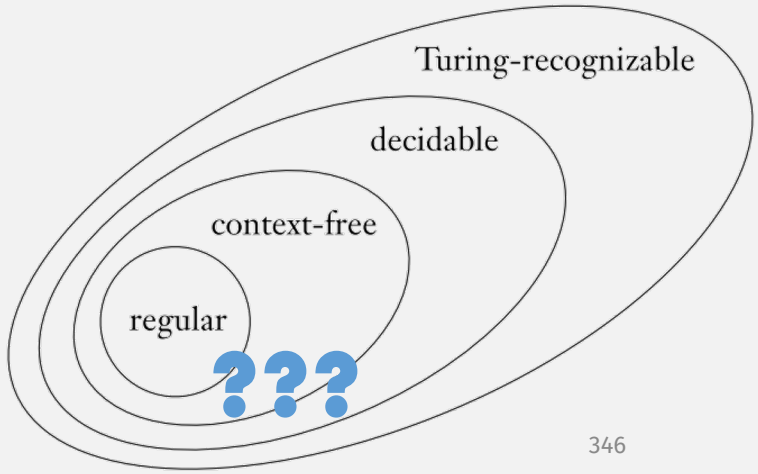
More Induction & Non-Regular Languages



Logistics

- Upcoming HW you will create a regexp matcher! Practically interesting!
- These are the last homework materials we'll cover for the first exam

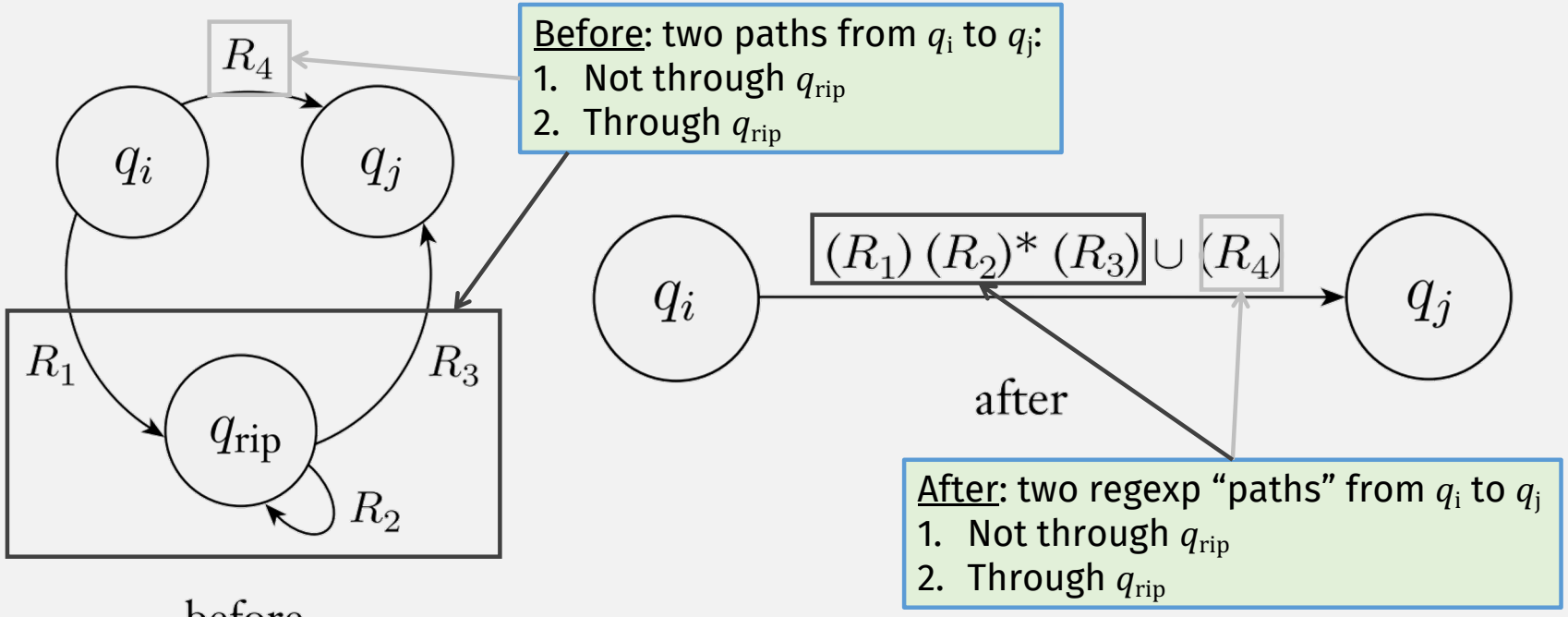
Last Time: Regular Language ⇔ Regular Expression

- => If a language is regular, it is described by a regular expression
  - We know a regular lang has an NFA recognizing it (Thm 1.40)
  - Use GNFA->Regexp function to convert NFA to equiv regular expression
- <= If a language is described by a regular expression, it is regular
  - Convert the regular expression to an NFA (Thm 1.55)

So a regular language has these equivalent representations:

- **DFA**
- **NFA**
- **Regular Expression**

Last time: GNFA->Regexp "Rip/Repair" Step



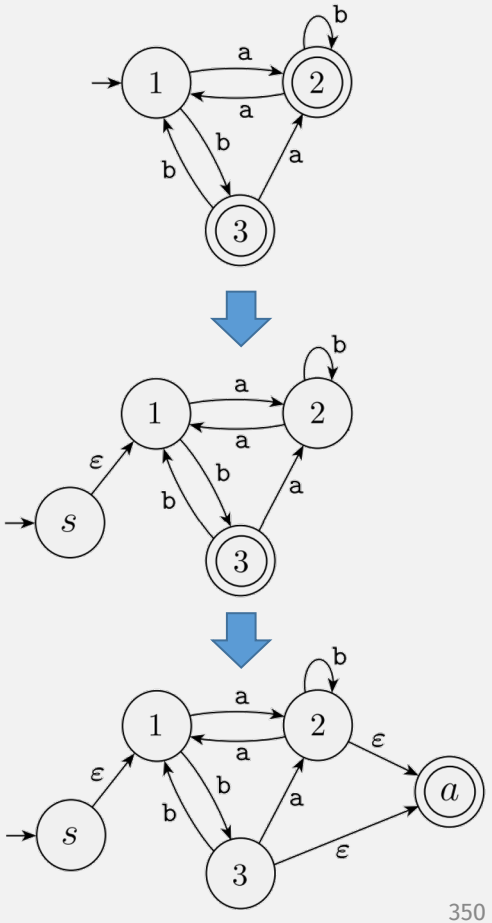
Before: two paths from q_i to q_j:
1. Not through q_{rip}
2. Through q_{rip}

After: two regexp "paths" from q_i to q_j
1. Not through q_{rip}
2. Through q_{rip}

Question: What if q_{rip} is an accept state?
Answer: q_{rip} cannot be a start or accept state

Last time: GNFA->Regexpr

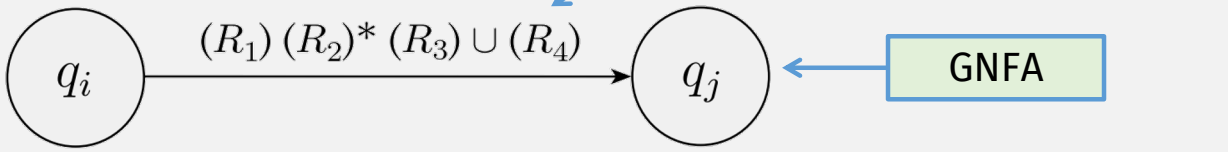
- First modifies input machine to have:
  - New start state
    - With no incoming transitions
    - And epsilon transition to old start state
  - New, single accept state
    - With epsilon transitions from old accept states



Last time: 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"

This is a recursive (inductive) definition!

Last time: Kinds of Mathematical Proof

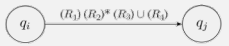
- Proof by construction
- Proof by contradiction
- Proof by induction 
  - Use to prove properties of recursive (inductive) defs or functions
  - Proof steps follow the inductive definition

Last time: Proof by Induction

EXAMPLE OF A "P":
LANGOF (G)
=
LANGOF (GNFA->Regexp (G))

To prove that a **property** P is true for a **thing** x:

1. Prove that P is true for the base case of x (usually easy)



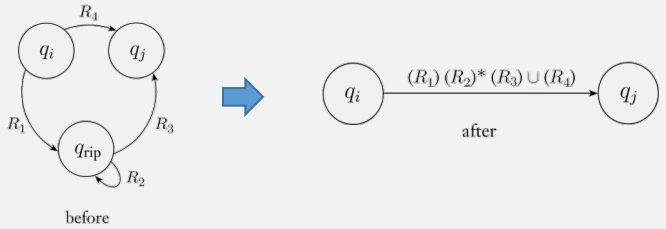
G has two states

2. Prove the induction step:
   - Assume the induction hypothesis (IH):
     - P(x) is true, for some $x_{smaller}$ that is smaller than x

LANGOF (G')
=
LANGOF (GNFA->Regexp (G'))
(Where G' smaller than G)

   - and use it to prove P(x)

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



Regular Expressions, Formal Definition

DEFINITION 1.52

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

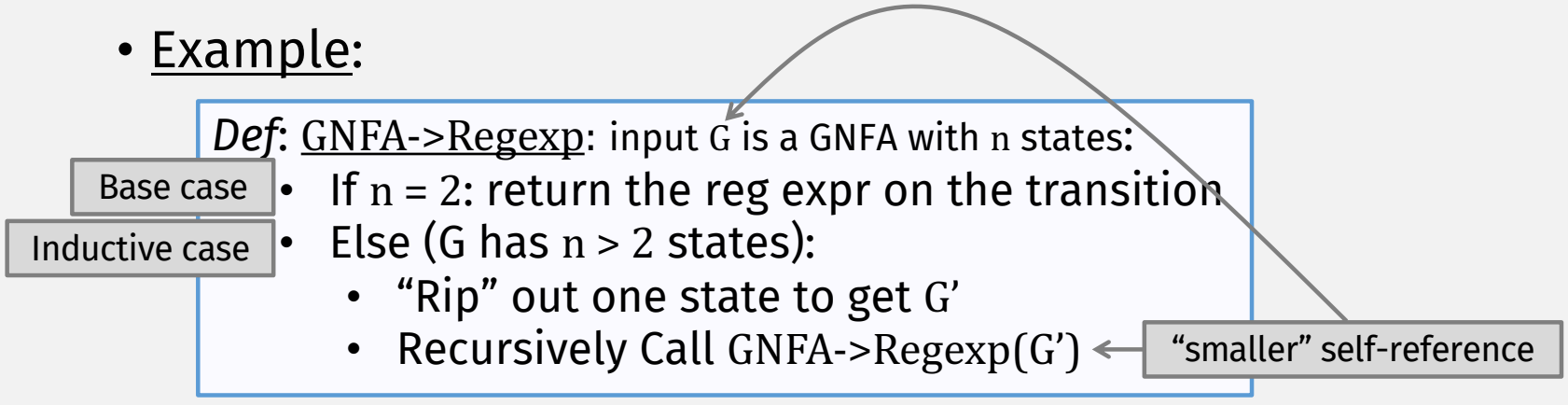
1. a for some a in the alphabet Σ,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is a regular expression.

Is this weird?
Regular expressions defined using regular expressions?

Recursive (Inductive) Definitions

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

- Example:



It's a Recursive Definition!

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

3 base cases

3 inductive cases

"smaller" self-references

How to prove a theorem about Reg ~~Exprs~~? Languages!

- Proof by construction
- Proof by contradiction
- Proof by induction 
  - On Regular Expressions!

How to prove a theorem about Reg ~~Exprs~~? Languages!

We now have 2 proof techniques! You choose

- **Proof by construction** (can still prove things this way)
  - Construct DFA or NFA 
- **Proof by contradiction**
- **Proof by induction**
  - On Regular Expressions!

Exercise:
Are regLangs closed under complement?

- The complement of a set is all members not in the set.

- Complement of lang L are the strings not in L but in the set of strings over the alphabet of L (i.e. $\Sigma^* - L$)

- Proof: You (all) go sketch a construction

RegLangs closed under complement!

- The construction: swap accept and reject states ☺

- You can implement this, and use this fact if it becomes useful.

In class Example: RegLangs closed under homomorphism

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

- Assume f can be used on both strings and characters
- E.g., like a secret decoder!
  - f("x") -> "c"
  - f("y") -> "a"
  - f("z") -> "t"
  - f("xyz") -> "cat"
- Prove: regular languages are closed under
  - E.g., if lang A is regular, then $f(A)$ is regular

How to prove a theorem about Reg ~~Exprs~~? Languages!

We now have 2 proof techniques! You choose

- Proof by construction ⟸
  - Construct DFA or NFA
- Proof by contradiction
- Proof by induction ⟸
  - On Regular Expressions!

Thm: Homomorphism Closed for Reg Langs

- Proof by construction
  - If a lang A is regular, then we know DFA M recognizes it.
  - So modify M such that transitions use the new alphabet
  - (Details left to you to work out)

- Proof by induction:
  - If a lang A is regular, then some reg expression R describes it.

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

Homomorphism Closure: Inductive proof

DEFINITION 1.52

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

1. a for some a in the alphabet Σ,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$
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.

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

3 base cases

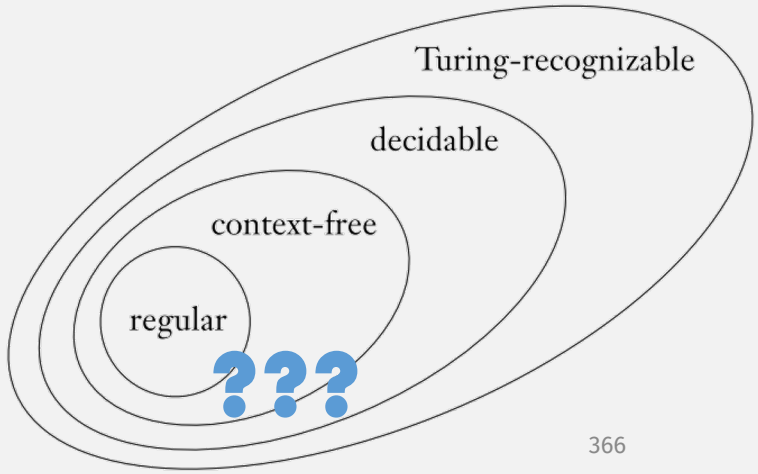
Inductive proof must handle all cases, e.g.,
- If: regexpr "a" describes a reg lang,
- then: f("a") is describes a reg lang
- because: it's still a single-char regexpr,
- so: homomorphism closed under reg langs (for this case)

IH: assume applying homomorphism f to **smaller** R_1 (and R_2) produces a regular lang, i.e., $f(R_1)$ and $f(R_2)$ are regular langs

To finish proof: need to show $f(R_1) \cup f(R_2)$ is a reg lang

(If only union operation were closed for reg langs ☺)

Non-Regular Languages



Non-Regular Languages

- We now have many ways to prove that a language is regular:
  - Construct a DFA or NFA (or GNFA)
  - Come up with a regular expression describing the language

- But how to show that a language is **not regular**?

- E.g., HTML / XML is not a regular language
  - But how can we prove it

- Preview: The Pumping Lemma!


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

Flashback: Designing DFAs or NFAs

- States = the machine's **memory**!
  - Each state "stores" some information
  - Finite states = finite amount of memory
  - And must be allocated in advance

- This means DFAs can't keep track of an arbitrary count!
  - would require infinite states

A Non-Regular Language

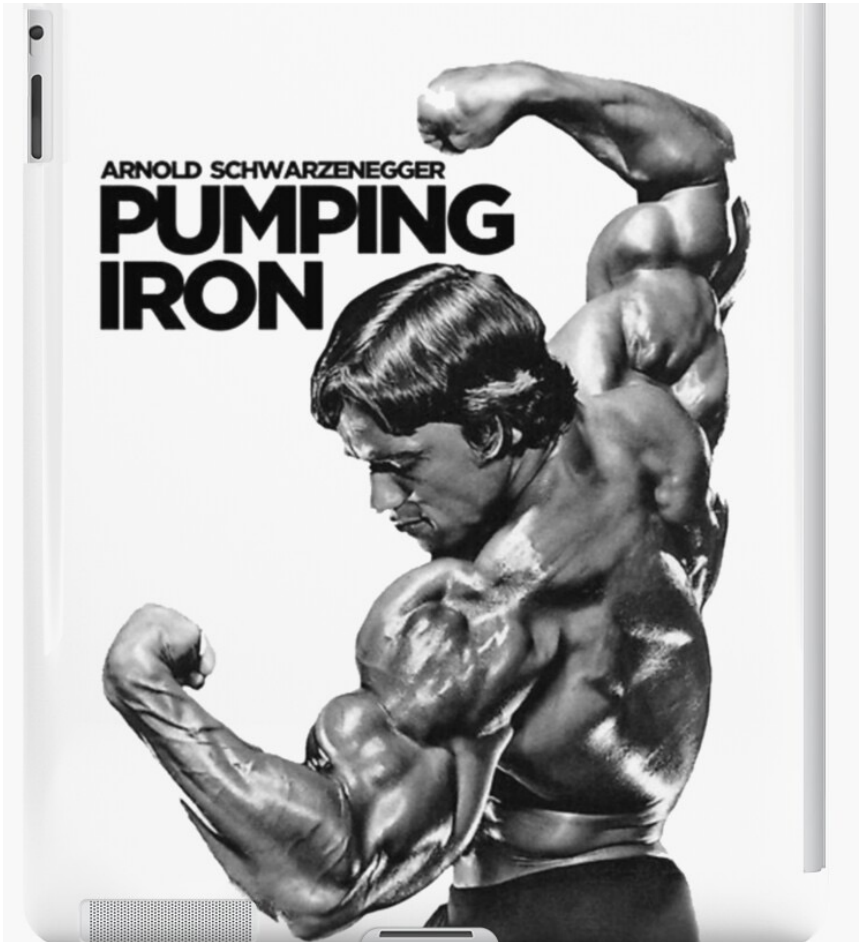
- $L = \{ 0^n 1^n \mid n >= 0 \}$
- A DFA recognizing L would require infinite states! (impossible)
- This language is the essence of XML!
  - To better see this replace:
    - "0" -> "<tag>"
    - "1" -> "</tag>"
- The problem is tracking the **nestedness**
  - Regular languages cannot count arbitrary nesting depths
  - So most programming languages are also not regular!

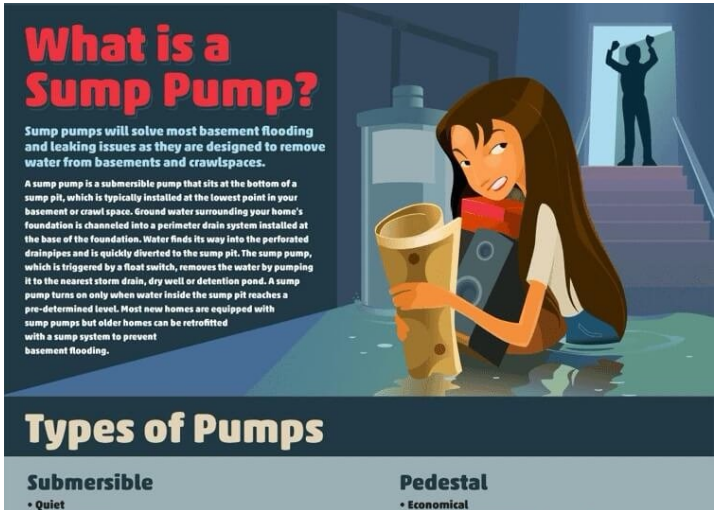
Still, how do we prove non-regularness?

The Pumping Lemma

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. $|xy| \leq p$.

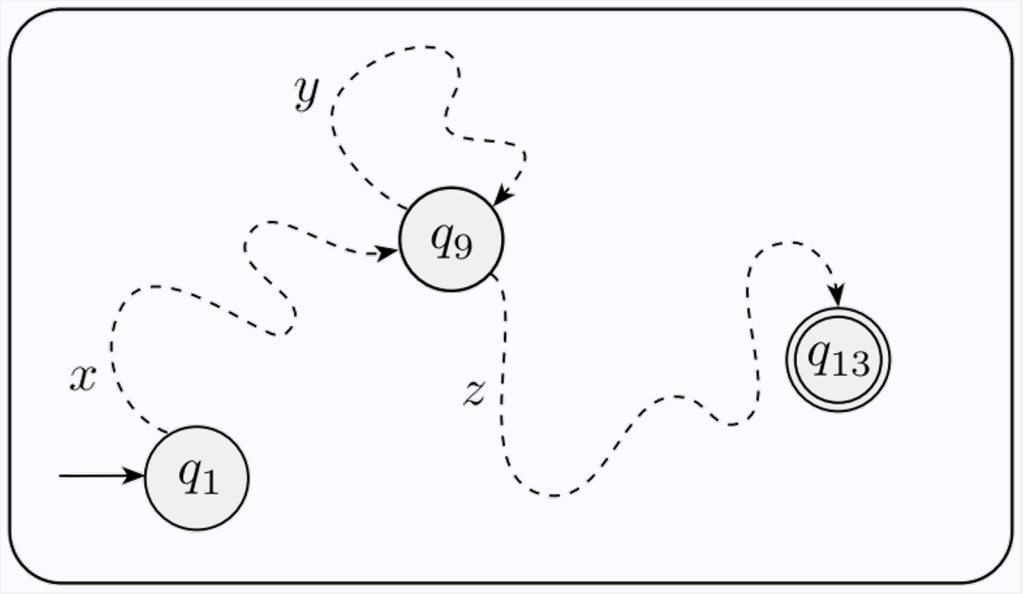

ARNOLD SCHWARZENEGGER
PUMPING
IRON


What is a
Sump Pump?
Sump pumps will solve most basement flooding and leaking issues as they are designed to remove water from basements and crawlspaces.
A sump pump is a submersible pump that sits at the bottom of a sump pit, which is typically installed at the lowest point in your basement or crawl space. Ground water surrounding your home's foundation is channeled into a perimeter drain system installed at the base of the foundation. Water finds its way into the perforated drainpipes and is quickly diverted to the sump pit. The sump pump, which is triggered by a float switch, removes the water by pumping it to the nearest storm drain, dry well or detention pond. A sump pump turns on only when water inside the sump pit reaches a pre-determined level. Most new homes are equipped with sump pumps but older homes can be retrofitted with a sump system to prevent basement flooding.
Types of Pumps
Submersible
• Quiet
Pedestal
• Economical


ARNOLD SCHWARZENEGGER
PUMPING
IRON


basement flooding
designed to remove
crawlspaces.
at sits at the bottom of a
the lowest point in your
basement or crawl space. Ground water surrounding your home's
foundation is channeled into a perimeter drain system installed at
the base of the foundation. Water finds its way into the perforated
drainpipes and is quickly diverted to the sump pit. The sump pump,
which is triggered by a float switch, removes the water by pumping
it to the nearest storm drain, dry well or detention pond. A sump
pump turns on only when water inside the sump pit reaches a
pre-determined level. Most new homes are equipped with
sump pumps but older homes can be retrofitted
with a sump system to prevent
basement flooding.
umps
Pedestal
• Economical

The Pumping Lemma



The Pumping Lemma

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. $|xy| \leq p$.

Pumping lemma specifies three conditions that a regular language must satisfy

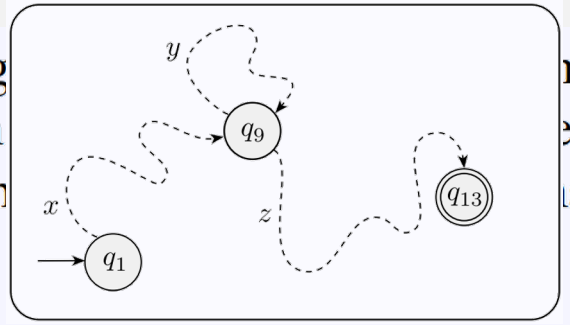
Specifically, strings in the language longer than some length p must satisfy the conditions

But it doesn't tell you an exact p ---you have to find it!

The Pumping Lemma, a Closer Look

Pumping lemma If A is a regular lang nber p (the pumping length) where if s is any string in en s may be divided into three pieces, $s = xyz$, satisfyin s:

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



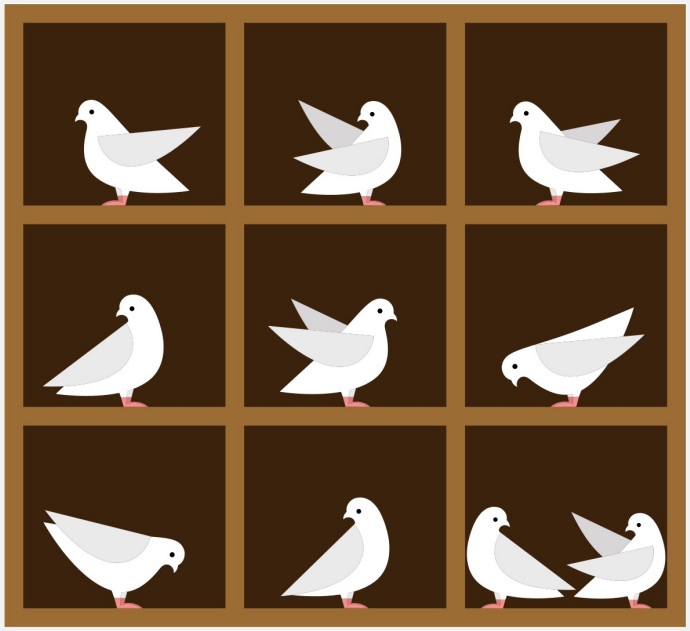
Pumping lemma says that for "long enough" strings, you should be able to repeat a part of it, and that "pumped" string will still be in the language

- Strings that have a repeatable part can be split into:
  - x = the part before any repeating
  - y = the repeated part
  - z = the part after any repeating

This makes sense because DFAs have a finite number of states, so for "long enough" inputs, some state must repeat

The Pigeonhole Principle!

The Pigeonhole Principle



y^i: the *interesting* portion

The "Pumping" Length

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. $|xy| \leq p$.

- What is the pumping length saying? (you must find an appropriate p!)
- Langs that are finite, e.g., {"ab", "cd"} or {} are obviously regular
  - Just choose pumping length > longest string
- Only infinite languages are interesting!
  - Pumping length p >= num states: guarantees repeated states

The Pumping Lemma

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. $|xy| \leq p$.

Because a finite lang is regular, then these conditions must be true for all strings in the lang "of length at least p"

- Example: a finite-sized language, e.g., {"ab", "cd"}
  - All finite langs are regular bc we can easily construct DFA/NFA recognizing them
  - One possible p = length of longest string in the language, plus 1
  - In a finite lang, # strings "of length at least p" = 0
    - Therefore "all" strings "of length at least p" satisfy the pumping lemma criteria!

The Pumping Lemma

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. $|xy| \leq p$.

In an *infinite* regular lang, these conditions must be true for all strings in the lang "of length at least p"

- Example: a *infinite* language, e.g., {"00", "010" , "0110" , "01110", ...}
  - This language is regular bc it's described by regular expression 01*0
  - E.g., "010" is in the lang, and we can split into three parts: x = 0, y = 1, z = 0
    - And any pumping (ie, repeating) of y creates a string that is still in the language
      - E.g., i = 1 -> "010", I = 2 -> "0110", i = 3 -> "01110"
    - This is what the pumping lemma requires

The Pumping Lemma

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. $|xy| \leq p$.

- Example: a *infinite* language, e.g., {"00", "010" , "0110" , "01110", ...}
  - This language is regular bc it's described by regular expression 0*
  - p = ????

The Pumping Lemma

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. $|xy| \leq p$.

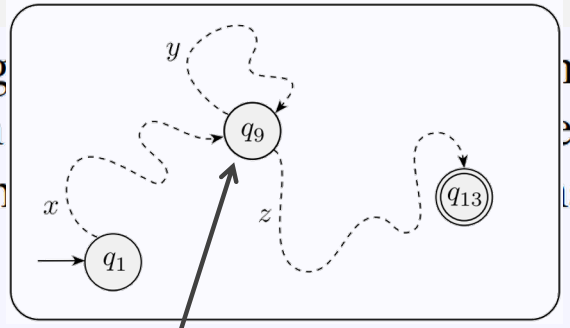
In an *infinite* regular lang, these conditions must be true for all strings in the lang "of length at least p"

- Example: a *infinite* language, e.g., {"00", "010" , "0110" , "01110", ...}
  - This language is regular bc it's described by regular expression 01*0
  - p = ????

The Pumping Lemma

Pumping lemma If A is a regular lang…nber p (the pumping length) where if s is any string in …en s may be divided into three pieces, $s = xyz$, satisfyin…s:

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



"pumpable" part of string

- Example: a *infinite* language, e.g., {"00", "010" , "01…
  - This language is regular bc it's described by regular ex…
  - p = number of states, plus 1
    - When running an input longer than p, one state is guaranteed to be visited twice
    - That state represents the "pumpable" part of the string

But how does this prove that a language is **NOT** regular??

Propositional Equivalence of Material Implications

- $P \rightarrow Q$
- $\neg Q \rightarrow \neg P$

Draw the truth table and verify for yourself!

Kinds of Mathematical Proof

- Proof by construction
  - Construct the object in question
- Proof by contradiction 
  - Proving the contrapositive
- Proof by induction
  - Use to prove properties of recursive definitions or functions

Pumping Lemma: Proving Non-Regularity

… then the language is **not** regular

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. $|xy| \leq p$.

IMPORTANT NOTE:
The pumping lemma **cannot** prove that a language is regular

If any of these are **not** true …

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

Pumping Lemma: Example

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 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 $i \geq 0$, $xy^i z \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

Check-in Quiz

On gradescope