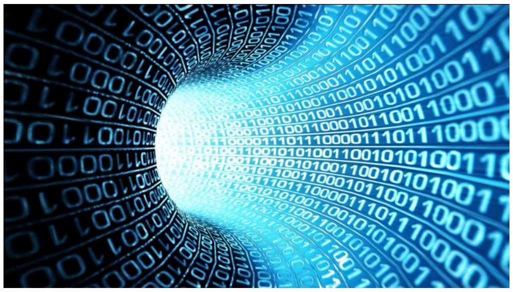
15-251: Great Ideas in Theoretical Computer Science

Lecture 2: Strings, Languages, Encodings, Problems



Aug 30th, 2018

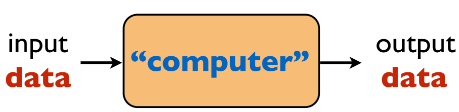
Objects/concepts we want to study and understand

↓

Mathematical model (formal, precise definitions)

↓

Mathematically/rigorously prove facts/theorems



Computation: manipulation of **data**.

How do we mathematically/formally represent **data**?

We have already done it for communication purposes.

e.g. written communication:

 → "apple"

 → "car"

 → "happy"

→ "three" or "3"

English alphabet

$\Sigma = \{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z\}$

Turkish alphabet

$\Sigma = \{a,b,c,ç,d,e,f,g,\bar{g},h,ı,i,j,k,l,m,n,o,ö,p,r,s,ş,t,u,ü,v,y,z\}$

What if we had more symbols?

What if we had less symbols?

Binary alphabet

$\Sigma = \{0, 1\}$

---

alphabet:

symbol/**character**:

string/**word**:

Length of a string s:

Back to Written English Example

$\Sigma = \{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z\}$

Objects/concepts of interest		String encoding
	⟶	apple
	⟶	car
	⟶	happy

Does every object have a corresponding encoding?

Can two objects have the same encoding?

Does every string correspond to a valid encoding?

encoding:

Examples

$A = \mathbb{N}$

Does Σ affect "encodability"?

Examples

$A = \mathbb{Z}$

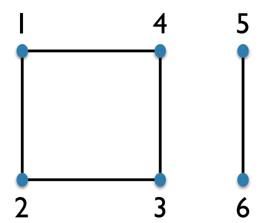
Examples

$A = \mathbb{N} \times \mathbb{N}$

Examples

$A =$ all undirected graphs

G

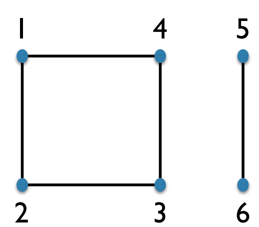


$\langle G \rangle =$

Examples

$A =$ all undirected graphs

G



	1	2	3	4	5	6
1	0	1	0	1	0	0
2	1	0	1	0	0	0
3	0	1	0	1	0	0
4	1	0	1	0	0	0
5	0	0	0	0	0	1
6	0	0	0	0	1	0

$\langle G \rangle =$

Examples

$A =$ all Python functions

```
def isPrime(N):
    if (N < 2):
        return False
    for factor in range(2, N):
        if (N % factor == 0):
            return False
    return True
```

$\langle \text{isPrime} \rangle =$

"def isPrime(N):\n   if (N < 2):\n   return False\n   for factor in range(2, N):\n   if (N % factor == 0):\n   return False\n   return True"

Does $|\Sigma|$ matter?

Going from $|\Sigma| = k$ to $|\Sigma'| = 2$:

Does $|\Sigma|$ matter?

$A = \mathbb{N}$	**Binary**	**vs**	**Unary**
0	0		ϵ
1	1		1
2	10		11
3	11		111
4	100		1111
5	101		11111
6	110		111111
7	111		1111111
8	1000		11111111
9	1001		111111111
10	1010		1111111111
11	1011		11111111111
12	1100		111111111111

Does $|\Sigma|$ matter?

Binary vs Unary

n has length in **binary**

n has length in **unary**

n has length in **base** k

Which sets are encodable?

Data is represented as finite length **strings** over some finite alphabet.

Reasoning about computation requires reasoning about **strings**.

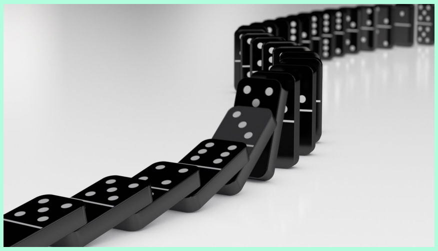
Induction

(powerful tool for understanding recursive structures)

Induction Review

Domino Principle

Line up any number of dominos in a row,
knock the first one over and they will all fall.



Induction Review

Domino Principle

Line up an infinite row of dominoes,
one domino for each natural number.
Knock the first one over and they will all fall.

Proof: Proof by contradiction: suppose they don't all fall.
Let **k** be the *lowest numbered domino* that remains standing.
Domino **k-1** did fall. But then **k-1** knocks over **k**, and **k** falls.
So **k** stands and falls, which is a contradiction.

Induction Review

Mathematical induction:

statements proved instead of dominoes fallen

Infinite sequence of dominoes	Infinite sequence of statements: $S_0, S_1, S_2, \ldots$
F_k = "domino k fell"	F_k = "S_k proved"

Establish: 1. F_0
2. for all k, $F_0, F_1, \ldots, F_k \Longrightarrow F_{k+1}$

Conclude: F_k is true for all k.

Different ways of packaging inductive reasoning

STRONG INDUCTION

METHOD OF MIN COUNTER-EXAMPLE

INVARIANT INDUCTION

STRUCTURAL INDUCTION

...

Structural Induction

Induction on objects with a recursive structure.

- arrays/lists
- strings
- graphs

⋮

Structural Induction

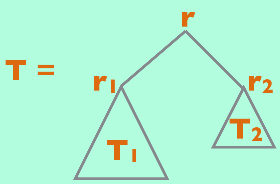
Recursive definition of a **string** over Σ :

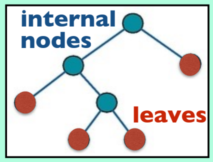
- the empty sequence ϵ is a string.
- if x is a string and $a \in \Sigma$, then ax is a string.

Structural Induction

Recursive definition of a **rooted binary tree**:

- a single node **r** is a binary tree with root **r**.
- if T_1 and T_2 are binary trees with roots r_1 and r_2, then **T** which has a node **r** adjacent to r_1 and r_2 is a binary tree with root **r**.





Every node has 0 or 2 children.

Structural Induction

Proposition: Let **T** be a binary tree.
Let L_T = # leaves in **T**.
Let I_T = # internal nodes in **T**.
Then $L_T = I_T + 1$.

Structural Induction

Proof (by structural induction):

Structural Induction

The outline of structural induction:

Base step**:** check statement true for base case(s) of def'n.

Recursive/induction step:
prove statement holds for **new objects** created by the recursive rule, assuming it holds for **old objects** used in the recursive rule.

Structural Induction

Why is that valid?

Usually another explicit parameter can be used to induct on.

Previous example: could induct on the parameter **height**.

Structural Induction

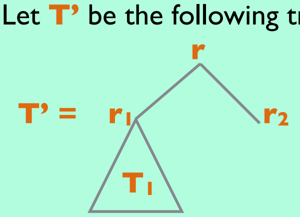
Be careful!
What is wrong with the following argument?

Strong induction on height.

Base case true.

Take an arbitrary binary tree **T** of height **h**.

Let **T'** be the following tree of height **h+1**:



blah blah blah

Therefore statement true for **T'** of height **h+1**.

Structural Induction

Another example with strings:

Let $L \subseteq \{0,1\}^*$ be recursively defined as follows:

- $\epsilon \in L$;
- if $x, y \in L$, then $0x1y0 \in L$.

Prove that for any $w \in L$, $\#(0, w) = 2 \cdot \#(1, w)$.

number of 0's in w

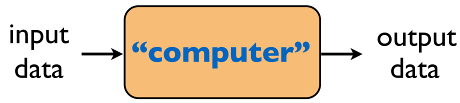
number of 1's in w

Structural Induction

Proof (by structural induction):

Back to string encodings

First Few Weeks



What is computation?

What is an algorithm?

How can we mathematically define them?

Seen so far:

Can encode/represent any kind of data
(numbers, text, pairs of numbers, graphs, images, etc...)
with a **finite length (binary) string**.

Before we define **algorithm** formally,
we should define **computational problem** formally.

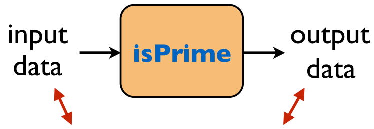
An **algorithm** *solves* a **computational problem**.

Example description of a **computational problem**:

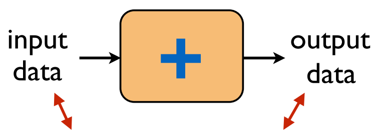
Given a natural number **N**, output *True* if **N** is prime,
and output *False* otherwise.

Example **algorithm** *solving* it:

```
def isPrime(N):
  if (N < 2): return False
  for factor in range(2, N):
    if (N % factor == 0): return False
  return True
```



Instance	Solution
0	No
1	No
2	Yes
3	Yes
4	No
⋮	⋮
251	Yes
⋮	⋮



Instance	Solution
0, 0	0
0, 1	1
1, 1	2
2, 2	4
2, 3	5
10, 1	11
100, 99	199
⋮	⋮

Instance

["vanilla", "mind", "Anil", "yogurt", "doesn't"]

Solution

["Anil", "doesn't", "mind", "vanilla", "yogurt"]

A computational problem is a function

$$f : I \to S \ .$$

$I =$ set of possible input objects (called instances)

$S =$ set of possible output objects (called solutions)

But in TCS, we don't deal with arbitrary objects,
we deal with **strings** (encodings).

Technicality:
What if $w \in \Sigma^*$ does not correspond to an encoding of an instance?

In TCS, there is only one type of data:

string

IMPORTANT DEFINITIONS

IMPORTANT RELATIONSHIP

There is a one-to-one correspondence between decision problems and languages.

Our focus will be on languages!
(decision problems)

---

computational problem
≈
corresponding decision problem

Integer factorization problem:

Given as input a natural number **N**, output its prime factorization.

Decision version:

Given as input natural numbers **N** and **k**,
does **N** have a factor between **1** and **k**?

INTERESTING QUESTIONS WE WILL EXPLORE ABOUT COMPUTATION

Are all languages computable/decidable?

If not, how can we prove that a language is not decidable?

How do we measure complexity of algorithms deciding languages?

How do we classify languages according to resources needed to decide them?

P = NP?