§1 2020-05-21

§1.1 Existence of Unsolvable Problems

Modern language makes this proof much easier than it was for Turing.

Theorem 1.1
There is no algorithm that can analyze algorithms and their inputs and determine whether the algorithm halts or not on the given input.

Proof. Write S for an algorithm. $S(x)$ for running algorithm on input x. $\#S$ for the code of S. Everything used to have to be coded up as an integer. $S(w)\downarrow$ indicates that S halts when run on w. $S(w)\uparrow$ for when S runs forever on the given input.

Assume we have a program $H(\#S, x)$ that returns true if $S(x)\downarrow$ and false if $S(x)\uparrow$. H itself always halts.

Define $P(x) :=$ if $H(x, x)$ then loop forever else halt.

Now we give P it's own code. The question is $P(\#P)\downarrow$ or $P(\#P)\uparrow$. Two cases

1. If $P(\#P)\downarrow$, then H returns true but this means that P loops. Contradiction
2. If $P(\#P)\uparrow$, then H returns false but this means that P halts. Contradiction

Therefore a program like P cannot exist. □

§1.2 Turing Machine

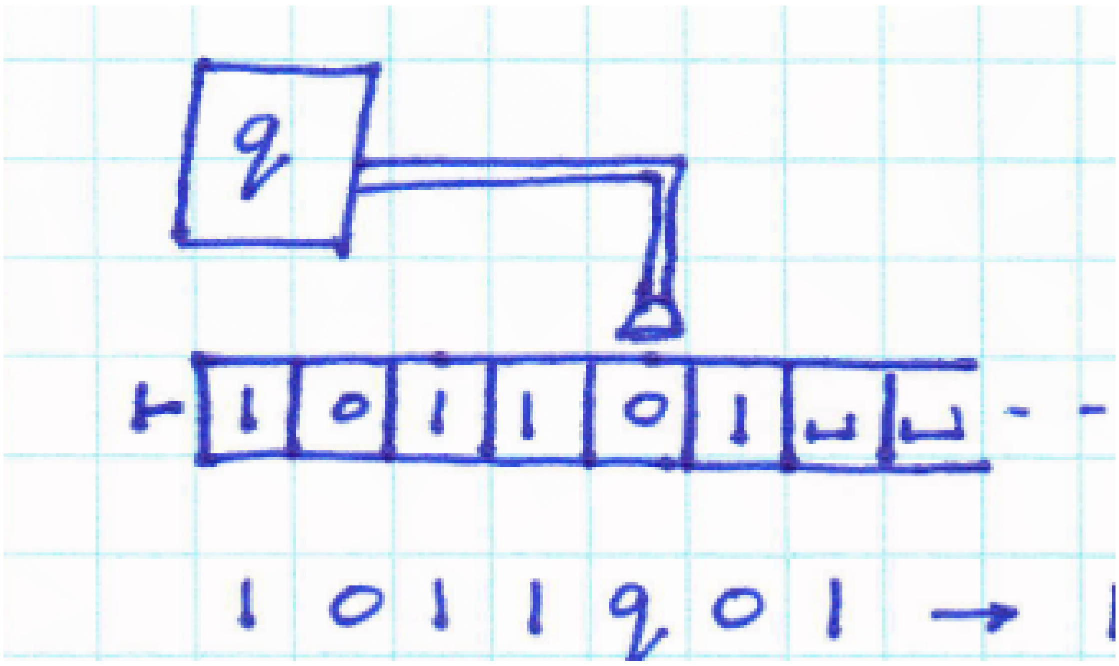
Notes

$\delta(q, a) = (q', b, L)$ means if the machine reads a and is in state q, it changes state to q', writes b in the place of a and moves one step to the left.

At each stage only a finite amount of information is processed.

Easy to see that $\{a^n b^n c^n \mid n \geq 0\}$ can be recognized by a turning machine.

A configuration of a TM is a description of its state, the tape, and the position of the reading head. Convention that you write all of the tape to the left of the state, then the state, and then the rest of the tape including the cell the head is looking at.



With this you can move backwards in the "stack" without deleting information.

uaq_ibv yields uq_jacv if $\delta(q_i, b) = (q_j, c, L)$. Emphasis on how only a three letter window is affected.

Acceptance of w by M. Start in a start configuration q_0w. Follow the transitions and end up in the accept state. Once you get to the accept state you can't leave.

A new phonomenon: The machine may go into an infinite loop. This doesn't happen in DFA or NFA because you read left to right and you are done.

The language of a machine. $L(M) = \{w \mid M \text{ halts and accepts } w\}$.

If $w \notin L(M)$ it does not mean that M explicitly rejects w. It could be looping forever. Basically there is a new "I don't know" option which makes a huge difference. Not everything can be answered.

Definition 1.2 (Turing Recognizable)**.** L is <u>Turing Recognizable</u> if $\exists TM\ M$ such that $L = L(M)$.

We say computably enumerable or CE for recognizable.

Definition 1.3 (Turing Decidable)**.** L is <u>Turing Decidable</u> if $\exists TM\ M$ such that $\forall w,\ M(w) \downarrow$ and $L = L(M)$. i.e. for every word you get a Yes/No answer.

We say <u>computable</u> for decidable.

Fact 1.4. Any decidable language is, of course recognizable.

Note 1.5. Old terminology. Recursively enumerable (RE) for CE. Recursive for decidable.

§1.3 Models of Computation

One can think of a turing machine as a computing function $f : \mathbb{N} \to \mathbb{N}$ instead of accepting languages. They are equivalent, but sometimes it is more helpful to think in one way than the other.

People proposed many variations all of which turned out to be equivalent to turing machine. Examples include Multitape Turing Machine, Nondeterministic Turing Machine, n-dim TM, Post Machine, 2 stack NFA, 2 counter, RAM using assembler, while programs, any modern programming language, λ-calculus, conbinatory logic, phrase structure grammas, post systems. This is not to say that some are not differences in efficiency/ease.

Are all possible formalisms equivalent? We don't know for sure but it is believed so. Church-turing thesis says so.

So we believe there isa notion of computable functoin independent of any specific model of computation.

Proved that certaint hings were impossible but in a completely different sense. When referring to distributed computing, the notion of what is computible

Turing machine writes down all the prime numbers. Never going to write them all because there are infinitely many.

If you define computing to be looking at the answer one it halts, if you say you have one TM who's tape is being fed into a second TM, the first guy doesn't even have to halt before being useful.

§1.4 Theory of Computability

We say 2 infinite sets are equipollent if $\exists$ a bijection between them. There is no bijection between $\mathbb{N}$ and $2^{\mathbb{N}}$ but there is a bijection between $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$.

Definition 1.6 (Equipollent)**.** Two sets are said to be equipollent if there is a bijective function mapping one onto the other.

"Equipollent" means "of equal power", where "power" here alludes to the size of the sets. Two sets are equipollent precisely if they have the same cardinality.

We now have algorithms that may loop forever and produce no output. We model this by partial functions, $f : \mathbb{N} \to \mathbb{N}$. Partial means that f is not defined for every $n \in \mathbb{N}$. Let $f(n)$ mean f is defined on n and $f(n)$ mean f is not defined on n.

$$dom(f) = \{n \mid f(n)\}$$

$f = g$ as partial functions means $dom(f) = dom(g)$ and $\forall n \in dom(f), f(n) = g(n)$. New notion that we don't typically have is $f \leq g$ means that $dom(f) \subseteq dom(g)$, $\forall n \in dom(f)$, $f(n) = g(n)$. This gives a partial order structure.

There is a minimal element, i.e. the everywher undefined function. It's domain is the empty set.

$$range(f) = \{m \mid \exists n \in dom(f) \text{ s.t. } f(n) = m\}$$

When discussing computability, $\mathbb{N}$ is not important; no real difference between $\mathbb{N}$, $\mathbb{N}^k$, and Σ^*. $\mathbb{R}$ is not comparable because uncountable; there is a fascinating field concerning $\mathbb{R}$ but we won't discuss it here.

Note 1.7. It is not algorithmic to write things like:

1. wait for M to halt, if it doesn't halt then...
2. Check on every word if...

Definition 1.8 (Computable Function)**.** A function $f : \mathbb{N} \to \mathbb{N}$ is computable if there is an algorithm A such that for all $n \in dom(f)$, A halts and the output if $f(n)$.

Definition 1.9 (Computable/Decidable Set)**.** A set of natural numbers $X \subseteq \mathbb{N}$ is computable or decidable if its characteristic function

$$\chi_x(n) = \begin{cases} 1, & n \in X \\ 0, & n \notin X \end{cases}$$

it total computable.

Definition 1.10 (Total function)**.** A total function is defined for all possible input values.

Proposition 1.11
An infinite set of natural numbers is decidable iff it is the range of a total non decreasing computable function.

Proof. Suppose we have X, we have f as above $X = range(f)$ and we have an always terminating algorithm for f called A. Let x, want to know if $x \in X$.

Run A on $0, 1, 2, 3, \ldots$. Eventually you may get $A(n) = x$ then say yes $x \in X$. Or you may get $A(n) > x$ then stop and say no, $x \notin X$.

For the reverse direction, assume we have a decision procedure for X, call it B. Run B on $0, 1, 2, \ldots$. As soon as it says "Yes" for the the first time you say $f(0) = x_0$. Second time you say $f(1) = x_1$. This gives a total non-decreasing computable mapping whose range is X. □

Definition 1.12 (Enumerable)**.** A set $X \subseteq \mathbb{N}$ is called enumerable or CE if $\exists$ algorithm A that lists all the members of X and only the members of X in some order, not necessarily increasing order. A may not halt but it produces every element of X eventually. No guarantee of when and in what order.

Theorem 1.13
A set X is CE iff

1. It is the domain of some computable function.
2. The range of a computable function
3. S_X is computable where

$$S_X = \begin{cases} 1 & n \in X \\ ? & n \notin X \end{cases}$$

Proof. Rigorous proof in course notes. □

Note 1.14. Computable set is the same as decidable set. Enumerable is the same as computably enumerable and is weaker then computable/decidable.

Review notes online

Theorem 1.15
The union and intersection of 2 CE sets is CE. This is how computers can do things concurrently.

Remark 1.16. The complement of a CE set is not always CE.

Proposition 1.17 (Post's Theorem)
If a set X and its complement $\overline{X}$ are both CE then they are both decidable.