LOGIC FOR GRAY CODE COMPUTATION

ULRICH BERGER AND KENJI MIYAMOTO AND HELMUT SCHWICHTENBERG
AND HIDEKI TSUIKI

Abstract. Gray code is a well-known binary number system such that neighboring values differ in one digit only. Tsuiki (2002) has introduced Gray code to the field of real number computation. He assigns to each number a unique $1\perp$-sequence, i.e., a (possibly infinite) sequence of $\{-1, 1, \perp\}$ such that at most one copy of $\perp$ (meaning undefinedness) is contained in the sequence not as the last character. In this paper we take a logical and constructive approach to study real number computation based on Gray code. Instead of Tsuiki’s indeterministic multihead Type-2 machine, we use pre-Gray code, which is a relaxed Gray code, to avoid the difficulty due to $\perp$ which prevents sequential access to a stream. We extract real number algorithms from proofs in an appropriate formal theory involving inductive and coinductive definitions. Examples are algorithms transforming pre-Gray code into signed digit representations of real numbers, and conversely, the average for pre-Gray code and a bounded translator from pre-Gray code into Gray code. These examples are formalized in the proof assistant Minlog.

Keywords: Gray code, real number computation, inductive and coinductive definitions, program extraction.

2010 Mathematics Subject Classification: 03D78, 03F60, 03B70, 03B35

1. Introduction

Gray code (also called the reflected binary code) is widely known in digital communication, due to its property that the Hamming distance between adjacent Gray codes is always 1. Based on Gray code, Tsuiki has studied an expansion of real numbers as streams of $\{0, 1, \perp\}$ each of which contain at most one $\perp$ standing for undefinedness, and called it the modified Gray expansion. Tsuiki has also studied computability of real numbers, and has presented several algorithms to do real number computation via Gray code. This work was supported by the International Research Staff Exchange Scheme (IRSES) Nr. 612638 CORCON and Nr. 294962 COMPUTAL of the European Commission, the JSPS Core-to-Core Program, A. Advanced research Networks and JSPS KAKENHI Grant Number 15K00015.
The motivation of this paper is to shed light on the logical aspect of Gray code computation from the constructive standpoint. We formalize pre-Gray code in the *Theory of Computable Functionals*, TCF in short, and also in the proof assistant Minlog\(^1\), which is an implementation of TCF, by means of inductive and coinductive definitions [12]. In order to make use of Tsuiki’s idea in TCF, we introduce pre-Gray code which is a modified Gray expansion represented as ordinary streams. Through the realizability interpretation we extract from proofs programs as terms (in an extension \(T^+\) of Gödel’s \(T\)) involving (higher type) recursion and corecursion operators. As case studies we extract real number algorithms in our setting of pre-Gray code. The correctness of the extracted programs is automatically ensured by the soundness theorem.

The rest of this paper is organized as follows. In Section 2 we describe realizability in our framework TCF. Section 3 introduces two representations of real numbers via signed digits as well as via pre-Gray code, respectively. Section 4 describes the domain theory of the term calculus \(T^+\). Section 5 describes inductive and coinductive definitions in TCF, within which we prepare a suitable setting to study logical aspects of signed digit streams and pre-Gray code. Sections 6, 7, 8 and 9 describe case studies. The corresponding Minlog proof script\(^2\) is available in the official Minlog package. In Section 6 we give translators from signed digit streams into pre-Gray code, and vice versa. In Section 7 we study the average of two real numbers in pre-Gray code. In Section 8 we study a bounded translation from pre-Gray code into Gray code. Section 9 describes experiments with our extracted programs. Section 10 concludes the paper.

**Related work.** There are programming languages which can process modified Gray expansion directly. Tsuiki and Sugihara study an extension of Haskell with the non-deterministic choice operator \texttt{gamb} which works as McCarthy’s \texttt{amb} operator [16]. Tsuiki studies a logic programming language with guarded clauses and committed choice [15]. Terayama and Tsuiki study an extension of PCF with parallelism [13]. In this paper we avoid using the above features by adopting pre-Gray code. Concerning stream based real arithmetic, signed digit streams are used to study real number computation by Wiedmer [17, 18]. Its corecursive treatment is studied by Ciaffaglione and Di Gianantonio in Coq [5]. Berger and Seisenberger study program extraction to obtain programs dealing with signed digit streams [3]. Some of

\(^{1}\)See [http://www.minlog-system.de/](http://www.minlog-system.de/)

\(^{2}\)See examples/analysis/gray.scm in the home directory of Minlog.
their results are formalized by Miyamoto and Schwichtenberg in TCF and Minlog [8, 9]. Chuang studies the average and the multiplication of real numbers using coinduction in Agda via the Curry-Howard isomorphism [4].

2. Realizability

We now address (in general terms) the issue of extracting computational content from proofs. The method of program extraction is based on modified realizability as introduced by Kreisel [7] and described in detail in [12]. In short, from every constructive proof $M$ of a non-Harrop formula $A$ (in natural deduction) one extracts a program $\text{et}(M)$ “realizing” $A$, essentially by removing computationally irrelevant parts from the proof (proofs of Harrop formulas have no computational content). The extracted program has some simple type $\tau(A)$ which depends solely on the logical shape of the proven formula $A$. In its original form the extraction process is fairly straightforward, but often leads to unnecessarily complex programs. In order to obtain better programs, proof assistants (for instance Coq, Isabelle/HOL, Agda, Nuprl, Minlog) offer various optimizations of program extraction. Below we describe optimizations implemented in Minlog [11], which are relevant for our present case study.

Quantifiers without computational content. Besides the usual quantifiers, $\forall$ and $\exists$, Minlog has so-called non-computational quantifiers, $\forall^{nc}$ and $\exists^{r}$, which allow for the extraction of simpler programs. These quantifiers, which were first introduced by Berger in [1], can be viewed as a refinement of the Set/Prop distinction in constructive type systems like Coq or Agda. Intuitively, a proof of $\forall^{nc}_x A(x)$ ($A(x)$ non-Harrop) represents a procedure that assigns to any $x$ a proof $M(x)$ of $A(x)$ where $M(x)$ does not make “computational use” of $x$, i.e., the extracted program $\text{et}(M(x))$ does not depend on $x$. Dually, a proof of $\exists^{r}_x A(x)$ is a proof of $M(x)$ for some $x$ where the witness $x$ is “hidden”, that is, not available for computational use (the “r” stands for “right”); in fact, $\exists^{r}$ can be seen as inductively defined by the clause $\forall^{nc}_x (A \rightarrow \exists^{r}_x A)$. The types of extracted programs for non-computational quantifiers are $\tau(\forall^{nc}_x A) = \tau(\exists^{r}_x A) = \tau(A)$ as opposed to $\tau(\forall^{c}_x A) = \rho \rightarrow \tau(A)$ and $\tau(\exists^{c}_x A) = \rho \times \tau(A)$. The extraction rules are, for example in the case of $\forall^{nc}$-introduction and -elimination, $\text{et}((\lambda_x M^{A(x)} A) A) = \text{et}(M)$ and $\text{et}(M^{\forall^{nc}_x A(x) A}) = \text{et}(\lambda_x M^{A(x)} A)$ as opposed to $\text{et}((\lambda_x A^{A(x)}) A) = \text{et}(\lambda_x M)$ and $\text{et}((M^{\forall^{c}_x A(x) A}) A) = \text{et}(Mt)$. For the extracted programs to be correct the variable condition at $\forall^{nc}$-introduction must be strengthened by requiring in addition the abstracted variable $x$ not to occur in the extracted program.
et(M), and similarly for $\exists^r$. Note that for a Harrop formula $A$ the formulas $\forall^ncxA$, $\forall_xA$ and also $\exists^r_xA$, $\exists_xA$ are equivalent.

**Animation.** Suppose a proof of a theorem uses a lemma. Then the proof term contains just the name of the lemma, say $L$. In the term extracted from this proof we want to preserve the structure of the original proof as much as possible, and hence use a new constant $cL$ at those places where the computational content of the lemma is needed. When we want to execute the program, we have to replace the constant $cL$ corresponding to a lemma $L$ by the extracted program of its proof. This can be achieved by adding computation rules for $cL$. We can be rather flexible here and enable/block rewriting by using animate/deanimate as desired.

### 3. Gray code of real number

We define binary expansion as the expansion of the unit\(^3\) interval $I = [-1, 1]$ as infinite sequences of $PSD = \{-1, 1\}$ (proper signed digits) so that $p = a_0a_1a_2\ldots$ represents

$$\sum_{i=1}^{\infty} \frac{a_{i-1}}{2^i}. \tag{1}$$

With binary expansion, a finite sequence $a_0\ldots a_{n-1}$ denotes the interval $C_{a_0}(C_{a_1}(\ldots(C_{a_{n-1}}I)\ldots))$ for $C_a$ the function $C_a(x) = \frac{x + a}{2}$, and $a_0a_1\ldots$ denotes the limit of the intervals denoted by its finite truncations. Signed digit expansion (see Figure 1) is an expansion of the same interval with the same formula (1), but with three digits $SD = \{-1, 0, 1\}$. Note that it has a lot of redundancy, and $11$ and $01$ represent the same interval $[-1/2, 0]$ and $11$ and $01$ represent the same interval $[0, 1/2]$. Here, $I$ is the notation of $-1$ in a sequence. We view finite sequences of $SD$ as a free algebra $I$ with a nullary constructor $I$ and three unary constructors $C_{-1}$, $C_0$, $C_1$ of type $I \rightarrow I$.

Gray expansion is another way of expanding $I$ with $PSD$ which is based on binary reflected Gray code of natural numbers. With Gray expansion, the sequence is flipped after an appearance of 1. That is, we define the functions $LR_a$ for $a \in PSD$ as $LR_a(x) = -\frac{x + a}{2}$ so that $LR_{-1} = C_{-1}$ but $LR_1(x) = C_1(-x)$. The number represented by a sequence $p = a_0a_1a_2\ldots$ is the limit of the shrinking intervals $LR_{a_0}(LR_{a_1}(\ldots(LR_{a_{n-1}}I)\ldots))$ which is

\(^3\)For simplicity we base our study on $[-1, 1]$ rather than $[0, 1]$.\)
equal to
\[
\sum_{i=1}^{\infty} \frac{\prod_{j<i} (-a_j)}{2^i}.
\]

With Gray expansion, each dyadic rational number (i.e., \(k/2^i\) for integers \(-2^i \leq k \leq 2^i\)) is represented in two ways as is the case for the binary expansion. For example, 0 is expanded as \(\bar{1}1\bar{1}\omega\) and \(1\bar{1}\omega\). However, the two expansions differ only at one digit and the sequence after the digit they differ is always \(1\bar{1}\bar{1}\omega\).

In [14] a modified Gray expansion (see Figure 2) was defined, which assigns a unique \(L\)-sequence to each number. Here, a \(L\)-sequence is a (possibly infinite) sequence of \(\{-1, 1, \bot\}\) such that at most one copy of \(\bot\) (meaning undefinedness) is contained in the sequence not as the last character. It assigns the \(L\)-sequence \(s\bot1\omega\) to a dyadic rational number with the Gray expansions \(\bar{1}1\omega\) and \(1\bar{1}\omega\) for \(s \in \{\bar{1}, 1\}^*\). In this note, we consider its variant that assigns all the three sequences \(\bar{1}1\omega\), \(\bar{1}\bar{1}\omega\), and \(s\bot1\omega\) to it and call it the modified Gray expansion. A \(L\)-sequence \(s \in \{\bar{1}, 1\}^*\) denotes the same interval as in the Gray expansion. The meaning of a \(L\)-sequence \(s\bot1\omega\) for \(s \in \{\bar{1}, 1\}^*\) is the union of the two intervals denoted by \(s\bar{1}1\bar{1}\omega\) and
Note that $\perp$ is not an ordinary character and a machine cannot read or write a $\perp$ on a tape. In [14] an IM2-machine (indeterministic multihead Type-2 machine) was introduced: it has two heads on each input/output tape so that it can skip a $\perp$ and access the rest the sequence. Here, instead of such a direct manipulation of $1\perp$-sequences, we define pre-Gray code which is a “representation” of $1\perp$-sequences as sequences of constructors, and consider computation through usual stream programs instead of IM2-machines. One can consider pre-Gray code as the signed digit expansion of Gray-code.

Before explaining the algebra of pre-Gray code, we define the algebra of $1\perp$-sequences. An ordinary binary sequence is generated by two constructors $\text{cons}_a$ for $a \in \{-1, 1\}$ which prepend $a$ to a sequence. On the other hand, a $1\perp$-sequence is generated by, in addition to $\text{cons}_a$ for $a \in \{-1, 1\}$, two constructors $\text{ins}_a$ for $a \in \{-1, 1\}$ which insert $a$ as the second character to a sequence. For example, both $\text{ins}_1(\text{ins}_{-1}(\text{cons}_{-1}(\text{ins}_1([]))))$ and $\text{cons}_{-1}(\text{cons}_1(\text{cons}_{-1}(\text{ins}_1([]))))$ denote $111\perp1$, and $\text{ins}_1(\text{ins}_{-1}(\text{ins}_{-1}(\text{ins}_{-1} \ldots )))$ denotes $1\perp^\omega$. Here, a finite $1\perp$-sequence is identified with an infinite sequence of $\{1, -1, \perp\}$ by appending $\perp^\omega$ to the end of the sequence. If we read

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{gray_expansion.png}
\caption{Modified Gray expansion and pre-Gray code.}
\end{figure}
these sequences of applications of the four operations from left to right, then
we can consider them as procedures to construct 1⊥-sequences as follows.
We start with an infinite tape with the state 1⊥ω, and view consa as the operation
to fill the leftmost 1⊥ with a and insa as the operation to fill the second
1⊥ from the left with a. For example, according to ins1(ins−1(ins1(cons−1(ins1()))),
we construct 111⊥ as 111⊥ → 111⊥ → 111⊥ → 111⊥.

The set of finite 1⊥-sequences used for modified Gray expansion is {1, 1}∗ ∪
{1, 1} * 1⊥. In order to represent only this set, we define an algebra G of
pre-Gray code simultaneously with an auxiliary algebra H. Intuitively, the
algebra G corresponds to the right end of sequences in {1, 1}∗ and the
algebra H corresponds to the right end of sequences in {1, 1} * 1⊥. Both
algebras have constructors of the same argument types. The algebra G is
given by one nullary constructor I_G of type G and three unary constructors
L, R of type G → G and U of type H → G (for “undefined”), which
correspond to cons−1, cons1, and ins1, respectively. The algebra H has a
nullary constructor I_H of type H and three unary constructors FinL and
FinR of type G → H (representing a final L or R after delays) and D of
type H → H (for delay), which correspond to cons−1, cons1, and ins−1, re-
spectively. Thus, both U(D(FinL(U(I_H)))) and L(R(L(U(I_H)))) represent
the sequence 111⊥.

We define the meanings of the constructors of pre-Gray code as follows.
II_G and I_H denote the unit interval I. The constructors L, R, FinL, FinR, D,
and U denote the functions LR1, LR1, Fin1, Fin1, D, and U on I defined
as follows:

\[ LR_a(x) = -\frac{x - 1}{2}, \quad D(x) = \frac{x}{2}, \]
\[ Fin_a(x) = \frac{x + 1}{2}, \quad U(x) = \frac{x}{2}. \]

Note that the following equations hold:

\[ U \circ Fin_a = LR_a \circ LR_1, \]
\[ D \circ Fin_a = Fin_a \circ LR_1. \]

4. Algebras and their total and cototal ideals

Constructor expressions in our free algebras I, G and H can be seen as
standard rational intervals. To treat reals as limits of rational intervals we
view them as “cototal ideals”. For the convenience of the reader we repeat
here some material from [9].
Rather than working with algebras and coalgebras in a categorical setting (as for instance done in [3]), we just use (free) algebras to generate the basic domains of the Scott-Ershov model of partial continuous functionals. Among the ideals of such a domain we single out the total and cototal ones, which are our well-founded and non-well-founded objects, respectively. We construct domains by information systems, given by a set of tokens, a set of finite consistent sets of tokens, and an entailment relation. As an example, consider the algebra \( I \) of standard intervals introduced above, and let \( C \) range over its constructors. The following definitions are an adaption of the more general ones in [12] to the present case.

(a) The tokens \( a \) are the type correct constructor expressions (or trees) \( Ca^*_1 \ldots a^*_n \) where \( a^*_i \) is an extended token, i.e., a token or the special symbol \( * \) which carries no information.

(b) A finite set \( U \) of tokens is consistent if all its elements start with the same constructor \( C \), say of arity \( \tau_1 \rightarrow \ldots \rightarrow \tau_n \rightarrow I \), and all \( U_i \) are consistent \((i = 1, \ldots, n)\), where \( U_i \) consists of all (proper) tokens at the \( i \)-th argument position of some token in \( U = \{ Ca^*_1, \ldots, Ca^*_m \} \).

(c) \( \{ Ca^*_1, \ldots, Ca^*_m \} \vdash C'a^* \) (“entails”) is defined to mean \( C = C' \), \( m \geq 1 \) and \( U_i \vdash a^*_i \), with \( U_i \) as in (b) above (and \( U \vdash \ast \) taken to be true).

These are definitions by recursion on the height (of the syntactic expressions involved), defined by

\[
|Ca^*_1 \ldots a^*_n| := 1 + \max\{ |a^*_i| \mid i = 1, \ldots, n \}, \quad |*| := 0,
\]
\[
|\{ a_i \mid i \in I \}| := \max\{ 1 + |a_i| \mid i \in I \},
\]
\[
|U \vdash a| := \max\{1 + |U|, 1 + |a|\}.
\]

A set of tokens is deductively closed if it contains all tokens entailed from one of its finite subsets. An ideal is defined to be a (possibly infinite) consistent and deductively closed set of tokens. The intuition is that a finite consistent set \( U \) of tokens is seen as a “formal neighborhood” (Kreisel [7]) in a space of abstract points or ideals.

To define total and cototal ideals, consider a constructor expression \( P(*) \) with a distinguished occurrence of \( \ast \). Then an arbitrary \( P(Ca^\ast) \) is called one-step extension of \( P(*) \), written \( P(Ca^\ast) \succ_1 P(*) \). For example, writing \( C_d^na \) for \( C_d(C_d(\ldots(C_d(a)\ldots)) \), for \( P(*) := C_{-1}C_{1}^n \ast \) we have \( P(C_1\ast) = C_{-1}C_{1}^n \ast \succ_1 C_{-1}C_{2}^n \ast = P(*) \). An ideal \( x \) is called cototal if every constructor expression \( P(*) \in x \) has a one-step extension \( P(Ca^\ast) \in x \); it is called total if it is cototal and the relation \( \succ_1 \) on \( x \) is well-founded.
In our algebras for pre-Gray code every total ideal then can be seen as a standard interval
\[ I_{i,2^{-k},k} := \left[ \frac{i}{2^k}, \frac{i}{2^k} + \frac{1}{2^k} \right] \quad \text{for } -2^k < i < 2^k \]
whereas the cototal ideals are what we mean by a “stream representation” of reals. For instance, the cototal ideals include \( \{ C_{-1}^n \mid n \geq 0 \} \), a stream representation of the real \(-1\), and also \( \{ C_1 C_{-1}^n \mid n \geq 0 \} \) and \( \{ C_{-1} C_1^n \mid n \geq 0 \} \), which both represent the real 0. Generally, the cototal ideals give us all reals in \([-1, 1]\), in the (non-unique) stream representation via signed digits \(-1, 0, 1\).

Remark 4.1. Notice that the notion of a cototal ideal also makes sense when the underlying algebra does not have nullary constructors. Since we will only be concerned with cototal ideals we take advantage of this fact and from now on omit the nullary constructor \( I \) from our algebra \( I \), and also the nullary constructors \( I_G \) and \( I_H \) from our algebras \( G \) and \( H \). This will simplify the arguments below considerably (for instance in comparison with [9]). We also redefine our algebras \( I, G, H \) so that \( I \) has a binary constructor \( C \) of type \( SD \rightarrow I \rightarrow I \) and \( G \) has a binary constructor \( LR \) of type \( PSD \rightarrow G \rightarrow G \) and a unary constructor \( U \) of type \( H \rightarrow G \); the intention is \( L = LR(\bar{1}, \cdot) \) and \( R = LR(1, \cdot) \). Similarly \( H \) has a binary constructor \( Fin \) of type \( PSD \rightarrow G \rightarrow H \) and a unary constructor \( D \) of type \( H \rightarrow H \).

5. Inductive and coinductive definitions

Recall that our goal is to extract algorithms for real number computation from proofs in an appropriate formal theory involving inductive and coinductive definitions. In particular we want to transform reals in Gray code representation to reals in signed digit representation. Now how should the formal theory deal with real numbers? Here we adopt a proposal by Berger and Seisenberger [3]: one should use an abstract theory of reals. But then the immediate question is: how could one associate computational content with a formula \( \forall x \ldots \) where \( x \) ranges over abstract reals? By the very idea of abstractness we do (and should) not know the type of \( x \); in fact, it may be a type variable. Therefore the quantifier \( \forall x \) should have no computational significance. Following [2] we use a “non-computational” universal quantifier \( \forall^{nc}_x \) instead, and move the computational content of our assumption into a predicate, i.e., we consider \( \forall^{nc}_x (Qx \rightarrow \ldots) \) instead. This leaves the exact form of realizers for \( Qx \) and hence the data type we use for representing real numbers open. Now, how can we achieve that \( Qx \) has cototal ideals of
certain algebras (like \(I, G\) or \(H\)) as realizers? An obvious way is to take for \(Q\) an appropriate coinductively defined predicate.

For example, define inductively a set \(I\) of (abstract) reals, by the single clause

\[
\forall x \forall_d \left( Ix \rightarrow \frac{Ix + d}{2} \right)
\]

\((d,e \text{ range over } \text{SD}).\) Let \(\Phi\) be the corresponding operator (or predicate transformer):

\[
\Phi(X) := \{ \frac{x + d}{2} \mid d \in \text{SD}, x \in X \}
\]

or more precisely (i.e., using a decorated existential quantifier \(\exists^r\))

\[
y \in \Phi(X) \leftrightarrow \exists^r_{x \in X} \exists^r_d (y = \frac{x + d}{2})
\]

\((\exists^r_x\) indicates that the existentially quantified variable \(x\) is disregarded in the realizability interpretation; cf. Section 2).

For \(I\) we can (as for any inductive predicate) define its “companion” predicate \(\compl I\), written \(\compl I = \nu_X \Phi(X)\). This is understood as the greatest fixed point of \(\Phi\), expressed by the greatest-fixed-point (or coinduction) axiom \(X \subseteq \Phi(\compl I \cup X) \rightarrow X \subseteq \compl I\). A witness for a proposition \(\compl Ir\) is a stream of signed digits. Such objects can be seen as cototal ideals for the algebra \(I\).

Similarly we give a simultaneous inductive definition of the sets \(G\) and \(H\) of (abstract) reals. Let \(a,b\) be variables ranging over \(\text{PSD}\).

The clauses are

\[
\forall x \forall_a (Gx \rightarrow G(-\frac{x - 1}{2})), \quad \forall x \forall_a (Hx \rightarrow G(\frac{x}{2})),
\]

\[
\forall x \forall_a (Gx \rightarrow H(\frac{x + 1}{2})), \quad \forall x \forall_a (Hx \rightarrow H(\frac{x}{2})).
\]

The corresponding operators \(\Gamma, \Delta\) are defined by

\[
y \in \Gamma(X,Y) \leftrightarrow \exists^r_{x \in X} \exists^r_a (y = -a \frac{x - 1}{2}) \lor \exists^r_{x \in Y} (y = \frac{x}{2}),
\]

\[
y \in \Delta(X,Y) \leftrightarrow \exists^r_{x \in X} \exists^r_a (y = a \frac{x + 1}{2}) \lor \exists^r_{x \in Y} (y = \frac{x}{2})
\]

and we have \((\compl G, \compl H) = \nu_{(X,Y)} (\Gamma(X,Y), \Delta(X,Y)).\) This is understood as the greatest fixed point of \((\Gamma, \Delta)\), expressed by the simultaneous greatest-fixed-point (or simultaneous coinduction) axiom

\((X,Y) \subseteq (\Gamma(\compl G \cup X, \compl H \cup Y), \Delta(\compl G \cup X, \compl H \cup Y)) \rightarrow (X,Y) \subseteq (\compl G, \compl H)\),

where inclusion \(\subseteq\) is meant component-wise. Witnesses for \(\compl Gr\) and \(\compl Hr\) are cototal ideals in the algebra \(G\) or \(H\), respectively.
Remark 5.1 (Nested definition). As an alternative to the above simultaneous definition of $^\circ G$ and $^\circ H$, we can take the nested definition $^\circ G' = \nu_X \Gamma (X, ^\circ H' (X))$ where $^\circ H'_X = \nu_Y \Delta (X, Y)$. The witnessing algebras would also be changed. In this paper we adopt the simultaneous one, since the extracted programs are simpler and more efficient.

6. Proofs and their extracted terms

Now we can formulate the propositions to be proved. The computational content will be the algorithms we are interested in. In each case we sketch an informal proof, and then display the term extracted from a formalization of this proof.

We will make use of the following axioms on the abstract real numbers. All these axioms will be (Harrop formulas and hence) without computational content. Their only computational effect will be the way in which they influence the proofs of the propositions below. The axioms are

$$\forall x(1x = x), \quad \forall x((-1)x = -x), \quad \forall a,x,d \left( a \frac{x + d}{2} = \frac{ax + ad}{2} \right).$$

From these axioms we can easily prove

$$a(bx) = (ab)x, \quad -x = x, \quad -\frac{x + d}{2} = -\frac{x - d}{2},$$

$$-\frac{x - d}{2} = -\frac{x + d}{2}, \quad (-a)x = -(ax), \quad a(-x) = -(ax).$$

When formalizing the proofs below it will be convenient to use all these equations as rewrite rules (from left to right).

We first want to prove $^\circ G \subseteq ^\circ I$. However, to be able to prove this by coinduction we need to generalize our goal to

**Proposition 6.1** (CoGToCoIGen). $\forall x (\exists_a (^\circ G(ax) \lor ^\circ H(ax)) \rightarrow ^\circ I x)$.

**Proof.** For $X := \{ x \mid \exists_a (^\circ G(ax) \lor ^\circ H(ax)) \}$ we must show $X \subseteq ^\circ I$. By coinduction it suffices to prove $X \subseteq \Phi (^\circ I \cup X)$. Let $x_1 \in X$. We show $x_1 \in \Phi (^\circ I \cup X)$:

$$\exists x \in ^\circ I \cup X. \exists d (x_1 = \frac{x + d}{2}).$$

Since $x_1 \in X$ we have a such that $^\circ G(ax_1) \lor ^\circ H(ax_1)$.

**Case** $^\circ G(ax_1)$. The $^\circ G$-clause $^\circ G \subseteq \Gamma (^\circ G, ^\circ H)$ applied to $ax_1 \in ^\circ G$ gives us

$$\exists x \in ^\circ G. \exists b (ax_1 = -bx - \frac{1}{2}) \lor \exists x \in ^\circ H. (ax_1 = \frac{x}{2}).$$
If the left hand side holds, we have \( x_2 \in \text{co}G \) and \( b \) such that \( ax_1 = -bx_2 - \frac{1}{2} \). Then (4) holds for \( x := -abx_2 \) and \( d := ab \), since \(-abx_2 \in X \) (by \( x_2 \in \text{co}G \) and the definition of \( X \)), and

\[
x_1 = a^2x_1 = -ab\frac{x_2 - 1}{2} = \frac{-abx_2 + ab}{2} = \frac{x + d}{2}.
\]

If the right hand side holds, we have \( x_2 \in \text{co}H \) such that \( ax_1 = \frac{x_2}{2} \). Then (4) holds for \( x := ax_2 \) and \( d := 0 \), since \( ax_2 \in X \) (by \( x_2 \in \text{co}H \) and the definition of \( X \)), and

\[
x_1 = a^2x_1 = a\frac{x_2}{2} = \frac{ax_2 + 0}{2} = \frac{x + d}{2}.
\]

**Case \( \text{co}H(ax_1) \).** The \( \text{co}H \)-clause \( \text{co}H \subseteq \Delta(\text{co}G, \text{co}H) \) applied to \( ax_1 \in \text{co}H \) gives

\[
\exists x \in \text{co}G \exists b(ax_1 = b \frac{x + 1}{2}) \lor \exists x \in \text{co}H(ax_1 = \frac{x}{2}).
\]

If the left hand side holds, we have \( x_2 \in \text{co}G \) and \( b \) such that \( ax_1 = \frac{bx_2 + 1}{2} \). Then (4) holds for \( x := abx_2 \) and \( d := ab \), since \( abx_2 \in X \) (by \( x_2 \in \text{co}G \) and the definition of \( X \)), and

\[
x_1 = a^2x_1 = ab\frac{x_2 + 1}{2} = \frac{abx_2 + ab}{2} = \frac{x + d}{2}.
\]

If the right hand side holds, we have \( x_2 \in \text{co}H \) such that \( ax_1 = \frac{x_2}{2} \). Then (4) holds for \( x := ax_2 \) and \( d := 0 \), since \( ax_2 \in X \) (by \( x_2 \in \text{co}H \) and the definition of \( X \)), and

\[
x_1 = a^2x_1 = a\frac{x_2}{2} = \frac{ax_2 + 0}{2} = \frac{x + d}{2}.
\]
To understand this term recall the type of the corecursion operator for $I$:

$$(5) \quad \text{co}R^*_\tau I : \tau \rightarrow (\tau \rightarrow \text{SD} \times (I + \tau)) \rightarrow I \quad \text{with} \quad \tau := \text{PSD} \times (G + H).$$

The type $\text{SD} \times (I + \tau)$ appears since $I$ has the single constructor $C$ of type $\text{SD} \rightarrow I \rightarrow I$. The meaning of $\text{co}R^*_\tau I$ is defined by the conversion rule $\text{co}R^*_\tau I NM \mapsto C(ab, \text{inr}(ab, \text{inl}(p)))$.

Let us analyze the particular step function $M$ above, extracted from our proof. When applied to an argument $N$ of type $\tau = \text{PSD} \times (G + H)$, $M$ returns a result of type $\text{SD} \times (I + \tau)$; it will be in the right part of $I + \tau$ (i.e., here we do not use the fact that our coinductive definitions are in “strengthened” form). Consider the right hand side $N_2$ of $N$, of type $G + H$.

Case 1. If $N_2$ is of the form $\text{inl}(p)$ with $p$ of type $G$, destruct $p$. Recall that $G$ has two constructors, LR and U. If $p$ is of the form $\text{LR}_b(p')$, the result is $(ab, \text{inr}(-ab, \text{inl}(p')))$. If $p$ is of the form $\text{U}(q)$, the result is $(0, \text{inr}(a, \text{inr}(q)))$.

Case 2. If $N_2$ is of the form $\text{inr}(q)$ with $q$ of type $H$, destruct $q$. Recall that $H$ has two constructors, Fin and D. If $q$ is of the form $\text{Fin}_b(p)$, the result is $(ab, \text{inr}(ab, \text{inl}(p)))$. If $q$ is of the form $\text{D}(q')$, the result is $(0, \text{inr}(a, \text{inr}(q')))$. Hence we can write $\lambda y \text{co}R^*_\tau yM$ as a function

$$[f, g] : \text{PSD} \times G + \text{PSD} \times H \rightarrow I$$

defined by

$$f(a, \text{LR}_b(p)) = C_{ab}(f(-ab, p)), \quad g(a, \text{Fin}_b(p)) = C_{ab}(f(ab, p)),
\quad f(a, \text{U}(q)) = C_0(g(a, q)), \quad g(a, \text{D}(q)) = C_0(g(a, q)).$$

**Proposition 6.2** (CoGToCoI). $\forall x \forall y \forall a \forall b (coG(ax) \rightarrow coI x)$.

**Extracted term.** $[a, p] \text{CoGToCoI} \text{Gen}(a@I \text{N} p)$

**Remark.** Before proving Proposition 6.1 and obtaining the extracted term, we wrote a Haskell program to convert pre-Gray code to signed digit code,
which consisted of 12 rules corresponding to the cases of $a$ and $b$. Its correctness is not proved, and moreover, we did not realize that it can be compressed into 4 rules until we had the extracted term.

**Proposition 6.3** (CoIToCoG).

\[
\forall x^\mathsf{ac}\ (\exists a^\mathsf{co}(ax) \rightarrow \mathsf{co}Gx),
\forall x^\mathsf{ac}\ (\exists a^\mathsf{co}(ax) \rightarrow \mathsf{co}Hx).
\]

**Proof.** For $X := \{ x \mid \exists a(ax \in \mathsf{co}) \}$ we show $X \subseteq \mathsf{co}G$ simultaneously with $X \subseteq \mathsf{co}H$. By coinduction it suffices to prove (i) $X \subseteq \Gamma(\mathsf{co}G \cup X, \mathsf{co}H \cup X)$ and (ii) $X \subseteq \Delta(\mathsf{co}G \cup X, \mathsf{co}H \cup X)$. For (i), let $x_1 \in X$. We show $x_1 \in \Gamma(X)$:

\[
\exists r^\mathsf{co}G \exists a^\mathsf{co}(x_1 = \frac{x - 1}{2}) \vee \exists r^\mathsf{co}H \exists a^\mathsf{co}(x_1 = \frac{x}{2}).
\]

Since $x_1 \in X$ we have $a_1$ such that $\mathsf{co}I(a_1x_1)$. The $\mathsf{co}I$-clause $\mathsf{co}I \subseteq \Phi(\mathsf{co}I)$ applied to $a_1x_1 \in \mathsf{co}I$ gives us

\[
\exists r^\mathsf{co}G \exists a^\mathsf{co}(x_1 = \frac{x + d}{2}).
\]

Hence we have $x_2 \in \mathsf{co}I$ and $d$ such that $a_1x_1 = \frac{x_2 + d}{2}$.

**Case** $d = -1$. Then the left hand of (6) holds for $x := x_2$ and $a := -a_1$, since $x_2 \in X$ (by $x_2 \in \mathsf{co}I$ and the definition of $X$), and

\[
x_1 = a_1a_1x_1 = a_1 \frac{x_2 + d}{2} = a_1 \frac{x_2 - 1}{2}.
\]

**Case** $d = 1$. Then the left hand of (6) holds for $x := -x_2$ and $a := a_1$, since $-x_2 \in X$ (by $x_2 \in \mathsf{co}I$ and the definition of $X$), and

\[
x_1 = a_1a_1x_1 = a_1 \frac{x_2 + d}{2} = a_1 \frac{x_2 + 1}{2} = a_1 \frac{-x_2 - 1}{2}.
\]

**Case** $d = 0$. Then the right hand of (6) holds for $x := a_1x_2$, since $a_1x_2 \in X$ (by $x_2 \in \mathsf{co}I$ and the definition of $X$), and

\[
x_1 = a_1a_1x_1 = a_1 \frac{x_2 + d}{2} = a_1 \frac{x_2}{2}.
\]

This finishes the proof of (i). The proof of (ii) is similar, and we omit it. \qed

**Extracted term.** We use $bv$ as variable of type $\mathsf{PSD} \times \mathsf{I}$.

\[
[bv](\text{CoRec psd0@iv=>ag psd0@id=>ah} )bv
\]

\[
([bv0][\text{case (left Des right bv0)}]
\begin{align*}
\text{Lft} & \rightarrow \text{InL}(\text{inv left bv0@InR(PRht@right Des right bv0)})) \\
\text{Rht} & \rightarrow \text{InL}(\text{left bv0@InR(PLft@right Des right bv0)}))
\end{align*}
\]
(Mid -> InR(InR(left bv0@right Des right bv0))))
([bv0][case (left Des right bv0)
  (Lft -> InL(inv left bv0@InR(PLft@right Des right bv0)))
  (Rht -> InL(left bv0@InR(PRht@right Des right bv0)))
  (Mid -> InR(InR(left bv0@right Des right bv0))))]

To understand this term recall the type of the simultaneous corecursion
operators $\text{co}R_G(G, H), (\tau, \tau)$ and $\text{co}R_H(G, H), (\tau, \tau)$, or shortly $\text{co}R_G$ and $\text{co}R_H$:

\[
\text{co}R_G : \tau \to \delta \to \delta \to G
\]

\[
\text{co}R_H : \tau \to \delta \to \delta \to H
\]

with $\tau := PSD \times I$ and step types $\delta := \tau \to PSD \times (G + \tau) + (H + \tau)$. The
type $PSD \times (G + \tau) + (H + \tau)$ appears since $G$ has the two constructors
LR: $PSD \to G \to G$ and U: $H \to G$, and $H$ has the two constructors
Fin: $PSD \to G \to H$ and D: $H \to H$. The meaning of $\text{co}R_G N M M'$ and $\text{co}R_H N M M'$
is defined by the conversion rules

\[
\text{co}R_G N M M' \mapsto \begin{cases}
  LR_{u_1}([\text{id}^G \to G, \lambda_y (\text{co}R_G y M M')|u_2] & \text{if } M N = \text{inl}(u) \\
  U([\text{id}^H \to H, \lambda_y (\text{co}R_H y M M')]|v) & \text{if } M N = \text{inr}(v)
\end{cases}
\]

\[
\text{co}R_H N M M' \mapsto \begin{cases}
  \text{Fin}_{u_1}([\text{id}^G \to G, \lambda_y (\text{co}R_G y M M')]|u_2) & \text{if } M' N = \text{inl}(u) \\
  D([\text{id}^H \to H, \lambda_y (\text{co}R_H y M M')]|v) & \text{if } M' N = \text{inr}(v)
\end{cases}
\]

We again analyze the particular step functions $M, M'$ extracted from our
proof. When applied to an argument $N$ of type $\tau = PSD \times I$, $M$ returns a
result of type $PSD \times (G + \tau) + (H + \tau)$, in the right part of $G + \tau$ or $H + \tau$.
Let $N = (b, v)$ with $v$ of type $I$, of the form $C_d(v')$. The result is

\[
inl(-b, \text{inr}(1, v')) \text{ if } d = -1,
\]

\[
inl(b, \text{inr}(-1, v')) \text{ if } d = 1,
\]

\[
inl(\text{inr}(b, v')) \text{ if } d = 0.
\]

Similarly, when applied to an argument $N$ of type $\tau = PSD \times I$, $M'$ returns
a result of type $PSD \times (G + \tau) + (H + \tau)$. Let $N = (b, v)$ with $v$ of type $I$, of the form $C_d(v')$. The result is

\[
inl(-b, \text{inr}(-1, v')) \text{ if } d = -1,
\]

\[
inl(b, \text{inr}(1, v')) \text{ if } d = 1,
\]

\[
inl(\text{inr}(b, v')) \text{ if } d = 0.
\]
Hence we can write the two functions $\lambda_y^{co\mathcal{R}_G} y M'$ and $\lambda_y^{co\mathcal{R}_H} y M'$ as $g: \tau \to G$ and $h: \tau \to H$ defined by

\[
\begin{align*}
g(b, C_{-1}(v)) &= LR_b(g(1, v)), & h(b, C_{-1}(v)) &= Fin_b(g(-1, v)), \\
g(b, C_1(v)) &= LR_b(g(-1, v)), & h(b, C_1(v)) &= Fin_b(g(1, v)), \\
g(b, C_0(v)) &= U(h(b, v)), & h(b, C_0(v)) &= D(h(b, v)).
\end{align*}
\]

We add another example, which uses the fact that our coinduction axioms are in “strengthened” form, i.e., $X \subseteq \Phi(coI \cup X) \to X \subseteq coI$ instead of $X \subseteq \Phi(X) \to X \subseteq coI$, for example.

**Proposition 6.4 (CoGMinus).**

\[
\forall x^{coG}(-x) \to coGx, \\
\forall x^{coH}(-x) \to coHx.
\]

**Proof.** For $X := \{ x \mid -x \in coG \}$ and $Y := \{ x \mid -x \in coH \}$ we show $X \subseteq coG$ simultaneously with $Y \subseteq coH$. By coinduction it suffices to prove (i) $X \subseteq \Gamma(coG \cup X, coH \cup Y)$ and (ii) $Y \subseteq \Delta(coG \cup X, coH \cup Y)$. For (i), let $x_1 \in X$. We show $x_1 \in \Gamma(coG \cup X, coH \cup Y)$:

\[
(7) \exists x_2^{coG \cup Y} \exists a^{coH}(x_1 = -a \frac{x-1}{2} \lor \exists x_2^{coH}(x_1 = \frac{x}{2}).
\]

The $coG$-clause applied to $-x_1 \in coG$ gives us

\[
\exists x_2^{coG} \exists a(-x_1 = -a \frac{x-1}{2} \lor \exists x_2^{coH}(-x_1 = \frac{x}{2}).
\]

In the first case we have $x_2 \in coG$ and $a$ with $-x_1 = -a \frac{x-1}{2}$. Then the left hand side of (7) holds for $x_2$ and $-a$ (here we use that our coinduction axiom is in strengthened form). In the second case we have $x_2 \in coH$ with $-x_1 = \frac{x}{2}$. Then the right hand side of (7) holds for $-x_2$. This finishes the proof of (i). The proof of (ii) is similar, and we omit it. $\Box$
Recall the types of the simultaneous corecursion operators $^{co}R^G_{(G,H),(\sigma,\tau)}$ and $^{co}R^H_{(G,H),(\sigma,\tau)}$

\[
^{co}R^G_{(G,H),(\sigma,\tau)} : \sigma \to \delta_G \to H \to G
\]

\[
^{co}R^H_{(G,H),(\sigma,\tau)} : \tau \to \delta_H \to H \to H
\]

with step types

\[
\delta_G := \sigma \to \text{PSD} \times (G + \sigma) + (H + \tau),
\]

\[
\delta_H := \tau \to \text{PSD} \times (G + \sigma) + (H + \tau).
\]

Here we have $\sigma := G$ and $\tau := H$. Omitting the upper indices of $^{co}R$, the terms $^{co}R_G N M M'$ and $^{co}R_H N' M M'$ are defined by the conversion rules

\[
^{co}R_G N M M' \mapsto \begin{cases} LR_u \left( [\text{id}_{G \to G}, \lambda_y (^{co}R_G y M M')] u_2 \right) & \text{if } MN = \text{inl}(u) \\ U \left( [\text{id}_{H \to H}, \lambda_z (^{co}R_H z M M')] v \right) & \text{if } MN = \text{inr}(v) \end{cases}
\]

\[
^{co}R_H N' M M' \mapsto \begin{cases} \text{Fin}_u \left( [\text{id}_{G \to G}, \lambda_y (^{co}R_G y M M')] u_2 \right) & \text{if } M' N' = \text{inl}(u) \\ D \left( [\text{id}_{H \to H}, \lambda_x (^{co}R_H z M M')] v \right) & \text{if } M' N' = \text{inr}(v) \end{cases}
\]

By analyzing the particular step functions $M, M'$ extracted from our proof we see that we can write $\lambda_y (^{co}R_G y M M')$ and $\lambda_z (^{co}R_H z M M')$ as functions $f : \sigma \to G$ and $f' : \tau \to H$ defined by

\[
f(LR_a(p)) = LR_{-a}(p), \quad f'(\text{Fin}_{a}(p)) = \text{Fin}_{-a}(p),
\]

\[
f(U(q)) = U(f'(q)), \quad f'(D(q)) = D(f'(q)).
\]

7. Average

We now consider the problem to compute the average of two real numbers given in pre-Gray code. As said above, our aim here is to extract such a program from a proof of the existence of the average. Since it is easier to give such a proof for reals given as signed digit streams $[3, 8, 9]$, we will do this in a somewhat indirect fashion, by first transforming the Gray coded reals into signed digit streams, dealing with the average there, and then transforming the result back into Gray code. Compared with $[8, 9]$ we use a completely abstract approach here, by eliminating all reference to concrete representations of rationals or reals. To this end we need the axiom

\[
\frac{x + d}{2} + \frac{y + e}{2} = \frac{x + y + (d + e)}{2},
\]

where the $+$ in $x + y$ denotes addition of abstract reals, and in $d + e$ addition of signed digits, yielding a result in the “extended signed digits”
\[\text{SD}_2 := \{-2, -1, 0, 1, 2\}.\] We also benefit from the omission of the nullary constructor \(I_1\) from our algebra \(I\) (cf. Remark 4.1); this removes non-essential case distinctions.

The informal proof in [3] of the existence of the average w.r.t. signed digit stream coded reals proceeds as follows. To prove
\[\forall_{x,y} (\text{co}\; I_x \to \text{co}\; I_y \to \text{co}\; \frac{x+y}{2})\]
consider the sets
\[X := \left\{ \frac{x+y}{2} \mid x, y \in \text{co}\; I \right\}, \quad Y := \left\{ \frac{x+y+i}{4} \mid x, y \in \text{co}\; I, i \in \text{SD}_2 \right\}.\]

It suffices to show that \(Y\) satisfies the clause coinductively defining \(\text{co}\; I\), for then by the greatest-fixed-point axiom for \(\text{co}\; I\) we have \(Y \subseteq \text{co}\; I\). Since clearly \(X \subseteq Y\) we then have \(X \subseteq \text{co}\; I\), which is our claim.

To understand the program to be extracted from this proof it helps to spell out the two lemmas just used. Let \(i, j\) range over \(\text{SD}_2\).

**Lemma 7.1 \((X\text{Sub}\; Y)\).**
\[
\forall_{x,y} \exists_{x',y'} (x + y + i) = \frac{x' + y' + 4}{4}.
\]
This follows easily from the clause for \(\text{co}\; I\).

**Extracted term.**
\[\text{[v,v0]left Des v plus left Des v0@right Des v0}@right Des v0\]

This term can be read as follows. Given \(v, v0\), destruct both. Both are composed, i.e., of the form \(dv1\) and \(dv2\). Take their components \(d1, v1\) and \(d2, v2\). Then the result is \(d1\; \text{sdplus}\; d2\; \text{pair}\; v1\; \text{pair}\; v2\).

For Lemma 7.2 we need functions \(J : \text{SD} \to \text{SD} \to \text{SD}_2 \to \text{SD}_2\) and \(K : \text{SD} \to \text{SD} \to \text{SD}_2 \to \text{SD}\) such that \(d + e + 2i = J(d, e, i) + 4K(d, e, i)\). They can be defined easily by cases on \(d, e\) and \(i\). Using these we can relate our abstract average functions \(\frac{x+d}{2}\) and \(\frac{x+y+i}{4}\) by means of the JK-axiom
\[\frac{x+d}{2} + \frac{y+e}{4} + i = \frac{x+y+J(d,e,i)}{4} + K(d,e,i).\]

**Lemma 7.2 \((Y\text{Sat}\; \text{CoIClause})\).**
\[
\forall_i \forall_{x,y} \exists_{x',y'} (x + y + i) = \frac{x' + y' + i}{4} + d.
\]

The proof is similar to the one for Lemma 7.1 above: it uses the clause for \(\text{co}\; I\) and the JK-axiom (8).
Extracted term.

\[ i, v, v_0 \]

\begin{align*}
J & \leftarrow \text{Des } v \leftarrow \text{Des } v_0 \leftarrow \text{i@} \\
K & \leftarrow \text{Des } v \leftarrow \text{Des } v_0 \leftarrow \text{i@right } \text{Des } \text{v}@ \text{right } \text{Des } v_0
\end{align*}

This term can be read as follows. Given \( i, v, v_0 \), destruct the latter two. Both are composed, i.e., of the form \( dv_1 \) and \( dv_2 \). Take the components \( d_1, v_1 \) and \( d_2, v_2 \), and return \( J \leftarrow d_1 \leftarrow d_2 \leftarrow \text{i pair } K \leftarrow d_1 \leftarrow d_2 \leftarrow \text{i pair } v_1 \leftarrow \text{pair } v_2 \).

**Lemma 7.3 (YSubCol).**

\[
\forall_{x}^{\mathrm{nc}}(z = \frac{x + y + i}{4} \rightarrow \text{coIz})
\]

**Proof.** By coinduction from Lemma 7.2. \( \square \)

**Theorem 7.4 (CoIAverage).**

\[
\forall_{x,y}^{\mathrm{nc}}(\text{coI}_x \rightarrow \text{coI}_y \rightarrow \text{coI}_{\frac{x+y}{2}}).
\]

**Proof.** Immediate form Lemmata 7.1 and 7.3. \( \square \)

Extracted term. After animating Lemma 7.3 we obtain

\[
[v, v_0](\text{CoRec } \text{sdtwo@iv@iv=>iv})(\text{cXSubY } v \leftarrow v_0)
\]

\[
((\text{ivw})[\text{let } jdwv]
\]

\[
(\text{cYSatCoIClause left } \text{ivw left right } \text{ivw right right } \text{ivw})
\]

\[
(\text{left right } jdwv@\text{InR}(\text{left } jdwv@\text{right right } jdwv)))
\]

Here \( \text{ivw} \) is a variable of type \( \text{SD}_2 \times \text{I} \times \text{I} \), and \( \text{jdvw} \) is a variable of type \( \text{SD}_2 \times \text{SD} \times \text{I} \times \text{I} \). To obtain the \text{let} expression in the term above, we have used implicitly the "identity lemma" \( \text{Id}: P \rightarrow P \); its realizer has the form \( \lambda_{f,x}(fx) \). If \( \text{Id} \) is not animated, the extracted term has the form \( \text{cId}(\lambda_{x,M}N) \), which is printed as \( [\text{let } x N M] \). This avoids multiple evaluations of the term \( N \), which in our case contains \( \text{cYSatCoIClause} \), i.e., the computational content of Lemma 7.2. We may also animate Lemmata 7.1 and 7.2, and obtain

\[
[v, v_0](\text{CoRec } \text{sdtwo@iv@iv=>iv})
\]

\[
(\text{left Des v plus left Des v}@ \text{right Des v}@ \text{right Des v}@ v_0)
\]

\[
((\text{ivw})[\text{let } jdwv]
\]

\[
(\text{J left Des left right } \text{ivw}
\]

\[
\text{left Des right right } \text{ivw}
\]

\[
\text{left ivw@}
\]

\[
K \leftarrow \text{left Des left right } \text{ivw}
\]

\[
\text{left Des right right } \text{ivw}
\]
To understand this term recall the type (5) of the corecursion operator. We analyze the step function $M$ extracted from the proof. When applied to an argument $ivw$ of type $\tau = SD_2 \times I \times I$, $M$ gives a result of type $SD \times (I + \tau)$, as follows. Destruct $ivw$ into the form $(i, (d,v), (e,w))$, and let $jdvw$ be the quadruple $(J(d,e,i), K(d,e,i), v, w)$. Return $(K(d,e,i), \text{inr}(J(d,e,i), v, w))$.

Hence we can write $\lambda y. R^\tau y M$ as a function $f : \tau \rightarrow I$ defined by

$$f(i, C_d(v), C_e(w)) = C_K(d,e,i)(f(J(d,e,i), v, w)).$$

8. Transforming pre-Gray code into Gray code

With pre-Gray code, there are many ways of expressing the same real number as we noted in Section 3. In particular, the terms $U(D^k(Fin_a p))$ and $LR^a(LR^{k-1} p))$ denote the same number. Here, $D^k$ and $LR^{k-1}$ denote $k$-times repetition of the same constructor. In this section, we extract a program which transfers the former pattern in the first $n$ elements of a pre-Gray code into the latter pattern.

Similar to $G$ we inductively define a set $\mathcal{G}$ of (abstract) reals, this time with a zero (or initial) clause:

$$\forall_{x} \forall_{n} \mathcal{G}(x, 0),$$

$$\forall_{x, n} \forall_{a} (\mathcal{G}(x, n) \rightarrow \mathcal{G}(-a \frac{x - 1}{2}, n + 1)),\quad$$

$$\forall_{x, n} (\mathcal{H}(x, n) \rightarrow \mathcal{G}(\frac{x}{2}, n + 1))$$

with a separate $\mathcal{H}$ inductively defined by the clauses

$$\forall_{x} \forall_{n} \mathcal{H}(x, 0),\quad \forall_{x, n} (\mathcal{H}(x, n) \rightarrow \mathcal{H}(\frac{x}{2}, n + 1)).$$

The corresponding operators $\mathcal{\Gamma}$, $\mathcal{\Delta}$ for $\mathcal{G}$, $\mathcal{H}$ are defined by

$$(y, m) \in \mathcal{\Gamma}(X) \leftrightarrow m = 0 \lor$$

$$\exists_{(x,n) \in X} \exists_{a} (y = -a \frac{x - 1}{2} \land m = n + 1) \lor$$

$$\exists_{(x,n) \in X} (y = \frac{x}{2} \land m = n + 1),$$

$$(y, m) \in \mathcal{\Delta}(X) \leftrightarrow m = 0 \lor \exists_{x \in X} (y = \frac{x}{2} \land m = n + 1).$$
Hence \( z^G = \nu X z^\Gamma(X) \) and \( z^H = \nu X z^\Delta(X) \).

The above predicates carry an additional natural number argument, to be used as a bound. From a proof of \( \{ (x, n) \mid co^G x \} \subseteq z^G \), we extract the desired program to compute a prefix of a pre-Gray code of \( x \) of length \( n \) which does not contain a subsequence of the form UD \( a \) from a pre-Gray code of \( x \). The associated algebra for \( z^H \) it is just the natural numbers \( \mathbb{N} \), and for \( z^G \) it is \( z^G \) with constructors

\[ \text{Nz: } z^G, \quad \text{LRz: } \text{PSD} \rightarrow z^G, \quad \text{Uz: } \mathbb{N} \rightarrow z^G. \]

Additionally, the following axiom is used in the proof of Proposition 8.2.

\[ \forall x, d \left( \frac{x + d}{2} + 0 = \frac{x - d}{2} + d \right). \]

**Lemma 8.1 (GenCoGLR).** \( \forall x, a \left( co^G x \rightarrow co^G (-a x - 2) \right) \).

**Proof.** Easy by coinduction. \( \square \)

**Proposition 8.2 (CoGToBGGen).**

\[ \forall n \forall x, a \left( co^G x \rightarrow z^G(x, n) \right), \]

\[ \forall n \forall x, a \left( co^H x \rightarrow z^H(x, n) \vee \exists y \in co^G \exists a (z^G(y, n - 1) \land x = a y + 1 - 2) \right). \]

**Proof.** We prove both statements simultaneously by induction on \( n \). The case \( n = 0 \) is trivial. For the step case, we first assume \( co^G x_1 \) and prove \( (x_1, n + 1) \in z^G \). We have

\[ \exists x_2 \in co^G \exists a (x_1 = -a x_2 - 1 - 2) \vee \exists x_2 \in co^H (x_1 = x_2 - 2/2). \]

(Case A) Suppose that the left hand side holds. Then, by induction hypothesis applied to \( x_2 \), we have \( z^G(x_2, n) \). Therefore \( (x_1, n + 1) \in z^G \) because

\[ z^G(x_2, n) \rightarrow z^G(x_1 + 2 - 2/2). \]

(Case B) Suppose that the right hand side holds. Then, \( x_1 = x_2 - 2/2 \) for \( x_2 \in co^H \). Therefore, by induction hypothesis,

\[ z^H(x_2, n) \vee \exists x_3 \in co^G \exists a (z^G(x_3, n - 1) \land x_2 = a x_3 + 1 - 2/2). \]

(Case B1) Suppose that the left hand side holds. Then, since \( z^H(x_2, n) \) and \( x_1 = x_2 - 2/2 \), \( z^G(x_1, n + 1) \) holds.

(Case B2) Suppose that the right hand side holds. Then,

\[ x_1 = \frac{x_2}{2} = a x_3 + 1 + 2 = -a x_3 - 1 - 2 = -a x_3 - 1 - 2 = -a x_4 + 1 - 2. \]
22 U. BERGER, K. MIYAMOTO, H. SCHWICHTENBERG AND H. TSUIKI

for some \( a_3, x_3 \in \mathcal{G} \) and \( x_4 \coloneqq -\frac{x_3 - 1}{2} \). Since \( x_1 = -a_3 \frac{x_4 - 1}{2} \), for our goal \( \mathcal{G}(x_1, n + 1) \) it suffices to prove \( \mathcal{G}(x_4, n) \). In case \( n = 0 \) this follows from the initial clause for \( \mathcal{G} \), and in case \( n = m + 1 \) it follows from \( \mathcal{G}(x_3, n - 1) \) by the first generating clause for \( \mathcal{G} \), since \( x_4 = -\frac{x_3 - 1}{2} \).

Next, we suppose that \( \mathcal{H} x_1 \) and prove

\[
\mathcal{H}(x_1, n + 1) \lor \exists y \in \mathcal{G} \exists a (\mathcal{G}(y, n) \land x_1 = a \frac{y + 1}{2}).
\]

The argument is almost the same as above. Since \( \mathcal{H} x_1 \), we have

\[
\exists x_2 \in \mathcal{G} \exists a (x_1 = a \frac{x_2 + 1}{2}) \lor \exists x_2 \in \mathcal{H} (x_1 = \frac{x_2}{2}).
\]

(Case A) Suppose that the left hand side holds. We have \( x_1 = a_2 \frac{x_2 + 1}{2} \) for \( a_2 \) and \( x_2 \in \mathcal{G} \). By induction hypothesis, \( \mathcal{G}(x_2, n) \). Therefore

\[
\exists y \in \mathcal{G} \exists a (\mathcal{G}(y, n) \land x_1 = a \frac{y + 1}{2}).
\]

(Case B) Suppose that the right hand side holds. We have \( x_1 = \frac{x_2}{2} \) for \( x_2 \in \mathcal{H} \). By induction hypothesis,

\[
\mathcal{H}(x_2, n) \lor \exists x_3 \in \mathcal{G} \exists a (\mathcal{G}(x_3, n - 1) \land x_2 = a \frac{x_3 + 1}{2}).
\]

(Case B1) Suppose that the left hand side holds. Then, since \( \mathcal{H}(x_2, n) \) and \( x_1 = \frac{x_2}{2} \), we have \( \mathcal{H}(x_1, n + 1) \).

(Case B2) Suppose that the right hand side holds. Then,

\[
x_1 = \frac{x_2}{2} = a \frac{x_3 + 1}{4} = a \frac{\frac{x_3 - 1}{2} + 1}{2} = a \frac{x_4 + 1}{2}
\]

for \( x_4 := \frac{x_3 - 1}{2} \). We prove the right hand side of our goal for \( x_4 \) and \( a \). Since \( x_1 = a \frac{x_4 + 1}{2} \) it suffices to prove \( x_4 \in \mathcal{G} \) and \( \mathcal{G}(x_4, n) \). From \( x_3 \in \mathcal{G} \) we obtain \( x_4 \in \mathcal{G} \) by Lemma 8.1. To prove \( \mathcal{G}(x_4, n) \) we argue by cases on \( n \). In case \( n = 0 \) this follows from the initial clause for \( \mathcal{G} \), and in case \( n = m + 1 \) it follows from \( \mathcal{G}(x_3, n - 1) \) by the first generating clause for \( \mathcal{G} \), since \( x_4 = \frac{x_3 - 1}{2} \).

\[\square\]

Extracted term.

\[
\text{[n]} \text{(Rec nat=>}(\text{ag=>bg})@@(ah=>nat ysum psd@@ag@@bg))n
\]
\[
((\text{[p]}\text{Nz})@@(\text{[q]}\text{InL Zero}))
\]
\[
(\text{In0,psf})
\]
\[
(\text{[p]}\text{case} (\text{Des p})
\]
\[
(\text{InL ap} \rightarrow \text{LRZ left ap(left psf right ap)})
\]
\[
(\text{InR q} \rightarrow)
\]
Here \( psf \) is a variable for a “pair of step functions”, i.e., of type

\[
(G \to ^*G) \times (H \to N + PSD \times G \times ^*G)
\]

and \( apbg \) of type \( PSD \times G \times ^*G \). The structure of the extracted term closely mirrors the structure of the proof, hence we do not comment it any further.

9. Experiments

We experiment with the extracted programs in the previous sections. Let \( stog \), \( gtos \), \( av \) and \( gtobg \) be the extracted programs from the proofs of Proposition 6.3, Proposition 6.2, Theorem 7.4 and the first part of Proposition 8.2, respectively.

Example 9.1 (Flip mode). The signed digit stream \( C \ Rht \ C \ Lft \ C \ Rht \ (C \ Lft \ldots) \), which stands for \( 1, \bar{1}, 1, \bar{1}, \ldots \), is transformed by \( stog \) into the pre-Gray code \( LR \ PRht \ LR \ PRht \ LR \ PRht \ LR \ PRht \ldots \). On the other hand, \( C \ Rht \ C \ Rht \ C \ Lft \ C \ Rht \ C \ Lft \ldots \) is transformed into \( LR \ PRht \ LR \ PLht \ LR \ PRht \ LR \ PRht \ldots \). In pre-Gray code the occurrence of \( PRht \) outside of a block switches the flip mode. The program \( gtos \) works as the inverse.
Example 9.2 (Block). The signed digit stream \( \text{C Mid} \text{C Mid} \text{C Rht} \text{C Lft} \ldots \), which stands for \(0,0,1,\bar{1},\ldots\), is transformed by \(\text{stog}\) into the pre-Gray code \(\text{U} \text{D} \text{Fin} \text{PRht} \text{LR PLft} \ldots\). The program \(\text{gtos}\) works as the inverse.

Example 9.3 (Flip mode in a block). The signed digit stream \(\text{C Mid} \text{C Lft} \text{C Rht} \text{C Rht} \ldots\), which stands for \(0,1,1,\ldots\), is transformed by \(\text{stog}\) into the pre-Gray code \(\text{U} \text{Fin} \text{PLft} \text{LR PLft} \ldots\). On the other hand, \(\text{C Mid} \text{C Rht} \text{C Rht} \text{C Rht} \ldots\) is transformed into \(\text{U} \text{Fin} \text{PRht} \text{LR PRht} \text{LR PLft} \ldots\). The occurrence of \(\text{PLft}\) in a block switches the flip mode. The pre-Gray code \(\text{LR PRht} \text{U} \text{D} \text{Fin} \text{PRht} \text{LR PRht} \text{LR PLft} \ldots\), obtained from \(1,0,0,\bar{1},\bar{1},\bar{1},\ldots\), shows that the flip mode started before the block also effects. For both of the above cases, \(\text{gtos}\) transforms the output to the input.

Example 9.4 (Average). Consider the average of \(\frac{5}{8}\) and \(\frac{3}{4}\). Feeding \(\text{LR PRht} \text{U} \text{Fin} \text{PLft} \text{U} \text{D} \ldots\) and \(\text{LR PRht} \text{LR PRht} \text{U} \text{D} \ldots\) to \(\text{av}\), the result \(\text{LR PRht} \text{LR PLft} \text{U} \text{Fin} \text{PRht} \text{U} \text{D} \ldots\) is computed.

Example 9.5 (Gray code). From 5 and the pre-Gray code \(\text{U} \text{Fin} \text{PLft} \text{U} \text{Fin} \text{PRht} \ldots\), \(\text{gtobg}\) computes the Gray-code \(\bar{1},1,\bar{1},1,1\) as \(\text{LRz PLft} \text{LRz PRht} \text{LRz PLft} \text{LRz PRht} \text{NRz})\). From 5 and \(\text{U} \text{Fin} \text{PLft} \text{U} \text{D} \ldots\), \(\text{gtobg}\) computes \(1,1,\bot,1,0,0\) as \(\text{LRz PLft} \text{LRz PRht} \text{Uz} 2\)) by interpreting \(\text{Uz}\) to be \(\bot\) and 2 to be \(1,0,0\).

10. Conclusion

We studied real number computation based on Gray code, being focused on its logical aspects. The notion of pre-Gray code was introduced in order to formulate and prove properties of Gray code within the theory TCF using inductive and coinductive definitions. The case studies were presented to extract from proofs the following algorithms: translators between signed digit stream and pre-Gray code, average and a bounded translator from pre-Gray code to Gray code. Furthermore, those case studies were implemented in Minlog as running examples.

As for the future work, uniformly continuous functions for (pre-)Gray code can be studied with the same motivation. Concerning Gray code computation, the program constructions as \(\text{gamb}\) and \(\text{pif}\) used by Tsuiki, Sugihara and Terayama can suggest further development of the program extraction framework.
Extracted programs are provably correct because of the soundness theorem. However, Minlog’s feature to automatically generate correctness proofs still needs to be extended to also cover simultaneous coinductive definitions.

References


