

Where to Go from Here

# Announcements

- Problem Set 9 was due thirty minutes ago. Solutions are available on the course website.

***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  $\cup$  and  $\cap$  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 ***Tuesday, December 13<sup>th</sup>*** from ***8:30AM - 11:30AM***. It'll be held in ***Hewlett 200***.
- The final exam is cumulative and covers topics from PS1 - PS9 and L00 - L27. 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!***

# Preparing for the Final Exam

- We've posted a gigantic compendium of CS103 practice problems on the course website.
- You can search for problems based on the topics they cover, whether solutions are available, whether they're ones we particularly like, and whether they were used on past exams.
- As always, ***keep the TAs in the loop!*** Ask us questions if you have them, feel free to stop by office hours to discuss solutions, etc.

# Review Session

- Our fantastic TA Ophir will be holding a final exam review session this ***Sunday, December 11<sup>th</sup>*** starting at ***2:00PM*** in ***300-300***.
- Feel free to stop on by!
- SCPD students: we'll have a Zoom link you can use to join, though it'll be a little more low-tech than our standard lectures.

# 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!***

# The Big Picture

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  
Perfect Squares  
Tournaments  
Functions  
Injections  
Surjections  
Involutions  
Monotone Functions  
Idempotent Functions  
Bijections

Cardinality  
Graphs  
Connectivity  
Independent Sets  
Vertex Covers  
Graph Complements  
Dominating Sets  
Bipartite Graphs  
The Pigeonhole Principle  
Ramsey Theory  
Mathematical Induction  
Loop Invariants  
Complete Induction  
Formal Languages  
DFAs  
Regular Languages  
Closure Properties  
NFAs  
Subset Construction  
Kleene Closures  
Hard Reset Strings  
Regular Expressions  
State Elimination  
Monoids

Distinguishability  
Myhill-Nerode Theorem  
Nonregular Languages  
Context-Free Grammars  
Brzowski's Theorem  
Fixed Point Theorems  
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  
**R**, **RE**, and co-**RE**  
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.

# PRPs and PRFs

*From CS255*

- Pseudo Random Function (**PRF**) defined over  $(K, X, Y)$ :

$$F: K \times X \rightarrow Y$$

such that exists “efficient” algorithm to evaluate  $F(k, x)$

- Pseudo Random Permutation (**PRP**)

$$E: K \times X \rightarrow X$$

such that:

1. Exists “efficient” algorithm to evaluate  $E(k, x)$

Definitions in terms of efficiency!

2. The function  $E(k, \cdot)$  is one-to-one

“efficient” inversion algorithm  $D(k, y)$

Injectivity!

Functions between sets!  $K \times X$  is the set of all pairs made from  $K$  and  $X$ .

## Semantics of JOINS (2 tables)

Formalizing things with sets!

```
SELECT R.A  
FROM R, S  
WHERE R.A = S.B
```

### 1. Take **cross product**:

$$X = R \times S$$

Recall: Cross product ( $A \times B$ ) is the set of all unique tuples in  $A, B$

Ex:  $\{a, b, c\} \times \{1, 2\}$

$= \{(a, 1), (a, 2), (b, 1), (b, 2), (c, 1), (c, 2)\}$

### 2. Apply **selections / conditions**:

$$Y = \{(r, s) \in X \mid r.A = s.B\}$$

= Filtering!

### 3. Apply **projections** to get final output:

$$Z = (y.A) \text{ for } y \in Y$$

= Returning only *some* attributes

set-builder notation!

Remembering this order is critical to understanding the output of certain queries (see later on...)

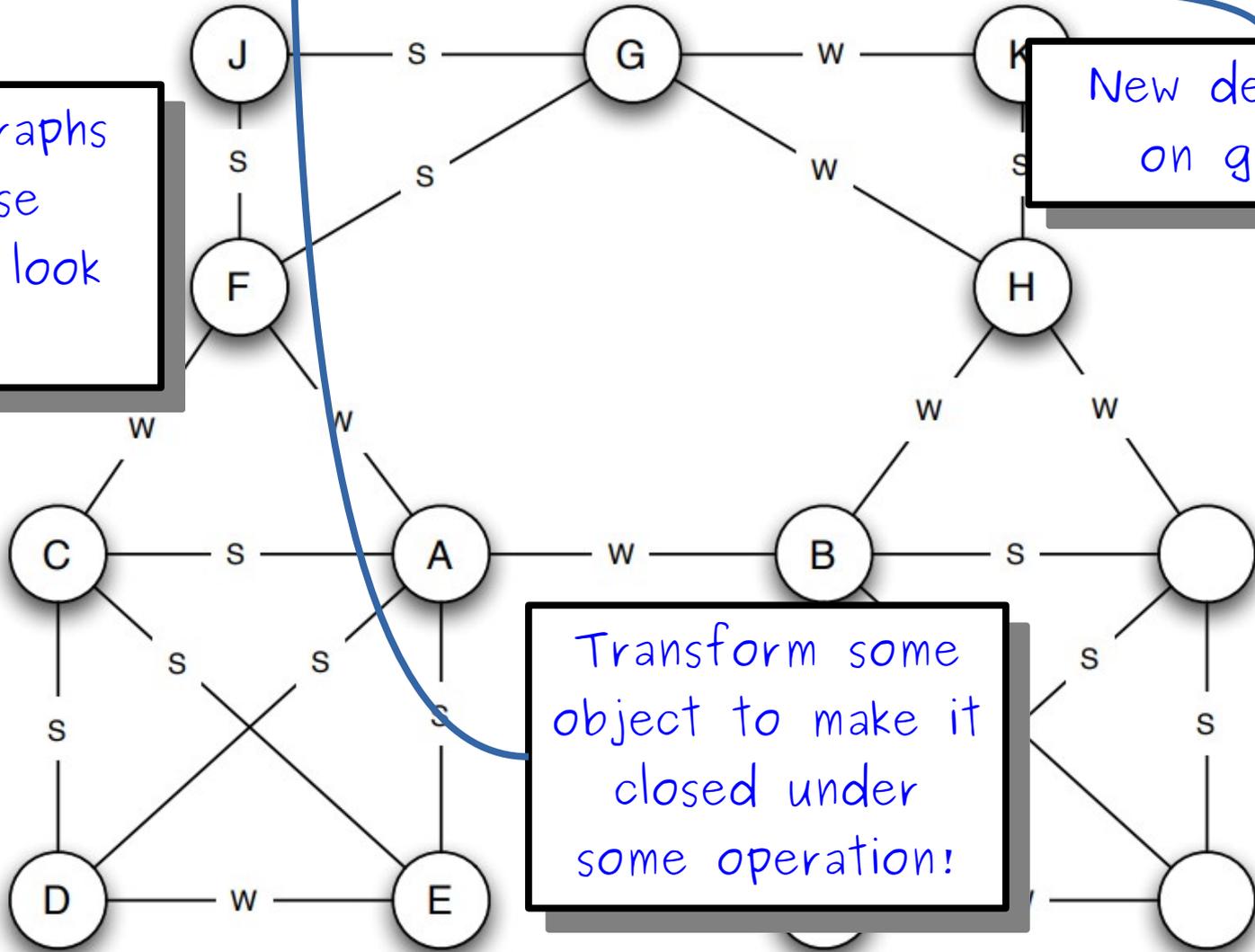
# Strong triadic closure

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!



# 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', '...']
```

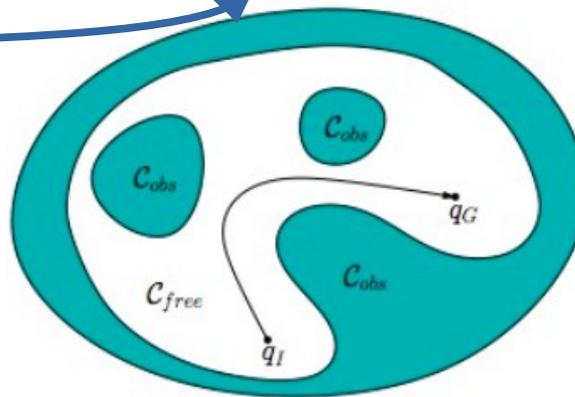
It's a big  
regex!

Describing the world in set theory!

# Planning in C-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:  $C_{free} = \{q \in C \mid R(q) \cap O = \emptyset\}$
- 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!



# Assignment #1

Due: 11:59pm on Mon., **Oct. 8, 2018**

Submit via Gradescope (each answer on a separate page) code: **9RZGVZ**

**Problem 1. Hash functions and proofs of work.** In class we defined two security properties for a hash function, one called collision resistance and the other called proof-of-work security. Show that a collision-resistant hash function may not be proof-of-work secure.

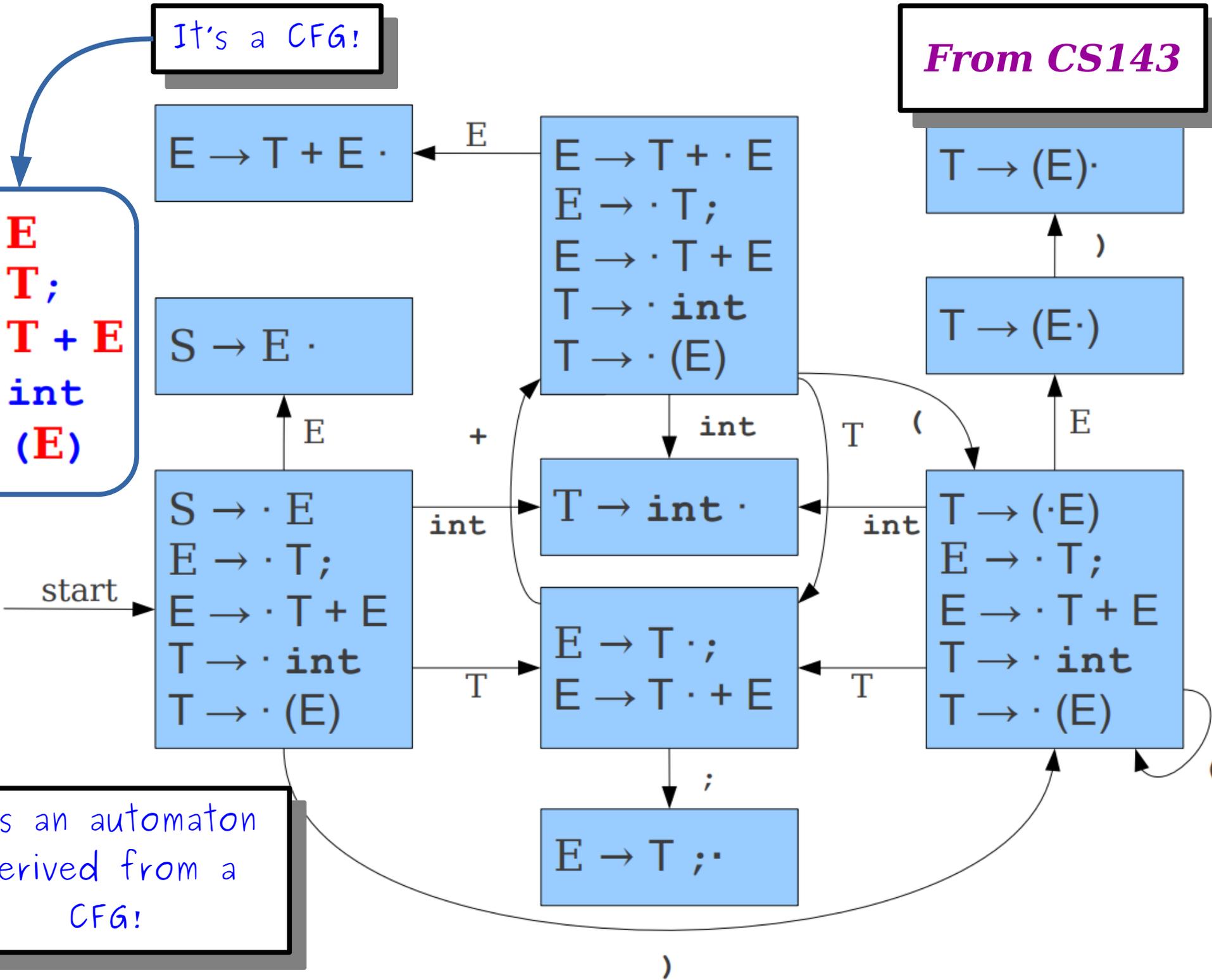
**Hint:** let  $H : X \times Y \rightarrow \{0, 1, \dots, 2^n - 1\}$  be a collision-resistant hash function. Construct a new hash function  $H' : X \times Y \rightarrow \{0, 1, \dots, 2^m - 1\}$  (where  $m$  may be greater than  $n$ ) that is also collision resistant, but for a fixed difficulty  $D$  (say,  $D = 2^{32}$ ) is not proof-of-work secure with difficulty  $D$ . That is, for every puzzle  $x \in X$  it should be trivial to find a solution  $y \in Y$  such that  $H'(x, y) < 2^m / D$ . This is despite  $H'$  being collision resistant. Remember to explain why your  $H'$  is collision resistant, that is, explain why a collision on  $H'$  would yield a collision on  $H$ .

Whoa, it's a function!

It's a CFG!

From CS143

**S** → **E**  
**E** → **T**;  
**E** → **T + E**  
**T** → **int**  
**T** → **(E)**



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

**Actions( $s$ ):** possible actions from state  $s$

**Succ( $s, a$ ):** where we end up if take action  $a$  in state  $s$

**Cost( $s, a$ ):** cost for taking action  $a$  in state  $s$

**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!

## II. Transfer Functions

- A family of transfer functions  $F$
- Basic Properties  $f: V \rightarrow V$

- Has an identity function
  - $\exists f$  such that  $f(x) = x$ , for all  $x$ .
- Closed under composition
  - if  $f_1, f_2 \in F$ ,  $f_1 \circ f_2 \in F$

It's functions  
with specific  
properties!

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

*From CS161*

$O(\dots)$  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,

$$\begin{aligned} T(n) = O(g(n)) \\ \iff \\ \exists c, n_0 > 0 \text{ s.t. } \forall n \geq n_0, \\ 0 \leq T(n) \leq c \cdot g(n) \end{aligned}$$

It's FOL and functions!

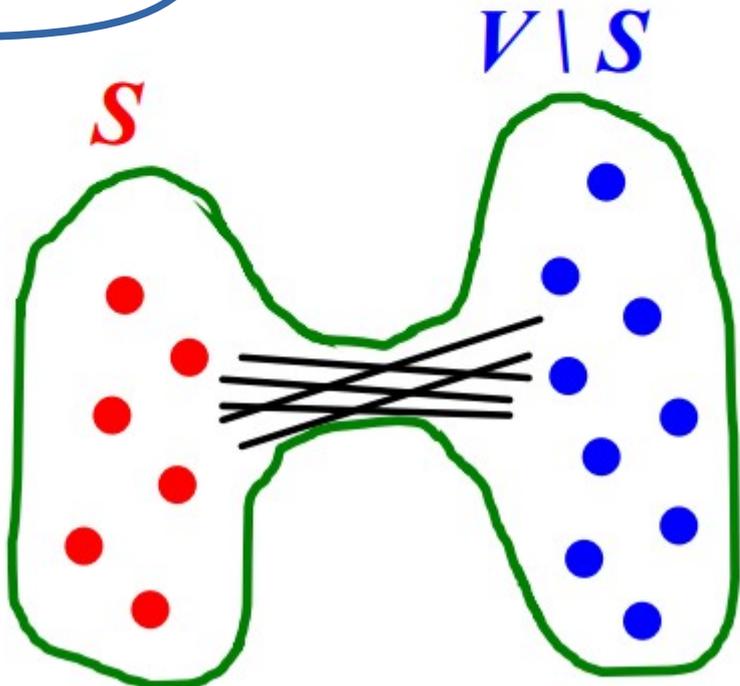
■ Graph  $G(V, E)$  has **expansion  $\alpha$** : if  $\forall S \subseteq V$ :  
# of edges leaving  $S \geq \alpha \cdot \min(|S|, |V \setminus S|)$

■ Or equivalently:

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

Set difference and cardinality!

First-order definitions on graphs!



# 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 $\tau ::=$	int	integer
	$\tau_1 \rightarrow \tau_2$	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 $\oplus ::=$	+   -   *   /	

It's a  
CFG!

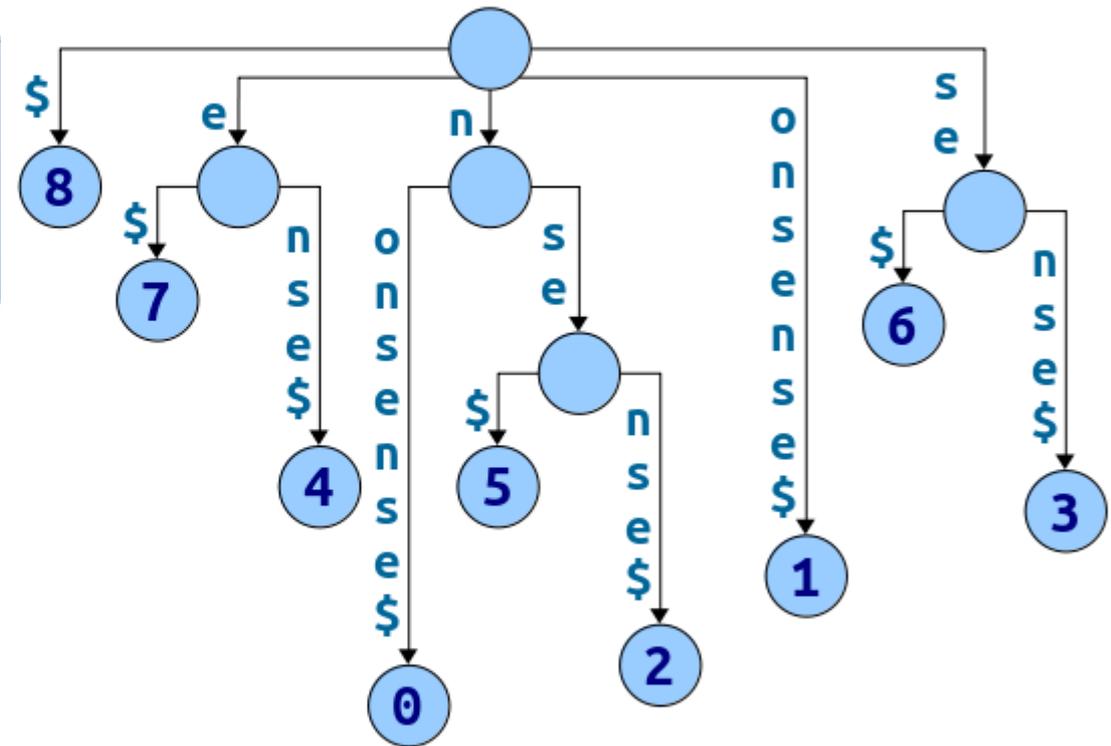
First, we introduce a language of types, indicated by the variable tau ( $\tau$ ). A type is either an integer, or a function from an input type  $\tau_1$  to an output type  $\tau_2$ . Then we extend our untyped lambda calculus with the same arithmetic language from the first lecture (numbers and binary operators)<sup>4</sup>. Usage of the language looks similar to before:

Definitions  
in terms of  
strings!

From CS166

# The Anatomy of a Suffix Tree

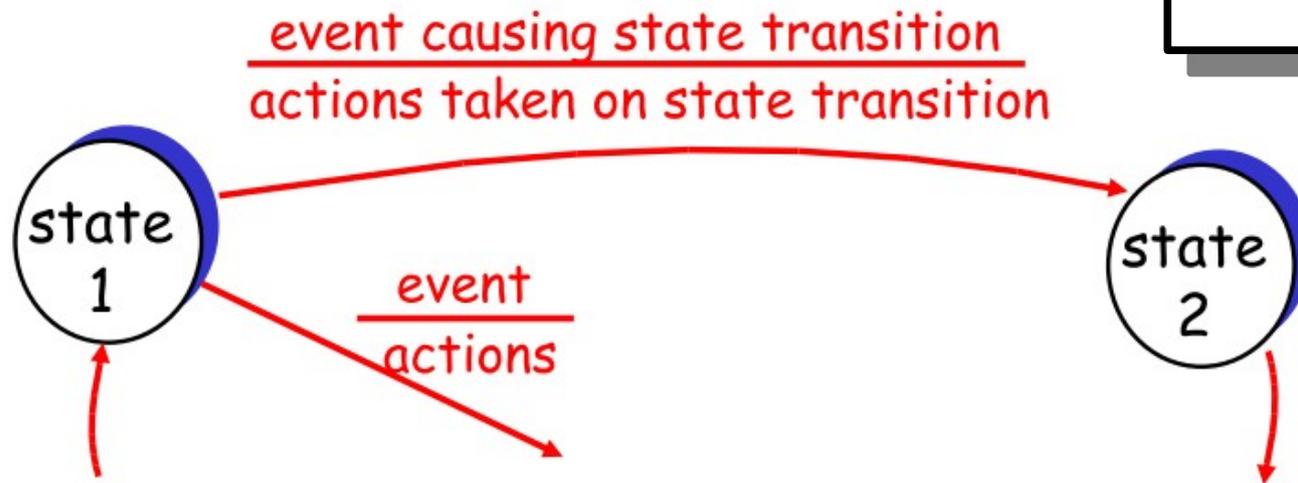
- A **branching word** in  $T\$$  is a string  $\omega$  such that there are characters  $a \neq b$  where  $\omega a$  and  $\omega b$  are substrings of  $T\$$ .
  - Edge case: the empty string is always considered branching.
- **Theorem:** The suffix tree for a string  $T$  has an internal node for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



nonsense\$  
012345678

# 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 transition

*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

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.<sup>5</sup>

A Myhill-  
Nerode-style  
argument!

## 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!

- Suppose we are given a set of documents  $D$ 
  - Each document  $d$  covers a set  $X_d$  of words/topics/named entities  $W$
- For a set of documents  $A \subseteq D$  we define

$$F(A) = \left| \bigcup_{d \in A} X_d \right|$$

Functions, set union, and set cardinality!

- Goal: We want to

$$\max_{|A| \leq k} F(A)$$

- Note:  $F(A)$  is a set function:  $F(A): \text{Sets} \rightarrow \mathbb{N}$

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 }
- Math 107 } ***Graphs***
- Math 108 }
- Math 113 }
- Math 120 } ***Functions***
- Math 161 } ***Set Theory***
- Math 152 } ***Number Theory***

## ***Applications***

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

Want to get involved in research?

***Learning patterns in randomness***  
**(Greg Valiant)**

***Fairness and models of computation***  
**(Omer Reingold)**

***Approximating NP-hard problems***  
**(Moses Charikar)**

*Optimizing programs... randomly*  
**(Alex Aiken)**

***Structure from symmetries***  
**(Leo Guibas)**

***Computing on encrypted data***  
**(Dan Boneh)**

***Correcting errors automatically***  
**(Mary Wootters)**

***Game theory, P, and NP***  
**(Aviad Rubenstein)**

***Expander graphs and random sampling***  
**(Nima Anari)**

***Logic circuits and random bits***  
**(Li-Yang Tan)**

***How powerful are quantum computers?***  
**(Adam Bouland)**

***Solving optimization problems quickly***  
**(Aaron Sidford)**

Your Questions

What do you want to know?

# 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!***



*Theory*

*Practice*

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?