CSE 322, Fall 2010

Nonregular Languages

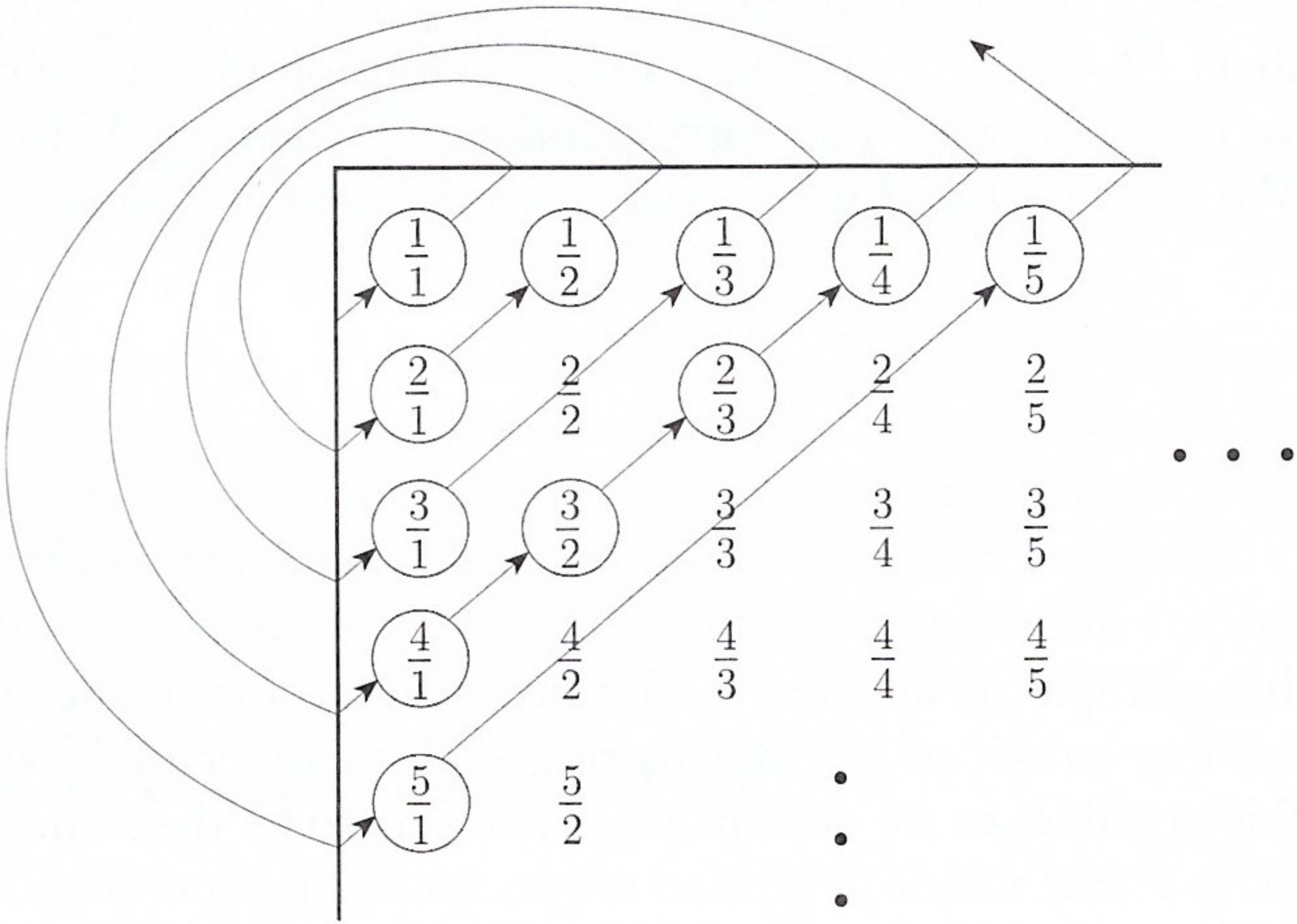
Cardinality

Two sets have equal cardinality if there is a *bijection* ("1-to-1" and "onto" function) between them

A set is *countable* if it is finite or has the same cardinality as the natural numbers

Examples:

- Σ^* is countable (think of strings as base-$|\Sigma|$ numerals)
- Even natural numbers are countable: $f(n) = 2n$
- The Rationals are countable


1/1 1/2 1/3 1/4 1/5
2/1 2/2 2/3 2/4 2/5
3/1 3/2 3/3 3/4 3/5
4/1 4/2 4/3 4/4 4/5
5/1 5/2

More cardinality facts

If f: A → B in an injective function ("1-1", but not necessarily "onto"), then

$|A| \leq |B|$

(Intuitive: f *is* a bijection from A to its *range*, which is a subset of B, & B can't be smaller than a subset of itself.)

Theorem (Cantor-Schroeder-Bernstein):

If $|A| \leq |B|$ and $|B| \leq |A|$ then $|A| = |B|$

The Reals are Uncountable

Suppose they were

List them in order

Define X so that its i^{th} digit $\neq$ i^{th} digit of i^{th} real

Then X is *not in the list*

Contradiction

	int	1	2	3	3	5	
1	0.	0	0	0	0	0	
2	3.	1	4	1	5	9	
3	0.	3	3	3	3	3	
4	0.	5	0	0	0	0	…
5	2.	7	1	8	2	8	
6	41.	9	9	9	9	9	
			⋮				⋱

A detail: avoid .000..., .9999... in X

X	1.	2	4	1	3	8	...

Number of Languages in Σ^* is Uncountable

Suppose they were

List them in order

Define L so that
$w_i \in L \Leftrightarrow w_i \notin L_i$

Then L is *not in the list*

Contradiction

I.e., the powerset of any countable set is uncountable

	w_1	w_2	w_3	w_4	w_5	w_6	
L_1	0	0	0	0	0	0	
L_2	1	1	1	1	1	1	
L_3	0	1	0	1	0	1	
L_4	0	1	0	0	0	0	…
L_5	1	1	1	0	0	0	
L_6	1	1	1	1	0	1	
			⋮				⋱

L	1	0	1	1	1	0	…

Are All Languages Regular?

Σ is finite (for any alphabet Σ)

Σ^* is countably infinite

Let $\Delta = \Sigma \cup \{$“ε”, “$\varnothing$”, “$\cup$”, “$\bullet$”, “$*$”, “(”, “)”$\}$

Δ is finite, so Δ^* is also countably infinite

Every regular lang. R = L(x) for some $x \in \Delta^*$

$\therefore$ the set of regular languages is countable

But the set of all languages over Σ (the powerset of Σ^*) is uncountable

$\therefore$ non-regular languages exist!

(In fact, “most” languages are non-regular.)

The same is true for any real "programming system" I can imagine – programs are finite strings from a finite alphabet, so there are only countably many of them, yet there are uncountably many languages, so there must be some you can't compute...

Above is somewhat unsatisfying – they exist, but what does one "look like"? What's a concrete example?

Next few lectures give specific examples of non-regular languages. *And* proof techniques to show such facts – for such and such a language, *none* of the infinitely many DFAs correctly recognize it.

Some Examples

$\Sigma = \{a, b\}$

$L_1 = \{ x \mid \#_a(x) = \#_b(x) \}$

$L_2 = \{ x \mid \#_{ab}(x) = \#_{ba}(x) \}$

a b b b a b a a b a

L_1 is not regular, L_2 is.

to be shown

$$L_3 = \{ ww \mid w \in \Sigma^* \} \qquad \Sigma = \{a, b\}$$

ε

a a

b b

ab ab

ba ba

aa aa

bb bb

$\vdots$

find middle;
does left = right?

$\left.\begin{array}{l} aba \\ abba \\ \dots \end{array}\right\}$ not in L_3

Intuitively, a DFA accepting L_3 must "remember" the entire left half as it crosses the middle. "Memory" = "states". As $|w| \to \infty$, this will overwhelm any finite memory.

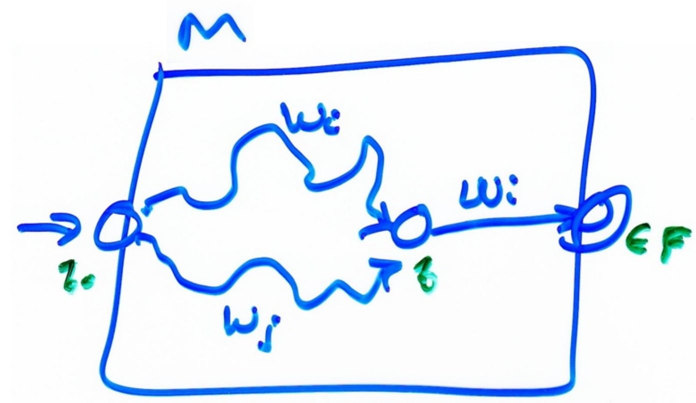
We make this intuition rigorous below...

L_3 is not a Regular Language

Proof: For a DFA $M=(Q,\Sigma,\delta,q_0,F)$, suppose M ends in the same state $q \in Q$ when reading x as it does when reading y, $x \neq y$. Then for any z, either both xz and yz are in L(M) or neither is.

Let $\Sigma=\{a,b\}$, $|Q|=p$, and pick k so that $2^k > p$. Consider all $n=2^k$ length k strings $w_1, w_2, \ldots, w_n$. Consider the set of states M is in after reading each of these strings. By the Pigeon Hole Principle there must be some state $q \in Q$ and some $w_i \neq w_j$ such that both take M to q. But then M must either accept *both* of $w_i w_i$ and $w_j w_i$ or *neither*. In either case, $L(M) \neq L_3$, since one is in L_3, but the other is not.

In pictures:



Since $2^k > p$, list of state names $r_1 \ldots r_{2^k}$ has duplicates, i.e.,

$\exists\, i \neq j$ st $\underbrace{r_i = r_j}_{= q \text{ on prev. slide}}$ (but $w_i \neq w_j$)

$L_3 = \{ ww \mid w \in \{a,b\}^* \}$ is not regular: Alternate Proof

Assume L_3 is regular. Let $M=(Q,\Sigma,\delta,q_0,F)$ be a DFA recognizing L_3. Let $p=|Q|$. Consider the p+1 strings

$x_i = a^i b, 0 \leq i \leq p.$

Again, by the Pigeon Hole Principle, $\exists\ q \in Q$ and $\exists\ 0 \leq i < j \leq p$ s.t. M reaches q from q_0 on *both* x_i & x_j.

Since M accepts both $x_i\, x_i$ and $x_j\, x_j$, it also accepts

$x_j\, x_i = a^j b\, a^i b.$

But j>i, so total length is odd or both b's in right half. Either way, $x_j\, x_i \notin L_3$, a contradiction. Hence L_3 is not regular.

NB: it's true, but not sufficient, to say "$x_i \neq x_j$", since x_j is not the left *half*.

$L_3 = \{ ww \mid w \in \{a,b\}^* \}$ is not regular: Alternate Proof

Assume L_3 is regular. Let $M=(Q,\Sigma,\delta,q_0,F)$ be … recognizing L_3. Let $p=|Q|$. Consider the p+…

$x_i = a^i \cancel{b}$, $0 \le i \le p$

Note importance of "b"; without it, implication falls apart

Again, by the Pigeon Hole Principle, $\exists\, q \in Q$ and $\exists\, 0 \le i < j \le p$ s.t. M reaches q from q_0 on *both* x_i & x_j.

Since M accepts both $x_i\, x_i$ and $x_j\, x_j$, it also accepts

$x_j\, x_i = a^j \cancel{b}\, a^i \cancel{b}$.

But j>i, so … so what? It's all a's, so in L_3 if i+j is even…

~~right half. Either way, $x_j\, x_i \notin L_3$, a contradiction.~~

~~Hence L_3 is not regular.~~

NB: it's true, but not sufficient, to say "$x_i \neq x_j$", since x_j is not the left *half*.

A third way: feed M many a's; eventually it will loop. Say a^i gets to q, then a^j more revisits.

Again, exploit this to reach a (many) contradictions

$a^{i+j} b a^{i+j} b$ ← go around loop once

$a^{i} b a^{i+j} b$ ← zero times

$a^{i+2j} b a^{i+j} b$ ← twice

$a^{i+3j} b a^{i+j} b$ ← 3 times

$\vdots$

$$\boxed{\forall k \geq 0 \quad a^{i+kj} b a^{i+j} b \in L(M)}$$

Notes on these proofs

All versions are proof by contradiction: assume some DFA M accepts L3. M of course has some fixed (but unknown) number of states, p. All versions also relied on the intuition that to accept L3, you need to "remember" the left half of the string when you reach the middle, "memory" = "states", and since every DFA has only a finite number of states, you can force it to "forget" something, i.e., force it into the same state on two different strings. Then a "cut and paste" argument shows that you can replace one string with the other to form another accepted string, proving that M accepts something it shouldn't.

Version 1 (slides 11-12): pick length so there are more such strings than states in M.

Version 2 (slides 13-14): pick increasingly long strings of a simple form until the same thing happens. This argument is a little more subtle, since the string length, hence middle, changes when you do the cut-and-paste, and so you have to argue that *where ever* the middle falls, left half ≠ right half. Some cleverness in picking "long strings of a simple form" makes this possible; in this case the "b" in "a^ib" is a handy marker.

Version 3 (slide 15): Generalizing version 2, accepted strings longer than p always forces M around a loop. Substring defining the loop can be removed or repeated indefinitely, generating many simple variants of the initial string. Carefully choosing the initial string, you can often prove that some variants should be rejected. Again, there is some subtlety in these proofs to allow for any start point/length for the loop.

Not all proofs of non-regularity are about "left half/right half", of course, so the above isn't the whole story, but variations on these themes are widely used. Version 3 is especially versatile, and is the heart of the "pumping lemma", (next few slides).

Those who cannot remember the past are condemned to repeat it.

-- George Santayana (1905) Life of Reason

Corollary

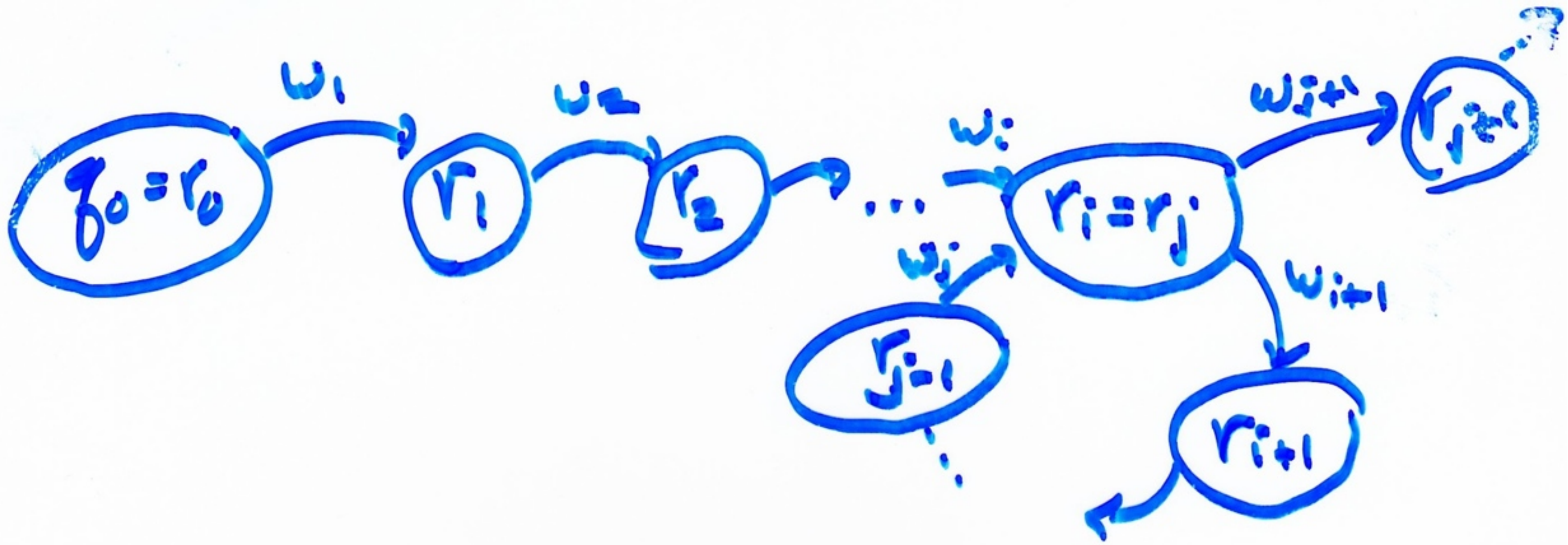
Every sufficiently long input string forces a DFA around a loop.

Proof

Let $p = |Q|$ and $|w| \geq p$.

Let r_i, $0 \leq i \leq |w|$ be state M's in after reading 1st i letter of w.

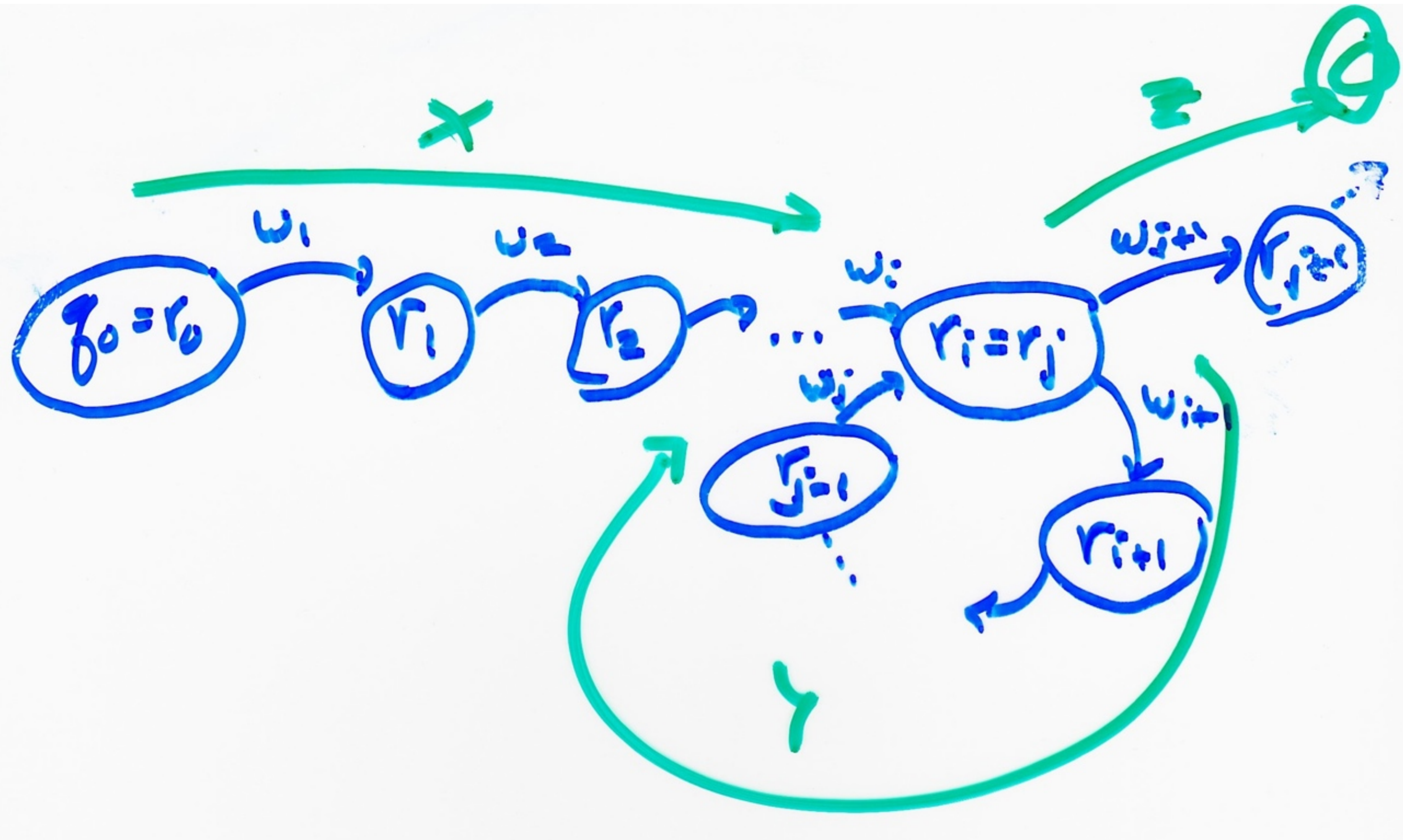
By pigeonhole principle $\exists\, 0 \leq i < j \leq |w|$ st $r_i = r_j$.


w1
w2
wi
wj
wj+1
wi+1
r0 = r0
r1
r2
...
ri = rj
rj-1
ri+1
rj+1

The Pumping Lemma

For all regular languages L, there is an integer $p > 0$ such that any string $w \in L$ with $|w| \geq p$ may be split into three substrings $x, y, z \in \Sigma^*$ so that

1. $w = xyz$
2. $y \neq \epsilon$
3. $|xy| \leq p$, and
4. $\forall i \geq 0, xy^i z \in L$


x
z
w1
w2
wi
wj+1
wj
wi+1
q0 = r0
r1
r2
...
ri = rj
rj+1
rj-1
ri+1
y

Example

$L = \{ a^n b^n \mid n \geq 0 \}$

if L is regular then by P.L.

$\exists p$ st ...

$w = a^p b^p$

$\exists\ x, y, z \in \Sigma^*$

st $xyz = w$

$|y| > 0$

$|xy| \leq p$

$x = a^i$ for some $0 \leq i < p$

$y = a^j$ for some $1 \leq j \leq p - i$

$z = a^{p-i-j} b^p$

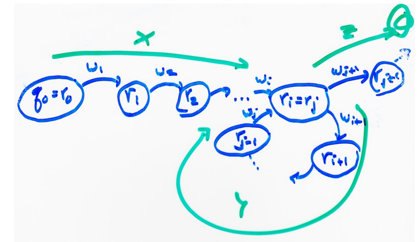
$x y^2 z = a^{p+j} b^p \notin L$

$\therefore$ L is not regular.

The Pumping Lemma

For all regular languages L, there is an integer $p > 0$ such that any string $w \in L$ with $|w| \geq p$ may be split into three substrings $x, y, z \in \Sigma^*$ so that

1. $w = xyz$
2. $y \neq \epsilon$
3. $|xy| \leq p$, and
4. $\forall i \geq 0, xy^iz \in L$



Proof:

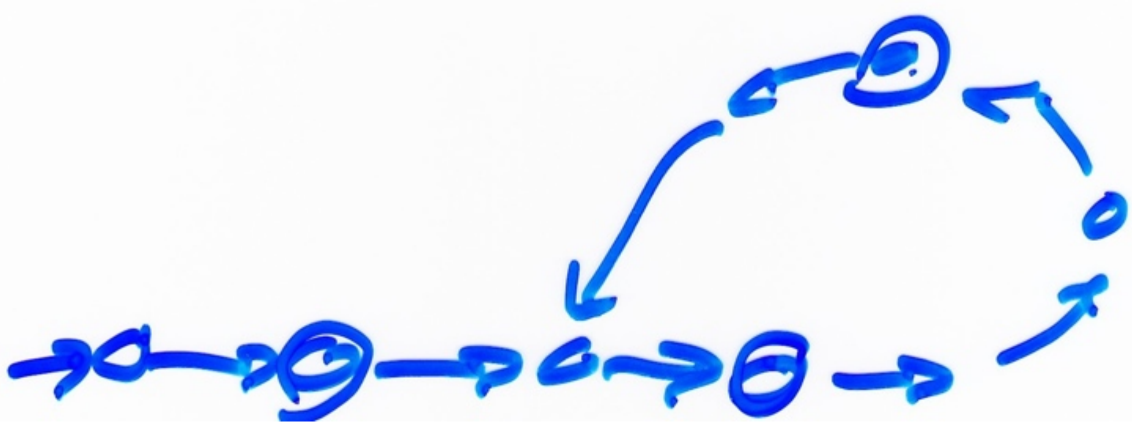
L is regular, so $\exists$ a DFA $M = (Q, \Sigma, \delta, q_0, F)$ such that $L = L(M)$. Let $p = |Q|$. Let w be *any* string in L. If $|w| < p$, the conclusion holds, vacuously. If $|w| \geq p$, let $q_0 = r_0, r_1, \ldots, r_p$ be the sequence of states entered by M after reading the first $0, 1, \ldots, p$ letters of w. There are $p + 1$ entries in that list, but only p states. So, by the Pigeon Hole Principle, $\exists i < j$ such that $r_i = r_j$. Let x be the 1st i letters of w, y be letters $i + 1$ through j, inclusive, and let z be the rest. Since M accepts $w = xyz$ passing through q both immediately before and immediately after y, it also accepts $xz, xyyz, xyyyz, \ldots$ using the q–y–q loop $0, 2, 3, \ldots$ times, resp.

Example

$$L = \{ a^{n^2} \mid n \geq 0 \} \qquad \Sigma = \{ a \}$$

Key Idea: perfect squares become increasingly sparse,
but PL => at most p gap between members

$$0 \underbrace{\quad}_{1} 1 \underbrace{\quad}_{3} 4 \underbrace{\quad}_{5} 9 \underbrace{\quad}_{7} 16 \underbrace{\qquad\qquad}_{9} 25 \underbrace{\qquad\qquad}_{11} 36$$



$L = \{ a^{n^2} \mid n \geq 0 \}$ $\Sigma = \{a\}$

Suppose L is regular. By P.L.

$\exists p$ ... let $w = a^{p^2}$ by P.L.

$\exists xyz$ st $w = xyz$

$0 < |y| \leq p$

Idea: Pick big enough square so that gap to next is larger than the short piece the P.L. repeats

$xy^2z = a^{p^2 + |y|}$

$(p+1)^2 = p^2 + 2p + 1$

$p^2 + |y| \leq p^2 + p < p^2 + 2p + 1$

$\therefore xy^2z \notin L$

Recall

$L = \{ a^n b^n \mid n \geq 0 \}$

if L is regular then by P.L.

$\exists p$ st ...

$w = a^p b^p$

$\exists \; x, y, z \in \Sigma^*$

st $xyz = w$

$|y| > 0$

$|xy| \leq p$

$x = a^i$ for some $0 \leq i < p$

$y = a^j$ for some $1 \leq j \leq p$

$z = a^{p-i-j} b^p$

$xy^2z = a^{p+j} b^p \notin L$

$\therefore$ L is not regular.

$$L = \{ w \mid \#_a(w) = \#_b(w) \}$$

Of course, direct proof via Pumping Lemma is possible.
E.g., a lot like the one for $\{a^n b^n \mid n \geq 0\}$. Alt way:

$$L \cap a^* b^* = \{ a^n b^n \mid n \geq 0 \}$$

regular ? regular not regular

So, by closure of regular languages under intersection, L cannot be regular

Example

$\Sigma = \{ (,) \}$

$L = \{ w \mid$ parens are balanced $\}$

ε, $()$, $()()$, $(()())$

not $)($

if L is regular, so is

$L' = L \cap (^*)^*$

$L' = \{ (^n)^n \mid n \geq 0 \}$

$\equiv \{ a^n b^n \mid n \geq 0 \}$

C – the programming language – satisfies the pumping lemma, but is non-regular

```
main(){return ((((0))));}
```

If C were regular, ∃p∀C programs ∃x,y,z, ...

e.g., x = ε, y = "m" : pumps nicely, giving new func names

But C is not regular

$L = C \cap L(\overline{\texttt{main()\{return(*0)*;\}}}^{\text{regexp}})$

L is not regular: ∃p...

Let w = `main(){return(`^{p}`0)`^{p}`;}`

then if y ∈ (*, i≠1 gives unbalanced parens

y ∉ (*, i≠1 gives an invalid prefix

Similar results possible for C++, Java, Python,...

P.L. suggests all regular languages are infinite!?? Surely false...

Eg. Suppose $L = \{a\}$

PL says $\exists p \; \forall w \in L \; |w| \geq p \Rightarrow \ldots$

Well, take $p=2$. Then, yes indeed for all strings in L of length 2 or greater $\exists xyz \ldots$ is vacuously true, since there are no such strings in L.

Ditto for any finite language —
$p = 1 +$ max length string in L.

Given a regular language L

& a string x how hard is it to decide

- $x \in L$?
- $L = \phi$?
- $L = \Sigma^*$?

A key issue: how is L (in general, an infinite thing) "given" as input to our program? Some options:

1. DFA
2. NFA
3. Reg. Exp.
4. A Java program

⋮

E.g., give as input: # of states, list those in F, size of Σ, a table giving δ(q,a) for each q,a, etc.

Does it matter?

Some Algorithm Qs

Given a string x and a regular language L, how hard is it to decide these questions?

	$x \in L$	$L = \emptyset$	$L = \Sigma^*$
DFA	O(n)	O(n)	O(n)
NFA	(exercise)	O(n)	$O(2^n)$
RegExp	(exercise)	(exercise)	$O(2^n)$
Java Prog	Undecidable – think "halting problem"		
Extended RegExp (¬)	time at least $2^{2^{\cdot^{\cdot^{2}}}}\Big\updownarrow h$, where h > log n		

Some Algorithm Sketches

DFA/$x \in L$: read in DFA, simulate it step by step

DFA/$L=\varnothing$: read DFA, build graph structure; depth-first-search to see if F is reachable from q_0; accept if not.

DFA/$L=\Sigma^*$: apply DFA complement constr; do above

NFA/$L=\varnothing$: like DFA/$L=\varnothing$:

NFA or regexp/$L=\Sigma^*$: *not* like DFA case;
do rexexp $\rightarrow$ NFA, NFA $\rightarrow$ DFA via subset constr

“Extended” Regular Exprs

Regular languages are closed under ops other than ∪, •, *, e.g., ∩, complement, and DROP-OUT. We could add them to regexp syntax and still get only regular languages. E.g.:

$$aa\bullet(\neg((a \cup b)^*(aaa \cup bbb)\ (a \cup b)^*))$$

denotes the strings starting with 2 a’s, followed by a string *not* containing 3 adjacent a’s or b’s. (I think you did something like that in a homework, and it’s kind of a nuisance with plain regexp.)

Why don’t standard RegExp packages support this? The added code is minor: just the closure-under-complement construction.

But the run-time cost is ...

How much can we compute?

Visualize a fast, small computer, say:

  One petaflop (10^{15} ops sec^{-1})

  Femtometer (10^{-15}) in diameter (~ size of a neutron)

Buy a few: say, enough to pack the visible universe

  Radius of visible universe:

   10^{10} light years x π x 10^7 s/year x 3 x 10^8 m/s = 10^{26} m

  Volume: $(10^{26})^3 = 10^{78}$ m^3

  # processors: $10^{78}/(10^{-15})^3 = 10^{123}$ (.1 yotta-googles)

Let it run for a little while, say 10^{10} years

  10^{10} yr x π x 10^7 s/yr x 10^{15} ops/s x 10^{123} processors

= 10^{155} ops since the dawn of time
(somewhat optimistically)

Towers of twos

$2 = 2$

$2^2 = 4$

$2^{2^2} = 2^4 = 16$

$2^{2^{2^2}} = 2^{16} = 65536$

$2^{2^{2^{2^2}}} \approx 10^{19728}$

Summary

There are (many) non-regular languages

Famous examples: $\{a^nb^n|n>0\}$, $\{\#_a = \#_b\}$, {ww}, {C}, {Java}

Famous ways to prove:

- Diagonalization
- M in same state on 2 strings it should distinguish
- One stylized way: Pumping Lemma
- Closure Properties

Simple algorithmic problems can be astronomically slow