CS103
Winter 2025

Lecture 27:
The Journey's End, The Journey Onward

Outline for Today

- ***Announcements***
  - Expectations as we head into the end of the quarter.
- ***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!***

Announcements

- Problem Set 9 was due thirty minutes ago.
  - You can use a late day to extend the deadline to tomorrow at 1:00PM.
  - Solutions will go online Monday at 1:00PM.

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?

Final Exam Logistics

- Our final exam is on ***Wednesday, March 19th*** from ***3:30 – 6:30 PM***.
    - Locations are now available on the course website; check your seat assignment ASAP and write it down somewhere easily accessible.
- The final exam is cumulative and covers topics from PS1 – PS9 and L00 – L26. 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

- Iris and Stanley are running a review session ***Monday*** from ***3:00 – 4:00 PM*** in ***CoDa E160***.
- 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.

EOQ Logistics

- This Sunday is the last day for OH!
- Ed will lock at noon Wednesday, March 19th.
- The team is grading the weekend after the exam.
- Grades are due to the Registrar at 11:59 PM on Tuesday, March 25.
- I use an anonymized spreadsheet when assigning final letter grades. I’m committed to equitability and impartiality in grading.

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.

Remember Problem Set 7?

Wow, so many names to shout out!

(suspense…)

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
Triangular Numbers
Tournaments
Functions
Injections
Surjections
Involutions
Monotone Functions
Minkowski Sums
Bijections

Graphs
Connectivity
Independent Sets
Vertex Covers
Trees
Bipartite Graphs
The Pigeonhole Principle
Ramsey Theory
Mathematical Induction
Complete Induction
The Spanning Tree Protocol
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
Merkle-Damgård Construction
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**
co-**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
Intro to Cryptography

- 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**)

  $$\boxed{E:\ K \times X \to X}$$

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

  such that:

  1. Exists $\boxed{\text{“efficient”}}$ algorithm to evaluate E(k,x)
  2. Th nction E(k, ·) is $\boxed{\text{one-to-one}}$

     “efficient” inversion algorithm D(k,x)

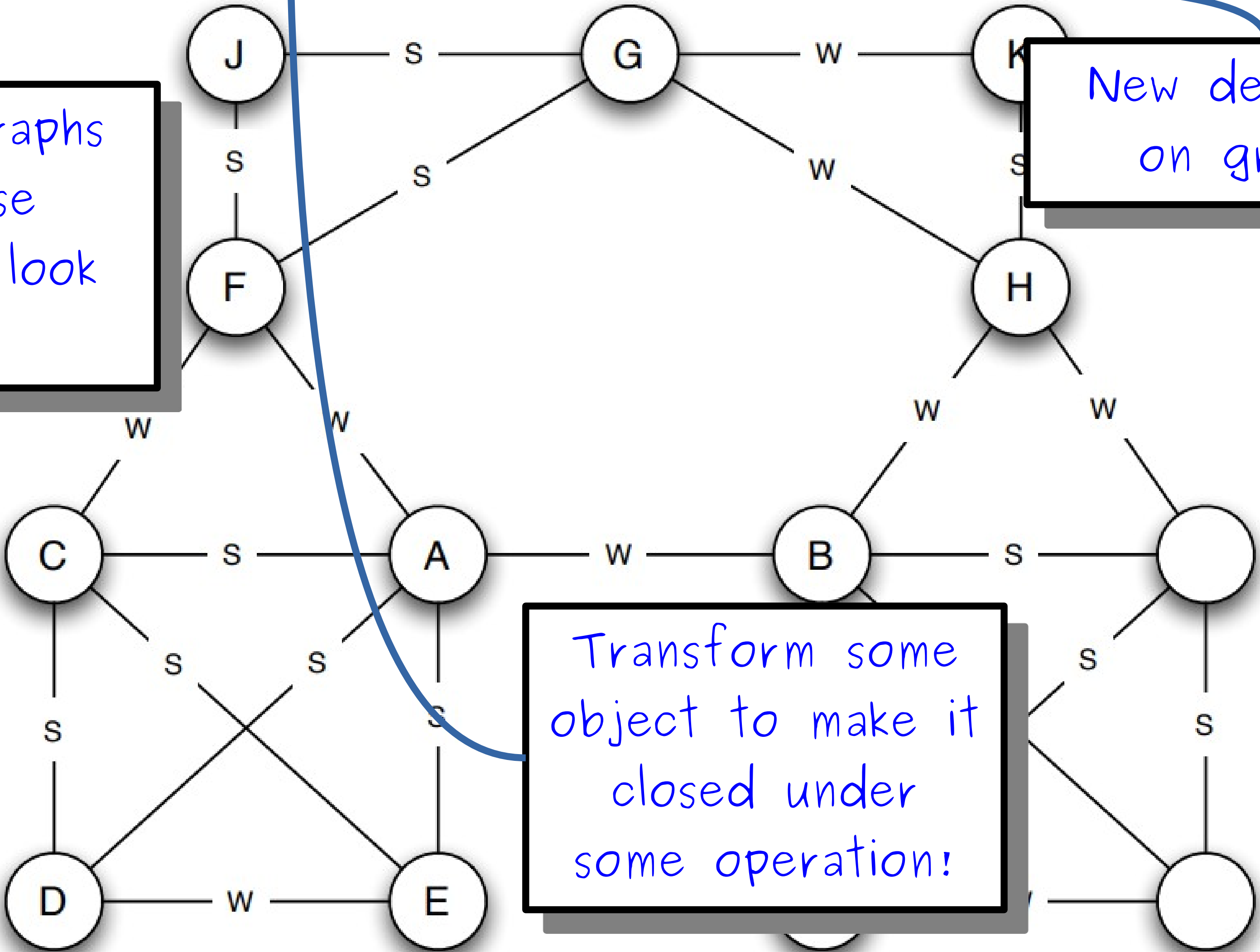
Definitions in terms of efficiency!

Injectivity!

Strong triadic closure

From CS124
From Languages to Information

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?

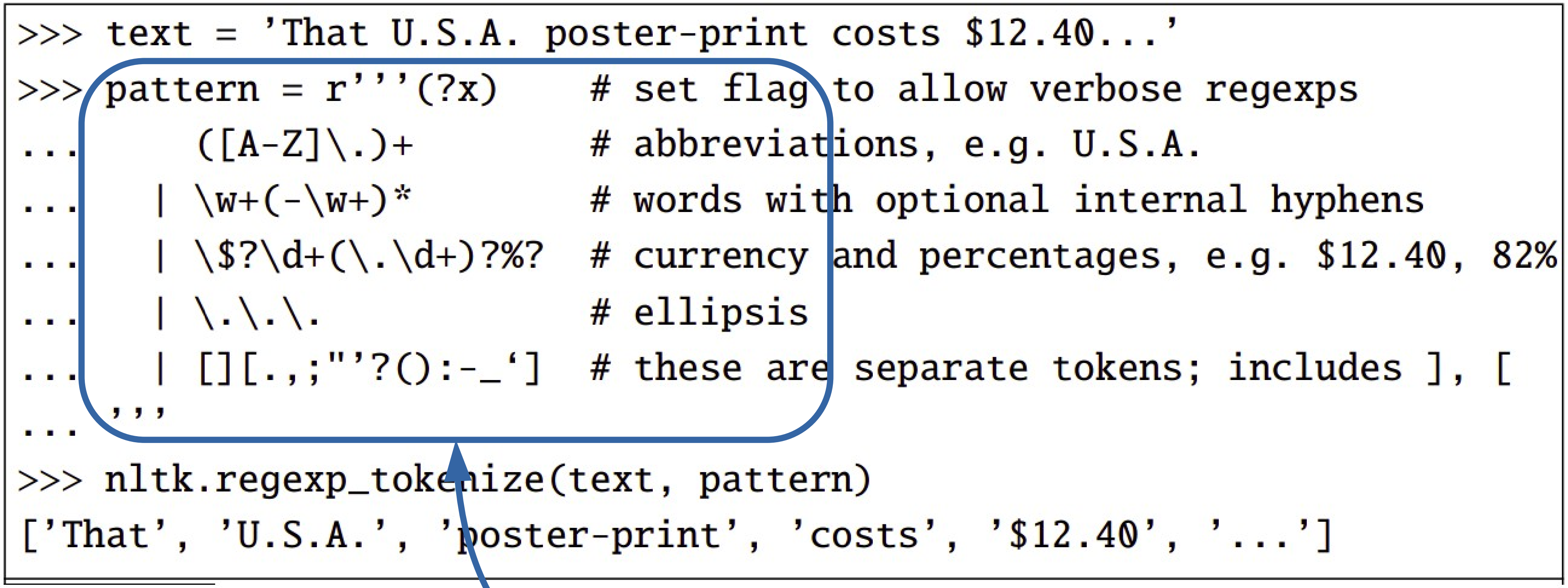
New definitions on graphs!

Transform some object to make it closed under some operation!

From CS124
From Languages to Information

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!

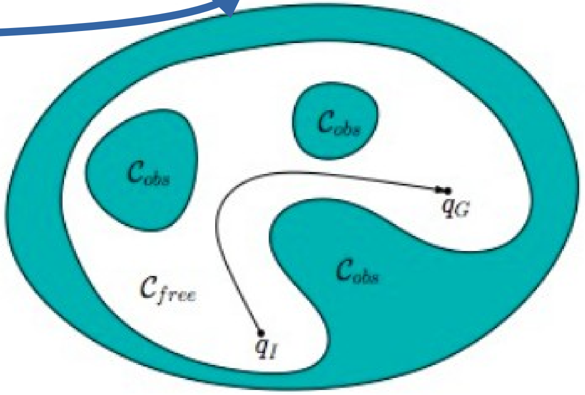
Plan space

From CS237A
Principles of Robot Autonomy

- 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: $C_{free} = \{q \in C | R(q) \cap O = \emptyset\}$

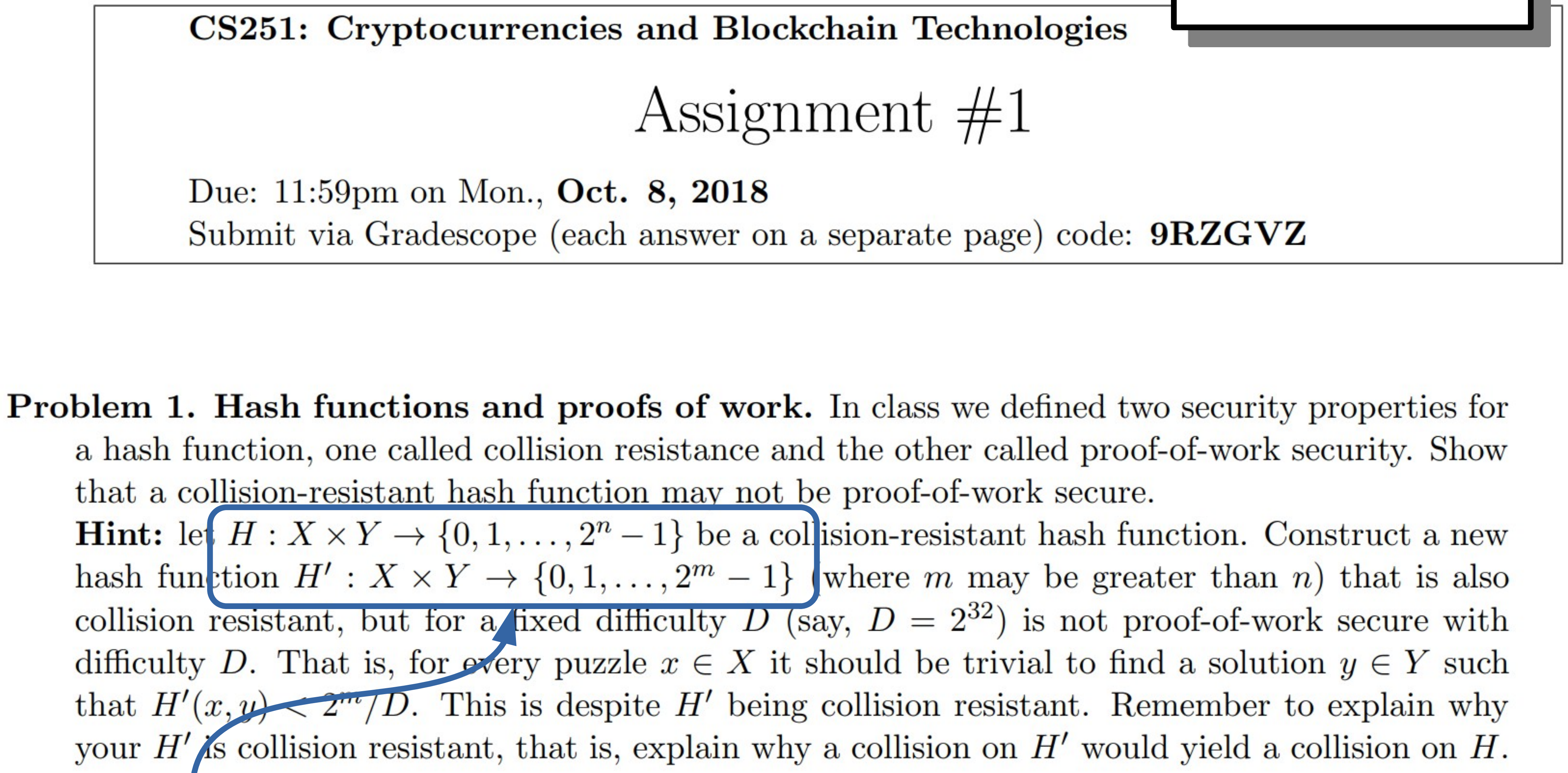
- Path planning problem in C-space: compute a **continuous** path: $\tau: [0,1] \rightarrow C_{free}$, with $\tau(0) = q_I$ and $\tau(1) = q_G$

Model paths as functions!



From CS251

Cryptocurrencies and Blockchain Technologies



CS251: Cryptocurrencies and Blockchain Technologies

Assignment #1

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

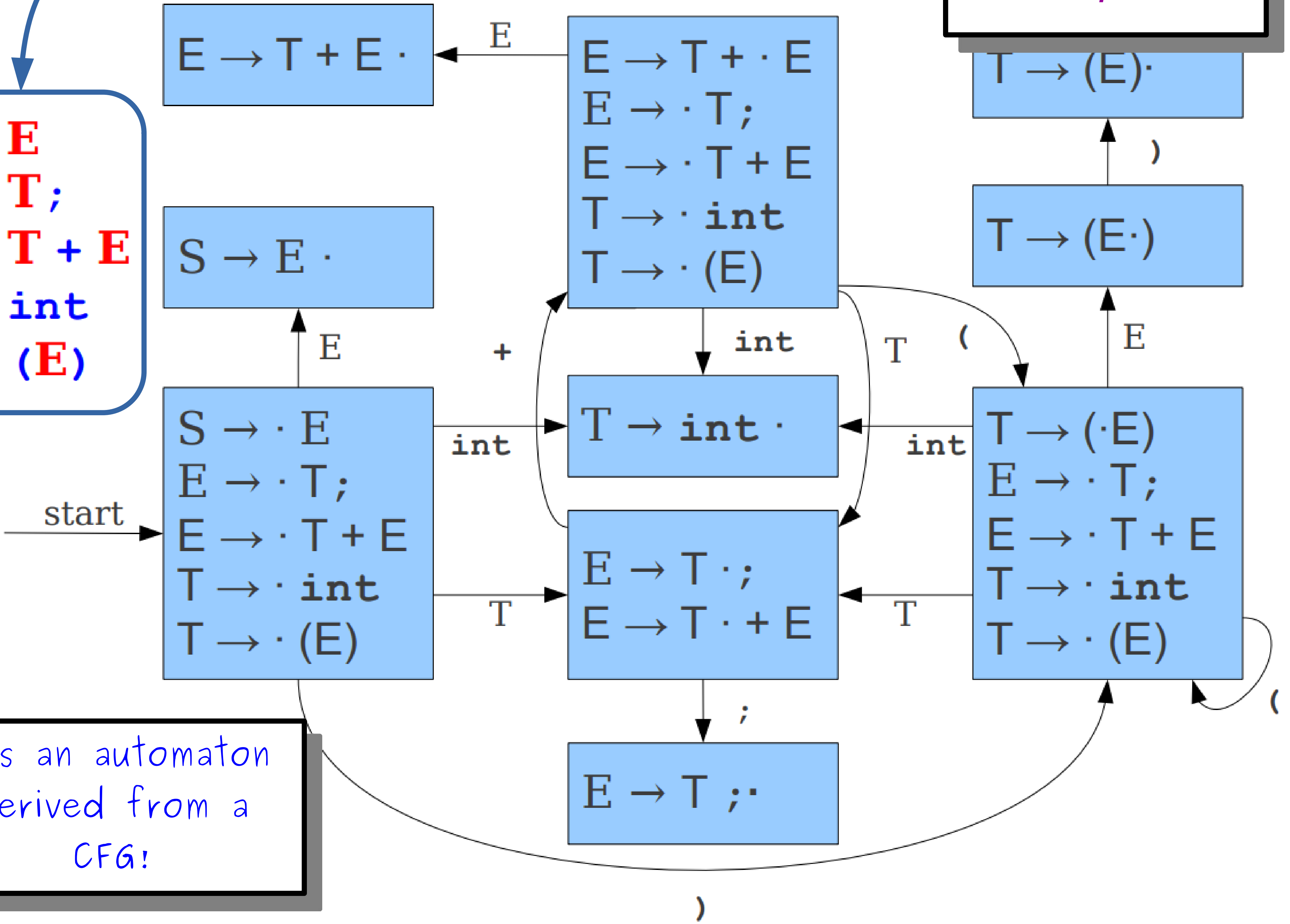
Problem 1. Hash functions and proofs of work. In class we defined two security properties for a hash function, one called collision resistance and the other called proof-of-work security. Show that a collision-resistant hash function may not be proof-of-work secure.
Hint: let $H : X \times Y \to \{0,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

$S \rightarrow E$
$E \rightarrow T;$
$E \rightarrow T + E$
$T \rightarrow$ `int`
$T \rightarrow (E)$



It's an automaton derived from a CFG!

Search problems

From CS221
Artificial Intelligence

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
Program Analysis with Optimizations

II. Transfer Functions

- A family of transfer functions F
- Basic Properties $f : V \to V$
  - 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 ..."

O(...) means an upper bound

From CS161
Design and Analysis of Algorithms

- 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,

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

It's FOL and functions!

From CS224W
Machine Learning with Graphs

- Graph $G(V, E)$ has **expansion α**: if $\forall\, S \subseteq V$:

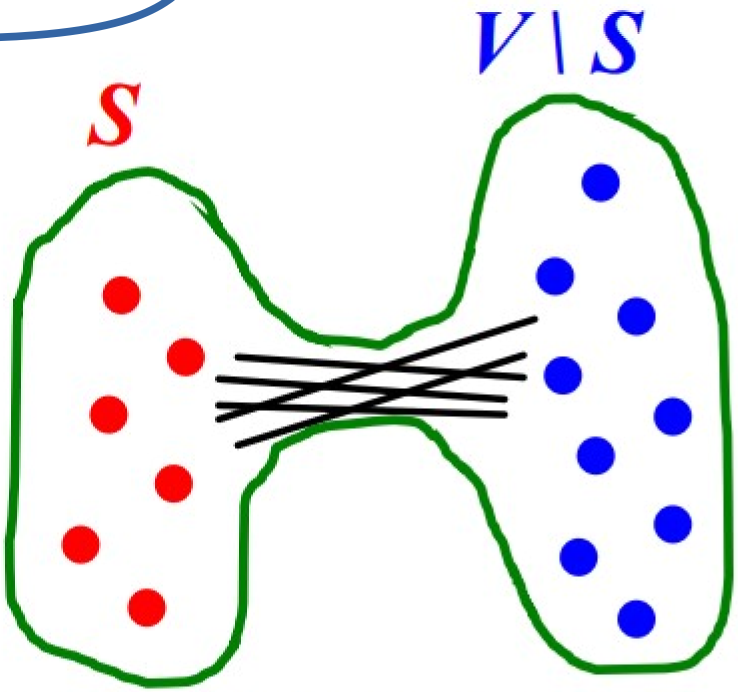
  $\#$ of edges leaving $S \geq \alpha \cdot \min(|S|, |V \setminus S|)$

- **Or equivalently:**

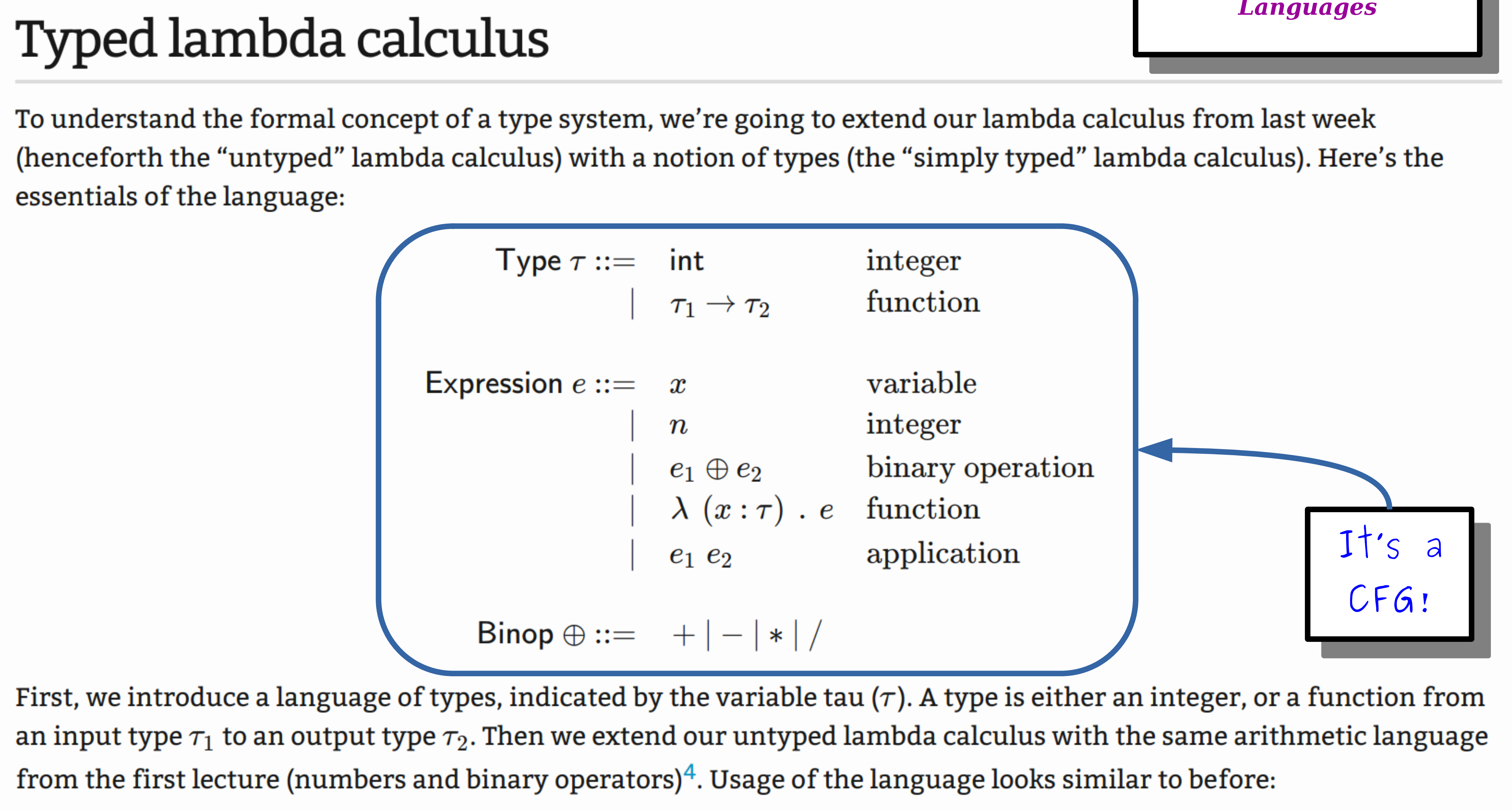
$$\alpha = \min_{S \subseteq V} \frac{\#\, edges\ leaving\ S}{\boxed{\min(|S|, |V \setminus S|)}}$$

Set difference and cardinality!

First-order definitions on graphs!



From CS242
Programming Languages



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:

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

It's a CFG!

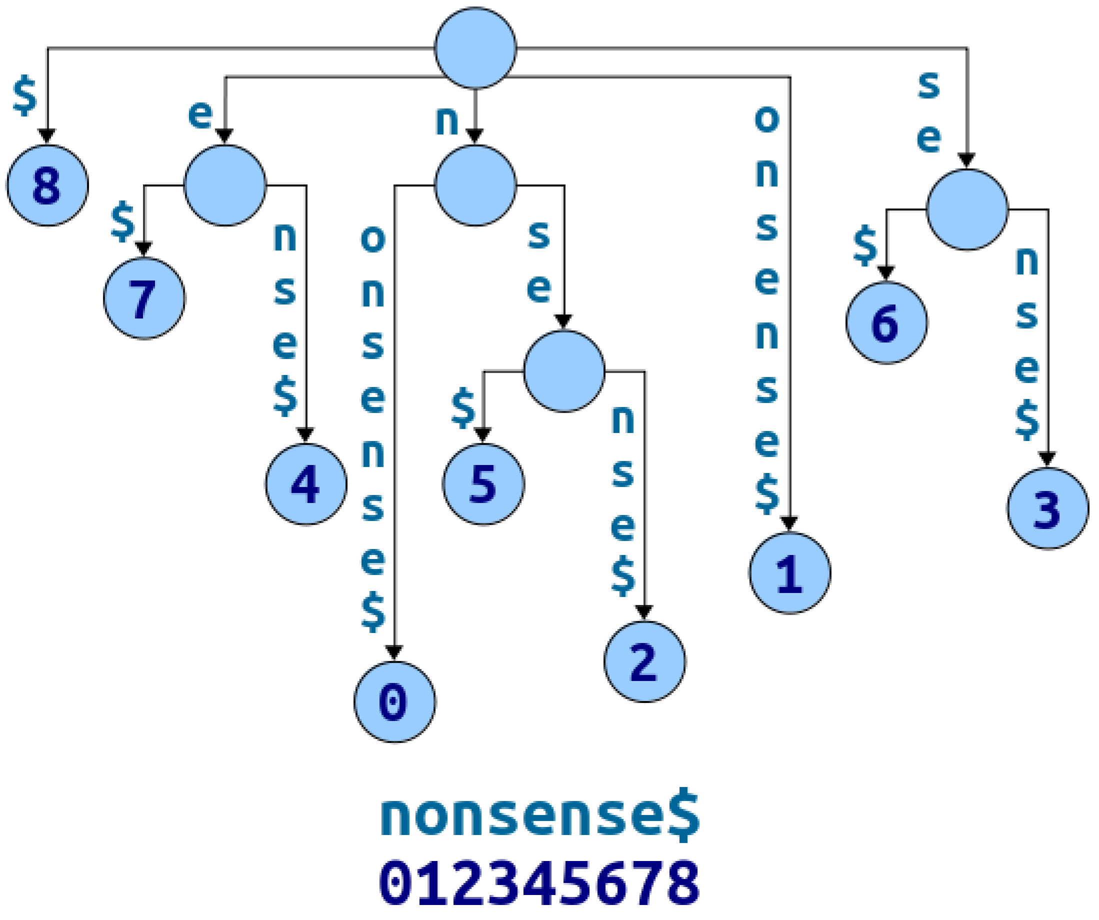
First, we introduce a language of types, indicated by the variable tau (τ). A type is either an integer, or a function from an input type τ_1 to an output type τ_2. Then we extend our untyped lambda calculus with the same arithmetic language from the first lecture (numbers and binary operators)[4]. Usage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

Definitions in terms of strings!

From CS166
Advanced Data Structures

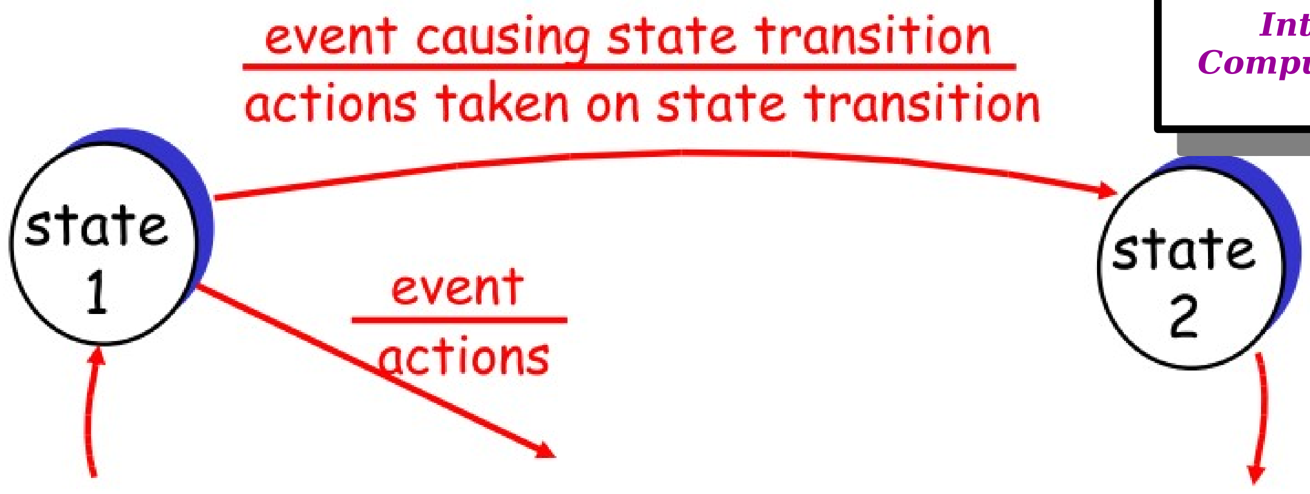
The Anatomy of a Suffix Tree

- 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
Introduction to Computer Networking



event causing state transition
actions taken on state transition
state 1
event
actions
state 2

- **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

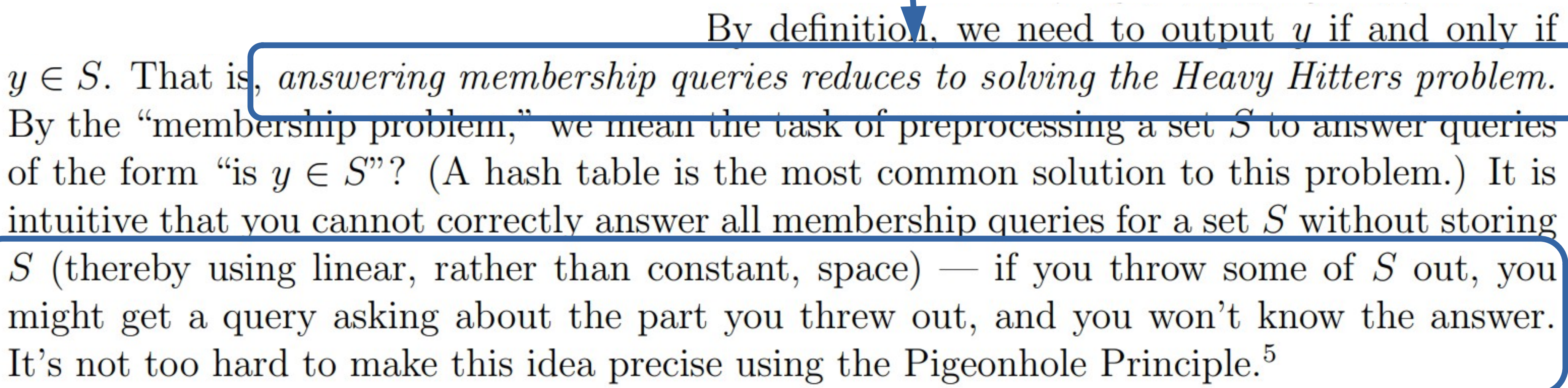
It's a generalization of DFAs!

- **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

The Modern Algorithmic Toolbox

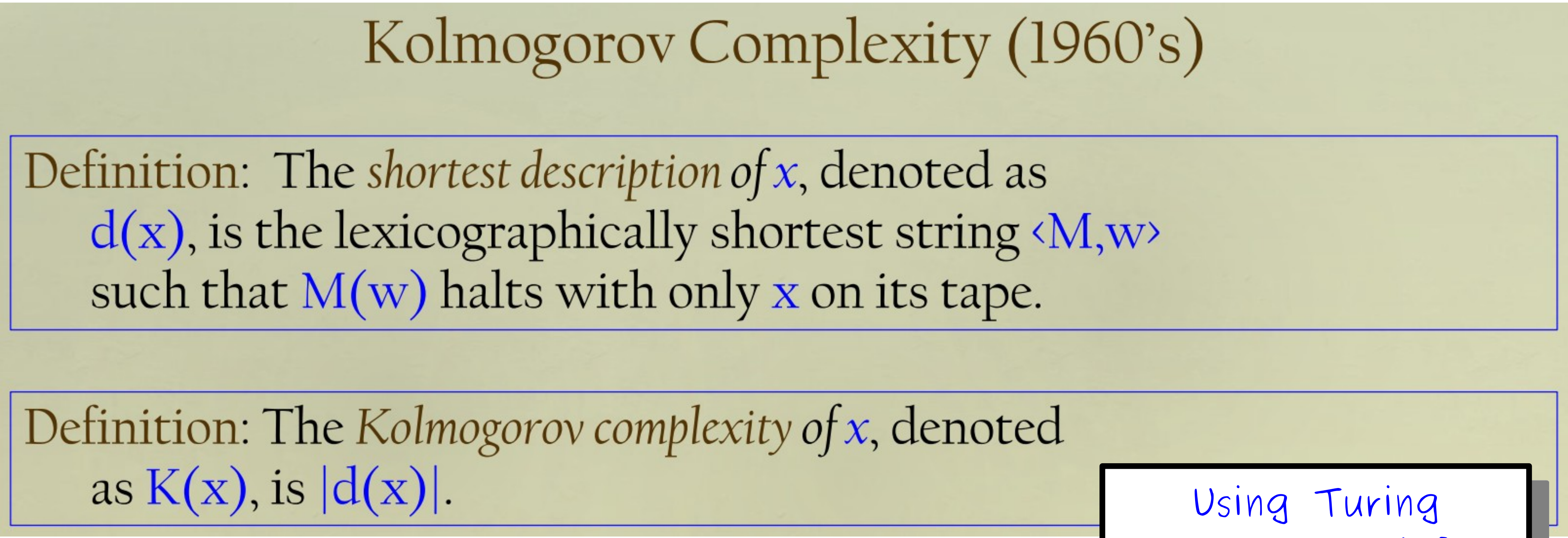
Reducibility!


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

Introduction to Automata and Complexity Theory



Kolmogorov Complexity (1960's)

Definition: The *shortest description of* x, denoted as d(x), is the lexicographically shortest string ‹M,w› 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
Mining Massive Data Sets

- **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**

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

Functions, set union, and set cardinality!

- **Goal: We want to**

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

- **Note: *F(A)* is a set function: $F(A): \mathbf{Sets} \to \mathbb{N}$**

From CS250
Algebraic Error Correcting Codes

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

- ***CS154:*** Introduction to the Theory of Computation
  - The “spiritual sequel” to CS103; does a deep dive into automata, TMs, and computability/complexity theory.
  - If you enjoyed the tail end of this course, highly recommended as a next step.
- ***CS161:*** Design and Analysis of Algorithms
  - A natural next course in CS theory, focusing on the design of efficient algorithms.
  - (Super helpful for job interviews!)
- ***CS143:*** Compilers
  - Use your automata and CFG prowess to translate source code into machine code. Extremely rewarding!
- ***CS257:*** Introduction to Automated Reasoning
  - See how to automate formal proofs, play around with SAT and propositional logic, etc.

Course Recommendations

Theoryland

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

Applications

- CS124 — ***Languages / Automata*** (CS124, CS143)
- CS143
- CS161
- CS224W — ***Graphs*** (CS124 – CS250)
- CS242
- CS243
- CS246 — ***Functions*** (CS224W – CS255)
- CS250
- CS251
- CS255

The CS Theory Group

- Stanford's has a world-class theory group in the CS department doing research in cryptography, error-correcting codes, algorithms, machine learning, complexity theory, algorithmic fairness, etc.
- The faculty are super approachable and down-to-earth. The theory group also has a stellar student-to-faculty ratio (something like 6:1 undergrads to professors).
- The group holds weekly Thursday lunches and "Theory Tea" events. Interested in learning more? ***Join their mailing list!***

Something Else to Consider

- If you enjoyed (parts of) this class:
  - CS is a great place for you!
  - Try out some of the courses listed above!
- If you did NOT enjoy this class:

Something Else to Consider

- If you enjoyed (parts of) this class:
  - CS is a great place for you!
  - Try out some of the courses listed above!

- If you did NOT enjoy this ~~class~~ **material**:

Something Else to Consider

- If you enjoyed (parts of) this class:
  - CS is a great place for you!
  - Try out some of the courses listed above!

- If you did NOT enjoy this ~~class~~ **material**:
  - CS is a great place for you!
  - Theoryland is a wild, exotic, and ***amazing*** place, but it’s not for everyone. You can find joy and fulfillment in other areas of CS.

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.

Your skills are ***rare***.
Your skills are ***powerful***.
Best of luck wherever they take you!