Gray Codes for Arbitrary Numeral Systems

Maximilian Spitz

April 18, 2024

Abstract

The original Gray code after Frank Gray, also known as reflected binary code (RBC), is an ordering of the binary numeral system such that two successive values differ only in one bit. We provide a theory for Gray codes of arbitrary numeral systems, which is a generalisation of the original idea to an arbitrary base as presented by Sankar et al. [1]. Contained is the necessary theoretical environment to express and reason about the respective properties.

Contents

1 An Encoding for Natural Numbers

theory *Encoding-Nat*
imports *Main*
begin

At first, an encoding of naturals as lists of digits with respect to an arbitrary base $b \geq 2$ is introduced because the presented Gray code and its properties are reasonably expressed in terms of a word representation of numbers.

1.1 Validity and Valuation

In the context of a given base, not all possible code words are valid number representations. A validity predicate is defined, that checks if a code word is valid and a valuation to obtain the number represented by a valid word.

type-synonym *base* = *nat*

type-synonym *word* = *nat list*

fun *val* :: *base* ⇒ *word* ⇒ *nat* **where**
val b [] = *0*
| *val b* (*a*#*w*) = *a* + *b*∗*val b w*

fun *valid* :: *base* ⇒ *word* ⇒ *bool* **where**
valid b [] ⟷ *2*≤*b*
| *valid b* (*a*#*w*) ⟷ *a*<*b* ∧ *valid b w*

Given a base, the value of a valid word is bound by its length.

lemma *val-bound*:
valid b w ⟹ *val b w* < *b*^*length*(*w*)
⟨*proof*⟩

lemma *valid-base*:
valid b w ⟹ *2*≤*b*
⟨*proof*⟩

1.2 Encoding Numbers as Words

It was stated that not all code words are valid. Similarly, numbers do not have a unique word representation in general. Therefore, it is reasonable to normalise representations with respect to either value or word length. A normal representation w.r.t. value is without leading zeroes. However, if the word length is fixed, numbers can be represented only up to an upper bound. Note that this bound is stated above.

fun *enc* :: *base* ⇒ *nat* ⇒ *word* **where**
enc - *0* = []
| *enc b n* = (*if 2*≤*b then n mod b*#*enc b* (*n div b*) *else undefined*)

fun *enc-len* :: *base* ⇒ *nat* ⇒ *nat* **where**
enc-len - *0* = *0*
| *enc-len b n* = (*if 2*≤*b then Suc*(*enc-len b* (*n div b*)) *else undefined*)

fun *lenc* :: *nat* ⇒ *base* ⇒ *nat* ⇒ *word* **where**
lenc 0 - - = []
| *lenc* (*Suc k*) *b n* = *n mod b*#*lenc k b* (*n div b*)

definition *normal* :: *base* ⇒ *word* ⇒ *bool* **where**
normal b w ≡ *enc-len b* (*val b w*) = *length w*

1.3 Correctness

Now, the expected properties of above definitions are proven as well as that they interact correctly.

lemma *length-enc*:
$2 \leq b \Longrightarrow length\ (enc\ b\ n) = enc\text{-}len\ b\ n$
$\langle proof \rangle$

lemma *length-lenc*:
$length\ (lenc\ k\ b\ n) = k$
$\langle proof \rangle$

lemma *val-correct*:
$valid\ b\ w \Longrightarrow lenc\ (length\ w)\ b\ (val\ b\ w) = w$
$\langle proof \rangle$

lemma *val-enc*:
$2 \leq b \Longrightarrow val\ b\ (enc\ b\ n) = n$
$\langle proof \rangle$

lemma *val-lenc*:
$val\ b\ (lenc\ k\ b\ n) = n\ mod\ b\hat{\ }k$
$\langle proof \rangle$

lemma *valid-enc*:
$2 \leq b \Longrightarrow valid\ b\ (enc\ b\ n)$
$\langle proof \rangle$

lemma *valid-lenc*:
$2 \leq b \Longrightarrow valid\ b\ (lenc\ k\ b\ n)$
$\langle proof \rangle$

lemma *encodings-agree*:
$2 \leq b \Longrightarrow lenc\ (enc\text{-}len\ b\ n)\ b\ n = enc\ b\ n$
$\langle proof \rangle$

lemma *inj-enc*:
$2 \leq b \Longrightarrow inj\ (enc\ b)$
$\langle proof \rangle$

lemma *inj-lenc*:
$inj\text{-}on\ (lenc\ k\ b)\ \{..<b\hat{\ }k\}$
$\langle proof \rangle$

lemma *normal-enc*:
$2 \leq b \Longrightarrow normal\ b\ (enc\ b\ n)$
$\langle proof \rangle$

lemma *normal-eq*:

$[\![valid\ b\ v;\ valid\ b\ w;\ normal\ b\ v;\ normal\ b\ w;\ val\ b\ v = val\ b\ w]\!] \Longrightarrow v = w$
⟨*proof*⟩

lemma *inj-val*:
$inj\text{-}on\ (val\ b)\ \{w.\ valid\ b\ w \wedge normal\ b\ w\}$
⟨*proof*⟩

lemma *enc-val*:
$[\![valid\ b\ w;\ normal\ b\ w]\!] \Longrightarrow enc\ b\ (val\ b\ w) = w$
⟨*proof*⟩

lemma *range-enc*:
$2 \leq b \Longrightarrow range\ (enc\ b) = \{w.\ valid\ b\ w \wedge normal\ b\ w\}$
⟨*proof*⟩

lemma *range-lenc*:
$2 \leq b \Longrightarrow lenc\ k\ b\ `\ \{..<b\ \hat{}\ k\} = \{w.\ valid\ b\ w \wedge length\ w = k\}$
⟨*proof*⟩

theorem *enc-correct*:
$2 \leq b \Longrightarrow bij\text{-}betw\ (enc\ b)\ UNIV\ \{w.\ valid\ b\ w \wedge normal\ b\ w\}$
⟨*proof*⟩

Given a valid base b and length k, we encode exactly the first b^k numbers.

theorem *lenc-correct*:
$2 \leq b \Longrightarrow bij\text{-}betw\ (lenc\ k\ b)\ \{..<b\hat{}k\}\ \{w.\ valid\ b\ w \wedge length\ w = k\}$
⟨*proof*⟩

1.4 Circular Increment Operation

It is beneficial for our purpose to have an increment operation on words of fixed length that wraps around. Mathematically, this corresponds to adding 1 in the additive group of the factor ring of the integers modulo (b^k). Correctness is proven in terms of previously verified operations.

fun *inc* :: *nat* ⇒ *word* ⇒ *word* **where**
$inc\ \text{-}\ [] = []$
$|\ inc\ b\ (a\#w) = Suc\ a\ mod\ b\#(if\ Suc\ a \neq b\ then\ w\ else\ inc\ b\ w)$

lemma *length-inc*:
$length\ (inc\ b\ w) = length\ w$
⟨*proof*⟩

lemma *valid-inc*:
$valid\ b\ w \Longrightarrow valid\ b\ (inc\ b\ w)$
⟨*proof*⟩

Note that the following fact shows that we do not only have an encoding in the sense that it is a bijection but we also preserve a certain structure, that is necessary for the purpose of reasoning about Gray codes.

theorem *val-inc*:
  $valid\ b\ w \Longrightarrow val\ b\ (inc\ b\ w) = Suc\ (val\ b\ w)\ mod\ b\hat{\ }length(w)$
$\langle proof \rangle$

lemma *inc-correct*:
  $inc\ b\ (lenc\ k\ b\ n) = lenc\ k\ b\ (Suc\ n)$
  $\langle proof \rangle$

lemma *inc-not-eq*: $valid\ b\ w \Longrightarrow (inc\ b\ w = w) = (w = [])$
  $\langle proof \rangle$

end

2 A Generalised Distance Measure

theory *Code-Word-Dist*
  imports *Encoding-Nat*
begin

In the case of the reflected binary code (RBC) it is sufficient to use the Hamming distance to express the property, because there are only two distinct digits so that one bitflip naturally always corresponds to a distance of 1.

2.1 Distance of Digits

We can interpret a bitflip as an increment modulo 2, which is why for the distance of digits it appears as a natural generalisation to choose the amount of required increments. Mathematically, the distance $d(x, y)$ should be $y-x \pmod b$. For example we have $d(0, 1) = d(1, 0) = 1$ in the binary numeral system.

definition *dist1* :: $base \Rightarrow nat \Rightarrow nat \Rightarrow nat$ **where**
  $dist1\ b\ x\ y \equiv if\ x{\le}y\ then\ y{-}x\ else\ b{+}y{-}x$

Note that the distance of digits is in general asymmetric, so that it is in paticular not a metric. However, this is not an issue and in fact the most appropriate generalisation, partly due to the next lemma:

lemma *dist1-eq*:
  $[\![x < b;\ y < b;\ dist1\ b\ x\ y = 0]\!] \Longrightarrow x = y$
  $\langle proof \rangle$

lemma *dist1-0*:
  $dist1\ b\ x\ x = 0$
  $\langle proof \rangle$

lemma *dist1-ge1*:

$[\![x < b;\ y < b;\ x \neq y]\!] \Longrightarrow \mathit{dist1}\ b\ x\ y \geq 1$
⟨*proof*⟩

lemma *dist1-elim-1*:
$[\![x < b;\ y < b]\!] \Longrightarrow (\mathit{dist1}\ b\ x\ y + x)\ \mathit{mod}\ b = y$
⟨*proof*⟩

lemma *dist1-elim-2*:
$[\![x < b;\ y < b]\!] \Longrightarrow \mathit{dist1}\ b\ x\ (x+y) = y$
⟨*proof*⟩

lemma *dist1-mod-Suc*:
$[\![x < b;\ y < b]\!] \Longrightarrow \mathit{dist1}\ b\ x\ (\mathit{Suc}\ y\ \mathit{mod}\ b) = \mathit{Suc}\ (\mathit{dist1}\ b\ x\ y)\ \mathit{mod}\ b$
⟨*proof*⟩

lemma *dist1-Suc*:
$[\![2 \leq b;\ x < b]\!] \Longrightarrow \mathit{dist1}\ b\ x\ (\mathit{Suc}\ x\ \mathit{mod}\ b) = 1$
⟨*proof*⟩

lemma *dist1-asym*:
$[\![x < b;\ y < b]\!] \Longrightarrow (\mathit{dist1}\ b\ x\ y + \mathit{dist1}\ b\ y\ x)\ \mathit{mod}\ b = 0$
⟨*proof*⟩

lemma *dist1-valid*:
$[\![x < b;\ y < b]\!] \Longrightarrow \mathit{dist1}\ b\ x\ y < b$
⟨*proof*⟩

lemma *dist1-distr*:
$[\![x < b;\ y < b;\ z < b]\!] \Longrightarrow \mathit{dist1}\ b\ (\mathit{dist1}\ b\ x\ y)\ (\mathit{dist1}\ b\ x\ z) = \mathit{dist1}\ b\ y\ z$
⟨*proof*⟩

lemma *dist1-distr2*:
$[\![x < b;\ y < b;\ z < b]\!] \Longrightarrow \mathit{dist1}\ b\ (\mathit{dist1}\ b\ x\ z)\ (\mathit{dist1}\ b\ y\ z) = \mathit{dist1}\ b\ y\ x$
⟨*proof*⟩

2.2 (Hamming-) Distance between Words

The total distance between two words of equal length is then defined as the sum of component-wise distances. Note that the Hamming distance is equivalent to this definition for $b = 2$ and is in general a lower bound.

fun *hamming* :: *word* ⇒ *word* ⇒ *nat* **where**
hamming [] [] = *0*
| *hamming* (*a*#*v*) (*b*#*w*) = (*if a*≠*b then 1 else 0*) + *hamming v w*

The Hamming distance is only defined in the case of equal word length. In the following definition of a distance we assume leading zeroes if the word length is not equal:

fun *dist* :: *base* ⇒ *word* ⇒ *word* ⇒ *nat* **where**
dist - [] [] = *0*
| *dist b* (*x*#*xs*) [] = *dist1 b x 0* + *dist b xs* []
| *dist b* [] (*y*#*ys*) = *dist1 b 0 y* + *dist b* [] *ys*
| *dist b* (*x*#*xs*) (*y*#*ys*) = *dist1 b x y* + *dist b xs ys*

lemma *dist-0*:
dist b w w = *0*
⟨*proof*⟩

lemma *dist-eq*:
⟦*valid b v*; *valid b w*; *length v*=*length w*; *dist b v w* = *0*⟧ ⟹ *v* = *w*
⟨*proof*⟩

lemma *dist-posd*:
⟦*valid b v*; *valid b w*; *length v*=*length w*⟧ ⟹ (*dist b v w* = *0*) = (*v* = *w*)
⟨*proof*⟩

lemma *hamming-posd*:
length v=*length w* ⟹ (*hamming v w* = *0*) = (*v* = *w*)
⟨*proof*⟩

lemma *hamming-symm*:
length v=*length w* ⟹ *hamming v w* = *hamming w v*
⟨*proof*⟩

theorem *hamming-dist*:
⟦*valid b v*; *valid b w*; *length v*=*length w*⟧ ⟹ *hamming v w* ≤ *dist b v w*
⟨*proof*⟩

end

3 A non-Boolean Gray code

theory *Non-Boolean-Gray*
imports *Code-Word-Dist*
begin

The function presented below transforms a code word into a gray code and the corresponding decode function is exactly its inverse. The key idea is to shift down a digit by the prefix sum of gray digits. A crucial property is the behavior of this prefix sum under increment as stated below.

fun *to-gray* :: *base* ⇒ *word* ⇒ *word* **where**
to-gray - [] = []
| *to-gray b* (*a*#*v*) = (*let g*=*to-gray b v in dist1 b* (*sum-list g mod b*) *a*#*g*)

fun *decode* :: *base* ⇒ *word* ⇒ *word* **where**
decode - [] = []

| *decode b (g#c) = (g+sum-list c mod b) mod b#decode b c*

3.1 The Correctness Proof

The proof of all properties that are necessary for a gray code is presented below. Also, some auxiliary lemmas are required:

lemma *length-gray*:
length (to-gray b w) = length w
⟨*proof*⟩

lemma *valid-gray*:
valid b w $\Longrightarrow$ *valid b (to-gray b w)*
⟨*proof*⟩

The sum of grays is congruent to the value (mod b):

lemma *prefix-sum*:
valid b w $\Longrightarrow$ *sum-list (to-gray b w) mod b = val b w mod b*
⟨*proof*⟩

lemma *decode-correct*:
valid b w $\Longrightarrow$ *decode b (to-gray b w) = w*
⟨*proof*⟩

The following theorem states that the transformation to gray is an encoding of the valid code words:

theorem *gray-encoding*:
inj-on (to-gray b) {w. valid b w}
⟨*proof*⟩

lemma *mod-mod-aux*: $1 \leq k \Longrightarrow$ *(a::nat) mod b^k mod b = a mod b*
⟨*proof*⟩

lemma *gray-dist*:
valid b w $\Longrightarrow$ *dist b (to-gray b w) (to-gray b (inc b w))* ≤ 1
⟨*proof*⟩

lemmas *gray-simps = decode-correct dist-posd inc-not-eq length-gray length-inc valid-gray valid-inc*

lemma *gray-empty*:
valid b w $\Longrightarrow$ *(dist b (to-gray b w) (to-gray b (inc b w)) = 0) = (w = [])*
⟨*proof*⟩

The central theorem states, that it requires exactly one increment operation of one place within the word to go from the gray encoding of a number to the gray encoding of its successor. Note also, that we obtain a cyclic gray code in all cases, because the increment operation wraps the last number around to zero. Only the pathological case of an empty word has to be excluded.

theorem *gray-correct*:
$\llbracket$*valid b w*; $w \neq [] \rrbracket \Longrightarrow$ *dist b* (*to-gray b w*) (*to-gray b* (*inc b w*)) = *1*
⟨*proof*⟩

lemmas *hamming-simps* = *gray-dist hamming-dist le-trans length-gray length-inc valid-gray valid-inc*

theorem *gray-hamming*: *valid b w* $\Longrightarrow$ *hamming* (*to-gray b w*) (*to-gray b* (*inc b w*)) $\leq$ *1*
⟨*proof*⟩

end

References

[1] K. Sankar, V. Pandharipande, and P. Moharir. Generalized gray codes. In *Proceedings of 2004 International Symposium on Intelligent Signal Processing and Communication Systems. ISPACS 2004.*, pages 654–659, 2004. https://doi.org/10.1109/ISPACS.2004.1439140.