 1: Accept words of length 3
- Task 2: Machine which accepts the regular expression:
 - a^*bba^*
 - (This means “any repetition of a’s” followed by “bb” and ended with “any repetition of a’s again”)
- Task 3: Machine which accepts pin number - pls use “2020”
 - Here the **alphabet** is $[0,1,2,...9]$

State Machines – Tasks

- Task 4: Accept even amounts of “a”.
 - “aa”, “aaaa”, “aaaaaa” would be accepted.
- Task 5: Accept if number of a’s is less than 3.
 - “” (empty word), “a”, “bbbbbbba”, “bbbbbaa” would be accepted.
 - *Hint: You can have more than one accepting state – and the starting state can also be accepting.*
- Task 6, Question: How do you make a machine accepting the inverse of all words of another machine?

State Machines – Tasks

- Task 7, Bonus: Can we build a State Machine which would accept words with less a's than b's?
 - *Hint: Can we somehow count? And can we do it for in theory infinitely long words?*

Pause 1

Programming

State Machines Project

- The task was: write a python code which would **accommodate any model of state machine and control something with it** (simple examples of outputs were: *colors, sounds in a loop*).
- **Idea**: This goes beyond simple “*press button -> do action*” behavior. We can mimic intelligent behavior with this (remember the Batman game example).

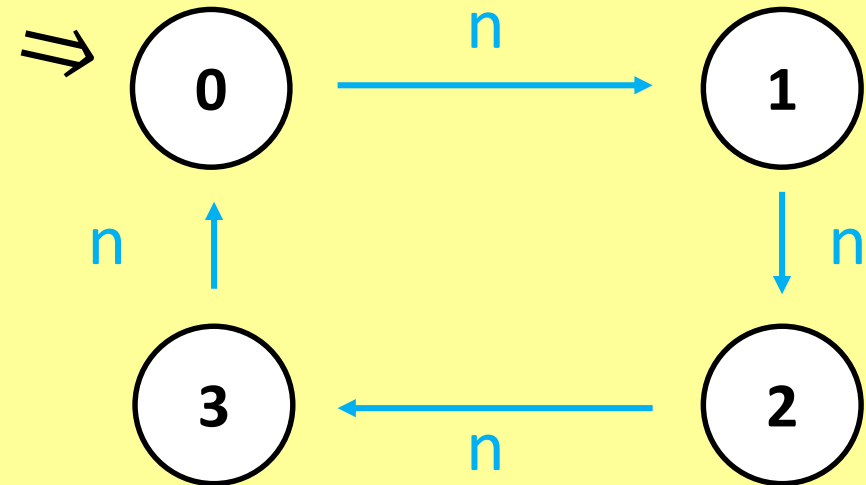
Programming

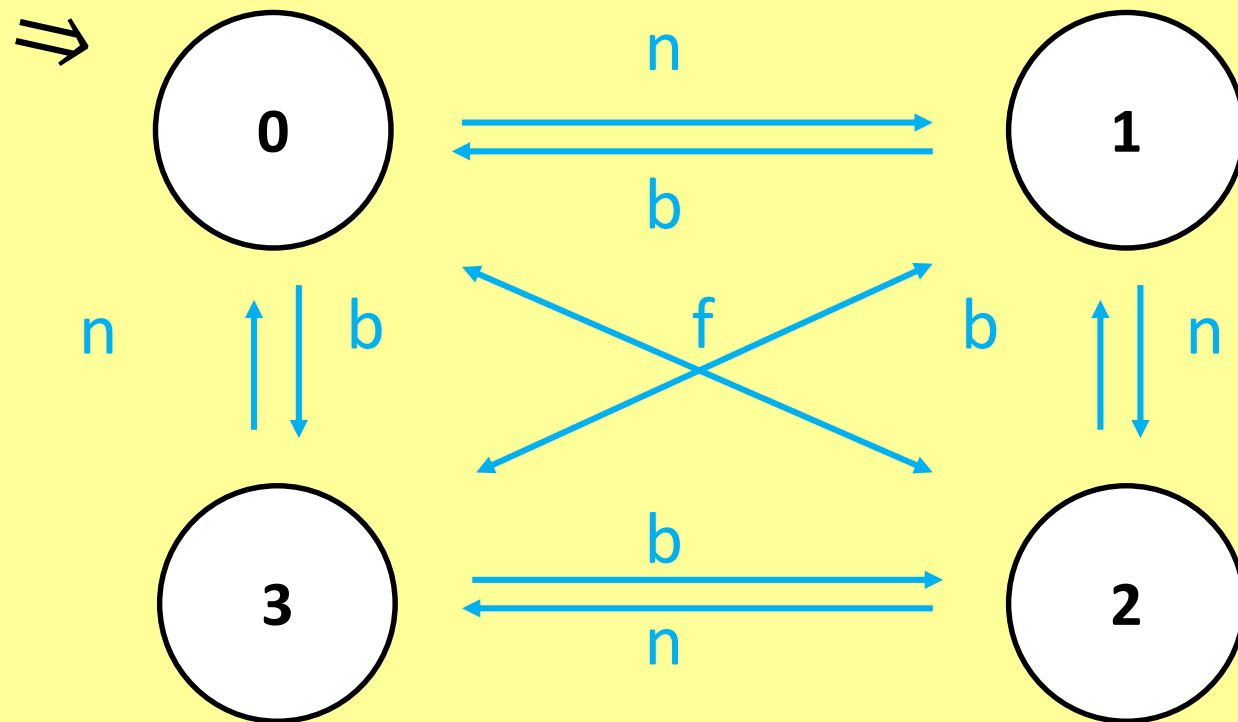
State Machines Project

- Does anyone have something to show?
 - Let's see the code + how it runs on your laptops. Please share your screens!
- Let's write this up on a simple demo in Colab together.
 - We can't use the same inputs and outputs (we are without a display), but we can use a simple loop which asks for a single letter and shows a plot in each iteration.
 - PS: We will be able to copy and paste this State Machine into the previous demos and have something interactive!

State Machines formalized:

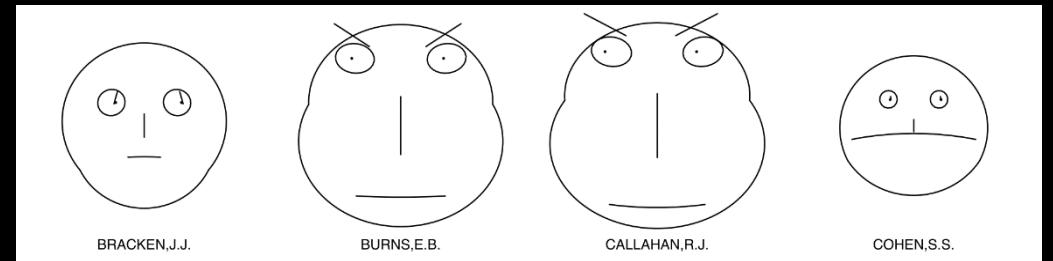
- Set of states (0,1,2,3)
 - Start: starting state (0)
- Accepting alphabet ("n")
- Transitions between states
 - $0 \Rightarrow 1$ with "n"
 - $1 \Rightarrow 2$ with "n"
 - ... etc
- **Mapping** between the state number and outputted color (for example)
 - $0 \Rightarrow$ red, $1 \Rightarrow$ orange, $2 \Rightarrow$ green, ...





Next class?

- **Probability and statistics** – how can we use this when looking at data such as list of number and images.
 - What is the average image in a dataset? What is the standard deviation then?
 - What ways can we use to visualize these statistics?



- **Types of random sampling** – and types of random noise (also as used to model textures, terrain etc.)