CPS220

Decidability

Review

We have learned about Turing Machines, which are Finite Automata (DFA, or NFA is equivalent) with an infinite tape that contains the input plus infinite blank space. The head of the TM can move left or right, and overwrite on any position.

A Turing Machine T recognizes a language L if T accepts every string in L, and never accepts a string not in L. However, a Turing Machine is not guaranteed to halt, if given an input not in L.

Turing Machines that halt in all inputs are called *deciders*. T decides a language L if T recognizes L, and halts in all inputs. That is, a decider T is guaranteed to either accept, or reject, and never fall into an infinite loop.

Languages recognized by a TM are called recognizable.
Languages decided by a TM are called decidable.

The Church-Turing Thesis formalized the notion of an *algorithm*, as a procedure that can be performed by a decider TM. Church's Thesis states that all sufficiently powerful and reasonable models of computation belong to the same class.

Turing Machine
- it's alright to use a high-level description instead of a detailed description
- TM take as input strings – which can encode any thing – numbers, graphs, etc.
- <O> notation for an encoded string

By accepting Church's Thesis we are able to prove that certain problems are unsolvable (undecidable) by any computer.

Decidable Languages

Consider the following language about DFAs:

A_{dfa} = { <M, w> | M is a DFA accepting input w}

Is this language decidable? That is, given as input, a DFA and a string, can we decide whether the DFA accepts the string? The answer is yes.

Theorem. A_{dfa} is decidable.

Proof. We construct a TM T, deciding A_{dfa}.

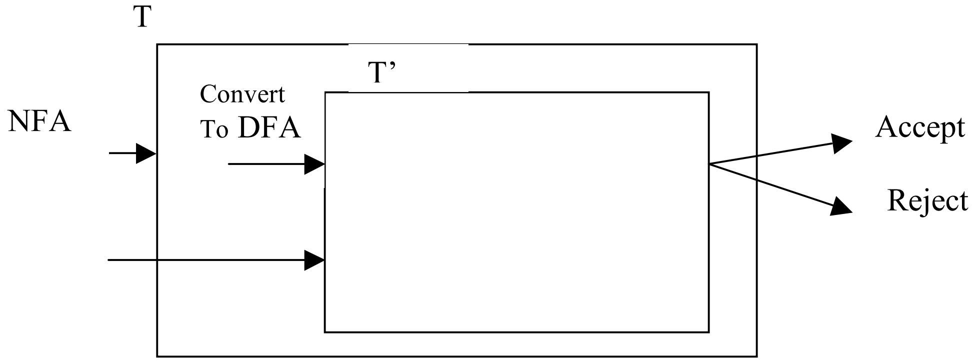
T = "On input <M,w>, after syntax check,
1. Simulate M on input w
2. If the simulation ends in an accept state, *accept* – otherwise, *reject*."

Input would be the 5 tuple description of the DFA – the TM would keep track of the current state and process the input tape based on transition function and finally accept if the string's final state is accept.

A_{nfa} = { <N, w> | N is an NFA accepting input w}

Theorem. A_{nfa} is decidable.

Proof. T' = " On input <N, w>, after syntax check,
1. Convert N to a DFA M.
2. Run T from proof above, on <M, w>
3. If T accepts, accept. Else, reject."



A_{REX} = { <R,w> | R is a regular expression that generates string w }

Theorem. A_{REX} is decidable.

E_{DFA} = { <M> | M is a DFA and L(M) = Ø }

Theorem. E_{DFA} is decidable.

Proof. The following machine decides it:

```
T := "On input  M, where M is a DFA,
        1. Mark the start state of M.
        2. Repeat until no new states are marked in an iteration through all states:
                - Mark any state that has a transition coming from a marked state
        3. If no accept state is marked, accept; otherwise reject."
```

EQ_{DFA} = { <M_1, M_2 > | M_1 & M_2 are DFAs and L(M_1) = L(M_2)}

Theorem. EQ_{DFA} is decidable.

Proof Idea: given two automata M_1 and M_2, the TM S will construct a new automaton that accepts the strings accepted by either M_1, or M_2, but not both. Then, S will simulate the machine T from previous proof, on automaton M.

(see page 169)

A_{CFG} = { <G, w> | G is a CFG deriving string w}

Theorem. A_{CFG} is decidable.

Proof. T = "On input <G, w>

1. Convert G to Chomsky Normal Form.
2. Test all possible derivations of length 2n-1, where n = |w|.
3. If any generate w, accept, otherwise reject."

Example:

$S0 \rightarrow AB|AB|BA$
$S \rightarrow AB|AB|BA$
$B \rightarrow b$
$A \rightarrow a$

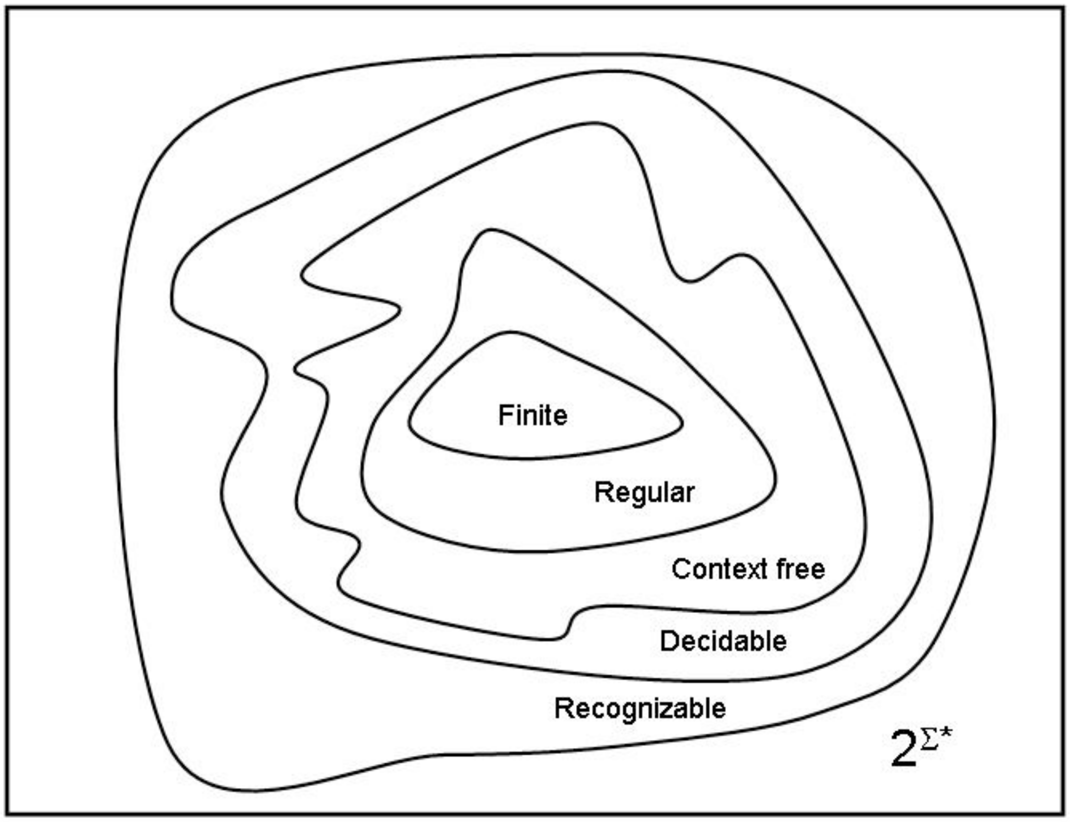
Theorem. Every Context Free Language is decidable.

Proof. Let L be a CFL, and G be a CFG generating L.

We build a TM D, deciding L:

```
D = "On input w,
        Run T on input (G, w)
        Accept if T accepts, reject otherwise."
```

Here is the picture of language classes we have seen so far:



E_{CFG} = { G | G is a CFG and L(G) = Ø }

Theorem. E_{CFG} is decidable.
Proof. The idea is to mark from bottom up all nonterminals that generate *any* string of terminals. In the end, if S is marked, G generates a string and so G $\notin E_{CFG}$. (see page 171 for an example)

EQ_{CFG} = { G, H | G and H are CFLs and L(G) = L(H) }

As we will see later on, EQ_{CFG} is not decidable.

The Halting Problem

The Halting problem asks whether a given Turing machine M accepts a given string w:

A_{TM} = { <M, w> | M is a TM that accepts w }

Theorem. A_{TM} is undecidable.

Before we prove that, we prove another (easier) result.

Theorem. A_{TM} is Turing-recognizable.

Proof. The following TM U recognizes A_{TM}:
1. Simulate M on input w
2. If M ever enters its accept state, accept; if M ever enters its reject state, reject.

Note: this machine loops on <M,w> if M loops on w – therefore it only recognizes $A_{TM.}$

SERIAL NUMBER OF THE TM

* Unary def. The <u>unary representation</u> of decimal 1,000,000 needs only one type of symbol, but that symbol is repeated a million times.
* Let the states be designated in UNARY NOTATION WITH ZEROES ... meaning n is encoded as a string of n zeroes as below.

state 1 = 0
state 2 = 00
state 3 = 000
...
state12 = 000000000000
etc.

* Let the input and output characters in an alphabet of n symbols be represented likewise, with an appropriate coding scheme. For example, in all of our examples, the standard input and output sets will be,

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

which leads to
0 = 0
1 = 00
b = 000

If we chose to include other special characters, we could define them likewise. For example,
$\Sigma = \{0, 1, b, \#, \$\}$
gives
0 = 0
1 = 00
b = 000
= 0000
$ = 00000

for the input characters.

* Let the direction be represented:
Left = 0
Right = 00

Then separate the pieces of the 5-tuple with spacing 1's. SO .... (1,0,b,R,2) becomes:

0 1 0 1 000 1 00 1 00

Then, list all the rules, separated by 11:

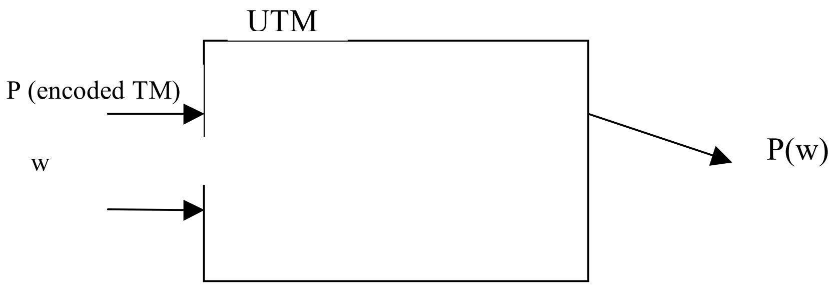
0101000100100 11 0100100010010000 11 etc.

Then surround the entire collection with 111's and we have a Turing Machine definition in binary. This is often called the SERIAL NUMBER of the TM.

UTM

The Universal Turing Machine. This machine begins with the serial number for any TM, P, and the input, x, to P and then simulates the output for P(x).

In other words, UTM is an interpreter for Turing Machines. It will simulate the action of any other TM.

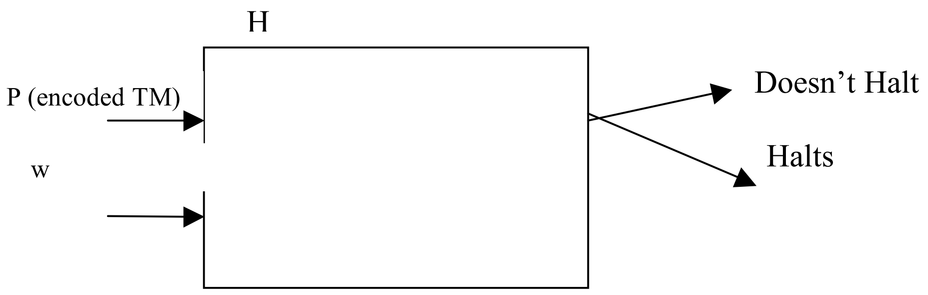


HALTING PROBLEM restated

A_{TM} = { <M, w> | M is a TM that either accepts or rejects w }

Theorem. A_{TM} is undecidable.

Helpful programmer utility if it existed:



However H does not exist.

PROOF - Proof by contradiction.

=========================

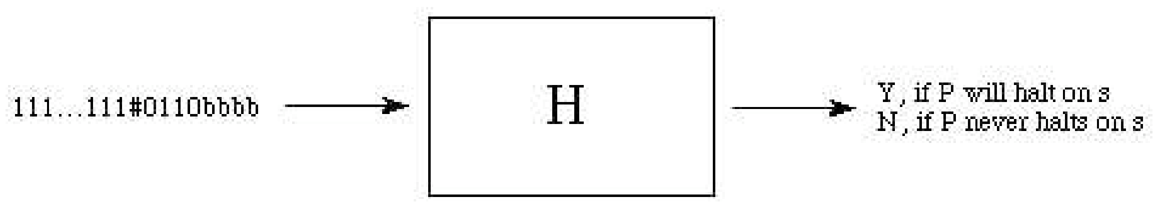
Step 1. Assume H exists.

H is a machine which uses two inputs on the tape:

The serial number of the program, P, to be checked
The input, s, to P

H always halts and correctly answers:

Y, if P(s) eventually will halt
N, if P(s) will run forever



=========================

Step 2. Create a machine called H'.

Use the powerful machine H to build the following machine, H':

input: The serial number of the program, P, to be checked

algorithm:

1. Use the COPY TM to duplicate the initial string. For example, the program P might be represented as

    1110101010010111

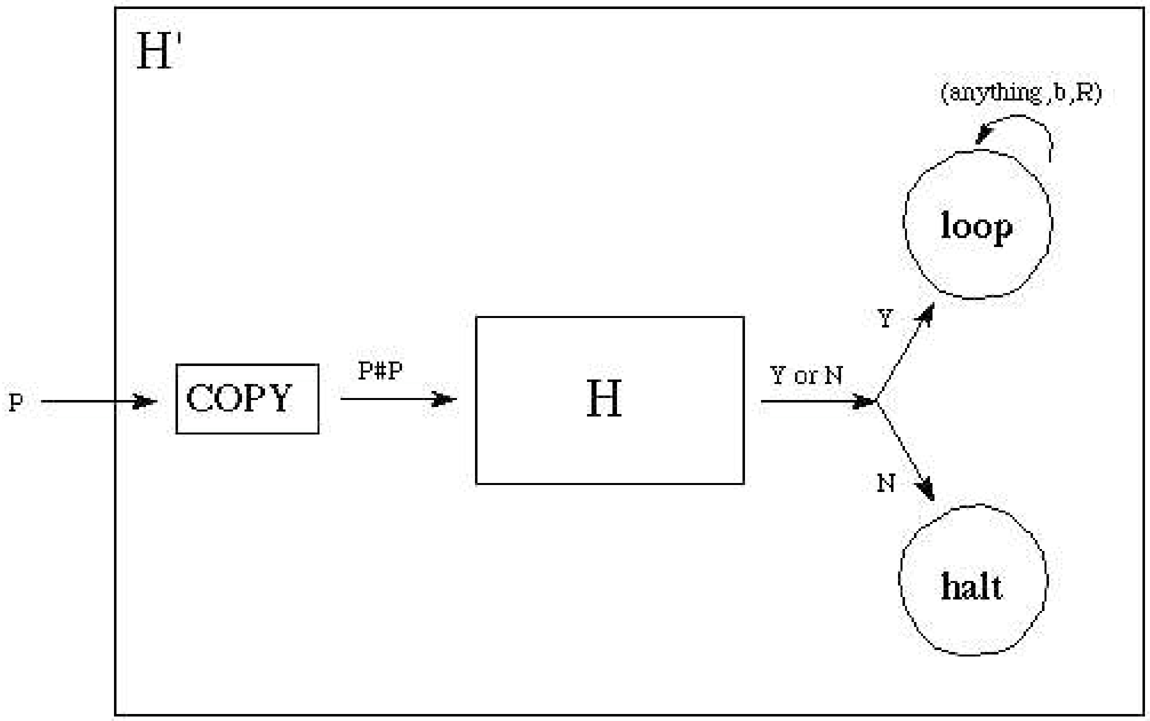
on the input tape. In this case, the tape is modified to

    1110101010010111#1110101010010111bbbb

Thus, the tape now contains the serial number of the machine P to represent a machine, and the serial number of P to represent input to the machine. P is using itself for input.

2. Send P#P into the H. H will determine whether the machine will eventually halt given its own serial number as input. The result will be boolean (Y or N).

```
3. If the result from H is Y,
      then send the machine into a deliberate infinite loop
   else
      force the machine to halt.
```



Trivial and true parts of this machine: copy a tape, go into a deliberate infinite loop, and to halt immediately.

Now, ... here's the magic ......, what happens after we build H' and determine its serial number if we send H' into H' as x?

If H' will run forever on H' as input, then H' stops.
If H' will halt on H' as input, then H' runs for ever.

CONTRADICTION:
Since all parts of H' are trivial except H, it follows that the initial assumption that H exists is incorrect.

DIAGONALIZATION

Now we make a short digression into a mathematical topic, that of **countable** & **uncountable** sets, and the **diagonalization** method.

Definition. A function f: A → B is

- One-to-one, if a ≠ b implies f(a) ≠ f(b).
- Onto, if for every b ∈ B, there is an a ∈ A such that f(a) = b.

Definition. Two (possibly infinite!) sets A and B are of the same size if there is a one-to-one and onto function f: A → B.

Example: The sets A = { mouse, cat, dog }, B ={ cheese, mouse, bone } are of same size because there is an obvious one-to-one and onto function $f: A \rightarrow B$.

Example. The sets N = { 1, 2, 3, 4, ….. } and Q = { 2,4,6,8… } are of the same size. Why? Because we can map the first number to the first even number and the second number to the second even number, etc.

Definition. A set is <u>countable</u> if it is the same size as N = { 1, 2, 3, 4, …. }

Theorem. The set of real numbers is uncountable.
(see proof in textbook – $x \neq f(n)$ for any n)

Theorem. The set of real numbers between 0 and 1, R, is uncountable.

Proof. <u>This is an example of the diagonalization method.</u>

Let's represent the numbers between 0 and 1 with decimal digital notation, as .xyzw… where x, y, z, w, … are among 0, …., 9.

Say that there is a function $f: N \rightarrow R$. Let's draw this function:

$f(1) = .x_{11}\ x_{12}\ x_{13}\ x_{14}$ …..
$f(2) = .x_{21}\ x_{22}\ x_{23}\ x_{24}$ …..
$f(3) = .x_{31}\ x_{32}\ x_{33}\ x_{34}$ …..
$f(4) = .x_{41}\ x_{42}\ x_{43}\ x_{44}$ …..
…

So, $f(i) = .x_{i1}\ x_{i2}\ x_{i3}\ x_{i4}$ ….., for any $i \geq 1$.

Then, consider the following number:

Let $y_{ij} = 5$, if $x_{ij} \neq 5$, and 6, if $x_{ij} = 5$.

$r = .y_{11}y_{22}y_{33}y_{44}y_{55}$….

Then, for any i, $r \neq f(i)$. In particular, r and f(i) always differ in the ith digit.

The above proof technique is called the <u>diagonalization method</u>.

Theorem. The set of all finite strings over Σ is countable.

Proof. (sketch) To list all strings, start by listing all strings of length 1, then all strings of length 2, etc. $\Sigma = \{0,1\}$ length 0 – e; length 1 – 0,1; length 2 – 00, 01, 10, 11, length 3…

Theorem. The set of all languages over Σ is uncountable.

Proof. Take all strings over Σ. This set is an infinite countable set which is the same size as { 1, 2, … }. A language L can be uniquely described by a binary number between 0 and 1, $.x_1x_2x_3\ldots$, where $x_i = 1$ if the ith string over Σ is in L, and 0 otherwise. This is a map which is one-to-one and onto. Since the numbers between 0 and 1 are uncountable, so are the languages over Σ. Infinite binary sequence is uncountable.

Theorem. The set of Turing Machines is countable.

Proof. For any given machine, we can give a full description of the machine with a finite string. $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$, and each of the elements of the 7-tuple is finite. The function δ for example, is $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{ \text{Left}, \text{Right} \}$, and can be written down as a sequence of $|Q| \times |\Gamma| \times |Q| \times |\Gamma| \times 2 = 2\,|Q|^2\,|\Gamma|^2$ 5-tuples (q_a, x, q_b, y, d).

Corollary. There are some languages that are not Turing-recognizable. (!!)

Proof. If not, then there would be a map from the set of languages, to the set of Turing Machines that recognize them. That is impossible because the set of languages is uncountable, and the set of Turing machines is countable. Therefore, some languages are not recognized by any Turing machine.

Theorem. The Halting Problem, A_{TM}, is undecidable.

Proof. Let's assume that a Turing Machine H decides A_{TM}.

On input (M, w), H accepts if M accepts w, H halts and rejects if M does not accept w.

We will use diagonalization to derive a contradiction. Consider the following matrix:

On the rows, let's list all the TMs $M_1, M_2, \ldots$. For example, we can list them alphabetically according to their description.

On the columns, let's list all the possible strings $\in \Sigma$ (e.g., $s_1 = 0$, $s_2 = 1$, $s_3 = 00$, $s_4 = 01$, $s_5 = 10$, $s_6 = 11$, ….).

On each cell (i,j), if machine M_i accepts string s_i, then put a 1. Otherwise (if M_i rejects or loops forever), put a 0.

$$\begin{array}{lllll} & S_1 & s_2 & s_3 & \ldots \\ M_1 & 0 & 0 & 1 & \ldots \\ M_2 & 1 & 0 & 0 & \ldots \\ M_3 & 0 & 0 & 1 & \ldots \end{array}$$

…   …   …   …   …

The machine H can decide for any column (i, j), whether the value in the column is 0 or 1. Now construct the following machine D that uses H as a subroutine:

```
D = "On input (M), where M is a Turing Machine and (M) is a finite string describing M,
        1. Run H on input (M, (M))
        2. If H accepts, then reject.
           If H rejects, accept."
```

Notice that since H always halts with either acceptance or rejection, D always halts. However, D cannot be listed on the table. The reason is that on column (D, (D)), D cannot contain a 0 or a 1: If (D, (D)) is 0, then D rejects input (D). Therefore by definition of H, H has to reject (D, (D)). But then by definition of D, D has to accept (D), contradiction. Similarly, if (D, (D)) is 1, then we reach a contradiction.

Therefore, D cannot exist. But D would be easy to construct if we had H. Therefore, H cannot exist. Therefore A_{TM} is not decidable.

	<M1>	<M2>	<M3>	<M4>	<D>
M1	accept	reject	accept	reject	accept
M2	accept	accept	accept	accept	accept
M3	reject	reject	reject	reject	reject
M4	accept	accept	reject	reject	
.					
.					
.					
D	reject	reject	accept	accept	?

Closure Properties of Decidable and Recognizable Languages

Theorem. Closure properties of Decidable languages. The class of decidable languages is closed under Union, Concatenation, Star, Intersection, and Complementation.

Theorem. Closure properties of Recognizable languages. The class of recognizable languages is closed under Union, Concatenation, Star, and Intersection.

Note: the complement of a non-decidable language is never decidable.

Are there any languages that are not even recognizable? Yes—the following theorem shows how to get such a language.

Theorem. A language is decidable if and only if it and its complement are recognizable.

Proof. If a language is decidable, then its complement is decidable (by closure under complementation).

For the other direction, let L and $\overline{L}$ be recognizable by M_1 and M_2, respectively. We construct machine M that decides L:

```
M := "On input w,
        Set n = 1
        1. Simulate M_1 on w for n steps. If it accepts, accept
        2. Simulate M_2 on w for n steps. If it accepts, reject
        3. Increment n and go to step 1"
```

Either $w \in L$, or $w \in \overline{L}$. Therefore either M_1 or M_2 will halt in a finite number of steps. Therefore M will halt in a finite number of steps.

Theorem. The language $\overline{A_{TM}}$ = { (M, w) | M does not accept w } is not recognizable.

Proof. A_{TM} is recognizable. If $\overline{A_{TM}}$ was also recognizable, then both languages would be decidable. But A_{TM} is not decidable.