Where to Go from Here

Announcements

- Problem Set 9 was due thirty minutes ago.
  - You can use a late day to extend the deadline to tomorrow at 4:00PM.
  - Solutions will go online tomorrow at 4:00PM.

Congratulations – you're done with CS103 problem sets!

- Take a minute to reflect on how much you've learned! Look back at PS1. Those problems seem a *lot* easier now, don't they?

A Fun Historical Note

- The results you've seen presented in CS103 were not discovered in the order you may have expected.
- For example:
  - Regular languages were developed after Turing machines.
  - Cantor had worked out different orders of infinity before the ∪ and ∩ symbols were invented.
- Check out the "Timeline of CS103 Results" on the course website for more information!

Please evaluate this course on Axess.

Your feedback really makes a difference.

Final Exam Logistics

- Our final exam is on ***Monday, March 18th*** from ***7:00PM – 10:00PM***. It'll be held in ***Hewlett 200/Hewlett 201***.
- The final exam is cumulative and covers topics from PS1 – PS9 and L00 – L24. Coverage will focus on material from the second half of the course which you haven't been tested on yet.
- The format is similar to that of the midterm, with a mix of short-answer questions and formal written proofs.
- Like the midterms, it's closed-book, closed-computer, and limited-note. You can bring one double-sided 8.5" × 11" notes sheet with you.
- ***Best of luck – you can do this!***

Outline for Today

- ***The Big Picture***
  - Where have we been? Why did it all matter?
- ***Where to Go from Here***
  - What's next in CS theory?
- ***Your Questions***
  - What do you want to know?
- ***Final Thoughts!***

Take a minute to reflect on your journey.

Set Theory
Power Sets
Cantor’s Theorem
Direct Proofs
Parity
Proof by Contrapositive
Proof by Contradiction
Modular Congruence
Propositional Logic
First-Order Logic
Logic Translations
Logical Negations
Propositional Completeness
Vacuous Truths
Tournaments
Functions
Injections
Surjections
Involutions
Monotone Functions
Bijections
Cardinality
Graphs
Connectivity
Independent Sets
Vertex Covers
Graph Complements
Dominating Sets
Bipartite Graphs
The Pigeonhole Principle
Mathematical Induction
Loop Invariants
Complete Induction
Formal Languages
DFAs
Regular Languages
Closure Properties
NFAs
Subset Construction
Kleene Closures
Regular Expressions
State Elimination
Distinguishability
Myhill-Nerode Theorem
Nonregular Languages
Context-Free Grammars
Turing Machines
Church-Turing Thesis
TM Encodings
Universal Turing Machines
Self-Reference
Decidability
Recognizability
Self-Defeating Objects
Undecidable Problems
The Halting Problem
Verifiers
Diagonalization Language
Complexity Class **P**
Complexity Class **NP**
P $\stackrel{?}{=}$ **NP** Problem
Polynomial-Time Reducibility
NP-Completeness

You’ve done more than just check
a bunch of boxes off a list.

You’ve given yourself the foundation to tackle problems from all over computer science.

From CS255

A **Shannon cipher** is a pair $\mathcal{E} = (E, D)$ of functions.

- The function E (the **encryption function**) takes as input a **key** k and a **message** m (also called a **plaintext**), and produces as output a **ciphertext** c. That is,

  $$c = E(k, m),$$

  and we say that c is the **encryption of** m **under** k.

- The function D (the **decryption function**) takes as input a key k and a ciphertext c, and produces a message m. That is,

  $$m = D(k, c),$$

  and we say that m is the **decryption of** c **under** k.

- We require that decryption "undoes" encryption; that is, the cipher [illegible] **correctness property**: for all keys k and all messages m, we have

  $$D(k,\ E(k,\ m)\ ) = m.$$

Kinda sorta like a left inverse!

To be slightly more formal, let us assume that $\mathcal{K}$ is the set of all keys (the **key space**), $\mathcal{M}$ is the set of all messages (the **message space**), and that $\mathcal{C}$ is the set of all ciphertexts (the **ciphertext space**). With this notation, we can write:

$$E : \mathcal{K} \times \mathcal{M} \to \mathcal{C},$$
$$D : \mathcal{K} \times \mathcal{C} \to \mathcal{M}.$$

Also, we shall say that $\mathcal{E}$ is **defined over** $(\mathcal{K}, \mathcal{M}, \mathcal{C})$.

Strong triadic closure

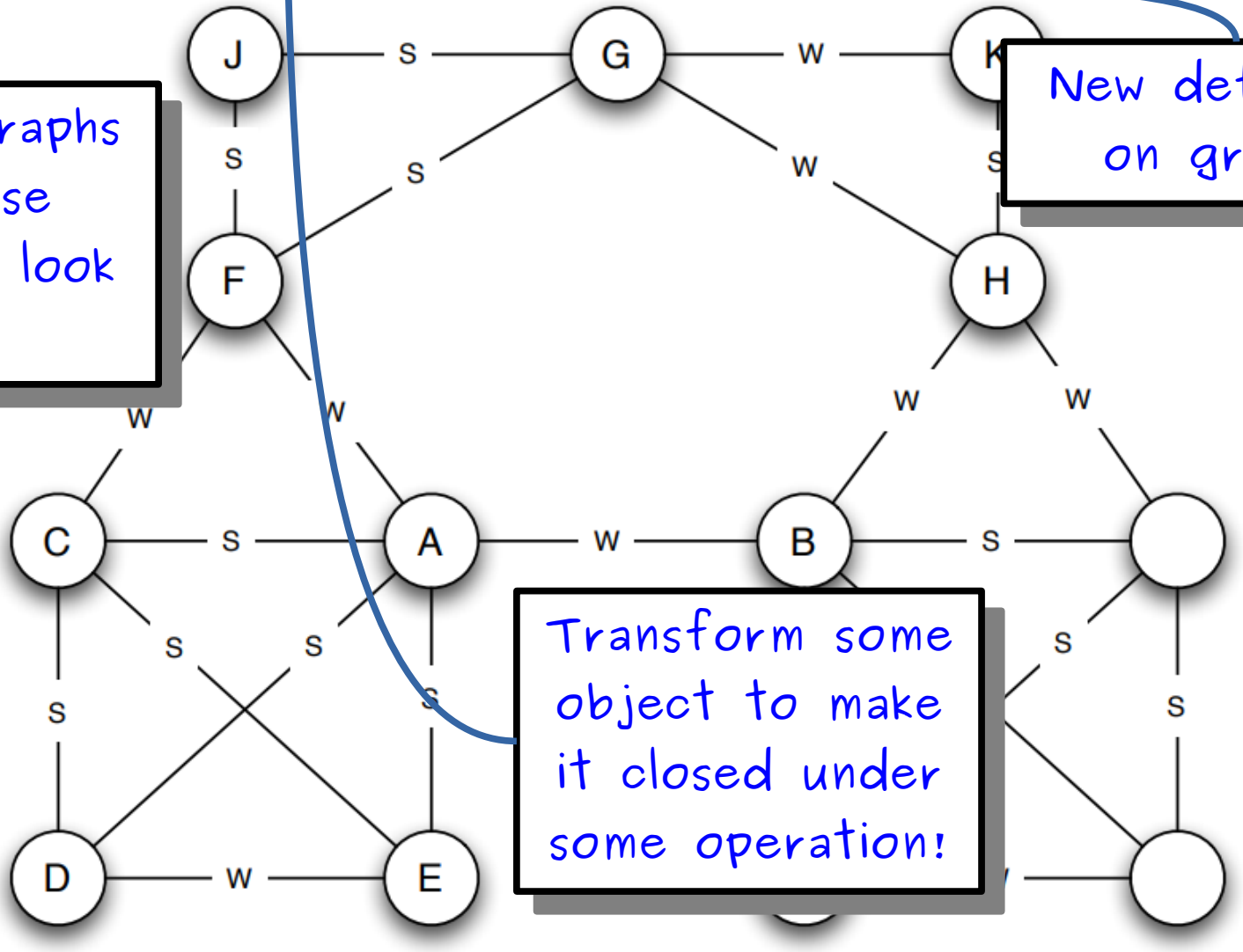
From CS124

If a node Q has two strong ties to nodes Y and Z, there is an edge between Y and Z

What do graphs with these properties look like?

New definitions on graphs!

Transform some object to make it closed under some operation!



From CS124

Tokenization in NLTK

Bird, Loper and Klein (2009), *Natural Language Processing with Python*. O'Reilly

```
>>> text = 'That U.S.A. poster-print costs $12.40...'
>>> pattern = r'''(?x)    # set flag to allow verbose regexps
...     ([A-Z]\.)+        # abbreviations, e.g. U.S.A.
...   | \w+(-\w+)*        # words with optional internal hyphens
...   | \$?\d+(\.\d+)?%?  # currency and percentages, e.g. $12.40, 82%
...   | \.\.\.            # ellipsis
...   | [][.,;"'?():-_`]  # these are separate tokens; includes ], [
... '''
>>> nltk.regexp_tokenize(text, pattern)
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

It's a big regex!

Describing the world in set theory!

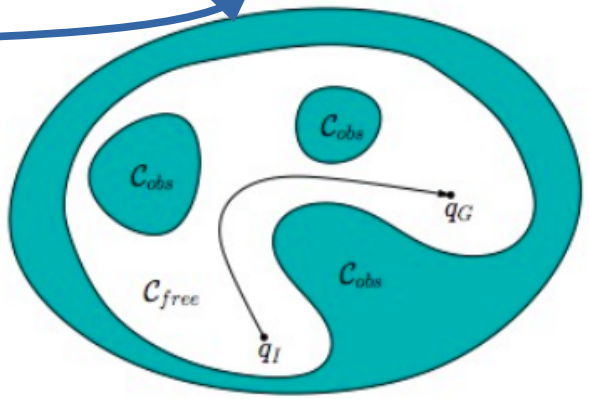
From CS237A

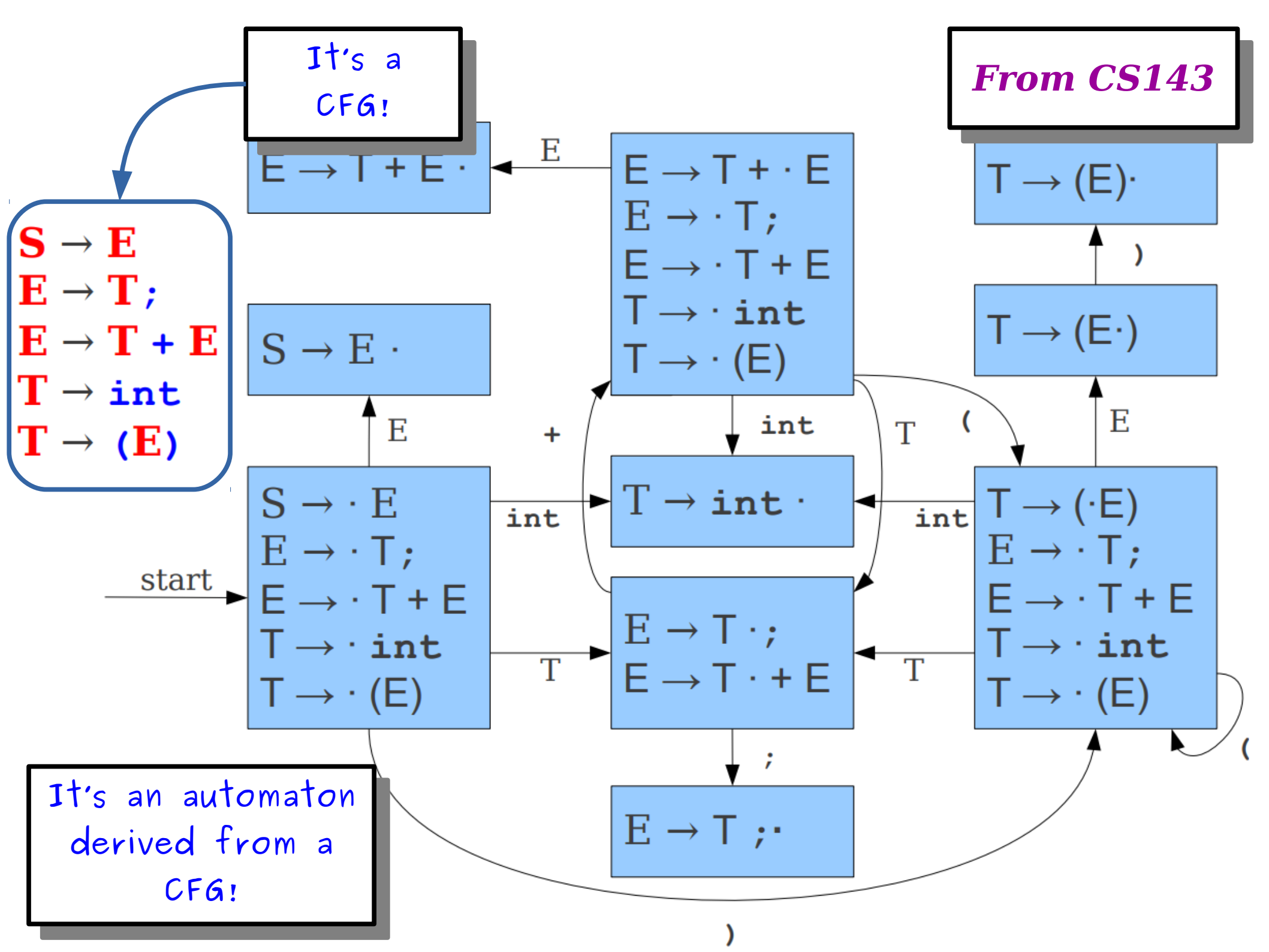
Plan[illegible]space

- Let $R(q) \subset W$ denote set of points in the world occupied by robot when in configuration q
- Robot in collision $\Leftrightarrow R(q) \cap O \neq \emptyset$
- Accordingly, *free* space is defined as: $C_{free} = \{q \in C | R(q) \cap O = \emptyset\}$

- Path planning problem in C-space: compute a **continuous** path: $\tau: [0,1] \rightarrow C_{free}$, with $\tau(0) = q_I$ and $\tau(1) = q_G$

Model paths as functions!




It's a CFG!
From CS143
S → E
E → T;
E → T + E
T → int
T → (E)
E → T + E ·
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)
T → (E)·
T → (E·)
S → E ·
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)
T → int ·
E → T ·;
E → T · + E
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)
E → T ;·
start
E
+
int
T
(
)
;
It's an automaton derived from a CFG!

Search problems

From CS221

Definition: search problem

States: the set of states

$s_{\text{start}} \in \text{States}$: starting state

$\text{Actions}(s)$: possible actions from state s

$\text{Succ}(s, a)$: where we end up if take action a in state s

$\text{Cost}(s, a)$: cost for taking action a in state s

$\text{IsEnd}(s)$: whether at end

- $\text{Succ}(s, a) \Rightarrow T(s, a, s')$
- $\text{Cost}(s, a) \Rightarrow \text{Reward}(s, a, s')$

It's a DFA!

pronounced “big-oh of …” or sometimes “oh of …”

From CS161

O(…) means an upper bound

- Let T(n), g(n) be functions of positive integers.
  - Think of T(n) as being a runtime: positive and increasing in n.

- We say “T(n) is O(g(n))” if g(n) grows at least as fast as T(n) as n gets large.

- Formally,

$$T(n) = O(g(n))$$
$$\Leftrightarrow$$
$$\exists c, n_0 > 0 \;\; s.t. \;\; \forall n \geq n_0,$$
$$0 \leq T(n) \leq c \cdot g(n)$$

It’s FOL and functions!

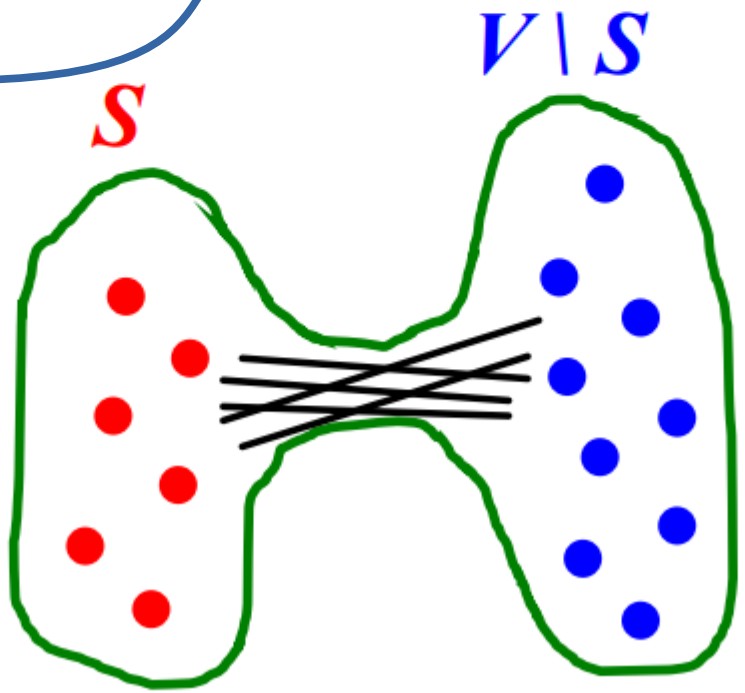
From CS224W

- Graph $G(V, E)$ has **expansion α**: if $\forall S \subseteq V$:
  $\#\text{ of edges leaving } S \geq \alpha \cdot \min(|S|, |V \setminus S|)$
- **Or equivalently:**

$$\alpha = \min_{S \subseteq V} \frac{\#\,edges\ leaving\ S}{\min(|S|, |V \setminus S|)}$$

Set difference and cardinality!

First-order definitions on graphs!



From CS242

Typed lambda calculus

To understand the formal concept of a type system, we're going to extend our lambda calculus from last week (henceforth the "untyped" lambda calculus) with a notion of types (the "simply typed" lambda calculus). Here's the essentials of the language:

Type τ ::=	int	integer
|	$\tau_1 \to \tau_2$	function
Expression e ::=	x	variable
|	n	integer
|	$e_1 \oplus e_2$	binary operation
|	$\lambda\ (x : \tau)\ .\ e$	function
|	$e_1\ e_2$	application
Binop $\oplus$ ::=	$+ \mid - \mid * \mid /$	

It's a CFG!

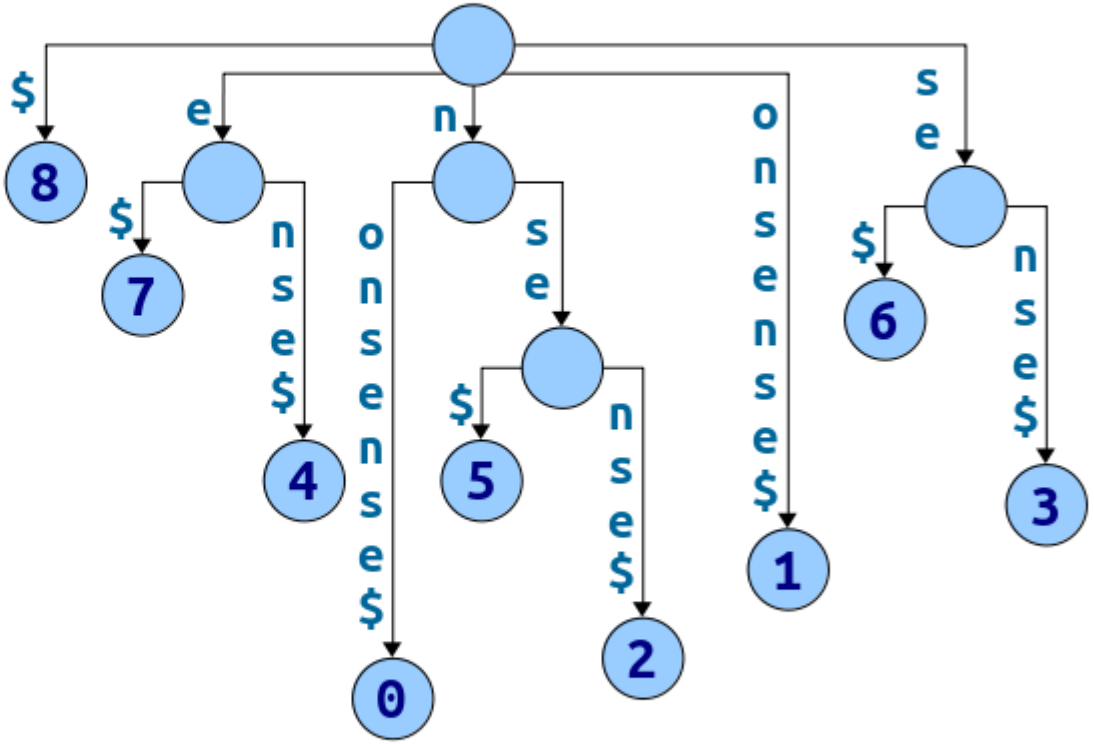
First, we introduce a language of types, indicated by the variable tau (τ). A type is either an integer, or a function from an input type τ_1 to an output type τ_2. Then we extend our untyped lambda calculus with the same arithmetic language from the first lecture (numbers and binary operators)[4]. Usage of the language looks similar to before:

Definitions in terms of strings!

From CS166

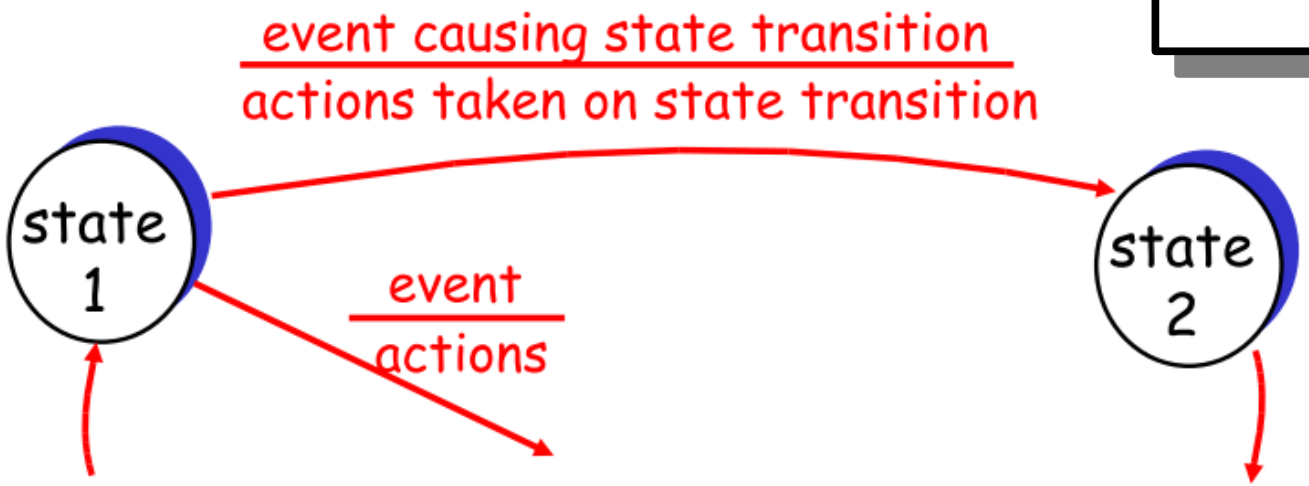
The Anatomy of a Suffix Tree

- A ***branching word*** in $T\$$ is a string ω such that there are characters $a \neq b$ where ωa and ωb are substrings of $T\$$.
  - Edge case: the empty string is always considered branching.
- ***Theorem:*** The suffix tree for a string T has an internal node for a string ω if and only if ω is a branching word in $T\$$.



Finite State Machines

From CS144



- **Represent protocols using state machines**
  - Sender and receiver each have a state
  - Start in some initial state
  - Events cause each side to select a state

It's a generalization of DFAs!

- **Transition specifies action taken**
  - Specified as events/actions
  - E.g., software calls `send`/put packet on network
  - E.g., packet arrives/send acknowledgment

From CS168

Reducibility!

By definition, we need to output y if and only if $y \in S$. That is, *answering membership queries reduces to solving the Heavy Hitters problem.* By the "membership problem," we mean the task of preprocessing a set S to answer queries of the form "is $y \in S$"? (A hash table is the most common solution to this problem.) It is intuitive that you cannot correctly answer all membership queries for a set S without storing S (thereby using linear, rather than constant, space) — if you throw some of S out, you might get a query asking about the part you threw out, and you won't know the answer. It's not too hard to make this idea precise using the Pigeonhole Principle.[5]

A Myhill–Nerode–style argument!

From CS154

Kolmogorov Complexity (1960's)

Definition: The *shortest description of* x, denoted as $d(x)$, is the lexicographically shortest string $\langle M,w \rangle$ such that $M(w)$ halts with only x on its tape.

Definition: The *Kolmogorov complexity of* x, denoted as $K(x)$, is $|d(x)|$.

Using Turing machines to define intrinsic information content!

From CS250

Alphabets!

(4) FORMAL DEFINITIONS

Let Σ be any finite set and let $n > 0$ be an integer.

DEF. A CODE C of BLOCK LENGTH n over an ALPHABET Σ is a subset $C \subseteq \Sigma^n$.
An element $c \in C$ is called a CODEWORD.

Sometimes I will say "length" instead of "block length."

Languages!

You’ve given yourself the foundation to tackle problems from all over computer science.

There’s so much more to explore.
Where should you go next?

Course Recommendations

Theoryland

- CS154 — *Complexity*
- Phil 151 — *Computability*
- Phil 152 — *Computability*
- Math 107 — *Graphs*
- Math 108 — *Graphs*
- Math 113 — *Functions*
- Math 120 — *Functions*
- Math 161 — *Set Theory*
- Math 152 — *Number Theory*

Applications

- CS124 — *Languages / Automata*, *Graphs*, *Functions*
- CS143 — *Languages / Automata*
- CS161 — *Graphs*
- CS224W — *Functions*
- CS242 — *Functions*
- CS243 — *Graphs*, *Functions*
- CS246 — *Functions*
- CS251 — *Functions*
- CS255 — *Functions*

Your Questions

“What's an underrated spot on campus to study?”

Here are some of my favorites:
1) The second story terrace at the Law School (go up the spiral staircase outside)
2) McMurtry roof, and the McMurtry Library
3) Bender Room at Green Library

“What's something I should be sure to do before I graduate?”

- Get to know your professors.
- Chat with the campus staff who clean your dorms, serve your food, and maintain the campus.
- Take that “just for fun” class you always wanted to try.
- Go on a road trip (or other spontaneous adventure) with friends.
- Dip your toes into research (either conducting it yourself, participating in studies, or even just talking to professors).
- Explore the campus beyond your daily routine. Some recommendations: Cantor/the Anderson Collection, the 3D listening room at CCRMA, this map of fruit trees on campus, the whispering circle near MemChu, the O’Donohue family farm.

“What were some struggles you faced when completing your BS CS degree at Stanford and how did you overcome them?”

In much of my time in college, I struggled with equating my self worth with academic success. Over time, I worked to actively change my mindset and view my academic performance as a snapshot of my current state of knowledge rather than a statement of what I could achieve in the future. I also worked on giving myself permission to try things and fail at them.

Specific to CS, I also struggled with feeling like I was “behind” my peers because I didn’t commit to CS until roughly end of my sophomore year. I got rejected from many things (internships, section leading) and I took a lot of that personally. Something that dramatically changed my perspective on this was when I started TA-ing for this class and got to see things from the other side.

“Would you say that you're more of a theory person when it comes to CS? If so, how did you come to that conclusion?”

In some senses, yes: I find myself particularly drawn to this notion of truth, and the idea that we can know something with absolute certainty. I also love that there is space in the theory realm to study things that might not have immediate practical value, but are worthy of our attention due to their inherent beauty.

In other senses, no: I have broad interests outside of theory, and limiting myself to being just a theory person ignores the other parts of computer science that I am passionate about.

"If you could change one thing about the 103 curriculum, what would it be? what would this ideal version of 103 look like to you?"

In my ideal world, CS103 would be a two quarter course, where the first quarter is focused solely on discrete math and the second quarter is computability/complexity theory.

Another thought I have is that in a lot of math education, most of the learning happens when we are struggling with concepts, trying things, and making mistakes, and yet in most classrooms we give you the kind of work you're meant to get correct. This feels backwards to me, and I am interested in seeing if there is a scalable way to operationalize this model.

"What's the hardest topic for you to teach in 103? Also what is your favorite to teach? When you took the class was your favorite topic different?"

Hardest to teach: this isn't exactly a "topic" in the traditional sense, but one thing I am working on is how to teach mathematical confidence, or this idea that you (yes you!) can be successful in and find a place for yourself in mathematics.

Favorite to teach: so many good choices here! Cantor's theorem is my favorite proof, and I'm also a huge fan of the pigeonhole principle.

As a student, I was completely blown away by computability theory. How cool is it that we get to make these little models and hold computation in our hands, and to ponder these big, abstract, philosophical ideas like the nature of truth and infinity?

“What do you think is the point of taking theory classes when I don't think I will be using this in the workforce? How many people at Linkedin ask you if a language is regular on a yearly basis?

What is the big WHY of this course for people who won't use it later?

How/where do you feel your CS103 knowledge being useful for your work in industry?

Sure! I'll touch on three things:

1) Direct applications

2) Problem solving skills

3) The intrinsic value of learning math

How do you identify what skills are in-demand by region?

Who in my network can connect me to a job?

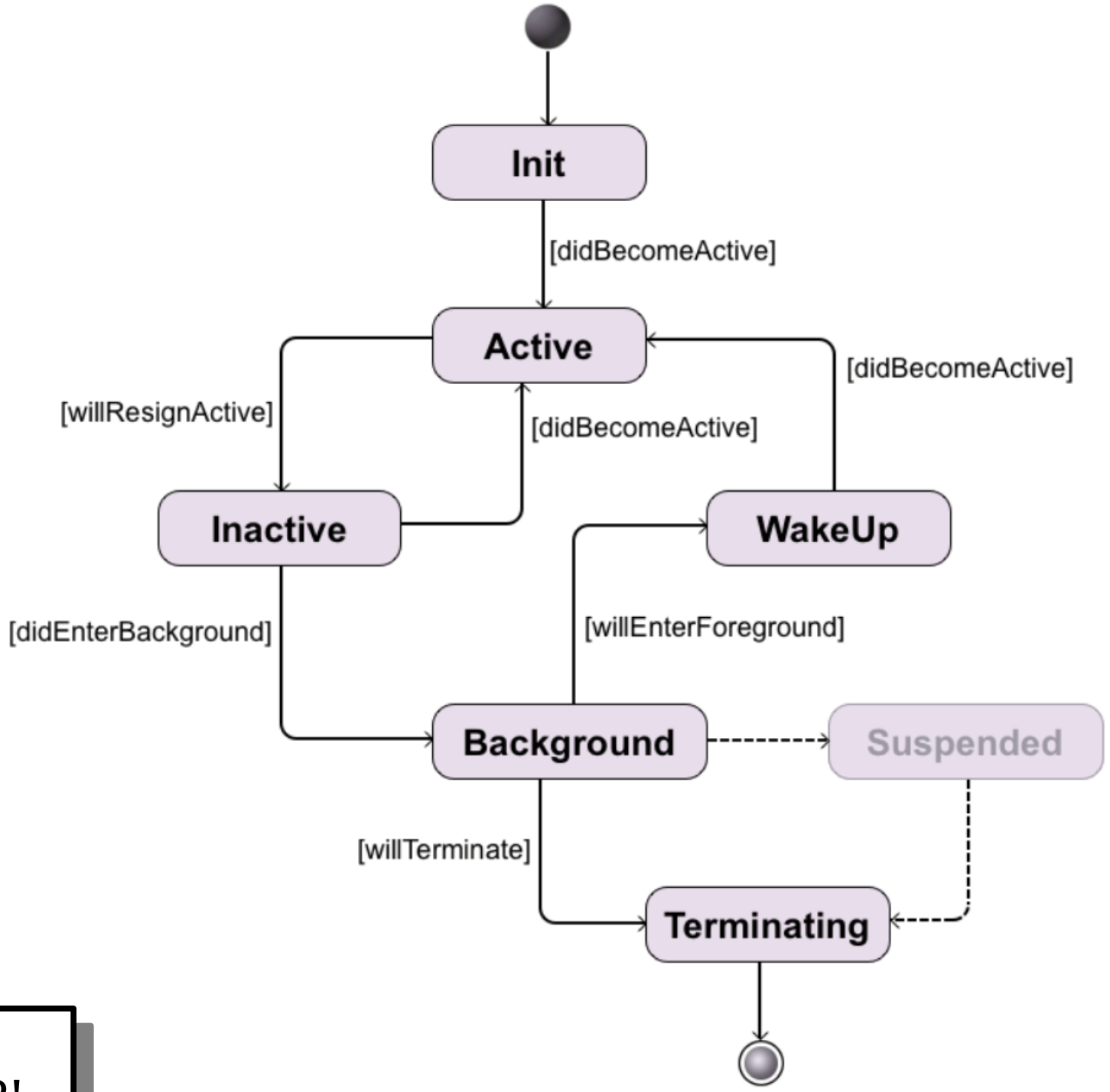
How do we efficiently determine how many connections two people have in common?

Graph theory in practice!

Buttons as Finite-State Machines:

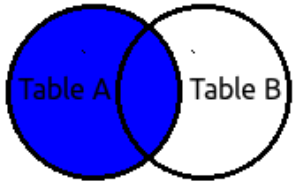
http://cs103.stanford.edu/tools/button-fsm/

Take
CS148!

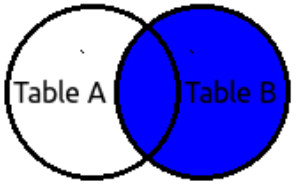


Take CS193P!

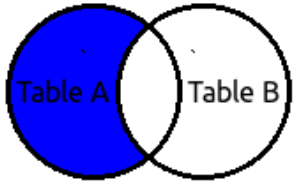
https://blog.codecentric.de/en/2016/07/handling-ios-app-states-state-machine/



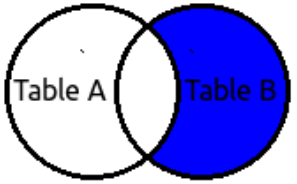

```
SELECT [list] FROM
    [Table A] A
LEFT JOIN
    [Table B] B
ON A.Value = B.Value
```



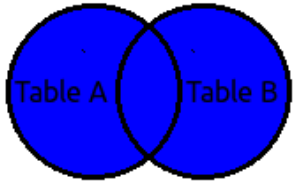

```
SELECT [list] FROM
    [Table A] A
RIGHT JOIN
    [Table B] B
ON A.Value = B.Value
```



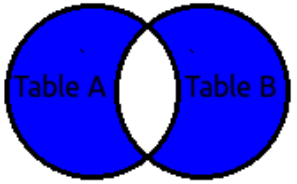

```
SELECT [list] FROM
    [Table A] A
LEFT JOIN
    [Table B] B
ON A.Value = B.Value
WHERE B.Value IS NULL
```



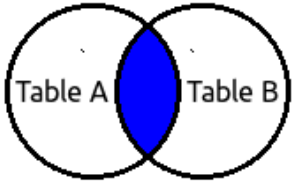

```
SELECT [list] FROM
    [Table A] A
RIGHT JOIN
    [Table B] B
ON A.Value = B.Value
WHERE A.Value IS NULL
```




```
SELECT [list] FROM
    [Table A] A
FULL OUTER JOIN
    [Table B] B
ON A.Value = B.Value
```




```
SELECT [list] FROM
    [Table A] A
FULL OUTER JOIN
    [Table B] B
ON A.Value = B.Value
WHERE A.Value IS NULL
OR B.Value IS NULL
```




```
SELECT [list] FROM
    [Table A] A
INNER JOIN
    [Table B] B
ON A.Value = B.Value
```

SQL (language used to query a database) is essentially not much more than set operations!

https://stackoverflow.com/questions/406294/left-join-vs-left-outer-join-in-sql-server

Aside from the content, CS103 teaches you problem solving skills. Just some of the things that proofwriting has taught me: how to communicate ideas clearly and precisely in writing, how to take a complex problem and break it down into logical steps using a given toolset, and how to debug and identify edge cases in an argument/procedure. I find myself using these skills literally every day.

While studying math has practical applications, it has *intrinsic* value in teaching you how to be curious, to wonder, to try things and struggle, to delight in beautiful results, and to hope. When we practice math, we build these virtues within ourselves that help us flourish towards having a fuller and well-lived life.

Check out this talk by Francis Su on "Mathematics for Human Flourishing".

Final Thoughts

A Huge Round of Thanks!

There are more problems to solve than there are programs capable of solving them.

There is so much more to explore and so many big questions to ask – ***many of which haven't been asked yet!***

You now know what problems we can solve, what problems we can't solve, and what problems we believe we can't solve efficiently.

Our questions to you:

What problems will you ***choose*** to solve?
Why do those problems matter to you?
And how are you going to solve them?