CS 301

Lecture 17 – Church–Turing thesis

Stephen Checkoway

March 19, 2018

An abridged modern history of formalizing algorithms

An algorithm is a finite, unambiguous sequence of steps for solving a problem

[1] Polynomial equation with integer coefficients

[2] Statements with $\forall$ and $\exists$

An abridged modern history of formalizing algorithms

An algorithm is a finite, unambiguous sequence of steps for solving a problem

The name *algorithm* comes from 9th Century Iranian mathematician Muḥammad ibn Mūsā al'Khwārizmī

[1] Polynomial equation with integer coefficients

[2] Statements with $\forall$ and $\exists$

An abridged modern history of formalizing algorithms

An algorithm is a finite, unambiguous sequence of steps for solving a problem

The name *algorithm* comes from 9th Century Iranian mathematician Muḥammad ibn Mūsā al'Khwārizmī

In the 17th century, German mathematician Gottfried Leibniz wanted to build a machine that could determine if mathematical statements were true or false

[1] Polynomial equation with integer coefficients

[2] Statements with $\forall$ and $\exists$

An abridged modern history of formalizing algorithms

An algorithm is a finite, unambiguous sequence of steps for solving a problem

The name *algorithm* comes from 9th Century Iranian mathematician Muḥammad ibn Mūsā al'Khwārizmī

In the 17th century, German mathematician Gottfried Leibniz wanted to build a machine that could determine if mathematical statements were true or false

In 1900, German mathematician David Hilbert posed a list of 23 open problems, the tenth asks for a general algorithm by which any Diophantine equation[1] could be solved

[1] Polynomial equation with integer coefficients

[2] Statements with $\forall$ and $\exists$

An abridged modern history of formalizing algorithms

An algorithm is a finite, unambiguous sequence of steps for solving a problem

The name *algorithm* comes from 9th Century Iranian mathematician Muḥammad ibn Mūsā al'Khwārizmī

In the 17th century, German mathematician Gottfried Leibniz wanted to build a machine that could determine if mathematical statements were true or false

In 1900, German mathematician David Hilbert posed a list of 23 open problems, the tenth asks for a general algorithm by which any Diophantine equation[1] could be solved

In 1928, Hilbert posed 3 more problems, the third of which became known as Hilbert's Entscheidungsproblem (decision problem). It asks for an algorithm which takes as input a statement in first-order logic[2]

[1] Polynomial equation with integer coefficients

[2] Statements with $\forall$ and $\exists$

Formalizing algorithms

In 1933, Austrian-American mathematician Kurt Gödel and French mathematician Jacques Herbrand[3] defined the class of general recursive functions; these were designed to be intuitively "computable"

[3] Herbrand had actually died at age 23 several years earlier

[4] Functional programming owes much to λ-calculus and lambdas and closures in many programming languages derive from this

Formalizing algorithms

In 1933, Austrian-American mathematician Kurt Gödel and French mathematician Jacques Herbrand[3] defined the class of general recursive functions; these were designed to be intuitively "computable"

In 1936, American mathematician Alonzo Church defined the λ-calculus as a model of computable functions[4]

[3] Herbrand had actually died at age 23 several years earlier

[4] Functional programming owes much to λ-calculus and lambdas and closures in many programming languages derive from this

Formalizing algorithms

In 1933, Austrian-American mathematician Kurt Gödel and French mathematician Jacques Herbrand[3] defined the class of general recursive functions; these were designed to be intuitively "computable"

In 1936, American mathematician Alonzo Church defined the λ-calculus as a model of computable functions[4]

Independently, in 1936, English mathematician Alan Turing defined his machines

[3] Herbrand had actually died at age 23 several years earlier

[4] Functional programming owes much to λ-calculus and lambdas and closures in many programming languages derive from this

Formalizing algorithms

In 1933, Austrian-American mathematician Kurt Gödel and French mathematician Jacques Herbrand[3] defined the class of general recursive functions; these were designed to be intuitively "computable"

In 1936, American mathematician Alonzo Church defined the λ-calculus as a model of computable functions[4]

Independently, in 1936, English mathematician Alan Turing defined his machines

Church and Turing proved that all three of these models of computation are equivalent: A function is λ-computable iff it is computable by a Turing machine iff it is general recursive

[3] Herbrand had actually died at age 23 several years earlier

[4] Functional programming owes much to λ-calculus and lambdas and closures in many programming languages derive from this

Church–Turing thesis

The Church–Turing thesis is that the intuitive notion of an algorithm (called an "effectively calculable function") is equivalent to a Turing machine

Hilbert's problems

Hilbert framed his tenth problem in 1900 and his Entscheidungsproblem in 1928 as a positive: Give an algorithm to solve the problems

As late as 1930, he didn't seem to realize that there would be any unsolvable problems

Hilbert's problems

Hilbert framed his tenth problem in 1900 and his Entscheidungsproblem in 1928 as a positive: Give an algorithm to solve the problems

As late as 1930, he didn't seem to realize that there would be any unsolvable problems

Unfortunately (depending on your point of view), both of these problems don't have general solutions

Types of problems

There are several types of problems we could consider, consider the problem of paths in a weighted, undirected graph G between two vertices u and v

Decision Is there some path between u and v in G of distance at most k?

Search What is the length of the shortest path between u and v in G?

Counting How many paths between u and v in G have distance at most k?

In this course, we're concerned with the simplest of these: decision problems

Decision problems as languages

We can frame decision problems as membership in a language.

What language corresponds to the decision problem: Is there some path between u and v in graph G of distance at most k?

Decision problems as languages

We can frame decision problems as membership in a language.

What language corresponds to the decision problem: Is there some path between u and v in graph G of distance at most k?

$$\{\langle G, u, v, k\rangle \mid G = (V, E) \text{ is an undirected graph, } u, v \in V, \text{ and there is a path from } u \text{ to } v \text{ of distance at most } k\}$$

$\langle G, u, v, k\rangle$ means some fixed representation of the graph G, vertices u and v, and natural number k

Representation

For the rest of the course, we're going to be working with languages whose elements are representations of mathematical objects like

- integers
- graphs
- finite automata
- regular expressions
- CFGs
- PDAs
- Turing machines
- finite sets of mathematical objects
- finite, ordered lists of mathematical objects

We need some way to represent these as strings over some alphabet

Explicit representation

Most of the time, we won't give an explicit representation

We'll mostly assume some representation exists and we can convert from one representation to another easily

Example: Graph representation

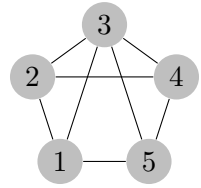
As a mathematical object, an undirected graph[5] is a pair $G = (V, E)$ where

V a finite set of vertices

E a finite set of pairs of vertices

We can specify the graph in a number of ways and easily convert between them

- Vertex and edge lists $G = (V, E)$ where $V = \{1, 2, 3, 4, 5\}$ and $E = \{(1, 2), (2, 3), (3, 4), (4, 5), (5, 1), (1, 3), (2, 4), (3, 5)\}$



[5] Review Sipser, chapter 0 as needed

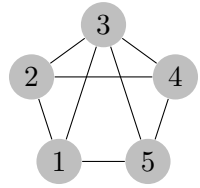
Example: Graph representation

As a mathematical object, an undirected graph[5] is a pair $G = (V, E)$ where

V a finite set of vertices

E a finite set of pairs of vertices

We can specify the graph in a number of ways and easily convert between them



- Vertex and edge lists $G = (V, E)$ where $V = \{1, 2, 3, 4, 5\}$ and $E = \{(1, 2), (2, 3), (3, 4), (4, 5), (5, 1), (1, 3), (2, 4), (3, 5)\}$
- Adjacency matrix $a_{ij} = 1$ if $(i, j) \in E$

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

[5] Review Sipser, chapter 0 as needed

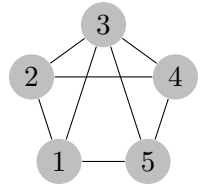
Example: Graph representation

As a mathematical object, an undirected graph[5] is a pair $G = (V, E)$ where

V a finite set of vertices

E a finite set of pairs of vertices

We can specify the graph in a number of ways and easily convert between them



- Vertex and edge lists $G = (V, E)$ where $V = \{1, 2, 3, 4, 5\}$ and $E = \{(1, 2), (2, 3), (3, 4), (4, 5), (5, 1), (1, 3), (2, 4), (3, 5)\}$
- Adjacency matrix $a_{ij} = 1$ if $(i, j) \in E$

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

- Adjacency list: $[\{2, 3, 5\}, \{1, 3, 4\}, \{1, 2, 4, 5\}, \{2, 3, 5\}, \{1, 3, 4\}]$

[5] Review Sipser, chapter 0 as needed

Explicit representation for a DFA

Consider $M = (Q, \Sigma, \delta, q_0, F)$ where $Q = \{1, 2, \ldots, m\}$, $\Sigma = \{1, 2, \ldots, n\}$, and $F = \{f_1, f_2, \ldots, f_k\}$

Build it up piece by piece using the alphabet $\Sigma = \{0, 1\}$

$\langle k \rangle = 0^k$ integers

Explicit representation for a DFA

Consider $M = (Q, \Sigma, \delta, q_0, F)$ where $Q = \{1, 2, \ldots, m\}$, $\Sigma = \{1, 2, \ldots, n\}$, and $F = \{f_1, f_2, \ldots, f_k\}$

Build it up piece by piece using the alphabet $\Sigma = \{0, 1\}$

$$\langle k \rangle = 0^k \qquad \text{integers}$$

$$\langle Q \rangle = 1\langle 1 \rangle 1\langle 2 \rangle 1 \cdots 1\langle m \rangle 1 \qquad \text{integers separated by 1s}$$

$$= 101001 \cdots 10^m 1$$

Explicit representation for a DFA

Consider $M = (Q, \Sigma, \delta, q_0, F)$ where $Q = \{1, 2, \ldots, m\}$, $\Sigma = \{1, 2, \ldots, n\}$, and $F = \{f_1, f_2, \ldots, f_k\}$

Build it up piece by piece using the alphabet $\Sigma = \{0, 1\}$

$$\langle k \rangle = 0^k \qquad \text{integers}$$

$$\langle Q \rangle = 1\langle 1 \rangle 1\langle 2 \rangle 1 \cdots 1\langle m \rangle 1 \qquad \text{integers separated by 1s}$$

$$= 101001 \cdots 10^m 1$$

$$\langle \Sigma \rangle = 1\langle 1 \rangle 1\langle 2 \rangle 1 \cdots 1\langle n \rangle 1 \qquad \text{integers separated by 1s}$$

$$= 101001 \cdots 10^n 1$$

Explicit representation for a DFA

Consider $M = (Q, \Sigma, \delta, q_0, F)$ where $Q = \{1, 2, \ldots, m\}$, $\Sigma = \{1, 2, \ldots, n\}$, and $F = \{f_1, f_2, \ldots, f_k\}$

Build it up piece by piece using the alphabet $\Sigma = \{0, 1\}$

$$\langle k \rangle = 0^k \qquad \text{integers}$$

$$\langle Q \rangle = 1\langle 1 \rangle 1\langle 2 \rangle 1 \cdots 1\langle m \rangle 1 \qquad \text{integers separated by 1s}$$

$$= 101001 \cdots 10^m 1$$

$$\langle \Sigma \rangle = 1\langle 1 \rangle 1\langle 2 \rangle 1 \cdots 1\langle n \rangle 1 \qquad \text{integers separated by 1s}$$

$$= 101001 \cdots 10^n 1$$

$$\langle \delta(q, t) = r \rangle = \langle q \rangle 1 \langle t \rangle 1 \langle r \rangle$$

$$= 0^q 1 0^t 1 0^r$$

Explicit representation for a DFA

Consider $M = (Q, \Sigma, \delta, q_0, F)$ where $Q = \{1, 2, \ldots, m\}$, $\Sigma = \{1, 2, \ldots, n\}$, and $F = \{f_1, f_2, \ldots, f_k\}$

Build it up piece by piece using the alphabet $\Sigma = \{0, 1\}$

$$\langle k \rangle = 0^k \qquad \text{integers}$$

$$\langle Q \rangle = 1\langle 1 \rangle 1\langle 2 \rangle 1 \cdots 1\langle m \rangle 1 \qquad \text{integers separated by 1s}$$

$$= 101001 \cdots 10^m 1$$

$$\langle \Sigma \rangle = 1\langle 1 \rangle 1\langle 2 \rangle 1 \cdots 1\langle n \rangle 1 \qquad \text{integers separated by 1s}$$

$$= 101001 \cdots 10^n 1$$

$$\langle \delta(q, t) = r \rangle = \langle q \rangle 1 \langle t \rangle 1 \langle r \rangle$$

$$= 0^q 1 0^t 1 0^r$$

$$\langle \delta \rangle = 11\langle \delta(1, 1) = r_{11} \rangle 11 \cdots 11 \langle \delta(q, s) = r_{qs} \rangle 11$$

Explicit representation for a DFA

Consider $M = (Q, \Sigma, \delta, q_0, F)$ where $Q = \{1, 2, \ldots, m\}$, $\Sigma = \{1, 2, \ldots, n\}$, and $F = \{f_1, f_2, \ldots, f_k\}$

Build it up piece by piece using the alphabet $\Sigma = \{0, 1\}$

$$\langle k \rangle = 0^k \qquad \text{integers}$$

$$\langle Q \rangle = 1\langle 1 \rangle 1\langle 2 \rangle 1 \cdots 1\langle m \rangle 1 \qquad \text{integers separated by 1s}$$

$$= 101001 \cdots 10^m 1$$

$$\langle \Sigma \rangle = 1\langle 1 \rangle 1\langle 2 \rangle 1 \cdots 1\langle n \rangle 1 \qquad \text{integers separated by 1s}$$

$$= 101001 \cdots 10^n 1$$

$$\langle \delta(q,t) = r \rangle = \langle q \rangle 1 \langle t \rangle 1 \langle r \rangle$$

$$= 0^q 1 0^t 1 0^r$$

$$\langle \delta \rangle = 11\langle \delta(1,1) = r_{11} \rangle 11 \cdots 11 \langle \delta(q,s) = r_{qs} \rangle 11$$

$$\langle q_0 \rangle = 0^{q_0} \qquad \text{integer}$$

Explicit representation for a DFA

Consider $M = (Q, \Sigma, \delta, q_0, F)$ where $Q = \{1, 2, \ldots, m\}$, $\Sigma = \{1, 2, \ldots, n\}$, and $F = \{f_1, f_2, \ldots, f_k\}$

Build it up piece by piece using the alphabet $\Sigma = \{0, 1\}$

$$\langle k \rangle = 0^k \qquad \text{integers}$$

$$\langle Q \rangle = 1\langle 1 \rangle 1\langle 2 \rangle 1 \cdots 1\langle m \rangle 1 \qquad \text{integers separated by 1s}$$
$$= 101001 \cdots 10^m 1$$

$$\langle \Sigma \rangle = 1\langle 1 \rangle 1\langle 2 \rangle 1 \cdots 1\langle n \rangle 1 \qquad \text{integers separated by 1s}$$
$$= 101001 \cdots 10^n 1$$

$$\langle \delta(q,t) = r \rangle = \langle q \rangle 1 \langle t \rangle 1 \langle r \rangle$$
$$= 0^q 1 0^t 1 0^r$$

$$\langle \delta \rangle = 11\langle \delta(1,1) = r_{11} \rangle 11 \cdots 11 \langle \delta(q,s) = r_{qs} \rangle 11$$

$$\langle q_0 \rangle = 0^{q_0} \qquad \text{integer}$$

$$\langle F \rangle = 1\langle f_1 \rangle 1 \langle f_2 \rangle 1 \cdots 1 \langle f_k \rangle 1 \qquad \text{integers separated by 1s}$$

Explicit representation for a DFA

Consider $M = (Q, \Sigma, \delta, q_0, F)$ where $Q = \{1, 2, \ldots, m\}$, $\Sigma = \{1, 2, \ldots, n\}$, and $F = \{f_1, f_2, \ldots, f_k\}$

Build it up piece by piece using the alphabet $\Sigma = \{0, 1\}$

$$\langle k \rangle = 0^k \qquad \text{integers}$$

$$\langle Q \rangle = 1\langle 1 \rangle 1\langle 2 \rangle 1 \cdots 1\langle m \rangle 1 \qquad \text{integers separated by 1s}$$

$$= 101001 \cdots 10^m 1$$

$$\langle \Sigma \rangle = 1\langle 1 \rangle 1\langle 2 \rangle 1 \cdots 1\langle n \rangle 1 \qquad \text{integers separated by 1s}$$

$$= 101001 \cdots 10^n 1$$

$$\langle \delta(q,t) = r \rangle = \langle q \rangle 1 \langle t \rangle 1 \langle r \rangle$$

$$= 0^q 1 0^t 1 0^r$$

$$\langle \delta \rangle = 11\langle \delta(1,1) = r_{11} \rangle 11 \cdots 11 \langle \delta(q,s) = r_{qs} \rangle 11$$

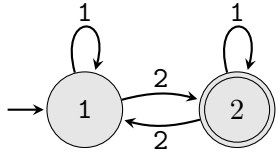
$$\langle q_0 \rangle = 0^{q_0} \qquad \text{integer}$$

$$\langle F \rangle = 1\langle f_1 \rangle 1 \langle f_2 \rangle 1 \cdots 1 \langle f_k \rangle 1 \qquad \text{integers separated by 1s}$$

$$\langle M \rangle = 111\langle Q \rangle 111 \langle \Sigma \rangle 111 \langle \delta \rangle 111 \langle q_0 \rangle 111 \langle F \rangle 111$$

Explicit representation

Consider the simple DFA:



$\langle Q \rangle$ = 1 0 1 00 1

$\langle \Sigma \rangle$ = 1 0 1 00 1

$\langle \delta \rangle$ = 11 01010 11 0100100 11 0010100 11 0010010 11

$\langle q_0 \rangle$ = 0

$\langle F \rangle$ = 1 00 1

$\langle M \rangle$ = 111 $\underbrace{101001}_{Q}$ 111 $\underbrace{101001}_{\Sigma}$

111 $\underbrace{11010101101001001100101001100100101 1}_{\delta}$

111 $\underbrace{0}_{q_0}$ 111 $\underbrace{1001}_{F}$ 111

Representations of other objects

We're not going to bother giving explicit representations of other objects

But we could if we really needed to and that's enough for our purposes

Taking representations as inputs to TMs

We're going to be dealing with languages like

$$A_{\text{DFA}} = \{\langle M, w\rangle \mid M \text{ is a DFA, } w \text{ is a string and } w \in L(M)\}$$

This language is related to the acceptance problem for DFAs which involves testing if the DFA M accepts a string w

Taking representations as inputs to TMs

We're going to be dealing with languages like

$$A_{\text{DFA}} = \{\langle M, w\rangle \mid M \text{ is a DFA, } w \text{ is a string and } w \in L(M)\}$$

This language is related to the acceptance problem for DFAs which involves testing if the DFA M accepts a string w

Theorem

The language A_{DFA} is decidable.

Taking representations as inputs to TMs

We're going to be dealing with languages like

$$A_{\text{DFA}} = \{\langle M, w\rangle \mid M \text{ is a DFA, } w \text{ is a string and } w \in L(M)\}$$

This language is related to the acceptance problem for DFAs which involves testing if the DFA M accepts a string w

Theorem

The language A_{DFA} is decidable.

Proof.

Let's build a TM to decide A_{DFA}.

M = "On input $\langle B, w\rangle$ where B is a DFA and w is a string,

1. Simulate B on input w.
2. If the simulation ends in an accept state, *accept*. If it ends in a nonaccepting state, *reject*."

□

Some notes about that construction

First, the form of the construction:
M = "On input $\langle \_\_ \rangle$,

1. First step.
2. Second step.
3. Last step."

Your constructions should follow exactly this form

Some notes about that construction

First, the form of the construction:
M = "On input $\langle \_\_ \rangle$,

1. First step.
2. Second step.
3. Last step."

Your constructions should follow exactly this form

Second, note that the input to this TM is supposed to be a representation of a DFA and a string

The representation is not the same thing as the mathematical object

If M is a DFA, then $\langle M \rangle$ is a string

TMs take strings as input, nothing else

How does this construction actually work?

First, the TM M is going to check that the input is a valid representation of a DFA and a string
Which representation is that?

Then, it needs to simulate the DFA by keeping track of the DFA's initial state and how much of the input it has read so far

Initially, the DFA starts in its initial state and it has read none of the input so far

The simulation proceeds step by step where M needs to compare the current state and the next input symbol to the representation of δ to find the next state

Next time

We'll discuss some more decidable languages related to

- the acceptance problem for other models of computation
- The emptiness problem for models of computation (E.g., is the language of a DFA the empty language?)
- The equivalence problem for models of computation (E.g., do two DFAs have the same language?)

Later, we'll discuss some related problems (and languages) that are *not* decidable!