
# Regular Expressions and Inductive Proofs

Wed Feb 17, 2021

# Logistics

- HW2 solutions posted

- HW3 due Sunday 2/21 11:59pm EST
  - Mostly a repeat of HW1-2 tasks, but for NFAs
  - <u>Note</u>: last question is non-coding

- Coding in this class:
  - Forces you to be precise
  - Reinforces that we are studying **computation**
    - and meta-computation!
    - Proof by construction = algorithm = computation by a more powerful computer!
    - (see next slide)
  - As computational models get complex, we will transition to on-paper proofs

- Questions?

# Review: HW2, Intersection Problem

## Password Requirements

- Passwords must have a minimum length of ten (10) characters - but more is better! ← State machine
- Passwords **must include at least 3** different types of characters:
  - upper-case letters (A-Z) ← State machine
  - lower-case letters (a-z) ← State machine
  - symbols or special characters (%, &, *, $, etc.) ← State machine
  - numbers (0-9) ← State machine
- Passwords cannot contain all or part of your email address ← State machine
- Passwords cannot be re-used ← State machine

**Combination** of these machines is also a state machine.

**But what kind of computer is needed to perform the combining?**

# Review: HW2, Intersection Problem

```
def DFA_Intersection(DFA1,DFA2):

    DFA = {'states':set(),'sigma':set(),'delta':{},'start':"",'accepts':set()}

    DFA['states'] = set(it.product(DFA1['states'],DFA2['states']))

    DFA['sigma'] = set.union(DFA1['sigma'],DFA2['sigma'])

    DFA['start'] = (DFA1['start'],DFA2['start'])

    DFA['accepts'] = set(it.product(DFA1['accepts'],DFA2['accepts']))

    for state in DFA['states']:
        DFA['delta'][state] = {}
        for string in DFA['sigma']:
            DFA['delta'][state][string] = (DFA1['delta'][state[0]][string],DFA2['delta'][state[1]][string])

    return DFA

M1_I_M2 = DFA_Intersection(M1,M2) # M1 and M2 intersection
M3_I_M4 = DFA_Intersection(M3,M4) # M3 and M4 intersection
DFA_Final = DFA_Intersection(M1_I_M2,M3_I_M4) # Final DFA i.e. intersection of M1,M2,M3,M4

# String check condition.
if(run(DFA_Final,string)):
    sys.stdout.write("valid")
```

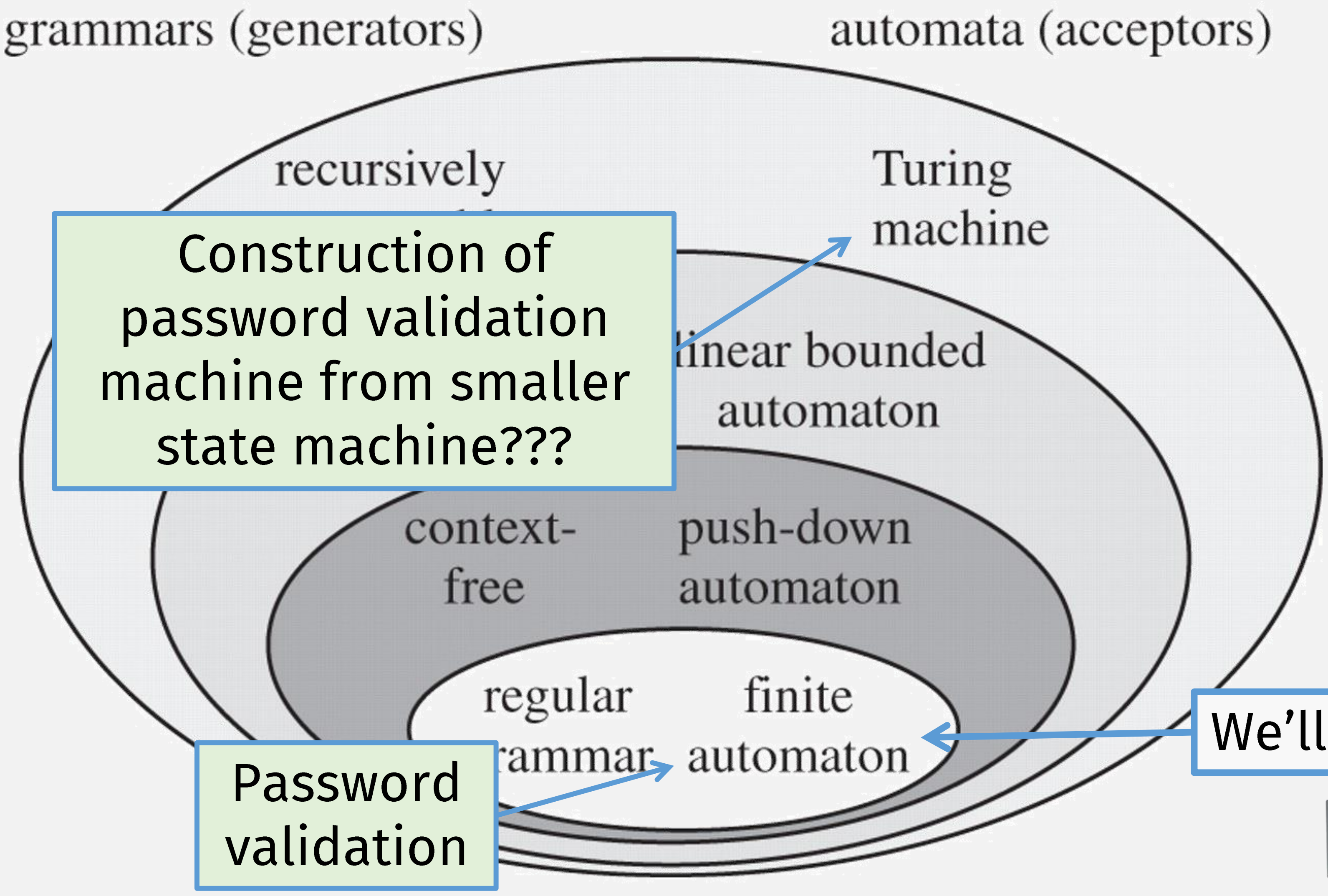
A more powerful "computer" needed to combine state machines

State machines

Combined state machine

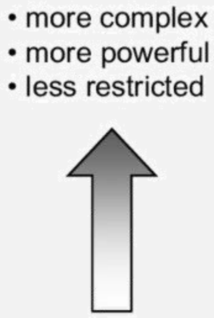
Password checked by state machine

# Flashback: Levels of Computational *Power*




Construction of password validation machine from smaller state machine???

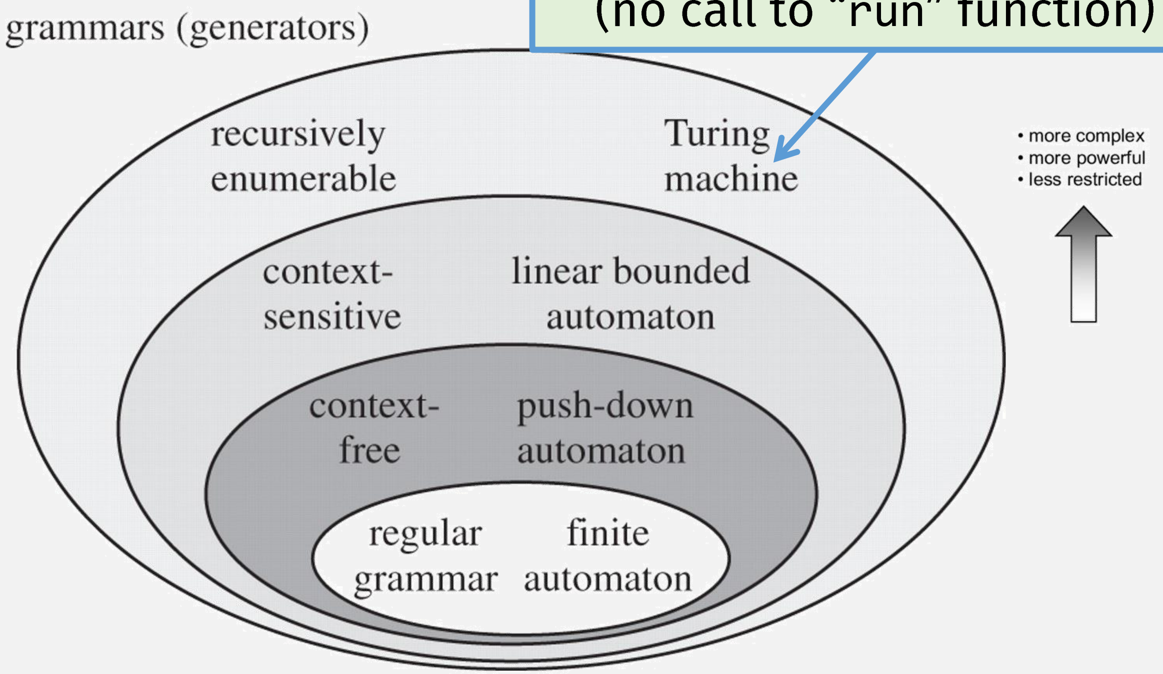
Password validation




We'll start here

# Review: HW2, Intersection Problem: A different answer

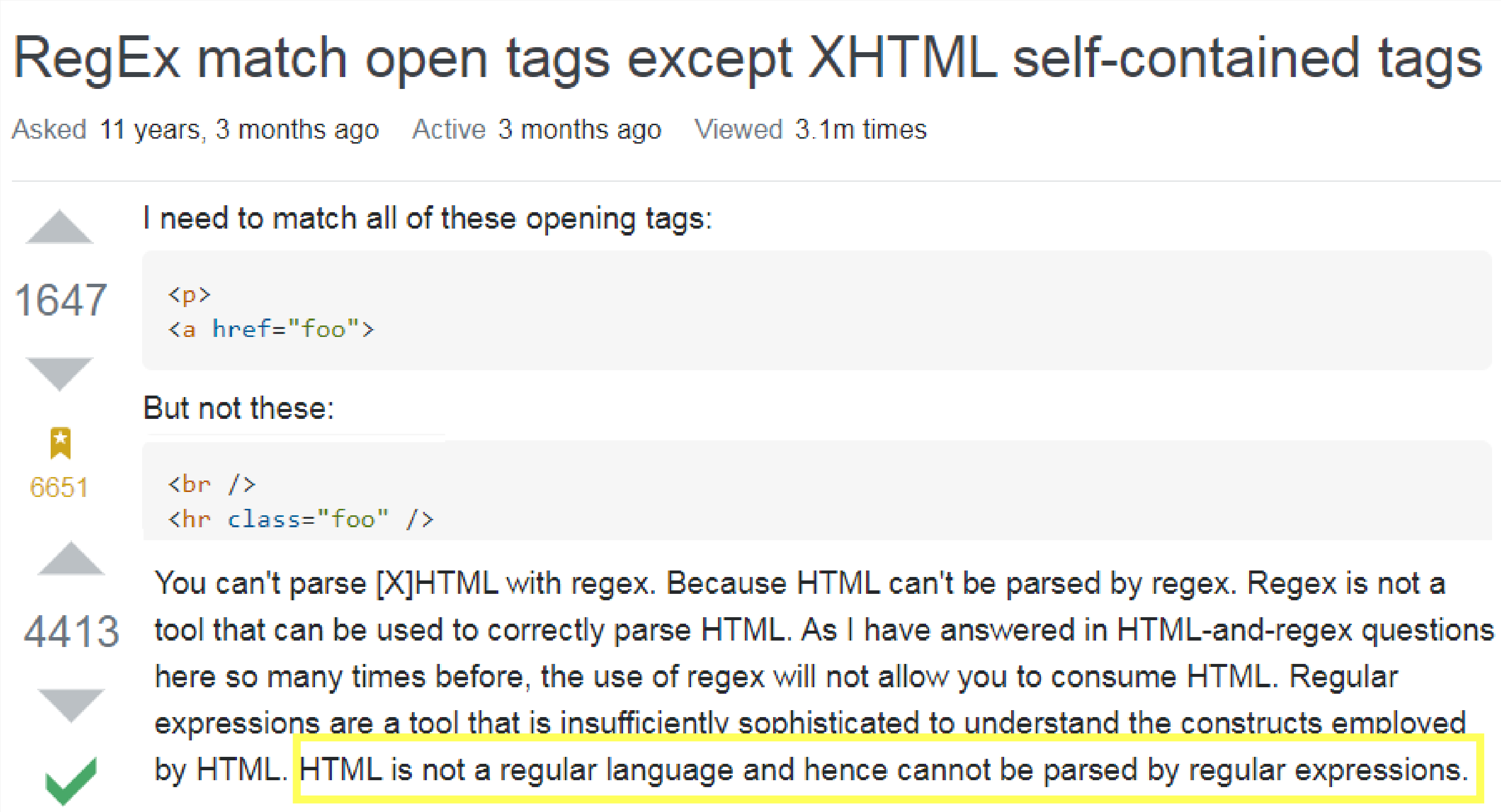
Password checking **not** computed by state machine (no call to "run" function)




```
def inrersection(dfa1, dfa2, dfa3, dfa4, password):
    flag1 = 0
    flag2 = 0
    flag3 = 0
    flag4 = 0
    for char in password:
        if (char in dfa1.alphabet): flag1 = 1
        elif (char in dfa2.alphabet): flag2 = 1
        elif (char in dfa3.alphabet): flag3 = 1
    i = len(password)
    j = len(dfa4.alphabet)
    if (i >= j): flag4 = 1
    if (flag1 and flag2 and flag3 and flag4):
        print("valid")
    else:
        print("invalid")
```

# Last time: Regular Expressions

- **Regular expressions** are widely used by programmers
  - But they can only match regular languages
  - So to *properly* use reg. exps, you must know what is/isn't a regular lang!



RegEx match open tags except XHTML self-contained tags

Asked 11 years, 3 months ago Active 3 months ago Viewed 3.1m times

1647

I need to match all of these opening tags:

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

But not these:

```
<br />
<hr class="foo" />
```

6651

4413

You can't parse [X]HTML with regex. Because HTML can't be parsed by regex. Regex is not a tool that can be used to correctly parse HTML. As I have answered in HTML-and-regex questions here so many times before, the use of regex will not allow you to consume HTML. Regular expressions are a tool that is insufficiently sophisticated to understand the constructs employed by HTML. HTML is not a regular language and hence cannot be parsed by regular expressions.

# Last time: Big Picture Road Map

- In this course, we must formally prove the equivalence:
  - Regular Languages ⇔ Regular Expressions ← Today!

- To do so, we need to prove these ops are closed under reg langs:
  - Union (**done**!)
  - Concatentation (**done**!)
  - Kleene star (**done**!)

- To prove closure properties, we using NFAs:
  - Need NFA ⇔ DFA equivalence theorem (**done**!)

# By the end of class today …

- We'll have proven that all these are equivalent:
  - Deterministic Finite Automaton (DFA)
  - Non-deterministic Finite Automaton (NFA)
  - Generalized Non-deterministic Finite Automaton (GNFA)
  - Regular Expressions

- They all represent a regular language!

# Regular Expressions, Formal Definition

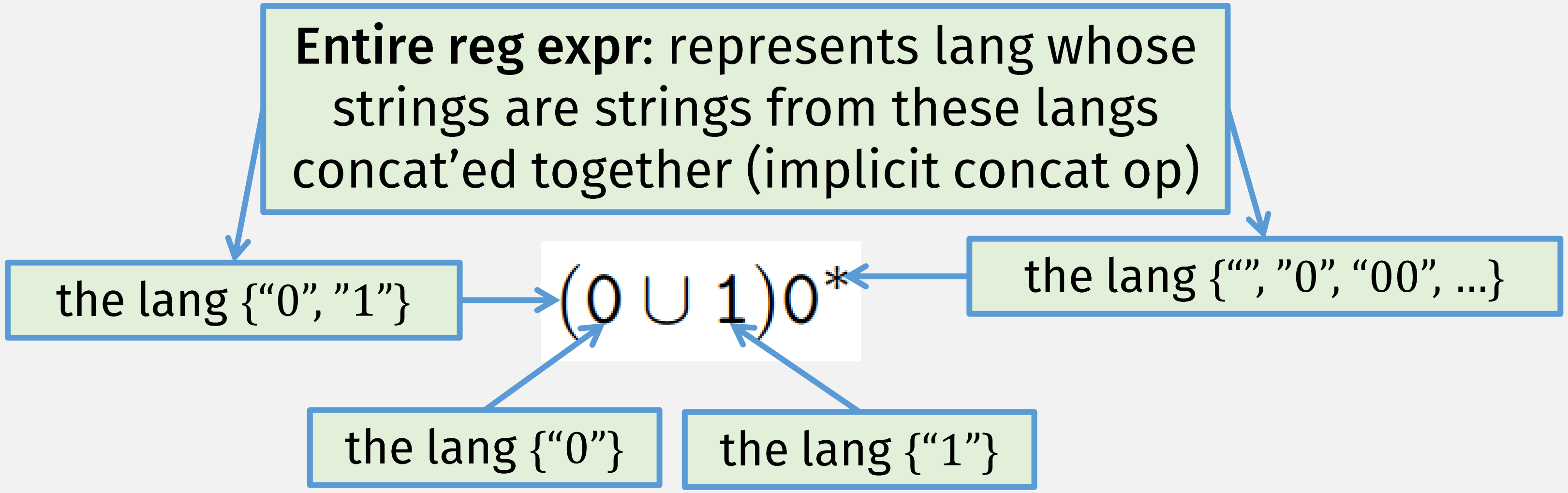
Remember:
A **Regular Expression** represents a (regular) language, i.e., a set of strings!

**DEFINITION 1.52**

Say that $R$ is a ***regular expression*** if $R$ is

1. $a$ for some $a$ in the alphabet $\Sigma$, — (A lang containing a) length-1 string
2. $\varepsilon$, — (A lang containing) the empty string
3. $\emptyset$, — The empty set (i.e., a lang containing no strings)
4. $(R_1 \cup R_2)$, where $R_1$ and $R_2$ are regular expressions, — Union of langs
5. $(R_1 \circ R_2)$, where $R_1$ and $R_2$ are regular expressions, or — Concat of langs
6. $(R_1^*)$, where $R_1$ is a regular expression. — Star of langs

# Regular Expression: Concrete Example




- Operator Precedence:
  - Parens
  - Star
  - Concat (sometimes implicit)
  - Union

# Thm: A lang is regular **iff** some reg expr describes it

- => If a language is regular, it is described by a reg expr

- <= If a language is described by a reg expr, it is regular
  - Easy!
  - For a given regexp, construct the equiv NFA!
  - See Lemma 1.55

How to show that a lang is regular?

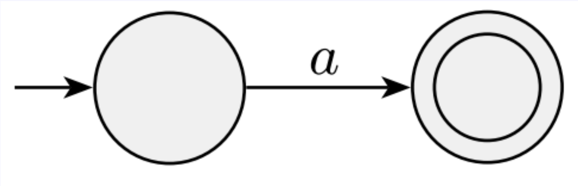
**Construct DFA *or* NFA!**

# Lemma 1.55: Regexp -> NFA

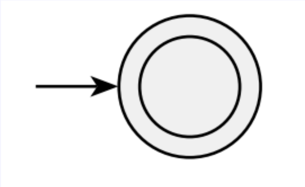
**DEFINITION 1.52**

Say that $R$ is a ***regular expression*** if $R$ is

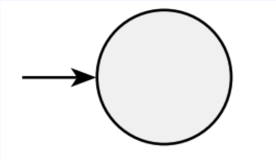
1. $a$ for some $a$ in the alphabet $\Sigma$,




2. $\varepsilon$,



3. $\emptyset$,



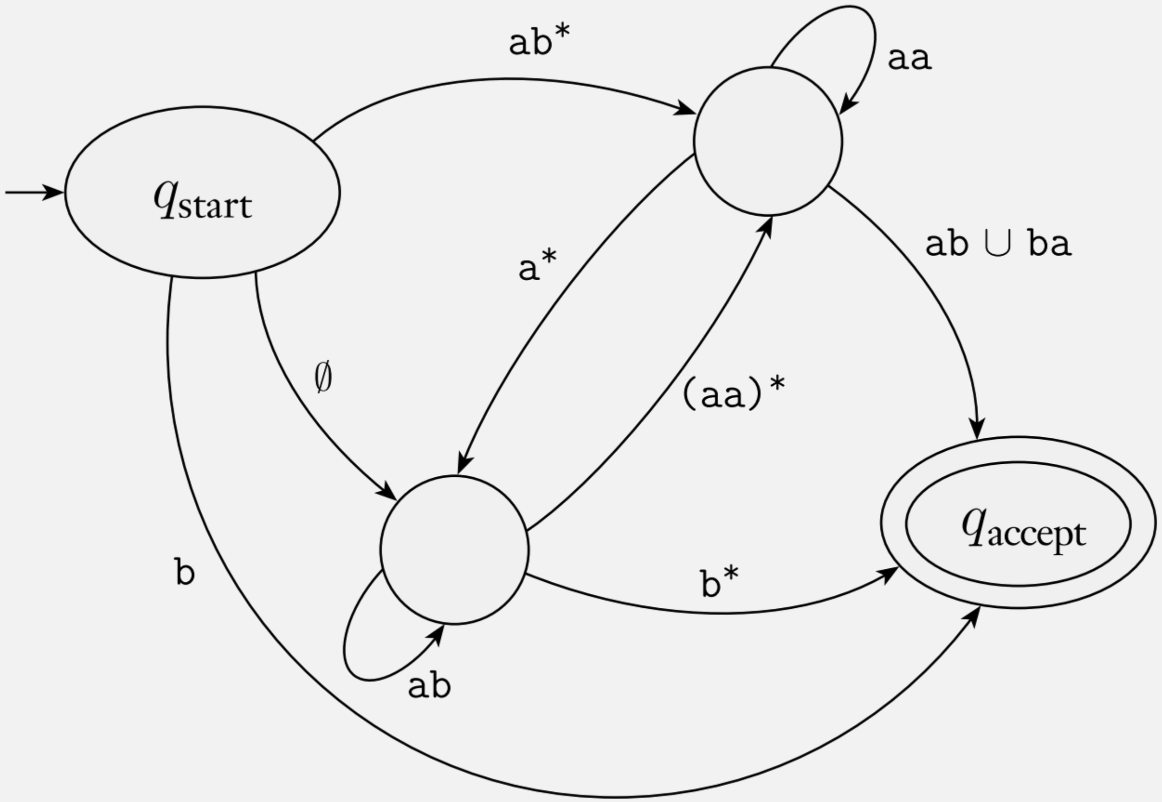
4. $(R_1 \cup R_2)$, where $R_1$ and $R_2$ are regular expressions,
5. $(R_1 \circ R_2)$, where $R$ … r
6. $(R_1^*)$, where $R_1$ is a regular expression.

Constructions from before!

# Thm: A lang is regular **iff** some reg expr describes it

- => If a language is regular, it is described by a reg expr
  - Hard!
  - Need to convert DFA or NFA to Regular Expression
  - Need something new: a GNFA
- <= If a language is described by a reg expr, it is regular
  - Easy!
  - Construct the NFA! (**Done**)

# Generalized NFAs (GNFAs)

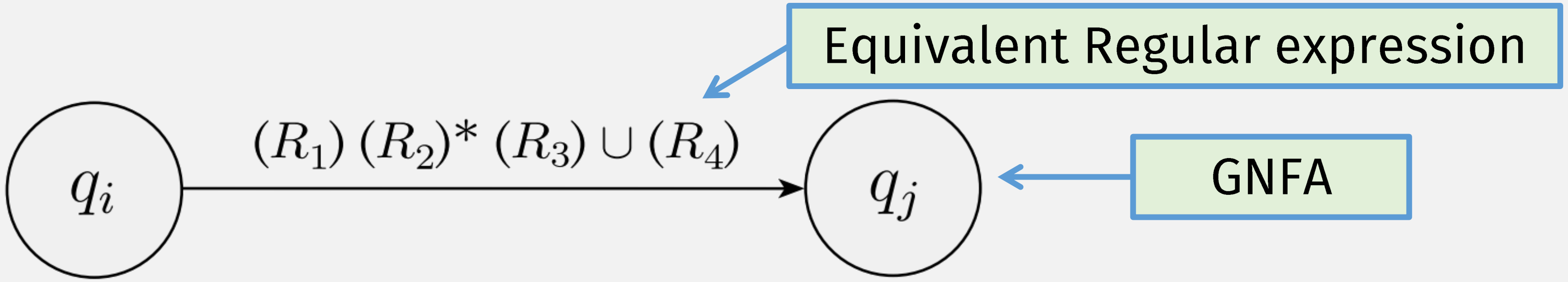



- GNFA = NFA with regular expression transitions

**Want to convert GNFAs to Reg Exprs**

# GNFA->Regexp function

- On GNFA <u>input</u> G:
- If G has 2 states, return the regular expression transition, e.g.:




- Else:
  - "Rip out" one state, and "repair", to get G' (has one less state than G)
  - <u>Recursively</u> call GNFA->Regexp(G')

**A recursive (function) definition!**

# Recursive (Inductive) Definitions

- (at least) two parts:
  - Base case
  - Inductive case
    - Self-reference must be "*smaller*" than the whole

- <u>Example</u>: factorial function

```
def factorial(n):

    if n == 0:
        return 1

    return n * factorial(n-1)
```

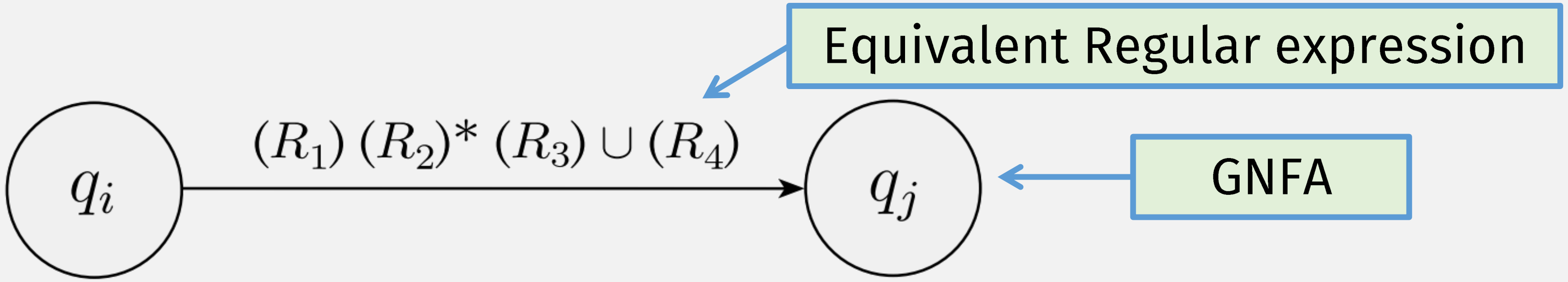
What's the base case?

Self-reference smaller than the whole

# GNFA->Regexp function

- On GNFA input G:
- If G has 2 states, return the regular expression transition, e.g.:

Base case



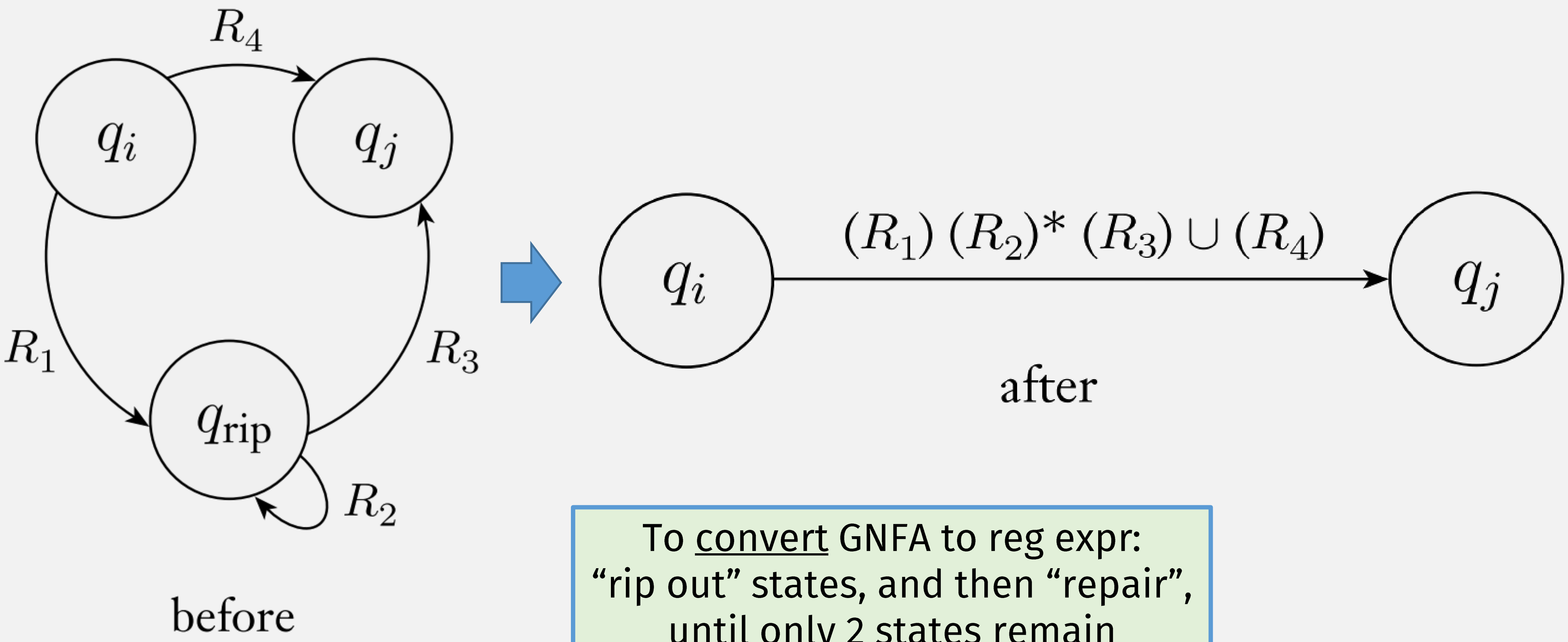

Inductive case

- Else:
  - "Rip out" one state, and "repair", to get G' (has one less state than G)
  - Recursively call GNFA->Regexp(G')

Recursive call is "smaller"

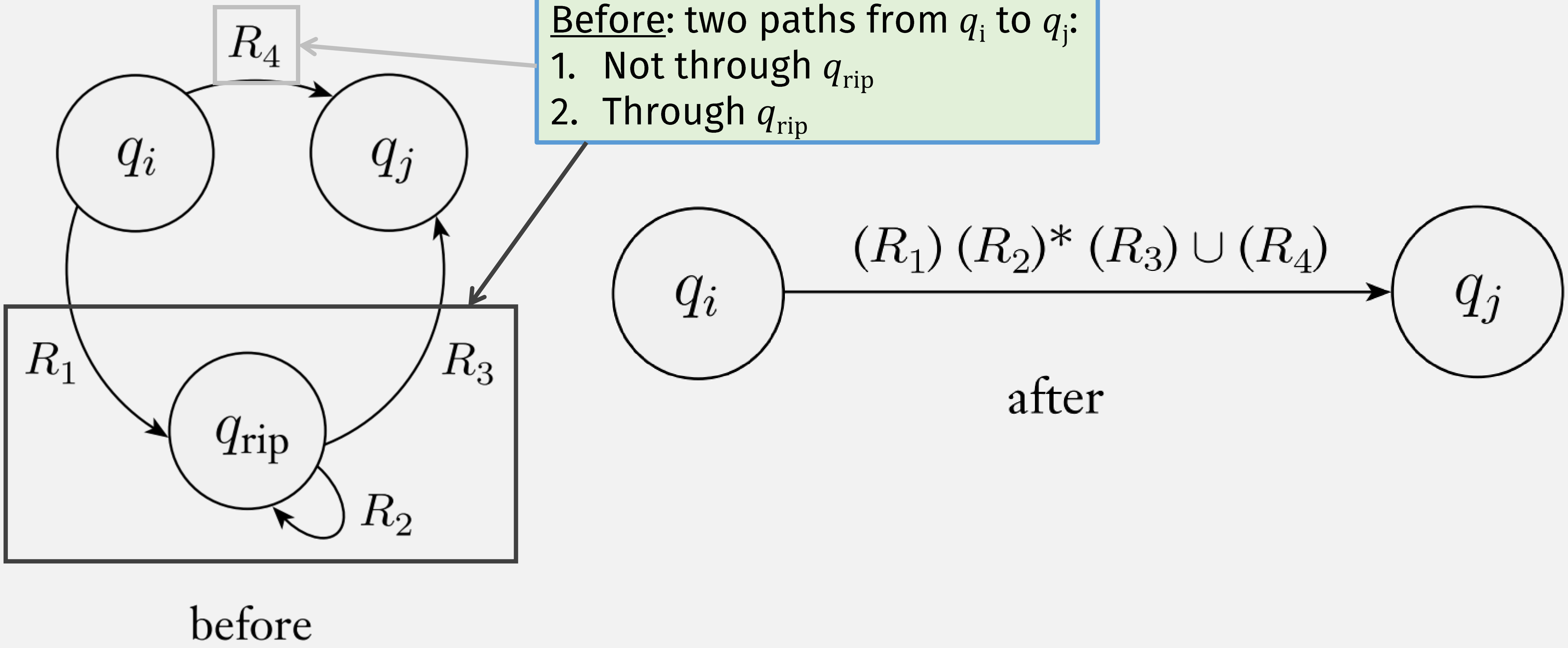
**A recursive (function) definition!**

# GNFA->Regexp function: "Rip/repair" step




To <u>convert</u> GNFA to reg expr:
"rip out" states, and then "repair",
until only 2 states remain

# GNFA->Regexp function: “Rip/repair” step



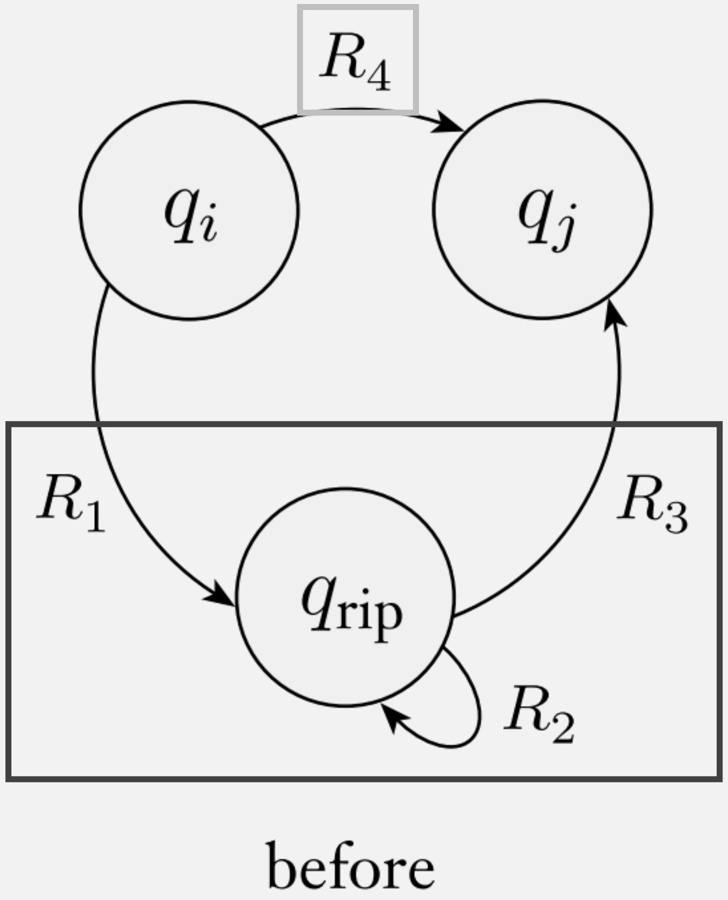

Before: two paths from $q_i$ to $q_j$:

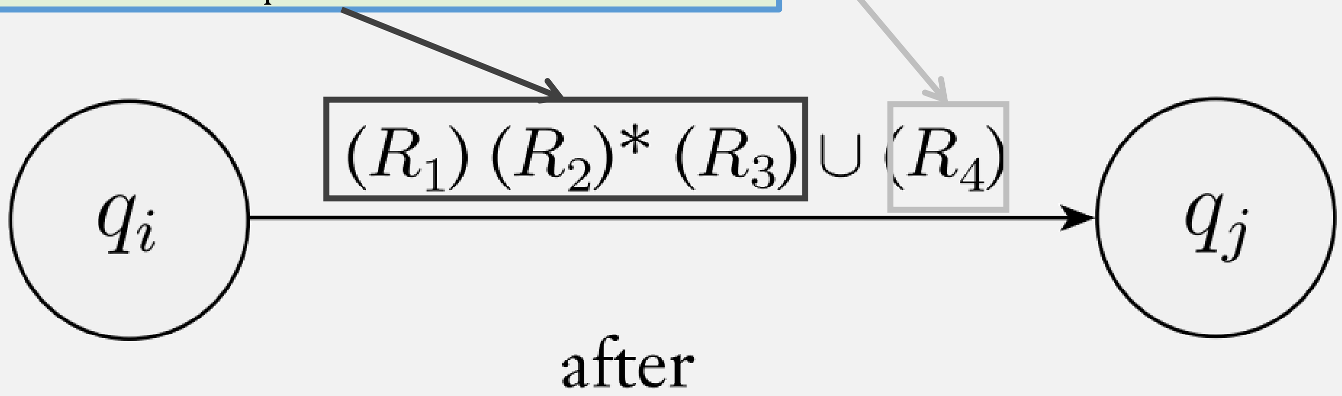
1. Not through $q_{rip}$
2. Through $q_{rip}$

# GNFA->Regexp function: "Rip/repair" step

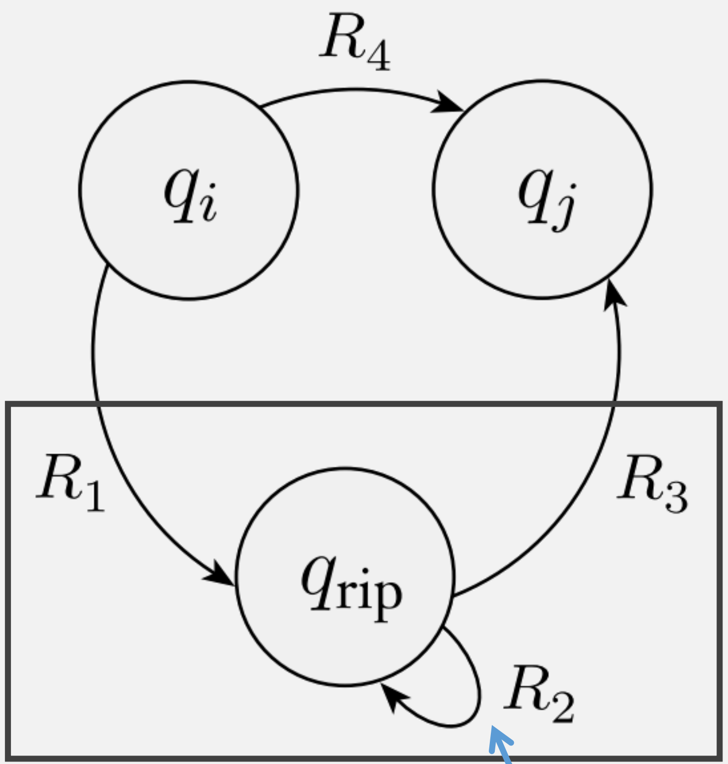
After: still two "paths" from $q_i$ to $q_j$

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

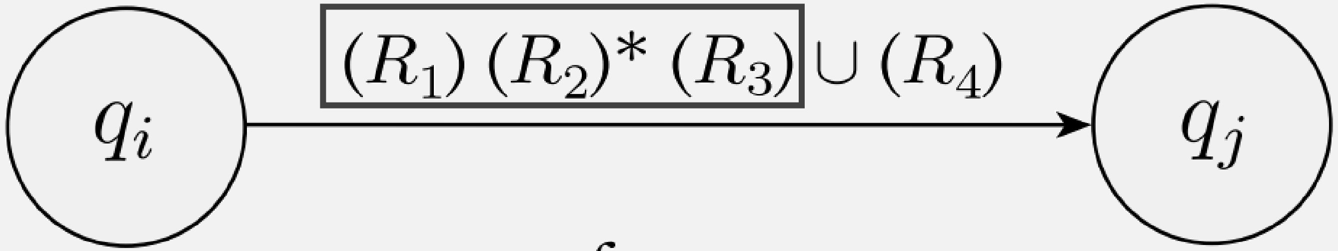


# GNFA->Regexp function: “Rip/repair” step




before

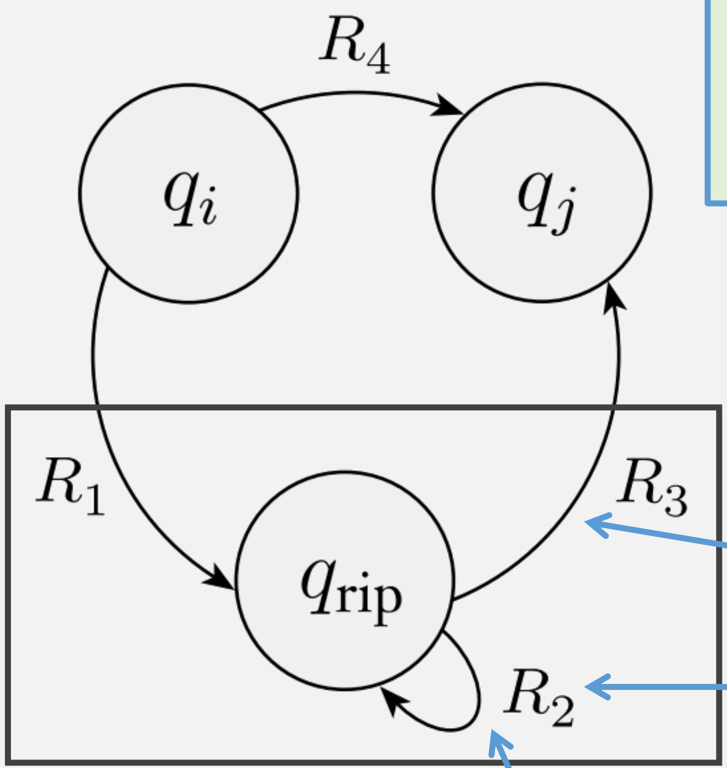



after

<u>Before</u>:

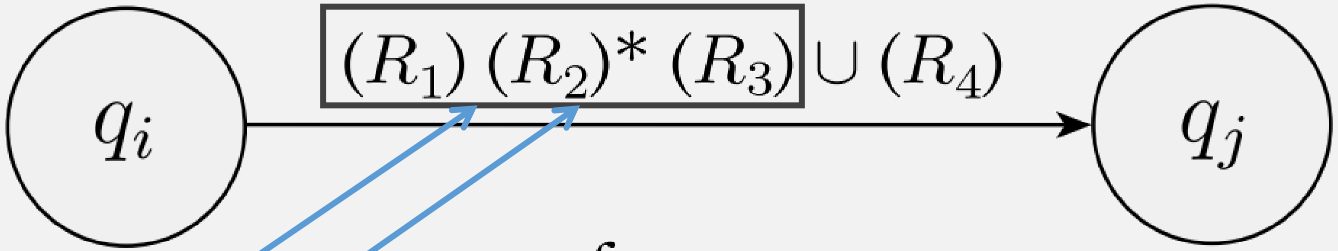
- path through $q_{\text{rip}}$ has 3 transitions
- One is self loop

# GNFA->Regexp function: "Rip/repair" step




before

**After**:
- Self loop becomes star operation
- Others are concat'ed together




after

concat

Star operation

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

**This is "informal" correctness**

**This course requires formal correctness, i.e., proof**

# Need to prove GNFA->Regexp “correct”

- Where “correct” means:

LANGOF ( G ) = LANGOF ( GNFA->Regexp ( G ) )

- i.e., GNFA->Regexp must not change the language!

# Kinds of Mathematical Proof

- Proof by construction

- Proof by contradiction

- Proof by induction
  - Use to prove properties of recursive (inductive) defs or functions

# Proof by Induction

- To prove that a **property** $\mathrm{P}$ is true for a **thing** $\mathrm{x}$
  - First, prove that $\mathrm{P}$ is true for the base case of $\mathrm{x}$ (usually easy)
  - Then, prove the induction step:
    - Assume the induction hypothesis (IH):
      - $\mathrm{P(x)}$ is true, for some $\mathrm{x_{smaller}}$ that smaller than $\mathrm{x}$
    - and use it to prove $\mathrm{P(x)}$
    - The *key* is $\mathrm{x_{smaller}}$ must be smaller than $\mathrm{x}$

- Why can we assume IH is true???
  - Because we can always start at base case,
  - Then use it to prove for slightly larger case,
  - Then use that to prove for slightly larger case …

# Need to prove GNFA->Regexp "correct"

- Where "correct" means:

LANGOF ( G ) = LANGOF ( GNFA->Regexp ( G ) )

This is the "thing" we want to prove it for

This is the property we want to prove

- i.e., GNFA->Regexp must not change the language!

# GNFA->Regexp is correct

LangOf ( G )
=
LangOf ( GNFA->Regexp ( G ) )

Def: GNFA->Regexp: input G is a GNFA with n states:
If n = 2: return the reg expr on the transition
Else (G has n > 2 states):
"Rip" out one state to get G'
Recursively Call GNFA->Regexp(G')

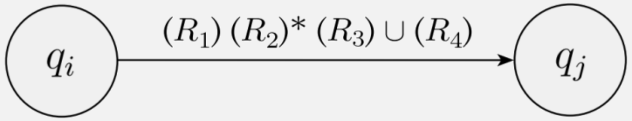
- Proof (by induction on size of G):

# GNFA->Regexp is correct

LANGOF ( G )
=
LANGOF ( GNFA->Regexp ( G ) )

Def: GNFA->Regexp: input G is a GNFA with n states:
    If n = 2: return the reg expr on the transition
    Else (G has n > 2 states):
        "Rip" out one state to get G'
        Recursively Call GNFA->Regexp(G')

- Proof (by induction on size of G):
    - Base case: G has 2 states
        - LANGOF ( G ) = LANGOF ( GNFA->Regexp ( G ) ) is true!

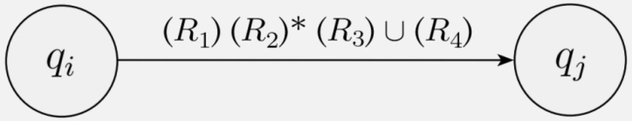


# GNFA->Regexp is correct

LANGOF ( G )
=
LANGOF ( GNFA->Regexp ( G ) )

<u>Def</u>: GNFA->Regexp: input G is a GNFA with n states:
- If n = 2: return the reg expr on the transition
- Else (G has n > 2 states):
  - "Rip" out one state to get G'
  - Recursively Call GNFA->Regexp(G')

- <u>Proof</u> (by induction on size of G):
  - <u>Base case</u>: G has 2 states
    - LANGOF ( G ) = LANGOF ( GNFA->Regexp ( G ) ) is true!
  - <u>IH</u>: Assume LANGOF ( G' ) = LANGOF ( GNFA->Regexp ( G' ) )
    - For some G' with <u>n-1</u> states

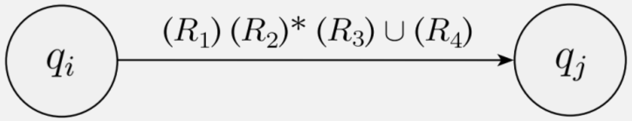


# GNFA->Regexp is correct

LANGOF ( G )
=
LANGOF ( GNFA->Regexp ( G ) )

Def: GNFA->Regexp: input G is a GNFA with n states:
If n = 2: return the reg expr on the transition
Else (G has n > 2 states):
"Rip" out one state to get G'
Recursively Call GNFA->Regexp(G')

- Proof (by induction on size of G):
  - Base case: G has 2 states
    - LANGOF ( G ) = LANGOF ( GNFA->Regexp ( G ) ) is true!
  - IH: Assume LANGOF ( G' ) = LANGOF ( GNFA->Regexp ( G' ) )
    - For some G' with n-1 states
  - ➢Induction Step: Prove it's true for G with n states

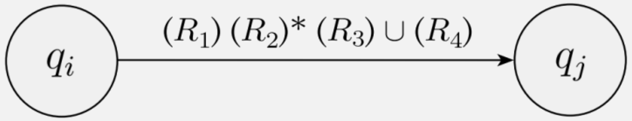


# GNFA->Regexp is correct

LangOf ( G )
=
LangOf ( GNFA->Regexp ( G ) )

Def: GNFA->Regexp: input G is a GNFA with n states:
If n = 2: return the reg expr on the transition
Else (G has n > 2 states):
"Rip" out one state to get G'
Recursively Call GNFA->Regexp(G')

- Proof (by induction on size of G):



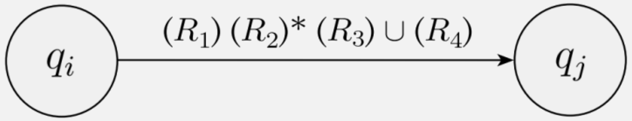

  - Base case: G has 2 states
    - LangOf ( G ) = LangOf ( GNFA->Regexp ( G ) ) is true!
  - IH: Assume LangOf ( G' ) = LangOf ( GNFA->Regexp ( G' ) )
    - For some G' with n-1 states
  - ➢ Induction Step: Prove it's true for G with n states
    - After "rip" step, we have exactly a GNFA with n-1 states
    - And we know LangOf ( G' ) = LangOf ( GNFA->Regexp ( G' ) ) from the IH!

# GNFA->Regexp is correct

LANGOF ( G )
=
LANGOF ( GNFA->Regexp ( G ) )

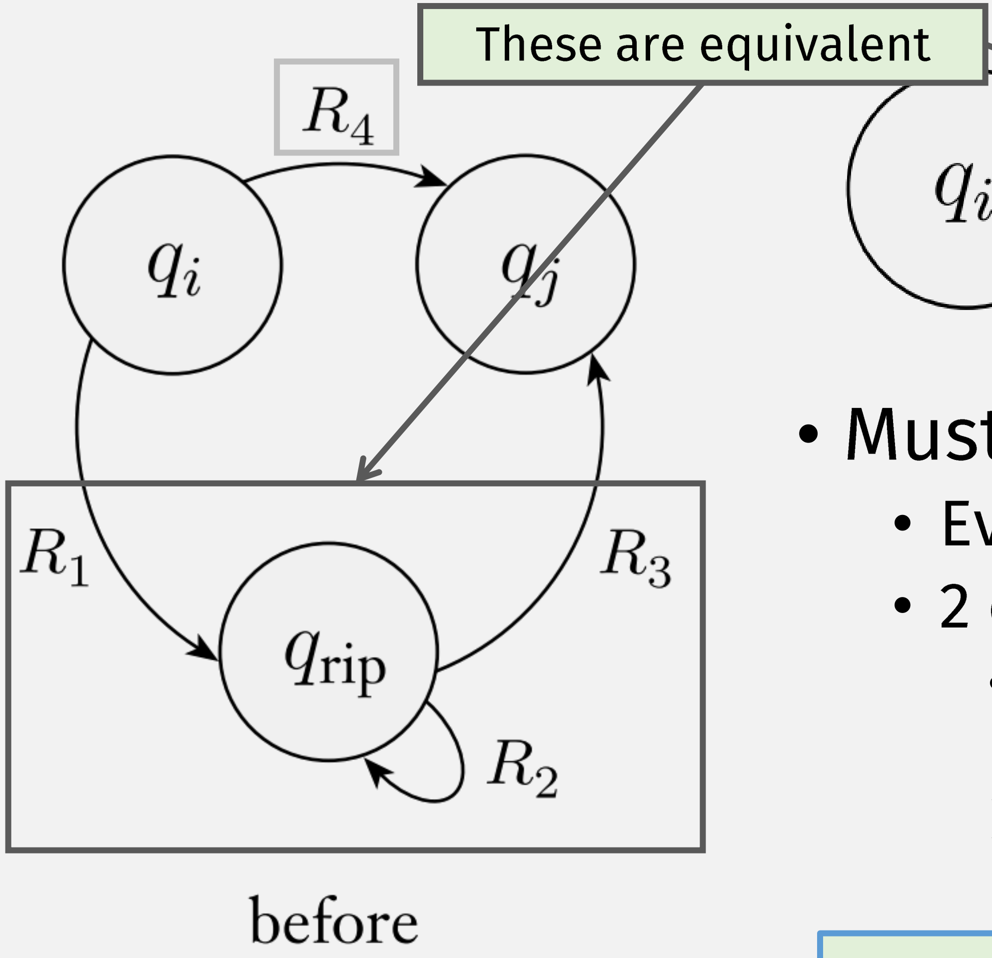
Def: GNFA->Regexp: input G is a GNFA with n states:
If n = 2: return the reg expr on the transition
Else (G has n > 2 states):
"Rip" out one state to get G'
Recursively Call GNFA->Regexp(G')

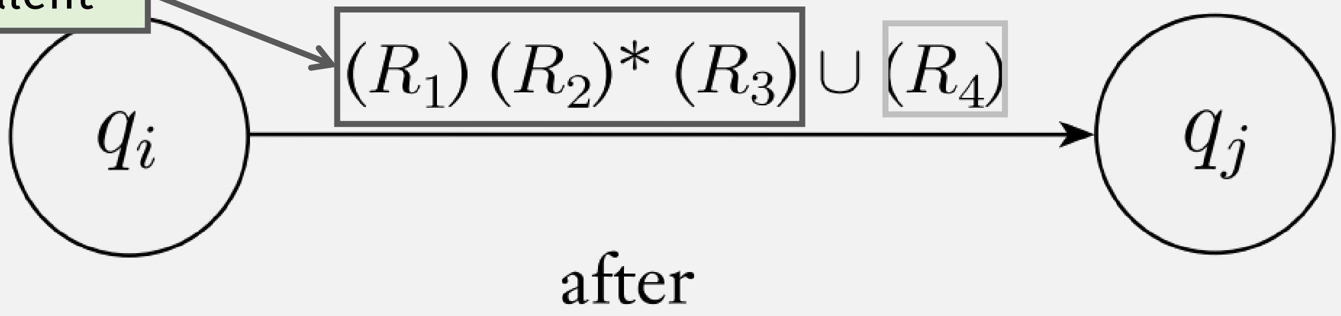
- Proof (by induction on size of G):
  - Base case: G has 2 states
    - LANGOF ( G ) = LANGOF ( GNFA->Regexp ( G ) ) is true!




  - IH: Assume LANGOF ( G' ) = LANGOF ( GNFA->Regexp ( G' ) )
    - For some G' with n-1 states
  - Induction Step: Prove it's true for G with n states
    - After "rip" step, we have exactly a GNFA with n-1 states
    - And we know LANGOF ( G' ) = LANGOF ( GNFA->Regexp ( G' ) ) from the IH!
    - ➢ To go from G to G': need to prove correctness of "rip" step

# GNFA->Regexp: "rip" step correctness




- Must <u>prove</u>:
  - Every string accepted <u>before</u>, is accepted <u>after</u>
  - 2 cases:
    - Accepted string does not go through $q_{rip}$
      - Acceptance unchanged (both use $R_4$ transition part)
    - ➢ String goes through $q_{rip}$
      - Acceptance unchanged?

**Mostly done this already!**
**Just need to state more formally**

# Thm: A lang is regular **iff** some reg expr describes it

- => If a language is regular, it is described by a reg expr
  - Hard!
  - Need to convert DFA or NFA to Regular Expression
  - Use GNFA->Regexp to convert GNFA to regular expression! **(Done!)**
- <= If a language is described by a reg expr, it is regular
  - Easy!
  - Construct the NFA! (**Done**)

Now we may confidently use regular expressions to represent regular langs.

# Check-in Quiz 10/17

On gradescope

# End of Class Survey 10/17

See course website