Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 218 (2008) 337–353
www.elsevier.com/locate/entcs

A Functional Algorithm for Exact Real Integration with Invariant Measures

Adam Scriven[1]

School of Computer Science
University of Birmingham
Birmingham, England

Abstract

We develop an algorithm for exact real integration over a class of self-similar spaces and measures defined by Hutchinson. We construct the algorithm in an idealised lazy functional programming language and prove its correctness using domain theory. The work generalises an algorithm developed by Alex Simpson for exact Riemann integration over the real line. We implement the algorithm in the functional language Haskell and give some preliminary results.

Keywords: Exact real number computation; integration; self-similarity; iterated function system; domain theory; lazy functional programming; Haskell.

1 Introduction

Exact real number computation is useful from both a practical and theoretical point of view. On the practical side, it allows calculations to be made without the propagation of errors inherent in floating point arithmetic, albeit at the inevitable loss of efficiency. Theoretically, the study of exact real arithmetic sheds light on questions regarding the computability of real numbers and function(al)s. One functional that has been the focus of detailed study is integration. Edalat and Escardó [3], and later Simpson [15] constructed algorithms for exact integration over the reals. They used different methods, but both cases made essential use of the following identity:

$$\int_0^1 f\ dx = \tfrac{1}{2}\int_0^1 f(x/2)\ dx + \tfrac{1}{2}\int_0^1 f((x+1)/2)\ dx \tag{1}$$

$$\int_0^1 f(x) + c\ dx = \int_0^1 f(x)\ dx + c \tag{2}$$

[1] Email: a.scriven.03@cantab.net

doi:10.1016/j.entcs.2008.10.020

Intuitively, the idea is to recursively apply (1) splitting the integrand into simpler and simpler fragments until constants can be extracted using (2).

In this paper we generalise Simpson's algorithm [15] over a large class of measurable spaces. The measures/spaces in question are the so called *invariant measures/spaces* [10], which we discuss in Section 2.6. Informally these spaces have the defining characteristic that when a segment is viewed, it resembles the original space topologically. Consequently these spaces can be defined in a recursive manner, making them computationally appealing. Such spaces arise naturally in mathematics: The real line exhibits this property, as do many fractal spaces such as the Sierpinski gasket and Falconers fractal. Indeed in this this study we show why it is precisely this property of invariance that allows the algorithms in [3,15] to work.

This paper is a condensed version of the masters thesis by the author[14].

1.1 Related Work

As we have already mentioned, algorithms for exact integration on the reals exist in the literature, but differ in their implementation. In [3], Edalat and Escardó used the programming language `RealPCF`, a form of PCF augmented with an extensional datatype `real` representing the reals. This language required special primitive constructs to work with the abstract dataset `real`. In particular parallel constructs were found to be a necessary component. Moreover in order to compute (1) it was necessary to evaluate both sub-integrals in parallel. This non-sequentiality made their integration algorithm difficult to implement in practice, and it was unclear how a sequential equivalent could be constructed - indeed Gianantonio conjectured in his thesis that no such algorithm existed [6]. Simpson [15] later achieved a sequential integration algorithm using an intensional data type, representing the reals as a stream of digits. This language required no additional primitives, as arithmetic was performed directly on the representations and through the use of Berger's universal quantifier [2]. The work in this study closely follows the style of Simpsons paper.

2 Background

2.1 Computational model

We develop our algorithm in an informal lazy functional programming language, much like that used in [15]. As Simpson remarks, such languages "provides a natural implementation style for exact real algorithms", as well as being readable and intuitive. Our informal language has the following type structure:

$$\sigma, \tau ::= \texttt{Int} \mid \texttt{Bool} \mid \sigma \times \tau \mid \sigma \to \tau \mid [\sigma]$$

where `Int` and `Bool` are the Integer and Boolean datatypes respectively. The constructors $\to$ and $\times$ are the familiar application and product, the term $[\sigma]$ denotes the datatype of lazy streams of elements of type σ. For clarity, we shall also make use of finite types in this paper. The stream datatypes have the usual constants

$\texttt{head} : [\sigma] \to \sigma$, $\texttt{tail} : [\sigma] \to [\sigma]$ and $\texttt{cons} : \sigma \times [\sigma] \to [\sigma]$, the latter having infix form $(\texttt{:})$.

Being an informal language, the denotational semantics is also informal. Suffices to say it follows the standard interpretation from the Scott model of partial continuous functionals P, which form a Cartesian closed category of dcpos. The ground types have interpretation $P_{\texttt{Int}} = \mathbb{N}$ and $P_{\texttt{Bool}} = \mathbb{B} := \{tt, ff\}$. The subset of total elements is denoted $T_\sigma \subseteq P_\sigma$. We assume the reader is familiar with domain theory, and refer them to [1,8,16] for a detailed survey. Denotationally, streams of type $[\sigma]$ can be interpreted as elements of type $P_{\texttt{Int}} \to P_\sigma$. Thus, in proving correctness we treat our programs as expressions in PCF interpreted in the Scott model, thus maintaining full mathematical rigour. Informally, we may consider our functional programming language to be PCF extended with lazy streams.

2.2 Denotation of streams

For a set X, denote the set of finite sequences of elements in X by X^*, and the set of infinite sequences by X^ω. We write $X^\infty := X^* \cup X^\omega$ for the set of all sequences. For a sequence $\alpha \in X^\infty$ we write $|\alpha|$ for the (possibly infinite) length of α. For $0 \leq i < |\alpha|$, the i^{th} element of α is denoted by α_i. The concatenation of a finite sequence $\alpha \in X^*$ and an arbitrary sequence $\beta \in X^\infty$, is denoted $\alpha\beta$. We write $\alpha\lceil_n$ for the largest prefix β, of α, such that $|\beta| \leq n$. As with domains, for a finite stream α we denote the set of all streams with α as a prefix by $\uparrow\alpha$, and define $\Box\alpha = \uparrow\alpha \cap X^\omega$ to be the subset of infinite streams with α as a prefix. Finally, for $x \in X$, we write x_ω for the infinite constant sequence of x's. In our denotational semantic, we interpret $[\![[\sigma]]\!]$ as $[\![\sigma]\!]^\infty$, the finite elements of which are the finite sequences of finite elements of $[\![\sigma]\!]$.

2.3 Universal quantification on streams

In a recent paper, Escardó [4] showed how to extend Berger's universal quantifier to the class of so called *exhaustible* spaces. Briefly, a set $K \subseteq D$ is exhaustible if there exists a computable functional $\forall_K : (D \to \mathbb{B}) \to \mathbb{B}$ such that for every predicate $p \in (D \to \mathbb{B})$ defined on K,

$$\forall_K(p) = \begin{cases} \text{true} & \text{if } p(x) = tt \text{ , for all } x \in K \\ \text{false} & \text{if } p(x) = ff \text{ for some, } x \in K, \end{cases} \tag{3}$$

that is, universal quantification is computable. Escardó showed that the generalised Cantor spaces over n objects, n^ω, is exhaustible, by supplying an algorithm `cantor`,

```
type Quantifier a = (a → Bool) → Bool
cantor :  N → Quantifier (N → N)
```

such that, for all $n \in \mathbb{N}$, $\texttt{cantor}(n)$ satisfies (3) with $K = n^\omega$. We shall make use of this algorithm in Section 3 to develop our generalised integration algorithm.

2.4 Exact real arithmetic

There are numerous approaches to representing the real numbers within a computational framework. Some methods seek to represent the reals extensionally i.e. as an abstract datatype such as the interval domain, while others are intensional in that the reals are represented using existing type structures (see e.g. [7,5] for a summary). We favour the latter approach, in particular infinite streams of digits as studied by Kreitz et. al. [11].

For our purposes, a representation of X is a surjective (partial) map $\delta : A \subseteq \mathbb{F} \to X$, where $\mathbb{F} = (\mathbb{N} \to \mathbb{N}) \equiv \mathbb{N}^\omega$. The Baire space $\mathbb{F}$ has the product topology, which coincides with the subspace topology of the Scott topology on $\mathbb{N}^\infty$. Given representations $\delta : A \to X$, $\gamma : B \to Y$ we say that a continuous $f : A \to B$ is *real* [2] if there exists a continuous $\tilde{f} : X \to Y$, such that $\gamma \cdot f = \delta \cdot \tilde{f}$. We call f the *realization* of $\tilde{f}$ (w.r.t. δ and γ). Furthermore we say f is *real-total* if it is total and the restriction to the total function is real. The following definition is due to Kreitz et. al. [11]

Definition 2.1 Let (X, τ) be a T_0-space with countable base. A representation $\delta :\subseteq \mathbb{F} \to X$ is *admissible* (with respect to τ) if it is continuous and for all continuous functions $\phi :\subseteq \mathbb{F} \to X$, there exists a continuous $\theta : \mathbb{F} \to \mathbb{F}$ such that $\phi = \delta \cdot \theta$.

Admissible representations have remarkable properties: Given representations $\delta : A \to X$ and $\gamma : B \to Y$ with γ admissible, every continuous $g : X \to Y$ is realized by some continuous $f : A \to B$. Moreover, the quotient topology induced on Y by δ corresponds precisely to the existing topology on Y, i.e. the representation preserves topological structure. Many typical representations such as decimal expansions and Cauchy sequences fail to be admissible. A typical counter example is the lack of a continuous (and hence computable) algorithm for multiplication by 3 in the decimal representation (see e.g. [7]).

2.5 The signed binary representation

One of the simplest admissible representation is the *signed binary representation* $q : \mathbf{3}^\omega \to [-1, 1]$, where $\mathbf{3} := \{-1, 0, 1\}$, and $q(\alpha) = \sum_{i=1}^{\infty} \alpha_i 2^{-i}$. This representation has been used successfully by many people in studying exact arithmetic [13,15]. In our lazy functional language 3^ω can be implemented via the type `interval`:

```
type three = {-1,0,1}
type interval = [three]
```

Thus $[\![\texttt{interval}]\!] = \mathbf{3}^\infty$, where only the (total) elements $\mathbf{3}^\omega$ are interpreted as real numbers. Similarly, only the (total) functions $f : \mathbf{3}^\infty \to \mathbf{3}^\infty$ correspond to functions on real numbers. If this were to be implemented in PCF say, then $\mathbf{3}^\infty$ would be interpreted as a (closed) subset of $(\mathtt{N} \to \mathtt{N})$, and the corresponding (partial) functionals would only be necessarily defined on $\mathbf{3}^\infty$.

[2] this terminology stems from [15]

The admissibility of the signed binary representation allows us to guarantee that every continuous real valued function has a continuous real-total counterpart in the domain model. The following results are generalisations of those given in [15, Proposition 1].

Lemma 2.2

(i) *Given representations $\delta : A \to X$, any continuous $\theta : X \to [0,1]$ can be realised by a map $\phi : A \to \mathcal{3}^\omega$ such that $\theta \cdot \delta = q \cdot \phi$.*

(ii) *Let σ be a type in our language. Then for any continuous $\phi : T_\sigma \to \mathcal{3}^\omega$, there exists a total $\phi : P_\sigma \to \mathcal{3}^\infty$ such that ϕ restricts to θ.*

Proof.

(i) Since $q : \mathbf{3}^\omega \to [-1,1]$ is admissible, for any surjective $f :\subseteq \mathbb{F} \to [-1,1]$ there exists a continuous $\phi : \mathbb{F} \to \mathbb{F}$ such that $f = q \cdot \phi$ on $\mathrm{dom}(f)$. Taking $f = \delta \cdot \theta$ suffices.

(ii) Immediate from T_σ being dense in P_σ, and 3^∞ being a Scott domain and therefore densely injective.

□

2.6 Invariant spaces and measures

Many interesting spaces can be constructed by repeatedly applying a set of simple transformations to an initial space. For example, the Cantor space embedded in $\mathbb{R}$, as shown in Figure 1, can be constructed by repeatedly applying the following set of maps

$$\mathcal{C} = \{S_1 : x \mapsto \tfrac{x}{3}, S_2 : x \mapsto \tfrac{x+2}{3}\} \text{ where } \mathrm{dom}(S_i) = [0,1] \tag{4}$$

Fig. 1. Partial illustration of the first 3 iterations for embedding the Cantor space in the real line.

These spaces were studied by Hutchinson [10], to whom the following definition is attributed.

Definition 2.3 Let (X,d) be a complete metric space. A closed, bounded, non-empty $K \subseteq X$ is called *invariant* w.r.t a (finite) set of contraction maps $\mathscr{S} = \{S_1, ..., S_n\}$ if:

$$K = \bigcup_{i=1}^{n} S_i(K)$$

The set $\mathscr{S}$ is often referred to as an *Iterated Function System* or *IFS* [9]. The startling result discovered by Hutchinson in [10] is that a (unique) invariant set always exists for any given $\mathscr{S}$.

Theorem 2.4 *Let (X, d) be a complete metric space. Then for any (finite) set of contractions $\mathscr{S}$ on X, there exists a unique closed bounded non-empty $K \subseteq X$ which is invariant w.r.t $\mathscr{S}$. Moreover K is compact.*

Throughout this paper we denote the invariant space w.r.t $\mathscr{S}$ by $|\mathscr{S}|$, or by K when there is no ambiguity as to the IFS.

2.7 The co-ordinate map

In order to perform constructive analysis on invariant spaces, we need a suitable representation. The co-ordinate map $\pi : n^\omega \to |\mathscr{S}|$ is a natural candidate, defined by

$$\{\pi(\alpha)\} = \bigcap_{i=1}^{\infty} S_{\alpha_1 \ldots \alpha_i}(X)$$

This map is well-defined, surjective and therefore a representation of $|\mathscr{S}|$. Moreover it is continuous w.r.t the product topology on n^ω and the subspace topology on $|\mathscr{S}|$ inherited from the metric space (X, d). However, in general π is not admissible, an obvious example being the familiar binary representation $\pi : \mathbf{2}^\omega \to [0, 1]$ where $S_1 : x \mapsto \frac{x}{2}$ and $S_2 : x \mapsto \frac{x+1}{2}$. Fortunately this is not a problem, as we shall see in Section 3.

2.8 Invariant measures

Hutchinson showed that just as $|\mathscr{S}|$ is exhibited as the fixed point of $\mathscr{S}$ applied to X, a natural measure exists on $|\mathscr{S}|$ as the fixed point of an analogous application of $\mathscr{S}$ on $\mathcal{M}$, the set of Borel regular measures on X having bounded support and finite mass. Hutchinson originally used these "invariant measures" to distinguish between IFS's that gave the same invariant space but were inherently different. However the same measures are also natural candidates for integration.

Suppose we have a set $\rho = \{\rho_1, ..., \rho_n\}$ such that $\rho_i \in (0, 1)$ and $\sum \rho_i = 1$. Given a set of contractions $\mathscr{S}$, define $(\mathscr{S}, \rho) : \mathcal{M} \to \mathcal{M}$ by:

$$(\mathscr{S}, \rho)(\mu)(A) = \sum_{i=1}^{n} \rho_i \mu(S_i^{-1}(A))$$

It is clear from the definition that $M((\mathscr{S}, \rho))(\mu) = M(\mu)$ so the map restricts to $(\mathscr{S}, \rho) : \mathcal{M}^1 \to \mathcal{M}^1$, where $\mathcal{M}^1 \subseteq \mathcal{M}$ is the subset of measures having mass $M(\mu) = 1$. Intuitively we interpret ρ_i as the weight/mass of the component $S_i(K)$ of K.

Theorem 2.5 *There exists a unique $\mu \in \mathcal{M}^1$ such that $(\mathscr{S}, \rho)\mu = \mu$.*

We denote the unique μ by $\|\mathscr{S}, \rho\|$. As an immediate corollary we obtain the following identity, which motivates the next section.

$$\int_{|\mathscr{S}|} f \, d\|\mathscr{S}, \rho\| = \sum_{i=1}^{n} \rho_i \int_{|\mathscr{S}|} (f \cdot S_i) \, d\|\mathscr{S}, \rho\| \tag{5}$$

3 Exact integration on invariant spaces

The definition of invariant spaces captures the property of the real line used by [15,3] to compute integration. Indeed equation (5) generalises the identity (1). This special case is arguably the simplest form of integration on a self-similar space, where it suffices to only compute the average of two real numbers. For general invariant spaces, it is clear from (5) that we need an algorithm for weighted sums of arbitrarily many real numbers. In order to develop such an algorithm, we need to consider an alternative representation of the unit interval.

3.1 A "streams-of-streams" representation of the unit interval

In order to be able to develop a Riemann integration algorithm, Simpson needed the averaging algorithm to have the property that $|\mathtt{avg}(x, y)| \geq \min(|x|, |y|)$ i.e. for every digit of input from x and y, `avg` outputs at least a single digit. We will call such programs (computationally) *stable*. Simpson showed that no stable algorithm for averaging existed in the signed binary expansion, and solved the problem by making use of an intermediate representation, namely, the dyadics $\mathbb{D} := \mathbb{Q}_d \cap [-1, 1]$ with representation $q_d : \mathbb{D}^\omega \to [-1, 1]$ defined by $q_d(\alpha) = \sum_{i=1}^{n} \alpha_i 2^{-i}$ as a natural extension of q. In our more general setting, we need yet another intermediate representation, as the following result illustrates.

Lemma 3.1 *There is no stable algorithm for performing weighed averaging in the signed binary* or *dyadic arithmetic.*

Proof. The existence of a stable weighted averaging algorithm would also imply the existence of a stable multiplication algorithm. A simple counter example is to consider the width of the interval $q(\square 11) \times q(\square 10) = [\frac{1}{8}, \frac{3}{4}]$. □

It is clear that this problem is not unique to the signed binary expansion, but relates to carry overs inherent in the arithmetic. While a possible solution would be to expand the digit set to include all of $\mathbb{Q} \cap [-1, 1]$, allowing multiplication to be performed digit-wise, this can lead to "integers explosion" in the numerator. To sidestep this problem, we make use of a stream-of-streams representation i.e. our digits are themselves streams representing real numbers.

Definition 3.2 Define the space $\mathbf{3}^{\omega\omega} := (\mathbf{3}^\omega)^\omega$ and the representation $q^\omega : \mathbf{3}^{\omega\omega} \to [-1, 1]$ by,

$$q^\omega(\alpha) = \sum_{i=1}^{\infty} q(\alpha_i) 2^{-i}$$

Since $\mathbf{3}^{\omega\omega}$ has the same cardinality as the continuum it can embedded in $\mathbb{F}$ and the machinery from [12] carries over with little difficulty. In particular, q^ω is admissible: it is continuous, and for any continuous $\phi :\subseteq \mathbb{F} \to [-1,1]$, there is a continuous $\theta : \mathbb{F} \to \mathbb{F}$ such that $\phi = q \cdot \theta$, thus $\phi = q^\omega \cdot \theta^\omega$ where $\theta^\omega(\alpha) = \theta(\alpha)^\omega$. We implement $\mathbf{3}^{\omega\omega}$ in our functional programming language using the datatype:

```
type w-interval = [interval]
```

Under this representation it is possible to implement stable multiplication and addition, presented in Figure 2, from which weighted sums can be computed.

```
mul ::  interval → w-interval → w-interval
a `mul` (b : y) = (a ∗ b :  a `mul` y)

add ::  w-interval → w-interval → w-interval
(a : x) `add` (b : y) = (a + b :  (x `add` y) )

sum ::  N → (N → w-interval) → w-interval
sum 1 f = f(1)
sum n f = f(n) `add` sum(n − 1, f)
```

Fig. 2. The operators + and ∗ denote addition and multiplication on the datatype `interval` respectively, as implemented in e.g. [13]. Note that there is no bounds checking on `add`.

Lemma 3.3

(i) $[\![mul]\!]$ *is real-total and stable, with* $q^\omega([\![mul]\!](a,\alpha)) = q(a) \times q^\omega(\alpha)$ *for all* $a \in 3^\omega$, $\alpha \in 3^{\omega\omega}$.

(ii) *sum is real-total and correct. That is, for* $f \in (\mathbb{N} \to$ *w-interval*$)$ *with* $|\sum_{i=1}^n \tilde{f}_i| \leq 1$, $[\![sum]\!]nf \in \mathcal{3}^{\omega\omega}$ *and* $\widetilde{[\![sum]\!]}n\tilde{f} = \sum_{i=1}^n \tilde{f}_i$.

(iii) *sum is stable, i.e.* $|[\![sum]\!]nf| \geq \min_{1\leq i\leq n} |f_i|$, *where here the "digits are elements of* $\mathcal{3}^\omega$.

We can easily convert from `w-interval`s to `interval`s using the algorithm `shift` in Figure 3. We make use of a secondary nine-digit representation $q' : \mathbf{9}^\omega \to [-1,1]$ given by $q'(\alpha) = \sum_{i=1}^\infty \alpha_i 2^{-(i+2)}$.

Lemma 3.4

(i) *For all* $\alpha \in \mathcal{9}^\omega$, *coerce*$(\alpha) \in \mathcal{3}^\omega$ *and* $\widetilde{[\![coerce]\!]} = id$ *i.e.* $q([\![\texttt{coerce}]\!](\alpha)) = q'(\alpha)$.

(ii) *For all* $\gamma \in \mathcal{3}^{\omega\omega}$, *shift*$(\gamma) \in \mathcal{3}^\omega$, *and* $\widetilde{[\![shift]\!]} = id$ *i.e.* $q([\![shift]\!](\gamma)) = q_\omega(\gamma)$.

Proof. It is a simple exercise to show (i). Whence for (ii) it suffices to prove that `shift'` is real-total with $\widetilde{[\![\texttt{shift'}]\!]} = \mathrm{id}$. Since `shift'` outputs one digit for every two digits[3] input, it is total. Finally, let $\gamma = ((a : b : \alpha) : (c : \beta) : \delta)$, then:

[3] where here a digit is an element in $\mathbf{3}^\omega$.

$$\begin{aligned} q_\omega(\gamma) &= \tfrac{1}{2}q(a:b:\alpha) + \tfrac{1}{4}q(c:\beta) + \tfrac{1}{4}q_\omega(\delta) \\ &= \frac{\frac{1}{2}a + \frac{1}{4}b + \frac{1}{4}q(\alpha)}{2} + \frac{\frac{1}{2}c + \frac{1}{2}q(\beta)}{4} + \frac{q_\omega(\delta)}{4} \\ &= \frac{1}{8}(2a+b+c) + \frac{1}{2}\left(\frac{q(\alpha)\oplus q(\beta)}{2} + \frac{q_\omega(\delta)}{2}\right) \\ &= \tfrac{1}{8}d + \tfrac{1}{2}q_\omega([\![\text{avg}]\!](\alpha,\beta):\delta) \end{aligned}$$

where $d = 2a+b+c$. The result follows. □

It is worth noting that we could have used the dyadic representation in place of **9**. The only modification to the algorithm would be that `shift'` outputs $d/4$ rather than d. We can convert from dyadics to signed binary using Simpson's `coerce` algorithm [15]. This approach has the added benefit of greater computational efficiency in performing multiplication, however we favour the nine digit method for its simplicity.

3.2 A first integration algorithm

With an intensionally stable weighted sum algorithm, we are now able to produce a generalised integration program over any (computable) invariant space. We present this algorithm in Figure 4.

Proposition 3.5 *Let $\rho : \mathbb{N} \to 3^\omega$ be such that $\tilde{\rho}(i) \in [0,1]$ for $i = 1,...,n$ and $\sum_{i=1}^{n} \tilde{\rho}(i) = 1$. Then, for any continuous real-total $\phi : n^\omega \to 3^\omega$ it holds that $[\![integrate]\!](n,\rho)\phi \in \mathcal{3}^w$ and*

$$q([\![integrate]\!](n,\rho)\phi) = \int_K \tilde{\phi}\, d\mu$$

where $\mu = \|\mathscr{S}, \rho\|$ is the invariant measure of K.

```
type nine = {−4, −3, −2, −1, 0, 1, 2, 3, 4}
type nine-stream = [nine]
coerce ::  nine-stream → interval
coerce(a:b:x) = let c = 2 ∗ a + b in cases
   c < −4      then −1:coerce(c + 8:x)
   c > 4       then  1:coerce(c − 8:x)
   otherwise then  0:coerce(c:x)

shift' ::  w-interval → nine-stream
shift'( (a : b : x) :  (c : y) :  z ) = let d = 2a + b + c in
   d:shift'( (x `avg` y) :  z)

shift = coerce.shift'
```

Fig. 3. An algorithm for converting from $\mathbf{3}^{\omega\omega}$ to $\mathbf{3}^\omega$. Here `avg` is an algorithm for computing the average of two signed binary streams, see e.g. [13].

```
type Baire = [N]

w-int ::  (N , N → interval) → (Baire → interval) → w-interval
w-int (n,p) = λf. let d = head(f(1_ω)) in
    if (cantor(n))(λv.head(f(v)) == d)
       then d^ωω :  w-int((n,p) tail.f)
       else sum(n,λi. p_i 'mul' w-int((n,p) λv.f(i : v))

integrate = shift.w-int
```

Fig. 4. The generalised integration algorithm: Here `Baire` has denotation $[\![\text{Baire}]\!] = \mathbb{N}^\infty$ and so is the datatype containing the Baire space $\mathbb{N}^\omega$. `cantor(n)` is the universal quantifier over the Cantor space on n^ω.

Note that Lemma 2.2 guarantees that any continuous function $f : K \to [0,1]$ has a continuous real-total realizer of the form $\phi : n^\omega \to 3^\omega$. The proof is similar in concept to a sketch given in [15].

Proof. For convenient, write the functional $[\![\texttt{w-int}]\!](n,\rho)$ as simply $[\![\texttt{w-int}]\!]$. `integrate` is total since `shift` is total and both `sum` and `mul` are stable. By Lemma 3.4 it is sufficient to prove $q^\omega([\![\texttt{w-int}]\!](\phi)) = \int_K \tilde{\phi}\, d\mu$. We will prove by induction on m that

$$\left| q^\omega([\![\text{w-int}]\!](\phi))\lceil_m - \int_K \tilde{\phi}(x)\, d\mu \right| \leq 2^{-m}$$

Where we have used the abuse of notation $q^\omega(\alpha)\lceil_m := \sum_{i=1}^m q(\alpha_i)2^{-i}$ for convenience[4] The case $m = 0$ is trivial. For $m > 0$, consider $h(\phi) : n^\omega \to \mathbf{3}$ defined by $h(\phi)(\alpha) = \text{head}(\phi(\alpha))$. Since ϕ is total, so is $h(\phi)$. We now proceed by inner induction on $emc(h(\phi))$, the extensional modulus of continuity (see e.g. [14,15]. If $emc(h(\phi)) = 0$ then $h(\phi) = d$ is constant. In particular, for all v

$$\phi(v) = d : [\![\text{tail}]\!](\phi(v)) \tag{6}$$

$$\Rightarrow \tilde{\phi}(v) = \tfrac{1}{2}d + \tfrac{1}{2}\widetilde{[\![\text{tail}]\!]}(\tilde{\phi}(v)) \tag{7}$$

In this case, `w-int` outputs:

$$\texttt{w-int}(\phi) = d : \texttt{w-int}(\texttt{tail}.\phi)$$

Thus,

$$[\![\text{w-int}]\!](\phi)\lceil_m = d : [\![\text{w-int}]\!](\text{tail}.\phi)\lceil_{m-1} \tag{8}$$

$$\Rightarrow q^\omega([\![\text{w-int}]\!](\phi))\lceil_m = \tfrac{1}{2}d + \tfrac{1}{2}q^\omega([\![\text{w-int}]\!](\text{tail}.\phi))\lceil_{m-1} \tag{9}$$

Where we have again used the notation $q(\alpha)\lceil_m$ to denote the partial evaluation $\sum_{i=1}^m \alpha_i 2^{-i}$. By the inductive hypothesis on m.

[4] it should be noted that we are not extending q over $\mathbf{3}^\infty$, this is discussed in e.g [7].

$$\left|q^\omega([\![\text{w-int}]\!]([\![\text{tail}]\!](\phi)))\lceil_{m-1} - \int_K \widetilde{[\![\text{tail}(\phi)]\!]}\, d\mu\right| \leq 2^{1-m}$$

by 6 and 8 $$\left|2q^\omega([\![\text{w-int}]\!](\phi))\lceil_m + d - \int_K (2\tilde{\phi}(x) - d)\, d\mu\right| \leq 2^{1-n}$$

$$\left|q^\omega([\![\text{w-int}]\!](\phi))\lceil_m - \int_K \tilde{\phi}(x)\, d\mu\right| \leq 2^{-m}$$

Because $\mu \in \mathcal{M}^1$ i.e. $\mu(K) = 1$, we have the identity,

$$\int_K f(x) + c\, d\mu = \int_K f(x)\, d\mu + c \tag{10}$$

For $emc(h(\phi)) > 0$, we have:

$$\texttt{w-int}((n,p)f) = \texttt{sum}(n, \lambda i.\, p_i \,\texttt{mul}\, \texttt{w-int}(n,p)\lambda v.f(i:v))$$

Since $emc(f.\texttt{cons}_i) < emc(f)$, it follows by the inner inductive hypothesis that $q^\omega([\![\texttt{w-int}]\!](\phi \cdot \texttt{cons}_i)) = \int_K (\tilde{\phi} \cdot \widetilde{[\![\texttt{cons}_i]\!]})(x)\, d\mu$. By Lemma 3.3 we can deduce that, for all m:

$$\left|\sum_{i=0}^{n} \tilde{\rho}_i \int_K \widetilde{f \cdot \texttt{cons}_i}\, d\mu - q^\omega\Big([\![\text{sum}]\!](n, \rho, \lambda i.\tilde{\rho}_i \times [\![\texttt{w-int}]\!](f \cdot \texttt{cons}_i))\Big)\Big\lceil_m\right| \leq 2^{-m}$$

By (5) the result now follows. □

3.3 A more general integration algorithm

Most "interesting" functions of the form $f : |\mathscr{S}| \subseteq \mathbb{R}^m \to \mathbb{R}$ are very hard to realize in the form $f : n^\omega \to \mathbf{3}^\omega$. One solution to this problem would be to find an algorithm that implemented the co-ordinate map $\pi : n^\omega \to X$. If such an algorithm existed, then functions of the form $f : X \to \mathbf{3}^\omega$ could be automatically converted to the form $g = f \cdot \pi : n^\omega \to \mathbf{3}^\omega$ which can be integrated by `integrate`. This approach is discussed briefly in Section 4.1, in this section we adopt a different approach.

The algorithm can be modified to integrate over (a representation of) the metric space X containing $|\mathscr{S}|$. This still gives the same result, as the measure has support $\text{spt}(\|\mathscr{S}, \rho\|) = |\mathscr{S}|$. The only condition on the representation of X is that it is exhaustible, as discussed in Section 2.3, in order that we may quantify over it.

The modified algorithm is given in Figure 5. The data type `IFS` defined therein encapsulates the essential information necessary to perform integration. Observe the original algorithm can be obtained from `x-integrate` given input $\mathscr{S} = (0_\omega, \texttt{cantor}(n), n, \lambda i.\texttt{cons}_i, \rho)$, so we may think of `x-integrate` as a generalisation of `integrate`.

There is a subtle complication to be addressed here: When K is represented by n^ω, the contractions S_i are realized as the functions $\texttt{cons}_i$, which is vital for the inductive step in reducing the *emc* of the integrand. That is, for a given representation q we need the S_i to be such that $|S_i(x)| > |x|$. This is a strengthening of the

```
type IFS a =
(a, Quantifier a, N, N → (interval → interval), N → interval)

x-int ::  IFS a → (Baire → interval) → w-interval
x-int 𝒮 = λf. let d = head(f(ι)) in
   if ∀_X(λv.head(f(v)) == d)
      then d_ωω :  x-int(𝒮 tail.f)
      else sum(n,ρ,λi. ρ_i 'mul' x-int(𝒮 f.S_i))
   where (ι,∀_X,n,S,ρ) = 𝒮

x-integrate = shift.x-int
```

Fig. 5. A more practical integration algorithm.

stability condition discussed in Section 3.1. We call such functions *computationally contractive* with respect to q, or simply computationally contractive (c.c.) when there is no ambiguity as to the representation.

Unfortunately, under a general representation $\delta :\subseteq \mathbb{F} \to X$, the maps $\mathscr{S}$ are not guaranteed to be c.c. under δ. For example the map $x \mapsto \frac{x}{3}$ has no c.c. representation in the signed binary arithmetic. This would seem to potentially limit the success of using an arbitrary representation space - a new result is needed:

Theorem 3.6 *Given any set of functions $S_i : \mathfrak{Z}^\infty \to \mathfrak{Z}^\infty$ corresponding to an IFS $\mathscr{S}$, as well as coefficients ρ, we can construct a set of computationally contractive functions $\mathscr{S}'$, and a set of coefficients ρ' such that:*

(i) *$\mathscr{S}$ and $\mathscr{S}'$ induce the same invariant space i.e. $|\mathscr{S}'| = |\mathscr{S}|$*

(ii) *$\mathscr{S}, \rho$ and $\mathscr{S}', \rho'$ induce the same invariant measure i.e. $\|\mathscr{S}', \rho'\| = \|\mathscr{S}, \rho\|$*

In short this result states that any invariant space we care to define can be generated by a set of contractions that have c.c. realizations. We shall prove the result in section 3.5, but for now satisfy ourselves with some examples.

3.4 Experimental results

We implemented `x-integrate` in Haskell and ran some initial experiments on various functions. For our experiments we took $X = [-1,1]^2$, represented by $q^2 : \mathbf{3}^\omega \times \mathbf{3}^\omega \to [-1,1]^2$; $q^2 : (\alpha,\beta) \mapsto (q(\alpha), q(\beta))$. The integrand itself was $f : (x,y) \mapsto x^2$, and we implemented c.c. IFS's for the Sierpinski gasket, and Falconers Fractal (see Figure 7) as follows: For the Sierpinski gasket,

$$\mathscr{S}_S = \left\{(x,y) \mapsto \left(\tfrac{x-1}{2}, \tfrac{y+1}{2}\right), (x,y) \mapsto \left(\tfrac{x-1}{2}, \tfrac{y-1}{2}\right), (x,y) \mapsto \left(\tfrac{x+1}{2}, \tfrac{y-1}{2}\right)\right\}$$

Space	Partial output	Approx. value	Analytic value	Time (secs)	Memory (bytes)
Falconer	$[1,-1,1,-1,1]$	$0.34375 \pm \frac{1}{32}$	$1/3$	20.65	$7.1 \cdot 10^8$
Sierpinski	$[1,-1,1,0,1]$	$0.40625 \pm \frac{1}{32}$	$11/27$	113.39	$4.2 \cdot 10^9$

Fig. 6. Results from the integration program applied to various self-similar spaces: The tests were run on a machine with 4, 2.40 GHz Intel® Xeon™ CPU's, with 4GB's of RAM and 8GB's of swap.

with coefficients $\rho_S = \{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\}$, and

$$\mathscr{S}_F = \left\{(x,y) \mapsto \left(\tfrac{x}{2}, \tfrac{y}{2}\right), (x,y) \mapsto \left(\tfrac{x+3}{4}, \tfrac{y+3}{4}\right), (x,y) \mapsto \left(\tfrac{x+3}{4}, \tfrac{y-3}{4}\right), \right.$$
$$\left. (x,y) \mapsto \left(\tfrac{x-3}{4}, \tfrac{y+3}{4}\right), (x,y) \mapsto \left(\tfrac{x-3}{4}, \tfrac{y-3}{4}\right) \right\}$$

for Falconers fractal, with coefficients $\rho_F = \{\frac{1}{2}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}\}$. The results are given in Figure 6.

(a) Falconers Fractal

(b) A variant of the Sierpinski gasket.

Fig. 7. Invariant spaces used in the preliminary experiments

3.5 Proof of Theorem 3.6

We now give a proof to Theorem 3.6. The is proof constructive, which is essential from a practical point of view: If we simply knew that a suitable $\mathscr{S}'$ existed but did not know how to construct it, then we would be no better off than before. We shall prove Theorem 3.6 in an informal manner, consisting of two steps:

1) We firstly show how, given any set of contractions $\mathscr{S}$, we can produce an equivalent (albeit larger) set of contractions with arbitrarily small contractivity.
2) Next, we show how the realizer of a function with sufficiently small contractivity can be manipulated so that it is computationally contractive.

Step 1

We start first with some definitions from [10]: Let I be a finite subset of finite ordered tuples i.e. $I \subseteq \{1,...,n\}^*$. Define $\hat{I} = \{\alpha_1...\alpha_p...|\alpha_i \in I\} \subseteq n^\omega$ where we are concatenating finite tuples from I in the obvious way. We say I is *secure* if for

every $\beta \in n^\omega$ there exists a $\alpha \in I$ such that $\alpha \sqsubseteq \beta$ i.e. α is a prefix of β. Moreover I is *tight* if this α is unique to β. The following result is due to Hutchinson [10].

Theorem 3.7 *Let I be a finite set as above and $\mathscr{S}$ a set of contractions,*

(i) *Let $\mathscr{S}_I = \{S_\alpha \,|\, \alpha \in I\}$. Then $|\mathscr{S}_I| = \{k_\beta \,|\, \beta \in \hat{I}\}$ where $k_\alpha := \bigcap_{p=1}^{\infty} S_{\alpha_1 \ldots \alpha_p}(K)$. Moreover if I is secure then $|\mathscr{S}_I| = |\mathscr{S}|$.*

(ii) *Let ρ be coefficients inducing a measure on $\mathscr{S}$. Define $\rho_I : I \to (0,1)$ by $\rho_I(i_1, ..., i_p) = \rho(i_1)...\rho(i_p)$. Then if I is tight one can check that $\sum_{\alpha \in I} \rho_I(\alpha) = 1$ and furthermore $\|\mathscr{S}_I, \rho_I\| = \|\mathscr{S}, \rho\|$.*

So now given a set $\mathscr{S}$ with $L = \max_{i=1,..,n} \mathrm{Lip}(S_i) < 1$, let $I = \{\text{all permutations of } \langle 1, ..., m\rangle\}$ for some integer $m > 0$. Then I is tight, $|\mathscr{S}_I| = |\mathscr{S}|$, $\|\mathscr{S}_I, \rho_I\| = \|\mathscr{S}, \rho\|$ for any suitable ρ and $L_I := \max_{\alpha \in I} \mathrm{Lip}(S_\alpha) = L^m$. Since L is necessarily less than 1, we can therefore make L_I arbitrarily small.

Step 2

For the second step, we start by outlining some preliminary results. The functions `addone` and `subone` in Figure 8 have the following properties (see e.g. [13,15]):

```
addone,subone ::  interval → interval

addone (1:x) = 1_ω
addone (0:x) = 1:addone(x)
addone (−1:x) = 1:x

subone (1:x) = −1:x
subone (0:x) = −1:subone(x)
subone (−1:x) = −1_ω
```

Fig. 8. The `addone` and `subone` algorithms.

Lemma 3.8 *For all $x \in [-1, 1]$.*

(i) $[\![\widetilde{\textit{subone}}]\!](x) = \begin{cases} x - 1 & \textit{when } x > 0 \\ -1 & \textit{otherwise} \end{cases}$

(ii) $[\![\widetilde{\textit{addone}}]\!](x) = \begin{cases} x + 1 & \textit{when } x < 0 \\ 1 & \textit{otherwise} \end{cases}$

Proof. See e.g. [13] □

Next define the q-equivalence relation, $x \equiv_q y \iff q(x) = q(y)$. From Lemma 3.8 the following result is apparent,

Corollary 3.9 *For any $x \in \mathscr{3}^\omega$,*

(i) *if $q(x) \geq 0$ then $1 : \textit{p-one}(x) \equiv_q x$*

(ii) *if $q(x) \leq 0$ then $-1 : \textit{p-negone}(x) \equiv_q x$*

```
p-one, p-negone, p-zero ::  interval → interval

p-one(0:x) = subone(x)
p-one(1:x) = x

p-negone(1:x) = addone(x)
p-negone(−1:x) = x

p-zero(0:x) = x
p-zero(1:x) = addone(x)
p-zero(−1:x) = subone(x)
```

(iii) *if* $-\frac{1}{2} \leq q(x) \leq \frac{1}{2}$ *then* $0 : \texttt{p-zero}(x) \equiv_q x$

Now suppose we have a function $S : 3^\infty \to 3^\infty$ with contractivity strictly less than $\frac{1}{4}$. Let $c = 4S(0_\omega)_0 + 2S(0_\omega)_1 + S(0_\omega)_2$.

- If $c \geq 3$ then $q(S(0_\omega)) \geq \frac{1}{4}$ and since S has contractivity strictly less than $\frac{1}{4}$ it follows that $q(S(x)) > 0$ for all $x \in 3^\omega$. Thus the function $S' \equiv \lambda x. -1 :$ p-negone $\cdot\, S(x)$ is q-equivalent to S i.e. $S' \equiv_q S$. Moreover S' is computationally contractive.
- Similarly if $c \leq -3$ then $q(S(0_\omega)) \leq -\frac{1}{4}$ and $S \equiv_q \lambda x.(1 :$ p-one $\cdot\, S(x))$
- Finally if $-2 \leq c \leq 2$ then $-\frac{1}{4} \leq q(S(0_\omega)) \leq \frac{1}{4}$ and $S \equiv_q \lambda x.(0 :$ p-zero $\cdot\, S(x))$

So for suitably contractive functions, we can find an equivalent c.c. set of realizers. Thus provided we have prior knowledge as to the contractivity of each $S_i \in \mathscr{S}$ we can construct an equivalent $\mathscr{S}_I$ with contractivity $< \frac{1}{4}$ and then manipulate these functions into a computationally contractive form. This makes `x-integrate` universally applicable on the condition that we have some domain knowledge, which is not an unreasonable assumption.

The crucial point here is that given any $\mathscr{S}$ we can either find a suitable c.c. representation of the functions in $((\mathbf{3}^\infty)^n \to \mathbf{3}^\infty)$ or, failing that, generate an equivalent c.c. set that induces the same invariant space and measure.

4 Conclusion

In this study we have extended Simpson's algorithm over the computable invariant spaces. This is a purely theoretical result, and to the best of our knowledge is the first instance of such an algorithm. In proving correctness we made essential use of domain theory, in particular for proving totality.

The crucial step in this study was the use of an intermediate "streams-of-streams" representation, in order to compute multiplication digit-wise and thus make the algorithm total. We have shown how the signed binary arithmetic cannot contain a suitable multiplication algorithm, and indicated how this may generalise to other representations. We may think of our stream-of-streams representation as

the natural extension of the dyadics, as used by Simpson, to achieve integration in [15].

The characteristic of the real line that motivated the integration algorithms in [15,3] is a fundamental trait of all invariant spaces. Moreover it is the defining characteristic of the invariant measures on invariant spaces i.e. every space that can be integrated in this way is an invariant space w.r.t some $\mathscr{S}$ and conversely so.

From a practical point of view, the algorithm `x-int` provides more flexibility in the definition of the integrands, and so is more applicable. Experimental results indicate that the algorithm can compute reasonably quickly on standard hardware.

4.1 Further Work

Theorem 3.6 was originally part of an algorithm that implemented the co-ordinate map $\pi : n^\omega \to X$, for an arbitrary representation X. The intention was to use the original integration algorithm `integrate` and convert the integrand $f : X \to \mathbf{3}^\omega$ to $g = f \cdot \pi : n^\omega \to \mathbf{3}^\omega$. However, the additional complexity of g proved to be unacceptable in practice. From a theoretical perspective, the existence of an algorithm for π is of interest in itself, and we hope to explore this approach in later work.

Acknowledgments

I am deeply grateful to Martín Escardó for the time, effort and continued encouragement he has given me. In particular I would like to credit him with the original idea for the algorithm `integrate`, and the idea of using the stream-of-streams representation. I would also like to thanks the reviewers for their helpful comments and suggestions.

References

[1] Abramsky, S. and A. Jung, *Domain theory*, Handbook of Logic in Computer Science **3**, Clarendon Press, 1994 pp. 1–168.

[2] Berger, U., "Totale Objecte und Mengen in Bereichstheorie," Ph.D. thesis, University of Munich (1990), phD Thesis.

[3] Edalat, A. and M. Escardó, *Integration in Real PCF*, Information and Computation **160** (2000), pp. 128–166.

[4] Escardó, M., *Infinite sets that admit fast exhaustive search*, Proceedings of LICS'2007 (2007), pp. 443–452.

[5] Geuvers, H., M. Niqui, B. Spitters and F. Wiedijk, *Constructive analysis, types and exact real numbers*, Mathematical Structures in Computer Science **17** (2007), pp. 3–36.

[6] Gianantonio, P. D., "A Functional Approach to Computability on Real Numbers," Ph.D. thesis, University of Pisa, Udine (1993).

[7] Gianantonio, P. D., *Real number computability and domain theory*, Information and Computation **127** (1996), pp. 12–25.

[8] Gierz, G., K. Hofmann, K. Keimel, J. Lawson, M. Mislove and D. Scott, "Continuous Lattices and Domains," Cambridge University Press, 2003.

[9] Hoggar, S. G., "Mathematics for Computer Graphics," Cambridge University Press, 1994.

[10] Hutchinson, J. E., *Fractals and self similarity*, Indiana University Mathematics Journal **30** (1981), pp. 713–747.

[11] Kreitz, C. and K. Weihrauch, *Theory of representations*, Theoretical Computer Science **38** (1985), pp. 35–53.

[12] Kreitz, C. and K. Weihrauch, *Representations of the real numbers and of the open subsets of the set of real numbers.*, Annals of Pure and Applied Logic **35** (1987), pp. 247–260.

[13] Plume, D., *A calculator for exact real number computation* (1998), 4th year project.

[14] Scriven, A., "Functional algorithms for Exact Real Integration over Invariant Measures," Master's thesis, Department of Computer Science, University of Birmingham (2007).

[15] Simpson, A. K., *Lazy functional algorithms for exact real functionals*, Lecture Notes in Computer Science **1450** (1998), pp. 456–465.

[16] Streicher, T., "Domain-Theoretic Foundations of Functional Programming," World Scientific Publishing, 2006.