Data, Math and Methods

Week 6, Math Repeat I

~ Vítek Růžička

Today

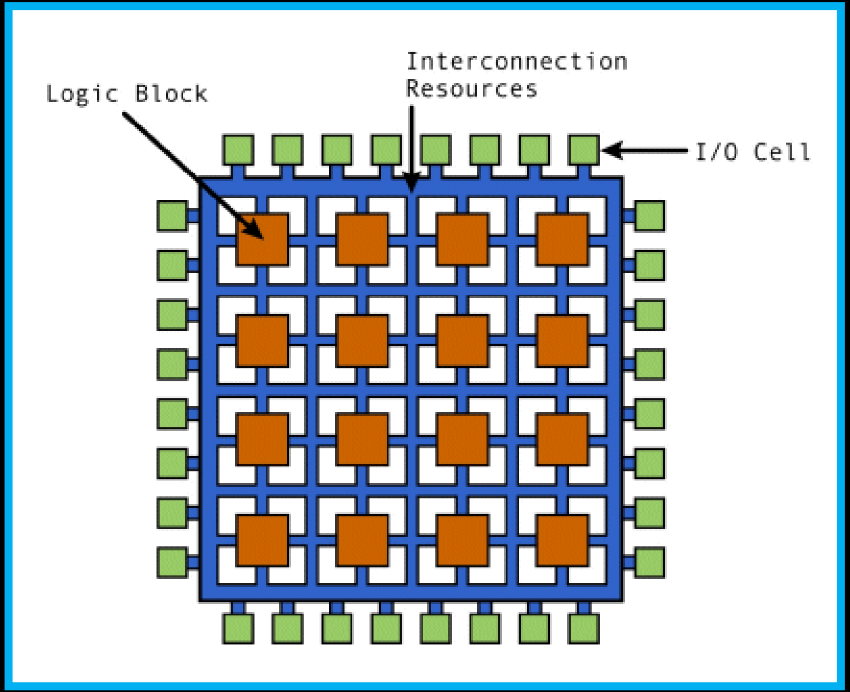
- **Repeating** what we learned last classes:
  - Tasks with logic formulas
  - And with state machines

- 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)

Motivation

- **Why do we care about logic?**

Because (for example) of FPGAs (Field Programmable Gate Arrays).



Motivation

- **Why do we care about logic?**

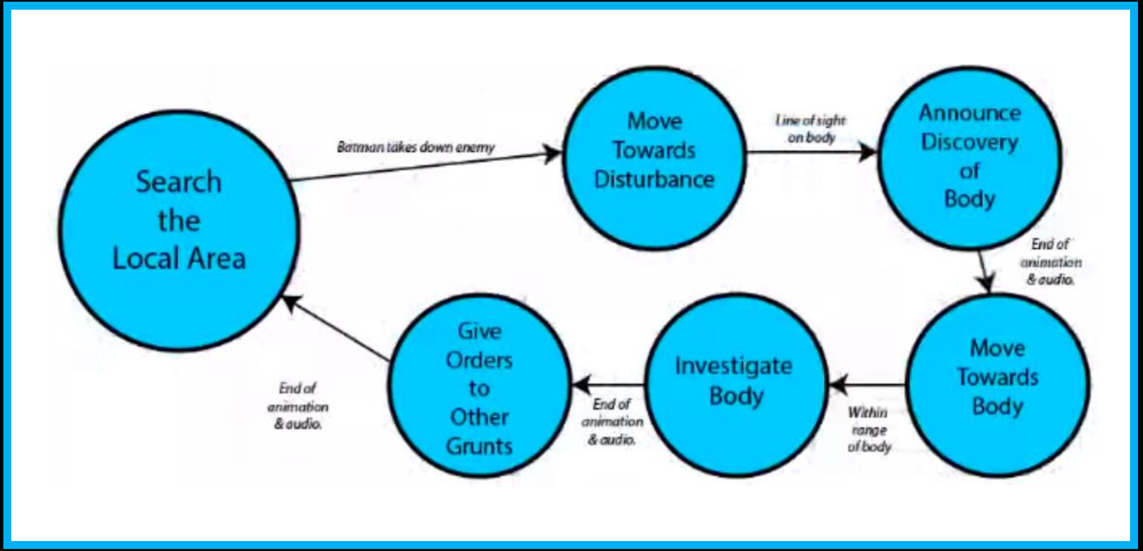
Because (for example) of FPGAs (Field Programmable Gate Arrays).

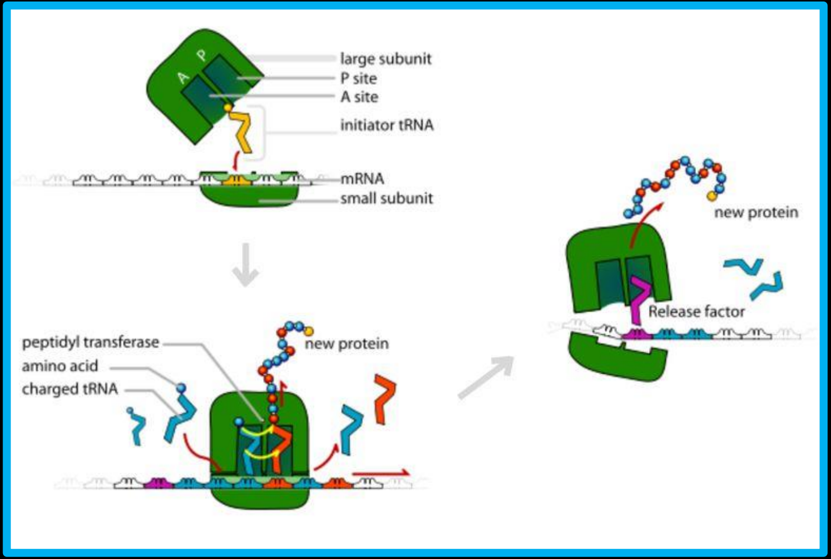
- These can build on the fly path from logical blocks as given by the design we submit to it. They combine the simple building blocks into something more complicated - to fulfill some expression, to basically run some program. *(Remember the NAND only circuits?)*
- These are btw super fast and super energy efficient. There is research is trying to use them with Machine Learning applications (having a fast and energy efficient box doing all the magic that a model was trained on somewhere else).
- … So … you don't need to do this all by hand. Or do it super fast.
  But I want you to know about it and to understand the concepts.

Motivation

- **Why do we care about state machines?**

Because of history of computer sciences … and because of AI in games.





Motivation

- **Why do we care about state machines?**

Because of history of computer sciences … and because of AI in games.

- State Machines are ways of programming, before there were computers in real world. As a design concept.
- On the last class task / today's practicum you'll see that they are not completely dead either.
- The more broad concepts of programming automated units from simple interaction into complex behaviors is something that can help in understanding (or modelling) biological processes – and also to help appreciate the beauty of them. *(Do you remember Turing machines and similarities with the Gene replication / expression?)*

Recap: Logic Formulas

- Enumerating all possible variable values in 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

exclusive or, not and, not or

a	b
0	0
0	1
1	0
1	1

$\neg a$	$a \land b$	$a \lor b$	$a \Rightarrow b$	$a \Leftrightarrow b$
1	0	0	1	**1**
1	0	1	1	0
0	0	1	0	0
0	**1**	**1**	**1**	**1**

a XOR b	a NAND b	a NOR b
0	**1**	**1**
1	**1**	0
1	**1**	0
0	0	0

Implies: statement "if $\underline{a}$ is true then $\underline{b}$ is also true"

Logic Formulas – Task 0

- Check if A ≡ B on all it’s logical evaluations
- ≡ means: logical equivalence (aka their tables are the same!)

A: a XOR b

B: $(a \lor b) \land \neg(a \land b)$

Logic Formulas – Task 1

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

1.) A: T (true on all possible a's and b's)
B: $\neg a \lor (a \land (a \lor b))$

2.) A: $a \land b \land c$
B: $a \lor b \lor c$

Logic Formulas – Task 2

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

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

A: $c \lor (a \land b)$

B: $(c \lor a) \land (c \lor b)$

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

PPS: this is called the “distributive property”

Logic Formulas – Task 3

- We have a long (potentially very long) row with “and” ∧ 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:
  - True ∧ True ∧ True ∧ True ∧ True ∧ False ∧ True ∧ True ∧ True ∧ True ≡ 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:
    - True $\wedge$ True $\wedge$ True $\wedge$ True $\wedge$ True $\wedge$ False $\wedge$ True $\wedge$ True $\wedge$ True $\wedge$ True $\equiv$ 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” V 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:
  - False V False V False V False V False V False V False V True V False V ≡ 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” V 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:
  - False V False V False V False V False V False V False V True V False V ≡ 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:

$$(a) \lor (b) \lor (\neg a) \lor (a) \lor (\neg a \land a) \lor (\neg a \lor a)$$

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

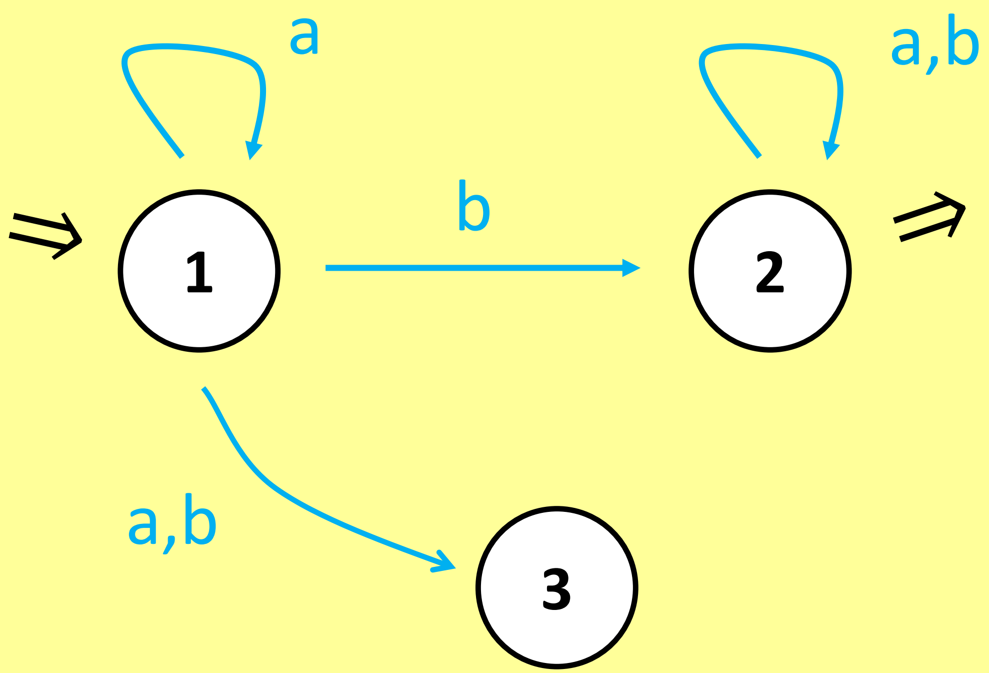
State Machines

- 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

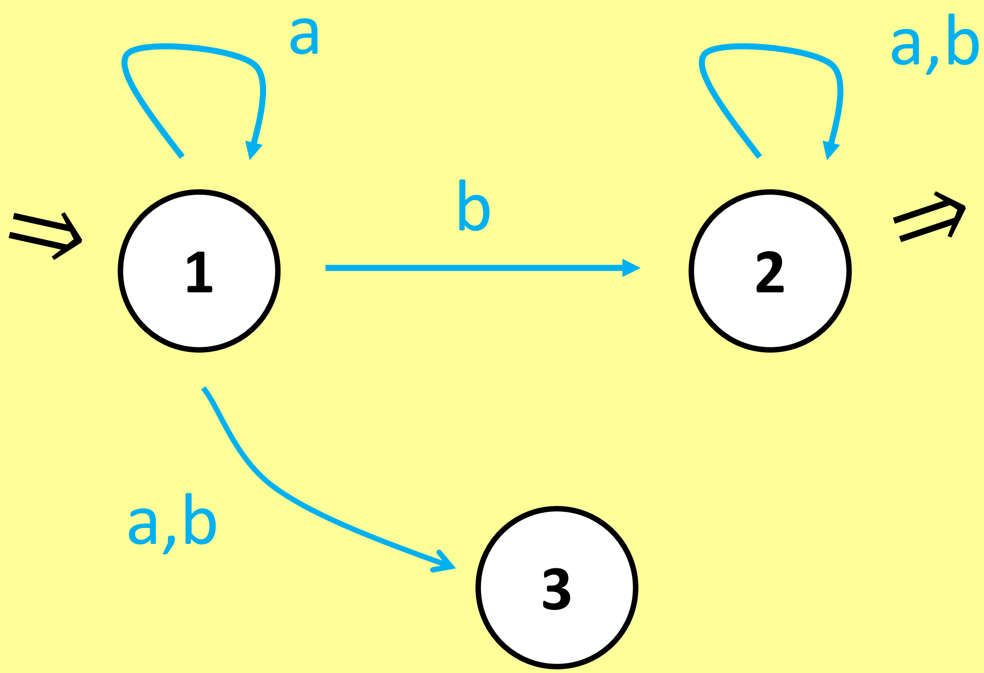
What would this machine do?


$$\begin{tikzcd}
\Rightarrow & 1 \arrow[loop above, "a"] \arrow[r, "b"] \arrow[d, "{a,b}"'] & 2 \arrow[loop above, "{a,b}"] & \Rightarrow \\
& 3 & &
\end{tikzcd}$$

What does it do?

State Machines example

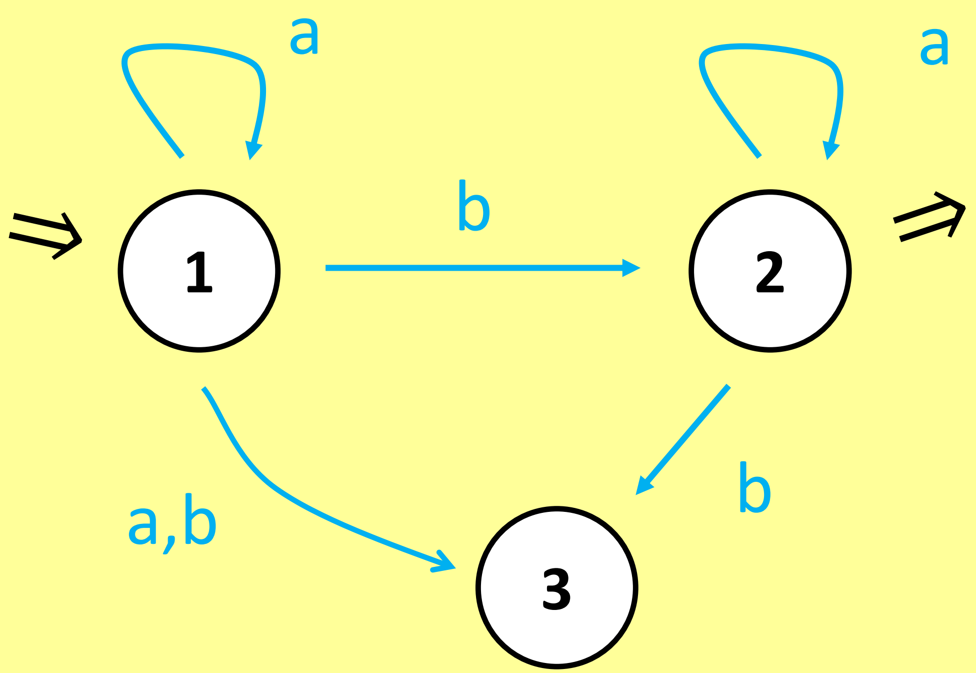
What would this machine do?


$$\begin{tikzcd} \arrow[r, Rightarrow] & 1 \arrow[loop above, "a"] \arrow[r, "b"] \arrow[dr, "{a,b}"'] & 2 \arrow[loop above, "{a,b}"] \arrow[r, Rightarrow] & {} \\ & & 3 & \end{tikzcd}$$

<u>**Words which contain at least one "b"!**</u>

State Machines example

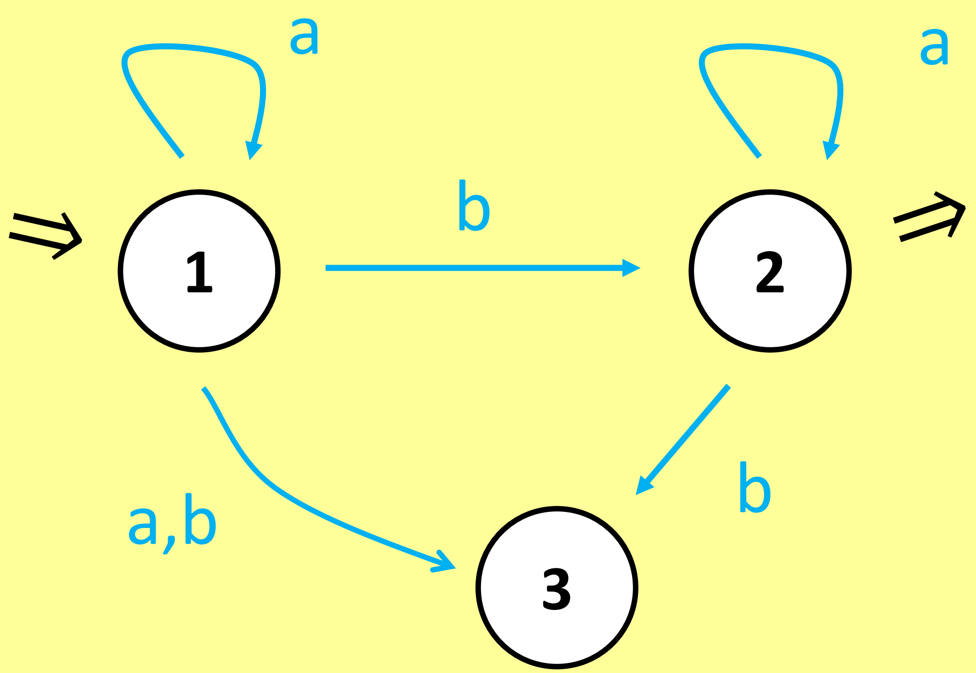
What would this machine do?



And what about this one?

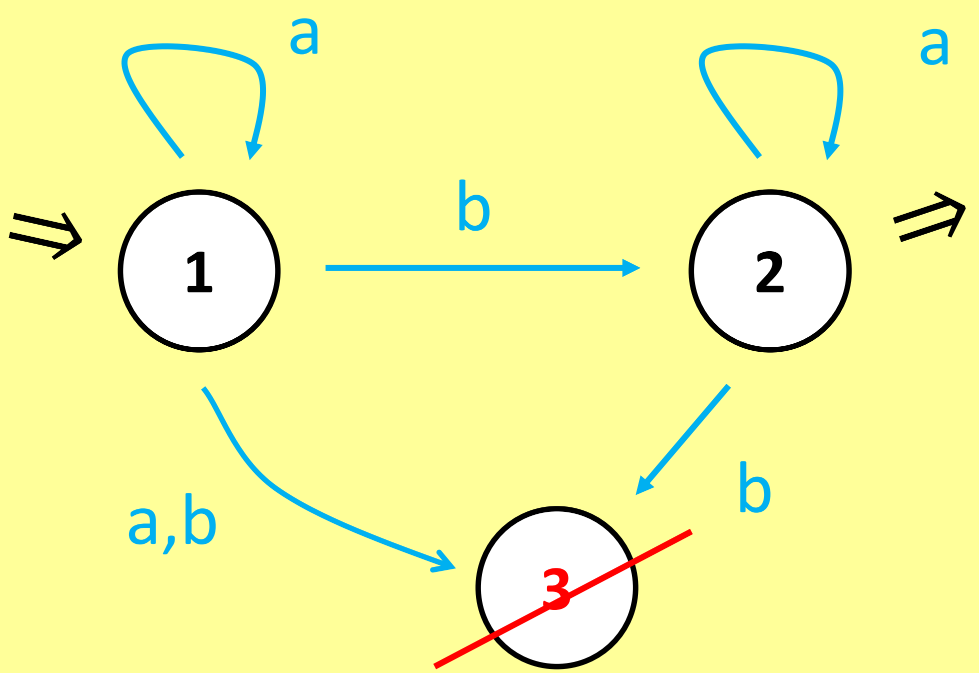
State Machines example

What would this machine do?



<u>Words which contain at exactly one “b”!</u>

Agreement:



Let's agree: When designing your State Machines, you don't have to include all the blind paths unless they would change the behavior. If we don't have State 3 then this machine would accept any number of "b"s and not just the one.

Sometimes we need the "death" state.

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", "bbbbbba", "bbbbaa" 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).

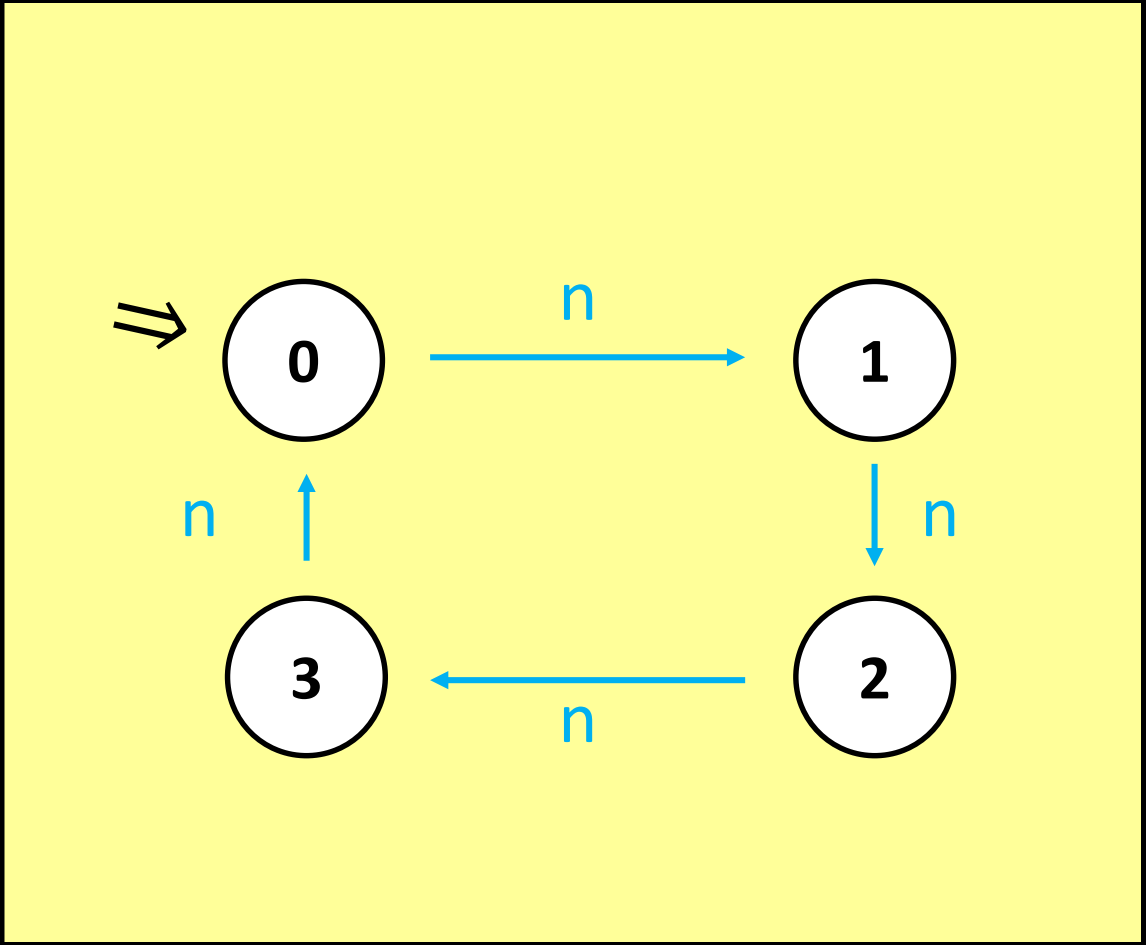
Programming

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 => 1 with “n”
  - 1 => 2 with “n”
  - … etc
- **Mapping** between the state number and outputted color (for example)
  - 0 => red, 1 => orange, 2=> green, …


$$\begin{tikzcd} \arrow[r, Rightarrow] & 0 \arrow[r, "n"] & 1 \arrow[d, "n"] \\ & 3 \arrow[u, "n"] & 2 \arrow[l, "n"] \end{tikzcd}$$

$$\begin{tikzcd}
{} \arrow[r, Rightarrow] & 0 \arrow[r, shift left, "n"] \arrow[d, shift left, "b"] \arrow[rd, leftrightarrow, "f"] & 1 \arrow[l, shift left, "b"] \arrow[d, shift left, "n"] \arrow[ld, leftrightarrow] \\
& 3 \arrow[u, shift left, "n"] \arrow[r, shift left, "b"] & 2 \arrow[l, shift left, "n"] \arrow[u, shift left, "b"]
\end{tikzcd}$$

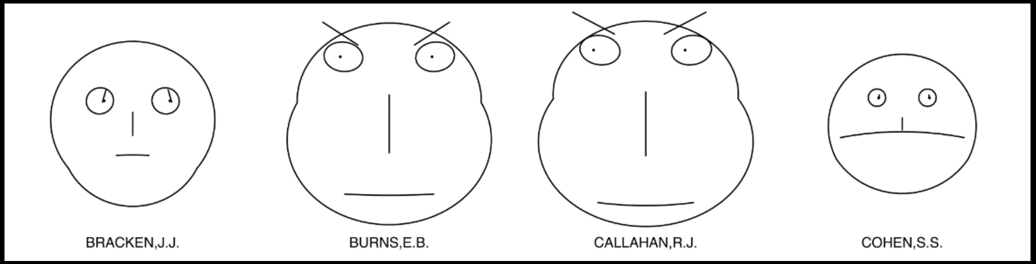
Programming

Open the **starter codes** placed on our github:

- week06_math-repeat-I/w6_state-machine-project-[starter].ipynb

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.)

Homework:

Task from Logical formulas

- Check if A ≡ B on all it's logical evaluations
- ≡ means: logical equivalence (aka their tables are the same!)

A: $(a \land b) \lor (a \land c) \lor (\neg a \land b \land c)$

B: $(a \lor b) \land (a \lor c) \land (b \lor c)$