**CS120: Intro. to Algorithms and their Limitations** Anshu & Hesterberg

# Lecture 17: The Church-Turing Thesis

*Harvard SEAS - Fall 2023* *2023-11-02*

## 1 Announcements

- Recommended reading: MacCormick §5.6–5.7, §7.7–7.9

## 2 Loose Ends: Efficient algorithm for 2-SAT

Our goal is to prove the following theorem:

**Theorem 2.1.** *There is an algorithm that solves the computational problem 2-SAT in time $O(nm)$, where $n$ is the number of variables and $m$ is the number of clauses.*

The algorithm is as follows.

```
1  TwoSATSolve($\phi$)
   Input   : A CNF $\phi$ with width 2
   Output  : A satisfying assignment, else $\perp$
2  Construct the implication graph $G = (V, E)$;
3  Set $(x_0, \ldots, x_{n-1})$ = (“unassigned”, . . . , “unassigned”);
4  foreach $i = 0, \ldots, n-1$ do
5      $\text{dist}_{1,i}$ =BFS($G, x_i, \neg x_i$)
6      $\text{dist}_{2,i}$ =BFS($G, \neg x_i, x_i$)


7  foreach $i = 0, \ldots, n-1$ do
8      if $x_i$ is assigned then $i = i + 1$ and GOTO line 8;
9      if $\text{dist}_{1,i} = \infty$ AND $\text{dist}_{2,i} < \infty$ then


10     if $\text{dist}_{1,i} < \infty$ AND $\text{dist}_{2,i} = \infty$ then


11     Delete all the edges incident to or from all assigned literals.
12 Assign the remaining variables such that the literals in each connected component of
     resulting graph $G$ get the same value.
13 return $(x_0, \ldots, x_{n-1})$;
```

**Runtime:**

**Correctness:**

## 3 Introduction to Limits of Computation

Thus far in CS 120, we've focused on what algorithms can do, or what they can do efficiently. In the remainder of the course, we'll talk about what algorithms can't do, or can't do efficiently.

In particular, recall Lecture 3's lemma about reductions:

**Lemma 3.1.** *Let $\Pi$ and $\Gamma$ be computational problems such that $\Pi \leq \Gamma$. Then:*

1. *If there exists an algorithm solving $\Gamma$, then there exists an algorithm solving $\Pi$.*

2. *If there does not exist an algorithm solving $\Pi$, then there does not exist an algorithm solving $\Gamma$.*

3. *If there exists an algorithm solving $\Gamma$ with runtime $g(n)$, and $\Pi \leq_{T,f} \Gamma$, then there exists an algorithm solving $\Pi$ with runtime $O(T(n) + g(f(n)))$.*

4. *If there does not exists an algorithm solving $\Pi$ with runtime $O(T(n)+g(f(n))$, and $\Pi \leq_{T,f} \Gamma$, then there does not exist an algorithm solving $\Gamma$ with runtime $O(g(n))$.*

In the last unit of the course, we'll use the item 2: we'll find a problem $\Pi$ which we can prove is not solved by any Word-RAM algorithm, then reduce $\Pi$ to other problems $\Gamma$ to prove that no Word-RAM algorithm solves them.

Similarly, in the upcoming second-last unit of the course, we'll use item 4: we'll assume that the problem $\Pi = SAT$ is not solved quickly by any Word-RAM algorithm, then reduce $SAT$ to other problems $\Gamma$ to prove that no Word-RAM algorithm solves them quickly.

Before we do so, let's consider how fundamental Word-RAM is to the statements above. That is, if we prove limitations of Word-RAM programs, are those limits specific to Word-RAM or are they more general/independent of technology? Could find substantially faster algorithms by choosing a different model of computation than Word RAM, like Python or Minecraft? The answer is conjectured to be "no".

To explain why, we'll first recall our simulation arguments which state that the same problems are solvable by Word-RAM programs, Python programs, and so on.

# 4 The Church–Turing Thesis

**Theorem 4.1** (Turing-equivalent models)**.** *If a computational problem* $\Pi$ *is solvable in one of the following models of computation, then it is solvable in all of them:*

*Moreover, there is an algorithm (e.g. a RAM program) that can transform a program in any of these models of computation into an equivalent program in any of the others.*

**The Church–Turing Thesis:** The equivalence of many disparate models of computation leads to the Church–Turing Thesis, which has (at least) two different variants:

1.

2.

This is not a precise mathematical claim, and thus cannot be formally proven, but it has stood the test of time very well, even in the face of novel technologies like quantum computers (which have yet to be built in a scalable fashion); every problem that can be solved by a quantum algorithm can also be solved by a RAM program, albeit much more slowly.

**Proof idea:**

**Simple and elegant models:**

**Input encodings:**

### 4.1 The Strong (or Extended) Church–Turing Thesis

The Church–Turing hypothesis only concerns problems solvable at all by these models of computation (Word-RAM programs, etc.). We haven't even seen any problems that are *not* solvable by Word-RAM programs—that will be a topic for the end of the course. There is, however, a stronger version of the Church–Turing hypothesis that also covers the efficiency with which we can solve problems.

Extended Church–Turing Thesis v1:

The Strong Church–Turing Thesis is not a precise mathematical claim, and thus cannot be formally proven. In fact, randomized algorithms, massively parallel computers, and quantum computers all could potentially provide an exponential savings in runtime. (For randomized algorithms, however, it is conjectured that they provide only a polynomial savings, as discussed in Lecture 8.)

If we modify the statement with some qualifiers, then these challenges no longer apply:

Extended Church–Turing Thesis v2:

"Deterministic" rules out both randomized and quantum computation, as both are inherently probabilistic. "Sequential" rules out parallel computation. This form of the Extended Church–Turing Thesis has stood the test of time for the approximately fifty years since it was formulated, even as computing technology has changed tremendously in that time.

Considering computational efficiency when comparing models of computation will be the subject of the next few lectures.