

COINDUCTIVE FORMAL REASONING IN EXACT REAL ARITHMETIC*

MILAD NIQUI^a

Department of Software Engineering, Centrum Wiskunde & Informatica, Kruislaan 413, NL-1098 SJ, Amsterdam, The Netherlands. Tel: +31 20 5924073, Fax: +31 20 5924200.
e-mail address: M.Niqui@cwi.nl

ABSTRACT. In this article we present a method for formally proving the correctness of the lazy algorithms for computing homographic and quadratic transformations — of which field operations are special cases — on a representation of real numbers by coinductive streams. The algorithms work on coinductive stream of Möbius maps and form the basis of the Edalat–Potts exact real arithmetic. We use the machinery of the *Coq* proof assistant for the coinductive types to present the formalisation. The formalised algorithms are only *partially productive*, i.e., they do not output provably infinite streams for all possible inputs. We show how to deal with this partiality in the presence of syntactic restrictions posed by the constructive type theory of *Coq*. Furthermore we show that the type theoretic techniques that we develop are compatible with the semantics of the algorithms as continuous maps on real numbers. The resulting *Coq* formalisation is available for public download.

INTRODUCTION

Exact real numbers constitute one of the prime examples of infinite objects in computer science. The ubiquity and theoretical importance of real numbers as well as recent safety-critical applications of exact arithmetic makes them an important candidate for applying various approaches to formal verification. Among such approaches one that is tailor-made for infinite objects is *coinductive reasoning*. A careful coinductive formalisation of real numbers has two advantages: (1) it provides a certified package of exact arithmetic; (2) it gives valuable insight into various notions of coinductive proof principles that can contribute to the area of formal verification for infinite objects.

Coinductive reasoning is dual to the usual approach of using algebraic and inductive data types both for computation and reasoning and can be studied from a set theoretical [BM96], category theoretical [JR97], or type theoretical [Coq94] point of view. In all these settings the coinductive structure of real numbers is usually expressible as *streams*

1998 ACM Subject Classification: F.3.1, D.2.4.

Key words and phrases: Lazy Exact Real Arithmetic Coinduction Corecursion Coq.

* Parts of this work have appeared in [Niq06, Niq07b].

^a Former address: Institute for Computing and Information Sciences, Radboud University Nijmegen, The Netherlands. Research supported by the Netherlands Organisation for Scientific Research (NWO).

which have a simple and well-understood shape. Although there are other coinductive objects (e.g. *expression trees* [EP97]) modelling exact real numbers, the stream approach has proven to be expressible enough for most computational purposes. In this approach a real number r is represented by a stream of nested intervals whose intersection is the singleton $\{r\}$. This approach has always been the basis of representing real numbers, as the usual decimal representation is an instance of this representation with digits denoting interval-contracting maps. Because of this, much work has been done in the study and implementation of various algorithms for specific stream based representations. In this context one rather generic approach is the work by Edalat et al. [EP97, PEE97, Pot98, EH02]. There the authors develop the general framework of representations using *linear fractional transformations* that covers all known representations of real numbers that are based on streams of nested intervals. In particular the Edalat–Potts *normalisation algorithm* is a unified algorithm for computing all elementary functions on real numbers.

The present work is part of the ongoing project of the author for formalising and verifying the Edalat–Potts normalisation algorithm. We use the constructive type theory extended with coinductive types to implement and formalise the homographic and quadratic algorithms. These algorithms originated in the exact continued fraction arithmetic [Gos72, Vui90, Les01] and form the basis of the Edalat–Potts algorithm. The two algorithms suffice for equipping the stream representations of real numbers with a field structure and thus are important in themselves both from a theoretical and a practical point of view. The theoretical importance is highlighted when we consider our work as a solution to the problem of equipping the *coalgebraic* structure of real numbers with the *algebraic* properties of a field. This is due to the innate relationship between coinductive types and final coalgebras which we mention in Section 1.

We use the machinery of the *Coq* proof assistant for coinductive types to present the formalisation. Throughout the article we use a syntax loosely based on the *Coq* syntax, adapted for presenting in an article. We present definitions and lemmas, depending on the usage context, as a *Coq* declaration (bound between two horizontal bars) or as ordinary mathematical expressions. Regarding lemmas we follow this convention: If a lemma is to be used as a *Coq* subterm of another lemma or definition then we will present its statement as a *Coq* declaration so that it gets a reusable name; otherwise we present it as an ordinary numbered lemma. In order to improve the legibility of the *Coq* declarations we use the curried version of the functions within *Coq* declarations, e.g. $F\ a\ b$ will be used in *Coq* mode and $F(a, b)$ within the text. All the lemmas in this article are formalised and proven in *Coq*, a list of their machine checked counterparts is given in Appendix B. Most of the proofs of lemmas are omitted in the paper, however in some cases the proof is given in human language. In any case all the machine checked version of the statements and proofs of all the lemmas and theorems can be found in the complete *Coq* formalisation of the material in this article which is available for download in [Niq07a].

Related Work. The stream representation of exact real numbers have been recently formalised in a coinductive setting by Ciaffaglione and Di Gianantonio [CDG06, Cia03], Bertot [Ber07], Hou [Hou06] and Gibbons [Gib07]. Ciaffaglione and Di Gianantonio use the *Coq* proof assistant to formalise a representation of real numbers in $[-1, 1]$ as ternary streams and to prove that paired with the natural number exponents they form a complete Archimedean ordered field. Bertot — using *Coq* as well — formalises a ternary representation of $[0, 1]$ using *affine maps* and formalises affine operations (multiplying by scalars),

addition, multiplication and infinite sums. Hou studies two coinductive representations of signed ternary digits and Cauchy sequences considered as streams, proves their equivalence using set-theoretic coinduction and defines the addition via the average function. Gibbons, as an application of his notion of metamorphism, shows how one can transform various stream representations of real numbers and use the same algorithms for different representations.

Our work, while related, is different from all of the above, in that we formalise two powerful algorithms that give us all field operations on real numbers, including division which seems to be the most difficult one in the other approaches. Furthermore due to the expressiveness of the Edalat–Potts framework, the algorithms that we formalise are in principle independent of any specific representation. For presentation purposes we use a specific representation, but our correctness proof can be adapted for other representations. This is because the correctness proofs have several layers and only one aspect of them is dependent upon the metric properties of the used representation. The coinductive aspect of our work is related to the above work. For example we follow Bertot’s and Hou’s idea of using a coinductive predicate to link real numbers and the streams representing them [Ber05a, Ber07, Hou06]. From a type theoretic point of view the notion of *cofixed point* equations has a central rôle in our development distinguishing it from the above work.

From a historical perspective, the Edalat–Potts algorithm was a step in designing a programming language with a built-in abstract data-type for real numbers [PEE97], in line with the work by Di Gianantonio [DG93] and Escardó [Esc97]. The trade-off between the expressibility and the existence of parallelism in these work led to some improvements on the domain theoretic semantics of the Edalat–Potts algorithm, e.g. as in the recent work in [MRE07] where a sequential language with a non-deterministic cotransitivity test is proposed. This line of research is an instance of the *extensional* approach to exact real arithmetic while our work in which we have direct access to the digits of the representation is a study in the *intensional* exact arithmetic in the sense of [BES02]. However, the actual programs written in the extensional approach do have a coalgebraic nature and are essentially formalisable in the coinductive type theory.

In other related work, Pavlović and Pratt [PP00] study the order properties of the continuum as the final coalgebra for the list functor and stream functor in category **Set** by specifying Cantor space and Baire space in terms of these functors. However, by characterising the continuum only up to its order type, their construction does not address the algebraic properties of real numbers. Freyd gives another characterisation of Dedekind reals (see [Joh02, § D4.7]) in terms of the diagonalisation of a ‘wedge’ functor in a category of posets. In [ES01] the unit interval is constructed as an initial algebra and the Cauchy completeness is defined by uniqueness of a morphism from a coalgebra to an algebra. The big picture that we are working on, i.e., the formalisation of the Edalat–Potts normalisation algorithm is related to the work in [PE98, Sim98, Pat03] that reconcile the coalgebraic structure of real numbers with algebraic operations on them.

The general issue of formalising functions from streams to streams within logical frameworks is studied by [MPC86, Pau94] (using Knaster–Tarski’s fixed point theorem) and [Mat99, DGM03] (using Banach–Mazur’s fixed point theorem). Finally, the recent work in [GHP06] tackles this problem by internalising the notion of *productivity* in a single data type for all such functions. Productive functions are those functions on infinite objects that produce provably infinite output. The above formalisations all focus on formalising *total* productive functions. This is not surprising, given that in type theory we deal with

total functions. However, from the programmer’s point of view, it might be desirable to have a way of dealing with partial functions. Our work differs from the above in that we embark on formalising partial algorithms on infinite objects. In this sense our work is related to the work on formalising general recursion for partial functions on inductive types [DDG98, BC01].

Finally, our focus on the partial productivity is related to the aforesaid domain-theoretic semantics [Esc97] where partial real numbers are denoted by interval and the *strong convergence* (akin to our notion of productivity) of the functions is studied [MRE07]. This relationship is not a coincidence as manifested by the original analytic proof of adequacy for the Edalat–Potts algorithm [PEE97].

1. TYPE THEORETIC COINDUCTION

The *Coq* proof assistant [CDT06] is an implementation of Calculus of Inductive Constructions (CIC) extended with coinductive types. This is an extension of Martin-Löf intensional type theory. Coinductive types are intended for accommodating infinite objects such as streams and infinite trees in type theory [MPC86]. This is in contrast to inductive types which are accommodating well-founded and thus essentially finitistic objects such as natural numbers and trees. The coinductive types were added to *Coq* by Giménez [Gim96]. Their implementation follows the same philosophy as that of inductive types in CIC, namely there is a general scheme that allows for formation of coinductive types if their *constructors* are given, and if these constructors satisfy the *strict positivity* condition. The definition of a strictly positive constructor is identical for inductive and coinductive types and similar to that of a monomial endofunctor (i.e., an endofunctor involving products and exponentials). Intuitively a constructor c is strictly positive with respect to x only if x does not appear to the left of a \rightarrow in a nested occurrence of \rightarrow in the type of c . A formal definition can be found in [PM92]. This means that the following forms an inductive (resp. coinductive) type I in *Coq*, provided that the keyword **Inductive** (resp. **CoInductive**) is given and that all c_i ’s are strictly positive constructors with respect to I .

```
(Co)Inductive I (x1: X1) ... (xi: Xi): ∀(y1: Y1) ... (ym: Ym), s :=
| c1: ∀(z11: Z11) ... (z1k1: Z1k1), I t11 ... t1m+i
    ⋮
| cn: ∀(zn1: Zn1) ... (znkn: Znkn), I t11 ... tnm+i.
```

In such a declaration s is a sort, i.e., $s \in \{\mathbf{Set}, \mathbf{Prop}, \mathbf{Type}\}$. Moreover x_i s (resp. y_i s) are *general* (resp. *recursive*) parameters of I .

For example one can define the set of streams as

```
CoInductive Streams (A : Set) : Set :=
| Cons : A → Streams A → Streams A.
```

Note that this is a polymorphic type forming the streams of elements of its general parameter A . From now on we shall use A^ω to denote the type **Streams A**.

After a coinductive type is defined one can introduce its inhabitants and functions into it. Such definitions are given by a *cofixed point* operator. This operator is similar to the fixed point operator for structural recursion. When given a well-typed definition that

satisfies a *guardedness condition*, this operator will introduce an infinite object that inhabits the coinductive type.

The typing rule for this operator is given by the following judgement (here, let I be a coinductive type with parameters P_0, \dots, P_i).

$$\text{cofix rule} \quad \frac{\Gamma, f: B \vdash N: B \quad B \equiv \forall x_0: X_0, \dots, x_j: X_j, (I \ P_0 \ \dots \ P_i) \ \mathbf{G}(f, B, N)}{\Gamma \vdash \text{cofix } f: B := N: B}$$

According to this rule, if f, B and N satisfy the side condition \mathbf{G} then $\text{cofix } f$ is an inhabitant of type B which is a function type with as codomain a coinductively defined type. In this case N is the body of the definition which may contain f . The side condition $\mathbf{G}(f, B, N)$ is called the *Coq guardedness condition* and is a syntactic criterion that is intended to ensure the productivity of infinite objects. This condition checks whether the declaration of f is guarded by constructors. This means that every occurrence of f in the body N should be the immediate argument of a constructor of some inductive or coinductive type. Note that it need not be the argument of only the constructors of I , and that the constructors can accumulate on top of each other. Thus f occurs guarded if it occurs as $c_0(c_1 \dots (c_m \ f) \dots)$ where each c_i is a constructor of some inductive or coinductive type I_i . This condition is due to Giménez [Gim96] and is based on earlier work of Coquand [Coq94]. A precise definition of \mathbf{G} can be found in [Gim96, p. 175].

Finally we mention the reduction (in fact expansion) rule corresponding to the cofix operator. Let $F \equiv \text{cofix } g: B := N$. Then the *cofixed point expansion* is the following rule.

$$\begin{aligned} & \text{match } (F \ P_0 \dots P_j): X \ \mathbf{with} \ | r_0 \Rightarrow R_0 \ | \dots \ | r_k \Rightarrow R_k \ \mathbf{end} \rightsquigarrow \\ & \text{match } (N[g \leftarrow F] \ P_0 \dots P_j): X \ \mathbf{with} \ | r_0 \Rightarrow R_0 \ | \dots \ | r_k \Rightarrow R_k \ \mathbf{end} . \end{aligned}$$

Thus, the expansion of a cofixed point is only allowed when a case analysis of the cofixed point is done.

It is well-known that coinductive types correspond to weakly final coalgebras in categorical models of intensional type theory [Hag87]. From a coalgebraic point of view this treatment of coinductive types by means of constructors and cofixed point operator might seem unnatural: final coalgebras are about observations and not constructions; final coalgebra should be given using its destructor. Nevertheless, presenting the coinductive types in the *Coq* way, is much closer to the syntax of lazy functional programming languages such as *Haskell*¹ and hence very useful for many applications. Moreover, as we show in Section 3, one can use *Coq* to define a general form of productive functions, allowing one to build more complicated coalgebraic structures. In any case, theoretically this does not change the coalgebraic semantics and the coinductive types can still be interpreted as weakly final coalgebras in any categorical model of CIC (see [AAG05] where a stronger results is proven). Furthermore, the usual coiteration and corecursion schemes can be derived in terms of the operator cofix [Gim95]. Therefore the method that we present in this article using the language of *Coq* can easily be translated into the standard categorical notations in any categorical model of CIC².

¹Note that in *Haskell*— where there is no distinction between inductive and coinductive types— all data-types can be considered to be potentially infinite and hence correspond to *Coq*'s coinductive types.

²In fact, in the present article we do not need the *universes* in CIC and therefore categorical models of simpler extensions of Martin-Löf type theory — such as Martin-Löf categories of Abbott et al [AAG05] — will suffice.

The guardedness condition of *Coq* is one among many syntactic criteria for ensuring productivity. Examples include *corecursion* [Geu92], *dual of course of value recursion* [UV99], *T-coiteration for pointed functors* [Len99], *λ -coiteration for distributive laws* [Bar01] and *bialgebraic T-coiteration* [CHL03], each handling an ever expanding class of specifications. However, the productivity of the algorithms on real numbers cannot be syntactically detected. In fact the productivity of the standard filter function on stream of natural numbers with the following specification is also not decidable (here P is a boolean predicate on natural numbers and we use $x :: xs$ to denote $\mathbf{Cons} \ x \ xs$).

$$\mathbf{filter} \ (x :: xs) := \begin{cases} x :: \mathbf{filter} \ xs & \mathbf{if} \ P(x) \ , \\ \mathbf{filter} \ xs & \mathbf{otherwise.} \end{cases}$$

By suitably choosing P one can reduce the problem of the productivity of the above function to an open problem in mathematics; see [Niq04, Example 4.7.6] for a choice of P which shows that the productivity of the above function is equivalent to whether there are infinitely many twin prime numbers.

Therefore it seems that providing syntactic productivity tests cannot cover the most general class of recursive specifications for infinite objects. One possible solution is to adhere to semantic means in order to be able to formalise such programs using one of the above schemes. For instance, for the case of filter on prime numbers, one has to (1) consider a number theoretic constructive proof of the infinitude of primes, (2) from this proof extract a function κ that returns the n th prime number, (3) use κ to rewrite **filter** in a way that it passes syntactic tests of productivity, i.e., using one of the above syntactic schemes [Niq07c, Niq04, § 4.7].

Another work-around, one that we follow in this article, is to adhere to advanced type-theoretic methods to bypass this condition. This is similar to the application of dependent inductive types for formalising general recursion using structural recursion [DDG98, BC01]. For coinductive types this has led to a method of general corecursion for filter-like functions [Ber05b] and a similar method that we use in Section 3 for formalising the homographic and quadratic algorithms in *Coq*.

The **cofix** operator and its expansion rule together with the guardedness condition constitute the machinery of the *Coq* system for coinductive types. This means that there is no separate tool for proofs by *coinduction*. This is in contrast to the set-theoretic greatest fixed point semantics for coinduction where for each coinductive object a coinduction proof principle is present which is inherent in the monotonicity of the set operator [BM96]. Instead in the type theoretic approach, where proofs are objects too, we use the **cofix** operator to directly *build* the coinductive proof as a proof object. This means that whenever we want to prove by coinduction, our goal should be a coinductive type. If necessary, specialised coinductive predicates should be created for formalising a proof that uses coinduction. These additional predicates are in most cases straightforward reformulation of the corresponding set-theoretic proof principle (cf. the extensional equality \cong below). However, sometimes special care has to be taken to overcome the restrictions put forward by the guardedness condition (cf. **rep** in Section 4). As a result, *Coq*'s direct approach to coinduction makes the coinductive proofs easier than their set-theoretic counterparts as long the guardedness condition does not get in the way.

For proving equalities by coinduction, in coalgebraic and set-theoretic settings one relies on the notion of *bisimulation* [BM96, JR97]. In the case of streams, a bisimulation is

a binary relation R satisfying the property that

$$R(\alpha, \beta) \implies \text{hd}(\alpha) = \text{hd}(\beta) \wedge R(\text{tl}(\alpha), \text{tl}(\beta)) .$$

Here hd and tl are functions on streams that give the head and the tail of a stream. Then one can prove that two streams are equal if they satisfy a bisimulation relation. The coinduction proof principle thus consists of finding a suitable bisimulation.

To translate this proof principle into the type theoretic coinduction note that the bisimulation relation leads to the extensional equality, which in the *intensional* type theories, such as CIC, is quite distinct from the built-in notion of equality. In fact each extensional equality should be defined and added to the type system. On the other hand, recall that we can only prove by coinduction in *Coq* if the goal of the proof has a coinductive type. This leads us to the following definition for a coinductive extensional equality on streams which we denote by \cong .

$$\begin{array}{l} \text{CoInductive } \cong : A^\omega \rightarrow A^\omega \rightarrow \mathbf{Prop} := \\ | \cong_c : \forall (\alpha_1 \alpha_2 : A^\omega), \text{hd } \alpha_1 = \text{hd } \alpha_2 \rightarrow \text{tl } \alpha_1 \cong \text{tl } \alpha_2 \rightarrow \alpha_1 \cong \alpha_2 . \end{array}$$

Note that \cong_c , the sole constructor of \cong , has the shape of a bisimulation property. The proof that this is an equivalence relation can be found in the standard library of *Coq* [CDT06]. Moreover, Giménez shows that this is a bisimulation equivalence relation and derives the usual principle of coinduction [Gim96, § 4.2]. In the present work we use \cong relation in our coinductive correctness proofs.

2. HOMOGRAPHIC AND QUADRATIC ALGORITHMS

The homographic and quadratic algorithms are similar to Gosper's algorithm [Gos72] for addition and multiplication on continued fractions and form the basis of the Edalat-Potts approach to lazy exact real arithmetic [EP97, Pot98].

Here we use a representation which is much simpler than the continued fractions but it is redundant enough to ensure productivity³. There is nothing special about this representation apart from the fact that it eases the *Coq* formalisation of the proofs of the metric properties that we use in this work, thus giving us a prototype formalisation of the algorithms that is concrete and hence can be computed with. A treatment of the general case where we abstract away both the digit set and the compact subinterval of $[-\infty, +\infty]$ can be found in [Niq04, § 5]. Thus, for practical and presentational purposes, we consider a fixed representation for $[-1, 1]$ containing 3 digits, each of which a Möbius map. *Möbius maps* are maps of the form

$$x \longmapsto \frac{ax + b}{cx + d} ,$$

where⁴ $a, b, c, d \in \mathbb{Q}$. Möbius maps are usually denoted by the matrix of their coefficients. A Möbius map is *bounded* if its denominator does not vanish in $[-1, 1]$. A Möbius map is

³The necessity of redundancy in the representations for real numbers is studied in the framework of computable analysis [Wei00] but is outside the scope of the present article.

⁴Note that we could equivalently take the coefficients to be in \mathbb{Z} .

refining if it maps the closed interval $[-1, 1]$ into itself. Assuming $\mu = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ we introduce two predicates that capture these properties:

$$\begin{aligned} \mathbf{Bounded}(\mu) &:= 0 < d+c \wedge 0 < d-c \vee d+c < 0 \wedge d-c < 0 \text{ ,} \\ \mathbf{Ref}(\mu) &:= \mathbf{Bounded}(\mu) \wedge \\ &\quad (0 < a+b+c+d \wedge 0 < a-b-c+d \wedge 0 < -a-b+c+d \wedge 0 < -a+b-c+d \vee \\ &\quad a+b+c+d < 0 \wedge a-b-c+d < 0 \wedge -a-b+c+d < 0 \wedge -a+b-c+d < 0) \text{ .} \end{aligned}$$

For our representation, we consider the set $\mathbf{DIG} = \{\mathbf{L}, \mathbf{R}, \mathbf{M}\}$ and denote the set of streams of elements of \mathbf{DIG} by \mathbf{DIG}^ω . We interpret each digit by a refining Möbius map as follows.

$$\mathbf{L} = \begin{bmatrix} \frac{1}{2} & \frac{-1}{2} \\ \frac{1}{2} & \frac{3}{2} \end{bmatrix} \text{ , } \mathbf{R} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{-1}{2} & \frac{3}{2} \end{bmatrix} \text{ , } \mathbf{M} = \begin{bmatrix} 1 & 0 \\ 0 & 3 \end{bmatrix} \text{ .}$$

In fact under the conjugacy map $S(x) = \frac{x-1}{x+1}$, these are the conjugates of the Stern–Brocot representation for $[0, +\infty]$ presented in [Niq04, § 5.7], hence the fact that \mathbf{DIG}^ω is a representation for $[-1, 1]$ is easily derivable from the properties of the Stern–Brocot representation (see also Section 4).

The *homographic algorithm* is the algorithm that given a Möbius map μ and a stream $\alpha \in \mathbf{DIG}^\omega$ representing r_α , outputs a stream γ that represents r_γ such that $\mu(r_\alpha) = r_\gamma$. In order to present the homographic algorithm we need an *emission condition* $\mathbf{Incl}(\mu, \phi)$ for a digit $\phi = \begin{bmatrix} \phi_{00} & \phi_{01} \\ \phi_{10} & \phi_{11} \end{bmatrix}$ and μ which checks the inclusion of intervals $\mu([-1, 1]) \subseteq \phi([-1, 1])$.

$$\begin{aligned} \mathbf{Incl}(\mu, \phi) &:= \mathbf{Bounded}(\mu) \wedge (d-c)(d-c)(\phi_{01}-\phi_{00}) \leq (d-c)(b-a)(\phi_{11}-\phi_{10}) \wedge \\ &\quad (d-c)(b-a)(\phi_{10}+\phi_{11}) \leq (d-c)(d-c)(\phi_{00}+\phi_{01}) \wedge \\ &\quad (d+c)(c+d)(\phi_{01}-\phi_{00}) \leq (d+c)(a+b)(\phi_{11}-\phi_{10}) \wedge \\ &\quad (d+c)(a+b)(\phi_{10}+\phi_{11}) \leq (d+c)(c+d)(\phi_{00}+\phi_{01}) \text{ .} \end{aligned}$$

Note that since the above are expressions involving only rational numbers the emission condition is a decidable predicate. This enables us to state the homographic algorithm:

homographic $\mu \ (x :: xs) :=$

$$\left\{ \begin{array}{ll} \mathbf{L} :: \text{homographic } (\mathbf{L}^{-1} \circ \mu) \ (x :: xs) & \mathbf{if } \mathbf{Incl}(\mu, \mathbf{L}) \text{ ,} \\ \mathbf{R} :: \text{homographic } (\mathbf{R}^{-1} \circ \mu) \ (x :: xs) & \mathbf{else if } \mathbf{Incl}(\mu, \mathbf{R}) \text{ ,} \\ \mathbf{M} :: \text{homographic } (\mathbf{M}^{-1} \circ \mu) \ (x :: xs) & \mathbf{else if } \mathbf{Incl}(\mu, \mathbf{M}) \text{ ,} \\ \text{homographic } \mu \circ x \ xs & \mathbf{otherwise.} \end{array} \right.$$

Here d^{-1} and \circ denote the usual matrix inversion and matrix product. The first three branches (resp. the last branch) are called *emission steps* (resp. *absorption step*). Note that due to the redundancy of the representation, the case distinction need not be mutually exclusive, but this does not affect the outcome.

The intuition behind the algorithm is that we start by considering an infinite product of Möbius maps, of which all but the first one are digits. We start pushing μ towards the infinity by absorbing digits (hence obtaining a new refining Möbius map) and emitting digits whenever the emission condition holds, i.e., whenever the range of Möbius map applied to

the interval $[-1, 1]$ fits inside the range of a digit.

$$\mu \circ \phi_0 \circ \phi_1 \circ \dots \quad \rightsquigarrow \quad \phi \circ (\phi^{-1} \circ \mu) \circ \phi_0 \circ \phi_1 \circ \dots \quad \text{if } \mathbf{Incl}(\mu, \phi) \text{ .}$$

For a more formal semantics for the algorithm see [PEE97] and the semantical proof of correctness that is given in [Niq04, § 5.6].

To compute binary algebraic operations we consider the *quadratic map* which is a map

$$\xi(x, y) := \frac{axy + bx + cy + d}{exy + fx + gy + h} \text{ ,}$$

with $a, b, c, d, e, f, g \in \mathbb{Q}$ and can be denoted by its $2 \times 2 \times 2$ tensor of coefficients. A quadratic map is *bounded* if its denominator does not vanish in $[-1, 1] \times [-1, 1]$. A *refining* quadratic map is a quadratic map ξ such that $\xi([-1, 1], [-1, 1]) \subseteq [-1, 1]$. The predicates **Bounded**(ξ) and **Ref**(ξ) can easily be stated in terms of inequalities on rational numbers (see Appendix A).

The *quadratic algorithm* is an algorithm that given a quadratic map ξ and two streams $\alpha, \beta \in \mathbf{DIG}^\omega$ representing r_α and r_β , outputs a stream γ that represents r_γ such that $\xi(r_\alpha, r_\beta) = r_\gamma$. Here too we need a decidable emission condition **Incl**(ξ, ϕ) that checks the inclusion of intervals $\xi([-1, 1], [-1, 1]) \subseteq \phi([-1, 1])$ for each digit ϕ ; its explicit definition is given in Appendix A. By $\mu \circ \xi$ we denote the composition of a Möbius map μ and a quadratic map ξ (note that the outcome is again a quadratic map). Moreover we use $\xi \bullet_1 \mu$ and $\xi \bullet_2 \mu$ to denote the two different ways of composing a quadratic map and a Möbius map by considering the Möbius map as its first (resp. second) argument. With this notation we can present the quadratic algorithm:

$$\begin{aligned} \text{quadratic } \xi \text{ (} x :: xs \text{) (} y :: ys \text{) :=} \\ \left\{ \begin{array}{ll} \mathbf{L} :: \text{quadratic } (\mathbf{L}^{-1} \circ \xi) \text{ (} x :: xs \text{) (} y :: ys \text{)} & \mathbf{if } \mathbf{Incl}(\xi, \mathbf{L}) \text{ ,} \\ \mathbf{R} :: \text{quadratic } (\mathbf{R}^{-1} \circ \xi) \text{ (} x :: xs \text{) (} y :: ys \text{)} & \mathbf{else if } \mathbf{Incl}(\xi, \mathbf{R}) \text{ ,} \\ \mathbf{M} :: \text{quadratic } (\mathbf{M}^{-1} \circ \xi) \text{ (} x :: xs \text{) (} y :: ys \text{)} & \mathbf{else if } \mathbf{Incl}(\xi, \mathbf{M}) \text{ ,} \\ \text{quadratic } (\xi \bullet_1 x \bullet_2 y) \text{ } xs \text{ } ys & \mathbf{otherwise.} \end{array} \right. \end{aligned}$$

The intuition behind this algorithm is similar to the homographic algorithm. The homographic algorithm can be used to compute the unary field operation of opposite, while the quadratic algorithm can be used for binary field operations of addition, multiplication and division. Simply taking $\xi := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ it gives the multiplication. Note that addition and division are not total functions on $[-1, 1]$. The quadratic algorithm applied to $\xi := \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ is a partial function that will calculate the addition (and also it is productive) if and only if the inputs add up to a number within $[-1, 1]$. However, the algorithms can also be used to calculate the binary *average* function and the restricted division that are defined in [CDG06].

In the present work we do not study the total version of field operations and computations on the whole real line. However, we mention that transferring the computation to the whole real line is possible. A possibility would be to first move to $[0, +\infty]$ via the inverse of the above conjugacy map. From here we can follow [Pot98] where a redundant sign bit is added by considering a fourth order elliptic Möbius map that leads to a cyclic group consisting of four signs for an unbiased exact floating point [Pot98, §9.2].

3. GENERAL CORECURSIVE VERSION OF THE ALGORITHMS

Algorithms of the previous section specify partial functions into the coinductive type of streams. This partiality is problematic for us. Translating these specifications into the language of *Coq* means that we should ensure that the returned value is provably an infinite stream, which is obviously not always true for a partial function. The algorithms resemble the general shape of the filter algorithm (see Section 1). Hence, as expected, they do not satisfy the guardedness test of *Coq*, and indeed any other one of the syntactic schemes used in the theory of coalgebras.

The usual way of dealing with partial functions in type theory is to consider them as total functions but on a new, restricted domain which corresponds to the values on which the partial function is defined. In our case, it is well-known [Pot98] that the algorithms are productive if they are applied with refining maps. Proof of this fact is a tedious semantic proof that deals with rational intervals. Thus we need to incorporate among the arguments of the function an additional argument, a so called *proof obligation*, that captures the property of being refining and hence the semantic proof of the infinitude of the outcome. But directly adding the refining property **Ref**, does not give us enough type theoretic machinery because **Ref** is just a simple propositional predicate that lacks any inductive or coinductive structure.

Instead our proposed proof obligation will have a more complex shape, enabling us to use type theoretic tools of structural recursion and coinduction. Later in Section 6 we show that our proposed proof obligation is a consequence of **Ref**. Instead of relying on properties of interval inclusion our predicate will rely on the intuitive idea of having an infinite output. Such a proof obligation is satisfied if at every step in the algorithm after absorbing a finite number of digits the emission condition eventually holds and hence we can output a digit. We plan to capture this inside a recursive function that at each step outputs the next digit, serving as a modulus for productivity. The original algorithms will then call this function at every step to obtain the next digit while keeping track of the new arguments that should be passed to future step. This idea is used by Bertot [Ber05b] to give a general method for defining *filter* in *Coq*. In this section we apply a modification of Bertot's method for our algorithms of exact arithmetic.

3.1. Homographic Algorithm. Let \mathbb{M} (resp. \mathbb{T}) be the set of Möbius maps (resp. quadratic maps) in *Coq*⁵. We are seeking to define a map $h: \mathbb{M} \times \mathbf{DIG}^\omega \rightarrow \mathbf{DIG}^\omega$, that corresponds to the homographic algorithm. But h is a partial function and is not productive at every point. So instead of defining h we shall define a map

$$\bar{h}: \Pi(\mu: \mathbb{M})(\alpha: \mathbf{DIG}^\omega). P_h(\mu, \alpha) \rightarrow \mathbf{DIG}^\omega$$

where $P_h(\mu, \alpha)$ is a predicate (i.e., a term of type **Prop**) with the intended meaning that the specification of the homographic algorithm is productive when applied to μ and α . In other words it specifies the domain of the partial function h . We shall call P_h a *productivity predicate*.

The definition of P_h is based on the modulus of productivity. This modulus is a recursive function

$$m_h: \mathbb{M} \times \mathbf{DIG}^\omega \rightarrow \mathbf{DIG} \times \mathbb{M} \times \mathbf{DIG}^\omega$$

⁵They can be considered as \mathbb{Q}^4 and \mathbb{Q}^8 respectively, forgetting about the refining and nonsingular properties. Those properties will enter the picture when we study the correctness of the algorithms.

with the intended meaning that $m_h(\mu, \alpha) = \langle \phi, \langle \mu', \alpha' \rangle \rangle$ if and only if

$$\text{homographic } \mu \ \alpha \quad \rightsquigarrow \quad \phi :: \text{homographic } \mu' \ \alpha' ,$$

where ‘ \rightsquigarrow ’ denotes multiple reduction steps after which ϕ is output (so after output of ϕ there are no more digits absorbed in μ'). We would like this to be a function with recursive calls on α , but this is not possible. The reason is that α has a coinductive type while in the structural recursion scheme we need an element with an inductive type. In other words we need to accommodate the domain of the function m_h with an inductively defined argument which will be used for recursive calls.

This situation is similar to the case of partial recursive functions or recursive functions with non-structurally recursive arguments. In order to formalise such function in constructive type theory, there is a method of adding an inductive domain predicate introduced in [DDG98] and extensively developed by Bove and Capretta [BC01]. According to this method we need to define an inductively defined predicate $E_h(\mu, \alpha)$ with the intended meaning that μ and α are in the domain of m_h which in turn means that the homographic algorithm should emit at least one digit when applied on μ and α . Thus, as a first step in the definition of the productivity predicate, we define E_h as the following inductive type.

```

Inductive Eh:  $\mathbb{M} \rightarrow \mathbf{DIG}^\omega \rightarrow \mathbf{Prop} :=
| E_{hL}: \forall (\mu: \mathbb{M}) (\alpha: \mathbf{DIG}^\omega), \mathbf{Incl}(\mu, \mathbf{L}) \rightarrow E_h \ \mu \ \alpha
| E_{hR}: \forall (\mu: \mathbb{M}) (\alpha: \mathbf{DIG}^\omega), \mathbf{Incl}(\mu, \mathbf{R}) \rightarrow E_h \ \mu \ \alpha
| E_{hM}: \forall (\mu: \mathbb{M}) (\alpha: \mathbf{DIG}^\omega), \mathbf{Incl}(\mu, \mathbf{M}) \rightarrow E_h \ \mu \ \alpha
| E_{hab}: \forall (\mu: \mathbb{M}) (\alpha: \mathbf{DIG}^\omega), E_h \ (\mu \circ (\text{hd } \alpha)) \ (\text{tl } \alpha) \rightarrow E_h \ \mu \ \alpha.$ 
```

Here E_{hL} , E_{hR} , E_{hM} and E_{hab} are constructors of E_h . Note that E_h has one constructor for each branch of the homographic algorithm.

This allows us to define the modulus of productivity, i.e., a recursive function

$$\overline{m}_h: \Pi(\mu: \mathbb{M})(\alpha: \mathbf{DIG}^\omega). E_h(\mu, \alpha) \longrightarrow \mathbf{DIG} \times \mathbb{M} \times \mathbf{DIG}^\omega$$

as follows.

```

Fixpoint  $\overline{m}_h(\mu: \mathbb{M})(\alpha: \mathbf{DIG}^\omega) (t: E_h \ \mu \ \alpha) \{\mathbf{struct} \ t\}: \mathbf{DIG} * (\mathbb{M} * \mathbf{DIG}^\omega) :=
match  $\mathbf{Incl}_{\text{dec}}(\mu, \mathbf{L})$  with
| left  $_ \Rightarrow \langle \mathbf{L}, \langle \mathbf{L}^{-1} \circ \mu, \alpha \rangle \rangle$ 
| right  $t_l \Rightarrow$ 
  match  $\mathbf{Incl}_{\text{dec}}(\mu, \mathbf{R})$  with
  | left  $_ \Rightarrow \langle \mathbf{R}, \langle \mathbf{R}^{-1} \circ \mu, \alpha \rangle \rangle$ 
  | right  $t_r \Rightarrow$ 
    match  $\mathbf{Incl}_{\text{dec}}(\mu, \mathbf{M})$  with
    | left  $_ \Rightarrow \langle \mathbf{M}, \langle \mathbf{M}^{-1} \circ \mu, \alpha \rangle \rangle$ 
    | right  $t_m \Rightarrow \overline{m}_h \ (\mu \circ (\text{hd } \alpha)) \ (\text{tl } \alpha) \ (E_{hab\text{-inv}} \ \mu \ \alpha \ t_l \ t_r \ t_m \ t)$ 
    end
  end
end
end.$ 
```

Here **Fixpoint** (resp. **struct**) are *Coq* keywords to denote a recursive definition (resp. recursive argument of structural recursive calls). Moreover, in the body of the definition

two terms $\mathbf{Incl}_{\text{dec}}$ and $E_{hab\text{-inv}}$ are used. Both terms can be proven as lemmas in *Coq*. The first lemma is the following.

Lemma $\mathbf{Incl}_{\text{dec}}$: $\forall (\mu : \mathbb{M}) (\phi : \mathbf{DIG}), \mathbf{Incl}(\mu, \phi) \oplus \neg \mathbf{Incl}(\mu, \phi)$.

This term extracts the informative computational content of the predicate \mathbf{Incl} which is a term of the type \mathbf{Prop} . This is necessary because in CIC one cannot obtain elements of the type \mathbf{Set} by pattern matching on propositions. Thus we have to use $\oplus : \mathbf{Prop} \times \mathbf{Prop} \rightarrow \mathbf{Set}$ — with \mathbf{left} and \mathbf{right} its coprojections — to transfer propositions into a boolean sum on which we can pattern match. Hence the need for the above lemma is inevitable, although its proof is quite trivial.

The second lemma states an inverse of the last constructor of E_{hab} in case no emission condition holds.

Lemma $E_{hab\text{-inv}}$: $\forall (\mu : \mathbb{M}) (\alpha : \mathbf{DIG}^\omega),$
 $\neg \mathbf{Incl}(\mu, \mathbf{L}) \rightarrow \neg \mathbf{Incl}(\mu, \mathbf{R}) \rightarrow \neg \mathbf{Incl}(\mu, \mathbf{M}) \rightarrow E_h \mu \alpha \rightarrow$
 $E_h (\mu \circ (\mathbf{hd} \alpha)) (\mathbf{tl} \alpha)$.

This lemma can be proven because E_h is an inductive type and hence all its canonical objects should be generated by one of its constructors⁶

Note that in \overline{m}_h the output is independent of the proof t . The term t only serves as a catalyst that allows for using recursion where all the other arguments are not inductive. Thus we should be able to prove a *proof irrelevance* result for \overline{m}_h .

Lemma 3.1. *Let $\mu \in \mathbb{M}, \alpha \in \mathbf{DIG}^\omega$ and t_1, t_2 be two proofs that $E_h(\mu, \alpha)$ holds. Then*

$$\overline{m}_h(\mu, \alpha, t_1) = \overline{m}_h(\mu, \alpha, t_2) \quad . \quad \square$$

The proof of the above lemma is based on a dependent induction scheme for E_h that is more specialised than the usual induction scheme attributed to the inductive types: the ordinary induction scheme can be used to prove a property $R : \mathbb{M} \rightarrow \mathbf{DIG}^\omega \rightarrow \mathbf{Prop}$ while the dependent induction scheme can be used to prove a property

$$R : \Pi(\mu : \mathbb{M})(\alpha : \mathbf{DIG}^\omega). E_h(\mu, \alpha) \rightarrow \mathbf{Prop} \quad .$$

The Lemma 3.1 enables us to prove the *fixed point equations* of the \overline{m}_h function. These are in fact unfolding of the body of the definition of \overline{m}_h ; they are crucial for proving similar results for the homographic algorithm. Hence we mention them in a lemma here:

Lemma 3.2. *Let $\mu \in \mathbb{M}, \alpha \in \mathbf{DIG}^\omega$ and t be a proof that $E_h(\mu, \alpha)$ holds.*

(1) *If $\mathbf{Incl}(\mu, \mathbf{L})$ holds then*

$$\overline{m}_h(\mu, \alpha, t) = \langle \mathbf{L}, \langle \mathbf{L}^{-1} \circ \mu, \alpha \rangle \rangle \quad .$$

(2) *If $\neg \mathbf{Incl}(\mu, \mathbf{L})$ but $\mathbf{Incl}(\mu, \mathbf{R})$ holds then*

$$\overline{m}_h(\mu, \alpha, t) = \langle \mathbf{R}, \langle \mathbf{R}^{-1} \circ \mu, \alpha \rangle \rangle \quad .$$

⁶Due to some technical issues with respect to the type theory of *Coq*, the proof has to be built using a specific method that is described in details in [BC04, §15.4]. These issues are out of the scope of the present work.

(3) If $\neg \mathbf{Incl}(\mu, \mathbf{L})$ and $\neg \mathbf{Incl}(\mu, \mathbf{R})$ but $\mathbf{Incl}(\mu, \mathbf{M})$ holds then

$$\overline{m}_h(\mu, \alpha, t) = \langle \mathbf{M}, \langle \mathbf{M}^{-1} \circ \mu, \alpha \rangle \rangle .$$

(4) If $\neg \mathbf{Incl}(\mu, \mathbf{L})$, $\neg \mathbf{Incl}(\mu, \mathbf{R})$ and $\neg \mathbf{Incl}(\mu, \mathbf{M})$ holds then for all t' a proof of property $E_h(\mu \circ (\text{hd}(\alpha)), \text{tl}(\alpha))$ we have

$$\overline{m}_h(\mu, \alpha, t) = \overline{m}_h(\mu \circ (\text{hd}(\alpha)), \text{tl}(\alpha), t') . \quad \square$$

Note that the last part states a more general fact than just the fourth branch of the recursive definition of \overline{m}_h because the proof obligation t' is abstracted. Nevertheless its proof is similar to the other three parts.

Having defined \overline{m}_h we need one more auxiliary predicate before defining P_h . This auxiliary predicate is an inductive predicate that ensures that E_h holds for some finite iteration of \overline{m}_h (here π_{ij} denotes the i -th projection of a j -tuple).

Inductive $\Psi_h: \mathbb{N} \rightarrow \mathbb{M} \rightarrow \mathbf{DIG}^\omega \rightarrow \mathbf{Prop} :=$
 $|\Psi_{h0}: \forall(\mu: \mathbb{M})(\alpha: \mathbf{DIG}^\omega), E_h \mu \alpha \rightarrow \Psi_h 0 \mu \alpha$
 $|\Psi_{hS}: \forall(n: \mathbb{N})(\mu: \mathbb{M})(\alpha: \mathbf{DIG}^\omega)(t: E_h \mu \alpha),$
 $\Psi_h n (\pi_{23}(\overline{m}_h \mu \alpha t)) (\pi_{33}(\overline{m}_h \mu \alpha t)) \rightarrow \Psi_h (n+1) \mu \alpha.$

We use the above predicate to define P_h , a predicate that captures the productivity of the homographic algorithm. This predicate will be an inductive type with one constructor.

Inductive $P_h: \mathbb{M} \rightarrow \mathbf{DIG}^\omega \rightarrow \mathbf{Prop} :=$
 $|P_{hab}: \forall(\mu: \mathbb{M})(\alpha: \mathbf{DIG}^\omega), (\forall(n: \mathbb{N}), \Psi_h (n+1) \mu \alpha) \rightarrow P_h \mu \alpha.$

The sole constructor of this type ensures that after each emission, which occurs because of E_h , the new Möbius map passed to the homographic algorithm results in a new emission. This fact is implicit in the following two properties of P_h that are needed in the definition of the homographic algorithm. First lemma states the relation between P_h and E_h :

Lemma $P_h \text{-} E_h: \forall(\mu: \mathbb{M})(\alpha: \mathbf{DIG}^\omega), P_h \mu \alpha \rightarrow E_h \mu \alpha.$

The second lemma relates \overline{m}_h and P_h , and shows that P_h is indeed passed to the future arguments.

Lemma $\overline{m}_h \text{-} P_h: \forall(\mu: \mathbb{M})(\alpha: \mathbf{DIG}^\omega)(t: E_h \mu \alpha),$
 $\text{let } \mu' := \pi_{23}(\overline{m}_h \mu \alpha t) \text{ in let } \alpha' := \pi_{33}(\overline{m}_h \mu \alpha t) \text{ in}$
 $P_h \mu \alpha \rightarrow P_h \mu' \alpha'.$

The proof of both of the above lemmas is based on the inverse of the constructors of Ψ_h , namely the following lemma which in turn is a consequence of the Lemma 3.1.

Lemma 3.3. *For all n let $\mu \in \mathbb{M} \alpha \in \mathbf{DIG}^\omega$.*

(1) *If $\Psi_h(n, \mu, \alpha)$ holds then $E_h(\mu, \alpha)$ holds.*

(2) Let t be a proof that $E_h(\mu, \alpha)$ holds. Then if $\Psi_h(n+1, \mu, \alpha)$ holds then

$$\Psi_h(n, \pi_{23}(\overline{m}_h(\mu, \alpha, t)), \pi_{33}(\overline{m}_h(\mu, \alpha, t))) . \quad \square$$

Finally we are ready to define the homographic algorithm as a function

$$\bar{h}: \Pi(\mu: \mathbb{M})(\alpha: \mathbf{DIG}^\omega). P_h(\mu, \alpha) \longrightarrow \mathbf{DIG}^\omega$$

that accommodates the proof of its own productivity as one of its arguments. Here the *Coq* keyword **CoFixpoint** denotes that we are using the **cofix** rule (see Section 1).

$$\begin{array}{l} \mathbf{CoFixpoint} \ \bar{h} \ (\mu: \mathbb{M}) \ (\alpha: \mathbf{DIG}^\omega) \ (p: P_h \ \mu \ \alpha) : \mathbf{DIG}^\omega := \\ \mathbf{Cons} \ \pi_{13}(\overline{m}_h \ \mu \ \alpha \ (P_h \text{-} E_h \ \mu \ \alpha \ p)) \\ \quad (\bar{h} \ \pi_{23}(\overline{m}_h \ \mu \ \alpha \ (P_h \text{-} E_h \ \mu \ \alpha \ p)) \\ \quad \quad \pi_{33}(\overline{m}_h \ \mu \ \alpha \ (P_h \text{-} E_h \ \mu \ \alpha \ p)) \\ \quad \quad (\overline{m}_h \text{-} P_h \ \mu \ \alpha \ (P_h \text{-} E_h \ \mu \ \alpha \ p) \ p)) . \end{array}$$

This definition passes the guardedness condition of *Coq*. Thus we have tackled the problem of productivity by changing the function domain and adding a proof obligation.

3.2. Cofixed Point Equations. Next we show that \bar{h} satisfies the specification of the homographic algorithm. At this point we need to use the extensional equality \cong on streams to prove an extensional proof irrelevance for \bar{h} . The proof of this lemma uses Lemma 3.1.

Lemma 3.4. *Let $\mu \in \mathbb{M}, \alpha \in \mathbf{DIG}^\omega$ and p, p' be two proofs that $P_h(\mu, \alpha)$ holds. Then the observable outcome of \bar{h} is independent of p and p' , i.e.,*

$$\bar{h}(\mu, \alpha, p) \cong \bar{h}(\mu, \alpha, p') . \quad \square$$

Subsequently, we use the above lemma together with Lemma 3.2 to prove that \bar{h} satisfies the specification of the homographic algorithm. We call these the *cofixed point equations* of the homographic algorithm because they can be considered as the dual of the fixed point equations for recursive functions.

Lemma 3.5. *Let $\mu \in \mathbb{M}, \alpha \in \mathbf{DIG}^\omega$ and p be a proof that $P_h(\mu, \alpha)$ holds.*

(1) If **Incl**(μ, \mathbf{L}) holds then

$$\bar{h}(\mu, \alpha, p) \cong \mathbf{Cons} \ \mathbf{L} \ \bar{h}(\mathbf{L}^{-1} \circ \mu, \alpha) .$$

(2) If $\neg \mathbf{Incl}(\mu, \mathbf{L})$ but **Incl**(μ, \mathbf{R}) holds then

$$\bar{h}(\mu, \alpha, p) \cong \mathbf{Cons} \ \mathbf{R} \ \bar{h}(\mathbf{R}^{-1} \circ \mu, \alpha) .$$

(3) If $\neg \mathbf{Incl}(\mu, \mathbf{L})$ and $\neg \mathbf{Incl}(\mu, \mathbf{R})$ but **Incl**(μ, \mathbf{M}) holds then

$$\bar{h}(\mu, \alpha, p) \cong \mathbf{Cons} \ \mathbf{M} \ \bar{h}(\mathbf{M}^{-1} \circ \mu, \alpha) .$$

(4) If $\neg \mathbf{Incl}(\mu, \mathbf{L})$, $\neg \mathbf{Incl}(\mu, \mathbf{R})$ and $\neg \mathbf{Incl}(\mu, \mathbf{M})$ holds then for all p' a proof of property $P_h(\mu \circ (\text{hd}(\alpha)), \text{tl}(\alpha))$ we have

$$\bar{h}(\mu, \alpha, p) \cong \bar{h}(\mu \circ (\text{hd}(\alpha)), \text{tl}(\alpha), p') . \quad \square$$

Hence we have shown that our function \bar{h} satisfies the specification of Section 2 and is indeed a formalisation of the homographic algorithm.

So far we have only tackled the formalisation of the homographic algorithm as a productive coinductive map, and *not* its correctness. As we stated earlier the above algorithm (without enforcing any condition on μ) is not always productive for non-refining Möbius maps. It is important to have in mind that we have separated the issue of productivity and correctness. This is in accordance with separation of *termination* and correctness in the method of Bove–Capretta for general recursion [BC01] or already in the Hoare logic. Moreover, this separation is also evident in the domain theoretic semantics of the real numbers [MRE07].

In order to prove the correctness we need to define a suitable semantics (for example use another model of real numbers) and prove that the effect of the above algorithm, when applied with a refining Möbius map, is equivalent to the effect of that Möbius map in $[-1, 1]$. This will be done in Sections 5–6.

3.3. Quadratic Algorithm. In the case of the quadratic algorithm we follow the same method that we used for the homographic algorithm. We start by defining the inductive type for the domain of the modulus function.

```

Inductive  $E_q : \mathbb{T} \rightarrow \mathbf{DIG}^\omega \rightarrow \mathbf{DIG}^\omega \rightarrow \mathbf{Prop} :=
| E_{qL} : \forall (\xi : \mathbb{T}) (\alpha \beta : \mathbf{DIG}^\omega), \mathbf{Incl}(\xi, \mathbf{L}) \rightarrow E_q \xi \alpha \beta
| E_{qR} : \forall (\xi : \mathbb{T}) (\alpha \beta : \mathbf{DIG}^\omega), \mathbf{Incl}(\xi, \mathbf{R}) \rightarrow E_q \xi \alpha \beta
| E_{qM} : \forall (\xi : \mathbb{T}) (\alpha \beta : \mathbf{DIG}^\omega), \mathbf{Incl}(\xi, \mathbf{M}) \rightarrow E_q \xi \alpha \beta
| E_{qab} : \forall (\xi : \mathbb{T}) (\alpha \beta : \mathbf{DIG}^\omega),
      E_q (\xi \bullet_1(\mathbf{hd} \alpha)) \bullet_2(\mathbf{hd} \beta) (\mathbf{tl} \alpha) (\mathbf{tl} \beta) \rightarrow E_q \xi \alpha \beta.$ 
```

Using this we define the modulus function by structural recursion on a term of the above type. Note that in this case the modulus function \bar{m}_q returns a quadruple $\langle \phi, \langle \xi', \langle \alpha', \beta' \rangle \rangle \rangle$ consisting of the emitted digit, the new quadratic map passed to the continuation of the quadratic algorithm and the remainder (unabsorbed part) of two the streams of digits.

```

Fixpoint  $\bar{m}_q (\xi : \mathbb{T}) (\alpha \beta : \mathbf{DIG}^\omega) (t : E_q \xi \alpha \beta) \{ \mathbf{struct} \ t \}
: \mathbf{DIG}^*(\mathbb{T}^*(\mathbf{DIG}^\omega * \mathbf{DIG}^\omega)) :=
match  $\mathbf{Incl}_{\text{dec}}(\xi, \mathbf{L})$  with
| left  $_ \Rightarrow \langle \mathbf{L}, \langle \mathbf{L}^{-1} \circ \xi, \langle \alpha, \beta \rangle \rangle \rangle$ 
| right  $t_l \Rightarrow
  match  $\mathbf{Incl}_{\text{dec}}(\xi, \mathbf{R})$  with
  | left  $_ \Rightarrow \langle \mathbf{L}, \langle \mathbf{R}^{-1} \circ \xi, \langle \alpha, \beta \rangle \rangle \rangle$ 
  | right  $t_r \Rightarrow
    match  $\mathbf{Incl}_{\text{dec}}(\xi, \mathbf{M})$  with
    | left  $_ \Rightarrow \langle \mathbf{L}, \langle \mathbf{M}^{-1} \circ \xi, \langle \alpha, \beta \rangle \rangle \rangle$ 
    | right  $t_m \Rightarrow \bar{m}_q (\xi \bullet_1(\mathbf{hd} \alpha)) \bullet_2(\mathbf{hd} \beta) (\mathbf{tl} \alpha) (\mathbf{tl} \beta)
      (E_{qab\text{-inv}} \xi \alpha \beta t_l t_r t_m t)
    end
  end
end
end.$$$$ 
```

Here E_{qab_inv} is an inverse of the last constructor of the inductive type E_q akin to E_{hab_inv} for the homographic algorithm. Furthermore we have to prove the proof irrelevance and the fixed point equations for \overline{m}_q . For brevity we do not mention them here but their statement and proofs can be found in [Niq07a].

Next we define the inductive predicate Ψ_q that ensures the validity of E_q for finite iterations of \overline{m}_q :

```

Inductive  $\Psi_q : \mathbb{N} \rightarrow \mathbb{T} \rightarrow \mathbf{DIG}^\omega \rightarrow \mathbf{DIG}^\omega \rightarrow \mathbf{Prop} :=
| \Psi_{q0} : \forall (\xi : \mathbb{T}) (\alpha \beta : \mathbf{DIG}^\omega), E_q \xi \alpha \beta \rightarrow \Psi_q 0 \xi \alpha \beta
| \Psi_{qS} : \forall (n : \mathbb{N}) (\xi : \mathbb{T}) (\alpha \beta : \mathbf{DIG}^\omega) (t : E_q \xi \alpha \beta),
  \Psi_q n (\pi_{24}(\overline{m}_q \xi \alpha \beta t)) (\pi_{34}(\overline{m}_q \xi \alpha \beta t)) (\pi_{44}(\overline{m}_q \xi \alpha \beta t)) \rightarrow
  \Psi_q (n+1) \xi \alpha \beta.$ 
```

This allows us to define the productivity predicate P_q :

```

Inductive  $P_q : \mathbb{T} \rightarrow \mathbf{DIG}^\omega \rightarrow \mathbf{DIG}^\omega \rightarrow \mathbf{Prop} :=
| P_{qab} : \forall (\xi : \mathbb{T}) (\alpha \beta : \mathbf{DIG}^\omega), (\forall (n : \mathbb{N}), \Psi_q n \xi \alpha \beta) \rightarrow P_q \xi \alpha \beta.$ 
```

Once again we need to prove two lemmas relating P_q with E_q and \overline{m}_q .

```

Lemma  $P_q\_E_q : \forall (\xi : \mathbb{T}) (\alpha \beta : \mathbf{DIG}^\omega), P_q \xi \alpha \beta \rightarrow E_q \xi \alpha \beta.$ 
```

```

Lemma  $\overline{m}_q\_P_q : \forall (\xi : \mathbb{T}) (\alpha \beta : \mathbf{DIG}^\omega) (t : E_q \xi \alpha \beta),
  let  $\xi' := \pi_{24}(\overline{m}_q \xi \alpha \beta t)$  in let  $\alpha' := \pi_{34}(\overline{m}_q \xi \alpha \beta t)$  in
  let  $\beta' := \pi_{44}(\overline{m}_q \xi \alpha \beta t)$  in
   $P_q \xi \alpha \beta \rightarrow P_q \xi' \alpha' \beta'$ .$ 
```

Finally we can define the quadratic algorithm as a function into the coinductive type of streams

$$\bar{q} : \Pi(\xi : \mathbb{T})(\alpha \beta : \mathbf{DIG}^\omega). P_q(\xi, \alpha, \beta) \longrightarrow \mathbf{DIG}^\omega$$

using the cofixed point operator of *Coq*:

```

CoFixpoint  $\bar{q} (\xi : \mathbb{T}) (\alpha \beta : \mathbf{DIG}^\omega) (p : P_q \xi \alpha \beta) : \mathbf{DIG}^\omega :=
  Cons \pi_{14}(\overline{m}_q \xi \alpha \beta (P_q\_E_q \xi \alpha \beta p))
    (\bar{q} \pi_{24}(\overline{m}_q \xi \alpha \beta (P_q\_E_q \xi \alpha \beta p))
      \pi_{34}(\overline{m}_q \xi \alpha \beta (P_q\_E_q \xi \alpha \beta p))
      \pi_{44}(\overline{m}_q \xi \alpha \beta (P_q\_E_q \xi \alpha \beta p))
      (\overline{m}_q\_P_q \xi \alpha \beta (P_q\_E_q \xi \alpha \beta p) p)).$ 
```

To prove that \bar{q} satisfies the specification of the quadratic algorithm we first need the extensional proof irrelevance:

Lemma 3.6. *Let $\xi \in \mathbb{T}, \alpha, \beta \in \mathbf{DIG}^\omega$ and p, p' be two proofs that $P_q(\xi, \alpha, \beta)$ holds. Then the observable outcome of \bar{q} is independent of p and p' , i.e.,*

$$\bar{q}(\xi, \alpha, \beta, p) \cong \bar{q}(\xi, \alpha, \beta, p') . \quad \square$$

Applying this lemma and the fixed point equations of \overline{m}_q we can prove the cofixed point equations of \bar{q} .

Lemma 3.7. *Let $\xi \in \mathbb{T}, \alpha, \beta \in \mathbf{DIG}^\omega$ and p be a proof that $P_q(\xi, \alpha, \beta)$ holds.*

(1) *If $\mathbf{Incl}(\xi, \mathbf{L})$ holds then*

$$\bar{q}(\xi, \alpha, \beta, p) \cong \mathbf{Cons} \ \mathbf{L} \ \bar{q}(\mathbf{L}^{-1} \circ \xi, \alpha, \beta) .$$

(2) *If $\neg \mathbf{Incl}(\xi, \mathbf{L})$ but $\mathbf{Incl}(\xi, \mathbf{R})$ holds then*

$$\bar{q}(\xi, \alpha, \beta, p) \cong \mathbf{Cons} \ \mathbf{R} \ \bar{q}(\mathbf{R}^{-1} \circ \xi, \alpha, \beta) .$$

(3) *If $\neg \mathbf{Incl}(\xi, \mathbf{L})$ and $\neg \mathbf{Incl}(\xi, \mathbf{R})$ but $\mathbf{Incl}(\xi, \mathbf{M})$ holds then*

$$\bar{q}(\xi, \alpha, \beta, p) \cong \mathbf{Cons} \ \mathbf{M} \ \bar{q}(\mathbf{M}^{-1} \circ \xi, \alpha, \beta) .$$

(4) *If $\neg \mathbf{Incl}(\xi, \mathbf{L})$, $\neg \mathbf{Incl}(\xi, \mathbf{R})$ and $\neg \mathbf{Incl}(\xi, \mathbf{M})$ holds then for all p' a proof of property $P_q((\xi \bullet_1(\mathbf{hd}(\alpha))) \bullet_2(\mathbf{hd}(\beta)), \mathbf{tl}(\alpha), \mathbf{tl}(\beta))$ we have*

$$\bar{q}(\xi, \alpha, \beta, p) \cong \bar{q}(\xi \bullet_1(\mathbf{hd}(\alpha))) \bullet_2(\mathbf{hd}(\beta), \mathbf{tl}(\alpha), \mathbf{tl}(\beta), p') . \quad \square$$

Hence \bar{q} agrees with the specification of the quadratic algorithm.

3.4. General Corecursion? Evidently the method for formalising the quadratic algorithm mimics precisely the one used for the homographic algorithm. This suggests that one can generalise this method to obtain a scheme in style of [CHL03] for formalising specification of partial functions on coinductive types. Such a method would be the dual of the Bove–Capretta [BC01] for general recursion. For our situation the dual term *general corecursion* seems suitable. In this article we have not developed such a scheme, as our focus lies on the special case of exact arithmetic algorithms for the coinductive type of reals. Nevertheless, all the intermediate inductive predicates and recursive functions can be obtained by following the shape of the specification. Therefore we consider the method to be generic enough for formalising arbitrary partial coalgebra maps for strictly positive functors in any category modelling CIC.

In fact the method might work in categories for simpler extensions of Martin-Löf type theory. This is because the method does not rely on properties peculiar to CIC; even the distinction between **Set** and **Prop** is not necessary and we could put all the inductive predicates in **Set**. However, with an eye on program extraction, we prefer to keep the distinction between informative and non-informative objects. Note that if we extract the function \bar{h} the argument $P_h(\mu, \alpha)$ will be discarded, resulting in a function $\hat{h}: \mathbb{M} \times \mathbf{DIG}^\omega \rightarrow \mathbf{DIG}^\omega$ which is only different from the original specification modulo unfolding (see the discussion by Bertot [Ber05b]).

It remains to be seen whether the method can be applied in categories other than those modelling some extensions of Martin-Löf type theory.

Comparing our method with the one given by Bertot [Ber05b] we observe that both there and in our work the same idea of dualising Bove–Capretta’s method is pursued. One difference between our work and [Ber05b] is that we consider Ψ_h to be an inductive type while Bertot uses a coinductive predicate **F_infinite**. But our predicate P_h (which is a wrapper for Ψ_h) and Bertot’s **F_infinite** seem to be extensionally equal. Moreover we need the inductive predicate Ψ_h to capture the iteration of \bar{m}_h , a characteristic that

does not occur in Bertot’s method for `filter`. This is due to the slight difference between the homographic algorithm and the general form of `filter` function: in the homographic algorithm the property `Incl` is a dynamic property because the Möbius map, being passed to future steps of the function, is changing all the time; therefore the property that states the productivity should keep track of this. However, by considering a more dynamic form of `filter`, such as the function `etree.filter` introduced in [Niq04, p. 128] it might be possible to extend the method of [Ber05b] and apply it in our case.

Another notable difference is our use of bisimulation equality and the proofs for extensional cofixed point equations which are not present in [Ber05b] where instead another coinductive predicate is used to describe the *connectedness* of a stream with respect to a given property.

Finally, we remark that the rôle of the productivity predicate in our work is reminiscent of the `forall` function in [Sim98] (which is attributed to Berger). There, this function is employed to provide a universal quantifier for total predicates on streams and is used for obtaining higher order functions such as the numerical integration. However, this function which is the basis for defining other functions in [Sim98], itself does not satisfy the *Coq* guardedness condition and hence its formalisation in *Coq* will require additional trickery similar to what we did here for our algorithms. On the other hand, in our work the productivity predicates are inductively defined data-types rather than functions and hence are not hampered by the guardedness condition. It might, however, be possible to combine our method with the techniques in [Sim98] for defining higher order functions.

4. REPRESENTATION

As it is the case with all algorithms, ‘to prove the correctness’ of the homographic and quadratic algorithms can point to different concepts:

- (i) To prove that the algorithms satisfy their *Haskell*-like specification.
- (ii) To prove that the algorithms turn the set \mathbf{DIG}^ω to a partial field and behave as Möbius and quadratic maps on this partial field.
- (iii) To prove that the algorithms correspond to Möbius and quadratic maps on $[-1, 1]$.

Concept (i) tantamounts to proving the cofixed point equations and was carried out in Section 3.2. Concept (ii) requires that we focus on the field operations (via specific tensors for $+$, \times) and prove that they satisfy the algebraic properties of field operations such as commutativity and distributivity. Concept (iii) requires the use of a model of real numbers and indicates that we will project the algorithm to functions on this standard model. It is clear that (iii) is much less work, as we only have to prove the correspondence of the algorithms once and can reduce every question on \mathbf{DIG}^ω to a question on the standard model of \mathbb{R} . This way we do not have to prove one-by-one the field axioms for \mathbf{DIG}^ω . The remainder of this work is based on the concept (iii).

To prove that the algorithms are correct in the sense of (iii), first we should prove that every stream in \mathbf{DIG}^ω represents a real number in $[-1, 1]$. This means that there exists a

total⁷ map ρ from \mathbf{DIG}^ω to $[-1, 1]$ such that for all $\phi_0\phi_1\cdots \in \mathbf{DIG}^\omega$ we have

$$\{\rho(\phi_0\phi_1\cdots)\} = \bigcap_{i=1}^{\infty} \phi_0 \circ \dots \circ \phi_i([-1, 1]) .$$

This can be proven by coinduction, but one needs to define a coinductive predicate that captures the existence of ρ . This leads to the following definition for a binary predicate $\mathbf{rep}: \mathbf{DIG}^\omega \times [-1, 1] \rightarrow \mathbf{Prop}$ with the intended meaning that $\mathbf{rep}(\alpha, r)$ holds if $\rho(\alpha) = \{r\}$.

$$\begin{aligned} \text{CoInductive } \mathbf{rep} &: \mathbf{DIG}^\omega \rightarrow \mathbb{R} \rightarrow \mathbf{Prop} := \\ | \mathbf{rep}_L &: \forall (\alpha \beta: \mathbf{DIG}^\omega) (r: \mathbb{R}), -1 \leq r \leq 1 \rightarrow \\ &\quad \mathbf{rep} \alpha r \rightarrow \beta \cong \mathbf{Cons} \mathbf{L} \alpha \rightarrow \mathbf{rep} \beta (r-1)/(r+3) \\ | \mathbf{rep}_R &: \forall (\alpha \beta: \mathbf{DIG}^\omega) (r: \mathbb{R}), -1 \leq r \leq 1 \rightarrow \\ &\quad \mathbf{rep} \alpha r \rightarrow \beta \cong \mathbf{Cons} \mathbf{R} \alpha \rightarrow \mathbf{rep} \beta (r+1)/(-r+3) \\ | \mathbf{rep}_M &: \forall (\alpha \beta: \mathbf{DIG}^\omega) (r: \mathbb{R}), -1 \leq r \leq 1 \rightarrow \\ &\quad \mathbf{rep} \alpha r \rightarrow \beta \cong \mathbf{Cons} \mathbf{M} \alpha \rightarrow \mathbf{rep} \beta r/3. \end{aligned}$$

The constructors of this coinductive predicate spell out the effect of each digit and as such depend on the choice of the digits. However, they can easily be adapted or generalised for working with other digit sets. The predicate is similar to the predicate **represents** of Bertot [Ber05a, Ber07] and (to a lesser extent) to the predicate \sim' of Hou [Hou06] but has a notable difference: the clause $\beta \cong \mathbf{Cons} \ d \ \alpha$ that is added to each constructor. The purpose of this clause is to facilitate the use of cofixed point equations. Without this clause **rep** would still have the intended topological semantics in terms of ρ , but it would not be usable in the coinductive proof of correctness that we intend to give in the next section. The reason is due to the guardedness condition of *Coq*: even without the \cong clause in the constructors of **rep** we could find a proof X , by coinduction, for the property that

$$\forall \alpha \beta r, \mathbf{rep}(\alpha, r) \rightarrow \alpha \cong \beta \rightarrow \mathbf{rep}(\beta, r) . \quad (4.1)$$

This is the basic property of **rep** that *should have been* enough for the correctness proof. But upon rewriting (4.1) in the course of coinductive proof Δ of correctness we would violate the guardedness condition. This would happen because we would have supplied a recursive occurrence of the coinductive proof Δ which occurs in a subterm of the form

$$X \ \alpha_0 \ \beta_0 \ r_0 \ (\mathbf{rep}_\phi \ \Delta)$$

(where \mathbf{rep}_ϕ is a constructor of **rep**). In such a situation Δ is guarded by \mathbf{rep}_ϕ and X . This does not satisfy the guardedness condition because X is itself a cofixed point whose expansion takes the coinductive proof Δ as an argument in its recursive occurrence in a way that the guardedness condition is rejected. Using cofixed point equations instead of (4.1) we will not land in this situation. Thus we have decided to add the \cong clause which will eliminate the need for (4.1) and instead use the cofixed point equations in the correctness proofs.

Note that (4.1) is still a correct statement and can be used in other situations. In fact we can use it to prove that the inverse of constructors of **rep** hold, e.g.:

$$\forall \alpha r, \mathbf{rep}(\mathbf{Cons} \ \mathbf{L} \ \alpha, r) \rightarrow \mathbf{rep}\left(\alpha, \frac{3r+1}{-r+1}\right) . \quad (4.2)$$

⁷In fact \mathbf{DIG}^ω is a *representation* which means ρ is also surjective. This is easily provable [Niq04, § 5] but it is not needed in the correctness proofs for our algorithms.

The inversion lemmas in turn are used in proving the link between a stream and its future tails. Let α_n (resp. $\alpha|_n$) denote the $n + 1$ -st digit of α (resp. the stream obtained by dropping the first n digits of α). Then we can prove by induction on n and using the inversion lemmas that

$$\forall \alpha r, \mathbf{rep}(\alpha, r) \rightarrow \mathbf{rep}(\alpha|_n, \alpha_{n-1}^{-1} \circ \dots \circ \alpha_0^{-1}(r)) . \quad (4.3)$$

To show that **rep** satisfies its metric property we have to define a function $\llbracket _ \rrbracket$ that evaluates a stream and obtains the real number which is represented by it (cf. `real_value` in [Ber07]). In fact this function calculates the limit of converging sequence of shrinking intervals that is obtained by successive application of the digits starting from the base interval. To be able to define $\llbracket _ \rrbracket$ we should show this converging property. This proof is directly dependent on the metric properties of the specific digit set that we have chosen. Setting $\text{diam}([a, b]) = b - a$ we have to show that

$$\max\{ \text{diam}(\phi_0 \circ \phi_1 \circ \dots \circ \phi_{k-1}([-1, 1])) \mid \phi_i \in \mathbf{DIG} \} \leq \frac{2}{k+1} . \quad (4.4)$$

This is provable by induction on k [Niq04, Corollary 5.7.9] and it entails that the diameters of the intervals form a Cauchy sequence, and so do their endpoints. Hence if we define $l_k(\alpha)$ (resp. $u_k(\alpha)$) to be the lower bound (resp. upper bound) of the interval $\alpha_0 \circ \alpha_1 \circ \dots \circ \alpha_{k-1}([-1, 1])$ we can define⁸

$$\llbracket \alpha \rrbracket = \lim_{i \rightarrow \infty} l_i(\alpha) .$$

Note that (4.4) can be rewritten as

$$\forall \alpha k, u_k(\alpha) - l_k(\alpha) \leq \frac{2}{k+1} ; \quad (4.5)$$

and we can prove (by induction on k) that

$$\forall \alpha k r, \mathbf{rep}(\alpha, r) \rightarrow r \in [l_k(\alpha), u_k(\alpha)] ; \quad (4.6)$$

and hence

$$\forall \alpha k r, \mathbf{rep}(\alpha, r) \rightarrow r \in [-1, 1] . \quad (4.7)$$

Furthermore using the properties of limit we can prove for ϕ a digit

$$\llbracket \mathbf{Cons} \ \phi \ \alpha \rrbracket = \phi(\llbracket \alpha \rrbracket) , \quad (4.8)$$

$$\llbracket \alpha \rrbracket \in [-1, 1] . \quad (4.9)$$

Thus we can prove the following by an easy coinduction on the structure of **rep**.

$$\forall \alpha, \mathbf{rep}(\alpha, \llbracket \alpha \rrbracket) . \quad (4.10)$$

Finally we can prove the main properties of **rep**

$$\forall \alpha r, \llbracket \alpha \rrbracket = r \rightarrow \mathbf{rep}(\alpha, r) ; \quad (4.11)$$

$$\forall \alpha r, \mathbf{rep}(\alpha, r) \rightarrow \llbracket \alpha \rrbracket = r . \quad (4.12)$$

The proof of (4.11) follows from (4.10) and (4.9) while (4.12) needs in addition some properties of the limit.

Hence we have shown that **rep** satisfies its intended metric property with respect to the map ρ defined in the beginning of this section. We conclude the section by pointing out what **rep** does *not* entail. The most important aspect is that our representation \mathbf{DIG}^ω is an

⁸Note that we could have equivalently used the upper bounds.

admissible representation, i.e., it contains enough redundancy so that the usual computable functions are computable with respect to this representation [Niq04, Corollary 5.7.10]. However, the \cong equality does not know anything about this redundancy and it distinguishes the two streams representing the same real number. Therefore for two different representations α_1, α_2 of a real number r , there are two different proofs $\mathbf{rep}(\alpha_1, r)$ and $\mathbf{rep}(\alpha_2, r)$ that do not have any syntactic relation with each other. This, of course, is not an issue for our application of \mathbf{rep} in the correctness proofs of the next section.

5. COINDUCTIVE CORRECTNESS

We are going to prove that the homographic and quadratic algorithms correspond to Möbius and quadratic maps on $[-1, 1]$ as a subset of the standard model of \mathbb{R} . We base our correctness proofs on the coinductive predicate \mathbf{rep} and we prove that for the functions \bar{h} and \bar{q} of Section 3 we have

$$\forall \mu \alpha p r, \mathbf{rep}(\alpha, r) \rightarrow \mathbf{rep}(\bar{h}(\mu, \alpha, p), \mu(r)) ; \quad (5.1)$$

$$\forall \xi \alpha \beta p r_1 r_2, \mathbf{rep}(\alpha, r_1) \rightarrow \mathbf{rep}(\alpha, r_2) \rightarrow \mathbf{rep}(\bar{q}(\xi, \alpha, \beta, p), \xi(r_1, r_2)) . \quad (5.2)$$

It is clear that once we have proven these, applying the Properties (4.11)–(4.12) of \mathbf{rep} , we can derive

$$\begin{aligned} \forall \mu \alpha p r, \llbracket \alpha \rrbracket = r &\rightarrow \llbracket \bar{h}(\mu, \alpha, p) \rrbracket = \mu(r) ; \\ \forall \xi \alpha \beta p r_1 r_2, \llbracket \alpha \rrbracket = r_1 &\rightarrow \llbracket \beta \rrbracket = r_2 \rightarrow \llbracket \bar{q}(\xi, \alpha, \beta, p) \rrbracket = \xi(r_1, r_2) . \end{aligned}$$

Note that these statements require a proof obligation of the productivity predicates P_h and P_q , following our definition of \bar{h} and \bar{q} . This means that we prove the correctness modulo the existence of proofs of these predicates. In the remainder of this section we show how to prove (5.1) and (5.2).

5.1. Homographic Algorithm. We want to prove (5.1). This means that in addition to μ, α and r we are also given a proof p of the statement $P_h(\mu, \alpha)$ that ensures the productivity of $\bar{h}(\mu)$ at α . We use p to obtain some auxiliary tools that we will need in the proof of (5.1). We will also use the terms that were used in our technique for general corecursion (see Section 3). First we need a function

$$\bar{\delta}_h : \Pi(\mu : \mathbb{M})(\alpha : \mathbf{DIG}^\omega). E_h(\mu, \alpha) \longrightarrow \mathbb{N}$$

that counts the number of absorption steps before the first (eventually) coming emission step. Note the resemblance with the definition of the modulus of productivity \bar{m}_h .

```

Fixpoint  $\bar{\delta}_h$  ( $\mu : \mathbb{M}$ ) ( $\alpha : \mathbf{DIG}^\omega$ ) ( $t : E_h \mu \alpha$ ) {struct t} :  $\mathbb{N} :=$ 
match  $\mathbf{Incl}_{\text{dec}}(\mu, \mathbf{L})$  with
| left  $\_ \Rightarrow 0$ 
| right  $t_l \Rightarrow$ 
  match  $\mathbf{Incl}_{\text{dec}}(\mu, \mathbf{R})$  with
  | left  $\_ \Rightarrow 0$ 
  | right  $t_r \Rightarrow$ 
    match  $\mathbf{Incl}_{\text{dec}}(\mu, \mathbf{M})$  with
    | left  $\_ \Rightarrow 0$ 
    | right  $t_m \Rightarrow 1 + \bar{\delta}_h (\mu \circ (\text{hd } \alpha)) (t_l \alpha) (E_{\text{hab\_inv}} \mu \alpha t_l t_r t_m t)$ 
    end
end

```

end
end.

We will also need to prove the proof irrelevance of $\bar{\delta}_h$ (i.e., its value is independent of t), its fixed point equation and its relationship with \bar{m}_h . We state the latter:

Lemma 5.1. *Let $\mu \in \mathbb{M}$, $\alpha \in \mathbf{DIG}^\omega$ and t be a proof that $E_h(\mu, \alpha)$ holds. Then for all n if $\bar{\delta}_h(\mu, \alpha, t) = n$ then there exists $\phi \in \mathbf{DIG}$ such that*

$$\bar{m}_h(\mu, \alpha, t) = \langle \phi, \langle \phi^{-1} \circ \mu \circ \alpha_0 \circ \dots \circ \alpha_{n-1}, \alpha|_n \rangle \rangle . \quad qEd$$

Then we need to prove that if the value of $\bar{\delta}$ is n then after n steps emission will occur, i.e., the emission condition will be satisfied:

Lemma 5.2. *Let $\mu \in \mathbb{M}$, $\alpha \in \mathbf{DIG}^\omega$ and t be a proof that $E_h(\mu, \alpha)$ holds. Then for all n if $\bar{\delta}_h(\mu, \alpha, t) = n$ then one of the following three cases always holds.*

- (a) $\mathbf{Incl}(\mu \circ \alpha_0 \circ \dots \circ \alpha_{n-1}, \mathbf{L}) \wedge \pi_{13}(\bar{m}_h(\mu, \alpha, t)) = \mathbf{L}$;
- (b) $\mathbf{Incl}(\mu \circ \alpha_0 \circ \dots \circ \alpha_{n-1}, \mathbf{R}) \wedge \pi_{13}(\bar{m}_h(\mu, \alpha, t)) = \mathbf{R}$;
- (c) $\mathbf{Incl}(\mu \circ \alpha_0 \circ \dots \circ \alpha_{n-1}, \mathbf{M}) \wedge \pi_{13}(\bar{m}_h(\mu, \alpha, t)) = \mathbf{M}$. □

Both lemmas above are proven by induction on n . All this machinery is used in proving the following lemma which describes the observable (hence the use of \cong) situation of the homographic algorithm at the moment of emission. It explicitly mentions the new input Möbius map passed to the homographic algorithm, the emission condition and the necessary proof obligation.

Lemma 5.3. *Let $\mu \in \mathbb{M}$, $\alpha \in \mathbf{DIG}^\omega$ and p be a proof that $P_h(\mu, \alpha)$ holds. Then there exist $n \in \mathbb{N}$ and $\phi \in \mathbf{DIG}$ that satisfy the following three conditions.*

- (1) $P_h(\phi^{-1} \circ \mu \circ \alpha_0 \circ \dots \circ \alpha_{n-1}, \alpha|_n)$;
- (2) $\mathbf{Incl}(\mu \circ \alpha_0 \circ \dots \circ \alpha_{n-1}, \phi)$;
- (3) *If p' is a proof that $P_h(\phi^{-1} \circ \mu \circ \alpha_0 \circ \dots \circ \alpha_{n-1}, \alpha|_n)$ holds then*

$$\bar{h}(\mu, \alpha, p) \cong \mathbf{Cons} \ \phi \ \bar{h}(\phi^{-1} \circ \mu \circ \alpha_0 \circ \dots \circ \alpha_{n-1}, \alpha|_n, p') . \quad \square$$

Finally we need a property of refining Möbius maps whose proof is immediate, but we state it explicitly to highlight its use.

Lemma 5.4. *If $\mathbf{Incl}(\mu, \phi)$ then $\phi^{-1} \circ \mu$ is refining.* □

Now we have the necessary tools for proving the correctness of the homographic algorithm:

Theorem 5.5. *Let $\mu \in \mathbb{M}$, $\alpha \in \mathbf{DIG}^\omega$, $r \in \mathbb{R}$ and let p be a proof that $P_h(\mu, \alpha)$ holds. If $\mathbf{rep}(\alpha, r)$ holds then*

$$\mathbf{rep}(\bar{h}(\mu, \alpha, p), \mu(r)) .$$

Proof. By Lemma 5.3 there exist n , ϕ and p' such that

$$\mathbf{Incl}(\mu \circ \alpha_0 \circ \dots \circ \alpha_{n-1}, \phi) , \quad (5.3)$$

$$\bar{h}(\mu, \alpha, p) \cong \mathbf{Cons} \ \phi \ \bar{h}(\phi^{-1} \circ \mu \circ \alpha_0 \circ \dots \circ \alpha_{n-1}, \alpha|_n, p') . \quad (5.4)$$

By Property (4.3) of \mathbf{rep} we have

$$\mathbf{rep}(\alpha|_n, \alpha_{n-1}^{-1} \circ \dots \circ \alpha_0^{-1}(r)) .$$

Whence by coinduction applied to

$$\begin{aligned}\mu_c &:= \phi^{-1} \circ \mu \circ \alpha_0 \circ \dots \circ \alpha_{n-1} \text{ ,} \\ \alpha_c &:= \alpha|_n \quad p_c := p' \text{ ,} \\ r_c &:= \alpha_{n-1}^{-1} \circ \dots \circ \alpha_0^{-1}(r) \text{ ;}\end{aligned}$$

we obtain $\mathbf{rep}(\bar{h}(\mu_c, \alpha_c, p_c), \mu_c(r_c))$, i.e.,

$$\mathbf{rep}(\bar{h}(\phi^{-1} \circ \mu \circ \alpha_0 \circ \dots \circ \alpha_{n-1}, \alpha|_n, p'), \phi^{-1} \circ \mu \circ \alpha_0 \circ \dots \circ \alpha_{n-1} \circ \alpha_{n-1}^{-1} \circ \dots \circ \alpha_0^{-1}(r)) \text{ .} \quad (5.5)$$

Let $r_1 := \mu_c \circ \alpha_{n-1}^{-1} \circ \dots \circ \alpha_0^{-1}(r)$. According to Lemma 5.4, from (5.3) it follows that μ_c is refining. Note that by Properties (4.7) and (4.3) of \mathbf{rep} we have

$$\alpha_{n-1}^{-1} \circ \dots \circ \alpha_0^{-1}(r) \in [-1, 1] \text{ ;}$$

and thus according to the refining property $r_1 \in [-1, 1]$.

From here and (5.5), according to the statement of the constructor \mathbf{rep}_ϕ of \mathbf{rep} applied to r_1 and

$$\begin{aligned}\bar{h}(\phi^{-1} \circ \mu \circ \alpha_0 \circ \dots \circ \alpha_{n-1}, \alpha|_n, p') \text{ ,} \\ \bar{h}(\mu, \alpha, p) \text{ ;}\end{aligned}$$

we obtain

$$\mathbf{rep}(\bar{h}(\mu, \alpha, p), \phi(\phi^{-1} \circ \mu \circ \alpha_0 \circ \dots \circ \alpha_{n-1} \circ \alpha_{n-1}^{-1} \circ \dots \circ \alpha_0^{-1}(r))) \text{ ;} \quad (5.6)$$

(note that (5.4) satisfies the \cong clause in \mathbf{rep}_ϕ).

Finally, by simple rewriting and cancelling out the inverse matrices in (5.6) we obtain the conclusion:

$$\begin{aligned}\mathbf{rep}(\bar{h}(\mu, \alpha, p), \phi(\phi^{-1} \circ \mu \circ \alpha_0 \circ \dots \circ \alpha_{n-1} \circ \alpha_{n-1}^{-1} \circ \dots \circ \alpha_0^{-1}(r))) \\ = \mathbf{rep}(\bar{h}(\mu, \alpha, p), \phi \circ \phi^{-1} \circ \mu \circ \alpha_0 \circ \dots \circ \alpha_{n-1} \circ \alpha_{n-1}^{-1} \circ \dots \circ \alpha_0^{-1}(r)) \\ = \mathbf{rep}(\bar{h}(\mu, \alpha, p), \mu(r)) \text{ .}\end{aligned}$$

□

5.2. Quadratic Algorithm. The procedure for the correctness of the quadratic algorithm is quite similar to the case of the homographic algorithm, only the proof itself is more meticulous. First we define a function $\bar{\delta}_q: \Pi(\xi: \mathbb{T})(\alpha, \beta: \mathbf{DIG}^\omega). E_q(\xi, \alpha, \beta) \rightarrow \mathbb{N}$ that outputs the number of steps to the next emission step. We can prove the properties similar to those of $\bar{\delta}_h$.

The main auxiliary lemma in this case is the following.

Lemma 5.6. *Let $\xi \in \mathbb{T}, \alpha, \beta \in \mathbf{DIG}^\omega$ and p be a proof that $P_q(\xi, \alpha, \beta)$ holds. Then there exist $n \in \mathbb{N}$ and $\phi \in \mathbf{DIG}$ that satisfy the following three conditions.*

- (1) $P_q(\phi^{-1} \circ \xi \langle \alpha_0 \circ \dots \circ \alpha_{n-1}, \beta_0 \circ \dots \circ \beta_{n-1} \rangle, \alpha|_n, \beta|_n)$;
- (2) $\mathbf{Incl}(\xi \langle \alpha_0 \circ \dots \circ \alpha_{n-1}, \beta_0 \circ \dots \circ \beta_{n-1} \rangle, \phi)$;
- (3) *If p' is a proof that $P_q(\phi^{-1} \circ \xi \langle \alpha_0 \circ \dots \circ \alpha_{n-1}, \beta_0 \circ \dots \circ \beta_{n-1} \rangle, \alpha|_n, \beta|_n)$ holds, then*

$$\bar{q}(\xi, \alpha, \beta, p) \cong \mathbf{Cons} \ \phi \ \bar{q}(\phi^{-1} \circ \xi \langle \alpha_0 \circ \dots \circ \alpha_{n-1}, \beta_0 \circ \dots \circ \beta_{n-1} \rangle, \alpha|_n, \beta|_n, p') \text{ .} \quad \square$$

Note that $\xi\langle\alpha_0\circ\dots\circ\alpha_{n-1},\beta_0\circ\dots\circ\beta_{n-1}\rangle$ denotes the new tensor after n absorption steps, i.e., after n applications of \bullet_1 and \bullet_2 .

We also need a result on refining tensors which is immediately provable from the definition of refining and **Incl**.

Lemma 5.7. *If **Incl**(ξ, ϕ) then $\phi^{-1} \circ \xi$ is a refining tensor. \square*

From these we can prove the correctness of the quadratic algorithm. In particular we do *not* need any additional property of **rep** apart from those that were used for the homographic algorithm. The proof is quite similar to the proof of Theorem 5.5 and is formalised in *Coq* [Niq07a], and so we do not detail the proof here.

Theorem 5.8. *Let $\xi \in \mathbb{T}$, $\alpha, \beta \in \mathbf{DIG}^\omega$, $r_1, r_2 \in \mathbb{R}$ and let p be a proof that $P_q(\mu, \alpha, \beta)$ holds. If **rep**(α, r_1) and **rep**(β, r_2) hold then*

$$\mathbf{rep}(\bar{q}(\xi, \alpha, \beta, p), \xi(r_1, r_2)) . \quad \square$$

Note that the above theorems require the existence of proofs for productivity statements P_h and P_q . Deriving this property depends on the specific metric properties of each tensor and Möbius map. Next we should prove that for refining maps we can dispose of these productivity predicates.

6. FINAL STEP: REFINING, PRODUCTIVITY AND TOPOLOGICAL CORRECTNESS

So far we have shown that the homographic and quadratic algorithms are ‘correct’ modulo the existence of the productivity predicates P_h and P_q . In this section we will prove that if a Möbius map (resp. quadratic map) is refining then irrespective of the used input stream(s) the property P_h (resp. P_q) is always satisfied.

being refining is enough to ensure the correctness. In light of Theorems 5.5–5.8, this will entail that for refining maps the homographic and quadratic algorithms correspond to Möbius and quadratic maps on $[-1, 1]$. This is the final step in the correctness proof of the algorithm. We call this the *topological correctness* of the algorithms. The reason is that (1) it shows that **Ref** which is a purely metric property is enough to ensure the correctness and the (2) proofs are based on continuity arguments. It will also show that our type theoretic approach of dealing with general corecursion has been sound with respect to the metric semantics of the algorithms.

6.1. Homographic Algorithm. The productivity predicate P_h is the latest in a chain of type theoretic auxiliary predicates and functions E_h, \bar{m}_h and Ψ_h . By examining these predicates we observe that the only topological notion appears in the type of the constructors of E_h , in the form of the emission condition. We should follow this link to obtain the productivity predicate for a refining Möbius map.

First we state some elementary properties of the interval predicates that we introduced in Section 2. We omit the proofs which are trivial case distinctions on comparisons of the end points of the intervals.

Lemma 6.1. *Let μ be a Möbius map.*

- (1) μ is bounded (i.e., its denominator does not vanish in $[-1, 1]$) if and only if the property **Bounded**(μ) holds.
- (2) μ is refining (i.e., it maps $[-1, 1]$ into itself) if and only if **Ref**(μ) holds.

- (3) If μ is bounded and for each $r \in [-1, 1]$, $\mu(r) \in [-1, 0]$ then $\mathbf{Incl}(\mu, \mathbf{L})$.
 (4) If μ is bounded and for each $r \in [-1, 1]$, $\mu(r) \in [0, 1]$ then $\mathbf{Incl}(\mu, \mathbf{R})$.
 (5) If μ is bounded and for each $r \in [-1, 1]$, $\mu(r) \in [-\frac{1}{3}, \frac{1}{3}]$ then $\mathbf{Incl}(\mu, \mathbf{M})$. □

This lemma ensures us that we can comfortably work with the predicates \mathbf{Ref} and $\mathbf{Bounded}$ to prove results about refining maps. An easily provable consequence of the above lemma is the following.

Lemma 6.2. *Let μ_1 and μ_2 be Möbius maps.*

- (1) If $\mathbf{Bounded}(\mu_1)$ and $\mathbf{Ref}(\mu_2)$ then $\mathbf{Bounded}(\mu_1 \circ \mu_2)$.
 (2) If $\mathbf{Ref}(\mu_1)$ and $\mathbf{Ref}(\mu_2)$ then $\mathbf{Ref}(\mu_1 \circ \mu_2)$. □

Given a refining Möbius map μ , we define the *diameter* of μ to be

$$\mathbf{diam}(\mu) = |\mu(-1) - \mu(1)| .$$

Next we need a metric property of the representation, which measures the amount of the redundancy of the representation. For any set of Φ of refining Möbius maps we define

$$\mathbf{red}(\Phi) = \min\{|\phi_i(-1) - \phi_j(1)| \mid \phi_i, \phi_j \in \mathbf{DIG}, \phi_i(-1) \neq \phi_j(1)\} .$$

The above definition is based on the intuitive idea that the more overlap between ranges of the digits, the more choices one has for representing real numbers. The intended meaning is that for two digit sets Φ_1 and Φ_2 , with the same number of elements, if $\mathbf{red}(\Phi_1) > \mathbf{red}(\Phi_2)$ then Φ_1 has more redundancy. Note that this intended meaning does not work for adding extra digits (which decreases \mathbf{red}) but rather for comparing the redundancy of two digits sets with the same number of digits.

Clearly

$$\mathbf{red}(\mathbf{L}, \mathbf{R}, \mathbf{M}) = \frac{1}{3} .$$

Then using this quantity we state and prove the following lemma that shows that for refining Möbius maps with sufficiently small diameter the emission condition holds.

Lemma 6.3. *If μ is a refining Möbius map and $\mathbf{diam}(\mu) < \frac{1}{3} = \mathbf{red}(\mathbf{L}, \mathbf{R}, \mathbf{M})$ then there exists $\phi \in \mathbf{DIG}$ such that $\mathbf{Incl}(\mu, \phi)$.*

Proof. The proof uses Lemma 6.1.3–5, and case distinction on comparison of $\mu(-1)$ and $\mu(1)$ with $\mathbf{L}(1) = \mathbf{R}(-1) = 0$, $\mathbf{M}(-1) = -\frac{1}{3}$ and $\mathbf{M}(1) = \frac{1}{3}$. □

At this point the following question arises: can we decrease the diameter of an arbitrary refining Möbius maps by repeated absorption in a way that it becomes less than $1/3$? The answer is positive. First we should assess the diameter of the product of two Möbius maps because an absorption step is nothing but a product with a digit. Hence we prove the following lemma.

Lemma 6.4. *Let $\mu_1 = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ be a refining Möbius map.*

- (1) *If μ_2 is a refining Möbius map, then*

$$\mathbf{diam}(\mu_1 \circ \mu_2) = \frac{\mathbf{diam}(\mu_2) \cdot |\det \mu_1|}{|(c\mu_2(-1) + d)(c\mu_2(1) + d)|} .$$

(2) If $\alpha \in \mathbf{DIG}^\omega$, then

$$\mathbf{diam}(\mu \circ \alpha_0 \circ \dots \circ \alpha_{n-1}) = \frac{(u_n(\alpha) - l_n(\alpha)) \cdot |\det \mu|}{|(c \cdot u_n(\alpha) + d)(c \cdot l_n(\alpha) + d)|} .$$

Proof. (1) Note that by Lemma 6.2.2 the product is refining and thus the left hand side is well-defined. The identity follows by straightforward calculation (See [Hec02]).

(2) Since all the digits are refining, we can apply part (1) with $\mu_2 := \alpha_0 \circ \dots \circ \alpha_{n-1}$. \square

Next we can prove that after finitely many absorption steps the emitting condition holds and thus the algorithm is ‘informally’ productive.

Theorem 6.5. *Let $\mu = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ be a refining Möbius map and $\alpha \in \mathbf{DIG}^\omega$. Then there exist $n \in \mathbb{N}$ and $\phi \in \mathbf{DIG}$ such that $\mathbf{Incl}(\mu \circ \alpha_0 \circ \dots \circ \alpha_{n-1}, \phi)$ holds.*

Proof. Let $X := \max(\frac{1}{|c+d|}, \frac{1}{|d-c|})$ and take

$$n := \lceil 6 \cdot |\det \mu| \cdot X^2 \rceil \tag{6.1}$$

(here we take the ceiling using the Archimedean property of \mathbb{Q}).

Note that since μ is refining then it is bounded and hence $\begin{bmatrix} 0 & 1 \\ c & d \end{bmatrix}$ is bounded and monotone in $[-1, 1]$. Thus for all $x \in [-1, 1]$ we have

$$\frac{1}{|cx + d|} \leq X . \tag{6.2}$$

On that account we calculate:

$$\begin{aligned} \mathbf{diam}(\mu \circ \alpha_0 \circ \dots \circ \alpha_{n-1}) &= \frac{(u_n(\alpha) - l_n(\alpha)) \cdot |\det \mu|}{|(c \cdot u_n(\alpha) + d)(c \cdot l_n(\alpha) + d)|} && \text{by Lemma 6.4.2} \\ &\leq X^2 \cdot (u_n(\alpha) - l_n(\alpha)) \cdot |\det \mu| && \text{by (6.2)} \\ &\leq \frac{2 \cdot X^2 \cdot |\det \mu|}{n + 1} && \text{by (4.5)} \\ &< \frac{1}{3} && \text{by (6.1).} \end{aligned}$$

Hence we can apply Lemma 6.3 and obtain ϕ as required. \square

The above existential theorem gives us a pair of witnesses $\langle n, \phi \rangle$, but we would like to make the canonical choice of the *smallest* such witness. This is possible because we are dealing with a decidable predicate \mathbf{Incl} (see $\mathbf{Incl}_{\text{dec}}$), on the well-founded set $\mathbb{N} \times \mathbf{DIG}$. The idea is that once we have a witness we can perform a search bounded by this witness to obtain the smallest witness. This can be summarised as the following result.

Lemma 6.6. *Let μ be a refining Möbius map and $\alpha \in \mathbf{DIG}^\omega$. Then there exist $n \in \mathbb{N}$ and $\phi \in \mathbf{DIG}$ such that the following two conditions hold.*

- (1) $\mathbf{Incl}(\mu \circ \alpha_0 \circ \dots \circ \alpha_{n-1}, \phi)$;
- (2) $\forall m < n \forall \phi', \neg \mathbf{Incl}(\mu \circ \alpha_0 \circ \dots \circ \alpha_{m-1}, \phi')$.

At this point we are ready to embark on proving our type theoretic predicates. First we prove that being refining implies that E_h holds.

Lemma 6.7. *Let μ be a refining Möbius map and $\alpha \in \mathbf{DIG}^\omega$. Then $E_h(\mu, \alpha)$ holds.*

Proof. Assume n is obtained by applying Lemma 6.6 to μ and α . We proceed by *induction*⁹ on n . If $n = 0$ then $\mathbf{Incl}(\mu, \phi)$ should hold for some ϕ and we can apply the corresponding constructor of E_h among E_{hL}, E_{hR} and E_{hM} .

Now assume we have proven the conclusion for all refining maps for which the witness given by Lemma 6.6 is k and $n = k + 1$. Note that for $\mu \circ \alpha_0$ and $\alpha|_1$ the witness given by Lemma 6.6 must be k . Therefore by induction hypothesis we have

$$E_h(\mu \circ \alpha_0, \alpha|_1) . \quad (6.3)$$

Since $0 < k$ we know by Lemma 6.6.2 that $\mathbf{Incl}(\mu, \phi)$ does not hold for any ϕ , i.e.

$$\neg \mathbf{Incl}(\mu, \mathbf{L}) , \quad \neg \mathbf{Incl}(\mu, \mathbf{R}) , \quad \neg \mathbf{Incl}(\mu, \mathbf{M}) . \quad (6.4)$$

Consequently, we can apply the constructor E_{hab} to (6.4) and (6.3) to obtain a proof of $E_h(\mu, \alpha)$. \square

Next we need two technical lemmas for relating the refining property with the two auxiliary functions $\bar{\delta}_h$ (Section 5.1) and \bar{m}_h (Section 3.1).

Lemma 6.8. *Let μ be a refining Möbius map and $\alpha \in \mathbf{DIG}^\omega$. Let n be given by applying Lemma 6.6 to μ and α . Let t_0 be the proof given by Lemma 6.7 that $E_h(\mu, \alpha)$ holds. Then*

$$\bar{\delta}_h(\mu, \alpha, t_0) = n .$$

Proof. By induction on n . For $n = 0$ by Lemma 6.6.1 $\mathbf{Incl}(\mu, \alpha)$ should hold for some ϕ , and the conclusion follows from the definition of $\bar{\delta}_h$.

Now assume we have proven the conclusion for all refining maps for which the witness given by Lemma 6.6 is k and $n = k + 1$. Applying the induction hypothesis to $\mu \circ \alpha_0$ and $\alpha|_1$ we obtain

$$\bar{\delta}_h(\mu \circ \alpha_0, \alpha|_1, t_1) = k , \quad (6.5)$$

where t_1 is the proof given by Lemma 6.7 for

$$E_h(\mu \circ \alpha_0, \alpha|_1) .$$

Furthermore since $0 < k$ by Lemma 6.6.2

$$\neg \mathbf{Incl}(\mu, \mathbf{L}) , \quad \neg \mathbf{Incl}(\mu, \mathbf{R}) , \quad \neg \mathbf{Incl}(\mu, \mathbf{M}) .$$

From here together with the definition of $\bar{\delta}_h$ and (6.5) we obtain

$$\bar{\delta}_h(\mu, \alpha, t_0) = \bar{\delta}_h(\mu \circ \alpha_0, \alpha|_1, t_1) + 1 = k + 1 = n . \quad \square$$

Lemma 6.9. *Let μ be a refining Möbius map and $\alpha \in \mathbf{DIG}^\omega$. Let t_0 be the proof given by Lemma 6.7 that $E_h(\mu, \alpha)$ holds. Let $\bar{m}_h(\mu, \alpha, t_0) := \langle \phi, \langle \mu', \alpha' \rangle \rangle$. Then μ' is refining.*

Proof. Let n be obtained by applying Lemma 6.6 to μ and α . By Lemma 6.8 we have

$$\bar{\delta}_h(\mu, \alpha, t_0) = n . \quad (6.6)$$

Hence by applying Lemma 5.1 to μ, α, t_0 and n we obtain a digit ϕ' such that

$$\bar{m}_h(\mu, \alpha, t_0) = \langle \phi', \langle \phi'^{-1} \circ \mu \circ \alpha_0 \circ \dots \circ \alpha_{n-1}, \alpha|_n \rangle \rangle$$

Hence

$$\phi' = \phi , \quad \mu' = \phi'^{-1} \circ \mu \circ \alpha_0 \circ \dots \circ \alpha_{n-1} .$$

⁹This might seem odd, as n is a witness given to us; nevertheless we can carry out induction for a universal property for any m such that $m = n$.

Note that the inverse of digits are *not* refining, so we cannot use Lemma 6.2 to prove that μ' is refining. But instead we apply Lemma 5.4. So we have to show that $\mathbf{Incl}(\mu', \phi)$. But this is evident by applying Lemma 5.2 to (6.6). \square

Finally, we prove that being refining implies Ψ_h , which is the crux of the productivity property that we used for defining the homographic algorithm.

Lemma 6.10. *Let $n \in \mathbb{N}$, μ be a refining Möbius map and $\alpha \in \mathbf{DIG}^\omega$. Then $\Psi_h(n, \mu, \alpha)$ holds.*

Proof. By induction on n . If $n = 0$ then by Lemma 6.7 we know that $E_h(\mu, \alpha)$ holds, and hence we can apply the constructor Ψ_{h0} to obtain the conclusion. Assume the conclusion holds for $n = k$ and arbitrary refining Möbius maps. Let t_0 be the *specific* proof given by Lemma 6.7 that $E_h(\mu, \alpha)$ holds, and let

$$\overline{m}_h(\mu, \alpha, t_0) := \langle \phi, \langle \mu', \alpha' \rangle \rangle .$$

Due to our choice of t_0 , by Lemma 6.9 it follows that μ' is refining. Thus we can apply the induction hypothesis to μ' and α' to obtain a proof of $\Psi_h(k, \mu', \alpha')$ which can be rewritten as:

$$\Psi_h(k, \pi_{23}(\overline{m}_h(\mu, \alpha, t_0)), \pi_{33}(\overline{m}_h(\mu, \alpha, t_0))) . \quad (6.7)$$

Hence by applying the constructor Ψ_{hS} to μ, α, t_0 and (6.7) the result follows. \square

As a corollary we obtain the main result of this section. This states that the purely topological property **Ref** entails the type theoretic productivity predicate P_h which we had added to satisfy the guardedness condition of *Coq*.

Corollary 6.11. *Let μ be a refining Möbius map and $\alpha \in \mathbf{DIG}^\omega$. Then $P_h(\mu, \alpha)$ holds.*

6.2. Quadratic Algorithm. We should prove that if a quadratic map is refining then the predicates E_q, Ψ_q and P_q hold. We follow the same route as for the homographic algorithm to prove the counterpart of Theorem 6.5. There we defined the diameter of a refining Möbius map and calculated a uniform upper bound for it after finite absorption steps. Here the situation is slightly more complicated, because when assessing the effect of a quadratic map on two intervals (one for each argument) we should examine the values at 4 corners of the Cartesian product. So already the definition of the diameter will be slightly different. But first we state the properties of the interval predicates for a quadratic map (see Appendix A for the definitions), which are again provable by straightforward case analysis.

Lemma 6.12. *Let ξ be a quadratic map.*

- (1) ξ is bounded (i.e., its denominator does not vanish in $[-1, 1] \times [-1, 1]$) if and only if $\mathbf{Bounded}(\xi)$ holds.
- (2) ξ is refining (i.e., it maps $[-1, 1] \times [-1, 1]$ into itself) if and only if $\mathbf{Ref}(\xi)$ holds.
- (3) If ξ is bounded and for each $r_1, r_2 \in [-1, 1]$, $\xi(r_1, r_2) \in [-1, 0]$ then $\mathbf{Incl}(\xi, \mathbf{L})$.
- (4) If ξ is bounded and for each $r_1, r_2 \in [-1, 1]$, $\xi(r_1, r_2) \in [0, 1]$ then $\mathbf{Incl}(\xi, \mathbf{R})$.
- (5) If ξ is bounded and for each $r_1, r_2 \in [-1, 1]$, $\xi(r_1, r_2) \in [-\frac{1}{3}, \frac{1}{3}]$ then $\mathbf{Incl}(\xi, \mathbf{M})$. \square

Recall that there were two ways of composing a quadratic map and a Möbius map. Using the lemma above we can derive the following about these two products.

Lemma 6.13. *Let ξ be a quadratic map and μ_1 and μ_2 be Möbius maps.*

- (1) If $\mathbf{Bounded}(\xi)$, $\mathbf{Ref}(\mu_1)$ and $\mathbf{Ref}(\mu_2)$ then $\mathbf{Bounded}(\xi \bullet_1 \mu_1 \bullet_2 \mu_2)$.
- (2) If $\mathbf{Ref}(\xi)$, $\mathbf{Ref}(\mu_1)$ and $\mathbf{Ref}(\mu_2)$ then $\mathbf{Ref}(\xi \bullet_1 \mu_1 \bullet_2 \mu_2)$. \square

Note that for a refining quadratic map ξ , $\xi([-1, 1], -)$ is a function on subintervals of $[-1, 1]$. We define the *diameter* of ξ on *rational* subintervals $[x_0, y_0]$ and $[x_1, y_1]$ of $[-1, 1]$ to be

$$\mathbf{diam}_2(\xi, [x_0, y_0], [x_1, y_1]) = [\min(r_{00}, r_{01}, r_{10}, r_{11}), \max(r_{00}, r_{01}, r_{10}, r_{11})] ,$$

where

$$\begin{bmatrix} r_{00} & r_{01} \\ r_{10} & r_{11} \end{bmatrix} = \begin{bmatrix} \xi(x_0, y_0) & \xi(x_0, y_1) \\ \xi(x_1, y_0) & \xi(x_1, y_1) \end{bmatrix} .$$

We will usually use $\mathbf{diam}_2(\xi, [-1, 1], [-1, 1])$ however in some intermediate steps of the proofs we sometimes have to invoke diameter for other rational subintervals. Again we can prove a lemma relating the diameter, redundancy and **Incl**.

Lemma 6.14. *If ξ is a refining quadratic map for which $\mathbf{diam}_2(\xi, [-1, 1], [-1, 1]) < \frac{1}{3} = \mathbf{red}(\mathbf{L}, \mathbf{R}, \mathbf{M})$ then there exists $\phi \in \mathbf{DIG}$ such that $\mathbf{Incl}(\xi, \phi)$.*

Proof. If $\xi([-1, 1], [-1, 1]) = [x, y]$, we consider the following three cases.

If $y \leq 0$:	Incl (ξ, \mathbf{L})	by Lemma 6.12.3 ,	
else if $0 \leq x$:	Incl (ξ, \mathbf{R})	by Lemma 6.12.4 ,	
otherwise :	Incl (ξ, \mathbf{M})	by Lemma 6.12.5 .	□

Unlike what we did in Lemma 6.4, here we cannot find a closed formula for the diameter of the product of a quadratic map and two Möbius maps. We should find another way of ensuring that in the absorption steps the diameter can become smaller than $1/3$. At the first glance it seems that we really have to prove the uniform continuity of quadratic map considered as a binary function on rational numbers, but careful examination of the proof of Theorem 6.5 shows that the pointwise continuity could be enough. Thus we prove the following lemma.

Lemma 6.15. *Let ξ be a refining quadratic map. Then for all $0 < \varepsilon \in \mathbb{Q}^+$ there exist $0 < \vartheta_0, \vartheta_1 \in \mathbb{Q}^+$ such that for all $x_0, x_1, y_0, y_1 \in [-1, 1]$ if $|x_0 - x_1| < \vartheta_0$ and $|y_0 - y_1| < \vartheta_1$ then*

$$|\xi(x_0, y_0) - \xi(x_1, y_1)| < \varepsilon .$$

Proof. This is equivalent to the continuity of a refining (and hence bounded) quadratic map on $[-1, 1] \times [-1, 1]$. □

As a corollary we can locally bound the diameter of a refining quadratic map:

Corollary 6.16. *Let ξ be a refining quadratic map. Then for all $0 < \varepsilon \in \mathbb{Q}^+$ there exist $0 < \vartheta_0, \vartheta_1 \in \mathbb{Q}^+$ such that for all $x_0, x_1, y_0, y_1 \in [-1, 1]$ if $|x_0 - x_1| < \vartheta_0$ and $|y_0 - y_1| < \vartheta_1$ then*

$$\mathbf{diam}_2(\xi, [x_0, y_0], [x_1, y_1]) < \varepsilon .$$

At this point we are ready to state and prove the counterpart of Theorem 6.5, that ensures the flow of emission steps after a finite number of absorption steps.

Theorem 6.17. *Let ξ be a refining quadratic map and $\alpha, \beta \in \mathbf{DIG}^\omega$. Then there exist $n \in \mathbb{N}$ and $\phi \in \mathbf{DIG}$ such that $\mathbf{Incl}(\xi(\alpha_0 \circ \dots \circ \alpha_{n-1}, \beta_0 \circ \dots \circ \beta_{n-1}), \phi)$ holds.*

Proof. Let ϑ_0, ϑ_1 be given by applying Corollary 6.16 to $\varepsilon = 1/3$. Take

$$n := \max(\lceil \frac{2}{\vartheta_0} \rceil, \lceil \frac{2}{\vartheta_1} \rceil) .$$

Let

$$x_0 := l_n(\alpha) , \quad y_0 := l_n(\beta) , \quad x_1 := u_n(\alpha) , \quad y_1 := u_n(\beta) ,$$

Note that due to (4.5) we have

$$|x_0 - x_1| \leq \frac{2}{n+1} , \quad |y_0 - y_1| \leq \frac{2}{n+1}$$

From here together with Corollary 6.16 for x_0, x_1, y_0 and y_1 we obtain

$$\mathbf{diam}_2(\xi, [x_0, y_0], [x_1, y_1]) < \frac{1}{3} . \quad (6.8)$$

But an easy calculation shows that

$$\begin{aligned} \mathbf{diam}_2(\xi, [x_0, y_0], [x_1, y_1]) &= \mathbf{diam}_2(\xi, \alpha_0 \circ \dots \circ \alpha_{n-1}[-1, 1], \beta_0 \circ \dots \circ \beta_{n-1}[-1, 1]) \\ &= \mathbf{diam}_2(\xi \langle \alpha_0 \circ \dots \circ \alpha_{n-1}, \beta_0 \circ \dots \circ \beta_{n-1} \rangle, [-1, 1], [-1, 1]) . \end{aligned} \quad (6.9)$$

Therefore we can apply Lemma 6.14 with (6.8) and (6.9) to obtain the desired digit ϕ . \square

The above proof is based on the pointwise continuity of refining quadratic maps. Of course we can find a uniform bound to be applied in the proof of the algorithm, but that would require a formalisation of the bivariate version of the Heine–Borel theorem. For our purpose the above proof suffices, because it gives us a witness which we will use in a bounded search for finding the *smallest* such witness. Furthermore, the pointwise continuity gives a finer estimate of the complexity of the algorithm, but we will not pursue this matter here. Results concerning the complexity of these algorithms can be found in [Hec98, Krz01].

The discrepancy between the homographic algorithm and the quadratic algorithm ends here. This means that the remaining steps for deriving the productivity predicate P_q for the refining quadratic maps is essentially the same as those for the homographic algorithm. Therefore we only present the three important statements here, and we refrain from repeating the arguments. The proofs can be consulted in the formalisation package [Niq07a].

Lemma 6.18. *Let ξ be a refining quadratic map and $\alpha, \beta \in \mathbf{DIG}^\omega$. Then $E_q(\xi, \alpha, \beta)$ holds.*

Lemma 6.19. *Let $n \in \mathbb{N}$, ξ be a refining quadratic map and $\alpha, \beta \in \mathbf{DIG}^\omega$. Then $\Psi_q(n, \xi, \alpha, \beta)$ holds.*

Corollary 6.20. *Let ξ be a refining quadratic map and $\alpha, \beta \in \mathbf{DIG}^\omega$. Then $P_q(\xi, \alpha, \beta)$ holds.*

As expected the productivity predicate P_q , being a *local* property of the domain of the algorithm gives a finer description of the domain of the algorithm. For example the addition tensor is not refining on $[-1, 1]$ but the quadratic algorithm applied with the addition tensor is productive for $\alpha, \beta \in [-\frac{1}{4}, \frac{1}{4}]$. However, if we transfer the computations to the entire real line by adding a redundant sign bit, then the refining quadratic maps are enough for calculating elementary functions [Pot98].

7. REEXAMINING THE METHOD

We can outline the path that we followed in this article in the following steps.

- (1) Implementing algorithms in type theoretic language.
- (2) Proving that they satisfy their *Haskell*-like specification.
- (3) Proving that the algorithms correspond to *partial* Möbius and quadratic maps on $[-1, 1]$:

- (a) They are total on those subsets of $\mathbb{M} \times \mathbf{DIG}^\omega$ and $\mathbb{T} \times \mathbf{DIG}^\omega$ for which the productivity predicates P_h and P_q hold.
- (b) They are total on the subsets of \mathbb{M} and \mathbb{T} containing refining maps.

There are different aspects of the proofs in each of these phases that we would like to clarify.

Dependence on Representation. None of the steps above *depend* on the representation, although the specific proofs about this representation, as well as our choice of the base interval appeared frequently in our reasoning. In [Niq04, Chapter 5] we show that as long as a representation satisfies a few properties with respect to the effect of its digits on the chosen base interval, it will not affect the productivity and thus correctness behaviour of the algorithms. For example for any such representation and for any choice of base interval we can derive a counterpart of (4.5) — with a different bound— and Lemma 6.4.1.

Type Theoretic vs. Topological Properties. We pointed out this correlation throughout the article. Here we summarise it for all the above steps. Step 1 above is purely type theoretic. It simply consist of writing a function parametrised by a proof obligation that passes the type checking in the functional programming language of *Coq*. The proofs (proof irrelevance lemmas and termination certificates) are objects that are meaningful and expressible in a framework where dependent types and (co)inductive types exist. These proofs ensure an initial layer of correctness: that the input and output have the right type and whether the algorithms are productive for given inputs. One could say that in this step the domain and codomain of the algorithms are described.

Step 2 is the first phase in proving correctness with respect to the intended semantics, but it still has a purely type theoretic nature. The proofs are based on working with bisimulation and deriving cofixed point equations for the algorithms. They relate the algorithms to their specification written in a more liberal functional programming (e.g. Haskell), where termination and productivity are not hampering us.

Step 3 possesses both type-theoretic and topological elements. In Step 3a this is best captured in the relationship between **rep** (a type theoretic predicate) and $\llbracket _ \rrbracket$ (a purely topological operation). While in Step 3b the topological aspects dealing with the pointwise continuity of the underlying maps are dominant. Still we can observe that Lemmas 6.7–6.11 resort to type theoretic properties such as proof irrelevance and termination certificates. The correlation of these two aspects is highlighted in the final propositions about the purely metric property **Ref** and the productivity predicates that were required for the type-checking in Step 1.

Statistics on Formalisation. Finally we present some of the statistics pertaining to the formalised algorithms. They indicate the size (in kilobytes) and length (in number of lines¹⁰) of the ASCII code of the formalisation.

In Table 1 we present the relative size of the formalisation work for each of the above steps. We also add a separate category (last row) for parts of the formalisation that included general results about digits, Möbius maps, quadratic maps and several interval predicates. In Table 2 we take an alternative viewpoint and present the statistics for each algorithm separately. The first row (digits) denotes the part that was common to both algorithms.

¹⁰Number of lines is obtained using the command `coqwc` which disregards the commented and blank lines.

The last column gives the size of the *Haskell* program that is obtained by using the *program extraction mechanism* of *Coq* [CDT06, §18].

Step	Size (percentage of total)	Length
Step 1	27 K (5.5%)	529 lines
Step 2	8 K (1.6%)	166 lines
Step 3a	111 K (23.4%)	2099 lines
Step 3b	62 K (12.9%)	1107 lines
General facts	268 K (56.3%)	4596 lines
<i>Total</i>	475 K (100%)	8497 lines

Table 1: **Various phases of the formalisation.**

task	size	lines	extracted <i>Haskell</i>
digits	128 K	2541 lines	51 lines
homographic	147 K	2845 lines	61 lines
quadratic	200 K	3111 lines	126 lines
<i>Total</i>	475 K	8497 lines	238 lines

Table 2: **Relative size of proofs and programs for different algorithms.**

The presented statistics provide a good indication of the state of the art in formalising mathematical results and verifying algorithms. However, (and fortunately) these statistics are likely to be outdated as new versions of *Coq* featuring more automation tools will become available.

8. CONCLUSIONS AND FURTHER WORK

We have shown the correctness of the homographic and quadratic algorithms on a stream representation of real numbers in $[-1, 1]$. Following the general set-up of [Niq04, § 5] the method is easily extensible to any admissible digit set for any compact proper subinterval of the extended real numbers $[-\infty, +\infty]$. Our correctness proofs use an inductive productivity predicate and a coinductive predicate **rep** that relates \mathbf{DIG}^ω and $[-1, 1]$. We use the coinductive machinery of the *Coq* proof assistant to formalise functions on infinite objects and coinductive proofs. In particular we base our treatment of coinductive functions on their cofixed point equations. These exploit the inherent infinite nature of streams by adhering to \cong which is a bisimulation relation and is more suitable than the inductive (*Leibniz*) equality. The coinductive arguments themselves are independent of *Coq* and can be formalised in any proof assistant that accommodates coinductive types. Furthermore we prove — in *Coq* — that for the class of Möbius and quadratic maps that satisfy a metric property (being refining), the homographic and quadratic algorithms will output provably infinite streams for any input.

Among several perceivable directions for the future work, the more immediate one would be to continue the *Coq* formalisation of the algorithms, by developing a fully modular framework that axiomatises the properties of representations and refining maps that are needed for the formalisation. Each specific representation would then be portable into our formalisation if a suitable interface is satisfied. This will pave the way for applying our

formalisation to more efficient representations such as the one used by Edalat–Potts [EP97] or the ternary one used in [CDG06, Hou06, Ber07, MRE07].

The big picture would be to continue working on the formalisation of the Edalat–Potts framework for exact real arithmetic. The homographic and quadratic algorithm are the base case of Edalat and Potts’ *normalisation algorithm* which is defined on the coinductive type of expression trees [EP97, Pot98]. Therefore if we could apply the method of this article to formalise and verify this algorithm we could obtain all the elementary functions. Unfortunately this does not seem to be possible: the difficulty lies in the general corecursion used in the normalisation algorithm, The method of the Section 3 needs a more complicated machinery than that of CIC to be applicable to the normalisation algorithm. This is because the normalisation algorithm is a nested algorithm and therefore applying our method the modulus of productivity \overline{m}_h will be a nested function too. It is well-known that applying Bove–Capretta method for formalising nested recursive functions requires the presence of *inductive–recursive* types [BC01, Dyb00]. In this case the inductive domain predicate will become an inductive–recursive predicate that is defined simultaneously with the nested function. A similar phenomenon happens in our method, in the sense that we need a notion similar to induction–recursion that would allow for the simultaneous definition of an inductive predicate together with a cofixed point. The author is exploring the possibility of defining such a notion. Recent work by Setzer on combining induction–recursion and general recursion seems to open up new possibilities for our work in this directions [Set06].

One can contemplate of adapting and generalising our method for lazy exact arithmetic algorithms beyond the Edalat–Potts algorithm. One starting point in this direction would be to follow the technique used in [Sim98] (see also Section 3.4) for obtaining higher order functions on real numbers such as the numerical integration.

ACKNOWLEDGEMENT

The author wishes to thank the anonymous referees for their valuable comments and their proposed simplifications that helped improve the paper.

REFERENCES

- [AAG05] M. Abbott, T. Altenkirch, and N. Ghani. Containers - constructing strictly positive types. *Theoret. Comput. Sci.*, 342:3–27, September 2005.
- [Bar01] F. Bartels. Generalised coinduction. In A. Corradini, M. Lenisa, and U. Montanari, editors, *Proc. of 4th Workshop on Coalgebraic Methods in Computer Science, CMCS’01, Genova, Italy, 6–7 Apr. 2001*, volume 44(1) of *Electron. Notes Theor. Comput. Sci.* Elsevier Science Publishers, Amsterdam, 2001.
- [BBLT06] A. Beckmann, U. Berger, B. Löwe, and J. V. Tucker, editors. *Logical Approaches to Computational Barriers: Second Conference on Computability in Europe, CiE 2006, Swansea, UK, June 30–July 5, 2006. Proc.*, volume 3988 of *Lecture Notes in Comput. Sci.* Springer-Verlag, 2006.
- [BC01] A. Bove and V. Capretta. Nested general recursion and partiality in type theory. In R. J. Boulton and P. B. Jackson, editors, *Theorem Proving in Higher Order Logics, 14th International Conference, TPHOLS 2001, Edinburgh, Scotland, UK, September 3–6, 2001, Proc.*, volume 2152 of *Lecture Notes in Comput. Sci.*, pages 121–135. Springer-Verlag, 2001.
- [BC04] Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development. Coq’Art: The Calculus of Inductive Constructions*. EATCS Series. Springer-Verlag, 2004.
- [Ber05a] Y. Bertot. CoInduction in Coq. In *Lecture Notes of TYPES Summer School 2005, August 15–26 2005, Göteborg, Sweden. vol II*, 2005. http://www.cs.chalmers.se/Cs/Research/Logic/TypesSS05/Extra/lectnotes_vol12.pdf, [cited 9 July 2008].

- [Ber05b] Y. Bertot. Filters on coinductive streams, an application to Eratosthenes' sieve. In P. Urzyczyn, editor, *TLCA*, volume 3461 of *Lecture Notes in Comput. Sci.*, pages 102–115. Springer-Verlag, 2005.
- [Ber07] Y. Bertot. Affine functions and series with co-inductive real numbers. *Math. Structures Comput. Sci.*, 17(1):37–63, March 2007.
- [BES02] A. Bauer, M. H. Escardó, and A. K. Simpson. Comparing functional paradigms for exact real-number computation. In *ICALP*, volume 2380 of *Lecture Notes in Comput. Sci.*, pages 488–500. Springer-Verlag, 2002.
- [BM96] J. Barwise and L. Moss. *Vicious Circles: On the Mathematics of Non-Wellfounded Phenomena*. CSLI Publications, Stanford, California, 1996.
- [CDG06] A. Ciaffaglione and P. Di Gianantonio. A certified, corecursive implementation of exact real numbers. *Theoret. Comput. Sci.*, 351(1):39–51, February 2006.
- [CDT06] The Coq Development Team. *The Coq Proof Assistant Reference Manual, Version 8.1*. LogiCal Project, July 2006. <http://coq.inria.fr/v8.1/refman/index.html>, [cited 9 July 2008].
- [CHL03] D. Cancila, F. Honsell, and M. Lenisa. Generalized coiteration schemata. In P. Gumm, editor, *Proc. of 6th Workshop on Coalgebraic Methods in Computer Science, CMCS'03, Warsaw, 5-6 Apr. 2003*, volume 82(1) of *Electron. Notes Theor. Comput. Sci.* Elsevier Science Publishers, Amsterdam, 2003.
- [Cia03] A. Ciaffaglione. *Certified Reasoning on Real Numbers and Objects in Co-inductive Type Theory*. Dottorato di ricerca in informatica, Università di Udine, April 2003.
- [Coq94] T. Coquand. Infinite objects in type theory. In H. Barendregt and T. Nipkow, editors, *Types for Proofs and Programs, International Workshop TYPES'93, Nijmegen, The Netherlands, May 24–28, 1993, Selected Papers*, volume 806 of *Lecture Notes in Comput. Sci.*, pages 62–78. Springer-Verlag, 1994.
- [DDG98] C. Dubois and V. V. Donzeau-Gouge. A step towards the mechanization of partial functions: domains as inductive predicates. In M. Kerber, editor, *Proc. Workshop on Mechanization of Partial Functions, July 5 1998, Lindau, Germany*, pages 53–62, 1998. Available at <ftp://ftp.cs.bham.ac.uk/pub/authors/M.Kerber/98-CADE-WS/dubois-donzeau.ps.gz>, [cited 9 July 2008].
- [DG93] P. Di Gianantonio. *A Functional Approach to Computability on Real Numbers*. Dottorato di ricerca in informatica, Università di Pisa-Genova-Udine, June 1993.
- [DGM03] P. Di Gianantonio and M. Miculan. A unifying approach to recursive and co-recursive definitions. In H. Geuvers and F. Wiedijk, editors, *Types for Proofs and Programs: International Workshop, TYPES 2002, Berg en Dal, The Netherlands, April 24–28, 2002. Selected Papers*, volume 2646 of *Lecture Notes in Comput. Sci.*, pages 148–161. Springer-Verlag, 2003.
- [Dyb00] P. Dybjer. A general formulation of simultaneous inductive-recursive definitions in type theory. *J. Symbolic Logic*, 65(2):525–549, 2000.
- [EH02] A. Edalat and R. Heckmann. Computing with Real Numbers. In G. Barthe, P. Dybjer, L. Pinto, and J. Saraiva, editors, *Applied Semantics: Advanced Lectures*, volume 2395 of *Lecture Notes in Comput. Sci.*, pages 193–267. Springer-Verlag, 2002.
- [EP97] A. Edalat and P. J. Potts. A new representation for exact real numbers. In S. Brookes and M. Mislove, editors, *Mathematical Foundations of Programming Semantics, 13th Annual Conference (MFPS XIII), Pittsburgh, PA, USA, March 23–26, 1997*, volume 6 of *Electron. Notes Theor. Comput. Sci.*, pages 119–132. Elsevier Science Publishers, 1997.
- [ES01] M. H. Escardó and A. K. Simpson. A universal characterization of the closed Euclidean interval. In *Proc. of the 16th Annual IEEE Symposium on Logic in Computer Science*, pages 115–128. IEEE Computer Society, 2001.
- [Esc97] M. H. Escardó. PCF extended with real numbers: A domain-theoretic approach to higher-order exact real number computation. Ph.D. thesis, University of London, Imperial College, 1997.
- [Geu92] H. Geuvers. Inductive and coinductive types with iteration and recursion. In B. Nordström, K. Pettersson, and G. Plotkin, editors, *Informal Proc. of Workshop on Types for Proofs and Programs, Båstad, Sweden, 8–12 June 1992*, pages 193–217. Dept. of Computing Science, Chalmers Univ. of Technology and Göteborg Univ., 1992.
- [GHP06] N. Ghani, P. Hancock, and D. Pattinson. Continuous functions on final coalgebras. In Ghani and Power [GP06].

- [Gib07] J. Gibbons. Metamorphisms: Streaming representation-changers. *Sci. Comput. Program.*, 65(2):108–139, 2007.
- [Gim95] E. Giménez. Codifying guarded definitions with recursive schemes. In P. Dybjer, B. Nordström, and J. Smith, editors, *Types for Proofs and Programs, International Workshop TYPES'94, Båstad, Sweden, June 6–10, 1994, Selected Papers*, volume 996 of *Lecture Notes in Comput. Sci.*, pages 39–59. Springer-Verlag, 1995.
- [Gim96] E. Giménez. *Un Calcul de Constructions Infinies et son Application a la Verification des Systemes Communicants*. PhD thesis PhD 96-11, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, December 1996.
- [Gos72] R. W. Gosper. HAKMEM, Item 101 B. <http://www.inwap.com/pdp10/hbaker/hakmem/cf.html#item101b>, [cited 9 July 2008], February 29 1972. MIT AI Laboratory Memo No. 239.
- [GP06] N. Ghani and J. Power, editors. *Proc. of 8th International Workshop on Coalgebraic Methods in Computer Science, CMCS 2006*, volume 164(1) of *Electron. Notes Theor. Comput. Sci.* Elsevier Science Publishers, 20 October 2006.
- [Hag87] T. Hagino. *A Categorical Programming Language*. PhD thesis CST-47-87, Laboratory for Foundations of Computer Science, Dept. of Computer Science, Univ. of Edinburgh, September 1987.
- [Hec98] R. Heckmann. Big integers and complexity issues in exact real arithmetic. In A. Edalat, A. Jung, K. Keimel, and M. Z. Kwiatkowska, editors, *Third Workshop on Computation and Approximation (Comprox III), Birmingham, England, 11–13 September 1997*, volume 13 of *Electron. Notes Theor. Comput. Sci.* Elsevier Science Publishers, 1998.
- [Hec02] R. Heckmann. Contractivity of linear fractional transformations. *Theoret. Comput. Sci.*, 279(1–2):65–82, May 2002.
- [Hou06] T. Hou. Coinductive proofs for basic real computation. In Beckmann et al. [BBLT06], pages 221–230.
- [Joh02] P. T. Johnstone. *Sketches of an Elephant. A Topos Theory Compendium, vol 2*. Number 44 in Oxford Logic Guides. Oxford University Press, 2002.
- [JR97] B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. *Bull. Eur. Assoc. Theor. Comput. Sci. EATCS*, 62:222–259, 1997.
- [Krz01] M. Krznarić. Computing a required absolute precision from a stream of linear fractional transformations. In J. Blanck, V. Brattka, and P. Hertling, editors, *Computability and Complexity in Analysis: 4th International Workshop, CCA 2000, Swansea, UK, September 17–19, 2000, Selected Papers*, volume 2064 of *Lecture Notes in Comput. Sci.*, pages 169–186. Springer-Verlag, 2001.
- [Len99] M. Lenisa. From set-theoretic coinduction to algebraic coinduction: Some results, some problems. In B. Jacobs and J. Rutten, editors, *Proc. of 2nd Int. Wksh. on Coalgebraic Methods in Computer Science, CMCS'99, Amsterdam, The Netherlands, 20–21 March 1999*, volume 19 of *Electron. Notes Theor. Comput. Sci.* Elsevier Science Publishers, North-Holland, 1999.
- [Les01] D. Lester. Effective continued fractions. In N. Burgess and L. Ciminiera, editors, *15th IEEE Symposium on Computer Arithmetic: ARITH-15 2001: proceedings: Vail, Colorado, 11–13 June, 2001*, pages 163–172. IEEE Computer Society Press, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 2001.
- [Mat99] J. Matthews. Recursive function definition over coinductive types. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Théry, editors, *Theorem Proving in Higher Order Logics, 12th International Conference, TPHOLs'99, Nice, France, September, 1999, Proc.*, volume 1690 of *Lecture Notes in Comput. Sci.*, pages 73–90. Springer-Verlag, 1999.
- [MPC86] N. P. Mendler, P. Panangaden, and R. L. Constable. Infinite objects in type theory. In *Symposium on Logic in Computer Science (LICS '86)*, pages 249–255. IEEE Computer Society Press, Washington, D.C., USA, June 1986.
- [MRE07] J. R. Marcial-Romero and M. H. Escardó. Semantics of a sequential language for exact real-number computation. *Theoret. Comput. Sci.*, 379(1-2):120–141, 2007.
- [Niq04] M. Niqui. *Formalising Exact Arithmetic: Representations, Algorithms and Proofs*. Ph.D. thesis, Radboud Universiteit Nijmegen, September 2004.
- [Niq06] M. Niqui. Coinductive field of exact real numbers and general corecursion. In Ghani and Power [GP06], pages 121–139.

- [Niq07a] M. Niqui. <http://homepages.cwi.nl/~milad/ETrees/coinductive-field/> and http://coq.inria.fr/contribs/Coinductive_Reals.html [cited 9 July 2008], June 2007. Files for Coq 8.1.
- [Niq07b] M. Niqui. Coinductive correctness of homographic and quadratic algorithms for exact real numbers. In T. Altenkirch and C. McBride, editors, *Proc. of TYPES 2006 Workshop*, volume 4502 of *Lecture Notes in Comput. Sci.*, pages 203–220. Springer-Verlag, 2007.
- [Niq07c] M. Niqui. Productivity of Edalat–Potts exact arithmetic in constructive type theory. *Theory Comput. Syst.*, 41(1):127–154, July 2007.
- [Pat03] D. Pattinson. Computable functions on final coalgebras. In *Proc. of 6th Workshop on Coalgebraic Methods in Computer Science, CMCS'03, Warsaw, 5-6 Apr. 2003*, volume 82(1) of *Electron. Notes Theor. Comput. Sci.* Elsevier Science Publishers, 2003.
- [Pau94] L. C. Paulson. A fixedpoint approach to implementing (co)inductive definitions. In A. Bundy, editor, *Automated Deduction – CADE-12, 12th International Conference on Automated Deduction, Nancy, France, June 26 – July 1, 1994, Proc.*, volume 814 of *Lecture Notes in Comput. Sci.*, pages 148–161. Springer-Verlag, 1994.
- [PE98] D. Pavlović and M. H. Escardó. Calculus in coinductive form. In *Proc. of the 13th Annual IEEE Symposium on Logic in Computer Science*, pages 408–417, 1998.
- [PEE97] P. J. Potts, A. Edalat, and M. H. Escardó. Semantics of exact real number arithmetic. In *Proc. of the 12th Annual IEEE Symposium on Logic In Computer Science*, pages 248–257, 1997.
- [PM92] C. Paulin-Mohring. Inductive definitions in the system Coq: Rules and properties. Research report RR 92-49, Laboratoire de l’Informatique du Parallélisme, Ecole normale supérieure de Lyon, December 1992.
- [Pot98] P. J. Potts. *Exact Real Arithmetic using Möbius Transformations*. Ph.D. thesis, University of London, Imperial College, July 1998.
- [PP00] D. Pavlović and V. Pratt. On coalgebra of real numbers. In B. Jacobs and J. Rutten, editors, *Coalgebraic Methods in Computer Science, CMCS'99*, volume 19 of *Electron. Notes Theor. Comput. Sci.*, pages 103–117. Elsevier Science Publishers, 2000.
- [Set06] A. Setzer. Partial recursive functions in Martin-Löf type theory. In Beckmann et al. [BBLT06], pages 505–515.
- [Sim98] A. K. Simpson. Lazy functional algorithms for exact real functionals. In L. Brim, J. Gruska, and J. Zlatuska, editors, *Mathematical Foundations of Computer Science 1998, 23rd International Symposium, MFCS'98, Brno, Czech Republic, August 24–28, 1998, Proc.*, volume 1450 of *Lecture Notes in Comput. Sci.*, pages 456–464. Springer-Verlag, 1998.
- [UV99] T. Uustalu and V. Vene. Primitive (co)recursion and course-of-value (co)iteration, categorically. *Informatica*, 10(1):5–26, 1999.
- [Vui90] J. E. Vuillemin. Exact real computer arithmetic with continued fractions. *IEEE Trans. Comput.*, 39(8):1087–1105, August 1990.
- [Wei00] K. Weihrauch. *Computable Analysis. An introduction*. Springer-Verlag, Berlin Heidelberg, 2000. 285 pp.

APPENDIX A. INTERVAL PREDICATES FOR QUADRATIC ALGORITHM

Let $\xi = \begin{bmatrix} a & b & c & d \\ e & f & g & h \end{bmatrix}$ be a quadratic map and $\phi = \begin{bmatrix} \phi_{00} & \phi_{01} \\ \phi_{00} & \phi_{01} \end{bmatrix} \in \mathbf{DIG}$. Then:

$$\mathbf{Bounded}(\xi) := (0 < e+f+g+h \wedge 0 < e-f-g+h \wedge 0 < -e-f+g+h \wedge 0 < -e+f-g+h \bigvee \\ e+f+g+h < 0 \wedge e-f-g+h < 0 \wedge -e-f+g+h < 0 \wedge -e+f-g+h < 0) ;$$

$$\mathbf{Ref}(\xi) := \mathbf{Bounded}(\xi) \bigwedge$$

$$\left(\begin{aligned} &0 \leq a+b+c+d+e+f+g+h \wedge 0 \leq -a-b-c-d+e+f+g+h \wedge \\ &0 \leq a-b-c+d+e-f-g+h \wedge 0 \leq -a+b+c-d+e-f-g+h \wedge \\ &0 \leq -a-b+c+d-e-f+g+h \wedge 0 \leq a+b-c-d-e-f+g+h \wedge \\ &0 \leq -a+b-c+d-e+f-g+h \wedge 0 \leq a-b+c-d-e+f-g+h \bigvee \\ &a+b+c+d+e+f+g+h \leq 0 \wedge -a-b-c-d+e+f+g+h \leq 0 \wedge \\ &a-b-c+d+e-f-g+h \leq 0 \wedge -a+b+c-d+e-f-g+h \leq 0 \wedge \\ &-a-b+c+d-e-f+g+h \leq 0 \wedge a+b-c-d-e-f+g+h \leq 0 \wedge \\ &-a+b-c+d-e+f-g+h \leq 0 \wedge a-b+c-d-e+f-g+h \leq 0 \end{aligned} \right) ;$$

$$\mathbf{Incl}(\xi, \phi) := \mathbf{Bounded}(\xi) \wedge$$

$$\begin{aligned} &(e-f-g+h)(e-f-g+h)(\phi_{01}-\phi_{00}) \leq (e-f-g+h)(a-b-c+d)(\phi_{11}-\phi_{10}) \wedge \\ &(e-f-g+h)(a-b-c+d)(\phi_{10}+\phi_{11}) \leq (e-f-g+h)(e-f-g+h)(\phi_{00}+\phi_{01}) \wedge \\ &(-e-f+g+h)(-e-f+g+h)(\phi_{01}-\phi_{00}) \leq (-e-f+g+h)(-a-b+c+d)(\phi_{11}-\phi_{10}) \wedge \\ &(-e-f+g+h)(-a-b+c+d)(\phi_{10}+\phi_{11}) \leq (-e-f+g+h)(-e-f+g+h)(\phi_{00}+\phi_{01}) \wedge \\ &(-e+f-g+h)(-e+f-g+h)(\phi_{01}-\phi_{00}) \leq (-e+f-g+h)(-a+b-c+d)(\phi_{11}-\phi_{10}) \wedge \\ &(-e+f-g+h)(-a+b-c+d)(\phi_{10}+\phi_{11}) \leq (-e+f-g+h)(-e+f-g+h)(\phi_{00}+\phi_{01}) \wedge \\ &(e+f+g+h)(e+f+g+h)(\phi_{01}-\phi_{00}) \leq (e+f+g+h)(a+b+c+d)(\phi_{11}-\phi_{10}) \wedge \\ &(e+f+g+h)(a+b+c+d)(\phi_{10}+\phi_{11}) \leq (e+f+g+h)(e+f+g+h)(\phi_{00}+\phi_{01}) . \end{aligned}$$

APPENDIX B. CORRESPONDENCE WITH THE FORMALISED *Coq* FILES.

In the following table we present the correspondence between the terms and lemmas in the article and their formalised version in [Niq07a]. In the second column `foo.bar` refers to the *Coq* term `bar` in file `foo.v` which is available for public download at [Niq07a]. Note that for notations that are overloaded between the homographic and quadratic case, in the first column we explicitly mention an argument. For brevity we drop this argument to the *Coq* functions; e.g. $\mathbf{Bounded}(\mu)$ is in fact formalised as `digits.Bounded_M mu` but we ignore `mu`.

Item in article	Formalised Version
\cong	digits.bisim
Bounded (μ)	digits.Bounded_M
Bounded (ξ)	digits.Bounded_T
Ref (μ)	digits.Is_refining_M
Ref (ξ)	digits.Is_refining_T
DIG	digits.Digit
L	digits.LL
R	digits.RR
M	digits.MM
DIG ^{ω}	digits.Reals
Incl ($\mu, -$)	digits.Incl_M
Incl ($\xi, -$)	digits.Incl_T
◦ (two matrices)	digits.product
◦ (matrix and tensor)	digits.m_product
• ₁	digits.left_product
• ₂	digits.right_product
E_h	homographic.emits_h
\overline{m}_h	homographic.modulus_h
Incl _{dec} ($\mu, -$)	digits.Incl_M_dec_D
E_{hab} -inv	homographic.emits_h_absorbs_inv
Lemma 3.1	homographic.modulus_h_PI
Lemma 3.2.1	homographic.modulus_h_L
Lemma 3.2.2	homographic.modulus_h_R
Lemma 3.2.3	homographic.modulus_h_M
Lemma 3.2.4	homographic.modulus_h_absorbs
Ψ_h	homographic.step_productive_h
P_h	homographic.productive_h
P_h - E_h	homographic.productive_h_emits_h
\overline{m}_h - P_h	homographic.modulus_h_productive_h
Lemma 3.3.1	homographic.step_productive_h_inv_1
Lemma 3.3.2	homographic.step_productive_h_inv_2
\bar{h}	homographic.homographic
Lemma 3.4	homographic.homographic_EPI
Lemma 3.5.1	homographic.homographic_emits_L
Lemma 3.5.2	homographic.homographic_emits_R
Lemma 3.5.3	homographic.homographic_emits_M
Lemma 3.5.4	homographic.homographic_absorbs
E_q	quadratic.emits_q
\overline{m}_q	quadratic.modulus_q
Incl _{dec} ($\xi, -$)	digits.Incl_T_dec_D
E_{qab} -inv	quadratic.emits_q_absorbs_inv
Ψ_q	quadratic.step_productive_q
P_q	quadratic.productive_q
P_q - E_q	quadratic.productive_q_emits_q
\overline{m}_q - P_q	quadratic.modulus_q_productive_q
\bar{q}	quadratic.quadratic

Item in article	Formalised Version
Lemma 3.6	quadratic.quadratic_EPI
Lemma 3.7.1	quadratic.quadratic_emits_L
Lemma 3.7.2	quadratic.quadratic_emits_R
Lemma 3.7.3	quadratic.quadratic_emits_M
Lemma 3.7.4	quadratic.quadratic_absorbs
rep	rep.rep
(4.1)	rep.rep_step1
(4.2)	rep.rep_L_inv
$\alpha _n$	Streams_addenda.drop
$\alpha_{n-1}^{-1} \circ \dots \circ \alpha_0^{-1}$	digits.product_init_rev
(4.3)	rep.rep_drop
$\llbracket _ \rrbracket$	Cauchy_stream.real_value
(4.5)	ub.thesis_5_7_9
(4.6) (lower bound)	Cauchy_stream.rep_lb
(4.6) (upper bound)	Cauchy_stream.rep_ub
(4.7)	rep.rep_inv_interval
(4.8)	Cauchy_stream.real_value_digits
(4.9)	Cauchy_stream.real_value_base_interval
(4.10)	Cauchy_stream.rep_real_value
(4.11)	Cauchy_stream.real_value_implies_rep
(4.12)	Cauchy_stream.rep_implies_real_value
$\bar{\delta}_h$	hcorrectness.depth_h
$\mu \circ \alpha_0 \circ \dots \circ \alpha_{n-1}$	hcorrectness.product_init
Lemma 5.1	hcorrectness.depth_h_modulus_h
Lemma 5.2	hcorrectness.depth_h_Incl_M_inf_strong
Lemma 5.3	hcorrectness.homographic_emits_strong
Lemma 5.4	Refining_M.Incl_M_absorbs_Is_refining_M
Theorem 5.5	hcorrectness.homographic_correctness
$\bar{\delta}_q$	qcorrectness.depth_q
$\xi \langle \alpha_0 \circ \dots \circ \alpha_{n-1}, \beta_0 \circ \dots \circ \beta_{n-1} \rangle$	qcorrectness.product_init_zip
Lemma 5.6	qcorrectness.quadratic_emits_strong
Lemma 5.7	Refining_T.Incl_T_absorbs_Is_refining_T
Theorem 5.8	qcorrectness.quadratic_correctness
Lemma 6.1.1 (\Rightarrow)	Bounded_M.denom_nonvanishing_M_Bounded_M
Lemma 6.1.1 (\Leftarrow)	Bounded_M.Bounded_M_denom_nonvanishing_M
Lemma 6.1.2 (\Rightarrow)	Refining_M.Is_refining_M_property_fold
Lemma 6.1.2 (\Leftarrow)	Refining_M.Is_refining_M_property
Lemma 6.1.3	Incl_M.Incl_M_L_folded
Lemma 6.1.4	Incl_M.Incl_M_R_folded
Lemma 6.1.5	Incl_M.Incl_M_M_folded
Lemma 6.2.1	Refining_M.Is_refining_M_Bounded_M_product
Lemma 6.2.2	Refining_M.Is_refining_M_product
diam (μ)	digits.diameter
red	digits.redundancy
Lemma 6.3	productivity_M.thesis_5_6_9
$\begin{bmatrix} 0 & 1 \\ c & d \end{bmatrix}$ (for $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$)	digits.eta_discriminant

Item in article	Formalised Version
Lemma 6.4.1	productivity_M.diameter_product
Lemma 6.4.2	productivity_M.diameter_product_init
Theorem 6.5	productivity_M.thesis_5_6_10
Lemma 6.6	productivity_M.semantic_modulus_h
Lemma 6.7	productivity_M.Is_refining_M_emits_h
Lemma 6.8	productivity_M.Is_refining_M_depth_h
Lemma 6.9	productivity_M.Is_refining_M_modulus_h
Lemma 6.10	productivity_M.Is_refining_M_step_productive_h
Corollary 6.11	productivity_M.Is_refining_M_productive_h
Lemma 6.12.1 (\Rightarrow)	Bounded_T.denom_nonvanishing_T_Bounded_T
Lemma 6.12.1 (\Leftarrow)	Bounded_T.Bounded_T_denom_nonvanishing_T
Lemma 6.12.2 (\Rightarrow)	Refining_T.Is_refining_T_property_fold
Lemma 6.12.2 (\Leftarrow)	Refining_T.Is_refining_T_property
Lemma 6.12.3	Incl_T.Incl_T_L_folded
Lemma 6.12.4	Incl_T.Incl_T_R_folded
Lemma 6.12.5	Incl_T.Incl_T_M_folded
Lemma 6.13.1	Refining_T. Is_refining_T_Bounded_T_left_right_product
Lemma 6.13.2	Refining_T.Is_refining_T_left_right_product
$\mathbf{diam}_2(\xi)$	digits.diameter2
Lemma 6.14	productivity_T.thesis_5_6_20
Lemma 6.15	productivity_T.upper_bound.diameter2
Corollary 6.16	productivity_T.thesis_5_6_19
Theorem 6.17	productivity_T.thesis_5_6_10'
Lemma 6.18	productivity_T.Is_refining_T_emits_q
Lemma 6.19	productivity_T.Is_refining_T_step_productive_q
Corollary 6.20	productivity_T.Is_refining_T_productive_q