Chapter 14

Case study: The Gray code

In this chapter we use our new techniques for recursion in a slightly out-of-the-way case study, and in passing examine the issues of input and output.

14.1 The Gray code

The *Gray code* represents natural numbers as sequences of digits 0 or 1, but not in the usual way. If we let the function $\mathsf{gc} : \mathbb{N} \rightarrow \mathbf{seq}\{0, 1\}$ take a natural number to its Gray code, then we find it has these properties:

1. As for the usual binary code,

   $$\begin{aligned} \mathsf{gc}\,0 &= \langle 0 \rangle \\ \mathsf{gc}\,1 &= \langle 1 \rangle, \end{aligned}$$

   and for $n \geq 2$,

   $$\mathsf{gc}\,n \;=\; \mathsf{gc}(n \div 2) +\!\!+ \langle d \rangle, \text{ for } d \text{ either 0 or 1.}$$

   Alternatively, we could say $\mathsf{fr}\,\mathsf{gc}\,n = \mathsf{gc}(n \div 2)$.
2. But $\mathsf{gc}\,n$ and $\mathsf{gc}(n+1)$ differ in exactly one position, for all n.

Property 1 applies to both Gray and binary codes, with the difference being only the choice of d. Property 2, on the other hand, applies to the Gray code alone. Figure 14.1 lists the first few Gray codes, comparing them with the corresponding binary codes.

The Gray code is used in applications where it is important that only one bit changes at a time in counting up or down. We shall develop a program to output Gray codes; but first, we examine input and output in general.

n	$\mathsf{gc}\,n$	binary
0	0	0
1	1	1
2	1 1	1 0
3	1 0	1 1
4	1 1 0	1 0 0
5	1 1 1	1 0 1
6	1 0 1	1 1 0
7	1 0 0	1 1 1
8	1 1 0 0	1 0 0 0
9	1 1 0 1	1 0 0 1

Figure 14.1 The first Gray codes

14.2 Input and output

For programs using input and output we introduce two special-purpose variables named, by convention, α and ω. Their types will depend on the problem at hand, but usually α is a sequence of input values, and ω is a sequence of output values.

Input is performed by removing the first element of α, and that may be abbreviated as follows for any variable x:

$$\textbf{input}\ x \ \mathrel{\widehat{=}}\ \{\alpha \neq \langle\rangle\}\ x, \alpha := \mathsf{hd}\,\alpha, \mathsf{tl}\,\alpha\ .$$

Testing for end of file on input is just testing for emptiness of α:

$$\textbf{eof} \ \mathrel{\widehat{=}}\ \alpha = \langle\rangle\ .$$

Output is performed by appending an expression E to ω:

$$\textbf{output}\ E \ \mathrel{\widehat{=}}\ \omega := \omega +\!\!+ \langle E\rangle\ .$$

And output may be initialised by setting it to empty:

$$\textbf{rewrite} \ \mathrel{\widehat{=}}\ \omega := \langle\rangle\ .$$

Finally note that, since **input** x changes α (by removing an element), programs performing input should include α in the frame.

Our abstract program for this case study — a procedure — outputs the Gray code corresponding to a given integer n:

$$\textbf{procedure}\ \mathit{Gray}\ (\textbf{value}\ n : \mathbb{N}) \ \mathrel{\widehat{=}}\ \omega := \omega +\!\!+ \mathsf{gc}\,n\ .$$

We are assuming in this case that ω is a sequence of binary digits (thus ω : $\textbf{seq}\{0, 1\}$).

14.3 Isolate the base cases

Since the first two Gray codes are given explicitly, we treat them on their own, using an alternation. Our variant (for the recursion) is n itself:

$$\begin{array}{l} \sqsubseteq \ \textbf{variant}\ N \ \textsf{is}\ n\bullet \\ \quad \textbf{if}\ n < 2 \rightarrow \textbf{output}\ n \\ \quad [\!]\ \ n \geq 2 \rightarrow \{2 \leq n = N\}\ \omega := \omega +\!\!+ \ \textsf{gc}\, n \qquad \triangleleft \\ \quad \textbf{fi}\ . \end{array}$$

To make further progress, however, we must return to Properties 1 and 2. From Property 2, the number of 1's in $\textsf{gc}\,n$ must alternate between even and odd. Let the *parity* of a sequence as, written $\textsf{pr}\,as$, be 0 if the number of 1's in as is even, 1 if it is odd. Thus $\textsf{pr}\,\textsf{gc}\,n$ alternates as n increases. Since from Property 1 we know $\textsf{gc}\,0 = 0$, we have by a simple induction that

$$\textsf{pr}\,\textsf{gc}\,n \quad = \quad n \,\textsf{mod}\, 2\ . \tag{14.1}$$

The above suggests that to calculate $\textsf{gc}\,n$, we could first calculate $\textsf{gc}(n \div 2)$ and then use its parity to calculate the final bit d. That leads to

$$\begin{array}{lll} & \sqsubseteq\ \omega{:}\,[2 \leq n = N\ ,\ \omega = \omega_0 +\!\!+ \ \textsf{gc}\,n] & \\ & \sqsubseteq\ \text{“\textit{sequential composition} 8.4”};\ \textbf{con}\ \Omega : \textbf{seq}\{0,1\}\bullet & \\ & \quad \omega{:}\,[2 \leq n = N\ ,\ \omega = \omega_0 +\!\!+ \ \textsf{gc}(n \div 2)]\ ; & \text{(i)} \\ & \quad \omega{:}\,[\omega = \Omega +\!\!+ \ \textsf{gc}(n \div 2)\ ,\ \omega = \Omega +\!\!+ \ \textsf{gc}\,n] & \text{(ii)} \\ \text{(i)} & \sqsubseteq\ \{0 \leq n \div 2 < N\}\ \omega := \omega +\!\!+ \ \textsf{gc}(n \div 2) & \\ & \sqsubseteq\ \textit{Gray}\ (n \div 2)\ . & \end{array}$$

To refine (ii) we need a small calculation. For $a, b : \mathbb{N}$ let $a \oplus b$ be the sum $(a + b)$ **mod** 2; then we have

$$\begin{array}{ll} & \textsf{gc}(n \div 2) +\!\!+ \ \langle d \rangle = \textsf{gc}\,n \\ \Rrightarrow & \textsf{pr}(\textsf{gc}(n \div 2) +\!\!+ \ \langle d \rangle) = \textsf{pr}\,\textsf{gc}\,n \\ \equiv & \text{“from (14.1)”} \\ & (n \div 2)\ \textbf{mod}\ 2 \oplus d = n\ \textbf{mod}\ 2 \\ \Rrightarrow & d = n \div 2 \oplus n\ . \end{array}$$

Hence, with that equality, we have finally

$$\text{(ii)}\ \sqsubseteq\ \textbf{output}\ (n \div 2) \oplus n\ .$$

That concludes the development, which is summarised in Figure 14.2.

$$
\begin{array}{l}
\textbf{procedure } \mathit{Gray}\ (\textbf{value } n : \mathbb{N}) \\
\mathrel{\widehat{=}} \textbf{if } n < 2 \rightarrow \textbf{output } n \\
\quad [\!]\ n \geq 2 \ \rightarrow \\
\qquad\quad \mathit{Gray}\ (n \div 2); \\
\qquad\quad \textbf{output } (n \div 2) \oplus n \\
\quad \textbf{fi}\ .
\end{array}
$$

Figure 14.2 Recursive Gray-code code

14.4 Exercises

Ex. 14.1 ♡ Continue Figure 14.1 up to $n = 15$.

Ex. 14.2 ♡ The base case in recursions is often 0, but for the Gray code it is 0 and 1 together. Why?

Ex. 14.3 Develop recursive code for the following, where $\mathsf{br}\, n$ is the (ordinary) binary representation of n:

$$
\begin{array}{l}
\textbf{var } n : \mathbb{N};\ \ \omega : \textbf{seq}\{0,1\}\bullet \\
\omega :=\omega +\!\!+\ \mathsf{br}\, n\ .
\end{array}
$$

Of course, br itself is not code!

Ex. 14.4 Let $\mathsf{gcb} : \textbf{seq}\{0,1\} \rightarrow \textbf{seq}\{0,1\}$ take the binary representation of n to the sequence $\mathsf{gc}\, n$. For example,

$$\mathsf{gcb}\langle 1,0\rangle \quad = \quad \langle 1,1\rangle .$$

Develop *iterative* code for

$$
\begin{array}{l}
\textbf{var } n, g : \textbf{seq}\{0,1\}\bullet \\
\{n \neq \langle\rangle\}\ g, n :=\mathsf{gcb}\, n, ?\ .
\end{array}
$$

Ex. 14.5 Refine the following to iterative code:

$$
\begin{array}{l}
\textbf{var } n : \mathbb{N};\ \ \alpha : \textbf{seq}\{0,1\}\bullet \\
\alpha, n\!: [\neg\textbf{eof}\ ,\ \mathsf{gc}\, n = \alpha_0]\ \ .
\end{array}
$$

Ex. 14.6 ♡ For sequence q, define its reverse $\mathsf{rv}\, q$ as follows:

$$
\begin{array}{rcl}
\mathsf{rv}\langle\rangle & \mathrel{\widehat{=}} & \langle\rangle \\
\mathsf{rv}(\langle e\rangle +\!\!+\ q) & \mathrel{\widehat{=}} & \mathsf{rv}\, q +\!\!+ \langle e\rangle\ .
\end{array}
$$

Refine this program to recursive code, using a recursion block:

$$
\begin{array}{l}
\textbf{var } \alpha, \omega : \textbf{seq}\, E\bullet \\
\alpha, \omega :=?, \mathsf{rv}\, \alpha\ .
\end{array}
$$

Hint: Rewrite output first.