

Wrap-Up

Outline for Today

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

Announcements

- The final exam is Saturday 8:30am-11:30am
- Same ground rules as the midterms
 - 1 8.5"x11" sheet of notes
 - No electronic devices
 - Cumulative, all psets and lectures in scope
 - Except this lecture

Announcements

- There is a review session immediately after class 4:30-6:30
- Will work through a practice exam

Please evaluate this course on Axess.
Your feedback really makes a difference.

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!

The Big Picture

Take a minute to reflect on what we have covered.

Set Theory	Graphs	Myhill-Nerode Theorem
Power Sets	Connectivity	Nonregular Languages
Cantor's Theorem	Independent Sets	Context-Free Grammars
Direct Proofs	Vertex Covers	Merkle-Damgård Construction
Parity	Trees	Fixed Point Theorems
Proof by Contrapositive	Bipartite Graphs	Turing Machines
Proof by Contradiction	The Pigeonhole Principle	Church-Turing Thesis
Modular Congruence	Ramsey Theory	TM Encodings
Propositional Logic	Mathematical Induction	Universal Turing Machines
First-Order Logic	Complete Induction	Self-Reference
Logic Translations	The Spanning Tree Protocol	Decidability
Logical Negations	Formal Languages	Recognizability
Propositional Completeness	DFAs	Self-Defeating Objects
Vacuous Truths	Regular Languages	Undecidable Problems
Perfect Squares	Closure Properties	The Halting Problem
Triangular Numbers	NFAs	Verifiers
Tournaments	Subset Construction	Diagonalization Language
Functions	Kleene Closures	R and RE
Injections	Error-Correcting Codes	co- RE
Surjections	Regular Expressions	Complexity Class P
Involutions	State Elimination	Complexity Class NP
Monotone Functions	Monoids	P $\stackrel{?}{=}$ NP Problem
Minkowski Sums	Distinguishability	NP -Completeness
Bijections		

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) :

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

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!

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 .

Strong triadic closure

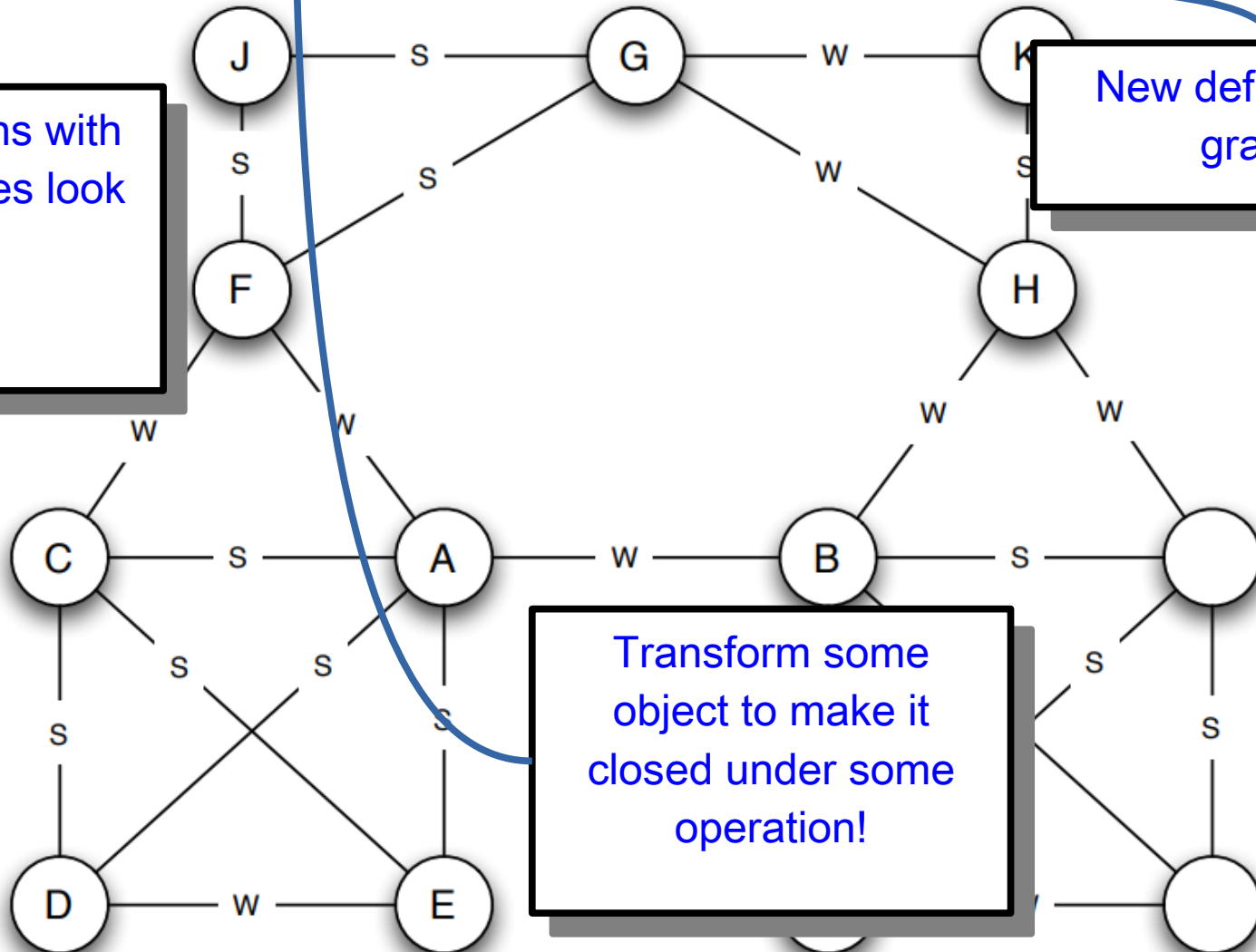
From CS124
From Languages to
Information

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!



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!

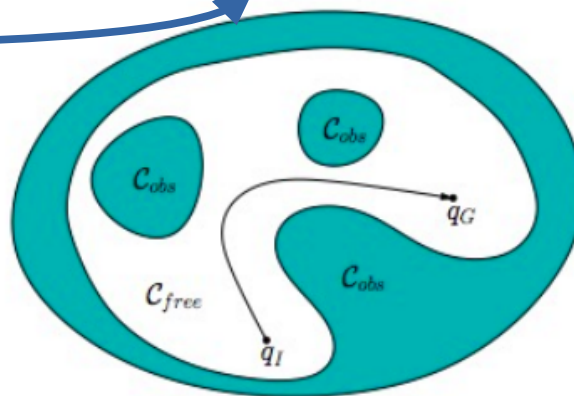
Describing the world in set theory!

From
CS237A
*Principles of Robot
Autonomy*

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!



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.

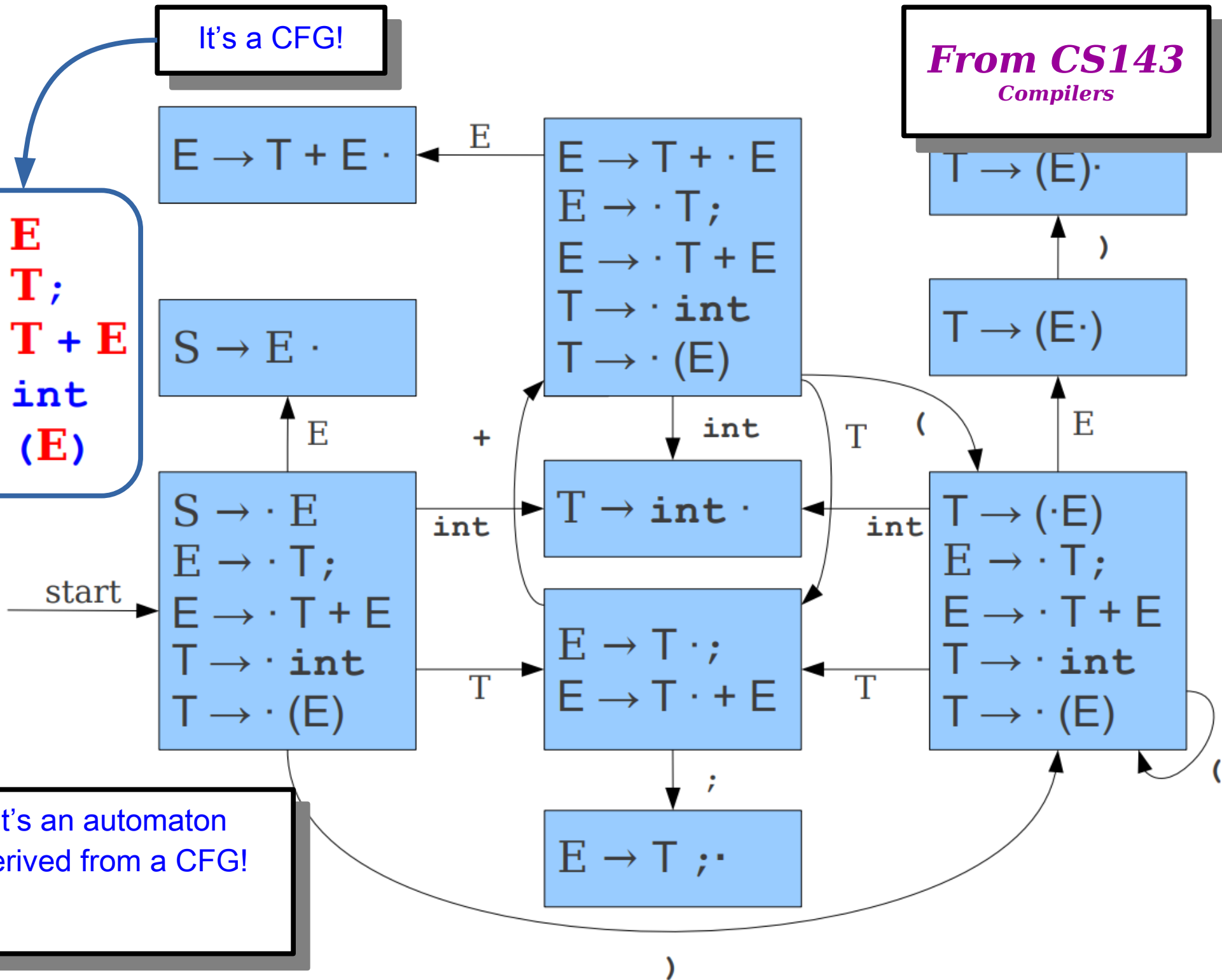
Hint: let $H : X \times Y \rightarrow \{0, 1, \dots, 2^n - 1\}$ be a collision-resistant hash function. Construct a new hash function $H' : X \times Y \rightarrow \{0, 1, \dots, 2^m - 1\}$ (where m may be greater than n) that is also collision resistant, but for a fixed difficulty D (say, $D = 2^{32}$) 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
Compilers*

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

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

It's functions with
specific
properties!

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

From CS161
Design and Analysis of
Algorithms

$O(\dots)$ 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,

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

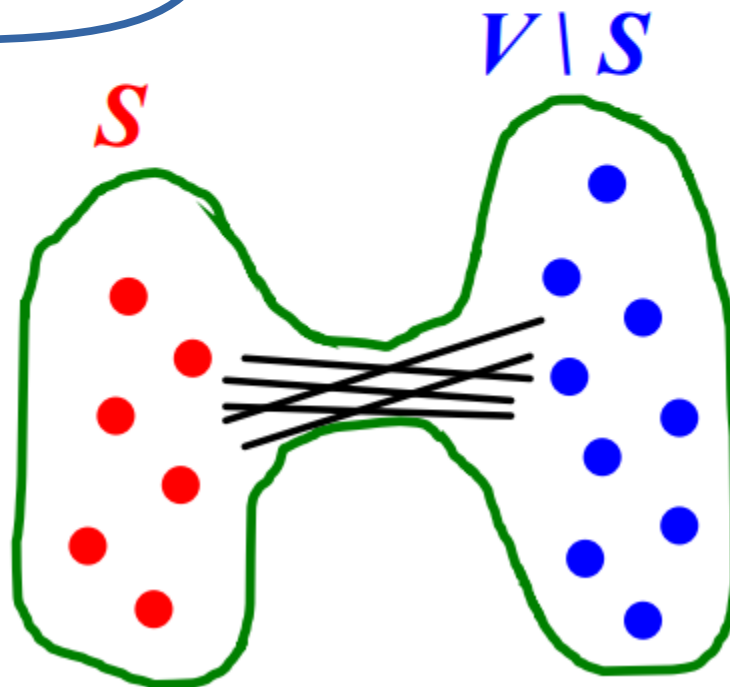
It's FOL and functions!

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

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

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:

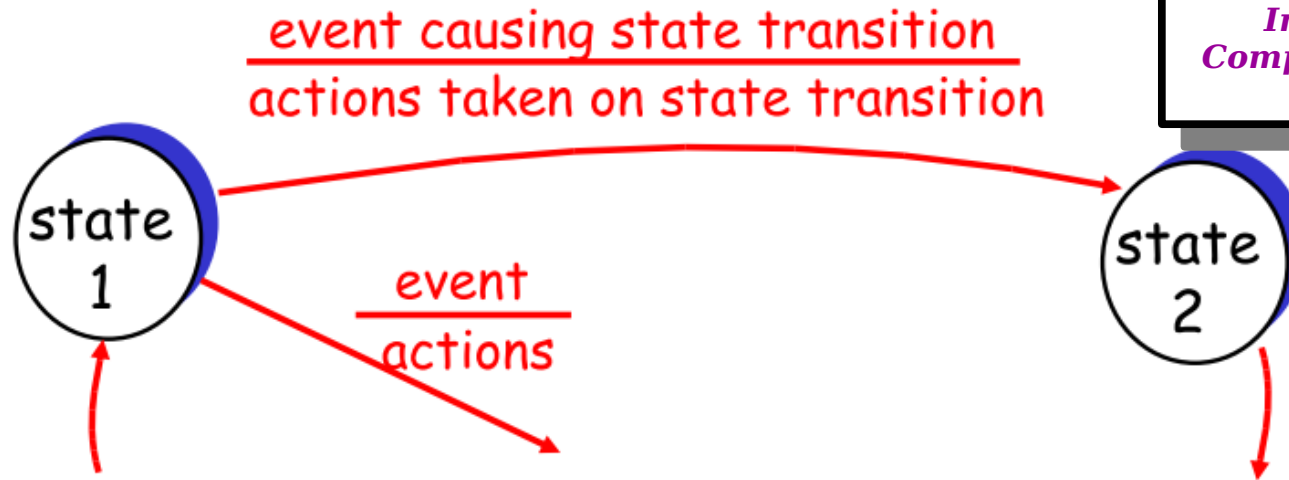
Type $\tau ::=$	int	integer
	$\tau_1 \rightarrow \tau_2$	function
Expression $e ::=$	x	variable
	n	integer
	$e_1 \oplus e_2$	binary operation
	$\lambda (x : \tau) . e$	function
	$e_1 e_2$	application
Binop $\oplus ::=$	+ - * /	

It's a
CFG!

First, we introduce a language of types, indicated by the variable tau (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:

Finite State Machines

From CS144
Introduction to
Computer Networking



- **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

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!

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

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 have the foundation
to tackle problems from all of
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

Theory

- CS154 - Automata
- CS161 - Algorithms
- CS229M - ML Theory
- CS250 - Codes
- CS255 - Cryptography
- CS259Q - Quantum Com.
- CS265 - Randomized Alg.

Applications

- CS112 - Operating Systems
- CS124 - LLMs
- CS131 - Computer Vision
- CS143 - Compilers
- CS224W - ML on Graphs
- CS242 - Prog. Languages
- CS243 - Prog. Optimization
- CS246 - Mining Data Sets
- CS251 - Blockchain

Your Questions

Thanks!