Note: Lemma 2, which says $f \subseteq g \wedge P(f) \Rightarrow P(g)$ for r.e. P, implies Rice's theorem.

Why: Sps $P(f)$ is recursive property of rec. function, then $\neg P$ also recursive

- both $P, \neg P$ monotone upward by Lemma 2
- one of them contains the empty function $\emptyset$, hence all rec. functions ✓

Ex: Define $P(f) \Leftrightarrow f$ is total and surjective onto $\mathbb{N}$.

- P is not recursive by Rice, since nontrivial (some rec. functions are in P, some are not)
- not r.e. either: by Rice-Shapiro: no finite function has property P.
- is it co-r.e?

We claim no:

- if $\neg P$ is r.e. then monotone up by Lemma 2.
- but clearly $\emptyset \in \neg P$
- hence $\neg P$ = all recursive functions, which is clearly a contradiction ✓

e.g. $f(n) = n$ not in $\neg P$ (i.e. in P)

- Note: extension of Rice-Shapiro to n-ary properties of recursive functions $P(f_1, \ldots, f_n)$ also holds.
- e.g. TFAE
  1. $P(f_1, f_2)$ is r.e.
  2. $P(f_1, f_2)$ iff $\exists \pi_1 \subseteq f_1$, $\pi_2 \subseteq f_2$ finite s.t. $P(\pi_1, \pi_2)$ and $\{(x,y) : P(\pi_x, \pi_y)\}$ is r.e.

etc...

Theorem (Second recursion theorem) (Kleene):

- Sps φ is an acceptable enum. of the rec. functions
- then for any recursive function $F(e, x_1, \ldots, x_n)$ there is $e \in \mathbb{N}$ s.t. $\varphi_e(\vec{x}) = F(e, \vec{x})$ (.. for all $\vec{x}$)

↪ How to read: can view a non-unary rec function $F(y, \vec{x})$ as an enumeration of its strands:

- i.e. for fixed e, define $F_e(\vec{x}) = F(e, \vec{x})$
- thm says: if we do this, then for some index e, the F enumeration agrees w/ our φ enumeration, i.e. $\varphi_e = F_e$

Pf: - Let S be our function s.t.
$\varphi_{S(e,x)}(y) = \varphi_e(x,y)$ (φ is acceptable)
- consider many function
$e(d) = S(d,d)$
↑ (= code for $\varphi_d(d,\cdots)$)
clearly recursive, since S is
- consider $F'(d,\vec{x}) := F(S(d,d),\vec{x})$
↑ clearly recursive
- Let d_0 be code for F'
- so $\varphi_{d_0}(d,\vec{x}) = F(S(d,d),\vec{x})$
- then $\varphi_{d_0}(d_0,\vec{x}) = F(S(d_0,d_0),\vec{x})$
- but we have
$\varphi_{d_0}(d_0,\vec{x}) = \varphi(d_0,d_0,\vec{x}) = \varphi(S(d_0,d_0),\vec{x})$
- so letting $e = S(d_0,d_0)$ we have
$\varphi_e(\vec{x}) = \varphi(e,\vec{x})$
$= F(e,\vec{x})$ (for all $\vec{x}$), as desired

Thm (Second recursion theorem; second form) Let $h: \mathbb{N} \to \mathbb{N}$ be a total recursive function and φ an acceptable enumeration.
Then for some n we have $\varphi_n = \varphi_{h(n)}$

How to read: no matter how we relabel (recursively, via h) our functions φ_e (changing e to $h(e)$) there is always some function w/ same label as before

e.g. if $h(e) = e+1$ then thm says $\varphi_e = \varphi_{e+1}$ for some e (some function occurs twice in a row)

Pf. Let $F(e,x) = \varphi(h(e), x)$
by first form can find n s.t.
$\varphi_n(x) = F(n,x) = \varphi(h(n), x)$
$= \varphi_{h(n)}(x)$ ✓

- Conversely: if we assume second form, we can get first.
- Given $F(e,\vec{x}) = F_e(\vec{x})$ want to find e s.t. $F_e = \varphi_e$
- Since F rec. let e_0 be code for F: $\varphi_{e_0} = F$
- i.e. $\varphi_{e_0}(e,x) = F(e,x)$
- i.e. $\varphi_{s(e_0,e)}(x) = F_e(x)$
$\varphi_{s(e_0,e)} = F_e$
- Let $h(e) = s(e_0,e)$
- then by second form can find e s.t. $\varphi_e = \varphi_{h(e)}$
$= F_e$ ✓

- We can use the 2nd recursion thm to show that certain functions that are defined by "apparently" recursive definitions are in fact recursive.

- ... without having to go thru rigmarole of defining them from the initial functions using comp'n, prim. rec'n, μ-search.

Ex: Recall: Ackermann function $A(n,x)$ is defined by:

$$A(n,x) = \begin{cases} x+1 & n=0 \\ A(n-1,1) & n>0, x=0 \\ A(n-1, A(n,x-1)) & n,x>0 \end{cases}$$

- We showed A is recursive by long-winded arg:
  - ↳ computation trees for A are p.r.
  - ↳ can do a μ-search over codes for comp'n trees ...

- Using second recursion theorem can get a much quicker proof.
- Idea is: want to find a code for A, i.e. e s.t. $\varphi_e = A$.
- So we want e s.t.

$$\varphi_e(n,x) = \begin{cases} x+1 & n=0 \\ \varphi_e(n-1,1) & n>0, x=0 \\ \varphi_e(n-1, \varphi_e(n,x-1)) & n,x>0 \end{cases}$$

- This is a self-referential def'n

- here's the trick:
- define $F(e,n,x)$

$$= \begin{cases} x+1 & n=0 \\ \varphi_e(n-1,1) & n>0,\ x=0 \\ \varphi_e(n-1,\varphi_e(n,x-1)) & x,n>0 \end{cases}$$

$\varphi(e,n-1,1)$ ⟶

- not self-referential!
- just plugging into $\varphi(e,n,x) = \varphi_e(n,x)$ when we need to (e can vary)
- F clearly recursive
  (precisely recursive w/ recursive conditions)
- by second recursion thm $\exists e \in \mathbb{N}$ s.t. $F(e,n,x) = \varphi_e(n,x)$
- but then φ_e satisfies conditions defining A
- can show (by induction) there is at most one function satisfying these conditions; hence $\varphi_e = A$.
  ↗ hence recursive!

- in general. "self-referential" or "apparently recursive" defns like this may have more than one sol'n
- but 2nd recursion theorem can often be used to show a solution exists! (and is recursive).

Continuous effective operators

Recall: an operator $\Phi(\vec{x}, \vec{f}) = f$ is effective if there is tot. rec. F s.t.

$$\Phi(\vec{x}, \varphi_{e_1}, \ldots, \varphi_{e_m}) = \varphi_{F(\vec{x}, \vec{e})}$$

(where φ is fixed acceptable enum)

- For operator Φ, have associated functional $\Phi^*(\vec{x}, \vec{f}, \vec{y}) = \Phi(\vec{x}, \vec{f})(\vec{y})$
- Graph of functional is the rel'n $G(\vec{x}, \vec{f}, \vec{y}, z)$ iff $\Phi^*(\vec{x}, \vec{f}, \vec{y}) = z$

- We'll write G_Φ for coded graph:
  $G_\Phi(\vec{x}, \vec{e}, \vec{y}, z)$ iff $\Phi^*(\vec{x}, \varphi_{e_1}, \ldots, \varphi_{e_m}, \vec{y}) = z$

and say the graph of Φ^* is r.e. if G_Φ is r.e.

- Also have F_Φ (as before):
  $F_\Phi(\vec{x}, \vec{e}, \vec{y}) = \Phi^*(\vec{x}, \varphi_{e_1}, \ldots \varphi_{e_m}, \vec{y})$

- G_Φ is (literal) graph of F_Φ, so F_Φ is recursive iff G_Φ is r.e.

Def'n An operator $\Phi(\vec{x}, \vec{f})$ is **continuous** if

$\Phi(\vec{x}, \vec{f})(\vec{y}) = z$ if and only if for every f_i, $\exists$ a finite $\pi_i \subseteq f_i$ s.t. $\Phi(\vec{x}, \vec{\pi})(\vec{y}) = z$

How to read: For a given input $\vec{y}$ [illegible] if $\Phi(\vec{x}, \vec{f})(\vec{y}) \downarrow$ need only a finite amount of info about the f_i's to compute $\Phi(\vec{x}, \vec{f})(\vec{y})$

Def'n