Definition 2.3.5. A function $f : \mathbb{R}_c \to \mathbb{R}_c$ is *Markov computable* if there exists a partial computable function $\nu : \omega \to \omega$ such that, given any index e of $x \in \mathbb{R}_c$, $\nu(e)$ exists, and is an index of $f(x)$. We call the function $\nu : \omega \to \omega$ the *index function* of f.

The uniform (functional) version of this definition is as follows.

Definition 2.3.6. We call a function $f : \mathbb{R}_c \to \mathbb{R}_c$ *Borel computable* if there exists an oracle Turing Machine Φ such that, for all $x \in \mathbb{R}_c$, any computable Cauchy name $(x_i)_{i\in\mathbb{N}}$ of x, and any $n \in \mathbb{N}$,

$$(\Phi^{(x_i)_i}(n))_{n\in\mathbb{N}}$$

is a fast Cauchy name of $f(x)$.

The definitions above are not due to Markov and Borel (more in §2.3.4); however, we shall use this terminology because it appears to be standard (e.g., [24, p.22]). Borel computability seems to be stronger than Markov computability. Nonetheless, these definitions are equivalent, as we now show.

Theorem 2.3.7 (Kreisel, Lacombe and Shoenfield [317], Markov [353])**.** *A function f is Markov computable iff it is Borel computable.*

Remark 2.3.8. In the literature, Theorem 2.3.7 is sometimes referred to as the Kreisel–Lacombe–Shoenfield–Ceitin Theorem. However, in his paper, Ceitin [82] merely extends the much earlier result of Markov (announced in [350] and published in [353]) to computable Polish spaces. We leave this more general version (for Polish spaces) to Exercise 2.4.35 since its proof is not really that different from the proof we present below. Exercise 2.4.35 will find an unexpected application in Part 2 of the book. In §8.2.4, it will be used to derive a theorem about effective reductions between classes of countable algebraic structures. See also Exercise 2.4.36 for an analogous notion for linear operators on computable Banach spaces.

Proof of Theorem 2.3.7. ($\Leftarrow$) Use the s-m-n Theorem 2.1.6 to compute an index for $(\Phi^{(x_i)_i}(n))_{n\in\mathbb{N}} = f(x)$ from the index for $(x_i)_{i\in\mathbb{N}}$. For that, replace the oracle Turing machine with a Turing machine in which the computable oracle becomes a part of its program.

($\Rightarrow$) This implication is quite neat. Assume f is Markov computable, and let ν be its index function, so if e is an index of a fast Cauchy name of a computable real x, then the function $\nu(e)$ is total and lists a fast Cauchy name of $f(x)$. Our task is to produce a uniform procedure that has to do something with sequences that are not necessarily computable Cauchy names. Since every finite partial sequence is extendible to a computable one, we still need to define what the functional does on them.

The naive idea is to use the first found computable Cauchy name, say an eventually constant one, that agrees with the input on a long enough initial segment. The obvious danger is that we may define the functional inconsistently. Indeed, the outputs for different computable fast Cauchy

names of x must converge to the same $f(x)$. It seems that to achieve this, one needs to be able to see the future.

The actual idea is to "set a trap" using the Recursion Theorem 2.1.13. (In fact, we shall use the Recursion Theorem with Parameters presented as Exercise 2.1.16.) Given φ_e, we will slow it down to define $\varphi_{n(e)}$, which tests what ν (given by Def. 2.3.5) does on index e. It will attempt to copy φ_e for a long enough initial segment, and then wait for an extension of this initial segment that gives a different output under ν. If it has the opportunity, it will choose the values that make the disagreement occur. The paradoxical self-referential nature of the Recursion Theorem will imply that for computable fast Cauchy sequences, from some point on, the disagreement cannot possibly happen. This will allow us to conclude that, no matter how we modify φ_e beyond this initial segment, all possible computable sequences extending this initial segment will yield the same computation for a few inputs. In the definition of the functional, we will refer to $\varphi_{n(e)}$ when we choose our computable approximations. This will resolve the issue informally discussed earlier.

The exact details are a bit tedious, though certainly not difficult. Unless the reader really wants to understand the technical details, the clever formal proof below can be skimmed through, as these techniques won't really be used anywhere in the sequel[2].

Formal proof.* We will use the following notation:

- For a finite tuple σ of natural numbers, let σ' be the infinite string with prefix σ in which the last bit of σ is repeated infinitely often. We identify σ' with a function taking i to the ith element $\sigma'(i)$ of σ', and with some index of this function uniformly computable from σ.
- We also restrict ourselves to σ that are valid partial fast Cauchy, thus in particular making σ' a fast Cauchy for every such σ. Note that ν must halt on each such σ'.
- For a function g, we write $g\restriction_t$ to denote the partial function that is the restriction of g to inputs $\{0,\dots,t-1\}$. We also identify strings with partial functions whose domains are initial segments of ω. For instance, $g\restriction_t$ is identified with the string $\langle g(0), g(1), \dots, g(t-1)\rangle$, provided that the values $g(0), g(1), \dots, g(t-1)$ are defined. Also, for a finite string σ, $g\restriction_t \subseteq \sigma$ means that $g\restriction_t$ is a prefix of σ (and, in particular, that $g(0), g(1), \dots, g(t-1)$ are defined).

Now, fix a parameter t. We will define a partial computable function $\varphi_{h(e,t,z)}$ by cases. Using the Recursion Theorem with Parameters (Exercise 2.1.16), replace $n(e,t)$ by z on the right-hand side of the definition below. We have that the index of the left-hand side depends uniformly on e, t, and z, and thus the function we define can be expressed as $\varphi_{h(e,t,z)}$. There is a computable n such that $\varphi_{h(e,t,n(e,t))} = \varphi_{n(e,t)}$. We omit t in $n(e,t)$ in the definition below, to simplify notation.

$$\varphi_{n(e)} = \begin{cases} \uparrow & \text{if } \nu(e)\uparrow; \\ \varphi_e & \text{if } \nu(e)\downarrow,\ \varphi_e \text{ is fast Cauchy and either } \nu(n(e))\uparrow \text{ or } \varphi_{\nu(n(e))}\restriction_t \neq \varphi_{\nu(e)}\restriction_t; \\ \sigma' & \text{if } \varphi_{\nu(n(e))}\restriction_t = \varphi_{\nu(e)}\restriction_t, \text{ where } \nu(n(e)) \text{ halts in } s \text{ steps}, \{0,\dots,s-1\} \subseteq \mathrm{dom}(\varphi_e), \\ & \sigma \supseteq \varphi_e\restriction_s, \text{ and } \varphi_{\nu(\sigma')}\restriction_t \neq \varphi_{\nu(e)}\restriction_t, \text{ where } \sigma \text{ is first found}; \\ \varphi_e\restriction_s & \text{otherwise.} \end{cases}$$

[2]We thank our student Ivan Baburin for pointing out that the version of the proof that appeared in print (2026) was incorrect. Below is a corrected version.

That is, we wait for s so that both $\nu(e)$ and $\nu(n(e))$ halt, then wait for φ_e to halt on all inputs $< s$ and demonstrate that the initial segment is fast Cauchy, and then search for such a σ; while we wait and search, we do not extend the domain of $\varphi_{n(e)}$ beyond $0, \ldots, s-1$. Note also that we set $\varphi_{n(e)}$ equal to $\varphi_e \upharpoonright_s$ if s is the first bit at which φ_e fails to be fast Cauchy. In the second case we copy φ_e on longer and longer initial segments that are fast Cauchy.

If φ_e is indeed a fast Cauchy sequence, then (in particular) $\nu(e)$ is defined. We cannot have $\nu(n(e))$ divergent or $\varphi_{\nu(n(e))} \upharpoonright_t \neq \varphi_{\nu(e)} \upharpoonright_t$, because then $\varphi_{n(e)}$ would just copy φ_e, thus contradicting the choice of ν. Therefore, $\varphi_{\nu(n(e))} \upharpoonright_t = \varphi_{\nu(e)} \upharpoonright_t$.

Since for every σ', we have that necessarily ν converges on (the fixed index for) σ', we cannot possibly have $\varphi_{n(e)} = \sigma'$, since this would contradict the properties of ν. Notice that this step of our analysis works even if φ_e is not fast Cauchy or not even total, as long as s is such that both $\nu(e)$ and $\nu(n(e))$ halt and we see that $\varphi_{\nu(n(e))} \upharpoonright_t = \varphi_{\nu(e)} \upharpoonright_t$. Indeed, if we had $\varphi_{n(e)} = \sigma'$, then by the properties of ν necessarily $\varphi_{\nu(\sigma')} = \varphi_{\nu(n(e))}$, where σ' is identified with its specific "nice" index, and $n(e)$ is perhaps some other index. Thus, we would have $\varphi_{\nu(n(e))} \upharpoonright_t = \varphi_{\nu(\sigma')} \upharpoonright_t \neq \varphi_{\nu(e)} \upharpoonright_t$ (as would be required by the definition of $\varphi_{n(e)}$) and $\varphi_{\nu(n(e))} \upharpoonright_t = \varphi_{\nu(e)} \upharpoonright_t$, which is nonsense.

So, in the case where we have parameter s defined and $\varphi_{\nu(n(e))} \upharpoonright_t = \varphi_{\nu(e)} \upharpoonright_t$, for any σ extending $\varphi_e \upharpoonright_s$, we have that $\varphi_{\nu(\sigma')} \upharpoonright_t = \varphi_{\nu(e)} \upharpoonright_t$, even if φ_e is itself not necessarily fast Cauchy, as may be discovered in the future.

Suppose now that we have φ_i for which $\nu(i)$ and $\nu(n(i))$ are also defined, and such that φ_i agrees with φ_e on inputs $\{0, \ldots, s-1\}$. Here $s = s(e)$ is the parameter defined above corresponding to e and $n(e)$. Recall also that t was fixed earlier, but was omitted throughout; we use the same t for φ_i and $\nu(i)$. Then, as before, we have $\nu(i) = \nu(n(i))$, in $s' = s(i)$ steps, and additionally $\mathrm{dom}(\varphi_i) \supseteq \{0, \ldots, s'-1\}$. We can pick an arbitrary σ that extends both $\varphi_e \upharpoonright_s$ and $\varphi_i \upharpoonright_{s'}$ to conclude that

$$(\ddagger): \quad \varphi_{\nu(i)} \upharpoonright_t = \varphi_{\nu(\sigma')} \upharpoonright_t = \varphi_{\nu(e)} \upharpoonright_t .$$

If φ_i is fast Cauchy, and we could somehow manage to find e' and $s(e')$ with a similar property for a larger t', and then e'' and $s(e'')$ for t'', and so on, it would give us a way of calculating $\lim_j \varphi_{\nu(i)}(j)$ without necessarily any reference to j.

Define a Turing functional Φ as follows. On input $(a_i)_{i\in\mathbb{N}}$ and an integer t, perform the following steps. Keep verifying that $|a_i - a_{i+1}| < 2^{-i-1}$; if an i is found for which it fails, then declare the functional divergent. Simultaneously, search for an index e such that $\nu(e)$ is defined, $\nu(n(e, t+1))$ halts in s steps, and φ_e agrees with $(a_i)_{i\in\mathbb{N}}$ on the first s bits. Decare

$$\Phi^{(a_i)_i}(t) = \varphi_{\nu(e)}(t).$$

Note that if $(a_i)_{i\in\mathbb{N}}$ is a computable fast Cauchy sequence, then it will have an index, and eventually (as we argued earlier), such a computation for this particular index will be found, unless some e satisfying the description is found earlier. By $(\ddagger)$ and the discussion after it, we conclude that $\lim_t \Phi^{(a_i)_i}(t) = \lim_t \varphi_{\nu(i)}(t)$, where i is any index for $(a_i)_{i\in\mathbb{N}}$. □

We remark that a lot of classical elementary real analysis can be effectivised using Markov computability. This is the gist of the book by Aberth [1].

2.3.2 The uniform approach to computability

We now consider several definitions of a computable real-valued function that are not restricted to computable reals. All these notions turn out to be equivalent.