pp. 201-210. **Undecidability** (Sec. 4.2)

- Remember A_{DFA} = {<B,w>| B a DFA that accepts w}
  - We proved it is decidable
  - I.e. Given any <B,w> some TM can
    - Decide if B accepts w, or not!
    - And the TM always halts
- *Consider **A_{TM} = {<M,w>| M is a TM and M accepts w}**
    - If A_{TM} is decidable, then
      - we can take ***ANY*** program and ***ANY*** input,
      - and determine ***yes/no*** if M accepts w in finite time
    - Good for doing automatic program verification
- Question: is this possible?
- **KEY**: we can write a *recognizer* U, *but not a decider*
  - U interprets M executing with w (i.e. your TM project)
  - If M stops, U stops
  - Thus if M accepts w, so does U
- This section: prove we cannot write a TM decider
  - Cannot write a TM U that always stops with correct answer when M does not halt

- (p. 202)* **Theorem 4.11 A_{TM} is undecidable**
  - First, simpler version of proof than book's
  - **ASSUME a TM H exists which decides A_{TM}**
  - Imagine following (large) table
    - ith row for all possible machines M_i
      - Ordered by "size" of <M>
    - one column for each possible string w
      - Ordered by length of w
    - Entry (i,j) has accept or reject in it, depending on what M_i does with string w_j

	w0	w1	w2	w3	…
M1	reject	accept	reject	accept	
M2	reject	accept	reject	reject	
M3	accept	reject	reject	reject	
M4	reject	reject	accept	accept	
…					

  - H should be able to compute this, one (M,w) entry at a time, notionally in a "diagonal" order

- If H always stops with accept/reject, then can define D
  - D accepts when H rejects and vice versa
    - Given $<M_i, w_j>$
    - Run H on $<M_i, w_j>$
    - If H accepts, D rejects and if H rejects then D accepts
  - If D is a TM, then it corresponds to some row in table
    - i.e. gives accept/reject for each w_j
  - So H applied to $<D, w_j>$ gives what D returns
- **BUT D SUPPOSED TO GIVE *<u>OPPOSITE</u>* OF WHAT H DOES**
- So assumption that H exists must be false

	w0	w1	w2	w3	…
M1	reject	accept	reject	accept	
M2	reject	accept	reject	reject	
D	accept	reject	reject	reject	
M4	reject	reject	accept	accept	
…					

- (p. 202) Book's Proof **Theorem 4.11 A_{TM} is undecidable**
  - Definitions: Assume sets A & B, & function f:A->B
    - f is **one-to-one (or injective)** if f(a) != f(b) when a != b.
    - f is **onto** (or **surjective**) if for all b, there is an a: f(a)=b
    - f is a **correspondences** (or **bijective**) if both
      - Equivalent to pairing each a with exactly one b
  - (p. 202) Step 1: **The diagonalization method**
      - Discovered by Cantor in 1873 to compare infinite sets
      - If there is some correspondence between 2 infinite sets, then they are "same size"
    - E.g. N = {1,2,3,4,…} E = {2,4,6,8,…} are the same size
      - For any n in N, pair up with f(n) = 2n in E
  - (p. 203) Set A is **countable** if finite or same size as N
    - i.e. each element of A matchable to an integer
  - Now consider Q = {m/n |m,n in N} (**Rationals**)
    - Q seems much larger than N, but *not so*
    - See p. 204 Fig. 4.16 for correspondence with N
      - I'th row contains all rationals with i as numerator
      - j'th column has all rationals with j as denominator
      - Count diagonally
      - Skip any i/j that reduces to an earlier #
    - Q has same size as N!

- **Uncountable** if no correspondence with N
- (p. 205) ***Theorem 4.17: Reals R is uncountable***
  - Proof by contradiction
  - Suppose bijective function f between N and R
    - i.e. can map each integer into a real and v.v.
  - Show that such an f always misses at least 1 number x
    - Suppose f exists
      - Then f(1) = …, f(2) = … for some numbers like pi
      - Construct an x not in correspondence
        - Let 1st digit of x be anything different from 1st digit of fraction of f(1) – thus x!=f(1)
        - Let 2nd digit of x be anything different from 2nd digit of fraction of f(2) – thus x!=f(2)
        - …
        - Thus x is different from f(n) **for any n** because it differs in nth digit!
        - Thus f is not a correspondence
  - (p. 206) **Aside: define B = Infinite Binary Sequences**: *unending* sequence of 0s & 1s
    - B is uncountable using similar proof as for R

- (p. 206) Corollary 4.18 **Some languages are not Turing Recognizable**
  - Proof:
    - **Set of all TMs is countable**
      - Each TM has an encoding into finite string <M>
      - If we omit all illegal encodings, we get set of all TMs
      - Each encoding can be converted into an integer
    - Now define L = set of all languages over ∑
      - |L| is infinite – but what about its size?
      - Let $\sum^* = \{s_1, s_2, s_3, \ldots\}$ = set of strings; ∑ is finite
        - Question: is this set countable? Yes
      - Each language A in L has a unique **binary sequence** from B = set of unending sequence of 1s and 0s
        - ith bit is 1 if s_i is in A, and 0 if not
        - set of bits called its **characteristic sequence**
        - See page 206 for example
        - Function f:L->B where f(A) is its characteristic sequence & B is set of binary sequences
          - Clearly one-to-one and onto
          - Thus B and L are same size
        - Since B is uncountable, so must L
      - **Which means there are more languages than TMs!**

- (p. 207) Now re-consider A_{TM} = {<M,w> }.
  - Assume A_{TM} is decidable by TM H
  - On input <M,w>
    - H halts and accepts <M,w> if M accepts w
    - H halts and rejects if M **fails to accept** w
  - Now construct TM D with input <M> as follows
    - D calls H to determine what M does given *its own description* <D> as its input string
    - i.e. look at language {<M,<M>>}
    - Whatever H does, D does the opposite
    - D = “On input <M>, where M is a TM
      - Run H on input <M,<M>>
      - Output the opposite of what H does
  - Note: <M,<M>> is like a compiler compiling itself
  - Thus D(<M>)
    - = accepts if M does not accept <M>
    - = rejects if M accepts <M>
  - Now run D on <D>:
    - D(<D>) **accepts** if D rejects <D>!
    - D(<D>) **rejects** if D accepts <D>!
- No matter what D does, it must do opposite.
- **THUS neither D nor H can exist**!

- See Fig. 4.19 – 4.21 for how diagonalization comes into play
- THUS **A_{TM} is undecidable! (but it is TM recognizable)**

- *Define L is **co-Turing-recognizable** if it is complement of a Turing-recognizable language
- (p. 209)* **Theorem 4.22. A is decidable iff it is both Turing recognizable and co-Turing recognizable**.
  - =>: if A is decidable then clearly it is both recognizable and co-recognizable
  - <=: Construct M from M1 for recognizer and M2 for co-recognizer. Then
    - Run machines in parallel on same input
    - If M1 accepts, accept; if M2 accepts, reject
  - Every string is either in A or not(A)
  - Thus one machine halts
  - Thus M is a decider, and thus A is decidable
- (p. 210) **Corollary 4.23 not(A_{TM}) is not Turing recognizable**
  - If it were then A_{TM} would be decidable
  - But A_{TM} is not decidable
  - Then not(A_{TM}) cannot be recognizable