Announcements

- Problem Set 7 was due at 5:30 PM. Solutions are available on the course website.

Congratulations – you're done with CS103 problem sets!

- Problem Set 6 has been graded, I'm calibrating some questions with the CAs. Grades will be released on Gradescope tonight.

Please evaluate this course on Axess.

Your feedback really makes a difference.

Final Exam Logistics

- Our final exam will be on Saturday, August 17th from 7:00 – 10:00 PM in Hewlett 201 (same as the lectures and the midterm).
- Exam is the same format as the midterm: 3 hours, open notes, closed communication with other humans/AI.
- If you have OAE accommodations, you should have heard from us already about exam room and time logistics.
- The exam is cumulative and covers Lectures 00 – 16 as well as PS1 – PS6.
- ***Best of luck on the exam – you've got this!***

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
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
Regular Expressions
State Elimination

Distinguishability
Myhill-Nerode Theorem
Nonregular Languages
Context-Free Grammars
Brzozowski's Theorem
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**
$\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ 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.

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 must satisfy the following **correctness property**: for all keys k and all messages m, we have

$$\boxed{D(k,\ E(k,\ m)\ ) = m.}$$

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:

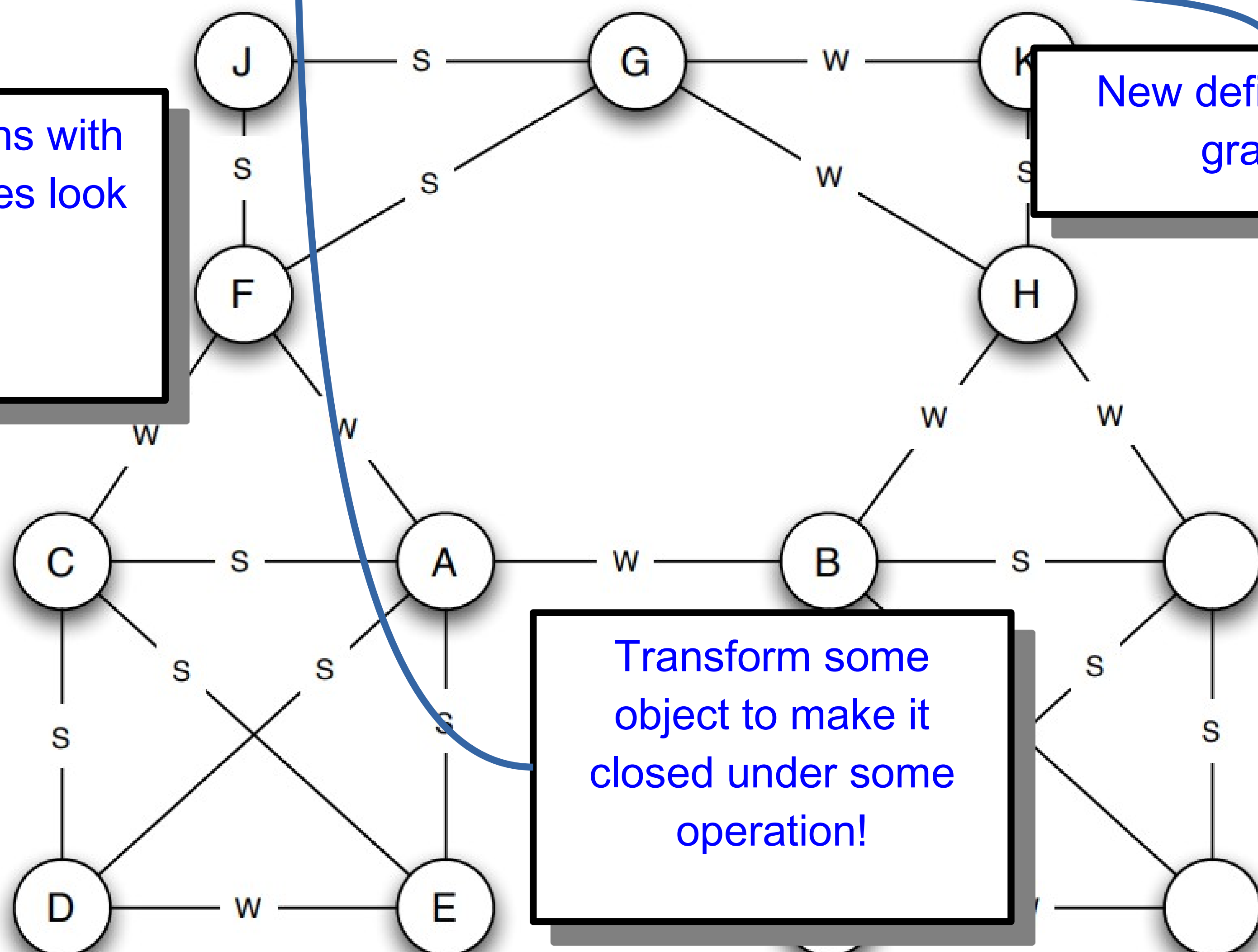
$$E : \mathcal{K} \times \mathcal{M} \to \mathcal{C},$$
$$D : \mathcal{K} \times \mathcal{C} \to \mathcal{M}.$$

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!

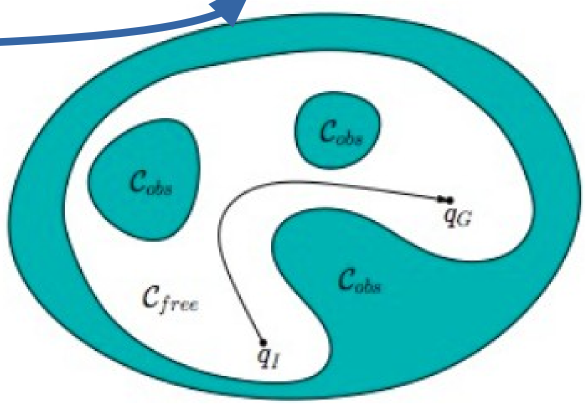
From CS237A

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

- 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

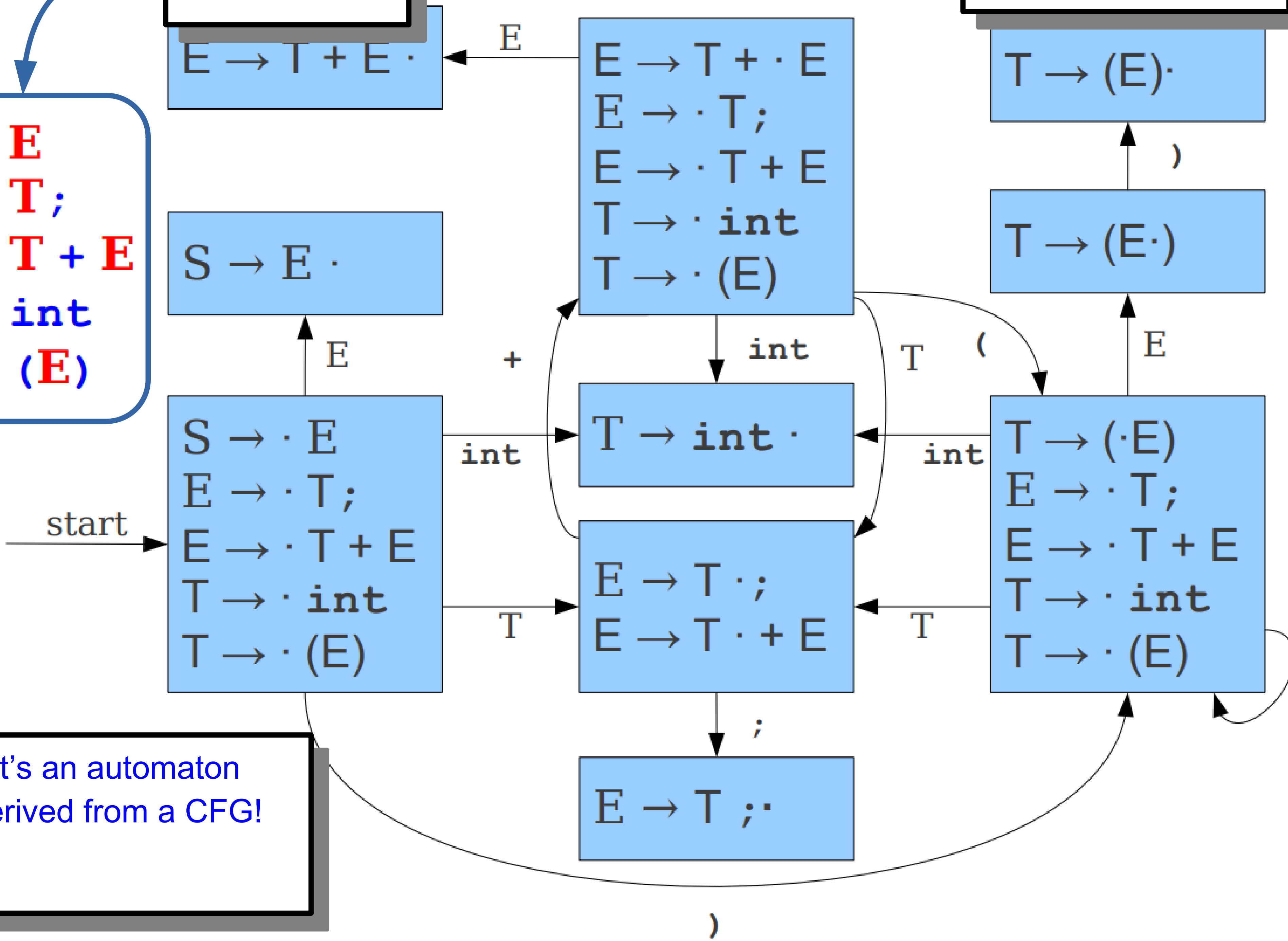
$S \rightarrow E$

$E \rightarrow T\,;$

$E \rightarrow T + E$

$T \rightarrow \texttt{int}$

$T \rightarrow (E)$



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!

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,

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

It's FOL and functions!

From CS224W

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

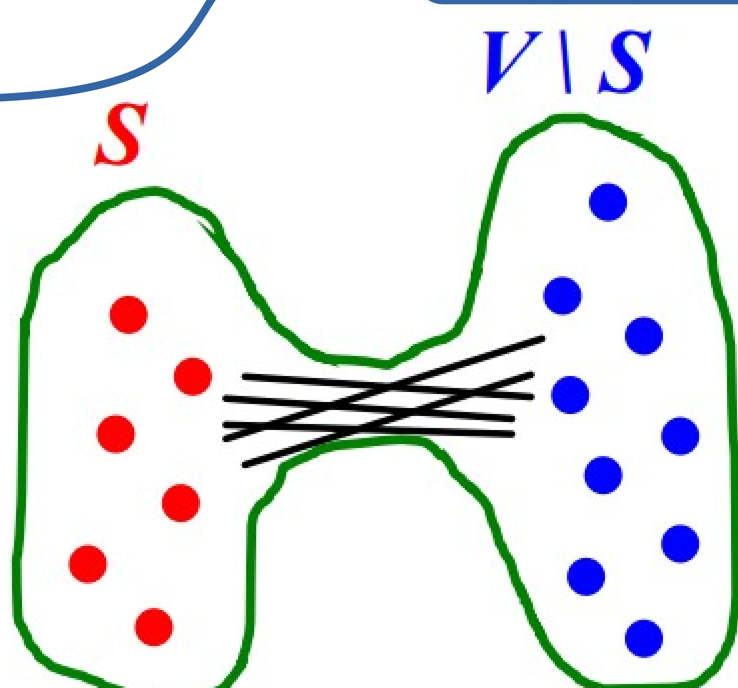
  $\#$ of edges leaving $S \geq \alpha \cdot \min(|S|, |V \setminus S|)$

- **Or equivalently:**

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

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:

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

It's a CFG!

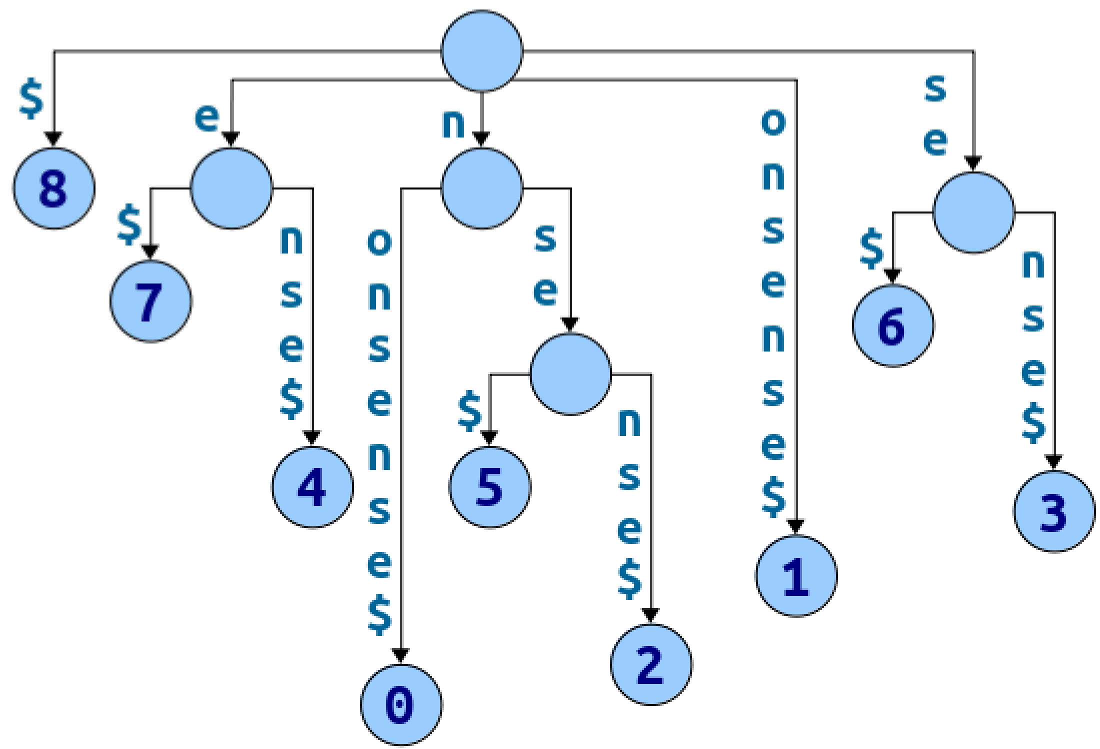
First, we introduce a language of types, indicated by the variable tau (τ). A type is either an integer, or a function from an input type τ_1 to an output type τ_2. Then we extend our untyped lambda calculus with the same arithmetic language from the first lecture (numbers and binary operators)[4]. Usage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

Definitions in terms of strings!

From CS166

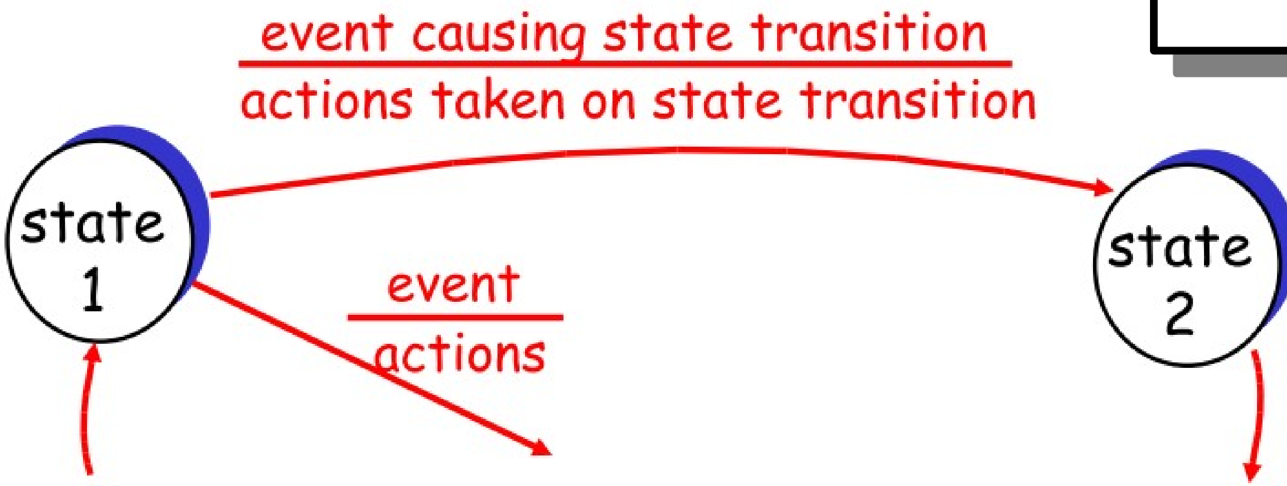
The Anatomy of a Suffix Tree

- 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

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!

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

Computability

Graphs

Functions

Set Theory

Number Theory

Applications

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

Languages / Automata

Graphs

Functions

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?