Computability Theory

Prof. Dr. Thomas Streicher

SS 2002

Introduction

For knowing that a function $f : \mathbb{N}^k \to \mathbb{N}$ is computable one does not need a definition of *what is computable in principle* simply because one recognizes an algorithm whenever one sees it.[1] However, for showing that f is not computable one definitly has to delineate *a priori* the collection of functions (on natural numbers) that are computable *in principle*. A stable such notion was found in the 1930ies by some of the pioneers of mathematical logic as e.g. S. C. Kleene, A. Church, A. Turing etc. The various different formalizations of the concept were proven to be equivalent which led A. Church to formulate his famous *Church's Thesis* saying that all these equivalent formalizations actually do capture the intuitive notion "computable in principle".
One should notice that computability in principle is fairly different from "feasible computation" where certain bounds (depending on the size of input) are required for the amount of time and space consumed by the execution of an algorithm. The former is a highly developed branch of mathematical logic whereas the latter does not have such a definite shape as some of the main questions have not been solved (e.g. the P = NP problem which has been declared as one of the outstanding mathematical problems for the 21st century).
In this lecture we concentrate on general computability theory whose results are already fairly old and well known. But they are most useful and every computer scientist should know at least the basic results because they clearly delineate the limits of his business. Moreover, a certain amount of basic knowledge in computability theory is indispensible for almost every branch of mathematical logic prominently including theoretical computer science.
It is fair to say that computability theory is actually rather a theory of what is *not computable*. For example the most basic result is the famous undecidability of the halting problem saying that there is no algorithm deciding whether for a given program P and input n the execution of $P(n)$ terminates. Of course, the halting problem is semi-decidable, i.e., one can find out that $P(n)$ terminates simply by running the program. One can show that any semi-decidable property can be decided using a hypothetical decision procedure for the halting problem. In this sense the halting problem is the most difficult semi-decidable property as all other can be *reduced* to it. Via

[1] Alas, for a given algorithm it is not easy at all do decide or verify whether it meets a given specification. Actually, as we shall see later this is an *undecidable* problem!

this notion of *reduction* computability theory allows one to scale undecidable problems according to so-called *degrees of unsolvability*. This is an important branch of computability theory which, however, we don't follow in full detail. Instead, after introducing the first notions and most basic results, we work towards the *Rice-Shapiro Theorem* providing a full chararcterisation of all semidecidable properties of programs which respect extensional equality, i.e., if two algorithms compute the same partial function then the first algorithm satisfies the property if and only if the second does. A most pleasing aspect of the Rice-Shapiro Theorem is that almost all undecidability results about extensional properties of programs are immediate corollaries of it. Furthermore, the Rice-Shapiro Theorem gives rise to the Myhill-Shepherdson Theorem characterising the computable type 2 functionals which take computable partial functions as arguments and deliver numbers as results in case of termination as continuous functionals satisfying a certain effectivity requirement. This inherent continuity property of type 2 functionals was taken as a starting point by D. Scott end of the 1960ies when he developed his Domain Theory as a mathematical foundation for the denotational semantics of programming languages.
These results are the cornerstones of our lectures on computability theory. They are accompanied by a few further results which are intrinsic for the metamathematics of constructive logic and mathematics.

1 Universal Register Machines (URMs)

In this section we give one possible formalization of the concept of computable function which is fairly close to (an idealized version of) the most common programming languages like BASIC.
There is a lot of other provably equivalent characterisations having their own merits. This coincidence of different attempts of formalizing the notion of computable function supports *Church's Thesis* saying that the notion of computabe function coincides with any of these mathematical formalisations. Often we use Church's Thesis as an *informal* justification for the existence of a program for a function which obviously is computable in the intuitive sense. Such a sloppy style of argument is adopted in most books and papers and we make no exception. For the novice this might be a bit unpleasant at the beginning but you surely will get used to it as generations before. Why should one be in logic more pedantic than in other fields of mathematics?

When you first saw Gauss' algorithm it was clear to you that it can be implemented in some programming language. Well, Gauss' algorithm is a useful thing for which it is worthwhile to write a program. However, in computability theory for most arguments one just needs the existence of an algorithm which, moreover, often is also relative to certain hypothetical assumptions. Accordingly, it's not worth the effort to write programs that never will be executed.
Now to the promised formalization of computable function.

Definition 1.1 (URM)
The Universal Register Machine (URM) *has infinitely many* storage cells

$$R_0, R_1, \ldots, R_n, \ldots$$

also called registers *which can store arbitrarily big natural numbers. A* state *is a function σ assigning a natural number $\sigma(R_n)$ to each register R_n where it is assumed that $\sigma(R_n) = 0$ for almost all $n \in \mathbb{N}$. We call $\sigma(R_n)$ the contents of R_n in state σ.*
An URM-program *is a finite list of commands*

$$P \equiv C_0 \ldots C_{n_p-1}$$

where the commands C_i are of one of the following four kinds

a) $Z(n)$ *meaning* "put the contents of R_n to 0" (R_n:=0)

b) $S(n)$ *meaning* "increase the contents of R_n by 1" (R_n:=R_n+1)

c) $T(m, n)$ *meaning* "transfer the contents of R_m to R_n without changing the contents of R_m" (R_n:=R_m)

d) $I(n, m, k)$ *meaning* "if $R_n = R_m$ then goto C_k and otherwise execute the next command" (`if` R_n=R_m `goto` k).

If one starts program P in state σ then this leads to a sequence of configurations

$$(\sigma, 0) = (\sigma_0, \ell_0) \to (\sigma_1, \ell_1) \to \ldots \to (\sigma_k, \ell_k) \to (\sigma_{k+1}, \ell_{k+1}) \to \ldots$$

which can be finite or infinite. The first component of a configuration (σ, ℓ) is the current state σ and its second component is the number ℓ of the command

of P which has to be executed next. How $(\sigma_{k+1}, \ell_{k+1})$ is obtained from (σ_k, ℓ_k) is specified by the above informal descriptions of the four kinds of commands. A configuration (σ, ℓ) is an end configuration w.r.t. *P iff one of the following three conditions is satisfied*

(i) $\ell \geq n_p$

(ii) $C_\ell \equiv I(n, m, k)$ *with* $\sigma(R_n) = \sigma(R_m)$ *and* $k \geq n_p$

(iii) $\ell = n_p - 1$ *with* $C_\ell \equiv I(n, m, k)$ *and* $\sigma(R_n) \neq \sigma(R_m)$.

A possibly partial function $f : \mathbb{N}^k \to \mathbb{N}$ is computed by an URM*-program P iff for all $\vec{a} = (a_0, \ldots, a_{k-1}) \in \mathbb{N}^k$ the program P started in state $\sigma_{\vec{a}} := (a_0, \ldots, a_{k-1}, 0, 0, 0 \ldots)$ terminates iff $f(\vec{a})$ is defined in which case $f(\vec{a})$ is the contents of R_0 in the final state of the execution.*
We call f URM-computable *iff there is an* URM*-program P computing f.* $\Diamond$

For every URM-computable function f there are infinitely many different URM-programs computing f (exercise!).
As there are only countably many commands there are also only countably many URM-programs. Accordingly, *there are only countably many* URM*-computable functions*! Thus, for cardinality reasons most functions over $\mathbb{N}$ are not URM-computable! Convince yourself that there are at least infinitely many URM-computable functions (exercise!).
Notice further, that every URM-program P during its execution can modify only those registers which are explicitly mentioned in one of the commands of P. Thus, for a particular program finitely many registers would suffice.

2 Partial Recursive Functions

Obviously, URM-programs are sort of idealized[2] assembler code. However, programming in assembly language is a cumbersome task because the pro-

[2]Idealized because we have assumed that registers can store arbitrarily big natural numbers. Compare this with Turing machines where each cell has only finite storing capacity (a letter of the alphabet under consideration). However, Turing machines are also idealized in the respect that there is a potential infinity of such limited cells. Thus, Turing machines can compute the same functions as URM-programs. But when programming a Turing machine the programmer is responsible for memory management. That's the reason why in computational complexity the Turing machine model is the preferred one.

grammer is in charge of organizing all details. Thus, we give now a characterisation of computable functionals which is much more abstract and, therefore, much more convenient.
The *partial recursive functions* will be defined inductively as a certain subset of the set

$$\bigcup_{k\in\mathbb{N}} [\mathbb{N}^k \rightharpoonup \mathbb{N}]$$

where $[X \rightharpoonup Y]$ stands for the set of partial functions from X to Y. This has the advantage that one doesn't have to worry about execution sequences or other nasty operational details of this kind and can all the time stay within the familair realm of sets and functions. Later we will sketch a proof of the fact that the partial recursive functions coincide with the URM-computable functions.

Definition 2.1 (*partial recursive functions)*
The set of partial recursive *or* μ-recursive *functions is defined inductively as the least subset* $\mathcal{P} \subseteq \bigcup_{k\in\mathbb{N}}[\mathbb{N}^k \rightharpoonup \mathbb{N}]$ *satisfying the following closure conditions*

(P1) *for every* $k \in \mathbb{N}$ *the function*

$$zero_k : \mathbb{N}^k \to \mathbb{N} : \vec{x} \mapsto 0$$

is in $\mathcal{P}$

(P2) *the function*

$$succ : \mathbb{N} \to \mathbb{N} : n \mapsto n{+}1$$

is in $\mathcal{P}$

(P3) *for all natural numbers* $i < n$ *the projection function*

$$pr^i_n : \mathbb{N}^n \to \mathbb{N} : (x_0, \ldots, x_{n-1}) \mapsto x_i$$

is in $\mathcal{P}$

(P4) *whenever* $f : \mathbb{N}^m \rightharpoonup \mathbb{N}$ *is in* $\mathcal{P}$ *and* $g_i : \mathbb{N}^n \rightharpoonup \mathbb{N}$ *are in* $\mathcal{P}$ *for* $i = 1, \ldots, m$ *then*

$$comp^m_n(f, g_1, \ldots, g_m) : \mathbb{N}^n \rightharpoonup \mathbb{N} : \vec{x} \mapsto f(g_1(\vec{x}), \ldots, g_m(\vec{x}))$$

is in $\mathcal{P}$*, too*

(P5) *whenever* $f : \mathbb{N}^n \rightharpoonup \mathbb{N}$ *and* $g : \mathbb{N}^{n+2} \rightharpoonup \mathbb{N}$ *are in* $\mathcal{P}$ *then* $R[f,g]$ *is in* $\mathcal{P}$*, too, where* $R[f,g]$ *is the unique* $h : \mathbb{N}^{n+1} \rightharpoonup \mathbb{N}$ *such that*

$$h(\vec{x},0) \simeq f(\vec{x}) \qquad h(\vec{x},n+1) \simeq g(\vec{x},n,h(\vec{x},n))$$

for all $\vec{x} \in \mathbb{N}^n$ *and* $n \in \mathbb{N}$

(P6) *whenever* $f : \mathbb{N}^{n+1} \rightharpoonup \mathbb{N}$ *is in* $\mathcal{P}$ *then* $\mu(f) : \mathbb{N}^n \rightharpoonup \mathbb{N}$ *is in* $\mathcal{P}$*, too, where* $\mu(f)$ *is the unique function* $h : \mathbb{N}^n \rightharpoonup \mathbb{N}$ *such that for all* $\vec{x} \in \mathbb{N}^n$ *and* $m \in \mathbb{N}$*,* $h(\vec{x}) = m$ *iff* $f(\vec{x},k) > 0$ [3] *for all* $k < m$ *and* $f(\vec{x},m) = 0$*.*

The least subset of $\bigcup_{k\in\mathbb{N}}[\mathbb{N}^k \rightharpoonup \mathbb{N}]$ *closed under* (P1)–(P5) *is called the class of* primitive recursive functions*. Obviously, the class of primitive recursive functions is contained in* $\bigcup_{k\in\mathbb{N}}[\mathbb{N}^k \to \mathbb{N}]$ *as total number theoretic functions are closed under* (P1)–(P5)*.* $\Diamond$

The schema (P5) is called *primitive recursion* and one easily shows by induction (on the last argument) that $R_n[f,g]$ is total whenever f and g are total. That (P1)–(P4) preserve totality of functions is obvious. Notice that primitive recursion is more general than *iteration* $It[a,f](n) = f^n(a)$ because in primitive recursion the step from n to $n+1$ depends on n and the parameters. Almost all functions considered in arithmetic are primitive recursive (as e.g. addition, multiplication, exponentition etc.). However, as we shall see later, not all *recursive* functions, i.e. total partial recursive functions, are primitive recursive. Actually, there is no programming language which allows one to implement precisely the recursive functions.
Obviously, the source of possible nontermination is the operator μ which allows one to *search without knowing* whether this search will eventually be successful. For example the function $\mu_0(succ) : \mathbb{N}^0 \rightharpoonup \mathbb{N}$ diverges, i.e. does not terminate. Notice that $\mu_k(f)(\vec{x}) = n$ does not mean only that n is the least number with $f(\vec{x},n) = 0$ but, moreover, that $f(\vec{x},m)$ is defined for all $m < n$. Otherwise $\mu_k(f)$ couldn't be implemented in general because it isn't decidable in general whether $f(\vec{x},m)\downarrow$. Typical instances of "searching without knowing" are for example

- searching for proofs in first order logic
- searching for integer solutions of a diophantine equation

[3] here $t > 0$ implies that t is defined for which we also write $t\downarrow$

as in both cases one can show that existence of a proof or a solution are undecidable properties.

Now we show that all partial recursive functions are URM-computable. A proof (sketch) of the reverse direction has to wait till section 4.

Theorem 2.1 *The partial recursive functions are all* URM*-computable.*

Proof: We proceed by induction on the definition of $\mathcal{P}$.
The cases of $zero_n$, $succ$ and pr^i_n are obvious as these functions can be easily implemented by an URM-program.
As every URM-program (independently from the input) reads and/or modifies only finitely many registers, i.e. uses only a finite fragment of the store, for every URM-computable function $f : \mathbb{N}^n \rightharpoonup \mathbb{N}$ there exists a program P such that

$$P(\sigma)(R_0) \simeq f(\vec{a})$$

whenever $\sigma(R_i) = a_i$ for $i < n$.
For this reason the composition of URM-computable partial functions is again URM-computable because one has arbitrary many registers available for storing the intermediate results. In order to compute

$$comp^m_n(f, g_1, \ldots, g_m)(\vec{a})$$

one first saves the input $\vec{a}$ to a region of the store not effected by the programs implementing f or some of the g_i. Then one successively computes the results of $g_0(\vec{a}), \ldots, g_{m-1}(\vec{a})$ and stores them in the "save" part of the store. If all these intermediary computations have terminated then store the intermediary results into the registers $R_0, \ldots, R_{m-1}$ and start computation of the program implementing f. Notice, however, that before using the URM-programs for f, $g_1, \ldots, g_m$ one has first to adapt the addresses in an appropriate way because URM-programs employ absolute addressing instead of relative addressing.
We leave it as an exercise to the reader to analogously verify that $R[f, g]$ can be implemented by an URM-program whenever this is the case for f and g.
Now if $f : \mathbb{N}^{n+1} \rightharpoonup \mathbb{N}$ is implemented by an URM-program P then $\mu(f)$ can be implemented as follows: first store the input to the save part of the store and and put a distinguished register Z to 0 which is also in the save part of the store; then start executing P after having copied the saved input and the contents of Z to the registers $R_0, \ldots, R_n$; if the computation terminates in a

state where the contents of R_0 is 0 then transfer the contents of Z to R_0 and terminate; otherwise increment Z by 1 and apply the same procedure again. □

3 Primitive Recursive Functions and Codings

In order to show that all URM-computable functions are μ-recursive we have to convince ourselves that the operational semantics of URM-programs can be formulated in terms of primitive recursive functions. Then using the μ-operator once one can search for codes of terminating computation sequences (which, of course, might fail if there doesn't exist any) and finally extract the result of the computation from this code. This will be described in the next section. In this section we argue in favour of the expressivity of primitive recursive functions.
It is a straightforward exercise to show that the following functions

- addition and multiplication
- "truncated subtraction" defined as

$$n \dot{-} m = \begin{cases} n - m & \text{if } n > m \\ 0 & \text{otherwise} \end{cases}$$

  for all $n, m \in \mathbb{N}$.

- integer division and remainder

are all primitive recursive. As a further even simpler example consider the predecessor function *pred* sending 0 to 0 and $n{+}1$ to n whose primitive recursive nature is exhibited by the equations

$$pred(0) = 0 \qquad\qquad pred(n{+}1) = n$$

for all $n \in \mathbb{N}$. More formally, we may write

$$pred(zero_0()) = zero_0() \qquad\qquad pred(succ(n)) = pr^0_2(n, pred(n))$$

and thus get $pred = R_0[zero_0, pr^0_2]$. If one isn't afraid of unnecessary work then one may write down codes for all primitive or partial recursive functions

this way using the notation of Definition 2.1. However, this wouldn't make things more readable and, therefore, we stick to first variant, i.e., simply write down the defining equations in ordinary mathematical format to argue in favour of primitive (of partial) recursiveness.
Let us consider some further examples. The signum function sg is defined by the equations

$$sg(0) = 0 \qquad\qquad sg(n{+}1) = 1$$

exhibiting its primitive recursive nature. Obviously, the function

$$leq(n, m) = sg(n \dot{-} m)$$

satisfies $leq(n, m) = 0$ if $n \leq m$ and $leq(n, m) = 1$ otherwise and, therefore, decides the predicate $\leq$. Accordingly, equality of natural numbers is decided by

$$eq(n, m) = leq(n, m) + leq(m, n)$$

which, obviously, is primitive recursive.
Moreover, primitive recursive functions are closed under case analysis as the function $cond : \mathbb{N}^3 \to \mathbb{N}$ defined by

$$cond(0, x, y) = x \qquad\qquad cond(n{+}1, x, y) = y$$

is obviously primitive recursive. That primitive recursive functions are closed under *iteration* follows immediately from the fact that the function $iter[f] : \mathbb{N}^2 \to \mathbb{N}$ with

$$iter[f](0, x) = x \qquad\qquad iter[f](n{+}1, x) = f(iter[f](n, x))$$

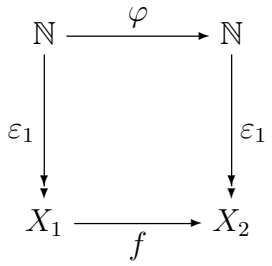
is primitive recursive whenever f is primitive recursive. As we shall see a bit later $\mathbb{N}^k$ and $\mathbb{N}$ are primitive recursively isomorphic. Thus, all functions which can be computed by `for`-loops with primitive recursive body are primitive recursive themselves. Thus, it is a simple exercise(!) to verify that for every primitive recursive $f : \mathbb{N}^{m+1} \to \mathbb{N}$ the function

$$\mu k{<}n.\ f(\vec{x}, k) = \mathsf{min}\{k \in \mathbb{N} \mid k = n \vee f(\vec{x}, k) = 0\}$$

is primitive recursive.

Next we consider a few codings of mathematical objects by numbers that will be used over and over again in the following.

A *numbering* or *coding* or *Goedelisation* of a set X is a *surjective* mapping $\varepsilon : \mathbb{N} \to X$. Notice that then X is necessarily countable. For numberings ε_1 and ε_2 of X_1 and X_2, respectively, a function $f : X \to Y$ is *computable* w.r.t. the numberings ε_1 and ε_2 iff there exists a computable function $\varphi : \mathbb{N} \to \mathbb{N}$ rendering the diagram



commutative. Very often, but not always (as for example in the case of $\mathcal{P}$ studied later on) numberings $\varepsilon : \mathbb{N} \to X$ can be chosen as bijective.

Now we study a few examples.

Let $\langle \_ , \_ \rangle : \mathbb{N}^2 \to \mathbb{N}$ be the primitive recursive function sending the pair (n, m) to the number $\langle n, m \rangle = 2^n(2m+1)-1$ which is obviously a bijection. Notice that $n, m \leq \langle n, m \rangle$ for all $n, m \in \mathbb{N}$. Thus, because bounded search is primitive recursive (as seen just a few lines above) the functions $\pi_0, \pi_1 : \mathbb{N} \to \mathbb{N}$ with

$$n = \langle \pi_0(n), \pi_1(n) \rangle$$

are primitive recursive (exercise!). Notice that this pairing function is exponential in the first argument but as remarked before complexity issues don't play any role in the context of the current lecture.[4]
This primitive recursive coding of pairs can be extended to finite sequences of natural numbers as follows. By recursion on k we define bijective functions $seq_k : \mathbb{N}^k \to \mathbb{N}$ as follows

$$seq_1(n) = n \qquad\qquad seq_{k+1}(n{\cdot}s) = \langle n, seq_k(s) \rangle$$

where $n{\cdot}s$ stands for the nonempty sequence whose head is n and whose tail is s. This gives rise to a bijection seq^+ from $\mathbb{N}^+ = \bigcup_{k=1}^{\infty} \mathbb{N}^k$ to $\mathbb{N}$ sending s

[4]However, inspired by Cantor's first diagonal argument we could choose a polynomial pairing function putting $\langle n, m \rangle = \frac{(n+m+1)(n+m)}{2} + n$.

to $seq^+(s) = \langle lgth(s), seq_{lgth(s)}(s)\rangle$ where $lgth(s)$ stands for the length of s. Finally we get a bijection seq from $\mathbb{N}^* = \bigcup_{k=0}^{\infty} \mathbb{N}^k$ to $\mathbb{N}$ putting

$$seq(\varepsilon) = 0 \qquad\qquad seq(s) = seq^+(s) + 1$$

where ε stands for the empty sequence. It is a lengthy, but straightforward exercise(!) that the usual operations on $\mathbb{N}^*$ like length, concatenation etc. can be implemented by primitive recursive functions on codes of sequences. Finite sets of natural numbers can be encoded via the bijective function

$$\varepsilon : \mathcal{P}_{\mathsf{fin}}(\mathbb{N}) \to \mathbb{N} : a \mapsto \sum_{i \in a} 2^i$$

and write e_n for the unique $a \in \mathcal{P}_{\mathsf{fin}}(\mathbb{N})$ with $n = \varepsilon(a)$. Thus, a natural number n codes the finite set a with $k \in a$ iff the kth digit in the binary expansion of n is 1. Again the usual operations and predicates on $\mathcal{P}_{\mathsf{fin}}(\mathbb{N})$ can be implemented by primitive recursive functions operating on codes.

4 Kleene's Normal Form Theorem and Admissible Numberings of $\mathcal{P}$

Now we convince ourselves of the fact that every URM-computable function is also partial recursive. Actually, we prove something slightly stronger, namely that one single use of μ is sufficient, which fact is also known as

Theorem 4.1 (Kleene's Normal Form Theorem) *For every natural number n there are primitive recursive functions $T_n : \mathbb{N}^{n+2} \to \mathbb{N}$ and $U_n : \mathbb{N} \to \mathbb{N}$ such that for every n-ary* URM*-computable function f there exists a natural number e such that*

$$f(\vec{x}) \simeq U_n(\mu k.\, T_n(e, \vec{x}, k))$$

for all $\vec{x} \in \mathbb{N}^n$.

Proof: We only sketch the proof idea and refer the reader who wants to know more details about coding to [Cut, TvD].
One easily can imagine how to code URM-programs by numbers as such programs are just finite sequences of commands which, obviously, can be coded by numbers. The configurations occuring during computations are

pairs (σ, ℓ) where σ is a memory state and ℓ is a natural number telling which command of the program under considerations has to be executed next. It is appropriate to code a memory state σ by the natural number $\ulcorner\sigma\urcorner$ with

$$\sigma(R_n) = \pi_0(\pi_1^n(\ulcorner\sigma\urcorner))$$

i.e. $\ulcorner\sigma\urcorner = \langle\sigma(R_0), \ldots \langle\sigma(R_n), 0\rangle \ldots\rangle$ where $\sigma(R_k) = 0$ for all $k > n$. Relative to the chosen encodings for a given program P and configuration (σ, ℓ) in a primitive recursive way one may decide whether (σ, ℓ) is a final configuration w.r.t. P and if not compute its successor configuration (σ', ℓ'). Obviously, one may also encode finite sequences of configurations and for these one may decide in a primitive recursive way whether for a given (code of) an URM-program and input $\vec{x}$

- every successor configuration arises from its predecessor configuration via P
- the last configuration of the sequence is final
- the first configuration equals $(\vec{x}0^\infty, 0)$.

Now let T_n be the corresponding primitive recursive function with $T_n(e, \vec{x}, k) = 0$ iff k codes a terminating computation sequence for the URM-program with code e and input $\vec{x}$. Let U_n be the primitive recursive function sending the code of a computation sequence with final configuration (σ, ℓ) to $\sigma(R_0)$, i.e., U extract the result of the computation. Obviously, if e is a code for a program computung f we have $f(\vec{x}) \simeq U_n(\mu k.\, T_n(e, \vec{x}, k))$ for all inputs $\vec{x}$ as desired. □

The primitive recursive functions T_n and U_n are called "Kleene's T-predicate" and "result extraction function", respectively. In the following we simply write T and U instead of T_1 and U_1, respectively.
Kleene's Normal Form Theorem immediately gives rise to the following important

Corollary 4.1 *All* URM*-computable functions are partial recursive.*

Proof: Let $f : \mathbb{N}^n \rightharpoonup \mathbb{N}$ be a function implemented by an URM-program with code e then we have

$$f(\vec{x}) \simeq U_n(\mu k.\, T_n(e, \vec{x}, k))$$

for all inputs $\vec{x}$ from which it follows by Kleene's Normal Form Theorem 4.1 that f is partial recursive. $\square$

Notice that it suffices to use μ-search at most once!
For all "machine-oriented" models of computation (as for example the famous Turing machine model discussed e.g. in [Cut, Rog]) the proof of equivalence with the partial recursive functions works according to the same pattern as we have seen for URM-programs:

(i) show that all partial recursive functions can be implemented in the language under consideration

(ii) code the programs and their operational semantics via functions T_n and U_n in Kleene's Normal Form Theorem.

In particular, such a coding is possible also for programming languages like BASIC, PASCAL, C etc. though the codings will get much more heavy than for the comparatively simple case of URM-programs.
However, luckily for the considerations in the rest of these lectures (and the theory of computability in general!) the precise nature of the functions T_n and U_n is fairly irrelevant as we need only their mere existence together with a few structural properties which we will discuss next.

Definition 4.1 *For every natural number n the partial recursive function $u_n : \mathbb{N}^{n+1} \rightharpoonup \mathbb{N}$ with*

$$u_n(e, \vec{x}) \simeq U_n(\mu k.\, T_n(e, \vec{x}, k))$$

for all $e \in \mathbb{N}$ and $\vec{x} \in \mathbb{N}^n$ is called universal for n-ary partial recursive functions. *For u_1 we write simply u and call it the* universal function.
For $e \in \mathbb{N}$ the partial recursive function $u_n(e, \_)$ mapping $\vec{x}$ to $u_n(e, \vec{x})$ will be denoted by $\varphi_e^{(n)}$ or $\{e\}^{(n)}$ and is called the e-th partial recursive function of arity n. *We simply write φ_e and $\{e\}$ for $\varphi_e^{(n)}$ and $\{e\}^{(n)}$, respectively. The notation $\{e\}$ is usually referred to as "Kleene brackets"*[5]. $\Diamond$

Notice that u_n may be understood as an *interpreter* for programs with n arguments and that for every partial recursive function f there are always

[5]It will always be clear from the context whether $\{e\}$ stands for the singleton set containing e or for φ_e

infinitely many e with $f = \varphi_e^{(n)}$ which fact is usually referred to as "padding" (i.e. adding irrelevant code).
Suppose that $f = \varphi_n^{(k+1)}$ and $m \in \mathbb{N}$ then the function $g : \mathbb{N}^k \rightharpoonup \mathbb{N}$ with

$$g(\vec{x}) \simeq f(m, \vec{x}) \simeq u_{k+1}(n, m, \vec{x})$$

is partial recursive, too, and actually a Goedel number for g can be computed from n and m in a primitive recursive way as claimed by the following

Theorem 4.2 (Parameter Theorem) *For every natural number k there is a primitive recursive function $s_k : \mathbb{N} \to \mathbb{N}$ such that*

$$\varphi_n^{(k+1)}(m, \vec{x}) \simeq \varphi_{s_k(\langle n,m \rangle)}^{(k)}(\vec{x})$$

for all $m \in \mathbb{N}$ and $\vec{x} \in \mathbb{N}^k$.

Proof: One easily can construct from a given URM-program P and $m \in \mathbb{N}$ a program which first shifts the contents of the first k registers to the right by 1, assigns m to R_0 and then executes the program P with addresses appropriatly shifted. Due to its simple nature this operation on programs can be implemented in terms of codes by an appropriately chosen primitive recursive function s_k. □

As for $k > 0$ the sets $\mathbb{N}^k$ and $\mathbb{N}$ are primitive recursively isomorphic one can always reduce k-ary functions to unary ones staying withing the realm of primitive recursiveness, total recursiveness or partial recursiveness. Thus, in the sequel w.l.o.g. we will consider all partial recursive functions as unary. As already announced above the rest of this section is devoted to an axiomatization of the Gödel numbering $\varphi = \varphi^{(1)}$ up to recursive equivalence. This numbering φ is *effective* or *computable* in the sense that there is an interpreter $u = u_1$ for φ, i.e. $\varphi_n(m) \simeq u(n, m)$ for all n, m. Furthermore, as an immediate consequence of the Parameter Theorem 4.2 we have

Theorem 4.3 (*snm*-Theorem) *There is a unary primitive recursive function s such that*

$$\varphi_n(\langle m, k \rangle) \simeq \varphi_{s(\langle n,m \rangle)}(k)$$

or equivalently

$$u(n, \langle m, k \rangle) \simeq u(s(\langle n, m \rangle), k)$$

for all natural numbers n, m, k.

Proof: One easily sees that there is a primitive recursive function t such that

$$u(n, \langle m, k \rangle) \simeq u_2(t(n), m, k)$$

for all n, m, k. Putting $s(x) \simeq s_1(\langle t(\pi_0(x)), \pi_1(x) \rangle)$ with s_1 as in the Parameter Theorem 4.2 we get

$$u(n, \langle m, k \rangle) \simeq u_2(t(n), m, k) \simeq u_1(s_1(\langle t(n), m \rangle, k)) \simeq u(s(\langle n, m \rangle), k)$$

as desired. □

Now we define the notion of *admissible* Goedel numbering of $\mathcal{P}$, the set of unary partial recursive functions.

Definition 4.2 (Admissible Gödel Numbering of $\mathcal{P}$) *An* admissible numbering of partial recursive functions *is a surjective function* $\psi : \mathbb{N} \stackrel{surj.}{\to} \mathcal{P}$ *such that*

(A1) *the binary partial function* $u^{(\psi)}(n, m) \simeq \psi_n(m)$ *is partial recursive and*

(A2) *there exists a total recursive function* $s : \mathbb{N} \to \mathbb{N}$ *such that*

$$\psi_n(\langle m, k \rangle) \simeq \psi_{s(\langle n,m \rangle)}(k)$$

for all $n, m, k \in \mathbb{N}$. ◊

Obviously, according to Theorems 4.1 and 4.3 the canonical numbering φ of $\mathcal{P}$ with

$$\varphi_n(m) \simeq U(\mu k.T(n, m, k))$$

is admissible in the sense of the above definition. The next theorem will show that in an appropriate sense this numbering is the only one.

Lemma 4.1 *Let* ψ *and* θ *be surjective functions from* $\mathbb{N}$ *to* $\mathcal{P}$ *such that* ψ *satisfies condition* (A1) *of Definition 4.2 and* θ *satisfies condition* (A2) *of Definition 4.2. Then there exists a total recursive function* $f : \mathbb{N} \to \mathbb{N}$ *with* $\theta \circ f = \psi$*, i.e.*

$$\theta_{f(n)} = \psi_n$$

for all $n \in \mathbb{N}$.

Proof: Due to the assumption on ψ the mapping $x \mapsto \psi_{\pi_o(x)}(\pi_1(x))$ is partial recursive. Thus, there exists a natural number e with

$$\theta_e(\langle n, m\rangle) \simeq \psi_n(m)$$

for all $n, m \in \mathbb{N}$. Due to the assumption on θ there is a total recursive function t with $\theta_{t(\langle n,m\rangle)}(k) \simeq \theta_n(\langle m, k\rangle)$ for all n, m, k. Now putting $f(n) = t(\langle e, n\rangle)$ we obtain

$$\theta_{f(n)}(m) \simeq \theta_{t(\langle e,n\rangle)}(m) \simeq \theta_e(\langle n, m\rangle) \simeq \psi_n(m)$$

as desired. □

As an immediate consequence we get the following

Theorem 4.4 *Any two admissible Gödel numberings ψ and θ of $\mathcal{P}$ are* recursively equivalent*, i.e., there exist total recursive functions f and g with $\theta \circ f = \psi$ and $\psi \circ g = \theta$. In particular, every admissible Gödel numbering of $\mathcal{P}$ is recursively equivalent to the canonical numbering φ.*

Proof: Apply Lemma 4.1 twice! □

Admissible numberings of $\mathcal{P}$ may be most naturally understood as universal[6] programming languages in the following sense. If one considers φ as the machine language then an admissible numbering, i.e. a function $\psi : \mathbb{N} \stackrel{surj.}{\to} \mathcal{P}$ with total recursive $f, g : \mathbb{N} \to \mathbb{N}$ such that $\varphi \circ f = \psi$ and $\psi \circ g = \varphi$, can be understood as a notation system for partial recursive functions such that

- ψ-programs can be **compiled** to φ-programs via f and
- φ-programs can be **decompiled** to ψ-programs via g

which situation should be familiar from modern programming languages (up to the "little" difference that real programming languages don't restrict focus to unary functions on $\mathbb{N}$). Notice, however, that compilation and decompilation are not mutually inverse on the level of programs, i.e. in general

$$f \circ g \neq id_{\mathbb{N}} \neq g \circ f$$

[6] "universal" here means "Turing universal", i.e. the class of partial functions from $\mathbb{N}$ to $\mathbb{N}$ which can be expressed in the programming language under consideration coincides with $\mathcal{P}$

though f and g are mutually inverse w.r.t. the extensional equality induced by φ and ψ, respectively, i.e.

$$\varphi \circ f \circ g = \varphi \qquad \text{and} \qquad \psi \circ g \circ f = \psi \ .$$

It is a simple exercise(!) to show that whenever $f, g : \mathbb{N} \to \mathbb{N}$ are total recursive functions with $\varphi \circ f \circ g = \varphi$ the function $\psi := \varphi \circ f : \mathbb{N} \to \mathcal{P}$ is an admissible numbering of $\mathcal{P}$. In such a situation we may define a variant T^ψ of Kleene's T-predicate as follows

$$T^\psi(e, n, k) \equiv T(f(e), n, k)$$

for which it holds that

$$\psi_e(n) \simeq \varphi_{f(e)}(n) \simeq U(\mu k.T(f(e), n, k)) \simeq U(\mu k.T^\psi(e, n, k))$$

as expected.
Summarising one may say that up to recursive equivalence there is just one "reasonable" coding of partial recursive functions where "reasonable" is captured by the precise technical meaning of "admissible".

5 Recursive and Semidecidable Sets

In this section we will introduce the key concepts of *decidable* and *semidecidable* sets of natural numbers.

Definition 5.1 (decidable and semidecidable sets)
A subset A of $\mathbb{N}$ is called decidable *iff there exists a total recursive function $p : \mathbb{N} \to \mathbb{N}$ such that*

$$n \in A \qquad \textit{iff} \qquad p(n) = 0$$

for all $n \in \mathbb{N}$.
A subset A of $\mathbb{N}$ is called semidecidable *iff there exists a partial recursive function f with*

$$n \in A \qquad \textit{iff} \qquad f(n)\!\downarrow$$

for all $n \in \mathbb{N}$. We say that e is a Gödel number for A *iff $A = \mathsf{dom}(\varphi_e)$, i.e. $n \in A$ iff $\exists k.T(e, n, k)$.*
We write W_e as an abbreviation for $\{n \in \mathbb{N} \mid \exists k.T(e, n, k)\}$. $\Diamond$

It is a straightforward exercise(!) to show that decidable sets are closed under finite unions, finite intersection and complementation. Using Kleene's T-predicate one also easily shows that semidecidable sets are closed under finite unions and finite intersections. As we shall see a bit later a set A is decidable if and only if A and its complement $\complement A = \mathbb{N}\backslash A$ are both semidecidable.
Semidecidable sets are usually called *recursively enumerable* (*r.e.*) as they can be characterised as follows.

Lemma 5.1 *A set $A \subseteq \mathbb{N}$ is semidecidable iff A is empty or $A = f[\mathbb{N}]$ for some total recursive f.*

Proof: Suppose A is semidecidable and non-empty. Let e be a natural number with $W_e = A$ and $n_0 \in A$. Then for the total recursive function f with

$$f(n) = \begin{cases} \pi_0(n) & \text{if } T(e, \pi_0(n), \pi_1(n)) \\ n_0 & \text{otherwise} \end{cases}$$

it obviously holds that $A = f[\mathbb{N}]$.
On the other hand the empty set is obviously semidecidable and if $A = f[\mathbb{N}]$ for some total recursive function f then a semidecision procedure for A is given by $p(x) = \mu k.eq(x, f(k))$. □

Notice that the function f constructed in the proof of the previous theorem is actually primitive recursive (as T is primitive recursive). Thus, we have actually proved something slightly stronger, namely that a set A of natural numbers is r.e. iff it is empty or the image of a primitive recursive function. Another characterisation of recursive enumerability is given by the following lemma whose proof is left to the reader as a simple exercise(!).

Lemma 5.2 *A set $A \subseteq \mathbb{N}$ is r.e. iff A is the image of some φ_e, i.e. iff*

$$x \in A \qquad \textit{iff} \qquad \exists n, k \in \mathbb{N}.\, T(e, n, k) \wedge x = U(k)$$

for all $x \in \mathbb{N}$.

Proof: Exercise(!) left to the reader. □

Next we give a characterisation of decidability in terms of semidecidability.

Lemma 5.3 *A set A of natural numbers is decidable iff A and $\complement A$ are semidecidable.*

Proof: The implication from left to right is trivial as one always may decide to diverge (i.e. to *not* terminate).
For the reverse direction assume that $A = W_{e_1}$ and $\complement A = W_{e_2}$. Then a decision procedure for A is given by

$$p(n) = f(n, \mu k.T(e_1, n, k) \vee T(e_2, n, k))$$

with

$$f(n,k) = \begin{cases} 0 & \text{if } T(e_1, n, k) \\ 1 & \text{otherwise} \end{cases}$$

The idea behind p is that for given n one checks for all numbers k whether they code a terminating computation sequence for $\{e_1\}(n)$ or $\{e_2\}(n)$ and in the first case one outputs 0 and in the second case one outputs 1. □

Probably the most important fact of recursion theory is that there are semidecidable sets which are not decidable. The paradigmatic such example is the so-called *halting problem.*

Theorem 5.1 (Undecidability of the Halting Problem) *The sets*

$$H = \{\langle n, m\rangle \mid \exists k{\in}\mathbb{N}.T(n, m, k)\} \quad \textit{and} \quad K = \{n \in \mathbb{N} \mid \exists k{\in}\mathbb{N}.T(n, n, k)\}$$

are both semidecidable but none of them is decidable.

Proof: Evidently the sets H and K are semidecidable via the partial recursive functions

$$p_H(n) = \mu k.T(\pi_0(n), \pi_1(n), k) \qquad \text{and} \qquad p_K(n) = \mu k.T(n, n, k)$$

respectively.
Obviously, if H were decidable then so were K as $n \in K$ iff $\langle n, n\rangle \in H$. Thus, it remains to show that K is not decidable. By Lemma 5.3 this is equivalent to showing that $\complement K$ is not r.e. Assume on the contrary that $\complement K = W_e$ for some $e \in \mathbb{N}$, i.e.

$$\{n\}(n)\uparrow \quad \text{iff} \quad \{e\}(n)\downarrow$$

and, therefore, in particular

$$\{e\}(e)\uparrow \quad \text{iff} \quad \{e\}(e)\downarrow$$

which clearly is impossible. □

We now will show that in some sense K is the "most difficult" r.e. set. The following notion allows one to compare the difficulty of the decision problem for arbitrary subsets of $\mathbb{N}$.

Definition 5.2 (many-one reducibility) *Let $A, B \subseteq \mathbb{N}$. We say that A* is many-one reducible to *B ($A \leq_{\mathsf{m}} B$) iff there exists a total recursive function f with $A = f^{-1}[B]$.* ◊

Obviously, if $A \leq_{\mathsf{m}} B$ via a total recursive function f then a decision procedure p for B gives rise to the decision procedure $p \circ f$ for A. In this sense B is at least as difficult to decide as A. The terminology "many-one" reflects the fact that f need not be one-to-one, i.e. that many different arguments may be mapped to the same value by f. One easily shows (exercise!) that $\leq_{\mathsf{m}}$ is a preorder on $\mathcal{P}(\mathbb{N})$, i.e. that $\leq_{\mathsf{m}}$ is reflexive and transitive. Equivalence classes w.r.t. the symmetrisation of $\leq_{\mathsf{m}}$ are called *many-one degrees* or simply m-*degrees*. Degree theory is a vast field within recursion theory (*cf.* [Cut, Rog]). Particular attention is given to r.e. degrees, i.e. equivalence classes of r.e. sets w.r.t. $\leq_{\mathsf{m}}$ (or some other notion of reducibility).

Lemma 5.4 *Every r.e. set is many-one reducible to K.*

Proof: Suppose $A = \mathsf{dom}(f)$ for some partial recursive f. Then the function

$$g(\langle n, m\rangle) \simeq f(n)$$

is partial recursive, too. Let e be a Gödel number for g. Then by the *snm*-theorem we have

$$f(n) \simeq g(\langle n, m\rangle) \simeq \{e\}(\langle n, m\rangle) \simeq \{s(\langle e, n\rangle)\}(m)$$

for all $n, m \in \mathbb{N}$ from which it follows that

$$f(n)\downarrow \qquad \text{if and only if} \qquad s(\langle e, n\rangle) \in K$$

as $f(n)\downarrow$ iff $\{s(\langle e, n\rangle)\}(s(\langle e, n\rangle))\downarrow$. Thus, we have

$$n \in A \qquad \text{if and only if} \qquad s(\langle e, n\rangle) \in K$$

from which it follows that $A \leq_{\mathsf{m}} K$ via the total recursive function mapping n to $s(\langle e, n\rangle)$. □

6 Why Partiality is Intrinsic

There arises the question whether it wouldn't be reasonable to restrict one's attention to the collection of *total* recursive functions as from a practical point of view one would rather like to avoid non-termination. The following theorem tells us that this is impossible. The proof idea is a minor variation of Cantor's 2nd diagonal argument showing that there cannot exist a surjection from a set X to the set X^X of all functions from X to X.

Theorem 6.1 *There does not exist a total recursive function $v : \mathbb{N}^2 \to \mathbb{N}$ such that for every unary total recursive function f there exists an $e \in \mathbb{N}$ with $v(e, n) = f(n)$ for all $n \in \mathbb{N}$.*

Proof: Suppose there exists a total recursive v such that for every total recursive f there is an e with $f = v(e, \_)$. Consider the total recursive function $f(n) = v(n, n) + 1$. Due to our assumption on v there exists an $e \in \mathbb{N}$ with $v(e, n) = f(n)$ for all $n \in \mathbb{N}$. Thus, in particular, we have

$$v(e, e) = f(e) = v(e, e) + 1$$

which certainly is a contradiction. □

This theorem tells us that there cannot exist a programming language allowing one to implement precisely the total recursive functions because it tells us that for every surjective function ψ from $\mathbb{N}$ to the set $\mathcal{R}$ of total recursive functions the associated interpreter $u^{(\psi)} : \mathbb{N}^2 \to \mathbb{N}$ itself cannot be computable.
Of course, it is possible to consider non-trivial programming languages where all programs terminate. For example one may consider a programming language where recursion is restricted to primitive recursion. Then, however, the interpreter for this class of total recursive functions falls out of this class as shown by the next theorem.

Theorem 6.2 *Let $\mathcal{C}$ be a class of unary total recursive functions containing $id_{\mathbb{N}}$ and succ, closed under composition and satisfying the requirement that $\langle f, g\rangle \in \mathcal{C}$ whenever $f, g \in \mathcal{C}$ (where $\langle f, g\rangle(n) := \langle f(n), g(n)\rangle$).*
Then for every enumeration $\psi : \mathbb{N} \overset{surj.}{\to} \mathcal{C}$ of $\mathcal{C}$ the interpreter function $u_\psi : \mathbb{N} \to \mathbb{N}$ with $u_\psi(\langle n, m\rangle) := \psi_n(m)$ is not contained in $\mathcal{C}$.

Proof: Suppose $u_\psi \in \mathcal{C}$. Then due to the assumed closure properties of $\mathcal{C}$ the function $f(n) := \psi_n(n) + 1$ is also in $\mathcal{C}$ (as $f = succ \circ u_\psi \circ \delta_{\mathbb{N}}$ where $\delta_{\mathbb{N}}(n) := \langle n, n \rangle$). As ψ was assumed as surjective there exists an $e \in \mathbb{N}$ with $f = \psi_e$ and we have

$$\psi_e(e) = f(e) = \psi_e(e) + 1$$

which obviously is a contradiction. □

As a consequence we get the following corollary.

Corollary 6.1 *An interpreter for primitive recursive functions cannot itself be primitive recursive.*

Proof: Immediate from Theorem 6.2 instantiating $\mathcal{C}$ by the set of unary primitive recursive functions. □

There is a lot of examples of classes of recursive functions satisfying at least the closure properties required for $\mathcal{C}$ in Theorem 6.2 as for example the polynomial time computable functions or classes of recursive functions whose termination can be proved in a fixed formal system T.[7] In Chapter III on Proof Theory we exhibit a programming language capturing the recursive functions provably total in Peano Arithmetic.
Though we have seen that there cannot exist a programming language capturing the class of total recursive functions there still remains the question whether every partial recursive function can be extended to a total recursive one. But the answer to this question is negative, too.

Theorem 6.3 *There exists a partial recursive function which cannot be extended to a total recursive function.*

Proof: Consider the partial recursive function $f(n) \simeq u(n, n) + 1$. Suppose that there is a total recursive function φ_e with $\varphi_e(n) = f(n)$ whenever $f(n)\downarrow$. As by assumption $\varphi_e(e)$ is defined it follows that $f(e) = u(e, e) + 1$ is defined and, therefore, that $\varphi_e(e) = f(e)$. Thus, we have

$$\varphi_e(e) = f(e) = \varphi_e(e) + 1$$

which obviously is impossible. □

[7] This formulation is somewhat inaccurate. More precisely, what we mean is that there is an algorithm which implements the function and whose termination can be proved in the formal system T.

7 Some Cheap Negative Results

Later we will show a nontrivial Theorem due to Rice and Shapiro characterising *semidecidable extensional properties of* $\mathcal{P}$, i.e. those $\mathcal{F} \subseteq \mathcal{P}$ for which $E_{\mathcal{F}}$ is semidecidable. As we shall see in the next section the Rice-Shapiro-Theorem allows us to give a negative answer to most questions concerning decidability or semidecidability of extensional predicates on $\mathcal{P}$ in a most simple way. Before embarking on the somewhat non-trivial proof of the Rice-Shapiro-Theorem in this section we will give a few direct proofs of negative results of the kind mentioned above.

Theorem 7.1 (Theorem of Rice)
Every decidable extensional predicate on $\mathcal{P}$ *is trivial, i.e. for every* $\mathcal{F} \subseteq \mathcal{P}$ *with* $E_{\mathcal{F}} = \{e \in \mathbb{N} \mid \varphi_e \in \mathcal{F}\}$ *decidable either* $\mathcal{F} = \mathcal{P}$ *or* $\mathcal{F} = \emptyset$.

Proof: Suppose that $\mathcal{F}$ is a nontrivial subset of $\mathcal{P}$ with $E_{\mathcal{F}}$ decidable. Then there exists $f \in \mathcal{P}$ with

$$f \in \mathcal{F} \quad \text{iff} \quad \emptyset \notin \mathcal{F}$$

where $\emptyset$ stands for the empty function in $\mathcal{P}$. Let $g : \mathbb{N}^2 \rightharpoonup \mathbb{N}$ be the partial recursive function with

$$g(e, n) = m \qquad \text{iff} \qquad e \in K \text{ and } m = f(n).$$

Then by the parameter theorem there exists a total recursive function h with $\varphi_{h(e)}(n) \simeq g(e, n)$ for all $e, n \in \mathbb{N}$. Obviously, we have that $\varphi_{h(e)} = f$ if $e \in K$ and $\varphi_{h(e)} = \emptyset$ otherwise. Assuming w.l.o.g. that $\emptyset \in \mathcal{F}$ we get

$$h(e) \in E_{\mathcal{F}} \quad \text{iff} \quad e \notin K$$

for all e. Thus, the complement of K were r.e. which is known to be impossible. □

Rice's Theorem gives a devastating answer to all attempts of checking in a purely mechanical way whether programs satisfy a non-trivial specification. Actually, we can even show that the set of Gödel numbers of a given total recursive function is not r.e. and, therefore, not finite.

Lemma 7.1 *If* f *is a total recursive function then* $E_f = \{n \in \mathbb{N} \mid \varphi_n = f\}$ *is not r.e.*

Proof: Let $g : \mathbb{N}^2 \rightharpoonup \mathbb{N}$ be the partial recursive function with

$$g(e,n) = m \quad \text{iff} \quad \neg T(e,e,n) \text{ and } f(n) = m$$

By the parameter theorem there exists a total recursive function h with $\varphi_{h(e)} = f$ iff $e \notin K$, i.e.

$$h(e) \in E_f \quad \text{iff} \quad e \notin K$$

from which it follows that $\complement K$ is r.e. which is impossible. □

From the Rice-Shapiro Theorem in the next section it will also follow that E_f is not r.e. even if f is only partial recursive.

8 The Rice-Shapiro Theorem

In this section we will prove the Rice-Shapiro Theorem characterising the extensionally r.e. subsets of $\mathcal{P}$ which are defined as follows.

Definition 8.1 (extensionally r.e.) *A subset $\mathcal{F}$ of $\mathcal{P}$ is called* extensionally r.e. *iff the set $E_{\mathcal{F}} := \{n \in \mathbb{N} \mid \varphi_n \in \mathcal{F}\}$ is r.e.* ◊

We first prove the crucial lemma which is of interest in its own right and from which later on one rather easily obtains the Rice-Shapiro Theorem.

Lemma 8.1 *For every extensionally r.e. set $\mathcal{F}$ it holds that*

(1) *if f_0 is a finite partial function in $\mathcal{F}$ and f is a partial recursive function with $f_0 \subseteq f$ then $f \in \mathcal{F}$, too, and*

(2) *for every $f \in \mathcal{F}$ there exists a finite partial function $f_0 \in \mathcal{F}$ with $f_0 \subseteq f$.*

Proof: ad (1) : Suppose that f_0 is a finite partial function in $\mathcal{F}$ and f is a partial recursive function $f \supseteq f_0$ with $f \notin \mathcal{F}$. Obviously, the partial function

$$g(\langle n,m\rangle) \simeq \begin{cases} f(m) & \text{if } m \in \mathsf{dom}(f_0) \text{ or } n \in K \\ \uparrow & \text{otherwise} \end{cases}$$

is partial recursive. Let $\varphi_e = g$. Then by the *snm*-theorem we have

$$\varphi_{s(\langle e,n\rangle)}(m) \simeq \varphi_e(\langle n,m\rangle)$$

for all $n, m \in \mathbb{N}$. The function $h(n) = s(\langle e,n\rangle)$ is total recursive and we have

(i) $n \in K \Rightarrow \varphi_{h(n)} = f \Rightarrow h(n) \notin E_{\mathcal{F}}$ and

(ii) $n \notin K \Rightarrow \varphi_{h(n)} = f_0 \Rightarrow h(n) \in E_{\mathcal{F}}$

from which it follows that

$$n \notin K \Leftrightarrow h(n) \in E_{\mathcal{F}}$$

for all n. Thus, as $E_{\mathcal{F}}$ is r.e. and h is total recursive it follows that $\complement K$ is r.e. which is known to be impossible.
ad (2) : Suppose $f \in \mathcal{F}$ such that $f_0 \notin \mathcal{F}$ for every finite partial function $f_0 \subseteq f$. Obviously, the partial function

$$g(\langle n, m\rangle) \simeq \begin{cases} f(m) & \text{if } m \in \mathsf{dom}(f_0) \text{ or } \forall k{\leq}m.\neg T(n,n,k) \\ \uparrow & \text{otherwise} \end{cases}$$

is partial recursive, i.e. $g = \varphi_e$ for some $e \in \mathbb{N}$. By the *snm*-theorem we have

$$\varphi_{s(\langle e,n\rangle)}(m) \simeq \varphi_e(\langle n, m\rangle)$$

for all $n, m \in \mathbb{N}$. Let h be the total recursive function with $h(n) = s(\langle e, n\rangle)$. Obviously, for all n we have $\varphi_{h(n)} \subseteq f$ and $n \in K$ iff $\varphi_{h(n)}$ is finite. Thus, we have

$$n \notin K \Leftrightarrow h(n) \in E_{\mathcal{F}}$$

for all n. Thus, as $E_{\mathcal{F}}$ is r.e. and h is total recursive it follows that $\complement K$ is r.e. which is known to be impossible. □

It is a simple consequence of this lemma that

Corollary 8.1 *Every extensionally r.e. subset $\mathcal{F}$ of $\mathcal{P}$ is upward closed, i.e., if $f \in \mathcal{F}$ and $f \subseteq g \in \mathcal{P}$ then $g \in \mathcal{F}$.*

Proof: Suppose $f \in \mathcal{F}$ and $g \supseteq f$ is partial recursive. By Lemma 8.1(2) there exists a finite partial function $f_0 \subseteq f$ with $f_0 \in \mathcal{F}$. Thus, by Lemma 8.1(1) it follows that $g \in \mathcal{F}$ as $f_0 \subseteq f \subseteq g$. □

Already this corollary allows us to show very easily that equality and inclusion[8] between partial recursive functions is not semidecidable in terms of Gödel numbers.

[8]We write $f \sqsubseteq g$ for $f \subseteq g$ from now on!

Lemma 8.2 *The sets*

$$E_= = \{\langle n, m\rangle \mid \varphi_n = \varphi_m\} \quad \textit{and} \quad E_\sqsubseteq = \{\langle n, m\rangle \mid \varphi_n \sqsubseteq \varphi_m\}$$

are neither r.e. nor co-r.e.

Proof: As $\langle n, m\rangle \in E_=$ iff $\langle n, m\rangle$ and $\langle m, n\rangle$ are both in $E_\sqsubseteq$ it suffices to show that $E_=$ is neither r.e. nor co-r.e. If $E_=$ were r.e. then $\{\emptyset\} \subseteq \mathcal{P}$ were extensionally r.e. which by Corollary 8.1 entails that $\mathcal{P} = \{\emptyset\}$ which is blatantly wrong as not all partial recursive functions are empty! On the other hand if $E_=$ were co-r.e., i.e. the complement of $E_=$ were r.e., then the set $\mathcal{F} = \{f \in \mathcal{P} \mid f \neq \lambda n.0\}$ were extensionally r.e. and, therefore, as $\emptyset \in \mathcal{F}$ according to Corollary 8.1 we would get $\lambda n.0 \in \mathcal{F}$ in contradiction to the definition of $\mathcal{F}$. □

The following further negative results can be obtained from Lemma 8.1.

Lemma 8.3

(1) *For every partial recursive function f the set $E_f = \{n \in \mathbb{N} \mid \varphi_n = f\}$ is not r.e. and it is not co-r.e. unless f is the empty function.*

(2) *Neither the set of total recursive functions nor its complement are extensionally r.e.*

(3) *Neither the set of partial recursive functions with finite domain of definition nor its complement are extensionally r.e.*

Proof: ad (1) Suppose E_f were r.e., i.e. $\{f\}$ were extensionally r.e. If f has an infinite domain of definition then by Lemma 8.1 the set $\{f\}$ would also contain some finite function f_0 below but different from f. If f were finite then by Lemma 8.1 the set $\{f\}$ would contain some infinite extension of f. Suppose E_f were co-r.e., i.e. $\mathcal{P}\backslash\{f\}$ were extensionally r.e. If f is different from the empty function then $\emptyset \in \mathcal{P}\backslash\{f\}$ and, therefore, $f \in \mathcal{P}\backslash\{f\}$ by Lemma 8.1. However, the set $E_\emptyset$ is co-r.e. as $\varphi_e \neq \emptyset$ iff $\exists n, k.T(e, n, k)$.
ad (2) If the set of total recursive functions were extensionally r.e. then it also would contain some finite function due to Lemma 8.1. If the complement of the set of total recursive functions were extensionally r.e., i.e. the set of properly partial recursive functions were extensionally r.e., then by Corollary 8.1

every total recursive function were properly partial as the empty function is properly partial.
ad (3) analogous to the proof of (2) and left to the reader. □

Now we will prove the Rice-Shapiro Theorem characterising the extensional r.e. sets as those subsets $\mathcal{F}$ of $\mathcal{P}$ such that there exists an r.e. set $\mathcal{F}_0$ of finite functions with

$$f \in \mathcal{F} \qquad \text{iff} \qquad \exists f_0 \in \mathcal{F}_0.\ f_0 \sqsubseteq f$$

for all partial recursive f. However, for this purpose we first have to introduce an appropriate Gödel numbering for the finite functions.

Theorem 8.1 *There exists a surjective function*

$$\varphi^{\mathsf{fin}} : \mathbb{N} \stackrel{surj.}{\to} \{f \in \mathcal{P} \mid \mathsf{dom}(f) \text{ finite}\}$$

such that

(1) *the function* $u^{\mathsf{fin}}(e,n) \simeq \varphi^{\mathsf{fin}}_e(n)$ *is partial recursive and*

(2) *the predicate* $\varphi^{\mathsf{fin}}_n \sqsubseteq \varphi_m$ *is semidecidable.*

Proof: First recall that $\mathcal{P}_{\mathsf{fin}}(\mathbb{N})$ can be goedelized by $\varepsilon_n = a$ iff $n = \sum_{i \in a} 2^i$. For this coding it holds that $m \in \varepsilon_n$ implies $m \leq n$. Accordingly, the predicate

$$sv(n) \equiv \forall m_1, m_2 \in \varepsilon_n.\ \pi_0(m_1) = \pi_0(m_2) \Rightarrow \pi_1(m_1) = \pi_1(m_2)$$

is primitive recursive. Obviously, the predicate sv (standing for "single-valued") holds for a natural number n iff $\varepsilon_n = \{\langle m, f(m)\rangle \mid m \in \mathsf{dom}(f)\}$ for some finite function f. Let f^{fin} be some (primitive) recursive function enumerating $\{n \in \mathbb{N} \mid sv(n)\}$. Then we can define φ^{fin} as

$$\varphi^{\mathsf{fin}}_n(m) = k \quad \text{iff} \quad \langle m,k\rangle \in \varepsilon_{f^{\mathsf{fin}}(n)}$$

which satisfies requirement (1) as

$$\varphi^{\mathsf{fin}}_n(m) \simeq \mu k.\langle m,k\rangle \in \varepsilon_{f^{\mathsf{fin}}(n)}$$

for all $n, m \in \mathbb{N}$. Requirement (2) holds as we have

$$\varphi^{\mathsf{fin}}_n \sqsubseteq \varphi_e \quad \text{iff} \quad \forall m \in \varepsilon_{f^{\mathsf{fin}}(n)}.\ \varphi_e(\pi_0(m)) = \pi_1(m)$$

which is obviously semidecidable as the quantification over all $m \in \varepsilon_{f^{\mathsf{fin}}(n)}$ may be replaced equivalently by the quantification over all $m \leq f^{\mathsf{fin}}(n)$ with $m \in \varepsilon_{f^{\mathsf{fin}}(n)}$. □

Now we are ready to prove the Rice-Shapiro Theorem in its full glory.

Theorem 8.2 (Rice-Shapiro Theorem) *A set $\mathcal{F}$ of partial recursive functions is extensionally r.e. iff*

$$\mathcal{F} = \{f \in \mathcal{P} \mid \exists n \in A.\ \varphi_n^{\mathsf{fin}} \sqsubseteq f\}$$

for some r.e. set A.

Proof: Obviously, the condition is sufficient as

$$e \in E_{\mathcal{F}} \Longleftrightarrow \varphi_e \in \mathcal{F} \Longleftrightarrow \exists n \in A.\ \varphi_n^{\mathsf{fin}} \sqsubseteq \varphi_e$$

and, therefore, $E_{\mathcal{F}}$ is r.e. by Theorem 8.1(2).
On the other hand if $\mathcal{F}$ is extensionally r.e. then by Lemma 8.1 we know that $f \in \mathcal{F}$ iff $f_0 \sqsubseteq f$ for some finite $f_0 \in \mathcal{F}$. Thus, we have

$$\mathcal{F} = \{f \in \mathcal{P} \mid \exists n \in A.\ \varphi_n^{\mathsf{fin}} \sqsubseteq f\}$$

for $A := \{n \in \mathbb{N} \mid \varphi_n^{\mathsf{fin}} \in \mathcal{F}\}$. It remains to show that A is r.e. Let $\varphi_e(\langle n, m\rangle) \simeq \varphi_n^{\mathsf{fin}}(m)$. Then by the *snm*-theorem we have $\varphi_{s(\langle e,n\rangle)} = \varphi_n^{\mathsf{fin}}$ for all n and, therefore,

$$A = \{n \in \mathbb{N} \mid \varphi_{s(\langle e,n\rangle)} \in \mathcal{F}\} = \{n \in \mathbb{N} \mid s(\langle e, n\rangle) \in E_{\mathcal{F}}\}$$

which clearly is r.e. as $E_{\mathcal{F}}$ is r.e. by assumption. □

The Rice-Shapiro theorem tells us that for an extensionally r.e. set $\mathcal{F}$ in order to check whether $f \in \mathcal{F}$ it suffices to check whether f extends some finite function $f_0 \in \mathcal{F}_0$ where $\mathcal{F}_0$ is an r.e. set of finite functions in $\mathcal{F}$. Moreover, for $\mathcal{F}_0$ one may take the set of all finite functions contained in $\mathcal{F}$. In order to check whether f extends a finite function f_0 one just needs extensional information about f.
The Rice-Shapiro theorem provides a very nice and very important connection between recursion theory and topology which we will explain next. The

set $[\mathbb{N} \rightharpoonup \mathbb{N}]$ of partial functions from $\mathbb{N}$ to $\mathbb{N}$ can be endowed with a topology of "finite information" whose basic[9] open sets are those of the form $\uparrow f_0 = \{f : \mathbb{N} \rightharpoonup \mathbb{N} \mid f_0 \sqsubseteq f\}$ for some finite function $f_0 : \mathbb{N} \rightharpoonup \mathbb{N}$. The induced subspace topology on $\mathcal{P} \subseteq [\mathbb{N} \rightharpoonup \mathbb{N}]$ we also call topology of "finite information". The Rice-Shapiro theorem tells us that a set $\mathcal{F}$ is extensionally r.e. iff it is open in $\mathcal{P}$ w.r.t. the finite information topology and the basic neighbourhoods contained in it are recursively enumerable.
This *prima facie* unexpected connection between computability and topology lies at the heart of D. Scott's Domain Theory. In the next section we will show that computable (= effective) functionals and operators are also continuous w.r.t. the finite information topology and can be characterised as continuous functionals satisfying a certain effectiveness requirement (notice the analogy to the above reading of the Rice-Shapiro theorem!).
In a sense the usual Euclidean topology on the real numbers is also a "finite information topology" as for a Cauchy sequence $(x_n)_{n\in\mathbb{N}}$ with a given rate of convergence in order to guarantee that its limit is contained in some open set it suffices to inspect a finite initial segment of the sequence!

9 Effective Operations, Effective Operators and the Myhill-Shepherdson Theorem

Due to admissible Gödel numberings of $\mathcal{P}$ one may reduce computability of second order objects like functionals and operators to first order as in the following definition.

Definition 9.1 (Effective Operations and Operators)
An effective operation *is a partial functional $\phi : \mathcal{P} \rightharpoonup \mathbb{N}$ such that for some natural number e it holds that*

$$\phi(\{n\}) \simeq \{e\}(n)$$

for all $n \in \mathbb{N}$ in which case we say that e realizes *ϕ and for which we write $e \Vdash \phi$.*
An effective operator *is a function $\Phi : \mathcal{P} \to \mathcal{P}$ such that for some natural number e it holds that*

$$\Phi(\{n\}) = \{\{e\}(n)\}$$

[9] notice that in the case under consideration the basic opens are closed under finite intersections!

for all $n \in \mathbb{N}$ *in which case we say that* e realizes Φ *and for which we write* $e \Vdash \Phi$. $\Diamond$

Notice that not every natural number e realizes an effective operation. For example if $\{e\} = id_{\mathbb{N}}$ then e doesn't realize any effective operation because $id_{\mathbb{N}}$ doesn't respect extensional equality. Similarly not every natural number e realizes an effective operator as for this purpose $\{e\}$ has to be total.
Using the Rice-Shapiro theorem we get that effective operations are continuous w.r.t. the "topology of finite information".

Lemma 9.1 (Myhill and Shepherdson)
For every effective operation ϕ *and natural number* n *the set* $\phi^{-1}(n)$ *is extensionally r.e. and, therefore, open in the topology of finite information which more explicitly means that*

(1) *from* $\phi(f) = n$ *and* $f \sqsubseteq g$ *it follows that* $\phi(g) = n$*, too, and*

(2) *if* $\phi(f) = n$ *then there is a finite function* $f_0 \sqsubseteq f$ *with* $\phi(f_0) = n$*.*

Proof: Suppose that e realizes ϕ and n is a natural number. Then

$$m \in E_{\phi^{-1}(n)} \quad \text{iff} \quad \phi(\{m\}) = n \quad \text{iff} \quad \{e\}(m) = n$$

from which it follows that $\phi^{-1}(n)$ is extensionally r.e. Claims (1) and (2) are immediate from the Rice-Shapiro theorem. $\square$

The above lemma says that if $\phi(f) = n$ one just needs a finite amount of extensional information about f, i.e. the values of f at finitely many arguments, for computing the value $\phi(f)$. Moreover, this value $\phi(f)$ depends only on *positive* information about the argument f because if $\phi(f)$ terminates and $f \sqsubseteq g$ then $\phi(f) = \phi(g)$.
The conditions (1) and (2) of Lemma 9.1 can be squeezed into the following single condition

$$\forall f \in \mathcal{P}.\forall n \in \mathbb{N}.\ \phi(f) = n \Rightarrow \exists f_0 \sqsubseteq_{\mathsf{fin}} f.\forall g \in \mathcal{P}.\ f_0 \sqsubseteq g \rightarrow \phi(f_0) = \phi(g)$$

which strongly resembles the "ε-δ-characterisation of continuity" (with n corresponding to ε and f_0 to δ).

This continuity result can be extended to effective operators.

Lemma 9.2 (Myhill and Shepherdson)
For every effective operator Φ and finite function f_0 the set $\Phi^{-1}(\uparrow f_0)$ is extensionally r.e.

Proof: Suppose $e \Vdash \Phi$. We have

$$\{n\} \in \Phi^{-1}(\uparrow f_0) \quad \text{iff} \quad f_0 \sqsubseteq \Phi(\{n\}) \quad \text{iff} \quad f_0 \sqsubseteq \varphi_{\{e\}(n)}$$

from which it follows by Theorem 8.1(2) that the set $\Phi^{-1}(\uparrow f_0)$ is extensionally r.e. □

As an immediate corollary we get that

Corollary 9.1 *Every effective operator Φ is monotonic, i.e., if $f \sqsubseteq g$ then $\Phi(f) \sqsubseteq \Phi(g)$.*

Proof: Straightforward exercise(!) when using Lemma 9.2. □

It follows from the previous two lemmas that effective operations and operators are determined by their behaviour on finite arguments. This motivates the following definition.

Definition 9.2 *For an effective operation ϕ its* graph *is defined as*

$$\mathsf{graph}(\phi) = \{\langle n, m\rangle \mid \phi(\{n\}^{\mathsf{fin}}) = m\}$$

and for an effective operator Φ its graph *is defined as*

$$\mathsf{graph}(\Phi) = \{\langle n, \langle m, k\rangle\rangle \mid \Phi(\{n\}^{\mathsf{fin}})(m) = k\}$$

where $\{n\}^{\mathsf{fin}}$ stands for φ_n^{fin}. ◊

Now we are ready to formulate and prove the Myhill-Shepherson Theorem(s).

Theorem 9.1 (Myhill-Shepherdson for Effective Operations)
For every effective operation ϕ its graph $\mathsf{graph}(\phi)$ is r.e. Moreover, for $\langle n_1, m_1\rangle, \langle n_2, m_2\rangle \in \mathsf{graph}(\phi)$ it holds that $m_1 = m_2$ whenever $\{n_1\}^{\mathsf{fin}}$ and $\{n_2\}^{\mathsf{fin}}$ have a common extension (for which we write $\{n_1\}^{\mathsf{fin}} \uparrow \{n_2\}^{\mathsf{fin}}$).
Suppose that G is a r.e. subset of $\mathbb{N}$ which is consistent *in the sense that for all $\langle n_1, m_1\rangle, \langle n_2, m_2\rangle \in G$, $\{n_1\}^{\mathsf{fin}} \uparrow \{n_2\}^{\mathsf{fin}}$ implies $m_1 = m_2$. Then ϕ_G with*

$$\phi_G(f) = m \quad \textit{iff} \quad \exists n.\ \{n\}^{\mathsf{fin}} \sqsubseteq f \wedge \langle n, m\rangle \in G$$

is an effective operation.

Proof: Suppose e realizes the effective operation ϕ. Let h be a total recursive function with $\{h(n)\} = \{n\}^{\mathsf{fin}}$ for all $n \in \mathbb{N}$. Then we have

$$\phi(\{n\}^{\mathsf{fin}}) = m \quad \text{iff} \quad \phi(\{h(n)\}) = m \quad \text{iff} \quad \{e\}(h(n)) = m$$

from which it follows that $\mathsf{graph}(\phi)$ is r.e. For showing that $\mathsf{graph}(\phi)$ is consistent suppose that $\phi(\{n_1\}^{\mathsf{fin}}) = m_1$, $\phi(\{n_2\}^{\mathsf{fin}}) = m_2$ and $\{n_1\}^{\mathsf{fin}}, \{n_2\}^{\mathsf{fin}} \sqsubseteq f$. Then by Corollary 9.1 we have $\phi(f) = m_1$ and $\phi(f) = m_2$ from which it follows that $m_1 = m_2$ as desired.
Suppose that G is a consistent r.e. set. The set

$$A_G = \{\langle n, m\rangle \mid \exists k.\ \{k\}^{\mathsf{fin}} \sqsubseteq \{n\} \wedge \langle k, m\rangle \in G\}$$

is r.e. as G is r.e. by assumption and the predicate $\{k\}^{\mathsf{fin}} \sqsubseteq \{n\}$ is r.e. by Theorem 8.1(2). Obviously $\phi_G(\{n\}) = m$ iff $\langle n, m\rangle \in A_G$. Then ϕ_G is realized by e with

$$\{e\}(n) \simeq \pi_0(\mu k.T(n_0, \langle n, \pi_0(k)\rangle, \pi_1(k)))$$

where $W_{n_0} = A_G$. □

A realizer e for an effective operation ϕ necessarily operates on the code, i.e. Gödel number, of its argument. The surprising fact expressed by the Myhill-Shepherdson theorem is that ϕ nevertheless does not really make use of this intensional information about the argument as the value of ϕ at some argument f is determined by some finite extensional information about f.
Notice, moreover, that a Gödelization of effective operations may be obtained using the fact that there is a total recursive function *shave* such that for all n it holds that $W_{shave(n)}$ is consistent and equal to W_n provided W_n itself is consistent.

All these considerations and the Myhill-Shepherdson theorem in particular can be extended to effective operators.

Theorem 9.2 (Myhill-Shepherdson for Effective Operators)
For every effective operator Φ *its graph* $\mathsf{graph}(\Phi)$ *is r.e. Moreover, for all* $\langle n_1, \langle k, m_1\rangle\rangle, \langle n_2, \langle k, m_2\rangle\rangle \in \mathsf{graph}(\Phi)$ *it holds that* $m_1 = m_2$ *whenever* $\{n_1\}^{\mathsf{fin}} \uparrow \{n_2\}^{\mathsf{fin}}$.
Moreover, if G *is a r.e. subset of* $\mathbb{N}$ *which is* consistent *in the sense that for all* $\langle n_1, \langle k, m_1\rangle\rangle, \langle n_2, \langle k, m_2\rangle\rangle \in G$, $\{n_1\}^{\mathsf{fin}} \uparrow \{n_2\}^{\mathsf{fin}}$ *implies* $m_1 = m_2$ *then* Φ_G *with*

$$\Phi_G(f)(k) = m \quad \textit{iff} \quad \exists n.\ \{n\}^{\mathsf{fin}} \sqsubseteq f \wedge \langle n, \langle k, m\rangle\rangle \in G$$

is an effective operator.

Proof: The proof is analogous to the proof of Theorem 9.1 and, therefore, left to the reader. □

From the Myhill-Shepherdson theorems 9.1 and 9.2 it follows rather immediately that

(1) effective operations $\phi : \mathcal{P} \rightharpoonup \mathbb{N}$ can be extended to partial continuous maps from $[\mathbb{N} \rightharpoonup \mathbb{N}]$ to $\mathbb{N}$ putting

$$\phi(f) = m \quad \text{iff} \quad \exists f_0 \sqsubseteq_{\mathsf{fin}} f.\ \phi(f_0) = m$$

for arbitrary $f : \mathbb{N} \rightharpoonup \mathbb{N}$ and

(2) effective operators $\Phi : \mathcal{P} \to \mathcal{P}$ can be extended to continuous maps from $[\mathbb{N} \rightharpoonup \mathbb{N}]$ to $[\mathbb{N} \rightharpoonup \mathbb{N}]$ putting

$$\Phi(f)(n) = m \quad \text{iff} \quad \exists f_0 \sqsubseteq_{\mathsf{fin}} f.\ \Phi(f_0)(n) = m$$

for arbitrary $f : \mathbb{N} \rightharpoonup \mathbb{N}$.

Obviously, this unique extension property holds also for continuous, not necessarily effective $\phi : \mathcal{P} \rightharpoonup \mathbb{N}$ and $\Phi : \mathcal{P} \to \mathcal{P}$.
This observation can be used for clarifying the notion of *relative computability.*

Definition 9.3 *Let $f, g : \mathbb{N} \rightharpoonup \mathbb{N}$ not necessarily computable. Then g* is partial recursive relative to *f iff $\Phi(f) = g$ for some effective operator Φ.*

Thus, any effective numbering $(\Phi_e)_{e\in\mathbb{N}}$ of effective operators gives rise to an admissible numbering of functions partial recursive relative to f putting $\{e\}^f_e = \Phi_e(f)$. One may show that the partial recursive functions relative to f can be defined inductively like the partial recursive function but adding f as a base function (which can be considered as sort of an *oracle*).
Relative computability can be used for defining a notion of reducibility between sets of natural numbers with better properties than the previously considered notion of many-one reducibility. One says that *A is Turing reducible to B* ($A \leq_T B$) iff the characteristic function for A is partial recursive relative to the characteristic function for B, i.e., iff there is an effective operator taking decision procedures for B to decision procedures for A in a *uniform*

way. It is easily seen that many-one reducibility entails Turing reducibility. But the reverse implication does not hold as shown in [Rog] which generally is a rich source of information about Turing degrees.

We conclude this section by remarking that the results of this section can be extended from 2nd order types to arbitrary finite (and even) recursive types. However, for this purpose one needs an appropriate class of topological spaces as provided by Dana Scott's *Domain Theory* from the end 60ies of the last century. Domain Theory was inspired by recursion-theoretic continuity results à la Myhill-Shepherdson but was developed for the purpose of providing a *semantic foundation for programming languages*.

10 Kleene's Fixpoint Theorems

In ordinary (functional) programming the basic language construct is *general recursion* which, however, does not show up as a construct in the definition of μ-recursive functions. Due to its somewhat stupid character the μ-operator is hardly ever used in actual programming whereas general recursion is ubiquitous. Definitions of functions by general recursion are of the form

$$f(x) \simeq E[f, x]$$

which can be written more compactly as

$$f = \Phi(f)$$

where $\Phi(f)(x) \simeq E[f, x]$. Thus, definition by general recursion postulates the existence of distinguished fixpoints for effective operators. For example if Φ is the identity on $\mathcal{P}$ then all f are fixpoints of Φ whereas when we write "f is defined recursively as $f = f$" we mean the **least** fixpoint $\emptyset$ of Φ.
The importance of the following 1st Fixpoint Theorem by Kleene is that it guarantees the existence of least (w.r.t. $\sqsubseteq$) fixpoints for all effective operators and thus provides the mathematical foundations for the intuitively appealing principle of function definition by general recursion.

Theorem 10.1 (1st Kleene's Fixpoint Theorem)
For every effective operator Φ there is a unique $\mu(\Phi) \in \mathcal{P}$ satisfying

(1) $\mu(\Phi) = \Phi(\mu(\Phi))$ *and*

(2) $\mu(\Phi) \sqsubseteq f$ *whenever* $\Phi(f) \sqsubseteq f$.

Moreover $\mu(\Phi) = \bigcup_{n\in\mathbb{N}} \Phi^n(\emptyset)$ *and can be computed recursively from realizers for* Φ *in a uniform way.*

Proof: Using monotonicity of Φ it follows by induction that the sequence $\Phi^n(\emptyset)$ is increasing w.r.t. $\sqsubseteq$ and, therefore, the union $\bigcup_{n\in\mathbb{N}} \Phi^n(\emptyset)$ is a functional relation, too. We leave it as an exercise(!) to verify that $\bigcup_{n\in\mathbb{N}} \Phi^n(\emptyset)$ is actually partial recursive and that a Gödel number for it can be computed from Gödel numbers for realizers for Φ in a *uniform*[10] way.
If $\Phi(f) \sqsubseteq f$ one easily shows by induction that $\Phi^n(\emptyset) \sqsubseteq f$ and, therefore, $\mu(\Phi) \sqsubseteq f$. Thus $\mu(\Phi)$ satisfies (2). Now if we can show that $\mu(\Phi)$ itself is a pre-fixpoint of Φ, i.e. that $\Phi(\mu(\Phi)) \sqsubseteq \mu(\Phi)$, then by monotonicity of Φ we have $\Phi^2(\mu(\Phi)) \sqsubseteq \Phi(\mu(\Phi))$ from which it follows by (2) that $\mu(\Phi) \sqsubseteq \Phi(\mu(\Phi))$ and, therefore, that $\mu(\Phi) = \Phi(\mu(\Phi))$, i.e. that $\mu(\Phi)$ satifies (1).
For showing $\Phi(\mu(\Phi)) \sqsubseteq \mu(\Phi)$ we will exploit the continuity of Φ as ensured by Lemma 9.2. Suppose that f_0 is a finite function with $f_0 \sqsubseteq \Phi(\mu(\Phi))$. Then by continuity of Φ there is a finite function $g_0 \sqsubseteq \mu(\Phi)$ with $f_0 \sqsubseteq \Phi(g_0)$. Thus, as $\mu(\Phi) = \bigcup_{n\in\mathbb{N}} \Phi^n(\emptyset)$ there is a natural number n with $g_0 \sqsubseteq \Phi^n(\emptyset)$ from which it follows that $f_0 \sqsubseteq \Phi(g_0) \sqsubseteq \Phi(\Phi^n(\emptyset)) \sqsubseteq \bigcup_{n\in\mathbb{N}} \Phi^n(\emptyset) = \mu(\Phi)$ by monotonicity of Φ. Thus, we have shown that for all finite $f_0 \sqsubseteq \Phi(\mu(\Phi))$ it follows that $f_0 \sqsubseteq \mu(\Phi)$ which entails that $\Phi(\mu(\Phi)) \sqsubseteq \mu(\Phi)$ as desired. □

As $\mu(\Phi) = \bigcup_{n\in\mathbb{N}} \Phi^n(\emptyset)$ we have

$$\mu(\Phi)(n) = m \quad \text{iff} \quad \exists k.\ \Phi^k(\emptyset)(n) = m$$

supporting the operational intuition that for computing $\mu(\Phi)(n)$ it suffices to "unfold" the recursive definition of $\mu(\Phi)$ finitely many, say k, times to get $\Phi^k(\emptyset)$ and apply it to the argument n.

There is a remarkable strengthening of Kleene's 1st fixpoint theorem, the so-called 2nd Kleene's fixpoint theorem, saying that for every partial recursive function f there exists a natural number e with $\varphi_e = \varphi_{f(e)}$ (where by a minor *abus de langage* $\varphi_{f(e)}$ stands for the empty function if $f(e)$ is undefined).

[10] meaning precisely that there is a partial recursive function fix_1 such that whenever e realizes some effective operator Φ then $fix_1(e)$ terminates and is a Gödel number for $\mu(\Phi)$

Theorem 10.2 (2nd Kleene's Fixpoint Theorem)
For every partial recursive function f there exists a natural number e with

$$u(e,n) \simeq u(f(e),n)$$

for all $n \in \mathbb{N}$.
Moreover, such an e can be computed recursively from a Gödel number for f, i.e. there exists a total recursive function fix_2 with

$$u(\mathit{fix}_2(e),n) \simeq u(u(e,\mathit{fix}_2(e)),n)$$

for all $e,n \in \mathbb{N}$.

Proof: Suppose that f is partial recursive. Then by the parameter theorem there exists a total recursive function g with

$$u(g(n),m) \simeq u(u(n,n),m)$$

for all $n,m \in \mathbb{N}$. Let n_0 be a Gödel number for $f \circ g$ and define $e = g(n_0)$. Then we have

$$u(e,n) \simeq u(g(n_0),n) \simeq u(u(n_0,n_0),n) \simeq u(f(g(n_0)),n) \simeq u(f(e),n)$$

as desired.
For uniformizing this construction consider the partial recursive function $h(e,n) \simeq u(e,g(n))$ where g is as above. By the parameter theorem there is a total recursive function $\widetilde{h}$ with $\varphi_{\widetilde{h}(e)}(n) \simeq u(e,g(n))$, i.e. $\widetilde{h}(e)$ is a Gödel number for $\varphi_e \circ g$. Now $\mathit{fix}_2 = g \circ \widetilde{h}$ does the job as required due to our previous considerations above. □

For those familiar with untyped λ-calculus it might be informative to point out the analogy of fix_2 with the λ-term

$$Y \equiv \lambda f.(\lambda x.\lambda y.f(xx)y)(\lambda x.\lambda y.f(xx)y)$$

for which we have

$$Yfa = f(Yf)a$$

by β-reduction.
Besides its immediate bizarre consequences (like the existence of a number e with $\varphi_e = \varphi_{e^5+3e+524}$) the second recursion theorem can be put to use for justifying function definitions by transfinite recursion in the theory of constructive ordinals (see e.g. [Rog]) or for the proof of the Kreisel-Lacombe-Shoenfield Theorem as in section 12.

11 Recursively Inseparable Sets and the Kleene Tree

First we show that there exists disjoint r.e. sets A, B which are *recursively inseparable* in the sense that there does not exists a decidable set C with $A \subseteq C$ and $B \subseteq \complement C$.

Lemma 11.1 *Let* $A_i := \{n \in \mathbb{N} \mid \{n\}(n) = i\}$ *for* i=0, 1. *Then* A_0 *and* A_1 *are recursively inseparable.*

Proof: If A_0 and A_1 were recursively separable, i.e., not recursively inseparable, then there would exist a a total recursive function $p : \mathbb{N} \to \{0, 1\}$ such that

$$n \in A_i \quad \text{implies} \quad p(n) = 1 \dot{-} sg(i)$$

for all $n \in \mathbb{N}$ and $i \in \{0, 1\}$. Let $p = \varphi_n$. Then $\varphi_n(n) \in \{0, 1\}$, i.e. $n \in A_0$ or $n \in A_1$. Suppose $n \in A_i$, i.e. $p(n) = \varphi_n(n) = i$. But then by the above consideration we have $p(n) = 1 \dot{-} sg(i)$ from which it follows that $i = 1 \dot{-} sg(i)$ which is impossible. □

Using the existence of recursively inseparable sets one may construct the so-called *Kleene tree.*

Theorem 11.1 *There exists a recursive binary tree* $T_{Kl} \subseteq \{0, 1\}^*$*, the so-called* Kleene tree*, which is infinite but contains no infinite* recursive *path.*

Proof: Let α, β range over $\{0, 1\}^{\mathbb{N}}$. We write $\overline{\alpha}(n)$ for $\langle \alpha(0), \ldots, \alpha(n-1) \rangle$. Let T_{Kl} be the prefix-closed recursive subset of $\{0, 1\}^*$ which is defined as follows

$$\overline{\alpha}(n) \in T_{Kl} \quad \text{iff} \quad \forall m, k < n.\, T(m, m, k) \to \big(\alpha(m) = 0 \Leftrightarrow U(k) = 0\big).$$

Suppose $\overline{\alpha}(n) \in T_{Kl}$ for all $n \in \mathbb{N}$. Then $k \in A_i$ implies $\alpha(k) = i$ for $i = 0, 1$ where A_0 and A_1 are the recursively inseparable sets from Lemma 11.1. Thus α cannot be recursive as then it would recursively separate A_0 and A_1 in contradiction to Lemma 11.1. However, if one takes for α the characteristic function of, say, A_0 then all initial segments of α are within the Kleene tree T_{Kl} from which it follows that T_{Kl} is infinite. □

Using the Kleene tree one can construct the computable functional

$$\phi(\alpha) \simeq \mu k.\, \overline{\alpha}(k) \notin T_{Kl}$$

total on all computable arguments but divergent on some non-computable arguments. This example demonstrates that the notion of totality depends on whether one takes non-computable arguments into account or not!
The Kleene tree is also of great importance in the metamathematics of (systems for) constructive mathematics because it shows that induction over well-founded (binary) trees is not admissible if all functions are assumed as computable, see [TvD] for further dicussion on these aspects.

12 Effective Operations on $\mathcal{R}$

In a previous section we studied effective operations on $\mathcal{P}$ and shown that they are continuous. Some effective operations $\phi : \mathcal{P} \rightharpoonup \mathbb{N}$ are total on total recursive arguments, i.e. $\phi(\{n\})\downarrow$ whenever $\{n\}$ is total. Now if e realizes such a ϕ then it also realizes the total functional on $\mathcal{R}$ which is obtained from ϕ by restriction to total arguments. Now one may consider quite generally *effective total operations on* $\mathcal{R}$ as in the folowing definition.

Definition 12.1 (effective total operations)
A function $F : \mathcal{R} \to \mathbb{N}$ is called an effective total operation on $\mathcal{R}$ *iff there exists a natural number e such that $F(\{n\}) = \{e\}(n)$ for all n with $\{n\}$ total in which case we say that e realizes F and for which we write $e \Vdash F$.* $\Diamond$

The following Kreisel-Lacombe-Shoenfield theorem says that every effective total operation on $\mathcal{R}$ appears as restriction of some effective operation on $\mathcal{P}$ to $\mathcal{R}$ entailing that all effective total operations are continuous w.r.t. the *initial segment topology*, i.e. that the value $F(f)$ is already determined by an initial segment of f. Notice, however, that the effective total operation F defined as

$$F(f) = \mu k.\ \overline{sg \circ f}(k) \notin T_{Kl}$$

(where T_{Kl} is the Kleene tree) cannot be extended to a total continuous functional on the space $\mathbb{N}^{\mathbb{N}}$ of *all* functions on $\mathbb{N}$ to $\mathbb{N}$ endowed with the product topology and often called *Baire space*.

Theorem 12.1 (Kreisel-Lacombe-Shoenfield)
For every total effective operation $F : \mathcal{R} \to \mathbb{N}$ there exists an effective operation $\phi : \mathcal{P} \rightharpoonup \mathbb{N}$ such that $F(f) = \phi(f)$ for all $f \in \mathcal{R}$.

Proof: Relative to an appropriate coding $\langle \dots \rangle$ of $\mathbb{N}^*$ we may consider the following effective enumeration θ of those total recursive functions which are eventually constantly 0

$$\theta_{\langle n_0,\dots,n_{k-1}\rangle}(i) = \begin{cases} n_i & \text{if } i < k \\ 0 & \text{otherwise.} \end{cases}$$

Now suppose that e_0 realizes some effective total operation F on $\mathcal{R}$. We define a 3-ary partial recursive function $h(y,z,x)$ given by the following algorithm:

(1) Compute $\{e_0\}(y)$.

(2) Check whether $\exists k < x.\, T(e_0,z,k)$. If not then deliver $\{y\}(x)$ as result.

(3) Otherwise, i.e. if $x > w := \mu k.T(e_0,z,k)$, check whether $\{e_0\}(y) = \{e_0\}(z)$. If not then deliver $\{y\}(x)$ as result.

(4) Otherwise compute $\{y\}(0),\dots,\{y\}(w)$ and then search for the least m with

$$F(\theta_m) \neq \{e_0\}(y) \quad \text{and} \quad \theta_m(i) = \{y\}(i) \ \text{ for all } \ i \leq w$$

and if you succed deliver $\theta_m(x)$ as result.

By the parameter theorem there exists a 2-ary total recursive function $\widetilde{h}$ with

$$\{\widetilde{h}(y,z)\}(x) \simeq h(y,z,x)$$

for all $x,y,z \in \mathbb{N}$. By inspection of the algorithm for h one gets that

$$\{\widetilde{h}(y,z)\} = \begin{cases} \emptyset & \text{if } \{e_0\}(y)\uparrow \\ \{y\} & \text{if } \{e_0\}(y)\downarrow, \{e_0\}(z)\downarrow \text{ and } \{e_0\}(y) \neq \{e_0\}(z) \\ \theta_m & \text{if } \{e_0\}(y) = \{e_0\}(z) \text{ and } T(e_0,z,w) \text{ where} \\ & w = \mu k.T(e_0,z,k) \text{ and } \{y\}(i)\downarrow \text{ for } i \leq w \text{ and} \\ & m \text{ is the least number with } F(\theta_m) \neq \{e_0\}(y) \\ & \text{and } \theta_m(i) = \{y\}(i) \text{ for } i \leq w \\ \{y\}^{[w]} & \text{otherwise, where } w := \mu k.T(e_0,z,k) \end{cases}$$

where $\{y\}^{[w]}$ stands for the restriction of $\{y\}$ to $\{0,\dots,w\}$.

Thus, by Kleene's 2nd recursion theorem we can find for every $y \in \mathbb{N}$ a natural number n_y with $\{n_y\} = \{\widetilde{h}(y, n_y)\}$ recursively in y. Then for every y we have

$$\{n_y\} = \begin{cases} \emptyset & \text{if } \{e_0\}(y)\uparrow \\ \{y\} & \text{if } \{e_0\}(y)\downarrow, \{e_0\}(n_y)\downarrow \text{ and } \{e_0\}(y) \neq \{e_0\}(n_y) \\ \theta_m & \text{if } \{e_0\}(y) = \{e_0\}(n_y) \text{ and } T(e_0, n_y, w) \text{ where} \\ & w = \mu k.T(e_0, n_y, k) \text{ and } \{y\}(i)\downarrow \text{ for } i \leq w \text{ and} \\ & m \text{ is the least number with } F(\theta_m) \neq \{e_0\}(y) \\ & \text{and } \theta_m(i) = \{y\}(i) \text{ for } i \leq w \\ \{y\}^{[w]} & \text{otherwise, where } w := \mu k.T(e_0, n_y, k) \end{cases}$$

where again $\{y\}^{[w]}$ stands for the restriction of $\{y\}$ to $\{0, \ldots, w\}$.
Now if $\{y\} \in \mathcal{R}$ then $\{e_0\}(y)\downarrow$. Then for n_y it holds that $\{e_0\}(n_y) = \{e_0\}(y)$ (as otherwise $\{n_y\} = \{y\}$ and, therefore, also $\{e_0\}(n_y) = \{e_0\}(y)$). Now let w be the least number with $T(e_0, n_y, w)$. Now if there existed an m with $F(\theta_m) \neq \{e_0\}(y)$ and $\theta_m(i) = \{y\}(i)$ for all $i \leq w$ then we would have $\theta_m = \{n_y\}$ and, therefore, also $\{e_0\}(n_y) = F(\{n_y\}) = F(\theta_m) \neq \{e_0\}(y)$ which clearly is impossible.
Thus, for every $y \in \mathbb{N}$ with $\{y\} \in \mathcal{R}$ one can find a w_y recursively in y such that

$$F(\{y\}) = F(\theta_m) \quad \text{whenever } \theta_m(i) = \{y\}(i) \text{ for all } i \leq w_y.$$

Thus, it follows that $F(\{y\}) = F(g)$ for all $g \in \mathcal{R}$ with $\{y\}(i) = g(i)$ for all $i \leq w_y$ (as otherwise one could decide the halting problem).
The extension of F to an effective operation $\phi : \mathcal{P} \rightharpoonup \mathbb{N}$ is given by the following algorithm. For a given $g \in \mathcal{P}$ search for natural numbers y and w such that

$$(\dagger)\ \{e_0\}(y) = U(w) \text{ and } T(e_0, n_y, w) \text{ and } \{y\}(i) = g(i) \text{ for all } i \leq w$$

where n_y depends recursively on y as described above. Now when one has found such y and w satisfying (†) one applies $\{e_0\}$ to some Gödel number of the "0-continuation"[11] of $\{y\}^{[w]}$ and delivers this as the result of $\phi(g)$.
We now show that this algorithm for computing $\phi(g)$ is extensional and coincides with $F(g)$ whenever g is total recursive. Suppose that for a given

[11] i.e. the function f with $f(i) = \{y\}(i)$ for $i \leq w$ and $f(i) = 0$ otherwise

g natural numbers y and w satisfy condition (†). Then $\{e_0\}(n_y)\downarrow$ and $\{e_0\}(n_y) = \{e_0\}(y)$. Now if there existed a θ_m with $F(\theta_m) \neq \{e_0\}(y)$ and $\theta_m(i) = \{y\}(i)$ for all $i \leq w$ then $\{n_y\}$ were such a θ_m. But then we had

$$\{e_0\}(y) = \{e_0\}(n_y) = F(\theta_m)$$

in contradiction to $F(\theta_m) \neq \{e_0\}(y)$. Thus, we have that

$$\{e_0\}(y) = F(\theta_m) = \phi(g)$$

whenever $\theta_m(i) = \{y\}(i)$ for all $i \leq w$. However, if θ_m is the "0-continuation" of $\{y\}^{[w]}$ then $F(\theta_m)$ depends only on g and w. Considering a minimal such w for the given g one easily sees that $\phi(g)$ actually does not depend on the choice of w. □

We conclude this section by a theorem of Friedberg ([Rog], p.362) showing that "partial effective operations on $\mathcal{R}$" need not be continuous.

Theorem 12.2 (Friedberg)
There exists a partial functional $\psi : \mathcal{R} \rightharpoonup \mathbb{N}$ which is realized by some natural number e, i.e. $\psi(\{n\}) \simeq \{e\}(n)$ for all $n \in \mathbb{N}$ with $\{n\} \in \mathcal{R}$, whose value at $\lambda n.0$ is not determined by an initial segment.

Proof: Let e be a Gödel number of the partial recursive function f with $f(x)\downarrow$ iff $f(x) = 0$ and one of the following two conditions is satisfied

(1) $\forall y \leq x.\ \{x\}(y) = 0$

(2) there exists a natural number z with $\{x\}(z) \neq 0$ and $\forall y < z.\ \{x\}(z) = 0$ such that for some $x' < z$ it holds that $\forall y \leq z.\ \{x'\}(y) = \{x\}(y)$.

The number e realizes a partial functional $\psi : \mathcal{R} \rightharpoonup \mathbb{N}$ because f is extensional on Gödel numbers of total recursive functions which can be seen as follows. If $\{x\} = \lambda n.0$ then $f(x) = 0$. If $\{x\}$ is a total recursive function different from $\lambda n.0$ let z be the least number with $\{x\}(z) \neq 0$. Let x' be the least number with $\{x'\}(y) = \{x\}(y)$ for all $y \leq z$. If $x' < z$ then $f(x) = 0$. If $x' \geq z$ then $f(x)\uparrow$ as condition (2) fails (as it is equivalent to $x' < z$) and condition (1) fails because $x' \leq x$ and, therefore, we have $\{x\}(z) \neq 0$ and $z \leq x' \leq x$.
Obviously, we have $\psi(\lambda n.0) = 0$. If ψ were continuous at $\lambda n.0$ then there would exist a natural number k with $\psi(g) = 0$ for all total recursive functions

g with $g(i) = 0$ for all $i < k$. Now let m be a natural number such that all Gödel numbers for the total recursive function g with

$$g(i) = \begin{cases} 0 & \text{if } i \neq k \\ m & \text{if } i = k \end{cases}$$

are strictly greater than k. Then $\psi(g)uparrow$ due to the definition of ψ wheras $\psi(g)\downarrow$ because g and $\lambda n.0$ have the same value 0 for arguments $i < k$. Thus, the effective functional ψ cannot be continuous at $\lambda n.0$. □

References

[Cut] N. Cutland *Computability* CUP, 1980.

[Rog] H. Rogers *Theory of Recursive Functions and Effective Computability* McGraw-Hill Pub. Comp. , 1967.

[Tr73] A. Troelstra (ed.) *Metamathematical Investigations of Intuitionistic Arithmetic and Analysis* SLNM 344, Springer Verlag, 1973.

[TvD] A. Troelstra, D. vanDalen *Constructivism in Mathematics* 2 vol.'s, North Holland, 1988.

Contents