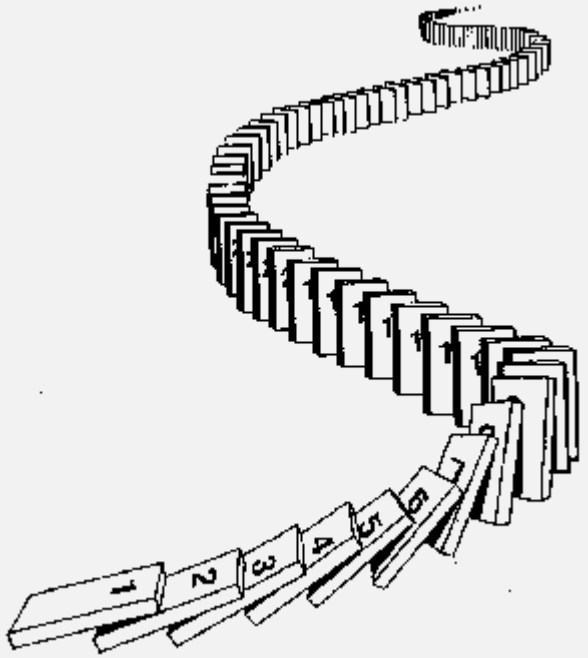
CS 420 / CS 620

Inductive Proofs

(recursive)

Wednesday, October 8, 2025

UMass Boston Computer Science



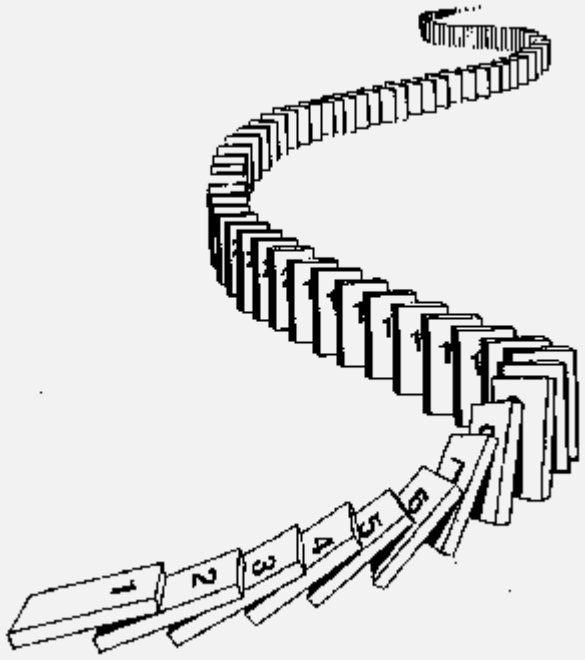
Announcements

- HW 5
  - Out: Mon 10/6 12pm (noon)
  - Due (unofficial): Mon 10/13 12pm (noon)
  - Due (up to): Wed 10/15 12pm (noon)

(stay on schedule!)

- HW 6 (most likely)
  - Out: Mon 10/13 12pm (noon)
  - Due: Wed 10/20 12pm (noon)

- No class: next Mon 10/13 (Indigenous Peoples)



ast Time

Regular Expressions = Regular Langs?

R is a ***regular expression*** if R is

1. a for some a in the alphabet Σ,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is a regular expression.

We would like to say:

a regular expression ... → equivalent → a regular language

the *set of all* regular expressions ... → equivalent → the *set of all* regular languages

(But we have to prove it)

Thm: A Lang is Regular iff Some Reg Expr Describes It

⇒ If a language is regular, then it's described by a reg expression

⇐ If a language is described by a reg expression, then it's regular (Easier)

- Key step: convert reg expr $\rightarrow$ equivalent NFA!
- (Hint: we mostly did this already when discussing closed ops)

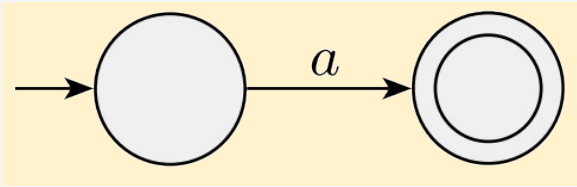
How to show that a language is regular?

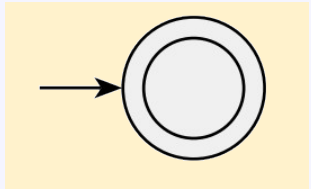
Construct a DFA *or* NFA!

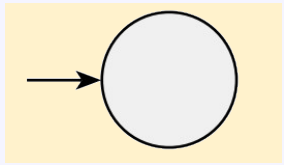
RegExpr→NFA

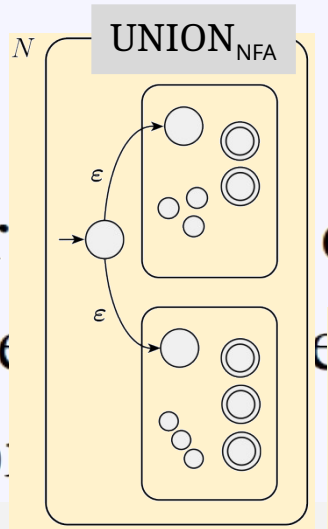
R is a ***regular expression*** if R is

1. a for some a in the alphabet Σ,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 ar… expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are…
6. (R_1^*), where R_1 is a regular exp…

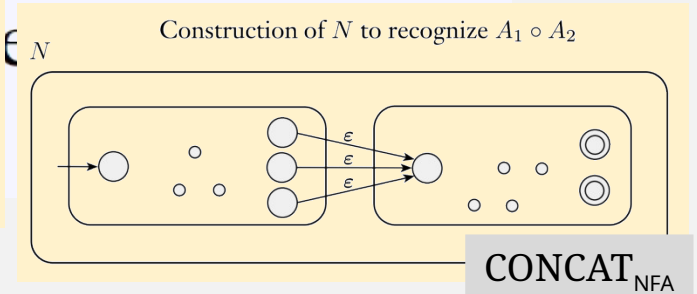




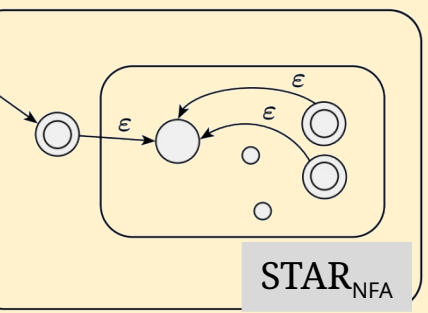




UNION$_{NFA}$



CONCAT$_{NFA}$

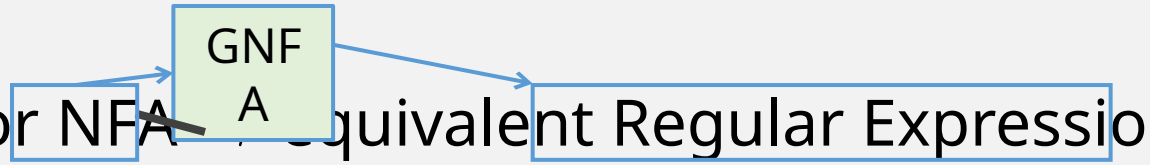


STAR$_{NFA}$

Thm: A Lang is Regular iff Some Reg Expr Describes It

⇒ If a language is regular, then it's described by a reg expression (Harder)

- Key step: Convert an ~~DFA or NFA~~ GNFA to an equivalent Regular Expression



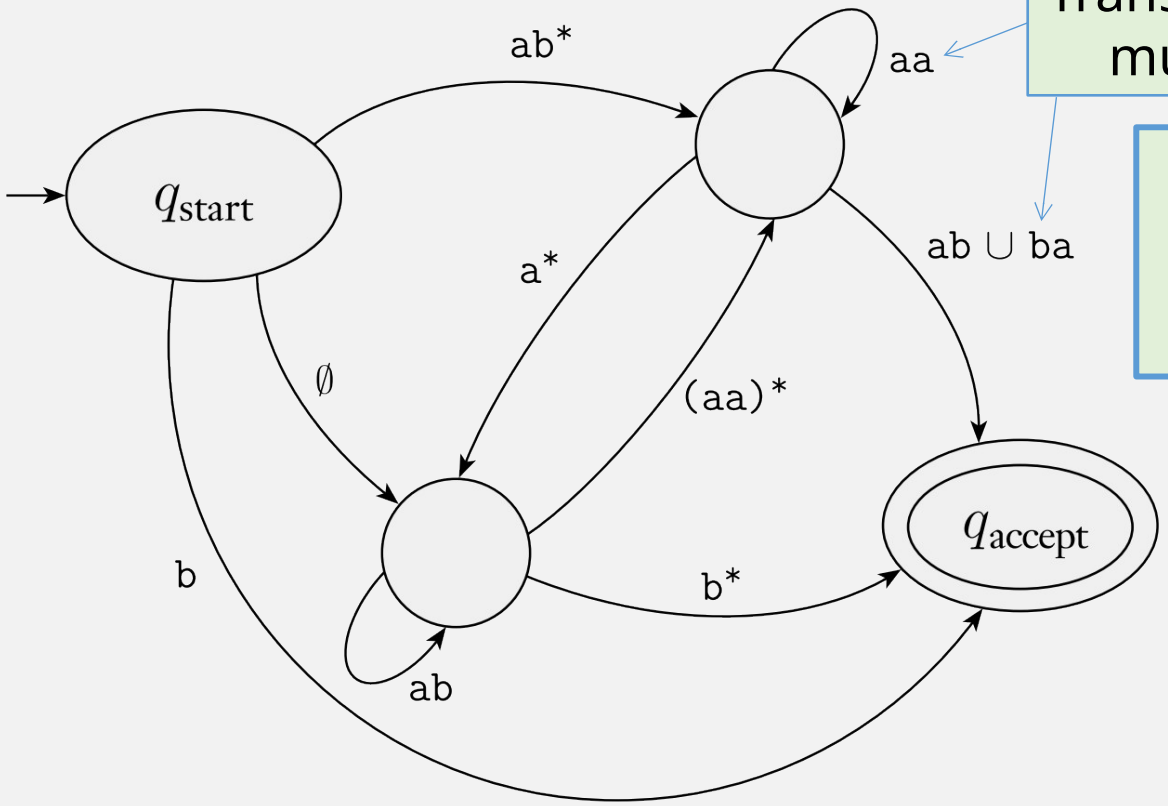
- First, we need another kind of finite automata: a GNFA

⇐ If a language is described by a reg expression, then it's regular (Easier)

- Key step: Convert the regular expression → an equivalent NFA!

(full proof requires writing Statements and Justifications, and creating an "Equivalence" Table)

Generalized NFAs (GNFAs)



Transition can read multiple chars

plain NFA
= GNFA with single char regular expr transitions

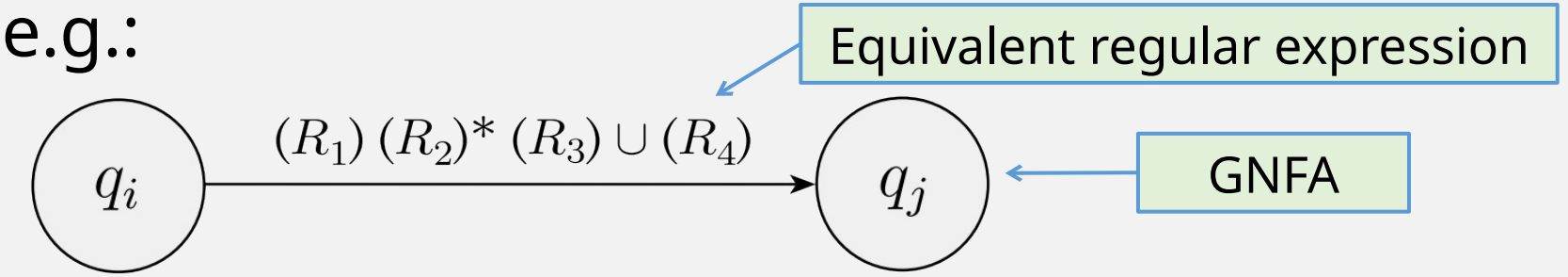
Goal: convert GNFAs to equivalent Regular Exprs

- GNFA = NFA with regular expression transitions

GNFA→RegExpr function :

On GNFA input G:

- If G has 2 states, return the regular expression (on the transition), e.g.:

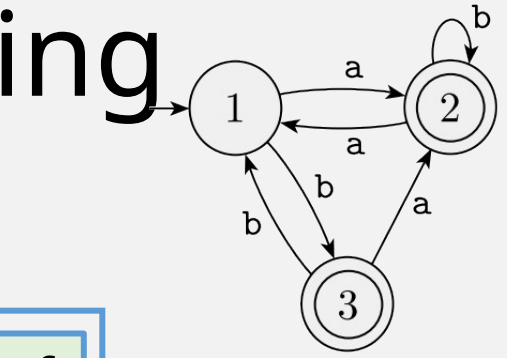


Could there be less than 2 states?

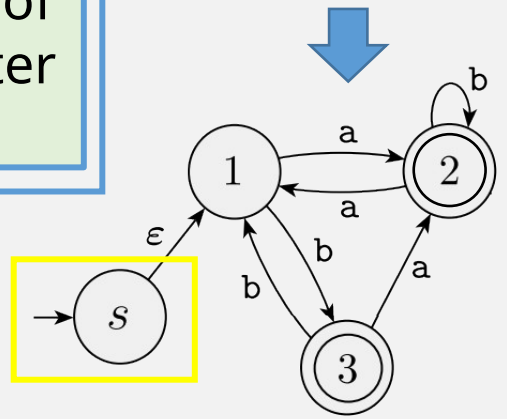
GNFA→RegExpr Preprocessing

- Modify input machine to have:

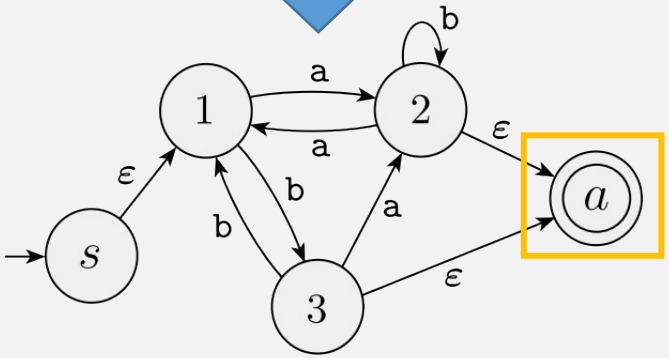
Does this change the language of the machine? i.e., are before/after machines equivalent?



- New start state:
  - No incoming transitions
  - ε transition to old start state



- New, single accept state:
  - With ε transitions from old accept states



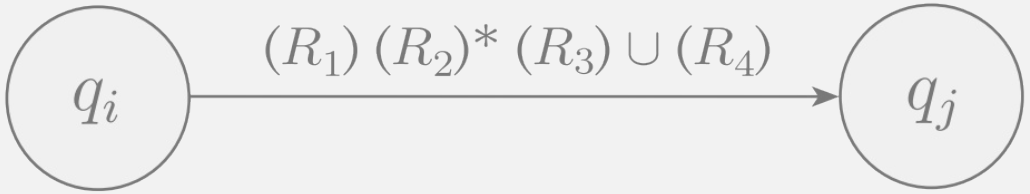
Modified machine always has 2+ states:
- New start state
- New accept state

GNFA→RegExpr function (recursive)

On GNFA input G:

Base Case

- If G has 2 states, return the regular expression (from transition), e.g.:

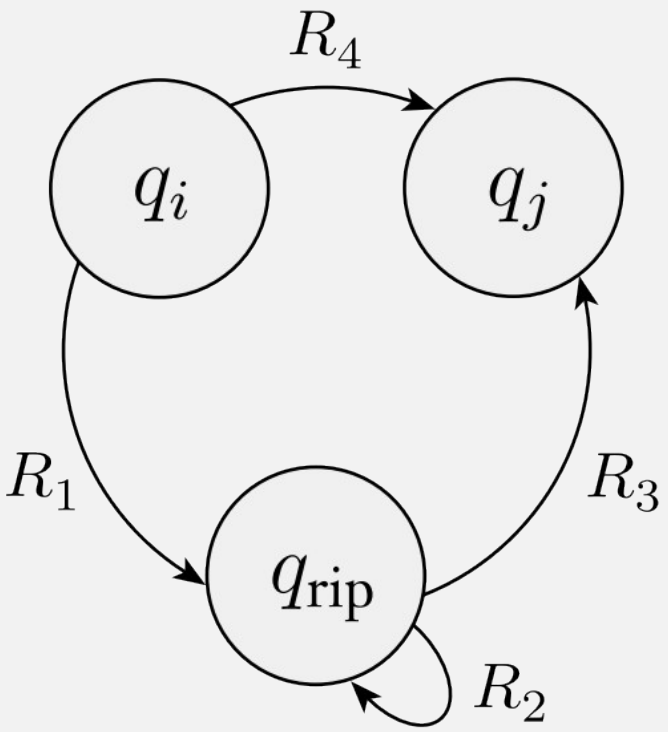


Recursive Case

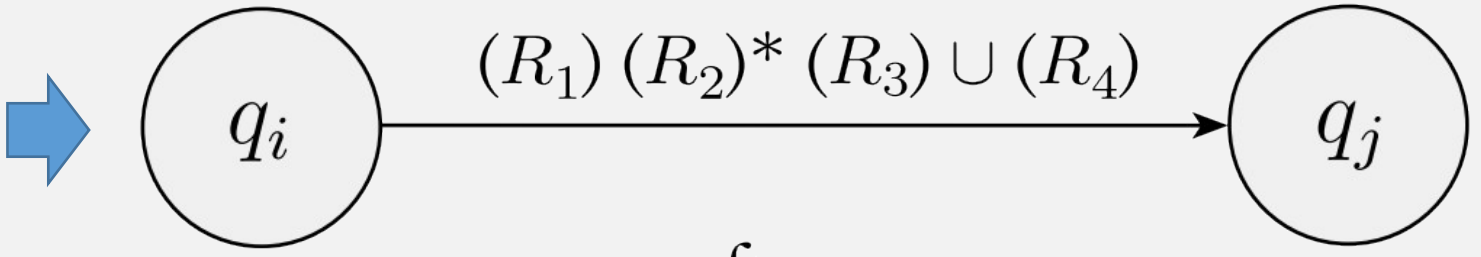
- Else:
  - "Rip out" one state
  - "Repair" the machine to get an equivalent GNFA G'
  - Recursively call GNFA→RegExpr(G')

Recursive definitions have:
- base case and
- recursive case
  (with "smaller" self-reference)

GNFA→RegExpr: “Rip/Repair” step



before

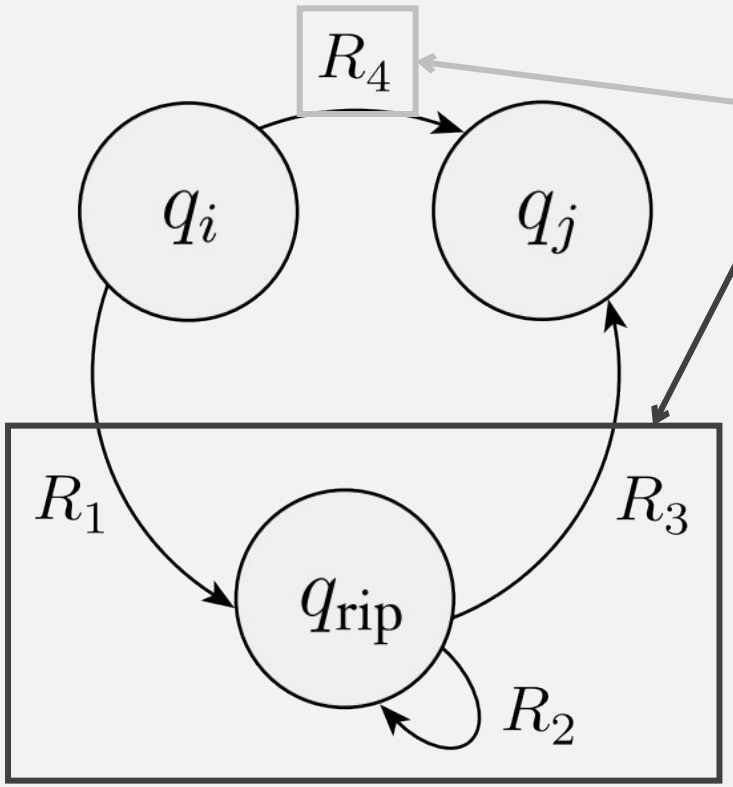


after

To convert GNFA -> regular expression:

1. “rip out” a state,
2. “repair” so machine is equivalent
3. repeat until only 2 states remain

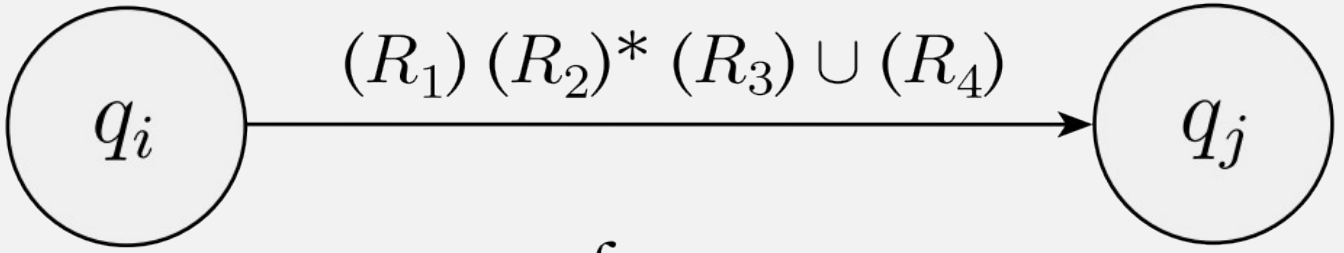
GNFA→RegExpr: “Rip/Repair” step



before

Before: two paths from q_i to q_j:

1. Not through q_{rip}
2. Through q_{rip}

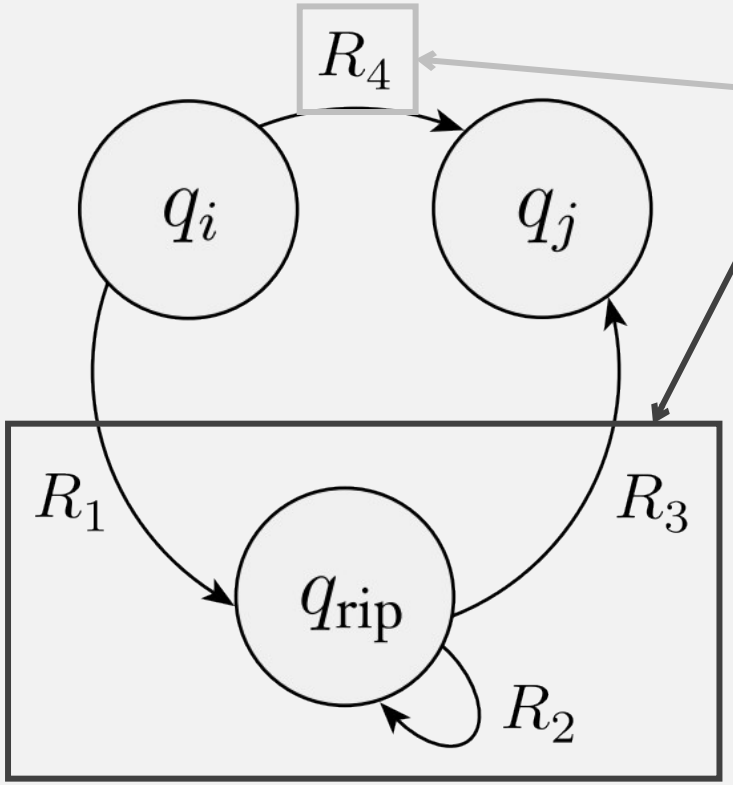


after

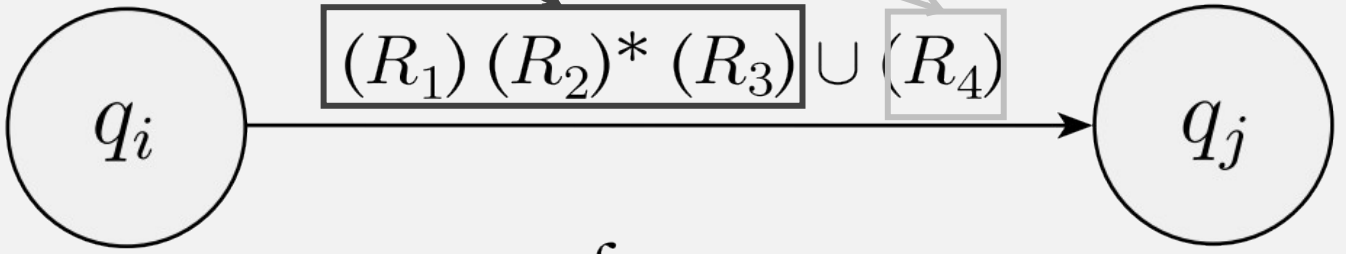
GNFA→RegExpr: “Rip/Repair” step

After: UNION of two “paths” from q_i to q_j

1. Not through q_{rip}
2. Through q_{rip}

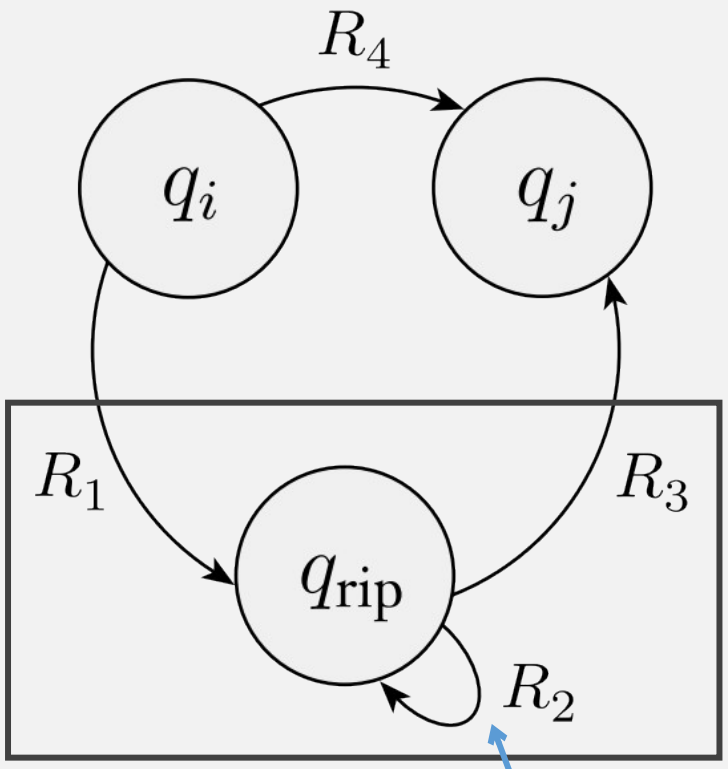


before

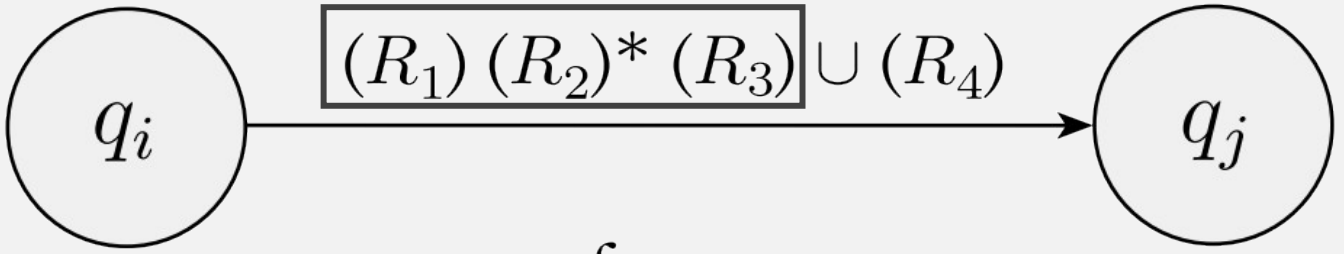


after

GNFA→RegExpr: “Rip/Repair” step



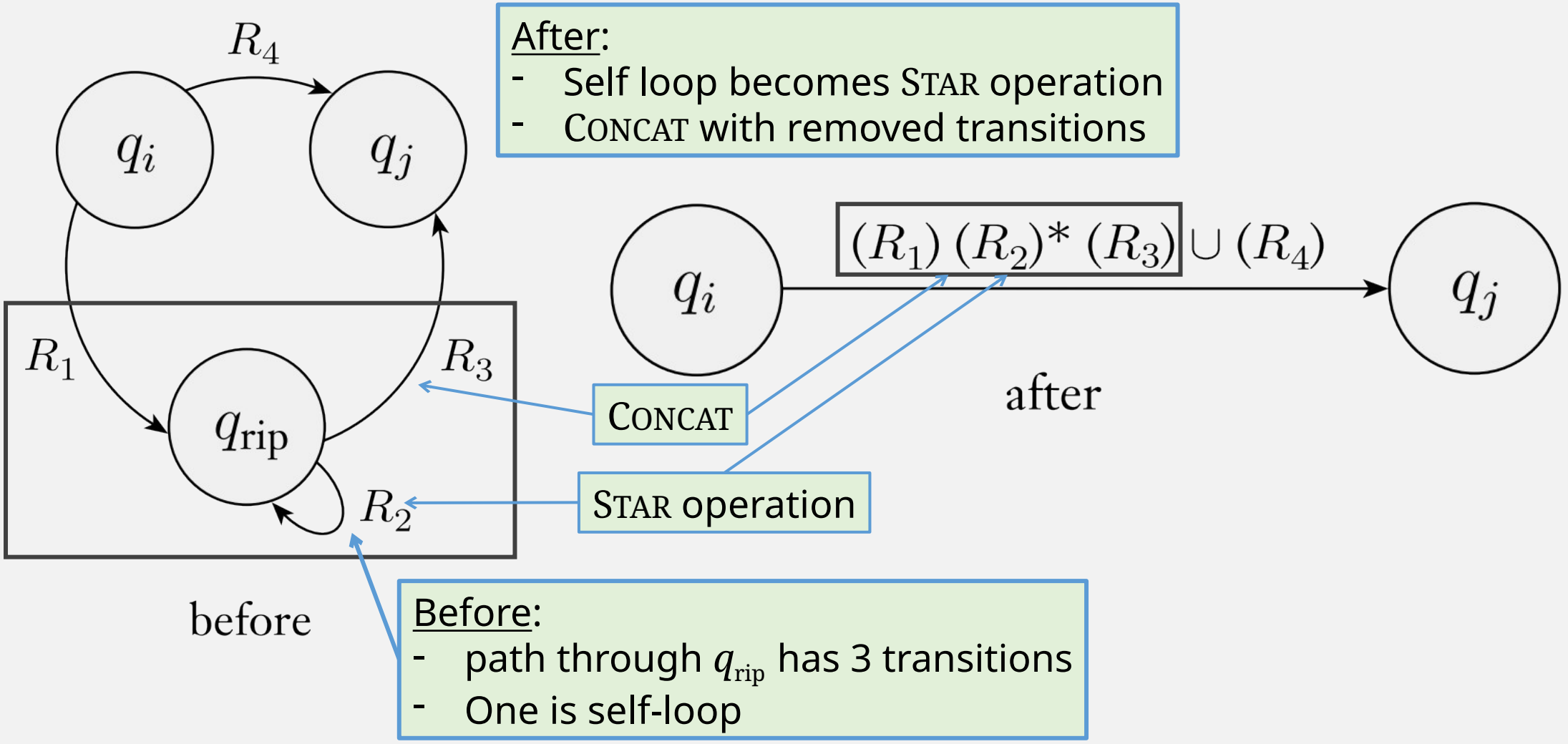
before



after

Before:
- path through q_{rip} has 3 transitions
- One is self-loop

GNFA→RegExpr: “Rip/Repair” step



Thm: A Lang is Regular iff Some Reg Expr Describes It

⇒ If a language is regular, then it's described by a regular expr

Need to convert DFA or NFA to Regular Expression …

- Use GNFA→RegExpr to convert GNFA → equiv regular expression!

???

This time, let's really prove equivalence!
(we previously "proved" it with an Examples Table)

⇐ If a language is described by a regular expr, then it's regular

- Convert regular expression → equiv NFA!

GNFA→RegExpr Correctness

- Correct = input and output are equivalent
- Equivalent = the language does not change (same strings)!

Statement to Prove:

???

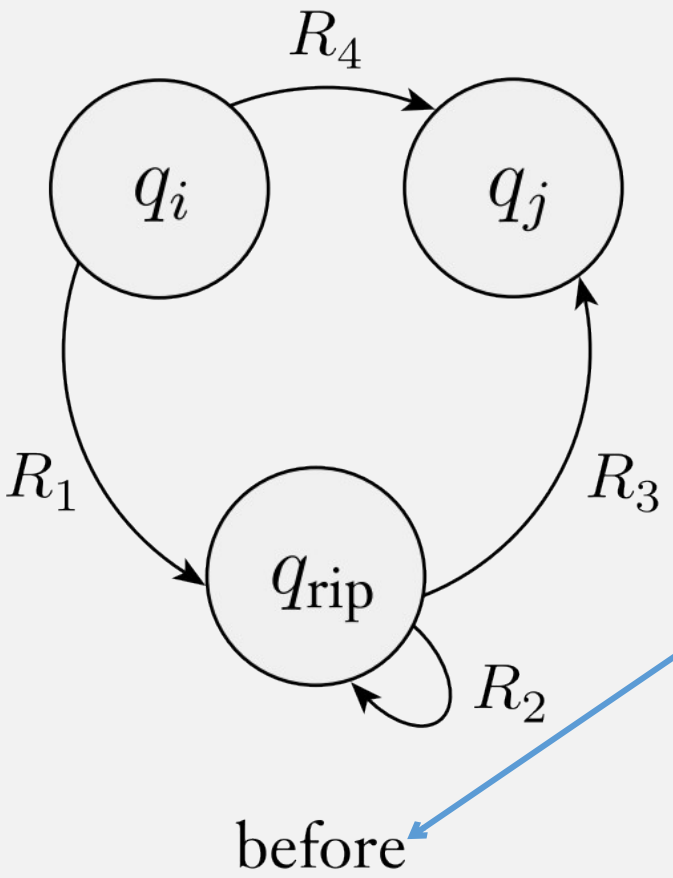
$$\text{LangOf}(G) = \text{LangOf}(R)$$

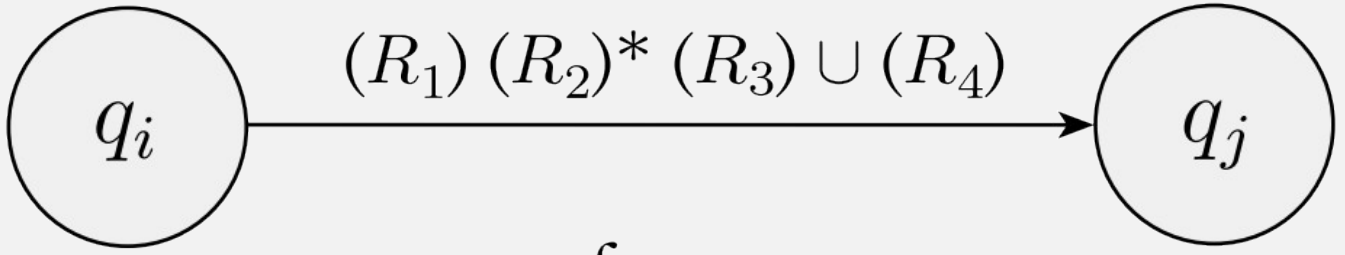
We are ready to <u>really prove</u> equivalence!
(we previously "proved" it with some examples)

- <u>where</u>:
  - G = a GNFA
  - R = a Regular Expression = GNFA→RegExpr(G)

- Key step: the rip/repair step

GNFA→RegExpr: Rip/Repair Correctness





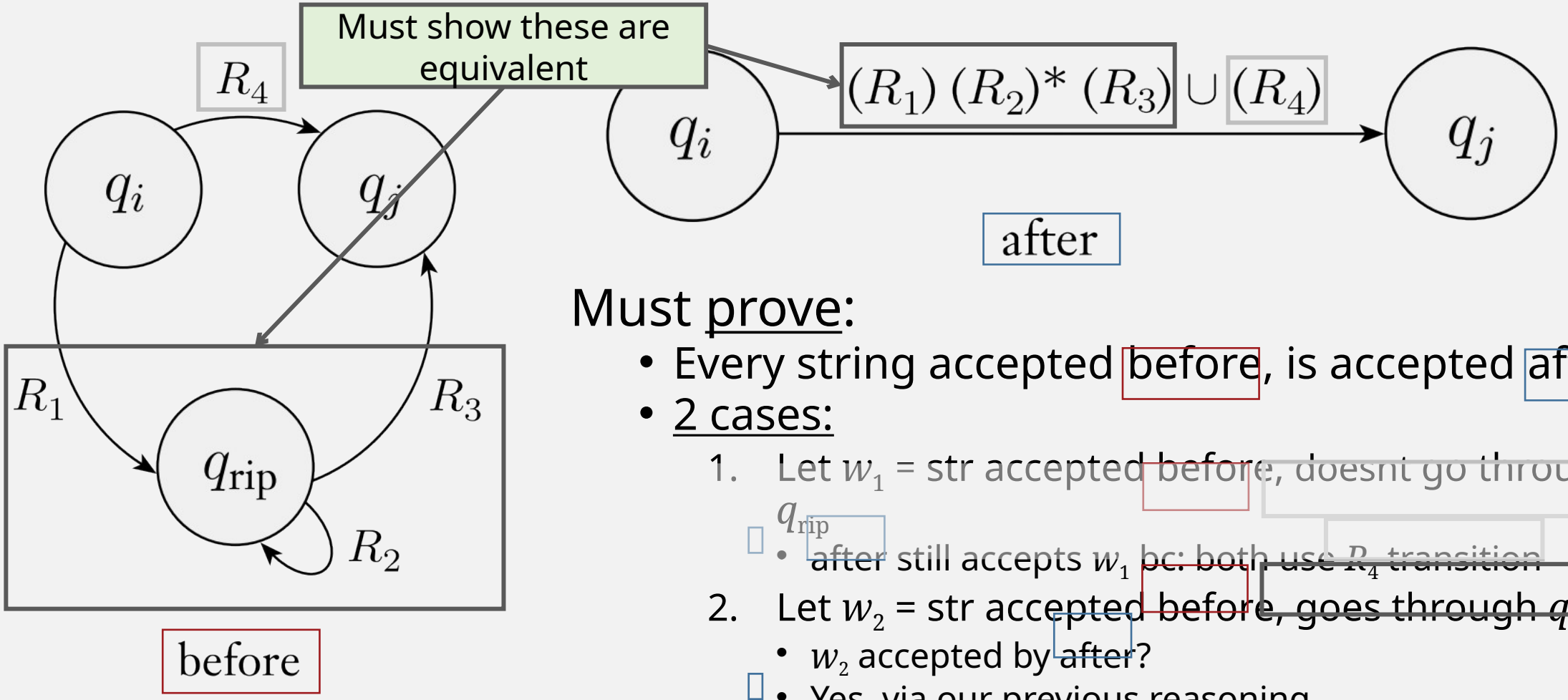
after

Must show these are equivalent

before

Equivalent =
same language =
accepts the same strings

GNFA→RegExpr: Rip/Repair Correctness



Must prove:

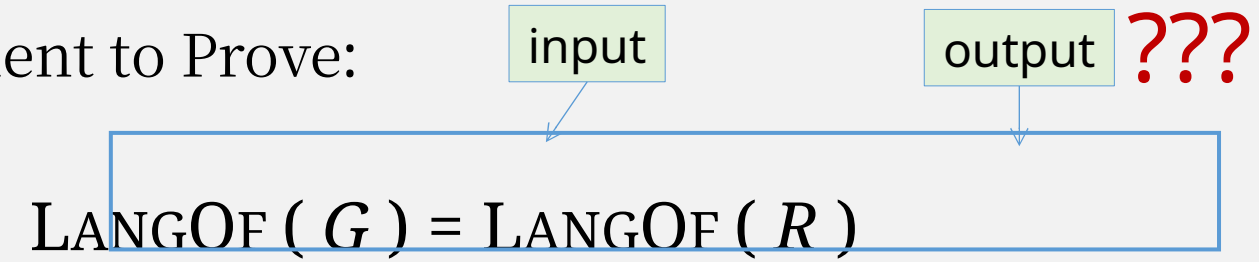
- Every string accepted before, is accepted after
- 2 cases:
  1. Let w_1 = str accepted before, doesn't go through q_{rip}
     - after still accepts w_1 bc: both use R_4 transition
  2. Let w_2 = str accepted before, goes through q_{rip}
     - w_2 accepted by after?
     - Yes, via our previous reasoning

(Similar to Equivalence Table!)

GNFA→RegExpr Equivalence

- Equivalent = the language does not change (i.e., same set of strings)!

Statement to Prove:



This time, let's really prove equivalence! (we previously "proved" it with some examples)

- where:
  - G = a GNFA
  - R = a Regular Expression = GNFA→RegExpr(G)

Language could be infinite set of strings!

(how can we show equivalence for a possibly infinite set of strings?)

Recursion!

Proof by Induction

Previously:

Recursive function

- Use it when: writing a function involving a recursive definition

Now:

Proof by induction (recursion) = “a recursive proof”

- Use it when: proving something involving a recursive definition

The recursive definition is (always) the key!

A valid recursive definition has:
- base case(s) and
- recursive case(s) (with “smaller” self-reference)

Proof by Induction

(A proof for each case of some recursive definition)

To Prove: ***Statement*** for recursively defined "thing" x:

1. Prove: *Statement* for base case of x
2. Prove: *Statement* for recursive case of x:
   - Assume: induction hypothesis (IH)

     I.e., *Statement* is true for some $x_{smaller}$ (This is just the recursive part from the recursive definition!)
     - E.g., if x is number, then "smaller" = lesser number

   - Prove: *Statement* for x, using IH (and known definitions, theorems …)
     - Typically: show that going from $x_{smaller}$ to larger x is true!

i.e., a normal proof

A valid recursive definition has:
- base case(s) and
- recursive case(s) (with "smaller" self-reference)

Natural Numbers Are Recursively Defined

A Natural Number is:

Base Case
- $\mathbf{0}$

Recursive Case
- Or $\boldsymbol{k} + \mathbf{1}$, where k is a Natural Number

Self-reference

Recursive definition is valid because self-reference is “smaller”

So, proving things about:
recursive Natural Numbers requires
recursive proof,
i.e., proof by induction!

A valid recursive definition has:
- base case(s) and
- recursive case(s) (with “smaller” self-reference)

Proof By Induction Example (Sipser Ch 0)

Prove true: $P_t = PM^t - Y\left(\frac{M^t - 1}{M - 1}\right)$

- P_t = loan balance after *t* months
- t = # months
- P = principal = original amount of loan
- M = interest (multiplier)
- Y = monthly payment

(Details of these variables not too important here)

Proof By Induction Example (Sipser Ch 0)

Prove true: $P_t = PM^t - Y\left(\frac{M^t - 1}{M - 1}\right)$

Proof: by induction on natural number t

A proof by induction follows the cases of the recursive definition (here, natural numbers) that the induction is "on"

A Natural Number is:

- 0 □
- Or $k + 1$, where k is a natural number

Base Case, $t = 0$:

Goal statement to prove (for $t = 0$):

$$P_0 = PM^0 - Y\left(\frac{M^0 - 1}{M - 1}\right) = P$$

Plug in $t = 0$

Simplify (Justification)

$P_0 = P$ is a true statement !
(amount owed at start = loan amount)

Proof By Induction Example (Sipser Ch 0)

A proof by induction follows cases of recursive definition (here, natural numbers) that the induction is "on"

A Natural Number is:
- 0
- $k+1$, for some nat num k

Prove true: $P_t = PM^t - Y\left(\frac{M^t - 1}{M - 1}\right)$

Inductive Case: $t = k + 1$, for some natural num k

- Inductive Hypothesis (IH), assume statement is true for some t = (s…

IH plugs in "smaller" k

$$P_k = PM^k - Y\left(\frac{M^k - 1}{M - 1}\right)$$

Write $t = k+1$ case in terms of "smaller" k

Plug in IH for P_k

Go…t to prove, for $t = k+1$:

$$P_{k+1} = PM^{k+1} - Y\left(\frac{M^{k+1} - 1}{M - 1}\right)$$

Simplify, to get to goal statement

- Proof of G…

$$P_{k+1} = P_k M - Y$$

Definition of Loan:
amt owed in month $k+1$ =
amt owed in month k * interest M – amt paid Y

In-class Exercise: Proof By Induction

A proof by induction follows cases of recursive definition (here, natural numbers) that the induction is "on"

A Natural Number is:

- 0
- $k + 1$, for some nat num k

Prove: ($z \neq 1$)

$$\sum_{i=0}^{m} z^i = \frac{1 - z^{m+1}}{1 - z}$$

Use Proof by Induction.

Make sure to clearly state:

what the induction is "on"

i.e., which recursively defined value (and its type) will the proof focus on

And the Goal statement to prove (for each case)

Proof by Induction: CS 420 Example

Statement to prove: $\text{LANGOF}(G) = \text{LANGOF}(R = \texttt{GNFA→RegExpr}(G))$

- Where:
  - G = a GNFA
  - R = a Regular Expression $\texttt{GNFA→RegExpr}(G)$

- i.e., GNFA→RegExpr must not change the language!

This time, let's really prove equivalence! (we previously "proved" it with some examples)

Proof by Induction: CS 420 Example

Statement to prove: $\text{LANGOF}(G) = \text{LANGOF}(\texttt{GNFA→RegExpr}(G))$

Recursively defined "thing"

Proof: by Induction on # of states in G

1. Prove *Statement* is true for base case G has 2 states

Why is this an ok base case (instead of zero)?

(Modified) Recursive definition:

A "NatNumber > 1" is:

- 2
- Or $k + 1$, where k is a "NatNumber > 1"

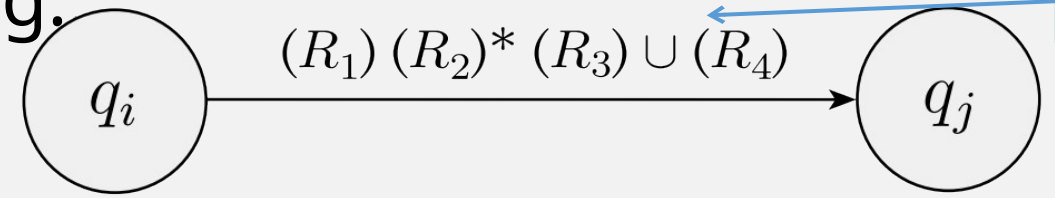
Last Time

GNFA→RegExpr (recursive) function

On GNFA input G:

Base Case

- If G has 2 states, return the regular expression (from the transition), e.g.:



Equivalent regular expression

GNFA

Proof by Induction: CS 420 Example

Statement to prove:

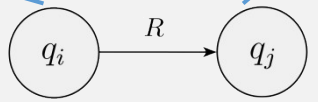
$$\text{LANGOF}(G) = \text{LANGOF}(\texttt{GNFA→RegExpr}(G))$$

Plug in

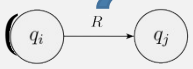
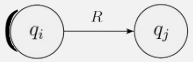
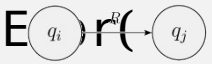
Proof: by Induction on # of states in G

1. Prove *Statement* is true for base case

G has 2 states



Statements

1. $\text{LANGOF}($  $) = \text{LANGOF}(R)$
2. $\texttt{GNFA→RegExpr}($  $) =$ $\text{LANGOF}($  $) = \text{LANGOF}($  $))$

Plug in R

Goal

Justifications

1. Definition of GNFA
2. Definition of `GNFA→RegExpr` (base case)
3. From (1) and (2)

Don't forget the Statements / Justifications !

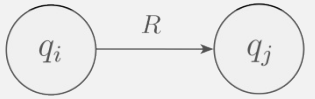
Proof by Induction: CS 420 Example

Statement to prove:

$$\text{LANGOF}(G) = \text{LANGOF}(\texttt{GNFA→RegExpr}(G))$$

Proof: by Induction on # of states in G

1. Prove *Statement* is true for base case (G has 2 states)



2. Prove *Statement* is true for recursive case (G has > 2 states)

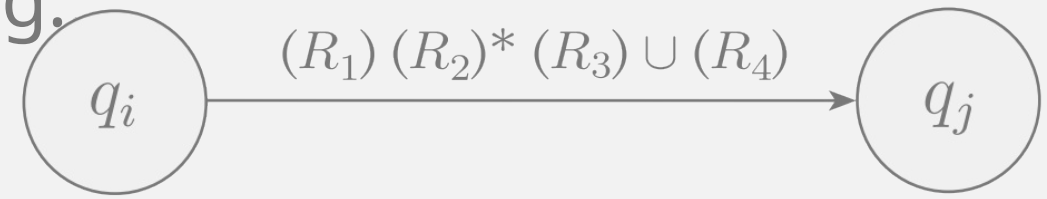
Last Time

GNFA→RegExpr (recursive) function

On GNFA input G:

Base Case

- If G has 2 states, return the regular expression (from the transition), e.g.:



- Else:

Recursive Case

- "Rip out" one state
- "Repair" the machine to get an equivalent GNFA G'
- Recursively call GNFA→RegExpr(G')

Recursive call (with a "smaller" G')

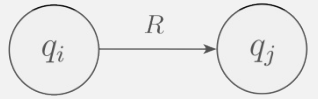
Proof by Induction: CS 420 Example

Statement to prove:

$$\text{LANGOF}(G) = \text{LANGOF}(\text{GNFA→RegExpr}(G))$$

Proof: by Induction on # of states in G

1. Prove *Statement* is true for base case (G has 2 states)

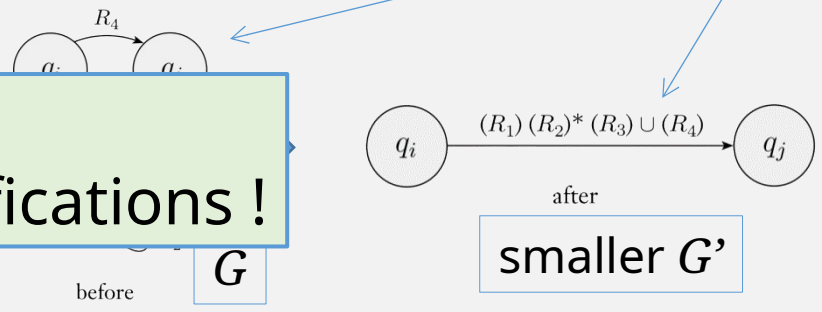


2. Prove *Statement* is true for recursive case (G has > 2 states)
   - Assume the induction hypothesis (IH):
     - *Statement* is true for smaller G'
   - Use it to prove *Statement* is true for G > 2 states
     - Need to show: going from G to smaller G' is true!

IH Assumption

$$\text{LANGOF}(G') = \text{LANGOF}(\text{GNFA→RegExpr}(G'))$$

(G' has less states than G)



smaller G'

Don't forget the Statements / Justifications !

Show that "rip/repair" step converts G to smaller, equivalent G'

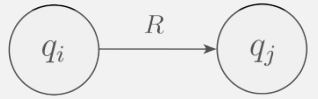
Proof by Induction: CS 420 Example

Statement to prove:

$$\text{LANGOF}(G) = \text{LANGOF}(\texttt{GNFA→RegExpr}(G))$$

Proof: by Induction on # of states in G

1. Prove *Statement* is true for base case — G has 2 states



2. Prove *Statement* is true for recursive case — G has > 2 states
   - Assume the
     - *Statement*
   - Use it to prove 2 states
     - Need to s ' is true!

Known "facts" available to use:
- IH
- Equiv of Rip/Repair step
- Def of GNFA->RegExpr

IH Assumption

$$\text{LANGOF}(G') = \text{LANGOF}(\texttt{GNFA→RegExpr}(G'))$$
(Where G' has less states than G)

Statements
1. $\text{LANGOF}(G') = \text{LANGOF}(\texttt{GNFA→RegExpr}(G'))$
2. $\text{LANGOF}(G) = \text{LANGOF}(G')$
3. $\texttt{GNFA→RegExpr}(G) = \texttt{GNFA→RegExpr}(G')$ — Plug in
4. $\text{LANGOF}(G) = \text{LANGOF}(\texttt{GNFA→RegExpr}(G))$ — Goal

Justifications
1. IH
2. Equivalence of Rip/Repair step (prev)
3. Def of GNFA→RegExpr (recursive call)
4. From (1), (2), and (3)

Thm: A Lang is Regular iff Some Reg Expr Describes It

⇒ If a language is regular, then it's described by a regular expr

- Use GNFA→RegExpr to convert GNFA → equiv regular expression!

⇐ If a language is described by a regular expr, then it's regular

- Convert regular expression → equiv NFA!

■

Now: we can use regular expressions to represent regular langs!

So we also have another way to prove things about regular languages!

So a regular language has these equivalent representations:

- DFA
- NFA
- Regular Expression

So Far: How to Prove A Language Is Regular?

Key step, either:

- Construct DFA
- Construct NFA
- Create Regular Expression

Slightly different because of recursive definition

R is a ***regular expression*** if R is

1. a for some a in the alphabet Σ,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is a regular expression.

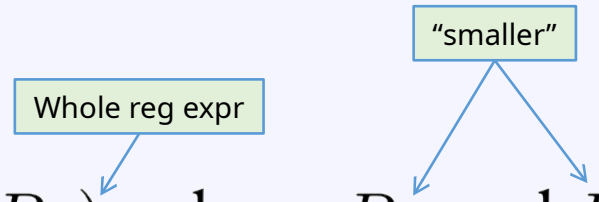
Proof by Induction

To Prove: a ***Statement*** about a recursively defined "thing" x:

1. Prove: *Statement* for base case of x
2. Prove: *Statement* for recursive case of x:
   - Assume: induction hypothesis (IH)

     I.e., *Statement* is true for some x_{smaller}
     - E.g., if x is number, then "smaller" = lesser number
     - E.g., if x is regular expression, then "smaller" = …
   - Prove: *Statement* for x, using IH (and known definitions, theorems …)
     - Usually, must show that going from x_{smaller} to larger x is true!

1. a for some a in the alphabet Σ,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is a regular expression.



Thm: Reverse is Closed for Regular Langs

Example string: **abc** $^{\mathcal{R}}$ = **cba**

For any string $w = w_1 w_2 \cdots w_n$, the ***reverse*** of w, written $w^{\mathcal{R}}$, is the string w in reverse order, $w_n \cdots w_2 w_1$.

For any language A, let $A^{\mathcal{R}} = \{w^{\mathcal{R}} \mid w \in A\}$

Example language:

{ **a**, **ab**, **abc** }$^{\mathcal{R}}$ = { **a**, **ba**, **cba** }

Theorem: if A is regular, so is $A^{\mathcal{R}}$

IF-THEN Statement to prove:
If A is a regular language described by regular expression R, then $A^{\mathcal{R}}$ is regular

Proof: by induction on R, the regular expression of A

Thm: Reverse is Closed for Regular Langs

if A is regular, so is $A^{\mathcal{R}}$

Proof: by Induction on R, the regular expression of A (6 cases)

Base cases

1. a for some a in the alphabet Σ,

same reg. expr. represents $A^{\mathcal{R}}$ so it is regular

2. ε,

same reg. expr. represents $A^{\mathcal{R}}$ so it is regular

3. $\emptyset$,

same reg. expr. represents $A^{\mathcal{R}}$ so it is regular

Inductive cases

4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is a regular expression.

"smaller"

IH1: if A_1 is a regular language, described by reg expr R_1, then $A_1{}^R$ is regular

IH2: if A_2 is a regular language, described by reg expr R_2, then $A_2{}^R$ is regular

Need to prove: if A is a regular language, described by reg expr $R_1 \cup R_2$, then A^R is regular

Thm: Reverse is Closed for Regular Langs

if A is regular, so is $A^{\mathcal{R}}$

Proof: by Induction on R, the regular expression of A (Case # 4)

Statements	Justifications
1. Language A is regular, with reg expr R_1 U R_2	1. Assumption of IF in IF-THEN
2. R_1 and R_2 are regular expressions	2. Def of Regular Expression
3. R_1 and R_2 describe regular langs A_1 and A_2	3. Reg Expr ⇔ Reg Lang (Prev Thm)
4. If A_1 is a regular language, then A_1^R is regular	4. IH1
5. If A_2 is a regular language, then A_2^R is regular	5. IH2
6. A_1^R and A_2^R are regular	6. By (3), (4), and (5)
7. A_1^R U A_2^R is regular	7. Union Closed for Reg Langs
8. A_1^R U $A_2^R = (A_1$ U $A_2)^R$	8. Reverse and Union Ops Commute
9. $A = A_1$ U A_2	9. By (1), (2), and (3)
10. A^R is regular	10. By (7), (8), (9)

Need to prove: if A is a regular language, described by reg expr R_1 U R_2, then A^R is regular by Stmts #1 and #10

Thm: Reverse is Closed for Regular Langs

if A is regular, so is $A^{\mathcal{R}}$

Proof: by Induction on regular expression of A: (6 cases)

Base cases

Inductive cases

1. a for some a in the alphabet Σ,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is a regular expression.

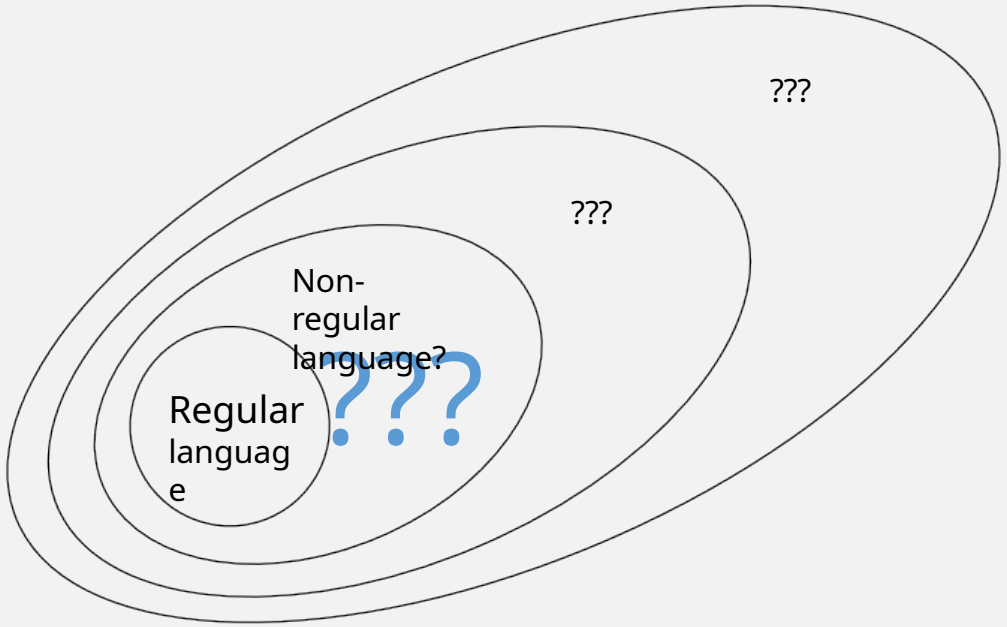
Remaining cases will use similar reasoning

Next Time

Non-Regular Languages?

- Are there languages that are not regular languages?

- How can we prove that a language is not a regular language?



A ***language*** is a set of strings.

So Far: Regular or Not?

- Many ways to prove a language is regular:
  - Construct a DFA recognizing it
  - Construct an NFA recognizing it
  - Create a regular expression describing the language

M ***recognizes language*** A
if $A = \{w | \ M \text{ accepts } w\}$

- Bc we proved: Regular Expression ⇔ NFA ⇔ DFA ⇔ Regular Language

- But *not* all languages are regular!
  - E.g., programming language syntaxes are not regular
    - language of all Python programs, or all HTML/XML pages, are not regular
  - That means:
    - There is *no* DFA or NFA that: accepts valid Python programs (and rejects invalid ones)
    - And, there is *no* regular expression that: describes all valid Python or HTML programs (a common mistake)!

Someone Who Did Not P

RegEx match open tags except XHTML self-cor

Asked 10 years, 10 months ago Active 1 month ago Viewed 2.9m times

1553

I need to match all of these opening tags:

```
<p>
<a href="foo">
```

6572

But not these:

4414

You can't parse [X]HTML with regex. Because HTML can't be parsed
Regex is not a tool that can be used to correctly parse HTML. As I h
HTML-and-regex questions here so many times before, the use of re
allow you to consume HTML. Regular expressions are a tool that is
sophisticated to understand the constructs employed by HTML. HTM
regular language and hence cannot be parsed by regular expressior
queries are not equipped to break down HTML into its meaningful pa
times but it is not getting to me. Even enhanced irregular regular exp
used by Perl are not up to the task of parsing HTML. You will never

Trying to use regular expressions to describe the non-regular HTML language

Someone who paid attention in 420 …

HTML is a language of sufficient complexity that it cannot be parsed by regular expressions. Even Jon Skeet cannot parse HTML using regular expressions. Every time you attempt to parse HTML with regular expressions, the unholy child weeps the blood of virgins, and Russian hackers pwn your webapp. Parsing HTML with regex summons tainted souls into the realm of the living. HTML and regex go together like love, marriage, and ritual infanticide. The cannot hold it is too late. The force of regex and HTML together in the same conceptual space will destroy your mind like mayonnaise on a hot stove. If you parse HTML with regex you are giving in to Them and their blasphemous ways which doom us all to inhuman toil for the One whose name cannot be expressed in the Basic Multilingual Plane, he comes. HTML-plus-regexp will liquify the nerves of the sentient whilst you observe, your psyche withering in the onslaught of horror. Regex-based HTML parsers are the cancer that is killing StackOverflow *it is too late it is too late we cannot be saved* the trangession of a child ensures regex will consume all living tissue (except for HTML which it cannot, as previously prophesied) *dear lord help us how can anyone survive this scourge* using regex to parse HTML has doomed humanity to an eternity of dread torture and security holes *using regex as a tool to process* HTML establishes a breach *between this world* and the dread realm of corrupt entities (like SGML entities, but *more corrupt) a mere glimpse* of the world of rege**x parsers for HTML will ins**tantly transport a *programmer's consciousness* into a world of ceaseless screaming, he comes, the pestilent slithy regex-infection will **devour your H**TML parser, application and existence for all time like Visual Basic only worse *he comes he comes do not fight he comes, h*is unholy radiance *destroying all enlightenment, HTML tags leaking from your eyes like liquid* pain, the song of regular expression parsing will ex*tinguish the voices of mortal man from the sphere I can see it can you see it it is beautiful* the f`inal snuf`fing *of the lies of Man ALL IS LOST* **ALL IS L**OST the *pony he come*s he comes he comes **the Fi**chor permeates all MY FACE *MY FACE oh god* **no NOOOOOO N**Θ stop *the an*gles `are` **not real ZALGO IS** TONY **THE PONY HE COMES**

ummm … this is getting a little weird

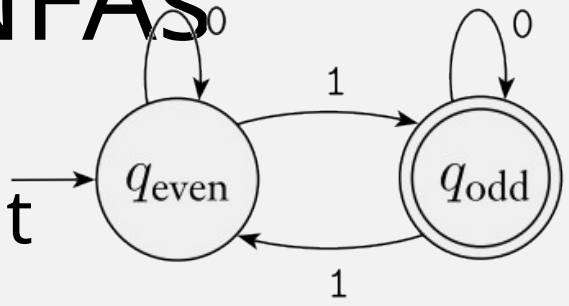
very weird …

?????

Have you tried using an XML parser instead?

hmm … what's this?

Flashback: Designing DFAs or NFAs



- Each state "remembers" information about input
  - E.g., q_{even} = "seen even # of 1s"

    q_{odd} = "seen odd # of 1s"
  - But <u>finite</u> states = <u>finite</u> amount of info storage (and must decide in advance)

- So <u>DFAs can't remember</u> an <u>arbitrary count</u>!
  - would require infinite states

A Non-Regular Language

An arbitrary count

$L = \{ \texttt{0}^n\texttt{1}^n \mid n \geq 0 \}$

- A DFA recognizing L would require infinite states! (impossible)
  - States representing zero `0`s seen, one `0` seen, two `0`s, …
- This language is the same as many PLs, e.g., HTML!
  - To better see this replace:
    - "`0`" with "`<tag>`" or "`(`"
    - "`1`" with "`</tag>`" or "`)`"

So, how can we *prove* non-regularness?

- The Problem: remembering nestedness
  - Need to count arbitrary nesting depths
    - E.g., `if { if { if { … } } }`
  - Thus: most programming language syntax is not regular!

Prove: Demons do not exist

???

Proving something not true is different (and usually harder) than proving it true



It's sometimes possible, but often needs new proof techniques!

We know how to: prove a language is regular
Can we: prove a language is not regular?

Quantified Logical Statements

- "Exists" (Existential)
  - "Easier" to prove TRUE
    - Just need one example!

$\exists x P(x)$ is true when $P(x)$ is true for at least one value of x.

"There exists a natural number n such that, $n \cdot n = 25$"

- "For all" (Universal)
  - "Harder" to prove TRUE
    - Need to prove true for all examples

$\forall x P(x)$ is true when $P(x)$ is true for all values of x.

"For all natural numbers n, $2 \cdot n = n + n$"

Quantified Logical Statements (this course)

- "Exists" (Existential)
  - "Easier" to prove TRUE
    - Just need one example!
  - "Harder" to prove FALSE
    - Need to prove false for all examples
- "For all" (Universal)
  - "Harder" to prove TRUE
    - Need to prove true for all examples
  - "Easier" to prove FALSE
    - Just need one (counter)example!

Language L is regular

There exists one DFA that recognizes L

Language L is not regular?

There are no possible DFAs that recognizes L

For all regular languages L, L^* is a regular language

For all ...

For all strings in a regular language ...

Key is finding such a statement about regular languages!

A Fact (Lemma) About Regular Languages

True for all regular languages!

∀A

If A is a regular language, then there is a number p (the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

This is an "If X then Y" statement

Remember: To *use* an "If X then Y" statement,
1. *prove* X is true,
2. *conclude* that Y is true

A Fact (Lemma) About Regular Languages

Pumping lemma If A is a regular language, then there is a number p (the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

This is an "If X then Y" statement

Remember: To *use* an "If X then Y" statement,
1. *prove* X is true,
2. *conclude* that Y is true

Flashback:

The Modus Ponens Inference Rule

If we know these statements are true …

- If P then Q

- P

Then we also know this statement is true …

- Q

A Lemma About Regular Languages

Pumping lemma If A is a regular language, then there is a number p (the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

… then we can *conclude* …

Uh … whatever this says …

To *use* The **Pumping lemma** for a language A …

… first *prove* that A is a regular language …

Q: Can we *use* The **Pumping lemma** to prove that a language is regular?

NO (but we already know many other ways to do that!)

(but maybe it *can* be used to prove that a language is not regular!)

Equivalence of Conditional Statements

- Yes or No? "If X then Y" is equivalent to:
  - "If Y then X" (converse)
    - No!
  - "If not X then not Y" (inverse)
    - No!
  - "If not Y then not X" (contrapositive)
    - Yes!

See Previously

If-then statement

… then the language is not regular!

Pumping lemma If A is a regular langua… pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

Equivalent (contrapositive):
If any of these are not true …

Contrapositive:
"If X then Y" is equivalent to "If **not** Y then **not** X"

Logical Inference Rules

Modus Ponens

Premises (known facts)

- If P then Q
- P is true

Conclusion (new fact)

- Q is true

Modus Tollens
(contrapositive)

Premises (known facts)

- If P then Q
- Q is *not* true

Step 1: find a fact that is true for all regular languages …

Step 2: where the fact can be proven not true!

Conclusion (new fact)

- P is *not* true

How to: prove a language is not regular?

Fact About Regular Languages: Details

Pumping lemma If A is a regular language, then there is a number p (the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

Conditions are on strings in the language with length $\geq$ ***p***

Any regular language satisfies these three conditions!

NOTE:
- Lemma doesn't give an exact p!
- Only that there is *some* string length p …

The exact value of p differs for every regular language

The Pumping Lemma: Finite Languages

Conclusion: pumping lemma is only interesting for infinite langs! (which contain strings with repeating parts)

Lemma doesn't say what p is! Just that "there is a p …"

Pumping le there is a number p (the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

So finite langs (specifically, all strings in the language "of length at least p") must satisfy these conditions (whatever they are)

Possible p for finite langs?

How about:
p = LENGTH(longest string) + 1

strings in the language with length ≥ p?

None !

Therefore, all strings with length ≥ p satisfy the pumping lemma conditions! ☺

Example: a finite language {"ab", "cd"}

- All finite languages are regular!
- (can easily construct DFA/NFA/Regular Expression recognizing them)

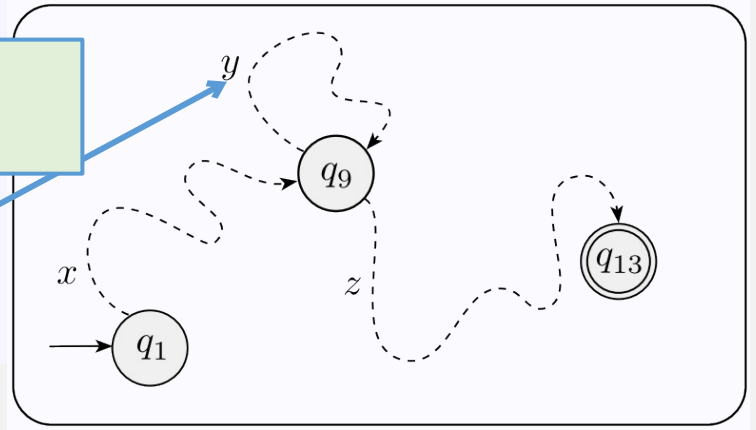
Langs With Strings With Repeatable Parts

Pumping lemma If A is a regular language, then there is a number p (the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

"pumped" string still in language

repeatable ("pumpable") part (= repeatable state in DFA!)



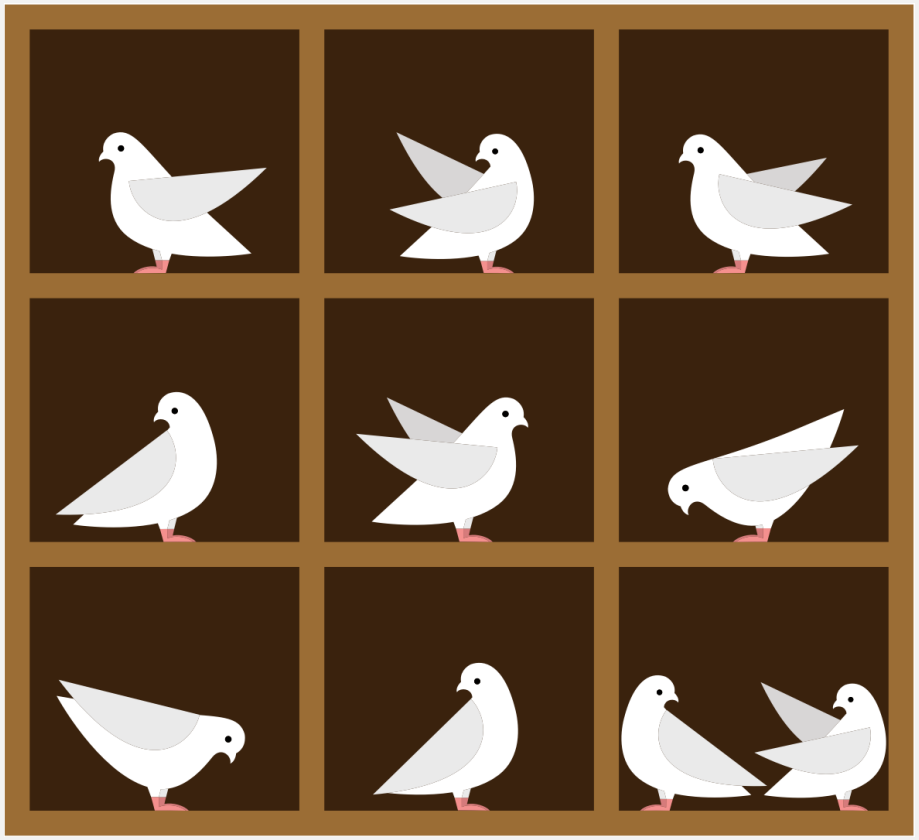
Strings that have a repeatable part can be split into 3 parts:

- x = part before any repeating
- y = repeated (or "pumpable") part
- z = part after any repeating

DFAs have finite states,
so for "long enough" (i.e., length ≥ p) inputs,
some state must repeat!

e.g., "long enough length" = p = # states +1 (The Pigeonhole Principle)

The Pigeonhole Principle



If # birds > # holes, then there must be > 1 bird in some hole

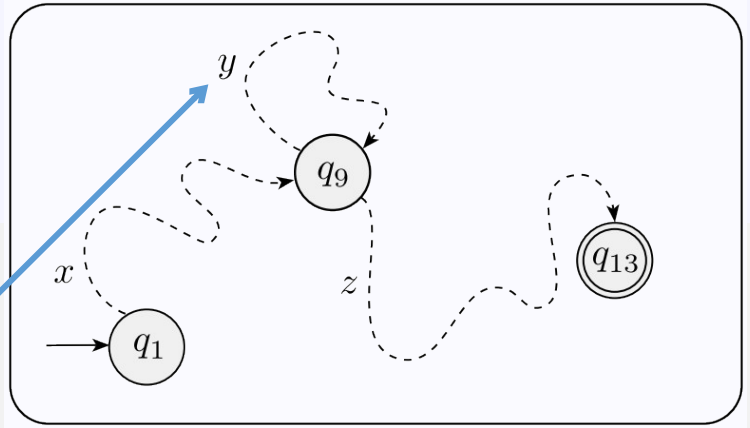
The Pumping Lemma, a Closer Look

Pumping lemma If A is a regular language, then there is a number p (the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

So a substring that *can* repeat once, *can also* be repeated any number of times

This is the only way for regular languages to have repeating patterns (Kleene star)



In essence, the Pumping lemma is a theorem about repeating patterns in regular languages

"long enough length" = p = # states +1
(some state must repeat)

In-class exercise: Infinite Languages

Split the string “010” into three parts xyz, e.g.
x = ??, y = ??, z = ??
so that repeating y part any number of times results in a new string still in A

Now do “0110”:
x = ??, y = ??, z = ??

Example: *infinite* language A = {“00”, “010” , “0110” , “01110”, …}

The Pumping Lemma: Infinite Languages

Pumping lemma If A is a regular language, then there is a number p (the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

"pumpable" part of string

Note: "pumpable" part cannot be empty

E.g., "010" $\in A$, so pumping lemma says it's splittable into three parts xyz, e.g.
$x = 0$, $y = 1$, $z = 0$

Example: *infinite* language A = {"00", "010" , "0110" , "01110" ,...}

- It's regular bc it has regular expression 01^*0

Pumping lemma summary:
"All infinite regular languages must have a star in its regular expression"!

... and "pumping" (repeating) middle y part creates a string that is still in the language
- repeat once (i = 1): "0**1**0",
- repeat twice (i = 2): "0**11**0",
- repeat three times (i = 3): "0**111**0"

Summary: The Pumping Lemma …

- … states properties that are true for all regular languages
- … specifically, properties about "long enough" strings in reg. langs
- In general, it describes repeating patterns in reg. langs

IMPORTANT:

- The Pumping lemma cannot prove that a language is regular!
- But … we can use it to prove that a language is not regular

Pumping lemma summary:
"All infinite regular languages must have a star in its regular expression"!

… by showing that the repeating pattern is not expressible with a star regular expression!

If-then statement

… then the language is not reguiar

Pumping lemma If A is a regular langua… pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

Equivalent (contrapositive):
If any of these are not true …

Contrapositive:
"If X then Y" is equivalent to "If **not** Y then **not** X"

Kinds of Mathematical Proof

- Deductive Proof
  - Logically infer conclusion from known definitions and assumptions
- Proof by induction
  - Use to prove properties of recursive definitions or functions
- Proof by contradiction 
  - Proving the contrapositive

How To Do Proof By Contradiction

3 easy steps:

1. Assume: the opposite of the statement to prove

2. Show: the assumption leads to a contradiction

3. Conclude: the original statement must be true

Pumping Lemma: Non-Regularity Example

This repetition pattern cannot be expressed with a star regular expression?

Let B be the language $\{0^n 1^n | n \geq 0\}$. We use the pumping lemma to prove that B is not regular. The proof is by contradiction.

Pumping lemma summary:
"All infinite regular languages must have a star in its regular expression"!

... by showing that the repeating pattern is not expressible with a star regular expression!

Want to prove: 0^n1^n **is not** a regular language

Pumping lemma If A is a regular language, then there is a number p (the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

Reminder: Pumping lemma says: all strings $0^n1^n \geq$ length p are splittable into xyz where y is

So find string $\geq$ length p that is not splittable into xyz where y is pumpable

Proof (by contradiction): Now we must find a contradiction …

- Assume: 0^n1^n **is** a regular language
  - So it must satisfy the pumping lemma
  - I.e., all strings $\geq$ length p are pumpable
- Counterexample = 0^p1^p

We must show that there is no possible way to split this string to satisfy the conditions of the pumping lemma!

Want to prove: 0^n1^n **is not** a regular language

Possible Split: y = all 0s

Proof (by contradiction):

- Assume: 0^n1^n **is** a regular language
  - So it must satisfy the pumping lemma
  - I.e., all strings $\geq$ length p are pumpable
- Counterexample = 0^p1^p
- Choose xyz split so y contains:
  - ~~all 0s~~

Contradiction?

Not yet!

p 0s p 1s

00 … 011 … 1

x y z

- Pumping y: produces a string with more 0s than 1s
  - … not in the language 0^n1^n !
  - So 0^p1^p is not pumpable? (according to pumping lemma)
  - So 0^n1^n is a not regular language? (contrapositive)
  - This is a contradiction of the assumption?

… then not true

Pumping lemma If A is a regular language, then there is a number p (the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

Contrapositive: If not true …

Reminder: Pumping lemma says: all strings $0^n1^n \geq$ length p are splittable into xyz where y is

So find string $\geq$ length p that is not splittable into xyz where y is pumpable

BUT … pumping lemma requires only one pumpable splitting

So the proof is not done!

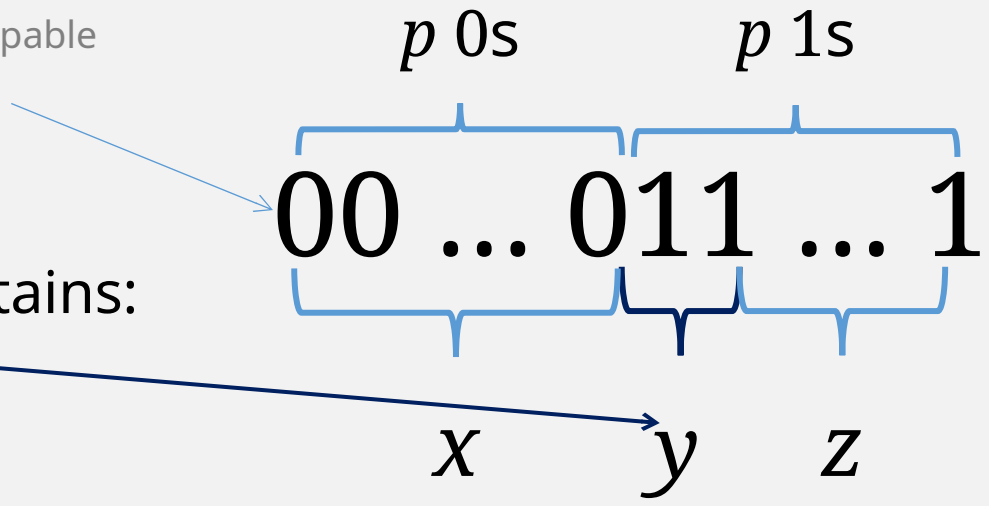
Is there another way to split into xyz ?

Want to prove: 0^n1^n is **not** a regular language

Possible Split: y = all 1s

Proof (by contradiction):

- Assume: 0^n1^n **is** a regular language
  - So it must satisfy the pumping lemma
  - I.e., all strings $\geq$ length p are pumpable
- Counterexample = 0^p1^p



- Choose xyz split so y contains:
  - ~~all 1s~~

Is there another way to split into xyz ?

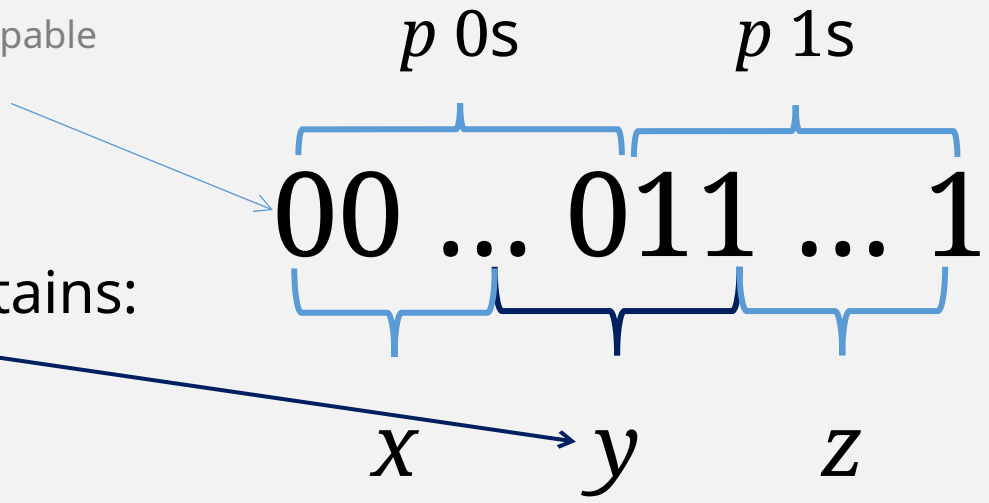
- Is this string pumpable (repeating y produces string still in 0^n1^n)?
  - No!
  - ~~By the same reasoning as in the previous slide~~

Want to prove: 0^n1^n is **not** a regular language

Possible Split: y = 0s and 1s

Proof (by contradiction):

- Assume: 0^n1^n **is** a regular language
  - So it must satisfy the pumping lemma
  - I.e., all strings $\geq$ length p are pumpable
- Counterexample = 0^p1^p
- Choose xyz split so y contains:
  - ~~both 0s and 1s~~



Did we examine every possible splitting?

Yes! QED

- Is this string pumpable (repeating y produces string still in 0^n1^n)?
  - No!
  - Pumped string will have equal 0s and 1s …
  - But they will be in the wrong order: so there is still a contradiction!

But maybe we did't have to …

The Pumping Lemma: Condition 3

Pumping lemma If A is a regular language, then there is a number p (the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

The repeating part y ...
must be in the first p characters!

p 0s

00 ... 011 ... 1

y must be in here!

The Pumping Lemma: Pumping Down

Pumping lemma If A is a regular language, then there is a number p (the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

Repeating part y must be non-empty
...
but can be repeated zero times!

Example: $L = \{0^i1^j \mid i > j\}$

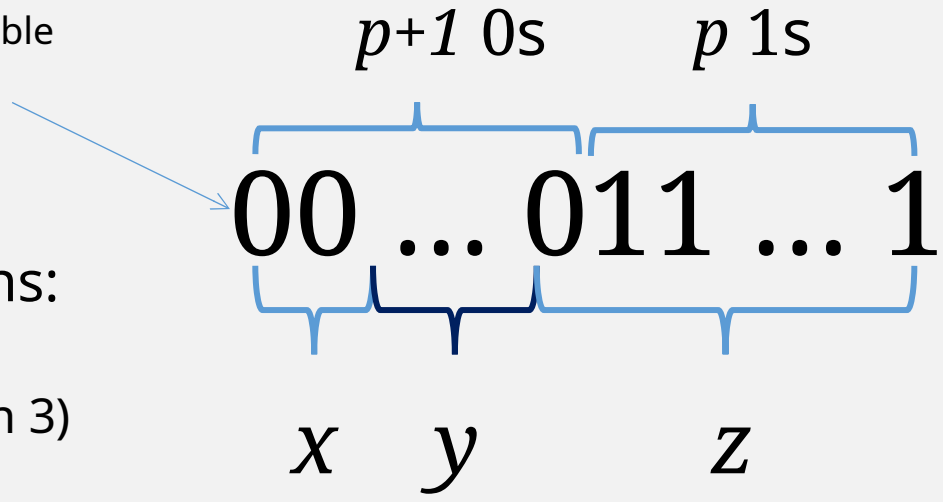
Want to prove: $L = \{0^i 1^j \mid i > j\}$ **is not** a regular language

Pumping Down

Proof (by contradiction):

- Assume: L **is** a regular language
  - So it must satisfy the pumping lemma
  - I.e., all strings $\geq$ length p are pumpable
- Counterexample = $0^{p+1}1^p$
- Choose xyz split so y contains:
  - all 0s
  - (Only possibility, by condition 3)

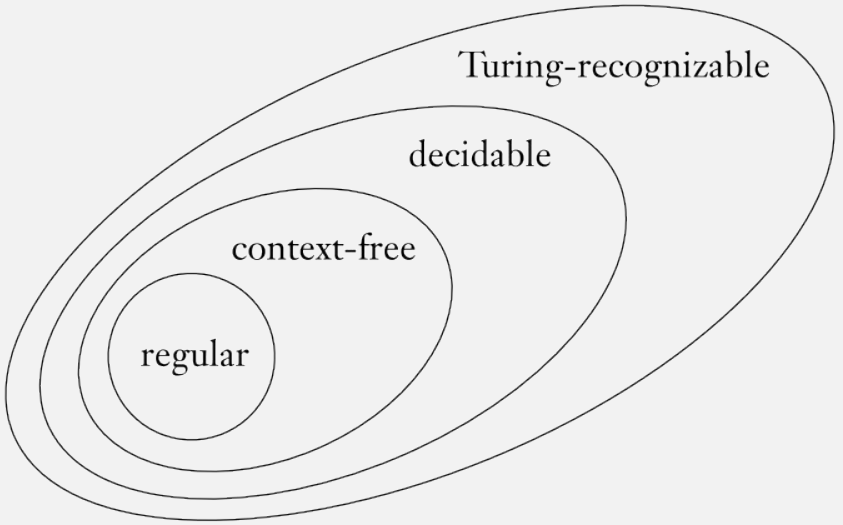
contradiction



- Repeat y zero times (pump down): produces string with # 0s ≤ # 1s
  - … not in the language $\{0^i 1^j \mid i > j\}$
  - So $\{0^i 1^j \mid i > j\}$ does not satisfy the pumping lemma
  - So it is a not regular language
  - This is a contradiction of the assumption!

Next Time (and rest of the Semester)

- If a language is not regular, then what is it?
- There are many more classes of languages!



Submit in-class work 3/6

On gradescope