Second part of course: normal form theorems, universal objects (functions, TMs, etc.), undecidability results, recursion theorems.

First: Kleene's normal form thm

↳ the big idea: build a primitive recursive rel'n $T(m,n,c)$ which is true iff

m is a (code for a) Turing Machine which halts on input n and c is the (code for the) computation of m on input n

↳ reason this is plausibly possible

- up to how we encode everything (a crucial step), checking $T(m,n,c)$ only involves verifying that c is the correct computation of m on input n (and is terminating)
- we're being handed c, just need to check it follows proper instructions from m on input n
- OTOH: checking whether m halts on a given n (w/o being handed c) would be recursive but not pr:
  ↳ have to run m on n and see if halts. or search for the correct (and terminating) computation of m on input n.

Once we have T. will use to build a lot of interesting objects (unwarsed recursive function, unwarsed TM)

- recall: code for a tuple of natural numbers $(m_0, m_1, \ldots, m_k)$ is
$\langle m_0, \ldots, m_k \rangle = p_0^{m_0+1} p_1^{m_1+1} \cdots p_k^{m_k+1} = \prod_{i=0}^{k} p_i^{m_i+1}$

($p_0 = 2$, $p_1 = 3$, etc)
- have pr decoding function $(s)_i$ = ith entry of tuple coded by s
(so $(\langle m_0, \ldots, m_k \rangle)_i = m_i$)
- $lh(s)$ = length of tuple coded by s
also p.r. $(lh(\langle m_0, \ldots, m_k \rangle) = k+1)$

- we're gonna code TMs, computations of TMs, etc. as single numbers
- how to code a TM?

- Sps M is a TM on $A = \{\sigma_1, \ldots, \sigma_k\}$ w/ states $\{Q_0, \ldots, Q_p\}$
↳ Q_0 start state, may assume Q_p only stop state
↳ $\sigma_0 = *$ symbol for empty square.

- we code the table of M
if $(Q_i, \sigma_j, \sigma_{j'}, m, Q_{i'})$ in table
let $t_w = \langle i, j, j', m, i' \rangle$

where $\bar{m} = m$ if $m = 0, 1$, $\bar{m} = 2$ if $m = -1$
(everything has to be a natural number to be encoded)

- now encode entire TM as:
  $\langle M \rangle := \langle k, p, t_{00}, t_{01}, \ldots, t_{0k}, t_{10}, \ldots, t_{1k}, \ldots, t_{p0}, \ldots, t_{pk} \rangle$
- so: code tells us # of symbols in A, # of states and (encoded) table of M.
- call $\langle M \rangle$ the Gödel number of M

Lemma 1: Define TM(m) iff m is the code of a Turing machine.
Then TM is pr relation.

Pf: write t_{ij} for $(m)_{2+i(m)_0+j}$ (pr in i,j)

then observe:
TM(m) iff $\mathrm{Seq}(m) \wedge \mathrm{lh}(m) = ((m)_0+1)((m)_1+1)+2$
$\wedge (m)_0 \geq 1 \wedge (m)_1 \geq 1$
$\wedge \forall i \leq (m)_1 \, \forall j \leq (m)_0 \, [\mathrm{Seq}(t_{ij})$
$\wedge \mathrm{lh}(t_{ij} = 5) \wedge (t_{ij})_0 = i \wedge (t_{ij})_1 = j$
$\wedge (t_{ij})_2 \leq (m)_0 \wedge (t_{ij})_3 \leq 2 \wedge (t_{ij})_4 \leq (m)_1]$

everything p.r. ✓

- next: how to encode a (finite) computation of M? need a bit more setup.

- start by encoding tape descriptions
- can think of current tape of our TM as a function

$$\tau : \mathbb{Z} \to \{\sigma_0, \sigma_1, \ldots, \sigma_k\}$$

- but $\tau(i) = \sigma_0$ for all but fin. many i
- want to encode τ, first need to encode $\mathbb{Z}$ by $\mathbb{N}$
  - ↪ use bijection $i \longmapsto \bar{i}$ from $\mathbb{Z} \to \mathbb{N}$

$$\bar{\imath} = 0 \text{ if } i = 0$$
$$\bar{\imath} = 2n-1 \text{ if } i > 0 \quad \text{(positives go to odds)}$$
$$\bar{\imath} = -2n \text{ if } i < 0 \quad \text{(negatives to evens)}$$

- w/ this bijection, now think of $\tau : \mathbb{N} \to \{\sigma_0, \ldots, \sigma_k\}$ as a function on $\mathbb{N}$

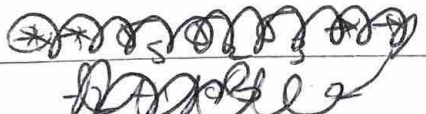
(moving around on negative squares on $\mathbb{Z}$-tape corresponds to even squares on $\mathbb{N}$-tape; positive squares to odd squares)

- code each symbol σ_i in A by $\langle \sigma_i \rangle = i$
- code τ as

$$\langle \tau \rangle = \prod p_i^{\langle \tau(i) \rangle}$$

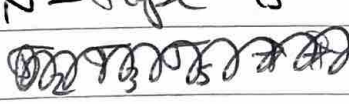
... finite product since $\langle \tau(i) \rangle = 0$ almost always.

e.g.: if $\mathbb{Z}$-tape is



$$\cdots * * \sigma_7 \sigma_4 \sigma_1 * * \cdots$$
$$\cdots -2 \; -1 \; 0 \; 1 \; 2 \cdots$$

then $\mathbb{N}$-tape is



$$\sigma_4 \sigma_1 \sigma_7 * * \cdots$$

and code is $\langle \tau \rangle = 2^4 3^1 5^7$.

Lemma 2 Define TapeDescr(m,t) iff TM(m) and t is code for a tape description τ in the alphabet of TM coded by m.
Then: TapeDescr is p.r.

Pf: TapeDescr(m,t) iff
TM(m) ∧ ∀l<t ∀i≤t[p_i^l | t ⇒ l≤(m)$_0$]
("all non-empty entries in tape coded by t are in alphabet of M (i.e. their codes are ≤ (m)$_0$)") ✓

- next want to encode description of where head is and what state it's in.
- if (Q_i, τ, k) is tuple of (current state, tape, current square) (a situation description) then encode as
$\langle i, \langle \tau \rangle, k \rangle$
(k ranges over N... working w/ N-tape)

Lemma 3: Define SitDescr(m,s) iff TM(m) and s is a code for a situation description for TM coded by M.
Then: SitDescr is p.r.

Pf: SitDescr(m,s) iff TM(m) ∧ Seq(s)
∧ lh(s) = 3 ∧ (s)$_0$ ≤ (m)$_1$ ∧ TapeDescr(m, (s)$_1$) ✓

Lemma 4 Define SingleAct(m, s_1, s_2)
iff TM(m), SitDescr(m, s_1), SitDescr(m, s_2)
and s_2 results from s_1 by a single action
(i.e. application of one table instruction) from
TM coded by m.
Then: SingleAct(m, s_1, s_2) is p.r.

PF: Exercise (i.e. too long to write down)
- but "clear" this is true: only involves
checking if pertinent table instruction
from m applied to s_1 yields s_2. ✓

— Sps $C = ((Q_{i_0}, \tau_0, k_0), \ldots, (Q_{i_m}, \tau_m, k_m))$
is a sequence of sit'n descr'ns of
some fixed TM M s.t. each succession
descr follows from prev. by single
action of M and Q_{i_m} is stop state
— will call such a sequence a
terminating computation

Note: For convenience allow more
than one stop state in such a list
(so actual computation may halt
before the end)

— We code a terminating computation as
$\langle C \rangle = \langle \langle i_0, \langle \tau_0 \rangle, k_0 \rangle, \ldots, \langle i_m, \langle \tau_m \rangle, k_m \rangle \rangle$

Lemma 5. Define $TC(m,c)$ iff $TM(m)$ and c is a code for a terminating comp'n of the TM coded by m.
Then: TC is p.r.

Pf. $TC(m,c)$ iff

$TM(m) \wedge Seq(c) \wedge lh(c) > 0 \wedge$
$\forall i \leq lh(c)\,[\,SitDescr(m, (c)_i) \wedge$
$i \geq 1 \Rightarrow SingleAct(m, (c)_{i-1}, (c)_i)\,] \wedge$
$((c)_{lh(c)-1})_0 = (m)_1$

"last state is a step state"

- now we'll zoom in on TMs that compute unary functions $F: \mathbb{N} \to \mathbb{N}$.
- conventions we'll use: consider TMs on $A = \{\sigma_1, \sigma_2\}$ where $\sigma_1 = 1$ $\sigma_2 =$, (comma) (still have $\sigma_0 = *$)
- given such a TM M, "function computed by M" means function we get by inputting n
  - ↪ new means: n 1's, starting at square 1, followed by a comma
  - ↪ head starts over square 1 in Q_0

and taking output (if M terminates on n, commas deleted)

Lemma 6 Define $Inp(n)$ = code for situation described above for "on input n"
Then $Inp(n)$ is p.r. function

Pf: Observe
$Inp(n) = \langle 0, g(n), 1\rangle$ where $g(n)$ is code for tape description on input n
i.e. $g(n) = p_1 p_3 \cdots p_{2n+1} p_{2n+1}^2$ if $n>0$
$= p_1^2$ if $n=0$
(remember positive entries of $\mathbb{Z}$-tape appear on odds in $\mathbb{N}$-tape: so first is code for n is followed by comma, second for a comma in Sq. 1)

$g(n)$ is p.r. (why?) hence so is Inp ✓

Lemma 7: Define $T(m,n,c)$ iff $T(m)$ and TM coded by m has two letter alphabet $\{\sigma_1,\sigma_2\}$ and c is a code of a terminating computation (form) on input n
Then $T(m,n,c)$ is p.r.

Pf: $T(m,n,c)$ iff $T(m) \wedge (m)_0 = 2 \wedge$
$TC(n,c) \wedge (c)_0 = Inp(n)$ ✓

Lemma 8 There is a p.r. function $U: \mathbb{N}\to\mathbb{N}$ s.t. for any $c\in\mathbb{N}$ that codes a terminating computation $((Q_{c_0}, \tau_0, k_0), \ldots, (Q_{c_m}, \tau_m, k_m))$ of <u>some</u> TM,

$U(c)$ = # of 1's on tape described by T_m

(if c not such a code, $U(c)$ defined but irrelevant).

PF: Define $R(i,z)$ iff $p_i | z \wedge p_i^2 \nmid z$

- clearly pr
- idea is: if z codes a tape descr of a string of 1's, $R(i,z)$ tells us if there's a 1 on square i.
- Let χ_R be char function (also p.r. of c)
- Define $f(z) = \sum_{i \leq z} (1 - \chi_R(i,z))$

  = # of primes that divide z but squares don't

  (so if z is a tape code, $f(z)$ = # of 1's on tape)
- clearly p.r.
- define $U(c) = f(((c)_{\text{enc}(-1)})_1)$

  $(c)_{\text{enc}(-1)}$: extracts final sit. descr

  $(\ )_1$: extracts last tape descr

  f: counts 1's

Theorem (Kleene normal form theorem)
There is a p.r. function $U: \mathbb{N} \to \mathbb{N}$ and a p.r. rel'n $T(e,x,c)$
(... same as above just writing $T(e,x,c)$ to match std notation)
s.t. for any recursive function $f: \mathbb{N}^n \to \mathbb{N}$, there is an e s.t.
$f(k_1,..,k_n) = U(\mu c\, T(e, \langle k_1,..,k_n\rangle, c))$

How to read: There is a (fixed!) pr. func U and rel'n s.t.
for any n-ary recursive f
there is a code e for a TM computing f, s.t. $\forall$ inputs $\vec{k}$
$f(\vec{k})$ = output of shortest halting computation of the TM on input $\langle \vec{k} \rangle$
(if such a computation exists, i.e. $f(\vec{k})\downarrow$)

Pf. - just need to code a given n-ary f into a unary f' then use our lemmas
- given such an f, define
$f'(x) = f((x)_0,..,(x)_{n-1})$
(... so that if x codes a sequence $\vec{x}$ of length n then $f'(x) = f(\vec{x})$)
- then f' is recursive too
- hence there is a TM M, may assume of alphabet $\{\sigma_1, \sigma_2\}$, with $e_M^1 N = f$