1 Recursiveness and computability

1.1 Recursive functions

Definition 0.1 (Primitive recursive function)**.**

Proposition 0.1 (Primitive recursive functions)**.**
The following are primitive recursive:

- Addition: $x + 0 = x, x + S(y) = S(x + y)$
- Predecessor: $pd(0) = 0, pd(x + 1) = x$
- Integer difference: $x - 0 = x, x - (y + 1) = pd(x - y)$

Proof. • Define $\begin{cases} f(x,0) = I_1^1(x) \\ f(x,y+1) = h(x,y,f(x,y)) \\ h(x,y,z) = S(I_3^3(x,y,z)) \end{cases}$

□

Definition 0.2 (Recursive μ-operator)**.**
content...

Definition 0.3 (Recursive function)**.**
A function f is defined from a relation R by μ-recursion if:

1. R is a regular predicate, i.e. $\forall \overrightarrow{x}.\exists y.R(\overrightarrow{x}, y)$
2. $f(\overrightarrow{x}) = \mu y.R(\overrightarrow{x}, y)$

Theorem 1 (Kleene's Normal Form Theorem)**.**
There exist:

- U primitive recursive function
- for each n, primitive recursive predicates T_n

such that for any recursive function $f^{(n)}$ there is a number e for which:

- $\forall \overrightarrow{x}.\exists y.T_n(e, \overrightarrow{x}, y)$

- $f(\overrightarrow{x}) = U(\mu y.T_n(e, \overrightarrow{x}, y))$

Proof. Let $f^{(n)}$ be a recursive function. This function can be obtained algorithmically in different forms according to the inductive procedure that we use to generate recursive functions. Once we fix this procedure, we have fixed an algorithmic description of every recursive function. This description is in essence a sequence of elementary steps or computations. The theorem says that for any input $\overrightarrow{x}$, there is a computation giving the value of f at $\overrightarrow{x}$ and that the final result can be obtained through primitive recursive operations using at most one minimization. The function U plays the role of an interpreter, extracting the result value from the computation code. Note also that programs are treated as data, i.e. recursive functions are also reduced to numbers via indexes. This is done so that we can define numeric functions U and T_n.

Recursive function coding:

$O \mapsto \langle 0 \rangle$

$S \mapsto \langle 1 \rangle$

$I_i^n \mapsto \langle 2, n, i \rangle$

$g(h_1(\cdot), \ldots, h_m(\cdot)) \mapsto \langle 3, b_1, \ldots, b_m, a \rangle$

prim. recursion$(g, h) \mapsto \langle 4, a, b \rangle$

bounded recusion$(g) \mapsto \langle 5, a \rangle$ □

Corollary 1.1.
The set of recursive functions is countable.

1.2 Recursive sets

Definition 1.1 (Recursive set)**.**
A set is recursive iff its characteristic function is recursive.

2 Basic recursion theory

2.1 Partial recursive functions

Definition 1.2 (Partial function)**.**
A partial function is a function that may be undefined form some arguments. The set of arguments for which it is defined is called its domain.

Definition 1.3 (Partial recursive μ-operator)**.**
A function f is defined from a relation R by μ-recursion if $f(\overrightarrow{x}) \simeq \mu y.R(\overrightarrow{x}, y)$.

Definition 1.4 (Partial recursive functions)**.**
The class of partial recursive functions is the smallest class of functions
1. Containing the inital functions O, S, I_i^n
2. Closed under composition: $\varphi(\overrightarrow{x}) \simeq \psi(\gamma_1(\overrightarrow{x}), \ldots, \gamma_m(\overrightarrow{x}))$
3. Closed under primitive recursion:

$$\varphi(\overrightarrow{x}, 0) \simeq \psi(\overrightarrow{x}), \varphi(\overrightarrow{x}, y+1) = \gamma(\overrightarrow{x}, y, \varphi(\overrightarrow{x}, y))$$

4. Closed under unrestricted μ-recursion:

$$\varphi(\overrightarrow{x}) \simeq \mu y[\forall z \leq y.\psi(\overrightarrow{x}, z) \downarrow \wedge \psi(\overrightarrow{x}, y) \simeq 0]$$

Theorem 2 (Normal form theorem for partial recursive functions)**.**
There exist:

- U primitive recursive function
- for each n, primitive recursive predicates T_n

such that for any partial recursive function $\varphi^{(n)}$ there is a number e for which:

- $\varphi(x_1, \ldots, x_n) \downarrow \iff \exists y.T_n(e, \overrightarrow{x}, y)$
- $\varphi(\overrightarrow{x}) \simeq U(\mu y.T_n(e, \overrightarrow{x}, y))$

Corollary 2.1 (Recursive as partial recursive)**.**
The recursive functions are exactly the total and partial recursive functions.

Recall the normal form expression $U(\mu y.T_n(e, \overrightarrow{x}, y))$. In the total setting, it is shown that for e an index of a total recursive function, this gives a value. In

particular, the minimized predicate must be regular. But what if e does not correspond to any recursive function? The predicate may not be minimizable, i.e. the corresponding search is not terminating. This problem is removed with the particular μ-operator. In that setting, $\varphi_e^n(\overrightarrow{x}) \simeq U(\mu y.T_n(e, \overrightarrow{x}, y))$ always defines a partial function.

Theorem 3 (Enumeration theorem)**.**
For each e, φ_e^n is a partial recursive function of n variables.

If ψ is a partial recursive function of n variables, then there is e such that $\psi \simeq \varphi_e^n$

There is a partial recursive function $\varphi^{(n+1)}(e, \overrightarrow{x}) \simeq \varphi_e^n(\overrightarrow{x})$

In summary, φ is a partial recursive function that enumerates all partial recursive functions with its parameter e.

Theorem 4 (Padding lemma)**.**
Let e be an index for a partial recursive function φ.

We can effectively generate infinitely many indices of the same function.

Proof. It would suffice to apply the identity function to the coding of φ or to add and subtract the same quantity. Note that the computation trees appearing in the proof of the normal form theorem have equations as nodes. So the proof is phrased in terms of "adding redundant equations" □

Theorem 5 (S_n^m/parametrization theorem)**.**
Let $m, n \in \omega$. There is a primitive recursive, injective function S_n^m such that:

$$\varphi_{S_n^m(e,x_1,\ldots,x_n)}(y_1, \ldots, y_m) \simeq \varphi_e(x_1, \ldots, x_n, y_1, \ldots, y_m)$$

Proof. We take a partial recursive function coded by a program e and we assume that some of its parameters $x_1, \ldots, x_n$ are fixed. The result is some function γ. The theorem says that we can compute a program $S_n^m(e, x_1, \ldots, x_n)$ using only primitive recursive means.

The computation of γ can be described by the function coded by e followed by the equation $\gamma(y_1, \ldots, y_m) \simeq \psi(C_{x_1}, \ldots, C_{x_n}, y_1, \ldots, y_m)$ (recall the computation trees). This function is clearly partial recursive and its index depends on the

code e and the fixed constant, call it $S_n^m(e, x_1, \ldots, x_n)$. By construction the index function is injective and its arithmetization is clearly primitive recursive. □

Note that the enumeration and parametrization theorems play an inverse role translating arguments to indices and viceversa. This is the essence of partial evaluation and supercompilation.

2.2 Recursively enumerable sets

We find the set analog of partial recursive function. Since characteristic functions are total we need to find an alternative definition.

Definition 5.1 (Recursively enumerable set)**.**
A set is recursively enumerable iff its the domain of a partial recursive function.

Notation 5.1.
Note W_e^n the domain of φ_e^n and $W_{e,s}^n$ the domain of $\varphi_{e,s}^n$.

Theorem 6 (Normal form theorem for r.e. relations)**.**
An n-ary relation $P^{(n)}$ is r.e. iff one of the following holds:

- For some recursive relation R:

$$P^{(n)} = \{\overrightarrow{x}.\exists y.R^{(n+1)}(\overrightarrow{x}, y)\}$$

- For some index e:

$$\exists e.P^{(n)} = W_e^n = \{\overrightarrow{x}.\exists y.T_n(e, \overrightarrow{x}, y)\}$$

Theorem 7 (Numerical characterization of r.e. sets)**.**
The r.e. sets are exactly the diophantine sets, i.e. sets of the form:

$$\{a.\exists \overrightarrow{x}.p(a, \overrightarrow{x}) = 0\}$$

where p is a polynomial with integer coefficients and $a \in \mathbb{Z}$, $\overrightarrow{x} \in \mathbb{Z}^n$. As a consequence there is no *algorithm* solving Hilbert's tenth problem.

TODO: there should be a match between this set and the expression of theorem 6. But they appear to list different kinds of elements.

Theorem 8 (Graph theorem)**.**
A function φ is partial recursive iff its graph is r.e.

A function f recursive iff its graph is recursive.

The theorem confirms that the choice of the *domain* is good for the canonical translation from sets to functions given by characteristic functions and from functions to sets given by graph.

But the original definition of being the domain of a partial recursive function can be made even more precise.

Theorem 9 (Uniformization property)**.**
Let P be a r.e. set.

$\exists \varphi \in \mathcal{P}. \forall \overrightarrow{x}. (\exists y. P(\overrightarrow{x}, y) \implies \varphi(\overrightarrow{x}) \downarrow \wedge P(\overrightarrow{x}, \varphi(\overrightarrow{x}))$

Furthermore, if P is regular, i.e. $\forall \overrightarrow{x}. \exists y. P(\overrightarrow{x}, y)$, then:

$\exists f \in \mathcal{R}. \forall \overrightarrow{x}. P(\overrightarrow{x}, f(\overrightarrow{x}))$

Theorem 10 (Characterization by enumeration properties)**.**
A is r.e $\iff \exists \varphi \in \mathcal{P}. A = range(\varphi) \iff A = \emptyset \vee \exists f \in \mathcal{R}. A = range(f)$

A is recursive $\iff A = \emptyset \vee \exists f \in \mathcal{R}.$ f is non-decreasing and $A = range(f)$

Thus, we see that recursive sets are those for which a *pattern* in membership can be described by a finite procedure (a decision procedure). Recursively enumerable sets have a pattern for membership but have no finite pattern for non-membership (just list the elements, via a generating procedure). In this sense, they are *half-recursive*. This idea also appears in the following:

Theorem 11 (Post's theorem)**.**
A is recursive iff A and $\overline{A}$ are recursively enumerable.

We may be interested in studying particular classes of programs and provide decision procedures or generating procedures for them. This may not be possible for a particular class of infinite programs.

Definition 11.1 (Inmune set)**.**
An infinite set without infinite r.e. subsets are called immune.

If we have a generating procedure for some set we also have a decision procedure for some other subset.

Proposition 11.1 (Infinite decided set of an enumerated set)**.**
Every infinite r.e. set has an infinite recursive subset.

Theorem 12 (Set theoretical structure of r.e. and recursive sets)**.**
1. R.e. sets form a distributive lattice (inclusion as order) with smallest and greatest element, and with the recursive sets as the only complemented elements.

2. The property of being r.e. is preserved under images and inverse images via partial recursive functions.

3.The recursive sets form a Boolean algebra (inclusion as order).

4. Recusrive sets are preserved under inverse images via recursive functions.

5. The union of two r.e. sets can be reduced to the union of two disjoint r.e. sets.

2.3 Diagonalization

Proposition 12.1 (Diagonal method)**.**
Given:

- a set S
- a function $d : S \to S$ such that $\forall a \in S.d(a) \neq a$
- an infinite matrix $(a_{i,j})$ in S.

the transformed diagonal $\{d(a_{n,n})\}_{n\in\omega}$ differs from every row $\{a_{i,j}\}_{j\in\omega}$.

Proof. The i-th element of the i-th row is different from the i-th element of the transformed diagonal, i.e. $d(a_{i,i}) \neq a_{i,i}$. □

Note that the diagonal method is a transformational method, in the sense that, we study properties that remain invariant under a class transformations. The transformation on the diagonal is d, the transformation of the rows are the identities. The property that is preserved is that the diagonal and the transformed diagonal do not match any of the rows.

The method of proof seems to encode the *recursivity* or *self-reference* as the repetition of a variable on a table. But other than that all aspects of the method may be generalized and one maybe asked to change d and the relevant subset of the matrix where other invariants holds. In this sense, the diagonalization method may be seen as **the first proof method in the transformational approach**.

2.3.1 Undecidability results

Proposition 12.2 (Recursive version of Cantor's theorem)**.**
No recursive function enumerates the recursive $0,1$-valued functions.

Proof. We will give a direct proof and a diagonalization proof.

1. Let f be a recursive function such that every function in the canonical enumeration (i.e. the only sort of enumeration we refer to in the statement of the lemma) $\{\varphi_{f(x)}\}_{x\in\omega}$ is recursive and $0,1$-valued. To reach a contradiction, define $g\cdot \simeq 1-\varphi_{f}\cdot\cdot$. We observe that:

- g is $\{0,1\}$-valued.
- g is total since f is and the enumeration is over total functions.
- g is partial recursive since substraction is and the enumeration theorem ensures that $\varphi_{f(x)}$ is also.

By the corollary to the normal form theorem, we have g is $\{0,1\}$-valued and total recursive. However, if there is an x where $g=\varphi_{f(x)}$ then $\varphi_{f(x)}(x) = 1-\varphi_{f(x)}(x)$ which is impossible since $\varphi_{f(x)}(x)$ is $\{0,1\}$-valued.

2. Take $S=\{0,1\}, d(a)=1-a$ which satisfies $\forall a\in S.d(a)\neq a$, finally take $a_{i,j}=\varphi_{f(i)}(j)$ so that on each row we picture the complete sequence of a given listed $\{0,1\}$-valued recursive function $\varphi_{f(i)}$. The diagonal method tells us that the diagonal sequence $\{d(a_{i,i})\}=\{\varphi_{f(i)}(i)\}$ does not correspond to

any row. So if we set $g(i) = 1 - \varphi_{f(i)}(i)$ it satisfies the three points above and is not in the list. Contradiction. $\square$

Corollary 12.1 ($\mathcal{R}$ are not r.e.)**.**
$\{x.\varphi_x \in \mathcal{R}\}$ is not r.e.

Proof. Note that using the substraction operation where $n - m = 0$ whenever $m \geq n$ the proof of the above theorem stays the same removing the condition that $\varphi_{f(x)}$ needs to be $\{0,1\}$-valued. So we have that no recursive function enumerating all recursive functions since there is always one $\{0,1\}$-valued function escaping. This is the same as saying that $\{x.\varphi_x \in \mathcal{R}\}$ is not r.e. $\square$

This happens in contrast with partial recursive and primitive recursive functions which give sets of codes which are r.e.

Theorem 13 (Combinatorial core of the undecidability results)**.**
$\{x.x \in W_x\} = \{x.\varphi_x(x) \downarrow\}$ is r.e. and nonrecursive.

The following is a strengthening of the above theorem.

Definition 13.1 (Recursive separability)**.**
Let A, B be disjoint sets.

A, B are recursively separable if $\exists C.A \subseteq C \wedge B \subseteq \overline{C}$.

Theorem 14 (Stronger combinatorial core)**.**
There are two disjoint, r.e. and recusively inseparable sets.

Another reformulation is the undecidability of the halting program. Originally stated, it says that no Turing machine can decide whether a universal Turing machine halts or not on given arguments.

Theorem 15 (Undecidability of halting problem)**.**
$\{\langle x, e\rangle.x \in W_e\} = \{\langle x, e\rangle.\varphi_e(x) \downarrow\}$ is r.e. and nonrecursive.

The halting problem is a first example where we study the solvability of properties of programs. To this end we may want to introduce notation to refer to sets that are generated by these programs.

Definition 15.1 (Index set)**.**
Let $\mathcal{A}$ a class of $\mathcal{P}$.

$A = \theta\mathcal{A} = \{x.\varphi_x \in \mathcal{A}\}$ is the index set of $\mathcal{A}$.

$\mathcal{A}$ is completely recursive if A is recursive.

Complete recursivity denotes those sets of programs for which decision procedures exists, i.e. procedures that will find them and rule out any other program.

Theorem 16 (Rice's theorem)**.**
$\mathcal{A}$ is completely recursive iff $\mathcal{A} = \emptyset \vee \mathcal{A} = \mathcal{P}$.

For instance, one could consider a finite set of programs. We can easily find a program listing them. But given another program, we also need to check it does not correspond to the behaviour of one in the list. This implies in particular solving halting problems for certain inputs.

For sufficiently complicated programs, the program itself is its best description. Because at the end, to decide each possible property P_i of the program I need a specific algorithm A_i. This is why we say that the behaviour of a mechanism (an object whose local behaviour is understood) does not need to be globally predictable. This occurs in any science whose basic principles are defined but their outcome may not.

2.3.2 Fixed-point theorem

The following theorem shows that $\mathcal{P}$ has a built-in defense against diagonalization (??, TODO).

Theorem 17 (Fixed-point theorem)**.**
$\forall f \in \mathcal{R}.\exists e.\varphi_e \simeq \varphi_{f(e)}$

Proof. 1. □

2.4 Indices and enumerations

Definition 17.1 (System of indices)**.**
A system of indices ψ is a family of maps $\psi^n : \omega \to \mathcal{P}^n$.

Note $\psi_e^n := \psi^n(e)$.

ψ satisfies enumeration if $\forall n.\exists a.\psi_a^{n+1}(e, x_1, \ldots, x_n) \simeq \psi_e^n(x_1, \ldots, x_n)$.

ψ satisfies parametrization if $\forall m, n.\exists s$ total recursive.$\psi^m_{s(e,x_1,\ldots,x_n)}(y_1, \ldots, y_m) \simeq \psi_e^{m+n}(x_1, \ldots, x_n, y_1, \ldots, y_m)$

Note φ the standard system of indices.

ψ is acceptable if $\exists f, g$ total recursive.$\psi_e^n \simeq \varphi^n_{f(e)}, \varphi_e^n \simeq \psi^n_{g(e)}$

Proposition 17.1 (Acceptable system characterization)**.**
A system of indices is acceptable $\iff$ it satisfies enumeration and parametrization.

3 Measures of complexity

4 Observable phenomena

4.1 Self-reference

The Fixed-point theorem predicts the existence of quines:

References