Where to Go from Here

Announcements

- Problem Set 9 was due at 3:00 PM
  - You can use a late day to extend the deadline to tomorrow at 3:00PM.
  - Solutions will go online Thursday night.

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 *Saturday, June 8* from 8:30-11:30am. 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.

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
Galois Connections
Bijections

Images and Preimages
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
Error-Correcting Codes
Regular Expressions
State Elimination
Monoids

Distinguishability
Myhill-Nerode Theorem
Nonregular Languages
Context-Free Grammars
Brzozowski'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 and **RE**
Complexity Class **P**
Complexity Class **NP**
$\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ 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):

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

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

- Pseudo Random Permutation (**PRP**)

  $$E: K \times X \to X$$

  such that:

  1. Exists “efficient” algorithm to evaluate E(k,x)
  2. The function E(k, ·) is one-to-one

  “efficient” inversion algorithm D(k,x)

Functions between sets: K × X is the set of all pairs made from K and X.

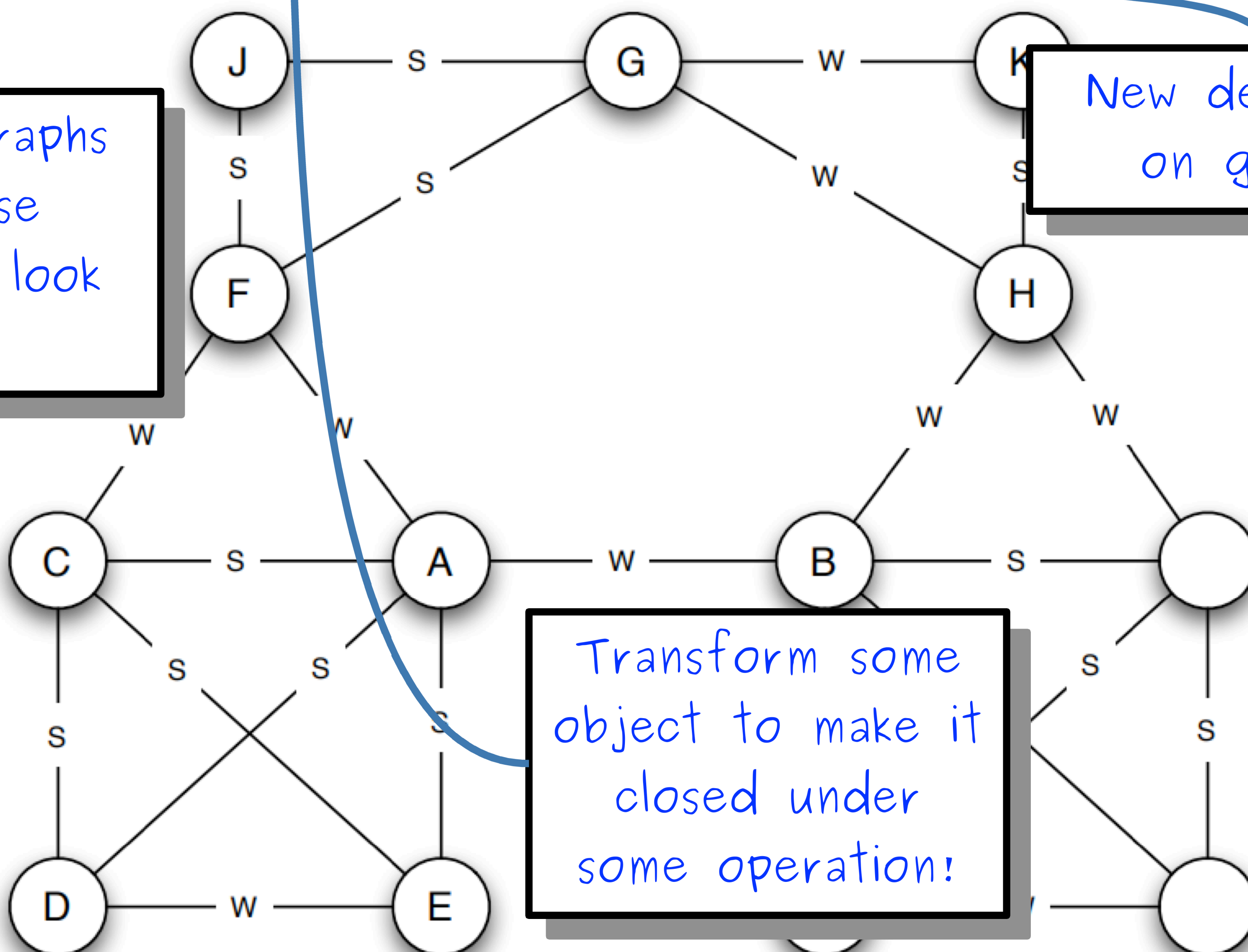
Definitions in terms of efficiency!

Injectivity!

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!

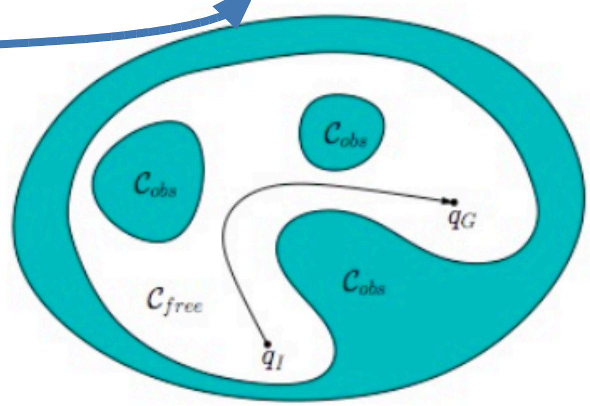
From CS237A

Pla[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: τ: $[0,1] \rightarrow C_{free}$, with $\tau(0) = q_I$ and $\tau(1) = q_G$

Model paths as functions!



From CS251

CS251: Cryptocurrencies and Blockchain Technologies **Fall 2018**

Assignment #1

Due: 11:59pm on Mon., **Oct. 8, 2018**
Submit via Gradescope (each answer on a separate page) code: **9RZGVZ**

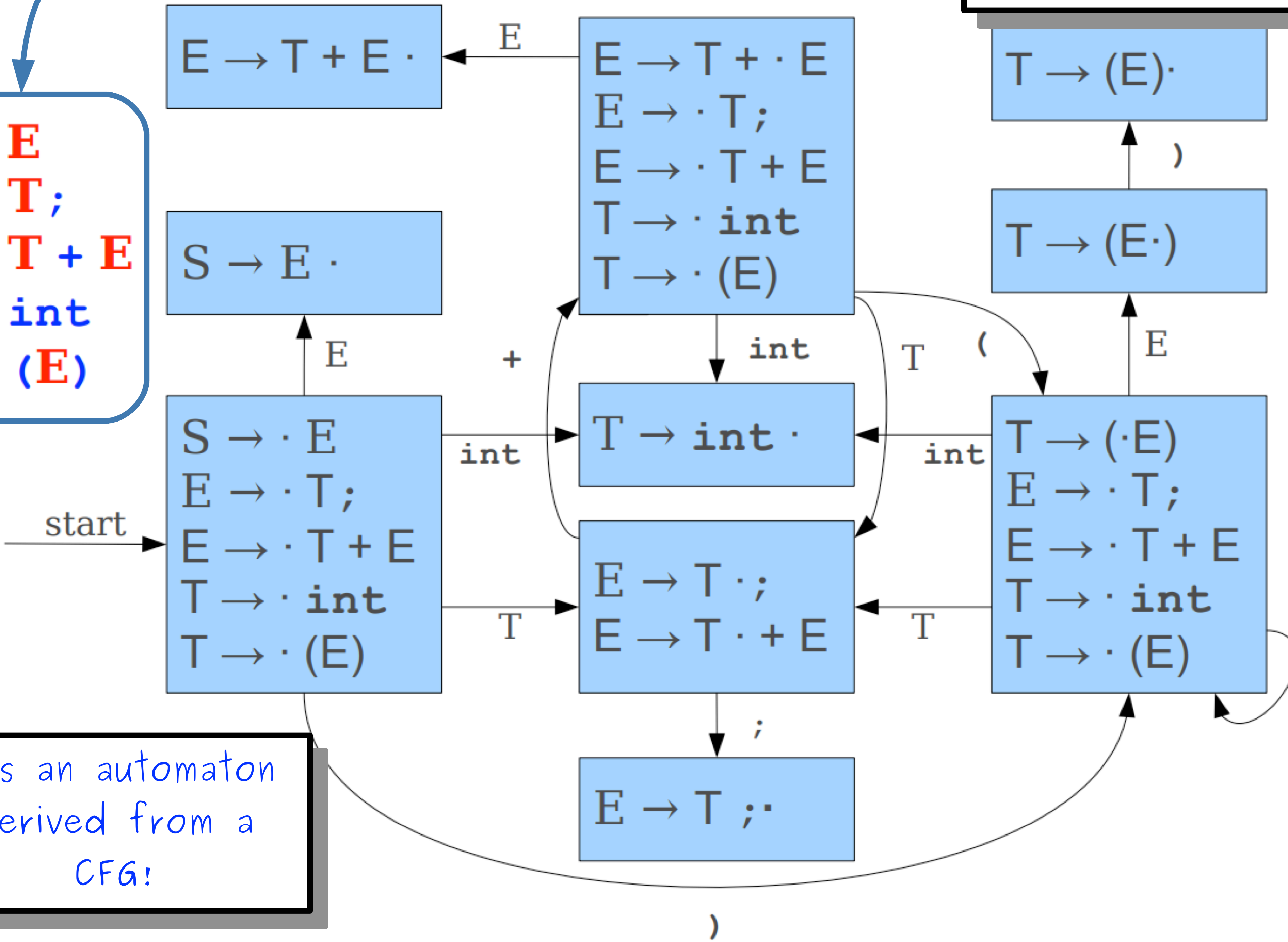
Problem 1. Hash functions and proofs of work. In class we defined two security properties for a hash function, one called collision resistance and the other called proof-of-work security. Show that a collision-resistant hash function may not be proof-of-work secure.
Hint: let $H : X \times Y \to \{0, 1, \ldots, 2^n - 1\}$ be a collision-resistant hash function. Construct a new hash function $H' : X \times Y \to \{0, 1, \ldots, 2^m - 1\}$ (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

$\text{Actions}(s)$: 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!

From CS243

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 $f_1, f_2 \in F$, $f_1 \bullet f_2 \in F$

It's functions with specific properties!

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 $\forall S \subseteq V$:

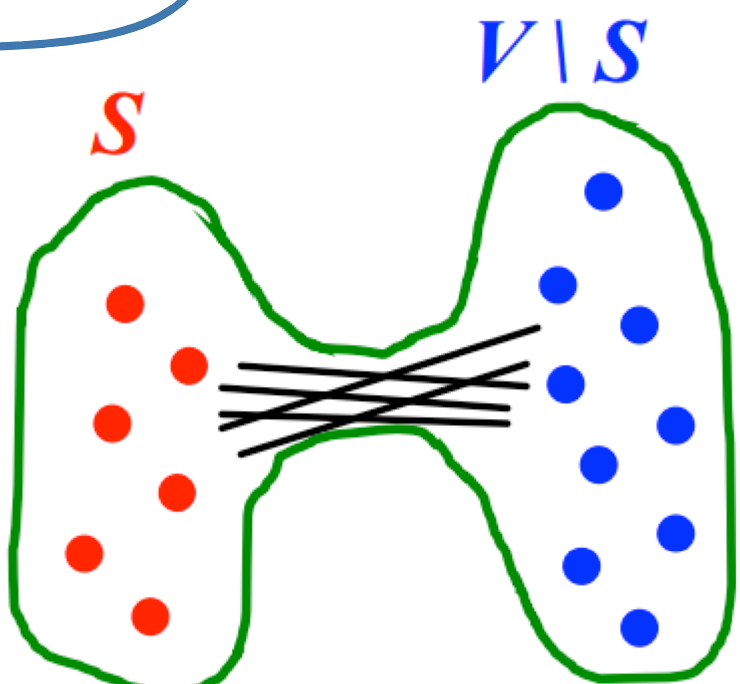
  $\#$ of edges leaving $S \geq \alpha \cdot \min(|S|, |V \setminus S|)$

- **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:

$$
\begin{array}{rcll}
\text{Type } \tau & ::= & \mathsf{int} & \text{integer} \\
& \mid & \tau_1 \to \tau_2 & \text{function} \\
\\
\text{Expression } e & ::= & x & \text{variable} \\
& \mid & n & \text{integer} \\
& \mid & e_1 \oplus e_2 & \text{binary operation} \\
& \mid & \lambda\ (x : \tau)\ .\ e & \text{function} \\
& \mid & e_1\ e_2 & \text{application} \\
\\
\text{Binop } \oplus & ::= & + \mid - \mid * \mid / &
\end{array}
$$

It's a CFG!

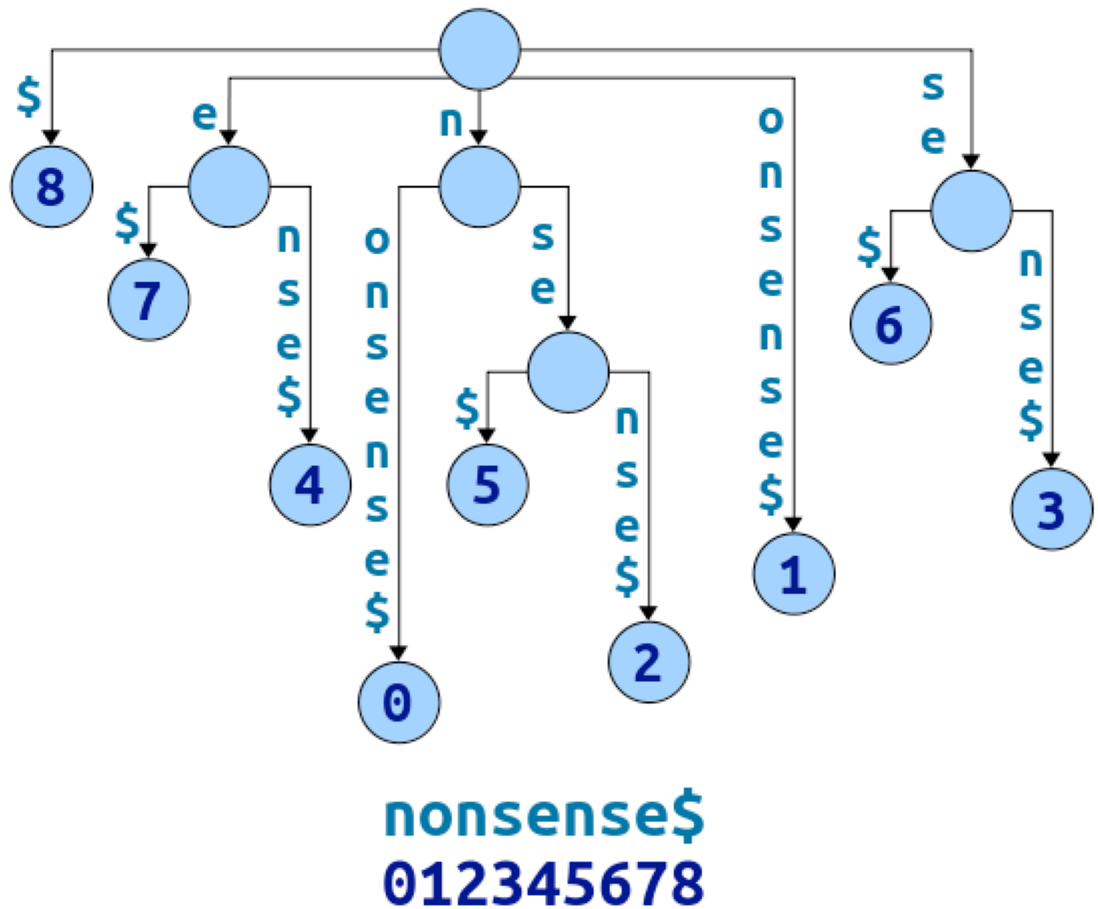
First, we introduce a language of types, indicated by the variable tau (τ). A type is either an integer, or a function from an input type τ_1 to an output type τ_2. Then we extend our untyped lambda calculus with the same arithmetic language from the first lecture (numbers and binary operators)[4]. Usage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

Definitions in terms of strings!

From CS166

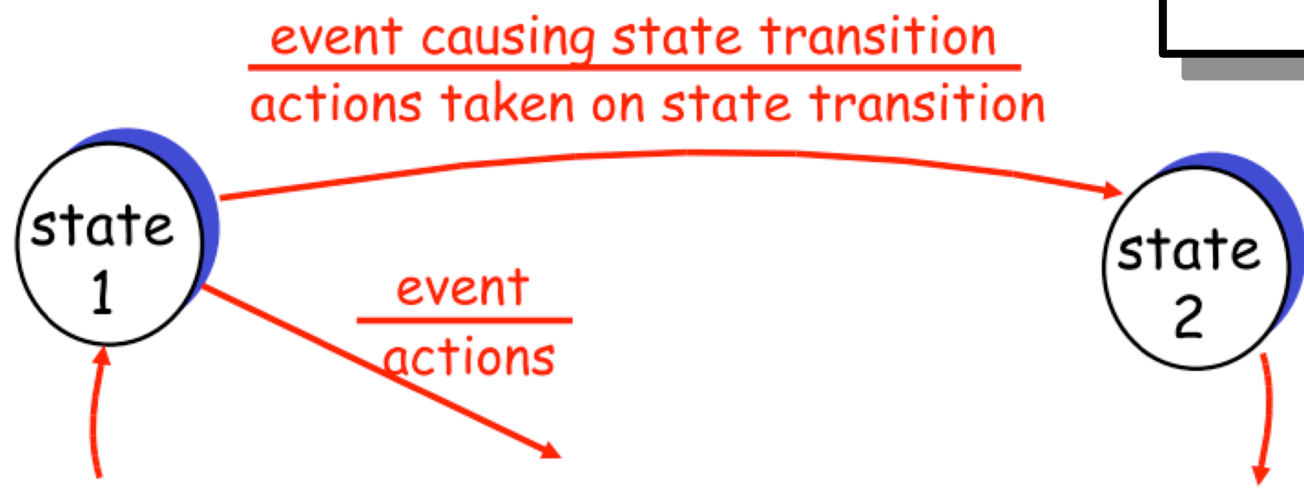
The Anatomy of a Suffix Tree

- 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 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

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 CS246

- **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: $F(A)$: Sets $\rightarrow \mathbb{N}$**

From CS250

Alphabets!

4 FORMAL DEFINITIONS

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

DEF. A CODE $\mathcal{C}$ of BLOCK LENGTH n over an ALPHABET Σ is a subset $\mathcal{C} \subseteq \Sigma^n$.
An element $c \in \mathcal{C}$ 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
- Phil 151
- Phil 152
- Math 107
- Math 108
- Math 113
- Math 120
- Math 161
- Math 152

Complexity (CS154)
Computability (CS154, Phil 151, Phil 152)
Graphs (Math 107, Math 108)
Functions (Math 113, Math 120, Math 161)
Set Theory (Math 161)
Number Theory (Math 152)

Applications

- CS124
- CS143
- CS161
- CS224W
- CS242
- CS243
- CS246
- CS250
- CS251
- CS255

Languages / Automata (CS124, CS143)
Graphs (CS124 – CS250)
Functions (CS224W – 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?