Theorem 5.2. *A language L is Turing acceptable (equivalently, recursively enumerable) if and only if there is a general grammar that generates L.*

- idea of proof
  - grammar → Turing acceptable
    - *M* generates every string derivable from the start symbol *S* –
      - start with $w\$S$ on the tape
      - repeatedly
        - for each string on the tape and each production $x \rightarrow y$, if x occurs in the string, append $\$$ to the end of the tape and copy the string, replacing x with y
        - compare the new string to w, halting if they match
    - if $w \in L$, eventually *M* will produce it and halt

Theorem 5.2. *A language L is Turing acceptable (equivalently, recursively enumerable) if and only if there is a general grammar that generates L.*

- idea of proof
  - Turing acceptable → grammar
    - idea: build a grammar whose rules simulate the steps of *M*
    - let the terminal symbols of *G* be the symbols from Σ
    - let the non-terminal symbols of *G* be the states of *M*, the alphabet symbols not in Σ, <, >, *S*, *A*
    - produce any string of the form $<q_0a^n>$
      - represents a configuration of *M* with *M* in its start state, positioned at the beginning of a string of *n a*'s
    - capture transition $\delta(q_i,s) = (\tau,R,q_j)$
    - capture transition $\delta(q_i,s) = (\tau,L,q_j)$, for each of the alphabet symbols λ
    - add and remove blanks from the end of the current portion of the tape as needed
    - clean up when *M* has halted

$$S \longrightarrow <q_0A>$$
$$A \longrightarrow aA$$
$$A \longrightarrow \varepsilon$$

$$q_i\sigma \longrightarrow \tau q_j$$

$$\lambda q_i\sigma \longrightarrow q_j\lambda\tau$$

$$< \longrightarrow <\#$$
$$<\# \longrightarrow <$$
$$> \longrightarrow \#>$$
$$\#> \longrightarrow >$$

$$<h \longrightarrow h$$
$$h> \longrightarrow \varepsilon$$

$$h\sigma \longrightarrow \sigma h$$

- *L* is Turing-decidable if and only if both *L* and its complement are Turing-acceptable
  - idea of proof, only if direction (acceptable → decidable) –
    - let *M* be a Turing machine accepting *L* and *M'* be a Turing machine accepting *L*'s complement
    - build *T* to decide *L*
      - simulate both *M* and *M'* on *w* one step at a time, halting with output 1 if *M* terminates and output 0 if *M'* terminates
      - if *L* and *L*'s complement are both Turing-acceptable, either *M* or *M'* will halt on *w* and thus *T* halts on *w*
  - idea of proof, if direction (decidable → acceptable)
    - let *M* be a Turing machine deciding *L*
    - build *T* to accept *L*
      - simulate *M* on *w*, then halt with output 1 if *M* halts with output 1 or go into an infinite loop if *M* halts with output 0
    - build *T'* to accept the complement of *L*
      - simulate *M* on *w*, then halt with output 1 if *M* halts with output 0 or go into an infinite loop if *M* halts with output 1

- this can be generalized to other models of computation –
  - *recursively enumerable* is a synonym for *Turing-acceptable* (Thm 5.1)
  - *recursive* is a synonym for *Turing-decidable*

Theorem 5.3. *Let Σ be an alphabet and let L be a language over Σ. Then L is recursive if and only if both L and its complement, $\Sigma^* \smallsetminus L$, are recursively enumerable.*

1. The language $L = \{a^m \mid m > 0\}$ is the range of the function $f(a^n) = a^{n+1}$. Design a Turing machine that computes this function, and find the grammar that generates the language L by imitating the computation of that machine.

3. Show that a language L over an alphabet Σ is recursive if and only if there are grammars G and H such that the language generated by G is L and the language generated by H is $\Sigma^* \setminus L$.

Computable Languages

- recursively enumerable languages are languages that can be defined by computation
- so far, every computational method developed for specifying languages produces only recursively enumerable languages
- yet, most languages are not recursively enumerable
  - there are uncountably many languages over a particular alphabet
  - there are only countably many recursively enumerable languages over the same alphabet

Uncomputable Languages

- most languages are not recursively enumerable
- what do these languages look like?
  - whatever property defines whether *w* is in *L* can't be computable
    - there is no Turing machine (or computer program) that tests whether *w* has the property

Symbolic Representation of Turing Machines

- consider a Turing machine *M*
- we can assume, without loss of generality –
  - *q* is the start state, *h* is the halt state, and the other states are named *q'*, *q''*, *q'''*, …
  - the symbols are 0, 1, *a*, # (blank) with auxiliary symbols *a'*, *a''*, *a'''*, …
- call such a Turing machine a *standard Turing machine*
- *M* can be represented with a string of symbols from the alphabet { *h*, *q*, *L*, *R*, #, 0, 1, *a*, ', $ }
  - the transition rule δ(*q''*,0) = (*a'''*,*L*,*q*) is encoded as `q''0a'''Lq`
  - encode a complete machine by listing the transition rules, separated by $

without loss of generality (w.l.o.g.) – this assumption does not limit what we can consider because states and symbols can be renamed without changing the machine's function

A Turing Machine Generator

- not every string involving the alphabet { *h*, *q*, *L*, *R*, #, 0, 1, *a*, ‘, $ } is an encoded standard Turing machine
  - but whether or not *w* is an encoded Turing machine can be checked
- a list of all strings encoding standard Turing machines can be generated –
  - generate all strings over { *h*, *q*, *L*, *R*, #, 0, 1, *a*, ‘, $ }
  - for each string *w*, check if it encodes a Turing machine
  - if so, add *w* to the output list
- the symbolic representation of standard Turing machines is a recursively enumerable set
  - let T_i be the machine encoded by the *i*th string produced
  - given $n \in \mathbb{N}$, the symbolic representation for T_n can be found by repeating the process n+1 times
- let *G* be a Turing machine which, when run with input a^n, halts with the encoding of T_n

Universal Turing Machine

- the universal Turing machine *U* simulates the computation of any standard Turing machine *T* on any input *w*
  - (the symbolic representation of) *T* and *w* are written on *U*’s tape
  - *U* keeps track of *T*’s state and position
    - *T*’s state is written after *w* on the tape
    - use a special symbol @ to the left of the current symbol in *w* to denote the current position
  - *U*’s operation
    - write @ at the beginning of *w* and *q* after *w*
    - for each step in the computation of *T*
      - determine the current state and symbol for *T*
      - locate a transition rule that applies in this case
      - update the representation of *T*’s state, position, and tape to reflect applying the transition
      - if the new state is *h*, halt
  - observation: *U* halts if and only if *T* halts on input *w*

Theorem 5.4. *Let T_0, T_1, T_2, ..., be the standard Turing machines, as described above. Let K be the language over the alphabet $\{a\}$ defined by*

$$K = \{a^n \mid T_n \text{ halts when run with input } a^n\}.$$

Then K is a recursively enumerable language, but K is not recursive. The complement

$$\overline{K} = \{a^n \mid T_n \text{ does not halt when run with input } a^n\}.$$

is a language that is not recursively enumerable.