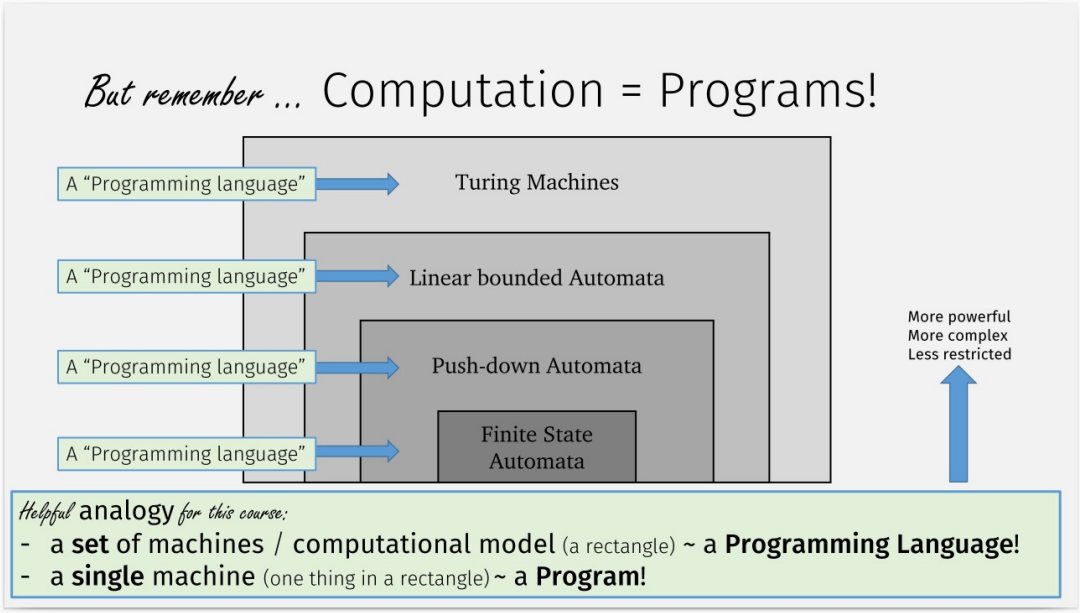
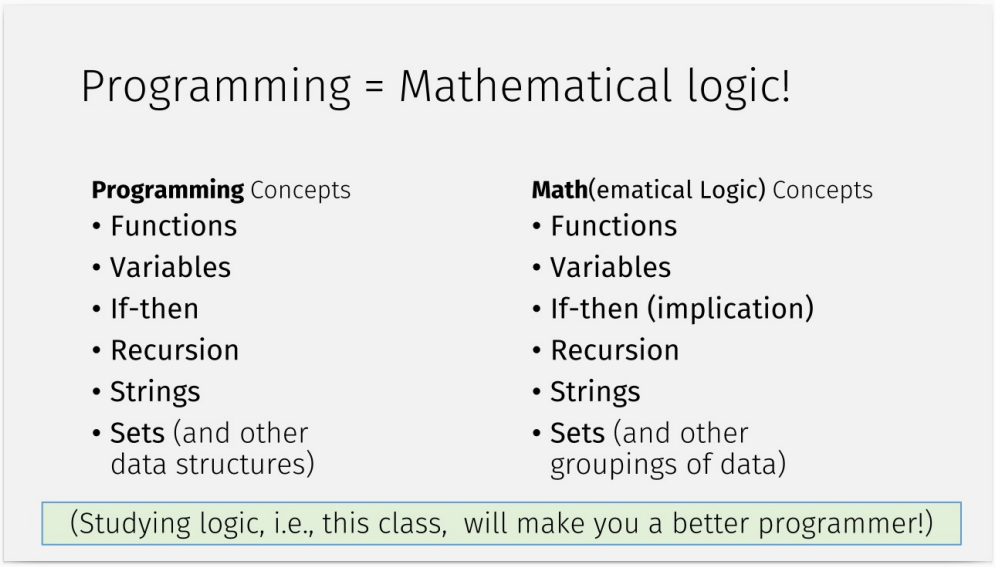


Lecture 1 slide 12



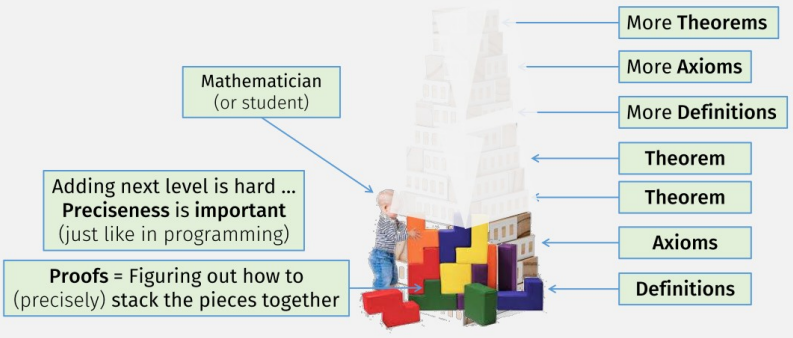
Lecture 1 slide 17

This semester, we will ...

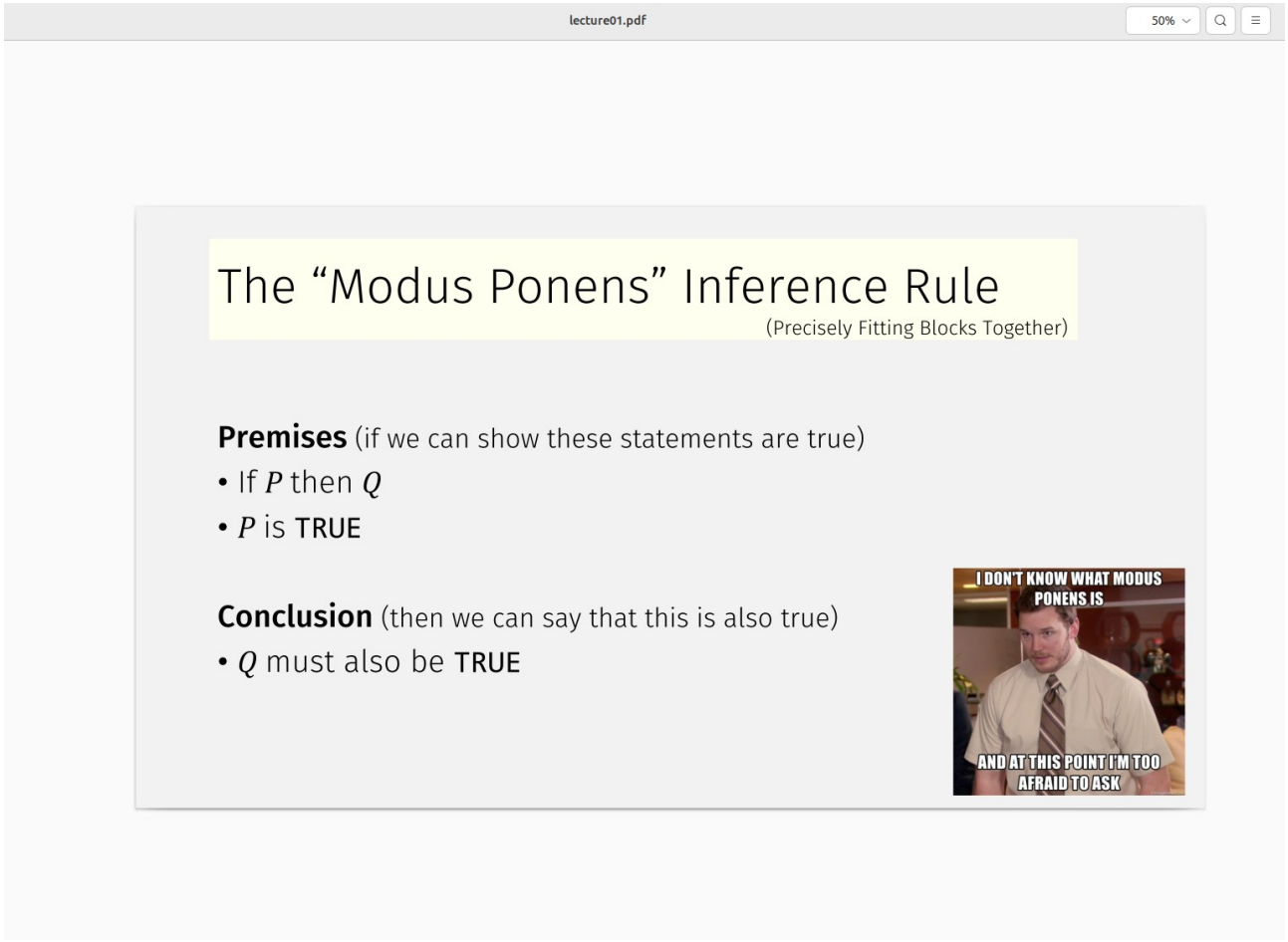
1. Define and study **models of computation**
   - models will be *as simple as possible* (to make them easier to study)
2. Compare and contrast models of computation
   - which "programs" are *included* by a model
   - which "programs" are *excluded* by a model
   - *overlap* between models?
3. Prove things about the models

Lecture 1 slide 18

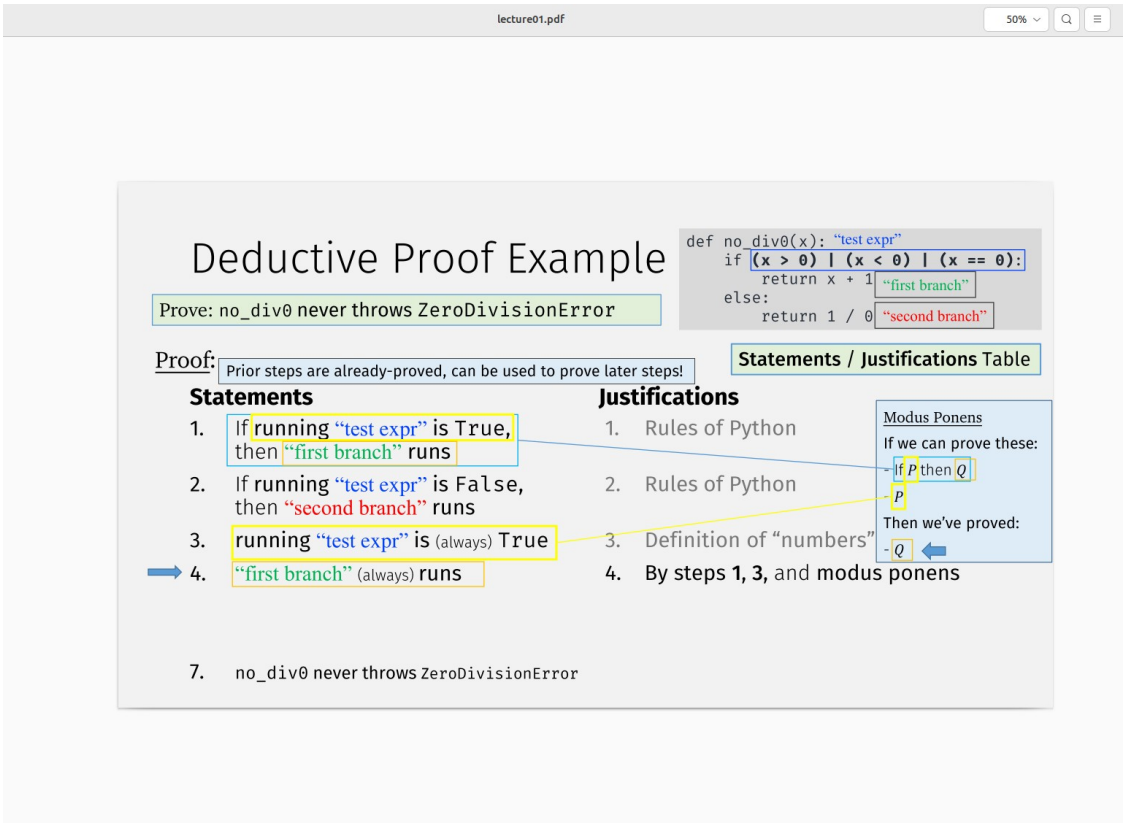
How Mathematics (Proofs) Work



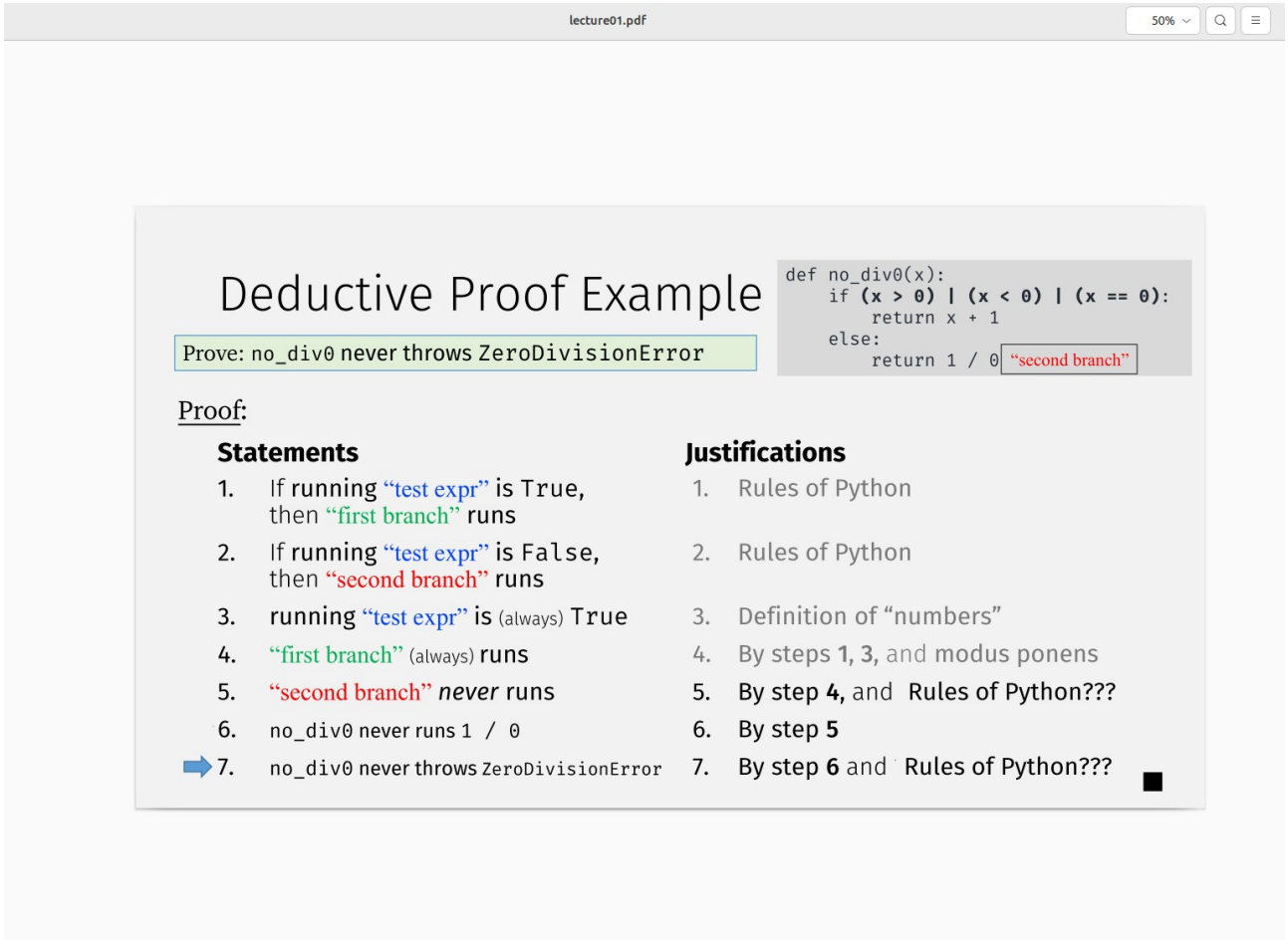
Lecture 1, slide 21



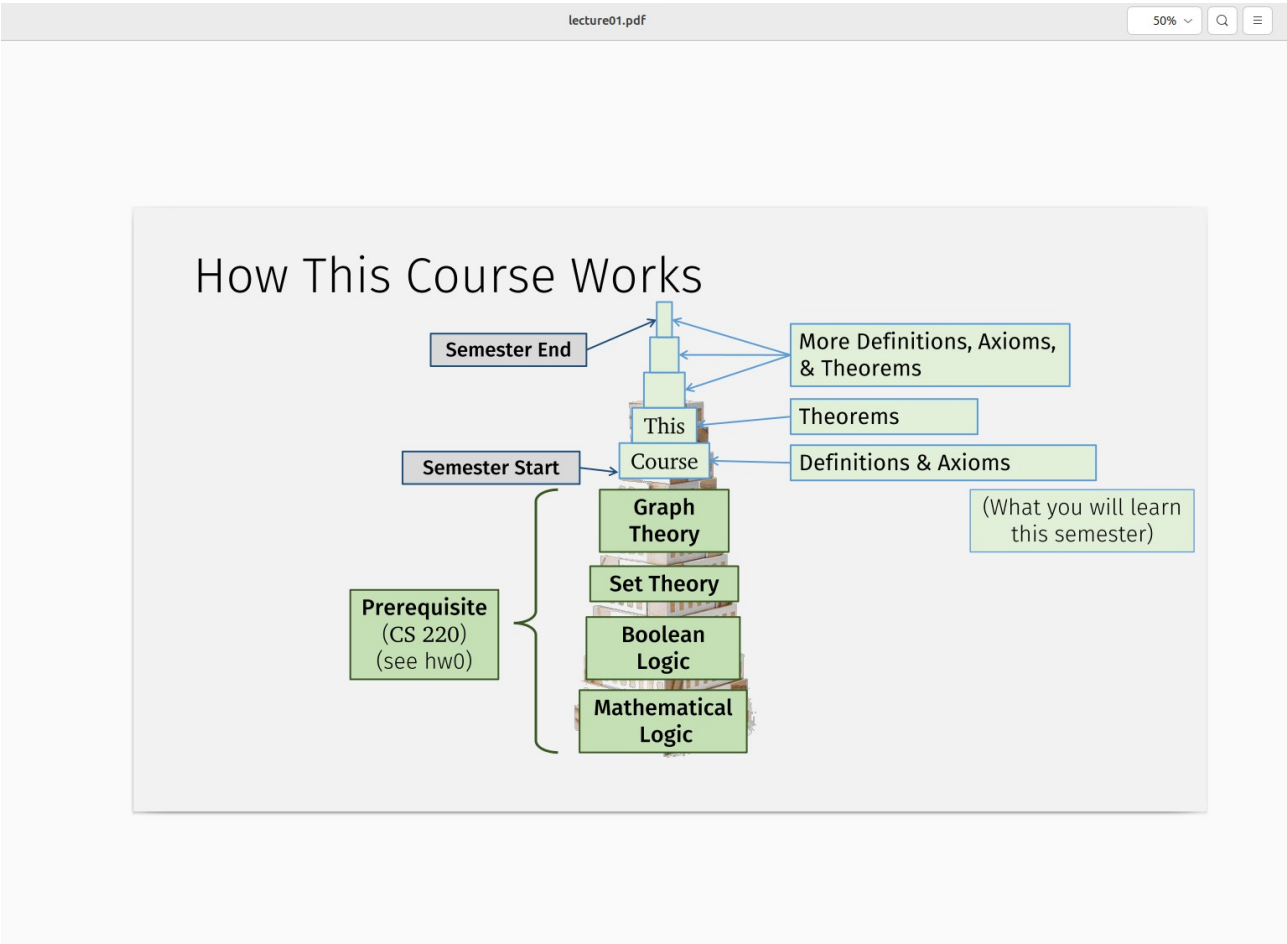
Lecture 1, slide 22



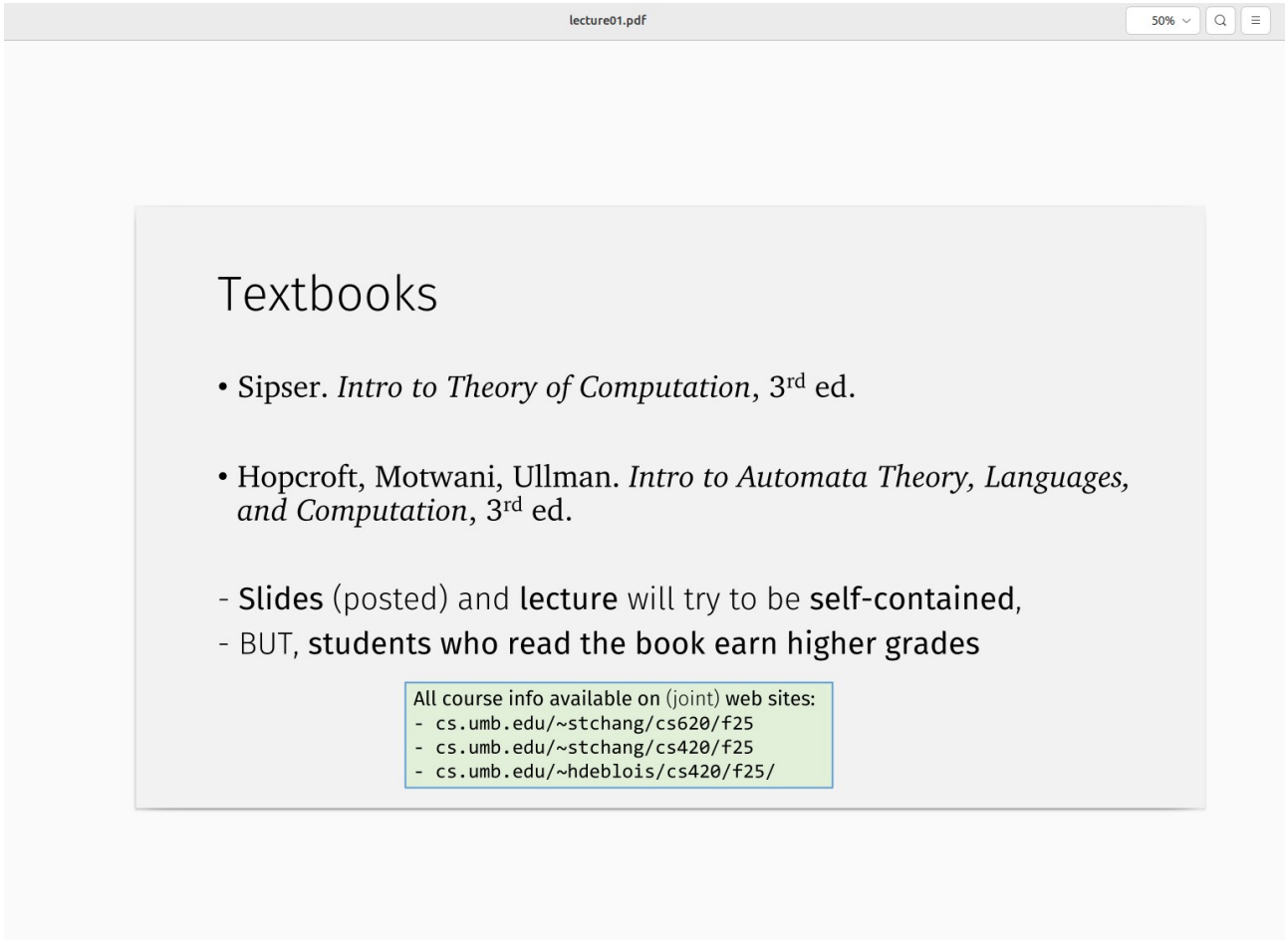
Lecture 1, slide 25



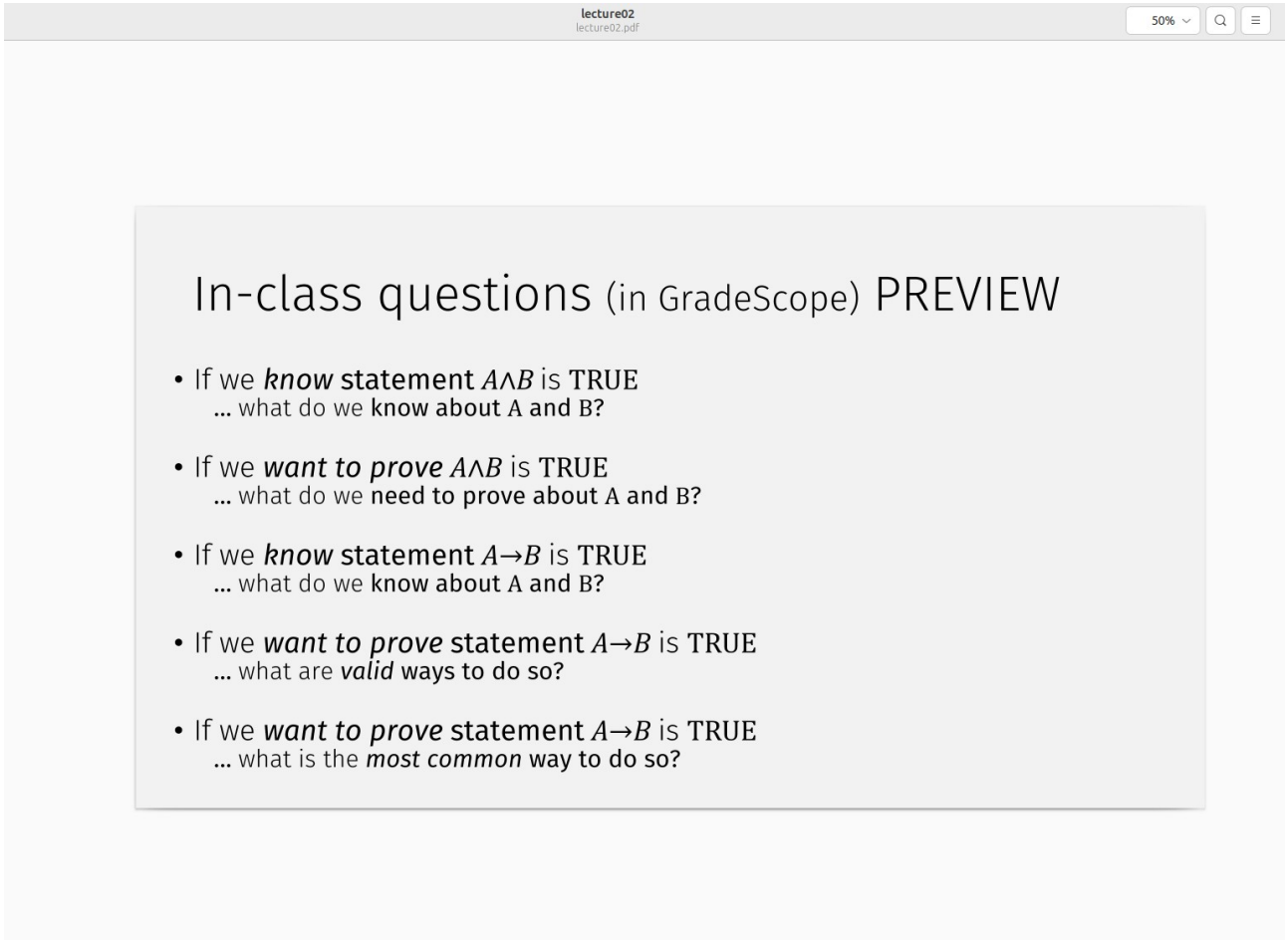
Lecture 1, slde 27



Lecture 1, slide 34



Lecture 1, slide 37

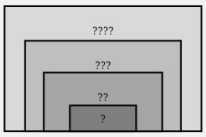


Lecture 2, slide 3

Last Time

This semester, we will ...

1. Define and study **models of computation** (formal, precisely-defined)
   - models will be *as simple as possible* (to make them easier to study)
2. Compare and contrast models of computation
   - which "programs" are *included* / *excluded* by a model
   - *Equality* or *overlap* between models?
3. Prove things about the models



Analogy:
Computation Model
(system of definitions and rules)
⇔
Programming Language

Analogy:
A **Computation** (in a model)
⇔
A Program

Lecture 2, slide 4

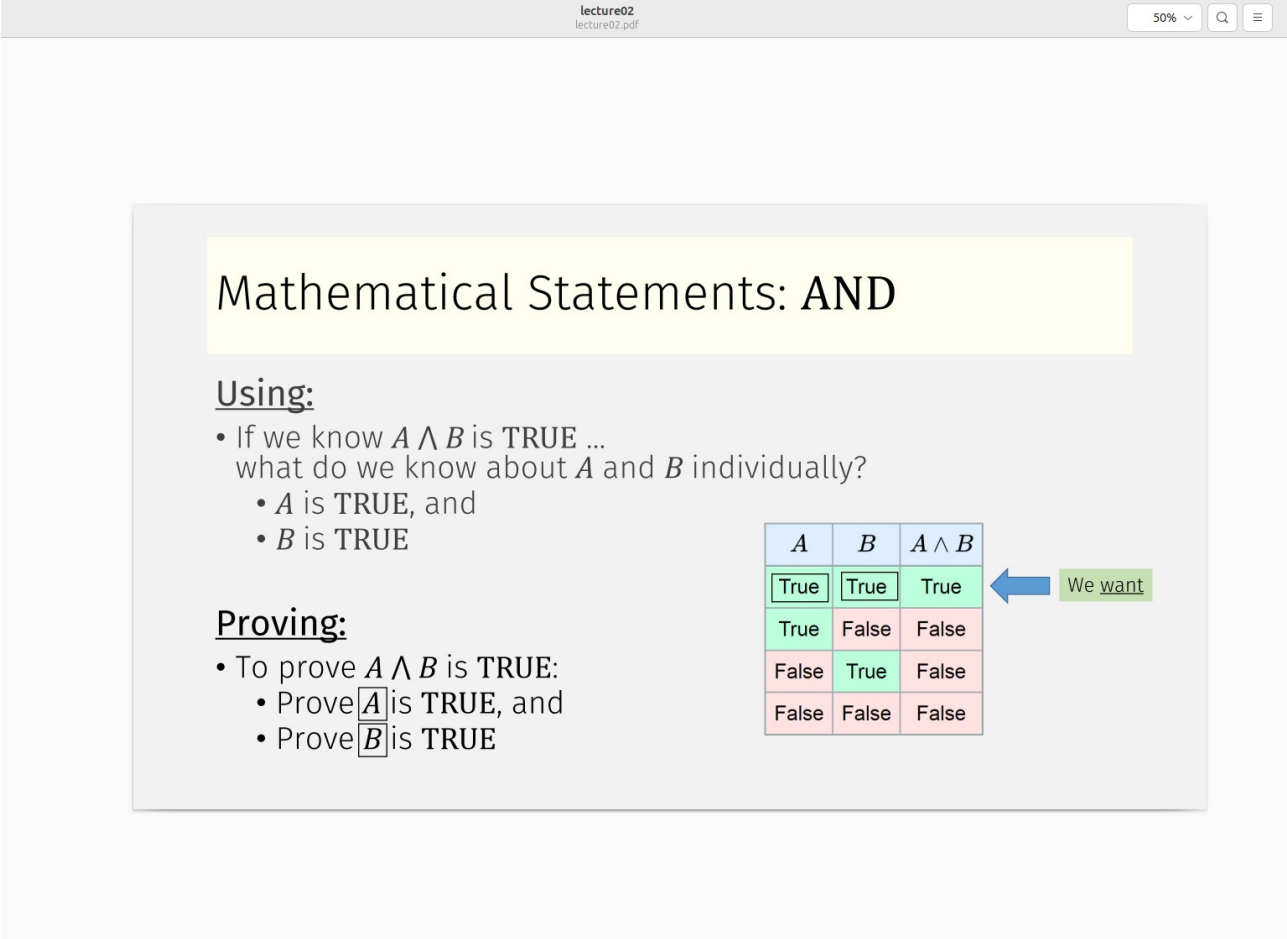
Mathematical Logic Operators

- **Conjunction** (AND, $\wedge$)
- **Disjunction** (OR, $\vee$)
- **Negation** (NOT, $\neg$, $\sim$)
- **Implication** (IF-THEN, $\Rightarrow$, $\rightarrow$)

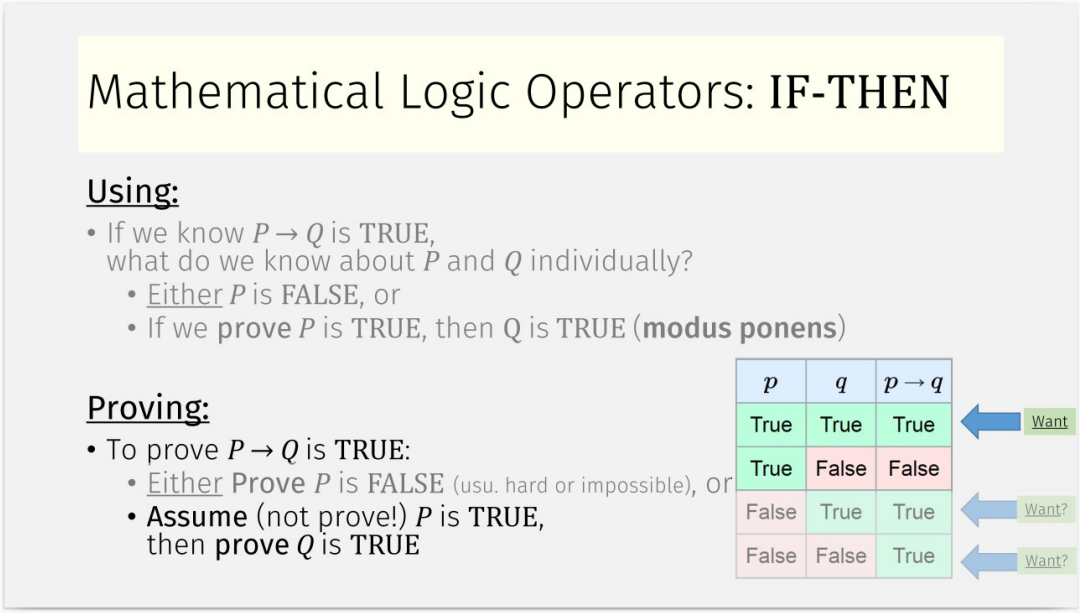
...

This semester:
Must understand difference between **Using** vs **Proving** a mathematical statement!

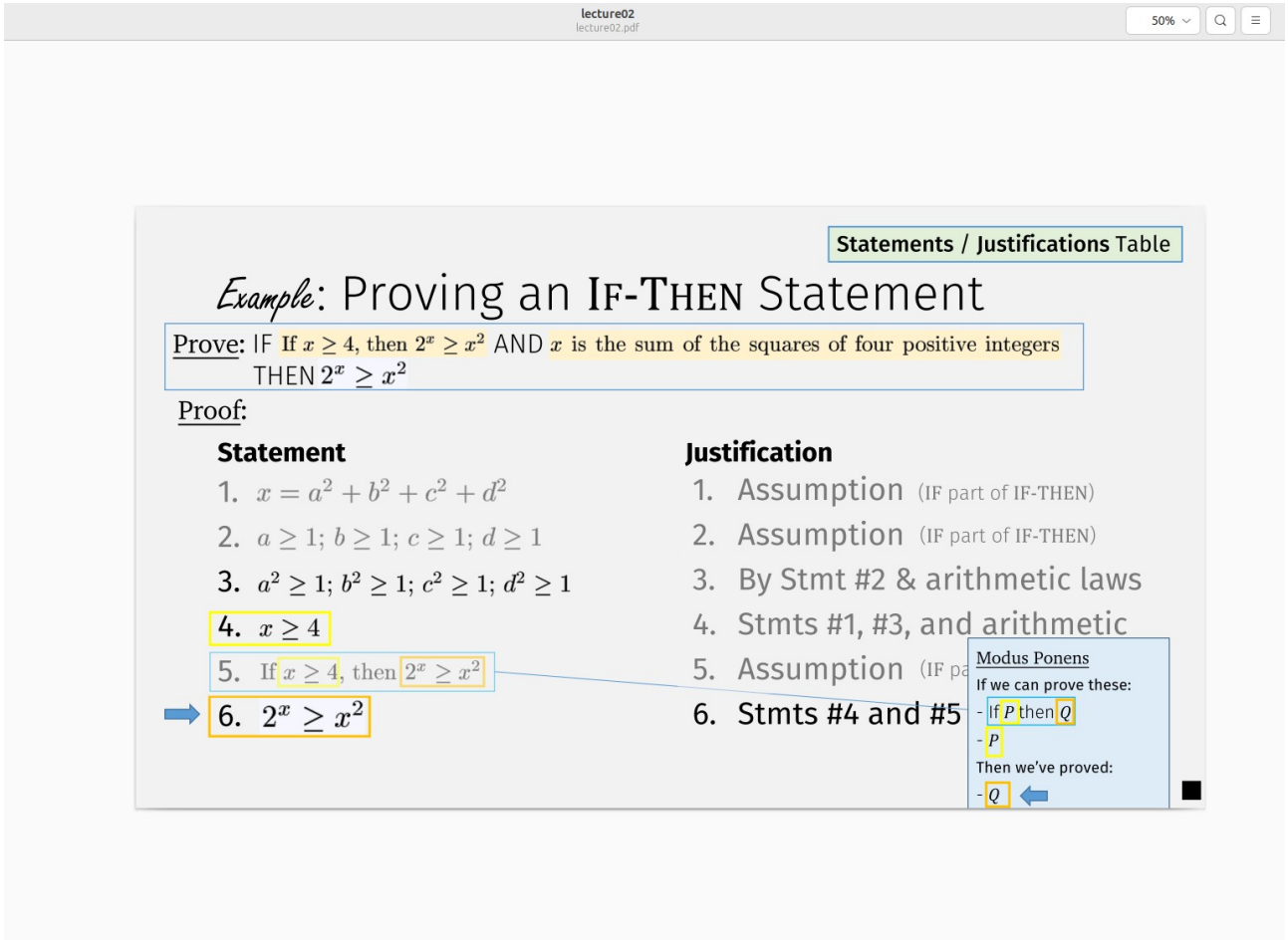
Lecture 2, slide 18



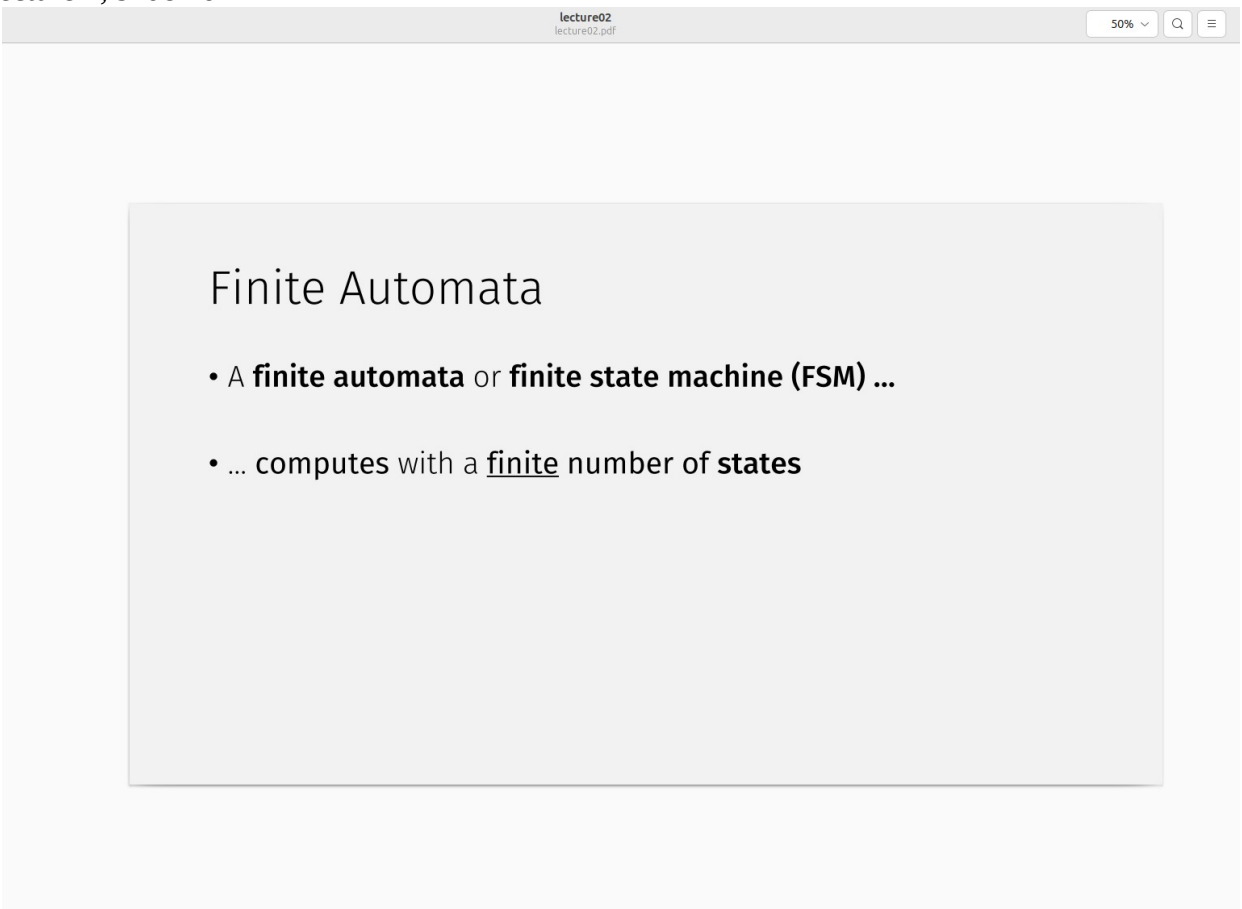
Lecture 2, slide 20



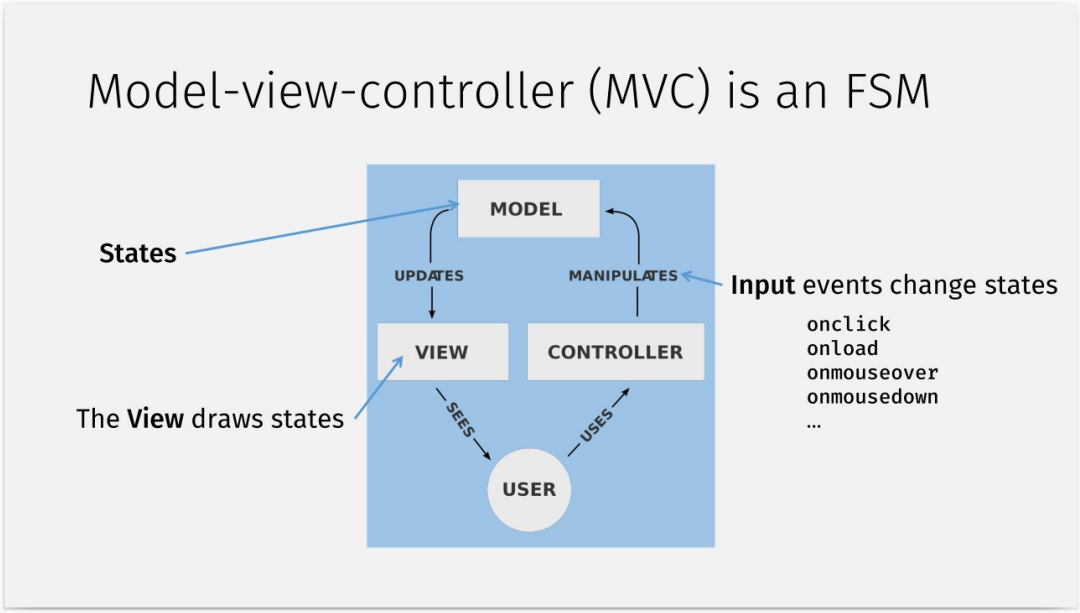
Lecture 2, slide 23



Lecture 2, slide 26



Lecture 2, slide 30



Lecture 2, slide 35

DEFINITION A "Programming Language"

A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

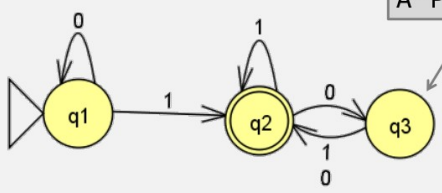
1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta\colon Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

An Example (as formal description)

$M_1 = (Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as

A "Program"



	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

4. q_1 is the start state, and
5. $F = \{q_2\}$.

"Programming" Analogy

This "analogy" is meant to help your intuition

But it's important not to confuse with **formal definitions.**

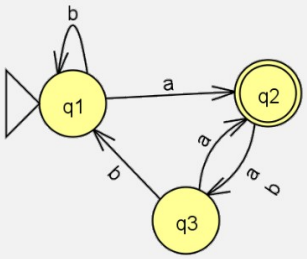
Lecture 2, slide 53

In-class Exercise: solution

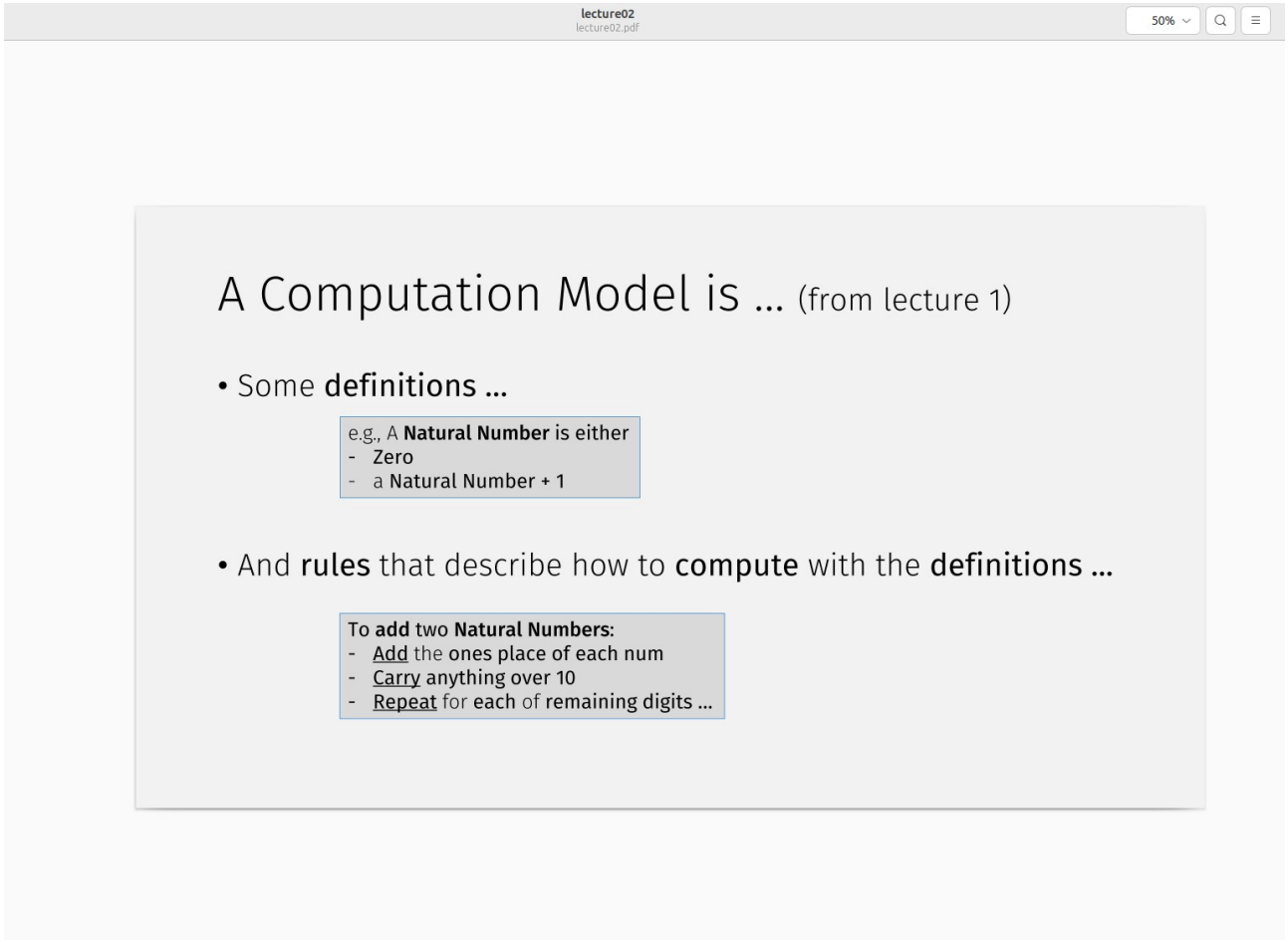
- Q = {q1, q2, q3}
- Σ= { **a**, **b** }
- δ
  - δ(q1, **a**) = q2
  - δ(q1, **b**) = q1
  - δ(q2, **a**) = q3
  - δ(q2, **b**) = q3
  - δ(q3, **a**) = q2
  - δ(q3, **b**) = q1
- q_0 = q1
- F = {q2}

(there are many ways to define a function, i.e., a mapping from domain elements to range elements)

$$M = (Q, \Sigma, \delta, q_0, F)$$



Lecture 2, slide 55
Given the definition of Finite Automata, then write the formal description of a particular machine.



Lecture 2, slide 56

A Computation Model is … (from lecture 1)

- Some **definitions** …

DEFINITION

A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta: Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

- And **rules** that describe how to **compute** with the **definitions** …

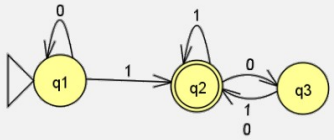
???

Lecture 2, Slide 58

Sipser Fig 1.4

Computation with DFAs (JFLAP demo)

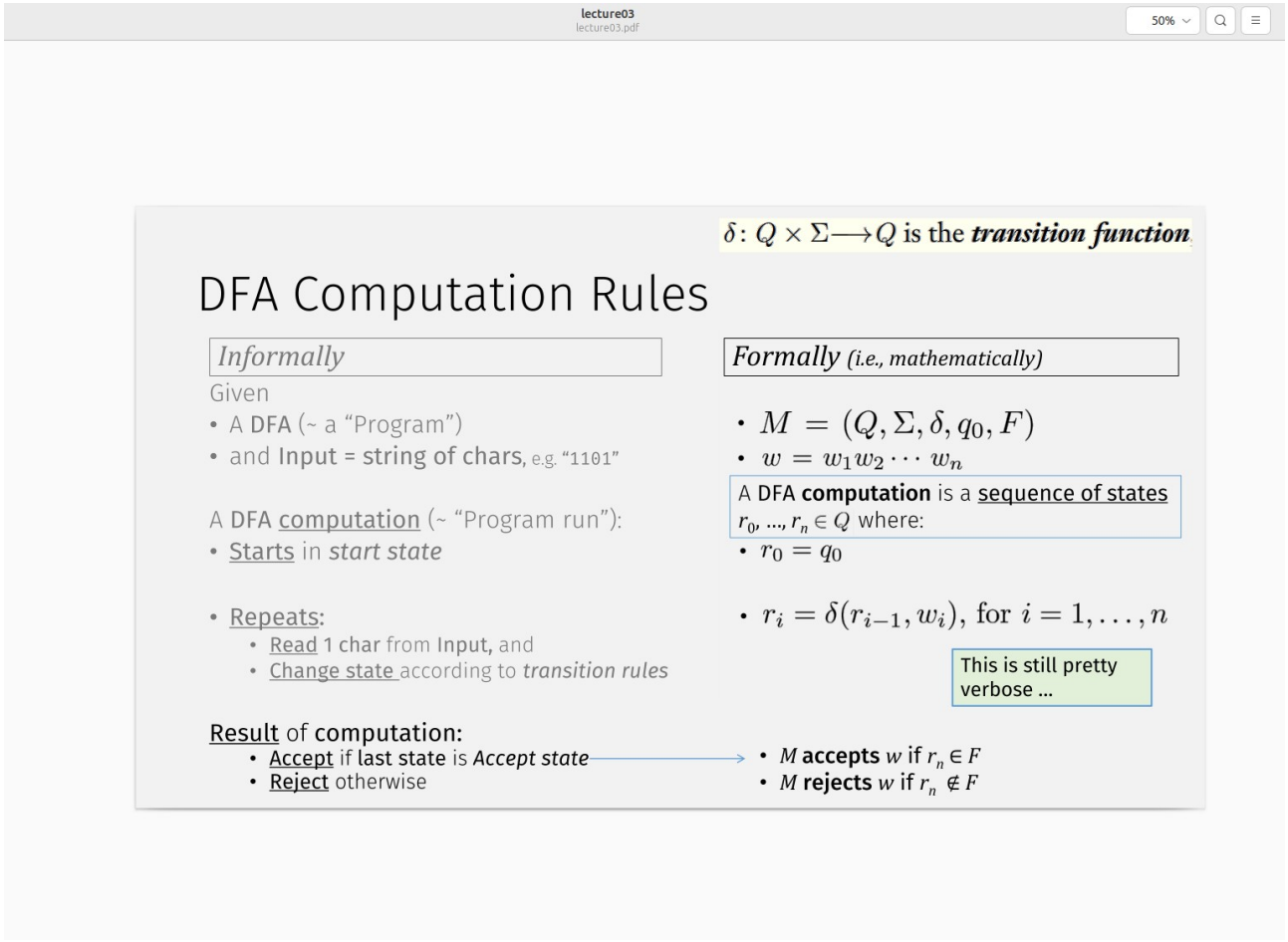
- DFA:



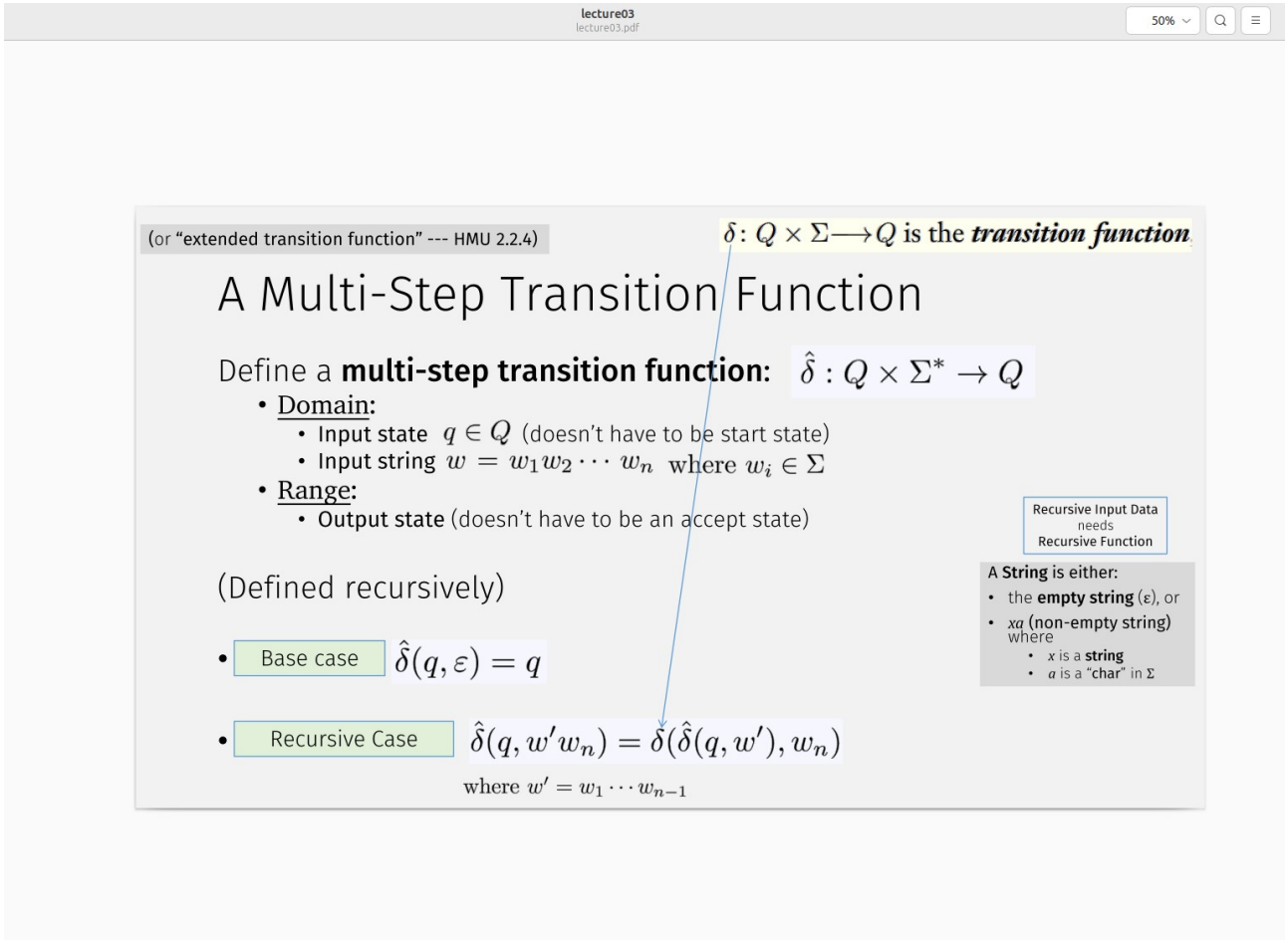
- Input: “1101”

HINT: always work out concrete examples to understand how a machine works

Lecture 2, slide 59



Lecture 3, slide 42 DFA computation rules (above)



Lecture 3, slide 53 (above)

DFA Computation Rules

Informally

Given
- A DFA (~ a "Program")
- and Input = string of chars, e.g. "1101"

A DFA computation (~ "Program run"):
- Starts in *start state*
- Repeats:
  - Read 1 char from Input, and
  - Change state according to *transition rules*

Result of computation:
- Accept if last state is *Accept state*
- Reject otherwise

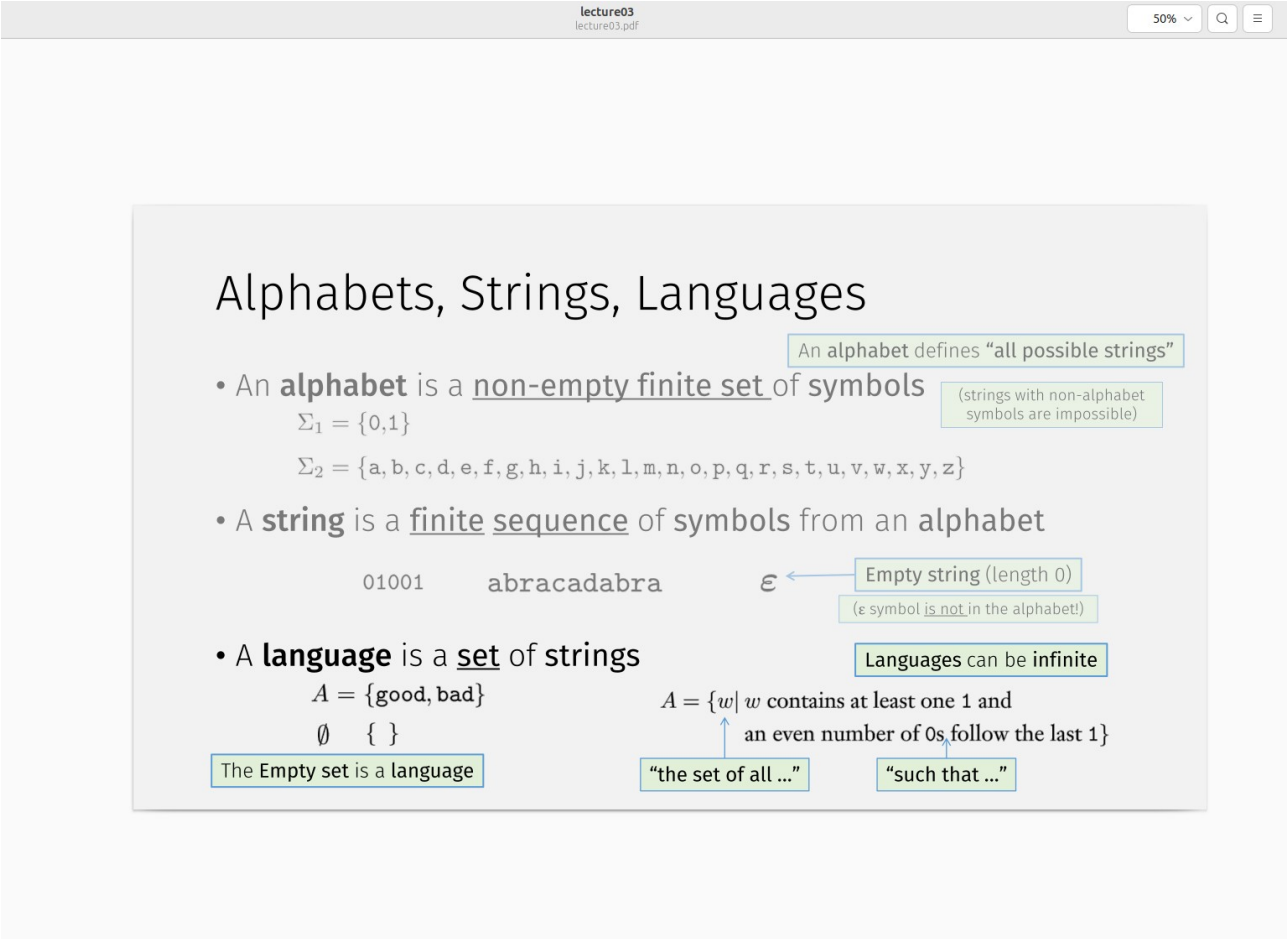
Formally (i.e., mathematically)

- $M = (Q, \Sigma, \delta, q_0, F)$
- $w = w_1 w_2 \cdots w_n$

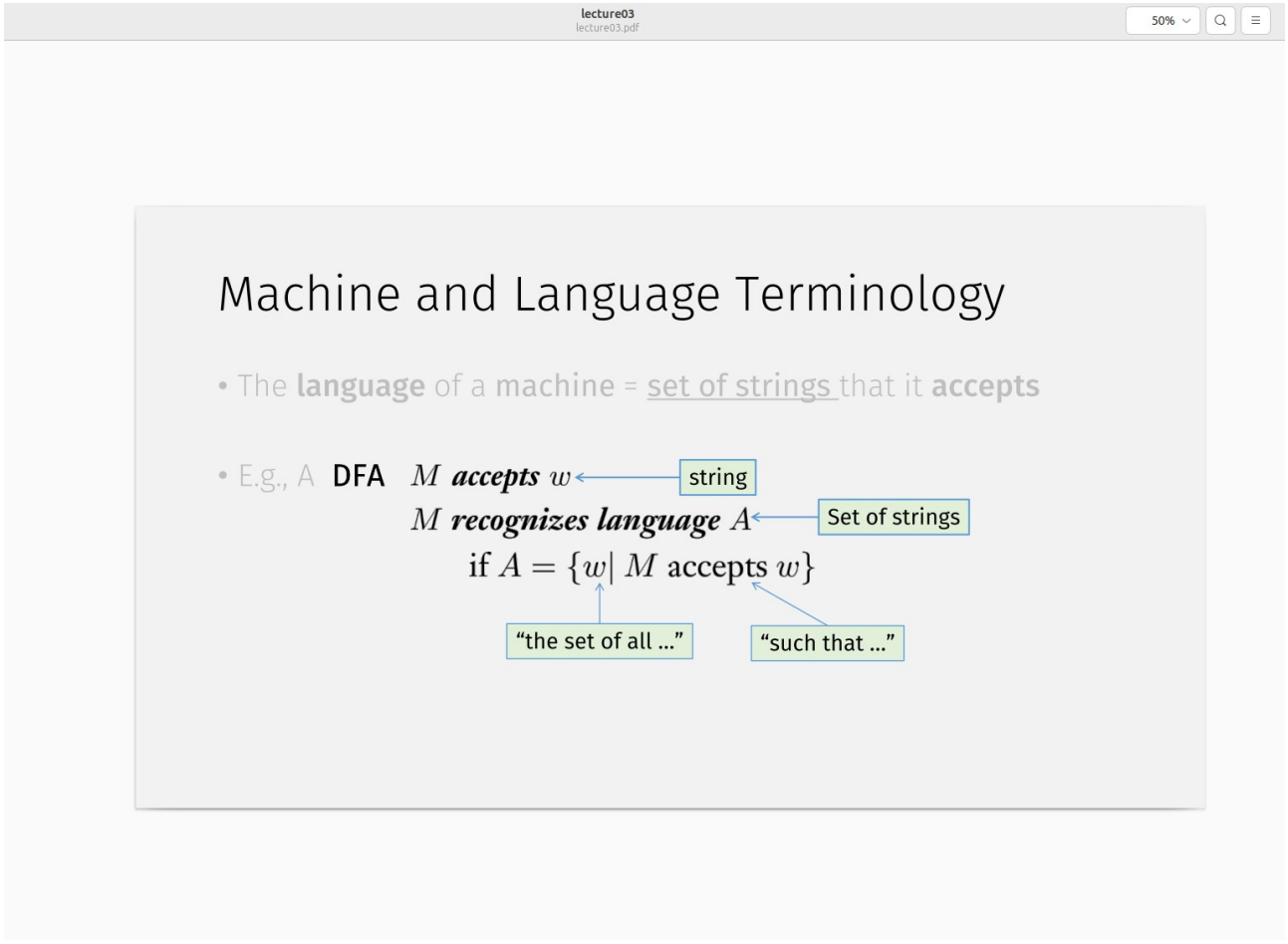
A DFA **computation** is a sequence of states $r_0, \ldots, r_n \in Q$ where:
- $r_0 = q_0$
- $r_i = \delta(r_{i-1}, w_i)$, for $i = 1, \ldots, n$

- M **accepts** w if $\hat{\delta}(q_0, w) \in F$
- M **rejects** w if $r_n \notin F$

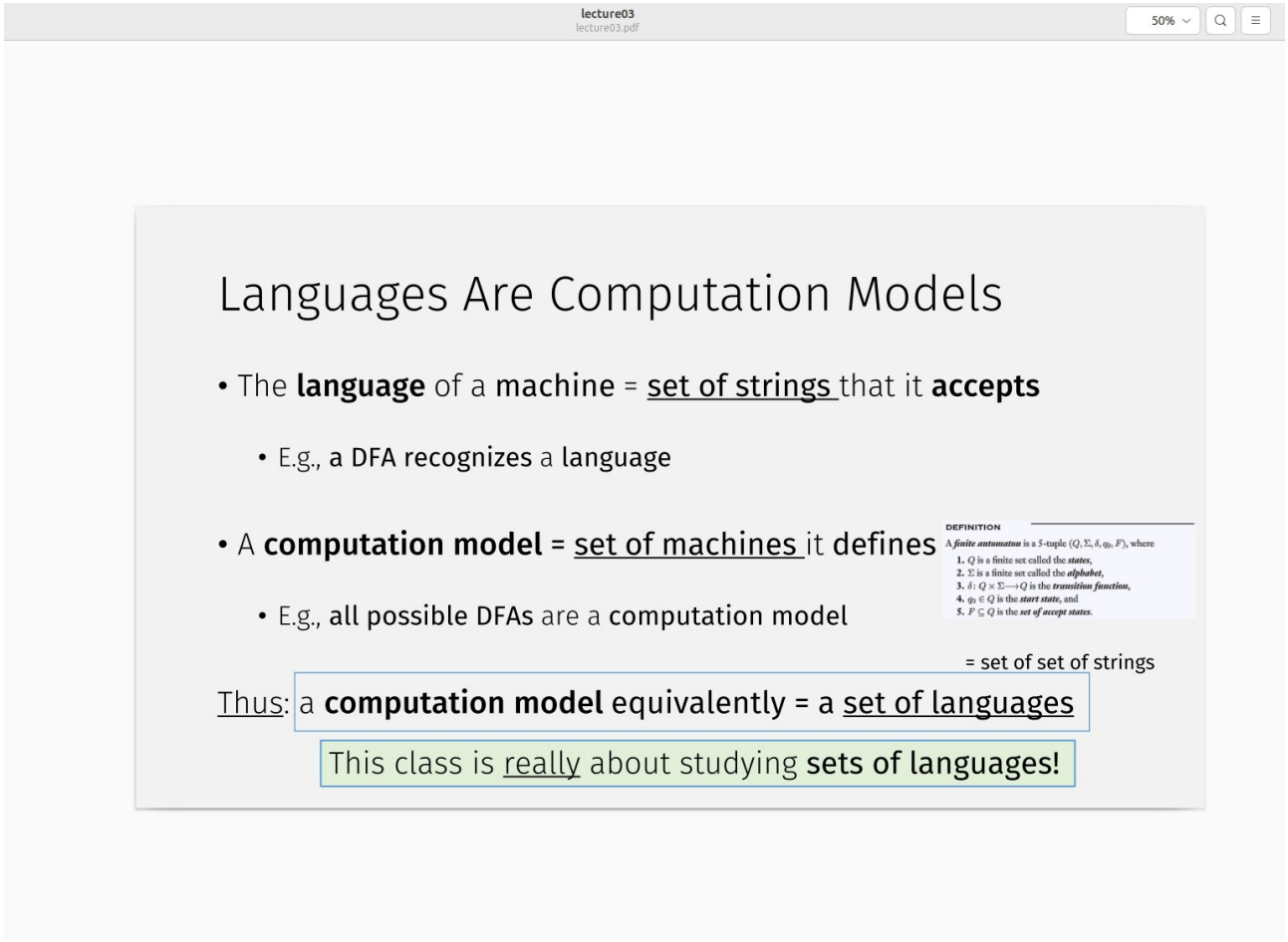
Lecture 3, slide 52



Lecture 3, slide 54



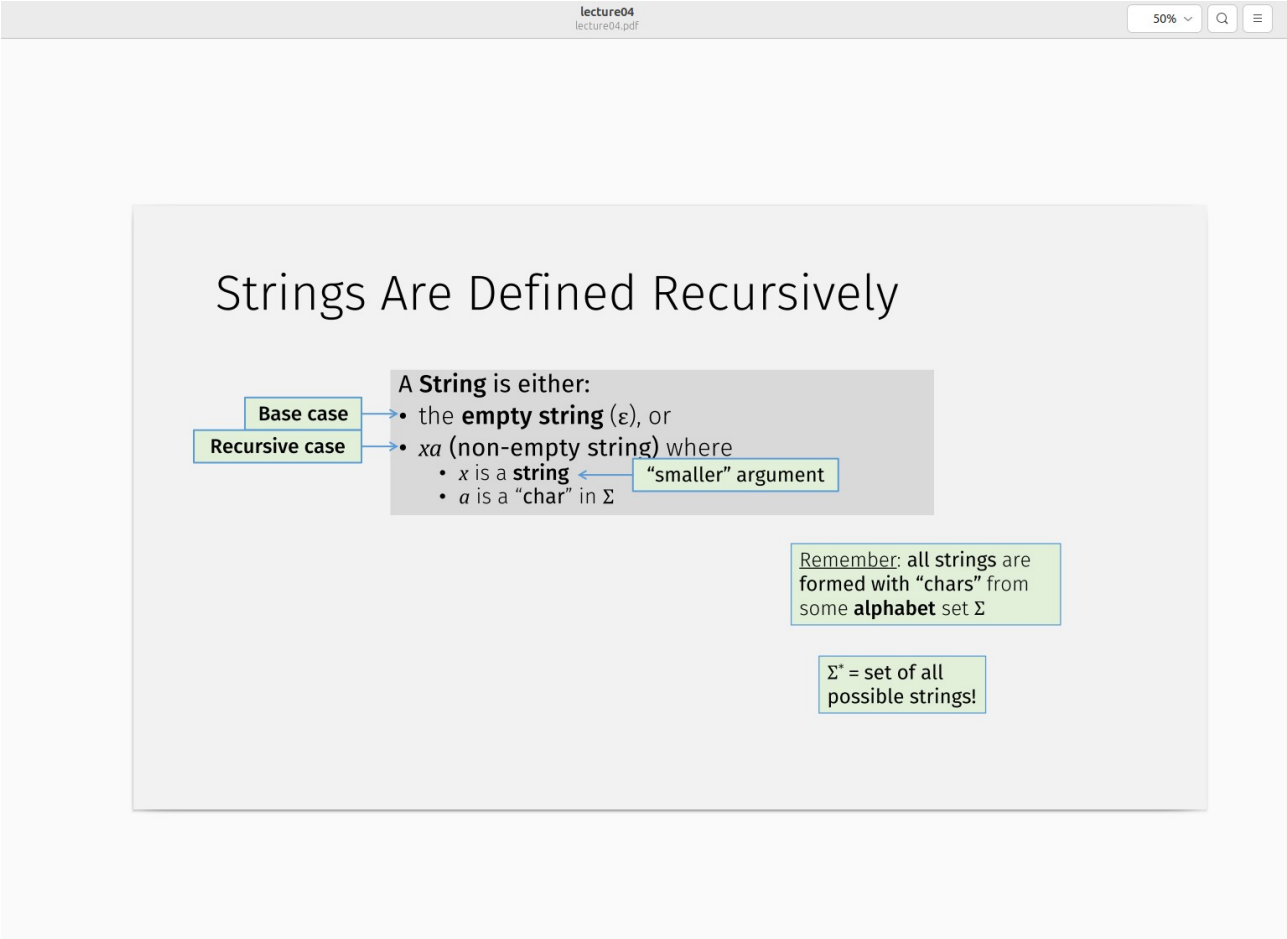
Lecture 3, slide 55



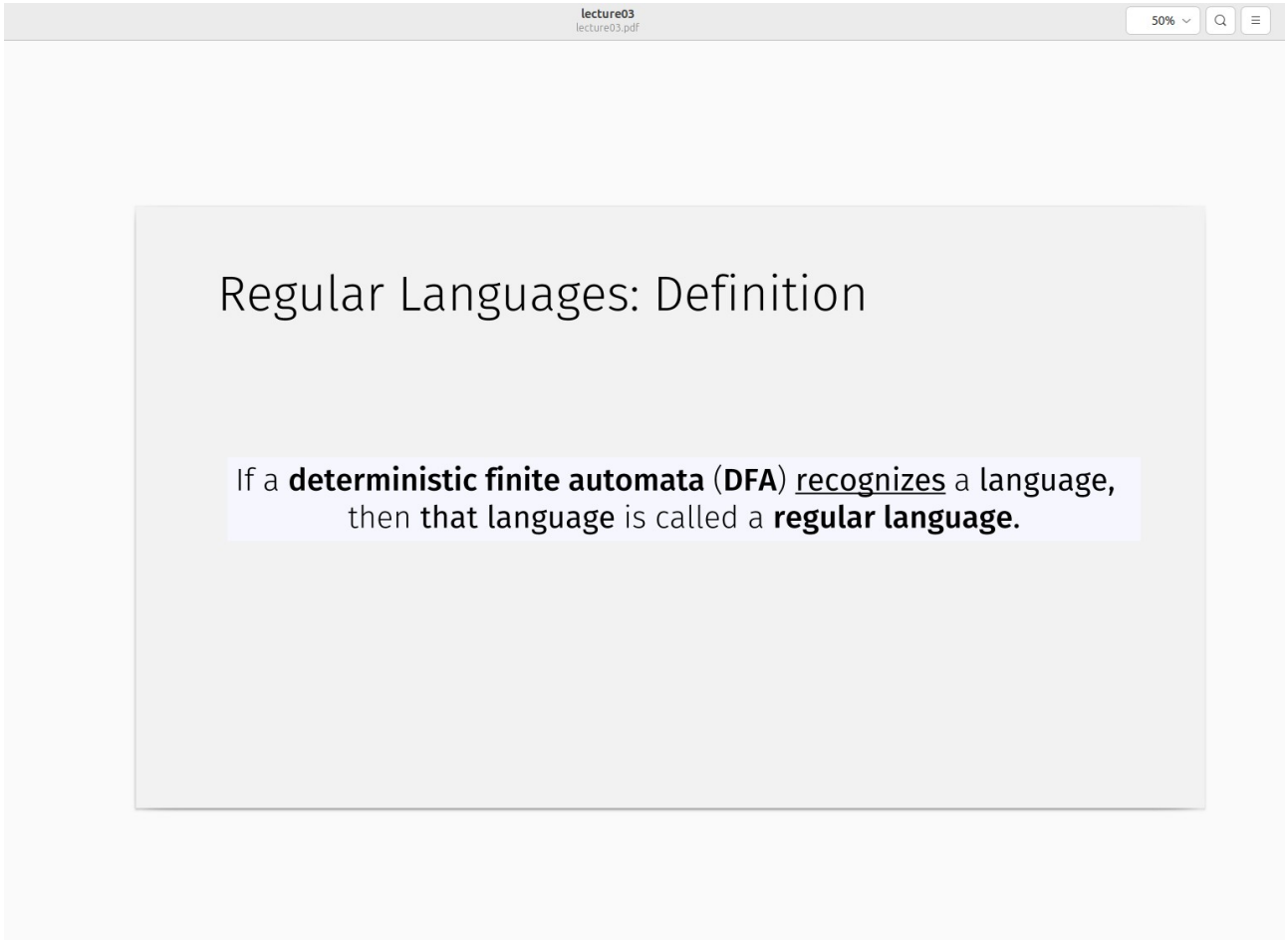
Lecture 3, slide 58

Definition of Regular Language skipped

Lecture 4, slide 1

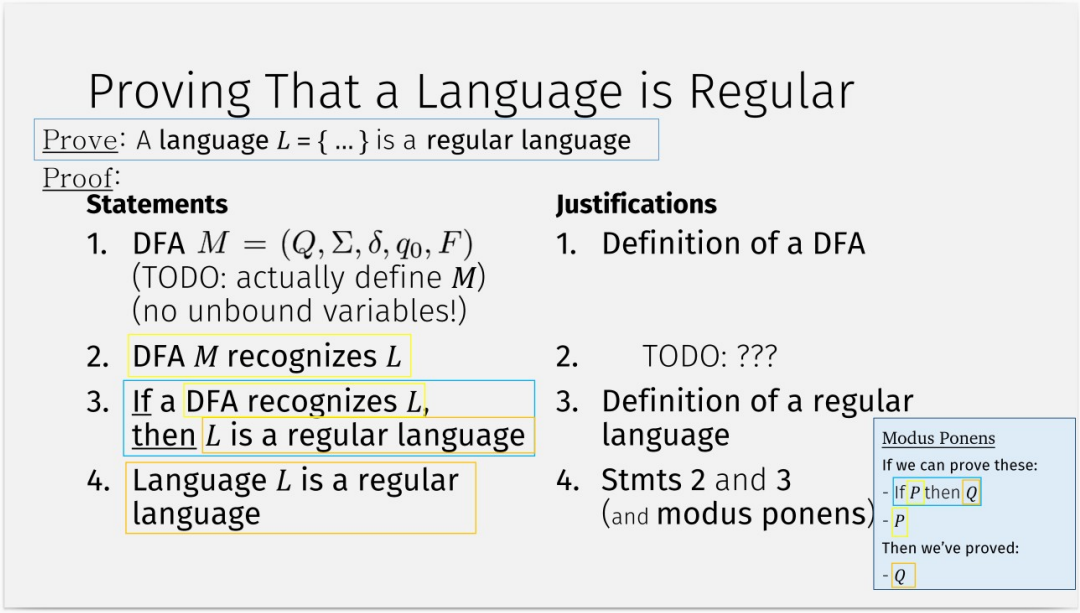


Lecture 4, slide 19



Lecture 3, slide next to last (above)

Lecture 4 continues…



Lecture 4, slide 34

"Prove" that DFA recognizes a language

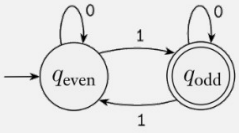
- In-class exercise (part 2):

These columns must match for the DFA to be "correct"!

String	In the language?
1	Yes
0	No
01	Yes
11	No
1101	Yes
ε	no

Confirm the DFA:
- Accepts strings in the language
- Rejects strings not in the language

$\Sigma = \{0,1\}$

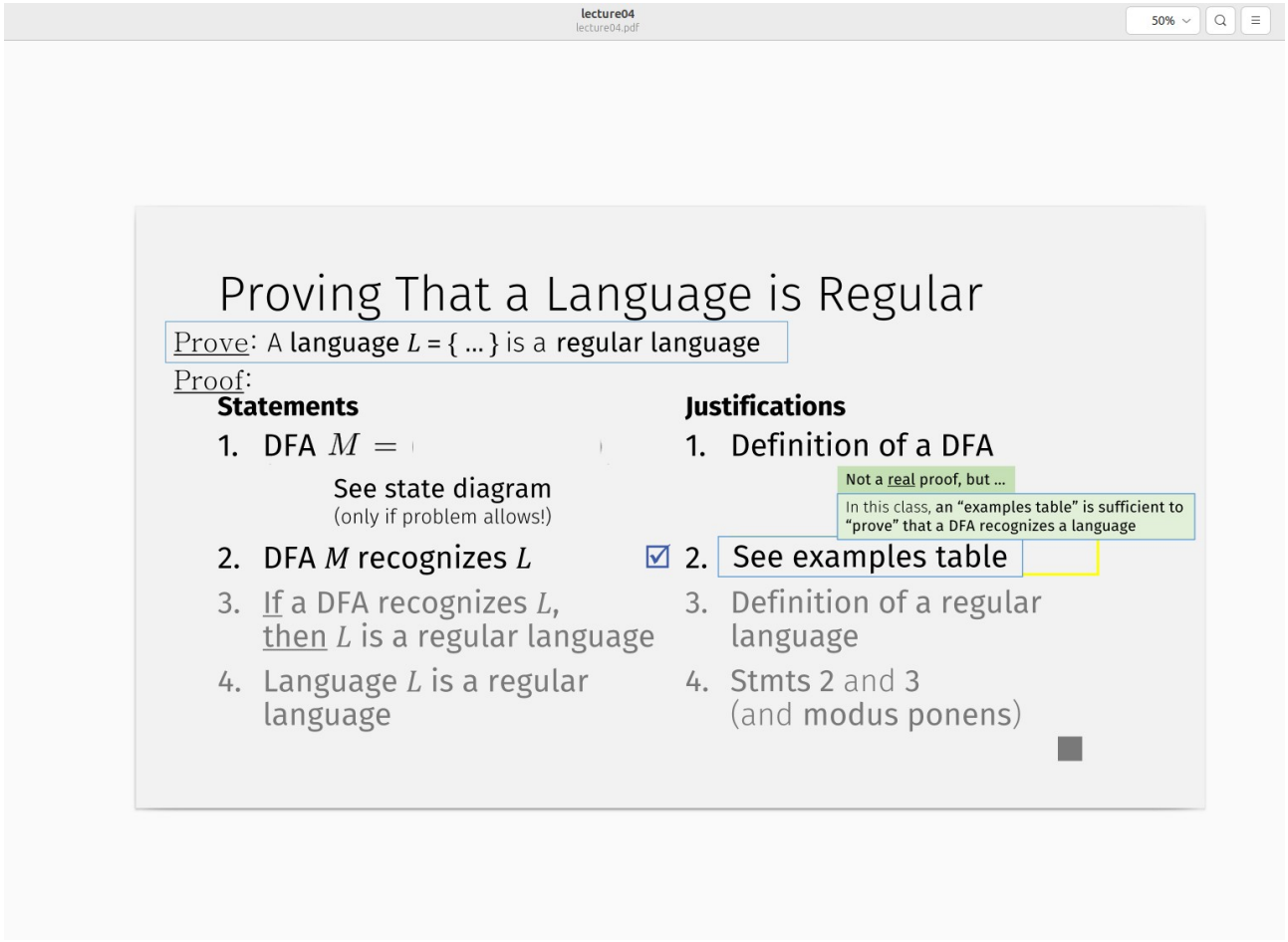


Not a real proof, but ...

In this class, a table like this is sufficient to "prove" that a DFA recognizes a language

Analogous to what programmers do (write tests) to "prove" their computation (code) "works"

Lecture 4, slide 40



Lecture 4, slide 42

In-class exercise 2

Remember:
To understand the language, always come up with string examples first (in a table)! Both:
- in the language
- and not in the language

- Prove: the following language is a regular language:
  - $A = \{ w \mid w \text{ has exactly three 1's} \}$

You will need this later in the proof anyways!

- Where Σ= {0, 1},

DEFINITION

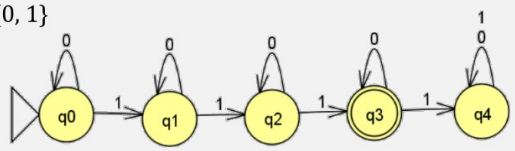
A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta: Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

Lecture 4, slide 42

In-class exercise Solution

- Design finite automata recognizing:
  - {*w* | *w* has exactly three 1's}
- *States*:
  - Need one state to represent how many 1's seen so far
  - $Q = \{q_0, q_1, q_2, q_3, q_{4+}\}$
- *Alphabet*: $\Sigma = \{0, 1\}$
- *Transitions*:



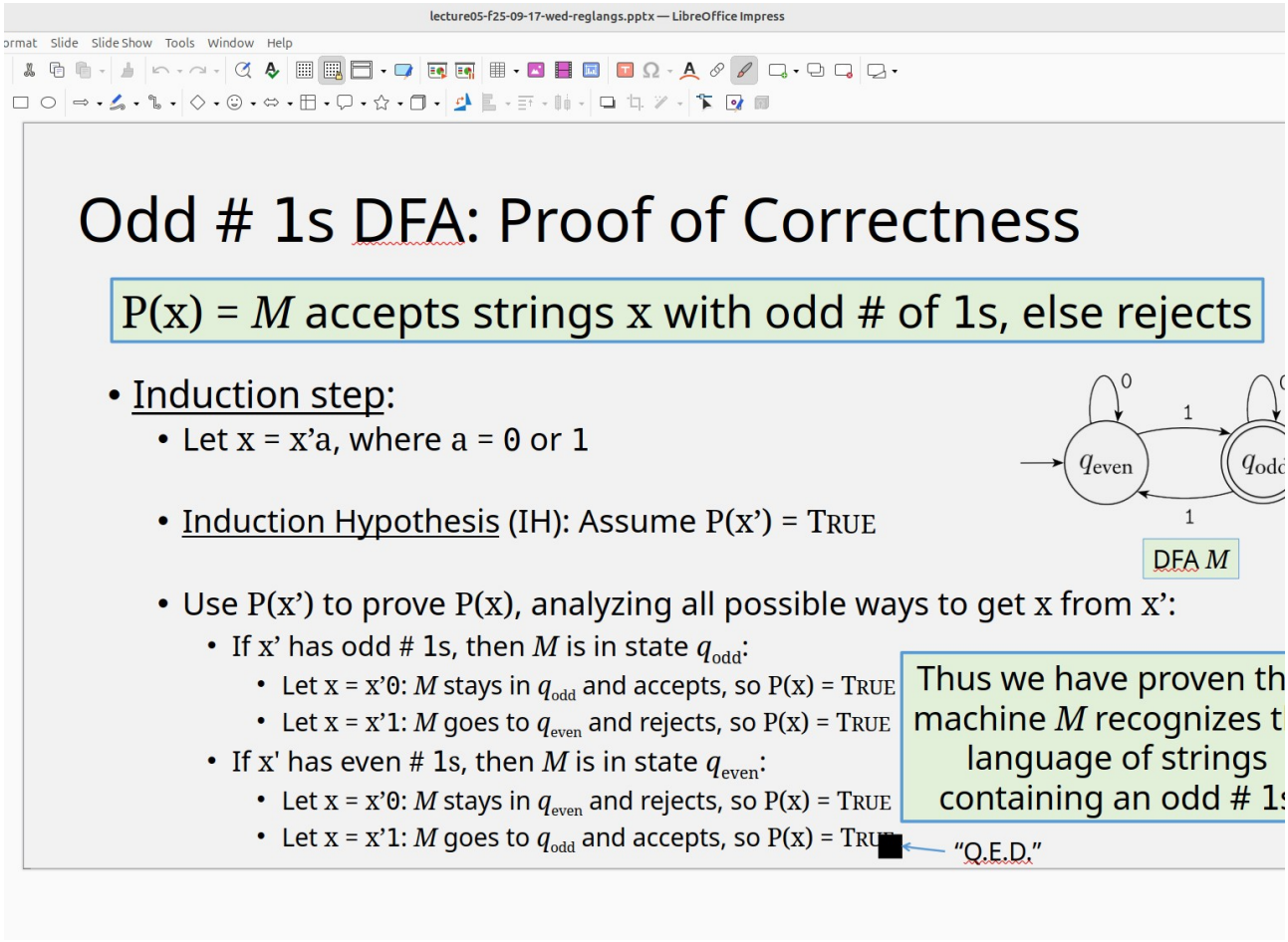
- *Start state*:
  - q_0
- *Accept states*:
  - $\{q_3\}$

So a DFA's computation recognizes simple string patterns?

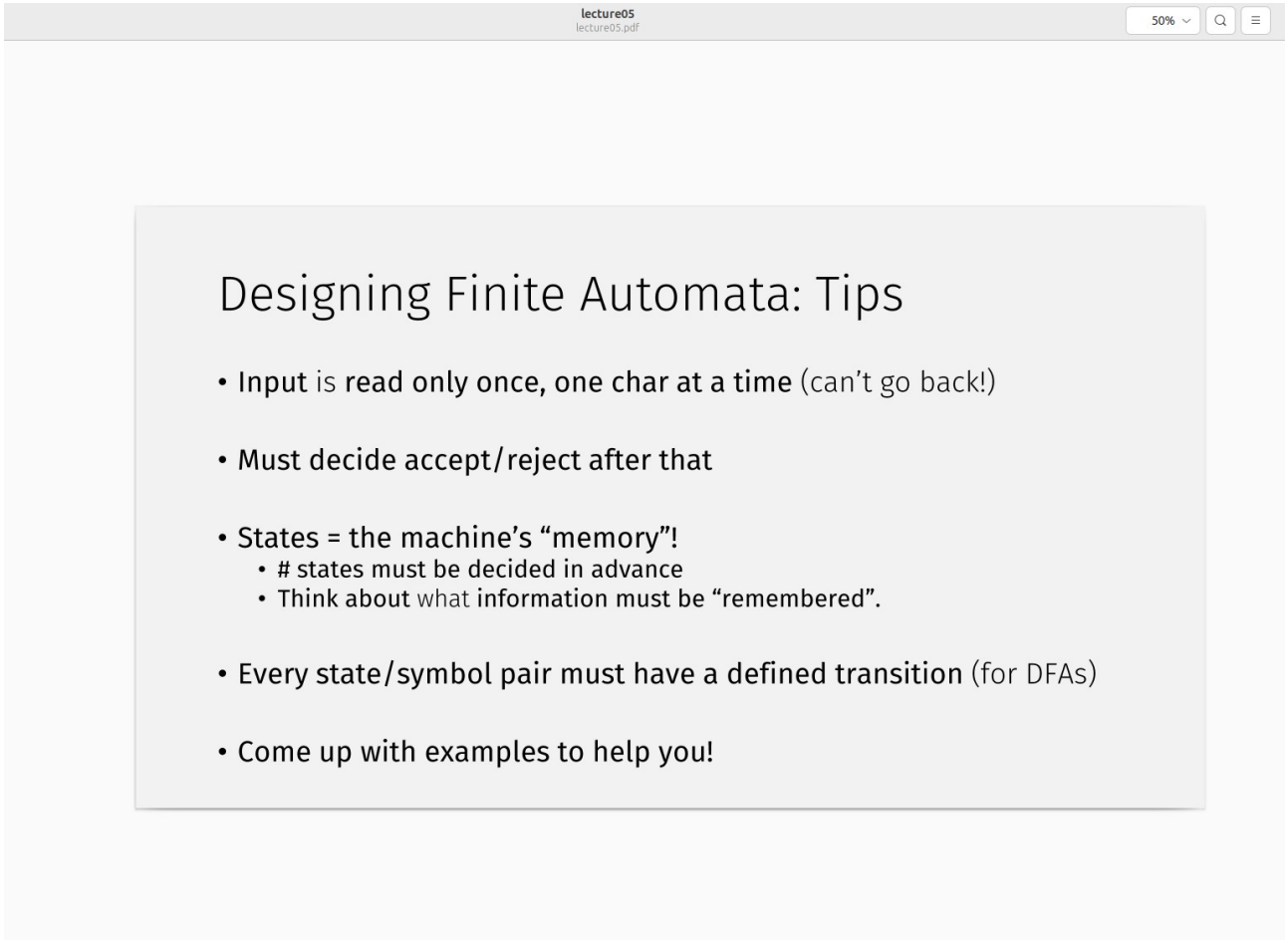
Yes!

Have you ever used a programming language feature to recognize simple string patterns?

Lecture 4, slide 44

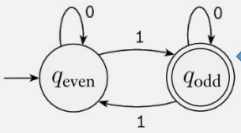

lecture05-f25-09-17-wed-reglangs.pptx — LibreOffice Impress
Format Slide Slide Show Tools Window Help
Odd # 1s DFA: Proof of Correctness
P(x) = M accepts strings x with odd # of 1s, else rejects
• Induction step:
• Let x = x'a, where a = 0 or 1
• Induction Hypothesis (IH): Assume P(x') = TRUE
• Use P(x') to prove P(x), analyzing all possible ways to get x from x':
• If x' has odd # 1s, then M is in state q_{odd}:
• Let x = x'0: M stays in q_{odd} and accepts, so P(x) = TRUE
• Let x = x'1: M goes to q_{even} and rejects, so P(x) = TRUE
• If x' has even # 1s, then M is in state q_{even}:
• Let x = x'0: M stays in q_{even} and rejects, so P(x) = TRUE
• Let x = x'1: M goes to q_{odd} and accepts, so P(x) = TRUE
0
1
q_{even}
q_{odd}
1
DFA M
Thus we have proven th
machine M recognizes t
language of strings
containing an odd # 1s
"Q.E.D."

Lecture 5:



Lecture 5, slide 15

"Prove" that DFA recognizes a language



???

Language = { w | w has odd # of 1s}

String	In the language?
1	Yes
0	No
01	Yes
11	No
1101	Yes
ε	no

$\Sigma = \{0,1\}$

Q1: How can we **"prove"** that a **computation** does what it's **"supposed to do"**?

These kinds of proofs (e.g., HMU 2.3.4) are usually **hard,** sometimes impossible!

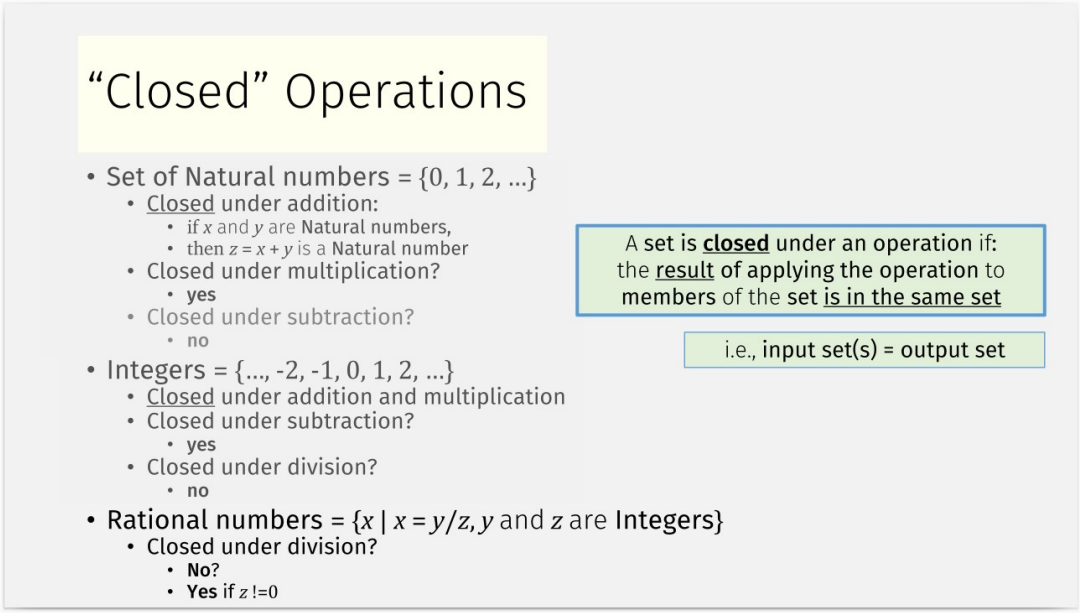
... so we will not do them in this course

But we still need to do something ...

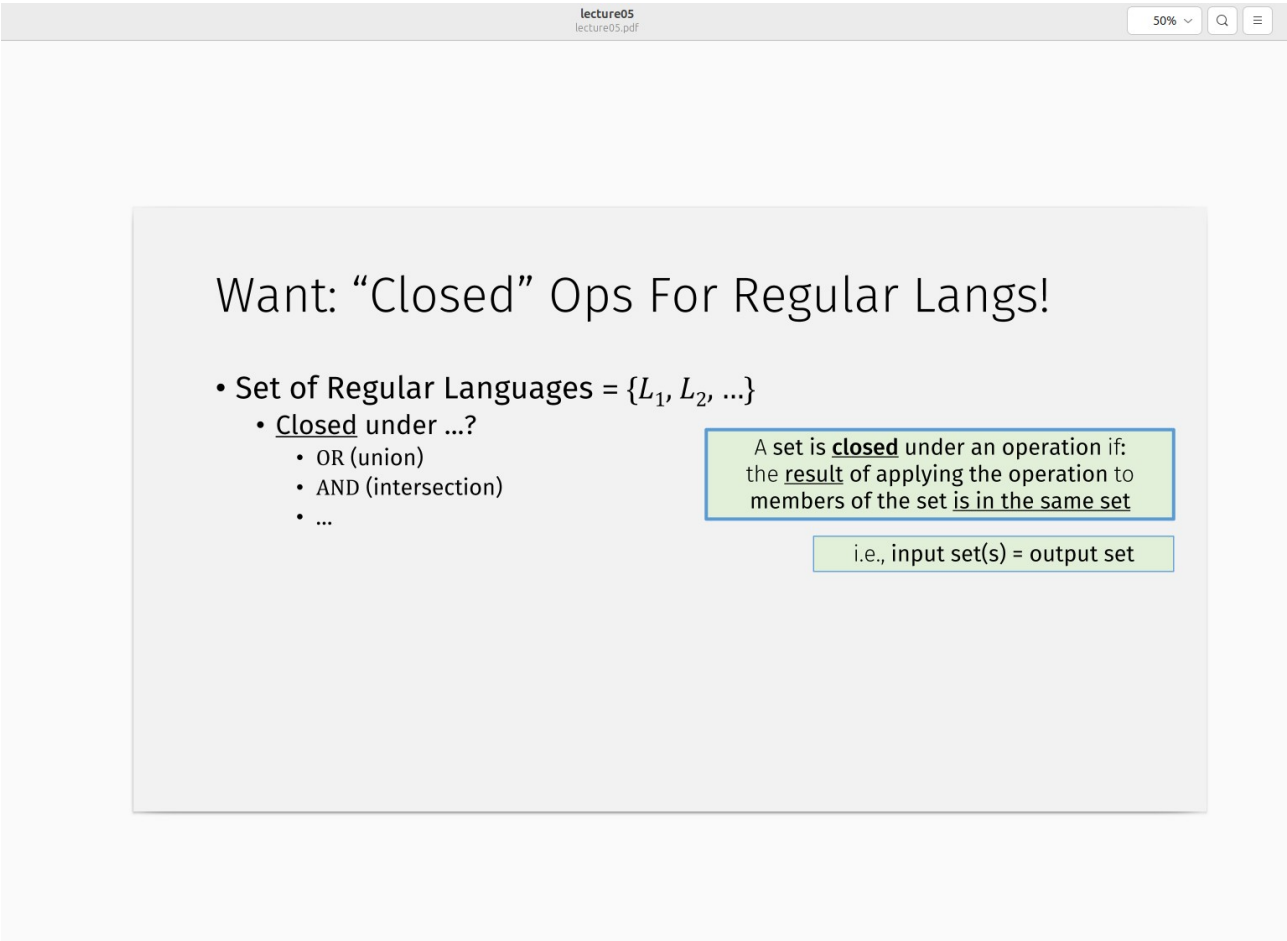
Q2: How do **programmers "prove"** that a **computation** does what it's **"supposed to do"**?

i.e., that the program is **"correct"**?

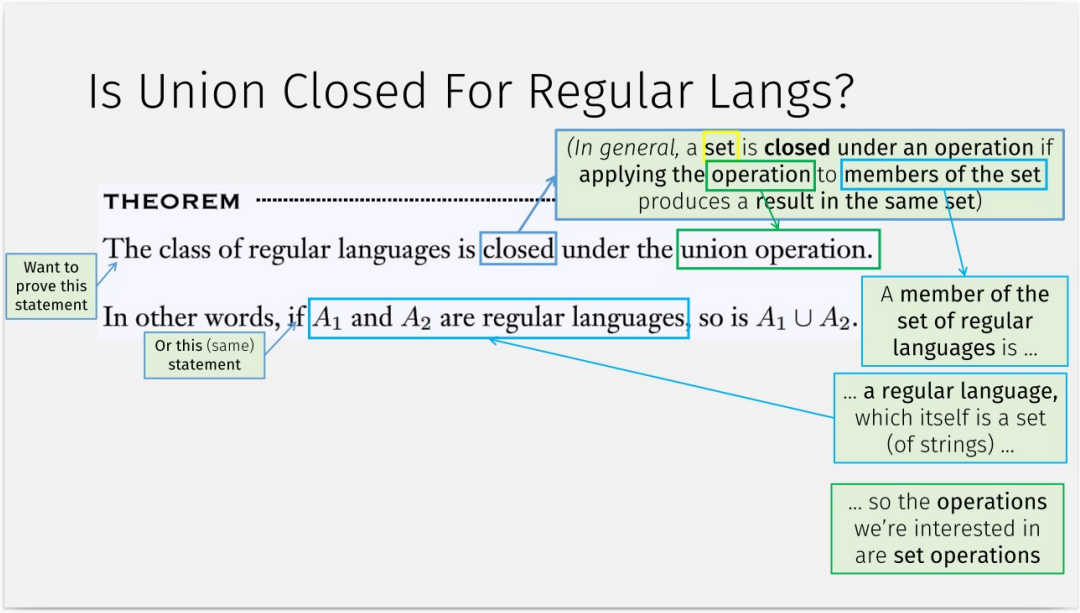
Lecture 5, slide 17



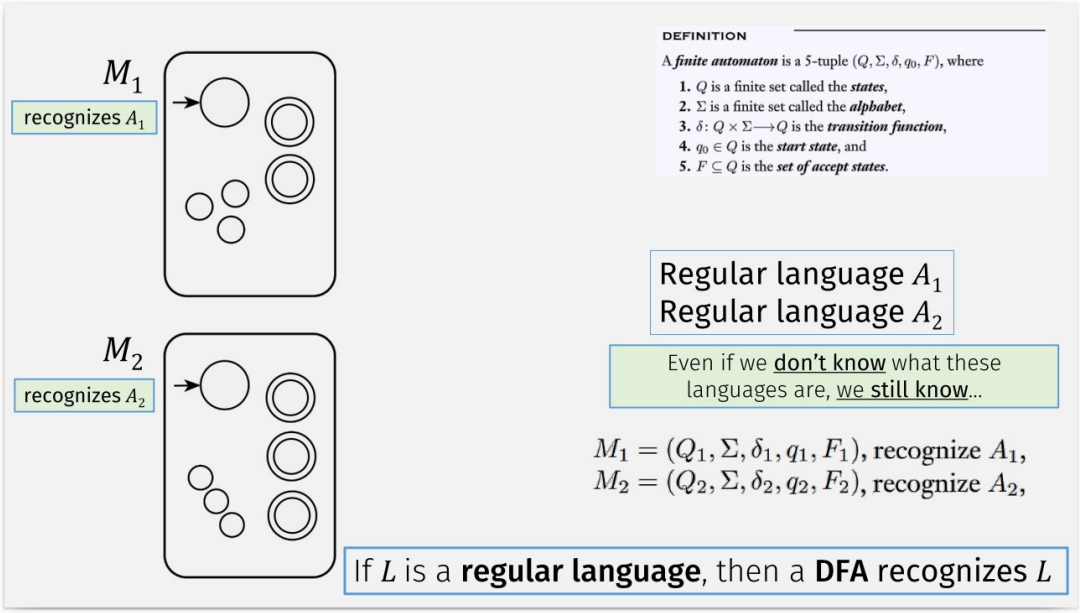
Lecture 5, slide 18



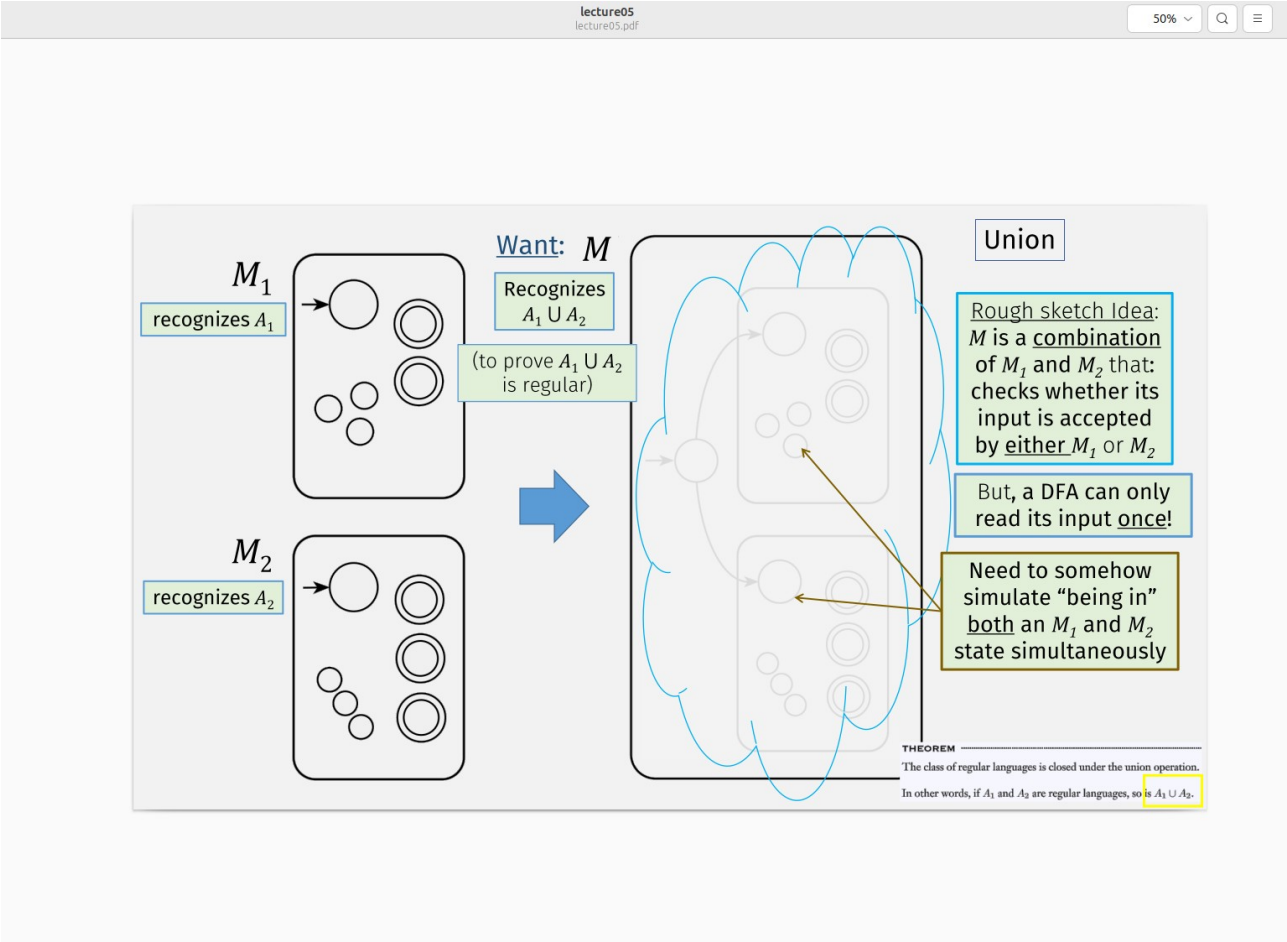
Lecture 5, slide 28



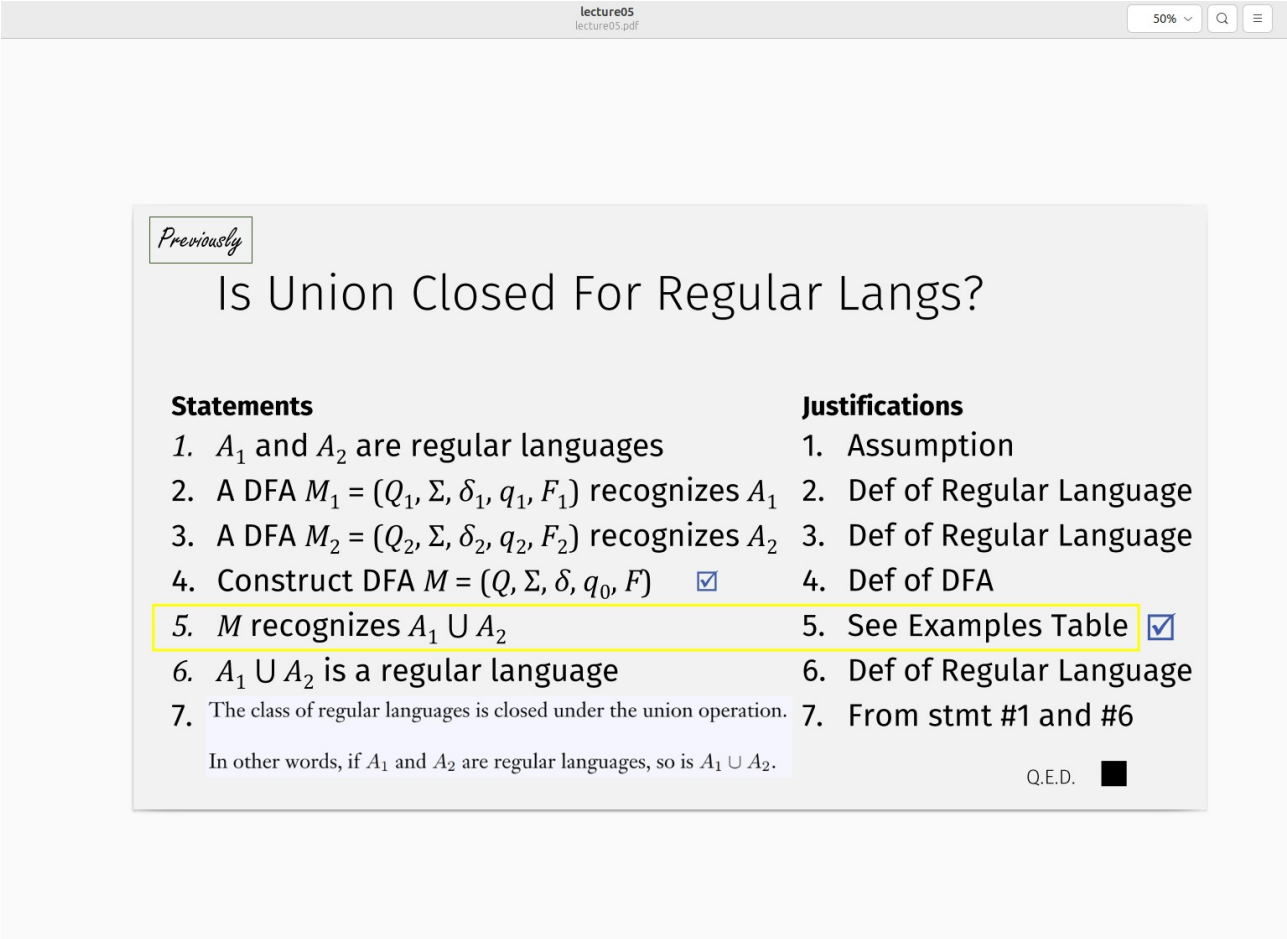
Lecture 5, slide 33



Lecture 5, slide 43



Lecture 5, slide 44



Lecture 5, slide 54

Is Concatenation Closed?

THEOREM

The class of regular languages is closed under the concatenation operation.

In other words, if A_1 and A_2 are regular languages then so is $A_1 \circ A_2$.

- **Construct a <u>new</u> machine M recognizing $A_1 \circ A_2$?** (like union)
  - Using **DFA M_1** (which recognizes A_1),
  - and **DFA M_2** (which recognizes A_2)

Lecture 5, slide 57

Union is Closed For Regular Languages

Proof (continuation)

- Given: $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, recognize A_1,
  $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, recognize A_2,

Want: M that can simultaneously "be in" both an M_1 and M_2 state

- Construct: $M = (Q, \Sigma, \delta, q_0, F)$, using M_1 and M_2, that recognizes $A_1 \cup A_2$
- states of M: $Q = \{(r_1, r_2) | \, r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$ $= Q_1 \times Q_2$

This set is the ***Cartesian product*** of sets Q_1 and Q_2

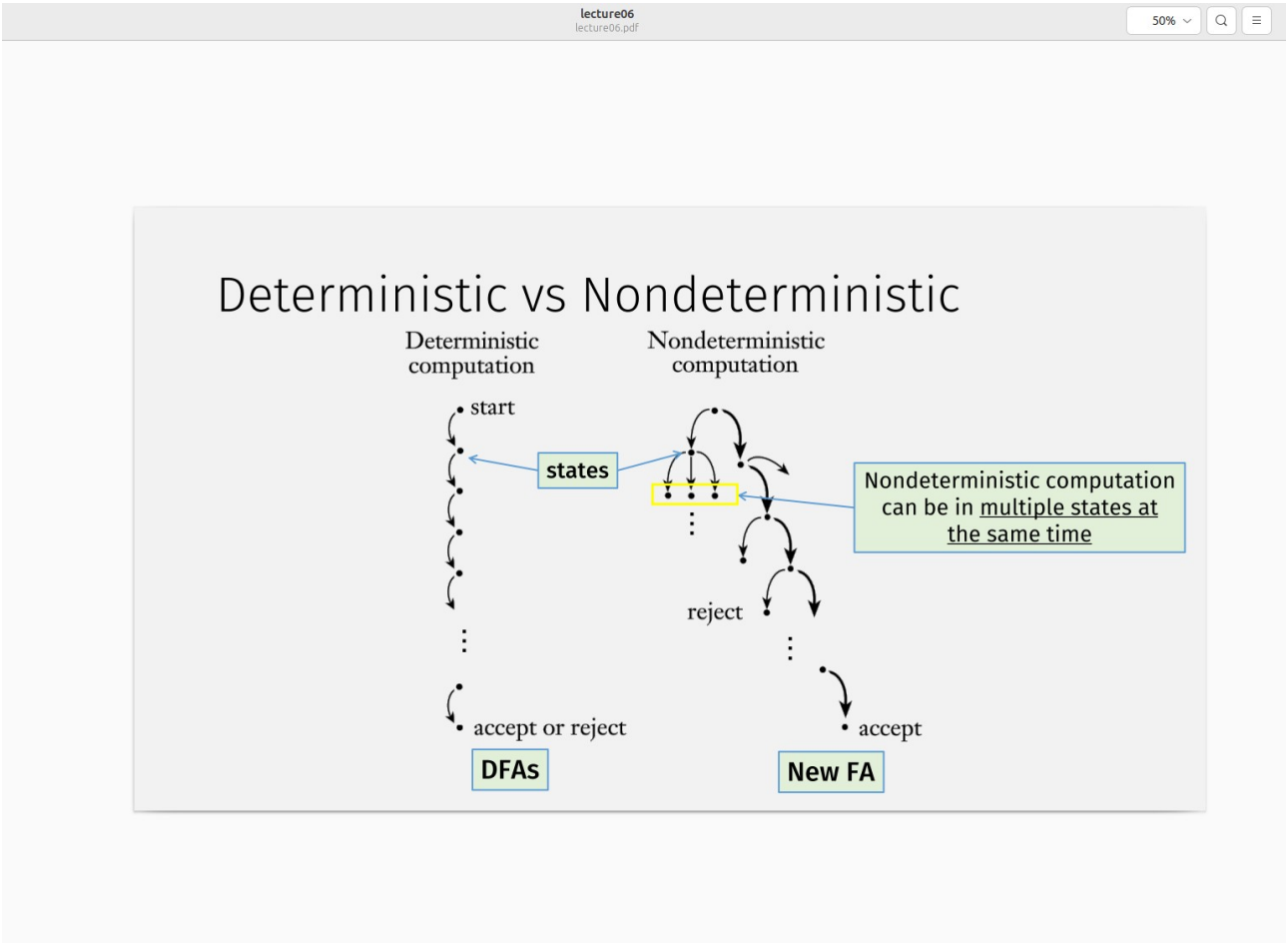
A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta: Q \times \Sigma \longrightarrow Q$ is the ***transition function***,[1]
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

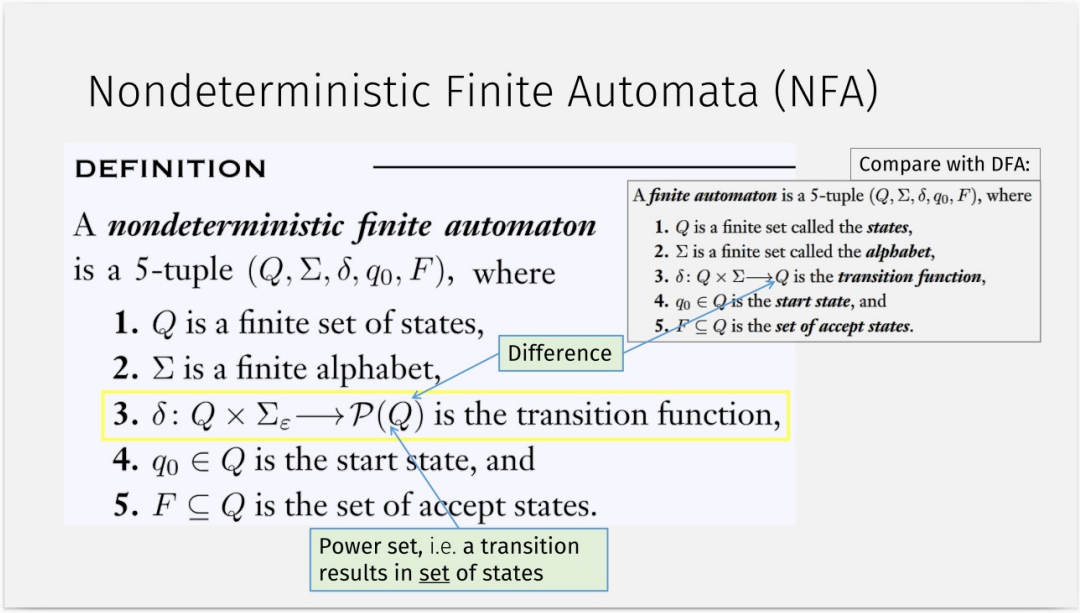
A state of M is a pair:
- first part: state of M_1
- second part: state of M_2

states of M: all possible pair combinations of states of M_1 and M_2

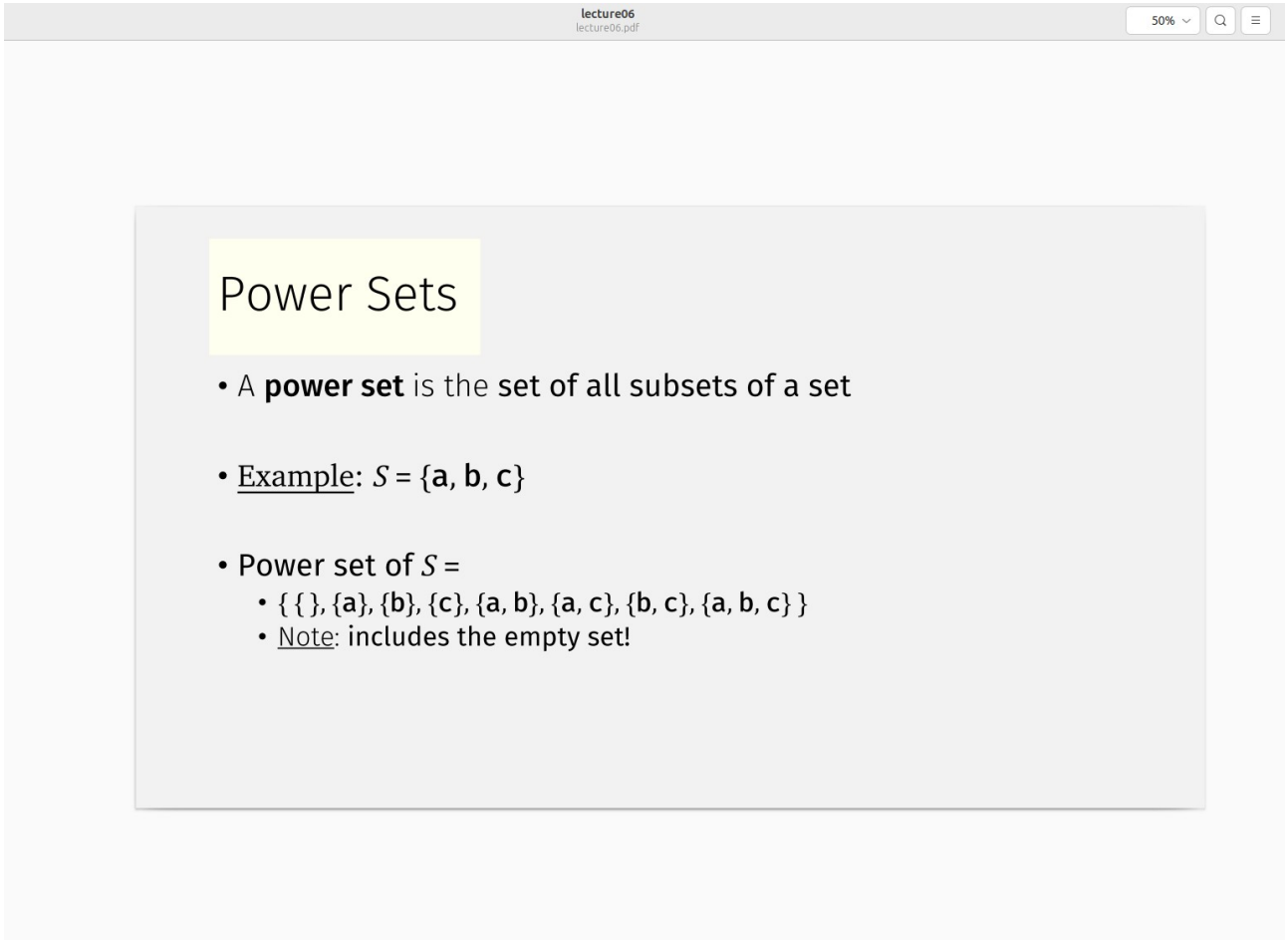
Lecture 6, slide 24



Lecture 6, slide 49 Nondeterminism



Lecture 6, slide 51



Lecture 6, slide 52 power sets

Nondeterministic Finite Automata (NFA)

DEFINITION

A ***nondeterministic finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta: Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

Transition label can be "empty", i.e., machine can transition without reading input

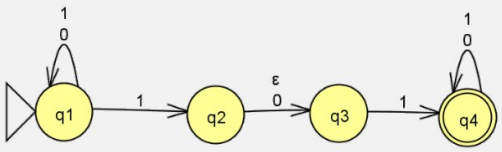
$\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$

CAREFUL:
ε symbol is reused here, as a transition label (ie, an argument to δ)
- **It's not the empty string!**
- And it's (still) not a character in the alphabet Σ!

Lecture 6, slide 53

NFA Example

- Come up with a formal description of the following NFA:



DEFINITION

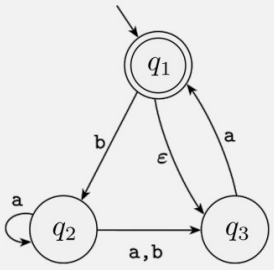
A ***nondeterministic finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta: Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

Lecture 6, slide 54 and the following ones that give the answer

In-class Exercise

- Come up with a formal description for the following NFA
  - $\Sigma = \{ \texttt{a}, \texttt{b} \}$

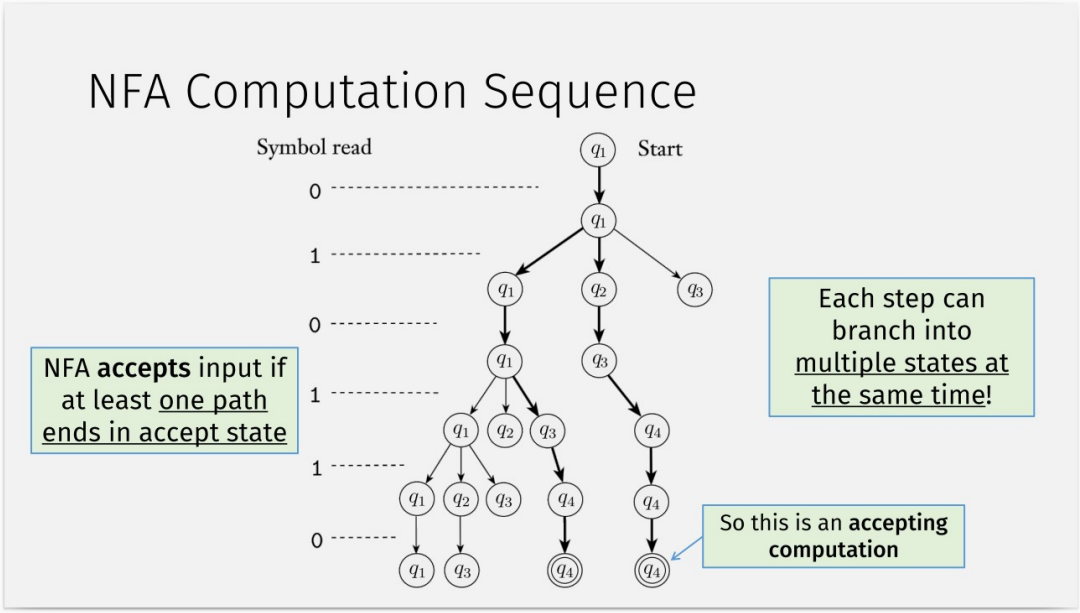


DEFINITION

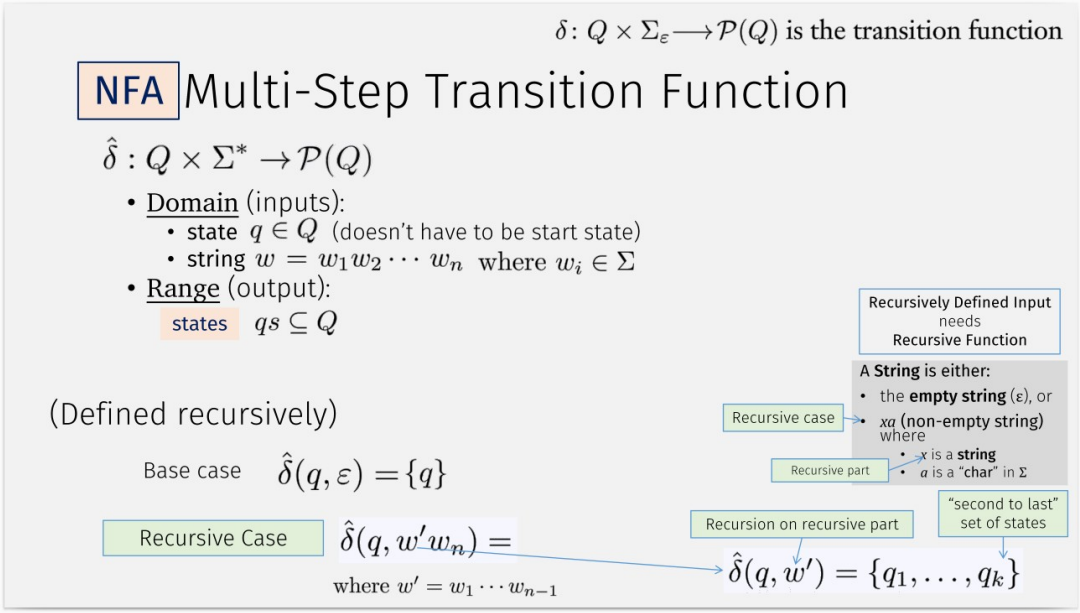
A ***nondeterministic finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta: Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

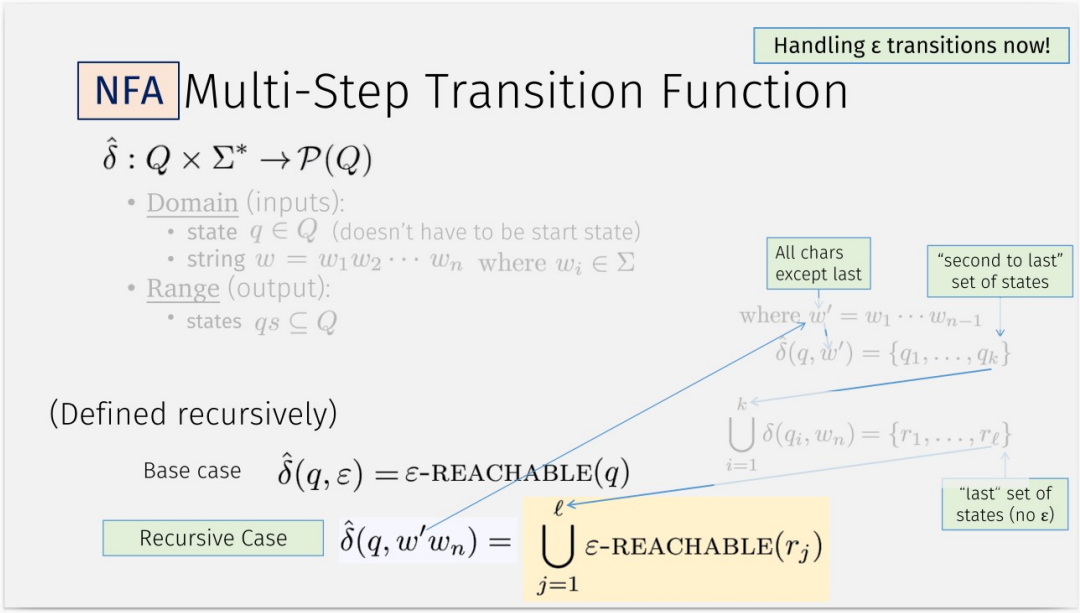
Lecture 6, slide 56



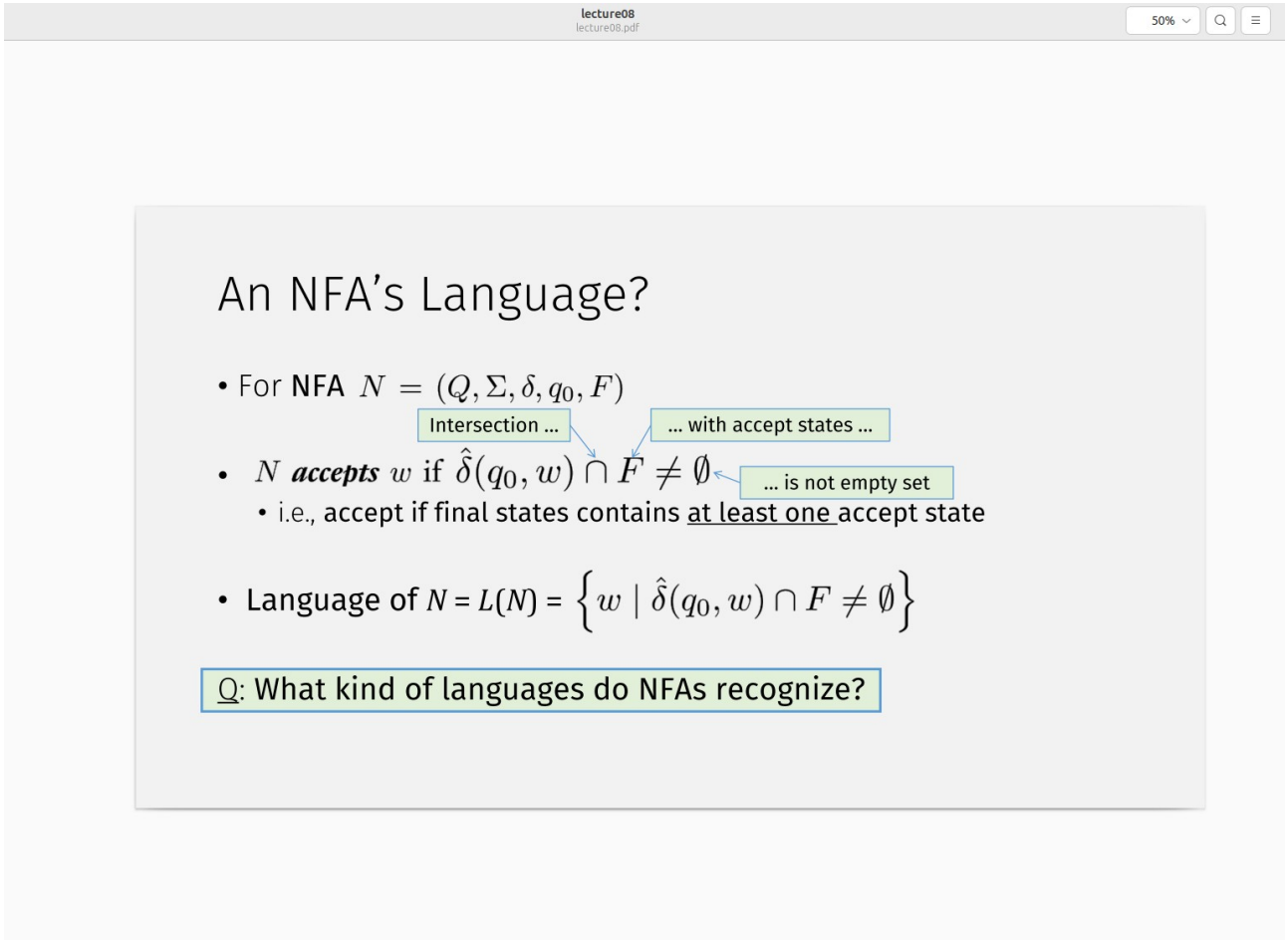
Lecture 7, slide 50



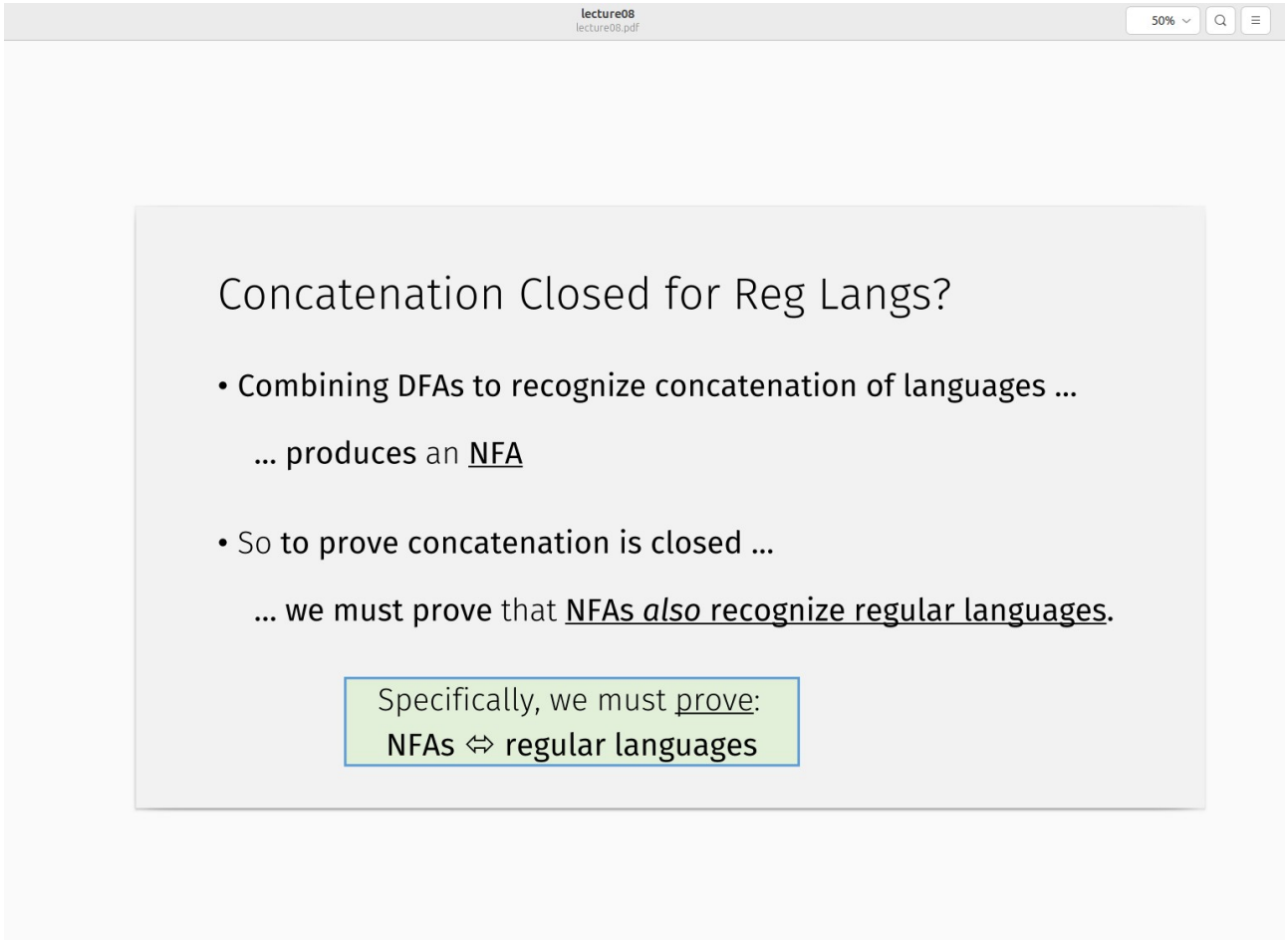
Lecture 7, slide 60



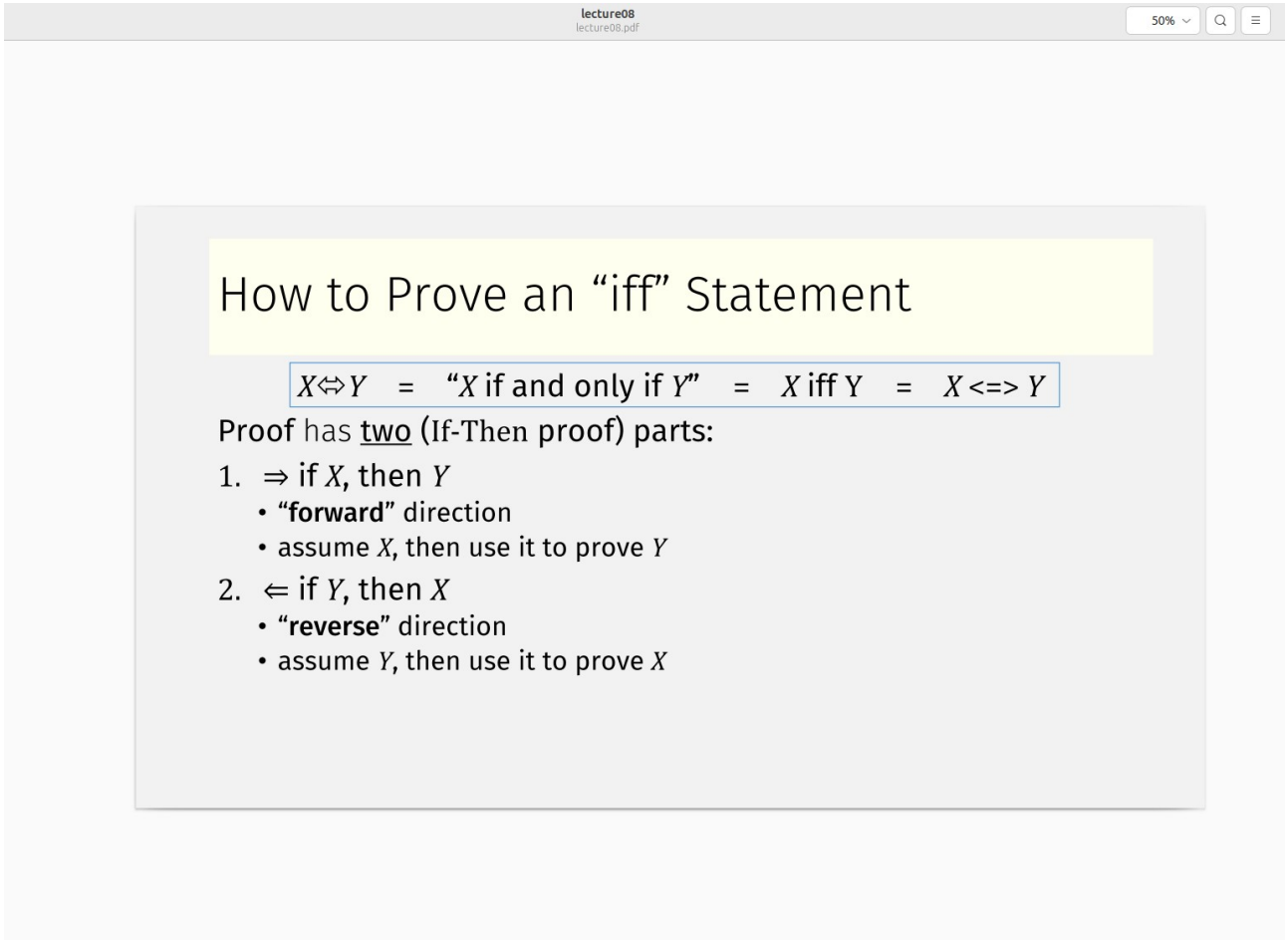
Lecture 7, slide 67



Lecture 7, slide 45 NFAs language



Lecture 8, slide 46



Lecture 8, slide 48

Proving NFAs Recognize Regular Langs

Theorem:
A language L is regular **if and only if** some NFA N recognizes L.

Proof:

☑ ⇒ If L is regular, then some NFA N recognizes it.
(Easier)
- We know: if L is **regular**, then a **DFA** exists that recognizes it.
- So to prove this part: Convert that DFA → an equivalent NFA! (see HW 4)

⇐ If an NFA N recognizes L, then L is regular.
(Harder)
- We know: for L to be **regular**, there must be a **DFA** recognizing it
- Proof Idea for this part: Convert given NFA N → an equivalent DFA

"equivalent" =
"recognizes the same language"

Lecture 8, slide 53

How to convert NFA→DFA?

A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

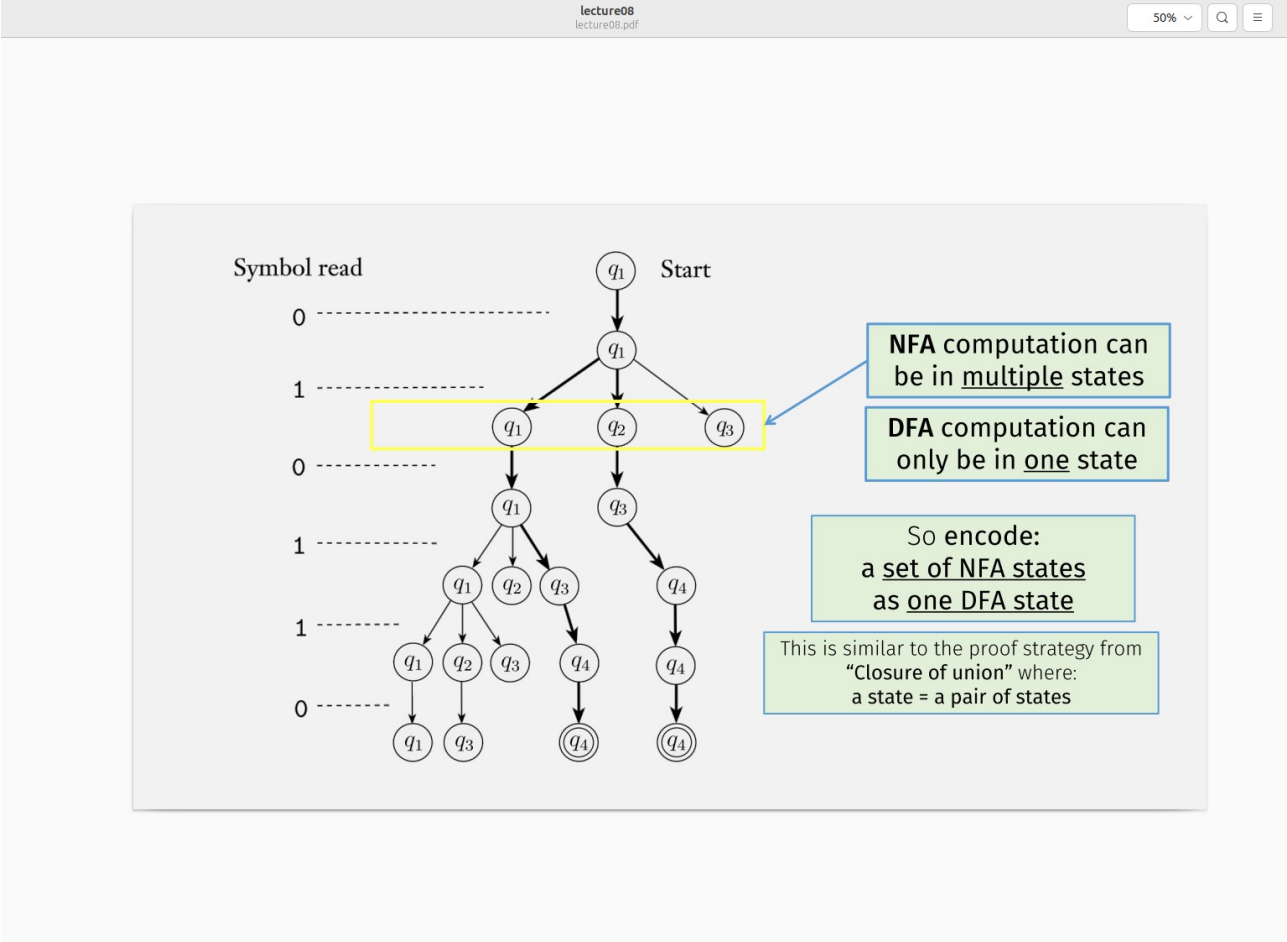
1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta: Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

A ***nondeterministic finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta: Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

Proof idea:
Let each "state" of the DFA
= set of states in the NFA

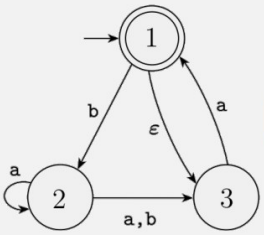
Lecture 8, slide 54



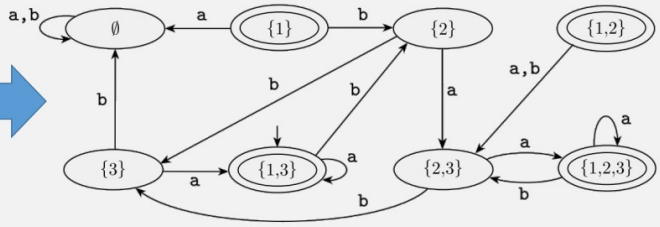
Lecture 8, slide 55

Example:

- Let NFA $N_4 = (Q, \Sigma, \delta, q_0, F)$
- An equivalent DFA D has states $= \mathcal{P}(Q)$ (power set of Q)



The NFA N_4



A DFA D that is equivalent to the NFA N_4

Lecture 8, slide 57

With empty transitions

NFA→DFA

Have: NFA $N = (Q_{NFA}, \Sigma, \delta_{NFA}, q_{0NFA}, F_{NFA})$
Want: DFA $D = (Q_{DFA}, \Sigma, \delta_{DFA}, q_{0DFA}, F_{DFA})$

Almost the same, except ...

1. $Q_{DFA} = \mathcal{P}(Q_{NFA})$
2. For $qs \in Q_{DFA}$ and $a \in \Sigma$
   - $\delta_{DFA}(qs, a) = S = \bigcup_{q \in qs} \delta_{NFA}(q, a)$ $\bigcup_{s \in S} \varepsilon\text{-REACHABLE}(s)$
3. $q_{0DFA} = \{q_{0NFA}\}$ $\varepsilon\text{-REACHABLE}(q_{0NFA})$
4. $F_{DFA} = \{ qs \in Q_{DFA} \mid qs \text{ contains accept state of } N \}$

Lecture 8, slide 60

New possible proof strategy!

Sipser Theorem 1.47

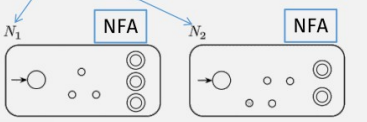
Concat Closed for Reg Langs: Use NFAs Only

PROOF

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1, and
$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2.

NFAs

If language is **regular**, then it has an **NFA** recognizing it ...



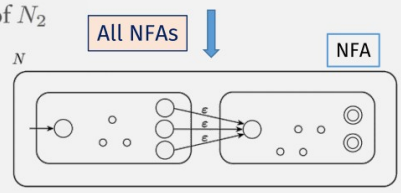
$\text{CONCAT}_{NFA}(N_1, N_2) = N = (Q, \Sigma, \delta, q_1, F_2)$ to recognize $A_1 \circ A_2$

1. $Q = Q_1 \cup Q_2$
2. The state q_1 is the same as the start state of N_1
3. The accept states F_2 are the same as the accept states of N_2
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

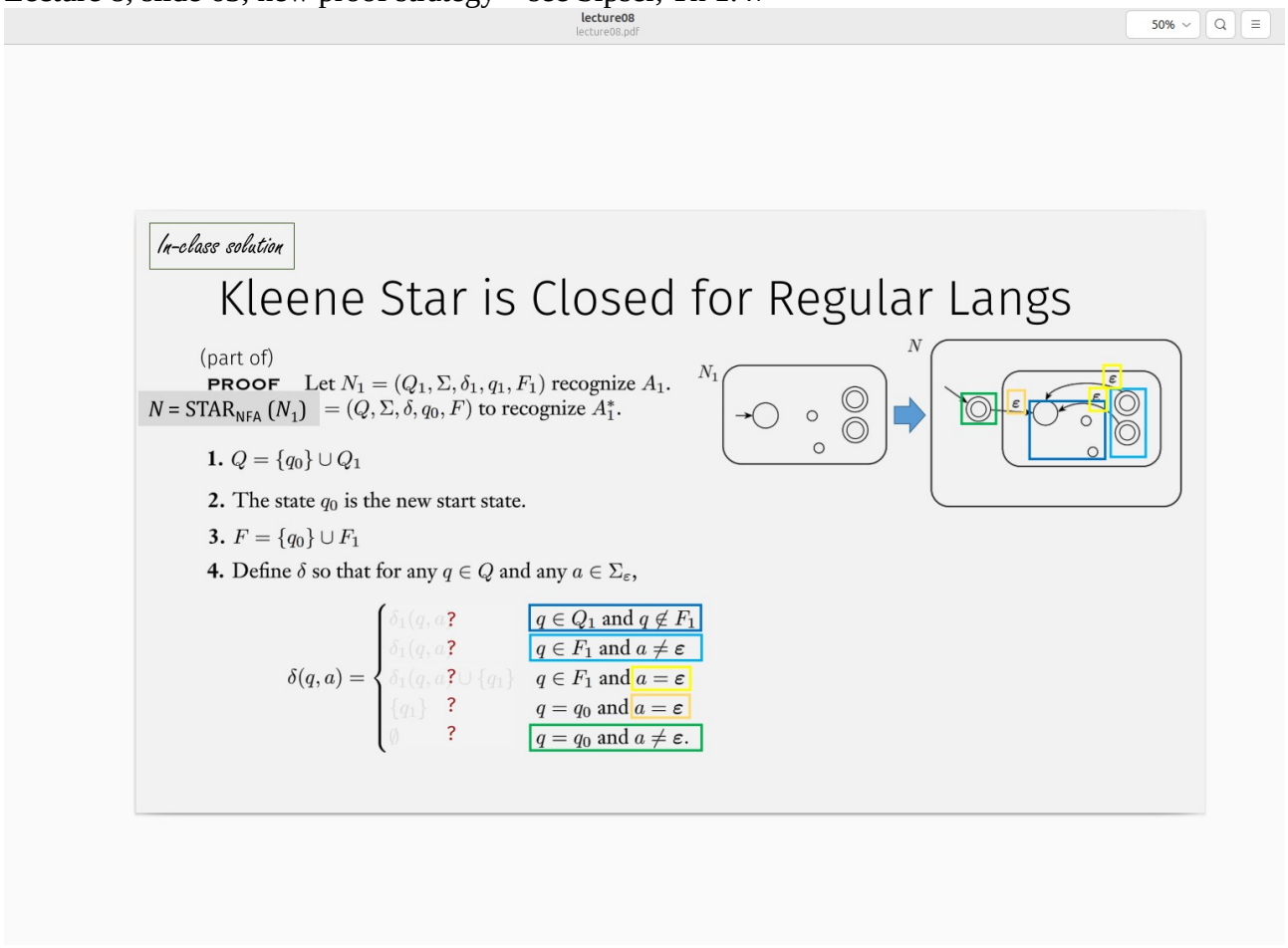
????

$$\delta(q,a) = \begin{cases} ? & q \in Q_1 \text{ and } q \notin F_1 \\ ? & q \in F_1 \text{ and } a \neq \varepsilon \\ ? \quad \{q_2\} & q \in F_1 \text{ and } a = \varepsilon \\ ? & q \in Q_2. \end{cases}$$

All NFAs



Lecture 8, slide 63, new proof strategy – see Sipser, Th 1.47



Lecture 8, slide 75

Next Time: Why These Closed Operations?

- Union
- Concat
- Kleene star

All regular languages can be constructed from:
- single-char strings, and
- these three combining operations!

Lecture 8, slide 76

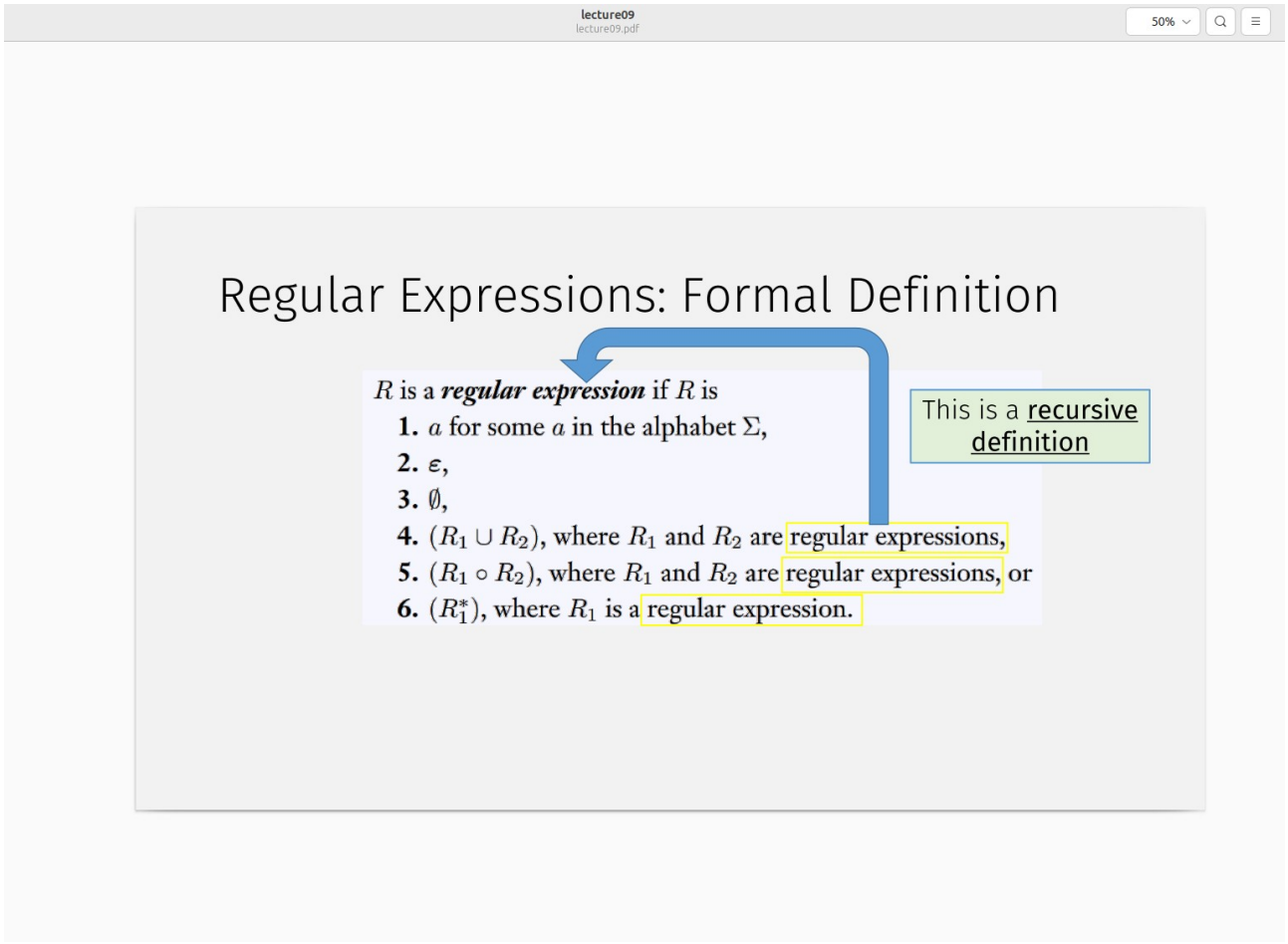
Why These (Closed) Operations?

- Union $A \cup B = \{x | \, x \in A \text{ or } x \in B\}$
- Concatenation $A \circ B = \{xy | \, x \in A \text{ and } y \in B\}$
- Kleene star (repetition) $A^* = \{x_1 x_2 \ldots x_k | \, k \geq 0 \text{ and each } x_i \in A\}$

All **regular languages** can be constructed from:
- (language of) **single-char strings** (from some alphabet), and
- these **three closed operations!**

They are used to define **regular expressions!**

Lecture 9, slide 46



Lecture 9, slide 47