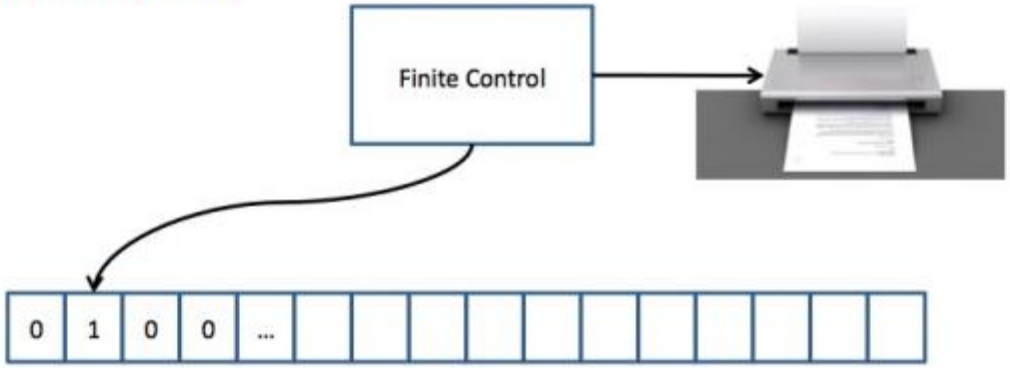
ENUMERATORS

- Remember we noted that some books used the term recursively enumerable for Turing-recognizable.
- This term arises from a variant of a TM called an enumerator.



- TM generates strings one by one.
- Everytime the TM wants to add a string to the list, it sends it to the printer.

Loosely defined, an enumerator is a Turing machine with an attached printer.

ENUMERATORS

- The enumerator E starts with a blank input tape.
- If it does not halt, it may print an infinite list of strings.
- The strings can be enumerated in any order; repetitions are possible.
- The language of the enumerator is the collection of strings it eventually prints out.

- We can assume that the enumerator E writes one string at a time over a tape (it can use a tape symbol # to separate strings).

Enumerators

Theorem

A language is Turing recognizable if and only if some enumerator enumerates it.

Proof.

The If-part: If an enumerator E enumerates the language A then a TM M recognizes A.

M = "On input w

1. Run E. Everytime E outputs a string, compare it with w.
2. If w ever appears in the output of E, *accept.*"

Clearly M accepts only those strings that appear on E's list.

□

The TM M accepts w only when E produces w as one of its output strings.

ENUMERATORS

THEOREM

A language is Turing recognizable if and only if some enumerator enumerates it.

PROOF.

The Only-If-part: If a TM *M* recognizes a language *A*, we can construct the following enumerator for *A*.

- For each possible string $s \in \Sigma^*$ we can verify whether M accepts s, if so output s on to the tape .

Following attempt, does not work.

- Assume that there is an enumerator which enumerates Σ^* in a standard order.
- For each possible string $s \in \Sigma^*$ we can verify whether M accepts s, if so output s on to the tape .
- The problem with this is, for some s the TM M can enter in to an infinite loop and never returns.
  - There may be other strings which are accepted by M but will never gets a chance to be verified (and thus never is outputted).

- A feasible way of doing this (without falling in to an infinite loop) is given in the next slide.
- Basic idea:
- For example if there are two strings w1 and w2 and one of them makes the TM to loop infinitely. Do the following.
  1. **k = 1**
  2. **Run TM for k steps on w1 and if accept occurs then output "accept" and stop.**
  3. **Run TM for k steps on w2 and if accept occurs then output "accept" and stop.**
  4. **k++; goto step 2.**

Enumerators

Theorem

A language is Turing recognizable if and only if some enumerator enumerates it.

Proof.

The Only-If-part: If a TM M recognizes a language A, we can construct the following enumerator for A. Assume $s_1, s_2, s_3, \ldots$ is a list of possible strings in Σ^*.

$E =$ "Ignore the input

1. Repeat the following for $i = 1, 2, 3, \ldots$
2. Run M for i steps on each input $s_1, s_2, s_3, \ldots s_i$.
3. If any computations accept, print out corresponding s_j."

If M accepts a particular string, it will appear on the list generated by E (in fact infinitely many times)

The Definition of Algorithm - History

- in 1900, Hilbert posed the following problem:

  "Given a polynomial of several variables with integer coefficients, does it have an integer root – an assignment of integers to variables, that make the polynomial evaluate to 0"

- For example, $6x^3yz^2 + 3xy^2 - x^3 - 10$ has a root at $x = 5, y = 3, z = 0$.
- Hilbert explicitly asked that an algorithm/procedure to be "devised". He assumed it existed; somebody needed to find it!
- 70 years later it was shown that no algorithm exists.
- The intuitive notion of an algorithm may be adequate for giving algorithms for certain tasks, but was useless for showing no algorithm exists for a particular task.

This is known as Hilbert's Tenth Problem.

The Definition of Algorithm - History

- In early 20^{th} century, there was no formal definition of an algorithm.
- In 1936, Alonzo Church and Alan Turing came up with formalisms to define algorithms. These were shown to be equivalent, leading to the

Church-Turing Thesis

Intutitive notion of algorithms $\equiv$ Turing Machine Algorithms

THE DEFINITION OF AN ALGORITHM

- Let $D = \{p \mid p \text{ is a polynomial with integral roots}\}$
- Hilbert's 10^{th} problem in TM terminology is "Is D decidable?" (No!)
- However D is Turing-recognizable!
- Consider a simpler version
  $D_1 = \{p \mid p \text{ is a polynomial over } x \text{ with integral roots}\}$
- M_1 = "The input is polynomial p over x.
  1. Evaluate p with x successively set to 0, 1, -1, 2, -2, 3, -3, ...
  2. If at any point, p evaluates to 0, *accept*."
- D_1 is actually decidable since only a finite number of x values need to be tested (math!)
- D is also recognizable: just try systematically all integer combinations for all variables.

- For D_1 (polynomial of single variable) there is a bound and we can abandon the search beyond that and declare "No".
- The roots of such a polynomial must lie between the values:

$$\pm k \frac{C_{max}}{C_1}$$

- k is the number of terms in the polynomial, cmax is the coefficient with the largest absolute value, and c1 is the coefficient of the highest order term.
- For D such a bound cannot exist (proof is given by Matijasevich (1971)).
  - So, if answer is "No" we enter in to an infinite loop.

In 1971, Yuri Matijasevich gave a resounding negative answer to Hilbert's tenth problem.

- This is called undecidability.
- Hilbert’s tenth problem is undecidable.

Decidability

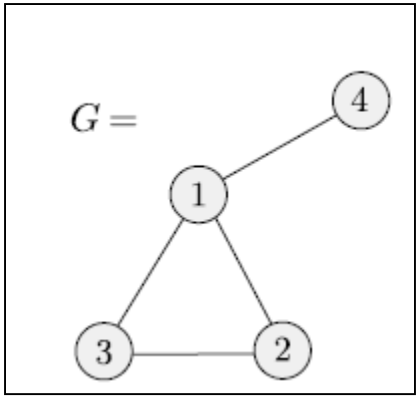
Theory behind existence of undecidable problems – Halting Problem --

Ref: https://www.andrew.cmu.edu/user/ko/pdfs/lecture-15.pdf

Describing Turing Machines and Their Inputs

- For the rest of the course we will have a rather standard way of describing TMs and their inputs.
- The input to TMs have to be strings.
- Every object O that enters a computation will be represented with an string $\langle O \rangle$, encoding the object.
- For example if G is a 4 node undirected graph with 4 edges $\langle O \rangle =$ `(1,2,3,4)((1,2),(2,3),(3,1),(1,4))`
- Then we can define problems over graphs,e.g., as:

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$$



Describing Turing Machines and Their Inputs

- A TM for this problem can be given as:
- M = "On input $\langle G \rangle$, the encoding of a graph G:
  1. Select the first node of G and mark it.
  2. Repeat 3) until no new nodes are marked
  3. For each node in G, mark it, if there is edge attaching it to an already marked node.
  4. Scan all the nodes in G. If all are marked, the *accept*, else *reject*"

- Important thing to understand is that a machine (computing machine) can be represented as a string in some language.
  - It will be a string over some alphabet.
  - This is, in some sense, equivalent to say that program is also data.

Representation Vs. Real Thing

- DNA represents a living thing (like a human-being).
- Some people believe (!) that this is a complete description of a human being.
- You can know his personality, you can even know what he will do in future?
  - Some Hollywood movies captured this idea.

- A representation in itself is lifeless.
- A program in itself cannot process the data.
- It has to be realized through a processing unit or DNA has to be realized through a laboratory test tube or so.

- But the processing unit, now can be entirely independent of the program.
- It can execute any program.

- We can give <program, data> to a general purpose computer which runs the program over the data and gives the output.

- A TM M also can be represented as a string.
- Call this string $< M >$.
- $< M >$ is like a program.
- Working of TM M on a string w can be realized by a *general purpose computer* like machine.
- This machine which can take input $< M, w >$ and outputs what the TM M does with w, is called the **U**niversal **T**uring **M**achine.

DECIDABILITY

- We investigate the power of algorithms to solve problems.
- We discuss certain problems that can be solved algorithmically and others that can not be.
- Why discuss unsolvability?
- Knowing a problem is unsolvable is useful because
  - you realize it must be simplified or altered before you find an algorithmic solution.
  - you gain a better perspective on computation and its limitations.