Last time we saw some examples of decidable languages (or, solvable problems). Today we will start by looking at the relationship between the decidable languages, and the regular and context-free languages. And then, we will see that there are languages that are not decidable.

Reminder, once again:

Definition: A language L is called *Turing-decidable* (or just *decidable*), if there exists a Turing Machine M such that on input x, M accepts if $x \in L$, and M rejects otherwise. L is called *undecidable* if it is not decidable.

Decidable Languages - relationship with other classes.

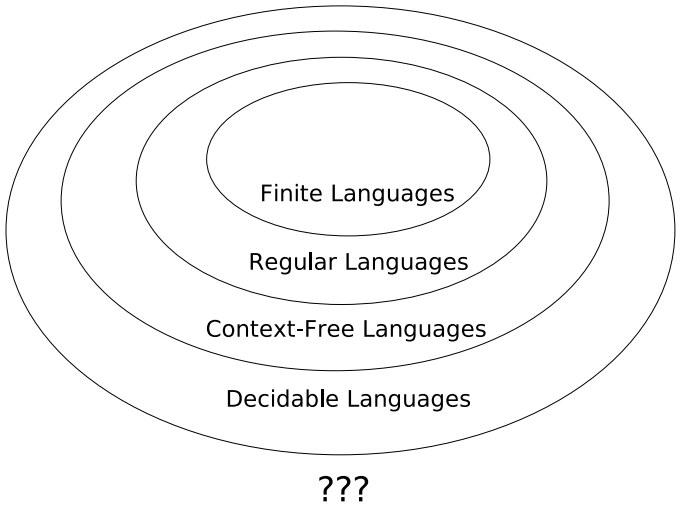
- **Theorem:** every context-free language is decidable.

  Proof: Let L be a CFL, we need to show that there exists a Turing Machine M that decides L. Since L is a CFL, there is a CFG G that generates L. So, for every string w, our machine will feed $< G, w >$ into the Turing Machine that decides A_{CFG} (from last lecture). So $M =$ "on input w

  1. run the Turing Machine for A_{CFG} on $\langle G, w \rangle$.
  2. *Accept* if it accepts, *reject* otherwise.

  □.

- We also know that there are decidable languages that are not context-free, for example $L = \{0^n 1^n 2^n \mid n \geq 0\}$. So the picture of the world thus far is this:



- Now we will see that there are languages that are undecidable – by Church-Turing Thesis, these correspond to the algorithmically unsolvable problems. In other words, these lie beyond the capabilities of computing devices.

The Halting Problem and Undecidable Languages

- Historically, the first undecidable problem was presented by Alan Turing in his 1936 paper "On Computable Numbers with an Application to the Entscheidungsproblem", *Proceedings of the London Mathematical Society*, ser. 2, vol. 42. His goal was to show that the decision problem for first-order logic (the "Entscheidungsproblem") is algorithmically unsolvable. One of the steps towards that goal was to show that the following problem is also unsolvable: given a Turing Machine M, does M halt when started with the blank tape. The more general version of this problem is now known as *The Halting Problem*:

  Given a Turing Machine M, and an input w, does M halt on w.

  Or, expressed as a language

  $$H_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM that halts on input } w\}.$$

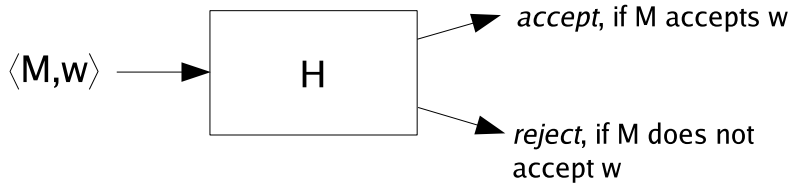
  The language H_{TM} is undecidable. We could prove this fact directly, but we will follow the textbook, and will prove directly that the acceptance problem for Turing Machines is undecidable, and then use that to prove the undecidability of H_{TM}.

- **Theorem:** The language

  $$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM that accepts } w\}$$

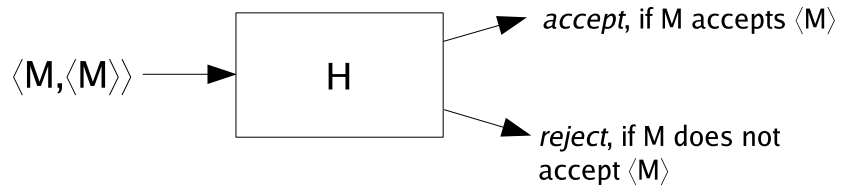
  is undecidable.

  Proof: By contradiction. Assume that A_{TM} is decidable. Let H be the Turing Machine that decides A_{TM}.

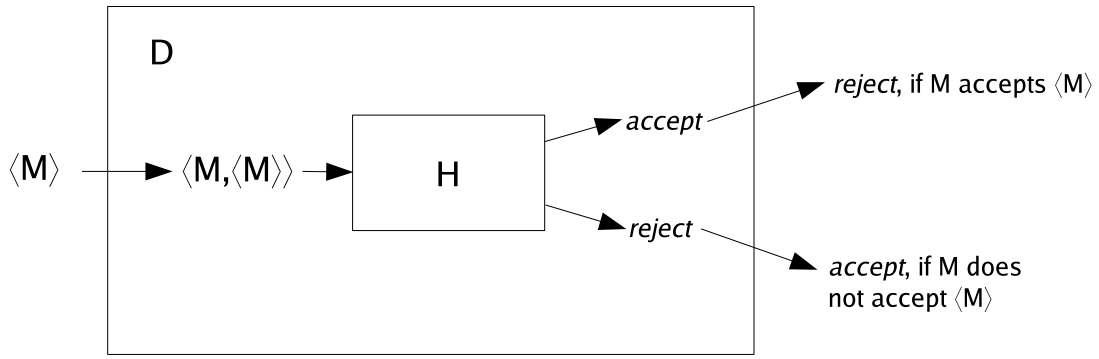


  So, H accepts $\langle M, w\rangle$ when M accepts w, and rejects otherwise.

Since w is just a string, nothing stops us from giving the string that encodes the Turing Machine itself as an input. This will look like this:



Now, we are going to construct a machine D that does something opposite from H. On input string $\langle M \rangle$, D is going to construct an input string $\langle M, \langle M \rangle \rangle$ and feed it to H. Once H is finished, D will do exactly the opposite from what H did: it will *reject* if H accepted, and *accept* otherwise. D will look like this:



So, D accepts $\langle M \rangle$ if M does not accept $\langle M \rangle$, and rejects M if M accepts $\langle M \rangle$. Note that D always halts, because H always halts.

And now, the contradiction: what will D do on input $\langle D \rangle$? Since D always halts, it must either accept $\langle D \rangle$ or reject $\langle D \rangle$. But, if D accepts $\langle D \rangle$, then D does not accept $\langle D \rangle$. On the other hand, if D rejects $\langle D \rangle$, then D accepts $\langle D \rangle$.

We conclude that A_{TM} is undecidable. □

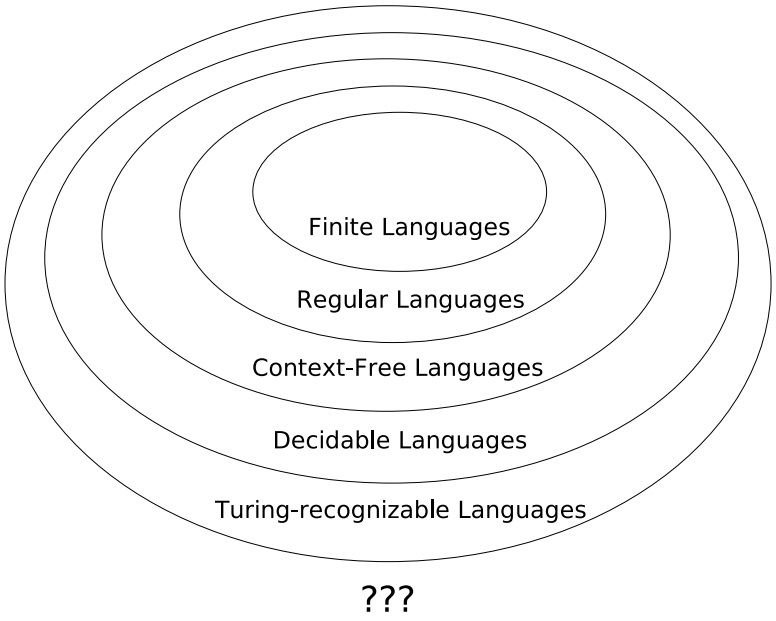
How does one come up with such clever constructions ? The technique used to construct the machine D is called *diagonalization*, and goes back to the work of the German mathematician Georg Cantor (1870's). The textbook explains the technique in more detail – please read.

- On the other hand, A_{TM} is Turing-recognizable. Here is the Turing Machine that recognizes A_{TM}:

  U = "on input $\langle M, w \rangle$

  - Simulate M on input w.
  - *Accept* if M accepts, *reject* of M rejects."

  Note that U may not finish step 1 (simulation) – if M happen to go into an infinite loop, then U will do so too. Hence, U *recognizes* A_{TM}. So, now, the picture of the world looks like this:



  The machine U is called a *Universal Turing Machine* – it is capable of simulating every other Turing Machine. We now believe that such machine U can be built, because we can describe an algorithm for it (and so, by Church-Turing thesis there is a Turing Machine that implements that algorithm). In 1936 paper, Alan Turing gave an *actual* description (i.e. state transition table) of this machine U.

  The Universal Turing Machine is a theoretical model of an an all-purpose computer – the input encoding of M acts as a text of a program in some programming language. And, in fact, the first all-purpose digital computer, EDVAC, architected in 1945 by Jon von Neumann, was in essence an implementation of Turing's Universal Machine (with some important additions, such as a separate arithmetic-logic unit). This architecture of EDVAC is now known as Von-Neumann architecture, and its the architecture of the vast majority of computers in existence today.

- Now, lets go back to the Halting Problem.

  Theorem: The language

  $$H_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM that halts on input } w\}$$

  is undecidable.

  Proof: Assume that H_{TM} is decidable. We will *reduce* A_{TM} to H_{TM} – in other we will show how to construct a Turing Machine M_A that decides A_{TM} using the Turing Machine M_H that decides H_{TM}. This gives us a contradiction, because we know that A_{TM} is undecidable, and so M_A cannot exist.

  The word "reduce" simply means solving a given problem by converting it into another problem which we already know to solve.

  So, the Turing Machine for A_{TM} can be constructed as follows:

  $M_A =$ "on input $\langle M, w\rangle$

  1. Run M_H (the Turing Machine for H_{TM}) on $\langle M, w\rangle$.
  2. If M_H rejects (which means that M does not halt on w), then *reject*.
  3. If M_H accepts, then simulate M on w (guaranteed to stop).
  4. *Accept* if M accepts w, *reject* is M rejects w."

  Since M_H always halts (by our assumption) the M_A also always halts. M_A accepts $\langle M, w\rangle$ if M accepts w, and M_A rejects $\langle M, w\rangle$ if M either rejects w, or goes into infinite loop. Thus, M_A decides A_{TM}. Contradiction. □