Regular Expressions and Inductive Proofs

Mon Sept 28, 2020

HW2 questions?

Big Picture Road Map

- We ultimately want to prove:
  - Regular Languages ⇔ Regular Expressions

Finish today!

- First, we need to show these operations are closed for reglangs:
  - Union (**done**!)
  - Concatentation (**done**!)
  - Kleene star (**done**!)

Thm: A lang is regular iff some regexp describes it

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

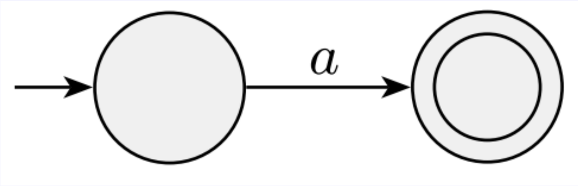
- <= If a language is described by a regexp, it is regular
  - Easy!
  - Construct the NFA! (Lemma 1.55)

`Regexp->NFA` (Lemma 1.55)

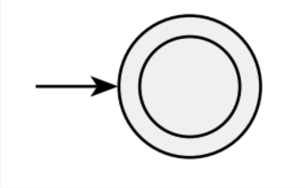
DEFINITION 1.52

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

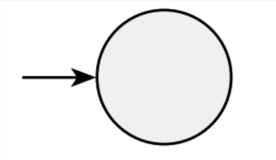
1. a for some a in the alphabet Σ,



2. ε,



3. $\emptyset$,



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

<u>Recursively</u> call `Regexp->NFA` on R_1 and R_2,
to get N_1 for R_1, and N_2 for R_2,
then combine NFAs!

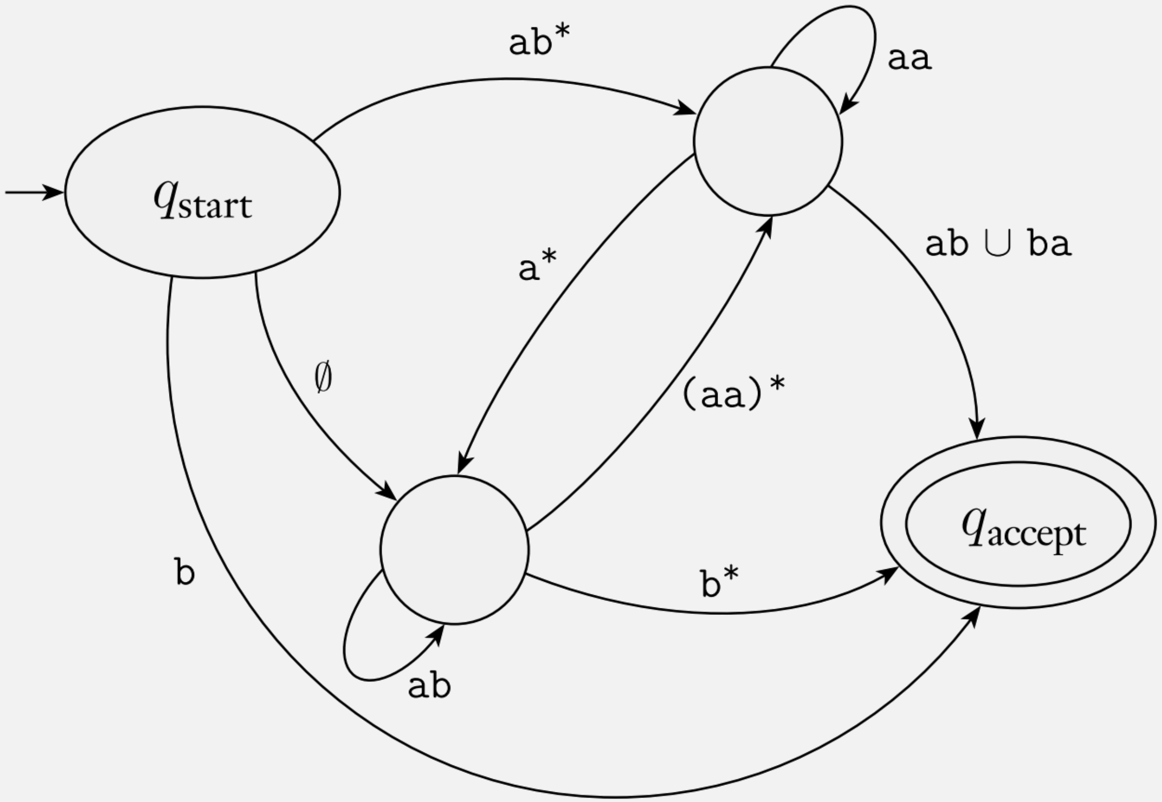
Thm: A lang is regular iff some regexp describes it

- => If a language is regular, it is described by a regexp
  - Hard!
  - Need something new: a GNFA

Next!

- <= If a language is described by a regexp, it is regular
  - Easy!
  - Construct the NFA! (Lemma 1.55)

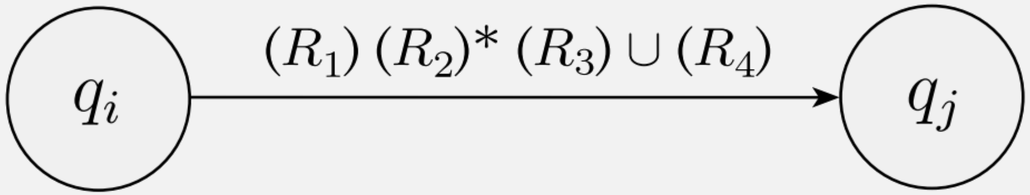
GNFA = NFA with regexp transitions



- To <u>convert</u> GNFA to regexp, repeatedly “rip out” states until 2 left

`GNFA->Regexp`(G) fn (where G is GNFA)

- If G has 2 states, return the regular expression



- Else:
  - “Rip” out one state to get G’
  - <u>Recursively</u> call `GNFA->Regexp`(G’)

Need to prove **`GNFA->Regexp`**(G) correct

- Specifically, need to prove Lang(G) = Lang(`GNFA->Regexp`(G))
- i.e., `GNFA->Regexp` should not change the language!

Kinds of Mathematical Proof

- Proof by construction
- Proof by contradiction
- Proof by induction
  - Use to prove properties of recursive definitions or functions

Proof by Induction

- To prove property P on all objects of a kind x
  - First, prove base case (usually easy)
  - Then, prove the induction step:
    - Assume the induction hypothesis (IH): P(x) is true, for some x
    - and use it to prove P(x+1)
    - The **key** is x must be smaller than x+1

Correctness of **`GNFA->Regexp`**(G)

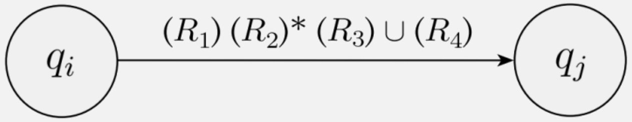
`GNFA->Regexp`(G): (G is an GNFA)
    If G has 2 states, return the regexp
    Else:
        "Rip" out one state to get G'
        Recursively Call **`GNFA->Regexp`**(G')

- <u>Prove</u> (by induction): Lang(G) = Lang(**`GNFA->Regexp`**(G))

Correctness of **GNFA->Regexp**(G)

GNFA->Regexp(G): (G is an GNFA)
    If G has 2 states, return the regexp
    Else:
        "Rip" out one state to get G'
        Recursively Call **GNFA->Regexp**(G')

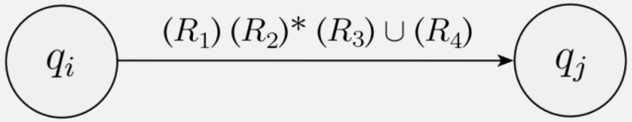
- Prove (by induction): Lang(G) = Lang(**GNFA->Regexp**(G))
  - Base case: G has 2 states
    - So Lang(G) = Lang(**GNFA->Regexp**(G))



Correctness of **GNFA->Regexp**(G)

GNFA->Regexp(G): (G is an GNFA)
  If G has 2 states, return the regexp
  Else:
    "Rip" out one state to get G'
    Recursively Call **GNFA->Regexp**(G')

- Prove (by induction): Lang(G) = Lang(**GNFA->Regexp**(G))
  - Base case: G has 2 states
    - So Lang(G) = Lang(**GNFA->Regexp**(G))

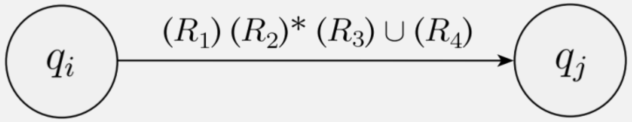


  ➢ IH: Assume Lang(G) = Lang(**GNFA->Regexp**(G)), for **any** G with n states

Correctness of **GNFA->Regexp**(G)

GNFA->Regexp(G): (G is an GNFA)
    If G has 2 states, return the regexp
    Else:
        "Rip" out one state to get G'
        Recursively Call **GNFA->Regexp**(G')

- Prove (by induction): Lang(G) = Lang(**GNFA->Regexp**(G))
  - Base case: G has 2 states
    - So Lang(G) = Lang(**GNFA->Regexp**(G))



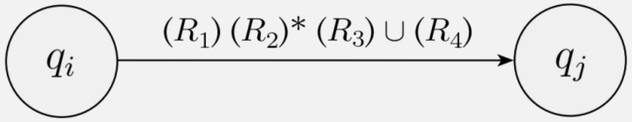
  - IH: Assume Lang(G) = Lang(**GNFA->Regexp**(G)), for **any** G with n states
  - Prove for G with n+1
    - ➢ After "rip" step, we have a G' with n states

Correctness of **GNFA->Regexp**(G)

GNFA->Regexp(G): (G is an GNFA)
    If G has 2 states, return the regexp
    Else:
        "Rip" out one state to get G'
        Recursively Call **GNFA->Regexp**(G')

In inductive proof, correctness of recursive call comes for free

- <u>Prove</u> (by induction): Lang(G) = Lang(**GNFA->Regexp**(G))
  - <u>Base case</u>: G has 2 states
    - So Lang(G) = Lang(**GNFA->Regexp**(G))

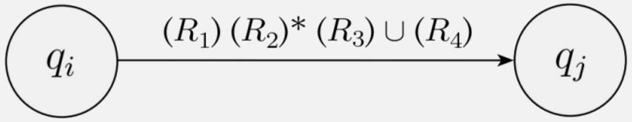


  - <u>IH</u>: Assume Lang(G) = Lang(**GNFA->Regexp**(G)), for **any** G with <u>n</u> states
  - Prove for G with <u>n+1</u>
    - After "rip" step, we have a G' with <u>n</u> states
    - ➢ Lang(G') = Lang(**GNFA->Regexp**(G')) (by assumption)

Correctness of **GNFA->Regexp**(G)

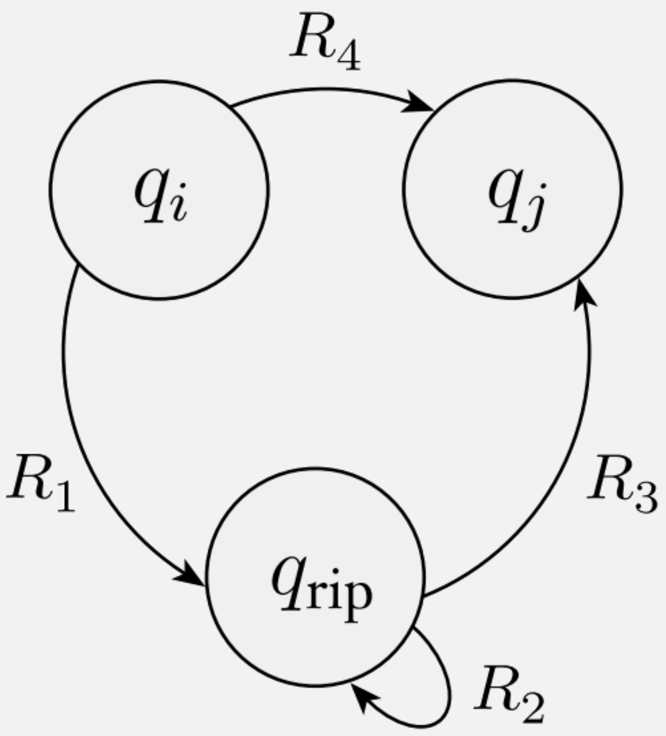
GNFA->Regexp(G): (G is an GNFA)
    If G has 2 states, return the regexp
    Else:
        "Rip" out one state to get G'
        Recursively Call **GNFA->Regexp**(G')

- <u>Prove</u> (by induction): Lang(G) = Lang(**GNFA->Regexp**(G))
  - <u>Base case</u>: G has 2 states
    - So Lang(G) = Lang(**GNFA->Regexp**(G))

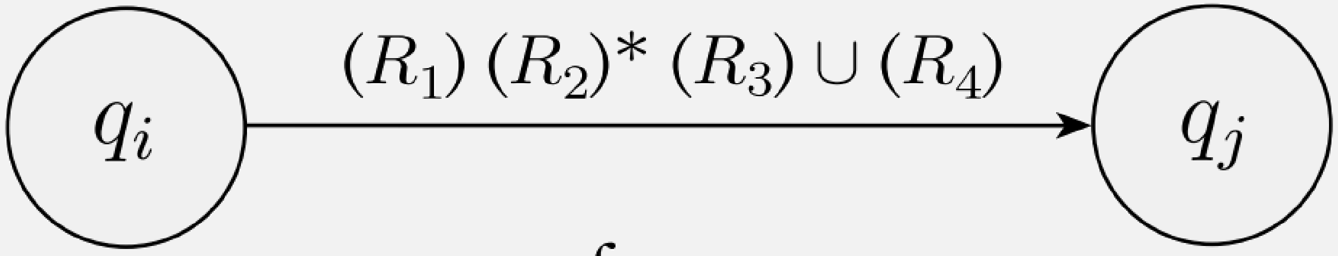


  - <u>IH</u>: Assume Lang(G) = Lang(**GNFA->Regexp**(G)), for **any** G with <u>n</u> states
  - Prove for G with <u>n+1</u>
    - After "rip" step, we have a G' with <u>n</u> states
    - Lang(G') = Lang(**GNFA->Regexp**(G')) (by assumption)
    - ➢ Now just need correctness of "rip" step

GNFA->Regexp: “rip” step correctness



before



after

- Must prove:
  - Every string accepted before is accepted after
  - 2 cases
    - String does not go through qrip
      - Acceptance unchanged
    - ➢ String goes through qrip
      - Acceptance unchanged?

Thm: A lang is regular iff some regexp describes it

- => If a language is regular, it is described by a regexp
  - Hard!
  - Use `GNFA->Regexp`(G) to convert GNFA to regexp!
- <= If a language is described by a regexp, it is regular
  - Easy!
  - Construct the NFA!

DONE!
Now we may use regular expressions to
to represent regular langs.

Regexps make some closure operations easier to prove, via induction!

Regexp is inductive definition; constructed from <u>smaller</u> regexps

DEFINITION 1.52

Say that 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.
6.

Smaller regular expressions

So any inductive proof of regular <u>languages</u> can just follow this definition!

Homomorphisms: closed under reg langs

A ***homomorphism*** is a function $f\colon \Sigma \longrightarrow \Gamma$ from one alphabet to another.

- extend f to operate on strings by defining
  $$f(w) = f(w_1)f(w_2)\cdots f(w_n),$$
  where $w = w_1w_2\cdots w_n$ and each $w_i \in \Sigma$.
- extend f to operate on languages by defining $f(A) = \{f(w)|\ w \in A\}$

- Think like a secret decoder!
  - E.g., if f(x) -> c, f(y) -> a, f(z) -> t, then “xyz” -> “cat”

- <u>Prove</u>: homomorphisms are <u>closed</u> under regular langs
  - E.g., if A is regular, then $f(A)$ is regular

Homomorphisms closed for reg langs

- Proof by construction
  - If lang L is regular, then DFA M recognizes it.
  - Create M’ from M such that all transitions use new alphabet
  - (Details left to you to work out)
- Proof by induction:
  - If lang L is regular, then some regexp R describes it.

Proof by Induction

- To prove property P on all objects of a kind x
  - First, prove base case (usually easy)
  - Then, prove the induction step:
    - Assume the induction hypothesis (IH) P(x) is true, for some x
    - and use it to prove P(x+1)
    - The **key** is x must be smaller than x+1

Homomorphisms closed: inductive proof

DEFINITION 1.52

Say that 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. $(R_1 \circ R_2)$, w
6. (R_1^*), where R_1 is a regular expression.

3 base cases

IH: assume true for <u>smaller</u> R1 (and R2),
i.e., applying homomorphism produces regular lang

Now we just need to show closure of union,
concat, and star operations for reg langs ☺

Next Time: Non-regular languages

- In general, we have many ways to show a language is regular
  - Construct DFA or NFA (or GNFA)
  - Create a regular expression
- But how to show a language is not regular?
- E.g., how do we know that XML is non-regular???
- Hint: The Pumping Lemma!

Check-in Quiz 9/28

On gradescope

End of Class Survey 9/28

See course website