The Thue-Morse sequence: a nice exercise

0 notational and other preliminaries

Infinite lists are constructed by means of the list constructor $\triangleright$ ("cons"), which has the following properties, for all b, x and for all natural i:

$$\begin{array}{lcl}(b \triangleright x)\cdot 0 & = & b \\ (b \triangleright x)\cdot(i{+}1) & = & x\cdot i\end{array}$$

In addition we will be using $\lfloor 1$ ("drop one", or "tail") to obtain the tail of a list, that is:

$$(b \triangleright x)\lfloor 1 \;=\; x \quad ,$$

and, hence, we also have, for all natural i:

$$(x\lfloor 1)\cdot i \;=\; x\cdot(i{+}1) \quad .$$

The list obtained from a given list x by applying a function f to all of x's elements is denoted by $f\bullet x$; operator $\bullet$ ("map") has these properties:

$$\begin{array}{lcl}f\bullet(b \triangleright x) & = & f\cdot b \;\triangleright\; f\bullet x \\ (f\bullet x)\cdot i & = & f\cdot(x\cdot i)\end{array}$$

The latter property states that, function-wise, operator $\bullet$ just implements function composition (while preserving list structure).

* * *

We will use the notion of *productivity*, but here I will recapitulate neither its formal definition nor the theory about it. Informally, a function F, mapping (so-called) *listoids*[0] to listoids, is productive if its value is defined better than its argument by at least one element. The most important property involved is that a productive function has a unique infinite list as its fixed point. As a simple example, function F, defined by $F\cdot x = b \triangleright f\bullet x$, for some given value b and function f, is productive. Its fixed point is the infinite list whose element at position i is $f^i\cdot b$. Generally, productivity always involves, one way or the other, operator $\triangleright$.

[0] in other contexts also called *partial lists*.

* * *

We study functional programs for infinite lists. In this note variables x, y, z will have type $\mathcal{L}_\infty(\mathsf{Nat})$ –infinite list of naturals–. In what follows we will be needing a (well-known) function *zip* that combines two infinite lists into a single infinite list, according to this (element-wise) specification –for all x, y and for natural i–:

(0) $\quad zip\cdot x\cdot y\cdot(2*i) \;=\; x\cdot i$, and: $zip\cdot x\cdot y\cdot(2*i+1) \;=\; y\cdot i$.

This specification tells us that infinite list $zip\cdot x\cdot y$ is obtained from x and y by interleaving the elements of x and y strictly alternatingly: the elements of x occur in $zip\cdot x\cdot y$ at the even positions whereas y's elements occur at the odd positions.

As function *zip* destroys no information, it has an inverse, consisting of a pair of functions *unze* ("unzip even") and *unzo* ("unzip odd"), say, mapping infinite lists to infinite lists and with these specifications –for all z and for natural i–:

(1) $\quad unze\cdot z\cdot i \;=\; z\cdot(2*i)$,

(2) $\quad unzo\cdot z\cdot i \;=\; z\cdot(2*i+1)$.

From these specifications we can now prove that *unze* and *unzo* together indeed constitute *zip*'s inverse, as they satisfy, for all x, y:

(3) $\quad unze\cdot(zip\cdot x\cdot y) \;=\; x$, and: $unzo\cdot(zip\cdot x\cdot y) \;=\; y$.

By means of straightforward calculations the following recursive definitions can be derived, to satisfy the above specifications:

(4) $\quad zip\cdot(b\triangleright x)\cdot y \;=\; b \triangleright zip\cdot y\cdot x$;

(5) $\quad unze\cdot(b\triangleright z) \;=\; b \triangleright unzo\cdot z$;

(6) $\quad unzo\cdot(c\triangleright z) \;=\; unze\cdot z$.

aside: That properties like (3) can be proved using the specifications, that is, (0) through (2), of the functions only, instead of their definitions, enhances *modularity*: thus, the validity of such properties does not depend on the actual definitions for the functions; they are valid for all definitions satisfying the specifications. In addition, proofs involving (usually simpler) specifications may be simpler than proofs involving (usually more complicated) definitions. This illustrates, once again,

the relevance of (the use of) proper specifications, also in functional programming.

For example, we could also have defined our functions in the following, alternative way; doing so does not bring about the need to reprove property (3):

(7) $zip\cdot(b \triangleright x)\cdot(c \triangleright y) \;=\; b \triangleright c \triangleright zip\cdot x\cdot y$;

(8) $unze\cdot(b \triangleright c \triangleright z) \;=\; b \triangleright unze\cdot z$;

(9) $unzo\cdot(b \triangleright c \triangleright z) \;=\; c \triangleright unzo\cdot z$.

If one takes the position that a specification preferably captures the properties one intends to need during the use of the specified object, one may rightly argue that (3) is to preferred, as a specification of *unze* and *unzo*, over (1) and (2): formula (3) expresses directly that *unze* and *unzo* constitute *zip*'s inverse. An advantage of (1) and (2), though, is that they are more explicit, which makes the derivation of definitions (somewhat) easier.

□

1 the Thue-Morse sequence

The other day I encountered the, so-called, "Thue-Morse sequence", which was completely new to me. It is defined as the infinite sequence of the *parities* of all natural numbers, where the parity of a natural number is defined as the number of ones in that number's binary representation, reduced modulo 2. Notice that, although the binary representation of a natural number is not unique – due to the possibility of redundant leading zeroes –, a number's parity is unique nevertheless, because it depends on the ones in the binary representation only.

So, with some case analysis, a number's parity is 0 if the number of ones in its binary representation is even, and it is 1 if the number of ones is odd.

To formalize this we introduce a function pr, of type $\mathsf{Nat} \to \{0, 1\}$, mapping the natural numbers to their parities. A recursive definition for pr can be formulated without much ado, directly from the recursive definition for a natural's binary representation – for all natural n –:

(10) $pr\cdot 0 \;=\; 0$

(11) $pr\cdot(2{*}n) \;=\; pr\cdot n$

(12) $pr\cdot(2{*}n{+}1) \;=\; 1 - pr\cdot n$

notes: Notice that $1-pr\cdot n$ is just a concise encoding of $(1+pr\cdot n)\,\mathsf{mod}\,2$. Also notice that (10) and (11) overlap one another, for the case $n=0$. This is harmless because, viewed as a proposition, (11) is also true for $n=0$. This directly reflects that the binary representation of natural numbers is not unique but also that the definition of pr is insensitive to this.

□

The Thue-Morse sequence now is the infinite list ms, say, representing function pr. Apart from the type requirement that ms be an infinite list, its specification simply states that, viewed as functions, ms and pr are the same; that is, for all natural i they must satisfy:

(13) $\quad ms\cdot i \;=\; pr\cdot i \;\;.$

All by itself this specification already enables us to derive interesting properties of ms, such as:

$$
\begin{array}{ll}
 & unze\cdot ms\cdot i \\
= & \quad \{\ \text{specification } (1)\text{, of } unze\ \} \\
 & ms\cdot(2{*}i) \\
= & \quad \{\ \text{specification } (13)\text{, of } ms\ \} \\
 & pr\cdot(2{*}i) \\
= & \quad \{\ \text{definition } (11)\text{, of } pr\ \} \\
 & pr\cdot i \\
= & \quad \{\ \text{specification } (13)\text{, of } ms\ \} \\
 & ms\cdot i \;\;.
\end{array}
$$

So, we obtain $unze\cdot ms\cdot i \;=\; ms\cdot i$, for all natural i, and hence, as both $unze\cdot ms$ and ms are infinite lists, we conclude:

(14) $\quad unze\cdot ms \;=\; ms \;\;,$

independently of how ms is defined – one may also say: independently of how it is *implemented* –.

* * *

A recursive definition for ms can be derived, in a completely elementary way, by following the recursion pattern in the definition of pr and by means of the standard techniques for *function listification*:

$$
\begin{array}{ll}
 & ms \cdot 0 \\
= & \quad \{ \text{ specification } (13) \text{, of } ms \ \} \\
 & pr \cdot 0 \\
= & \quad \{ \text{ definition } (10) \text{, of } pr \ \} \\
 & 0 \\
= & \quad \{ \text{ the } \triangleright \text{-trick, where “?” denotes a “don't care” } \} \\
 & (0 \triangleright ?) \cdot 0 \quad ,
\end{array}
$$

which shows that, as far as the element with index 0 is concerned, ms can be defined by an expression of the shape $0 \triangleright ?$. Furthermore, we derive:

$$
\begin{array}{ll}
 & ms \cdot (2{*}i{+}1) \\
= & \quad \{ \text{ specification } (13) \text{, of } ms \ \} \\
 & pr \cdot (2{*}i{+}1) \\
= & \quad \{ \text{ definition } (12) \text{, of } pr \ \} \\
 & 1 - pr \cdot i \\
= & \quad \{ \text{ specification } (13) \text{, of } ms \text{, by Induction Hypothesis } \} \\
 & 1 - ms \cdot i \\
= & \quad \{ \text{ sectioning, preparing for factoring out } (\cdot i) \ \} \\
 & (1-) \cdot (ms \cdot i) \\
= & \quad \{ \ \bullet \ (\text{“map”}) - ms \ \text{ is a list} - \} \\
 & ((1-) \bullet ms) \cdot i \\
= & \quad \{ \text{ specification } (0) \text{, of } zip \text{ to obtain } (2{*}i) \text{ back } \} \\
 & zip \cdot ((1-) \bullet ms) \cdot ? \cdot (2{*}i) \\
= & \quad \{ \text{ the } \triangleright \text{-trick } \} \\
 & (? \triangleright zip \cdot ((1-) \bullet ms) \cdot ?) \cdot (2{*}i{+}1) \quad ,
\end{array}
$$

and, finally, we derive for the *positive* even indices:

$$
\begin{array}{ll}
 & ms \cdot (2{*}i{+}2) \\
= & \quad \{ \text{ specification } (13) \text{, of } ms \ \} \\
 & pr \cdot (2{*}i{+}2) \\
= & \quad \{ \text{ definition } (11) \text{, of } pr \ \} \\
 & pr \cdot (i{+}1) \\
= & \quad \{ \text{ specification } (13) \text{, of } ms \text{, by Induction Hypothesis } \}
\end{array}
$$

$$
\begin{array}{ll}
 & ms\cdot(i{+}1) \\
= & \quad \{ \lfloor \text{ (“drop”) , motivation follows } \} \\
 & (ms\lfloor 1)\cdot i \\
= & \quad \{ \text{ specification } (0) \text{ , of } zip \ \} \\
 & (zip\cdot ?\cdot(ms\lfloor 1))\cdot(2{*}i{+}1) \\
= & \quad \{ \text{ the } \triangleright\text{-trick } \} \\
 & (\,?\triangleright zip\cdot ?\cdot(ms\lfloor 1))\cdot(2{*}i{+}2) \quad .
\end{array}
$$

Thus, by combining the three results, we obtain as recursive definition for ms :

$$(15) \qquad ms \;=\; 0 \triangleright zip\cdot((1{-})\bullet ms)\cdot(ms\lfloor 1) \quad .$$

The transition, in the last derivation, from $ms\cdot(i{+}1)$ to $(ms\lfloor 1)\cdot i$ is necessary so as to make the expression fit the pattern that already has emerged from the first two derivations. These first two derivations indicate a definition for ms of the shape:

$$ms \;=\; 0 \triangleright zip\cdot((1{-})\bullet ms)\cdot ? \quad ,$$

so the only freedom we are left with is the freedom to substitute a suitable expression for $?$. This sets the stage for the third derivation, and guides it.

Bij introducing an additional variable to represent $ms\lfloor 1$ we can also encode ms 's definition (15) as follows:

$$ms \;=\; 0 \triangleright x \;\; \mathsf{whr} \;\; x \;=\; zip\cdot((1{-})\bullet ms)\cdot x \;\; \mathsf{end} \quad .$$

* * *

The above solution is not unique: by means of definition (4) of function zip , definition (15) of ms can be rewritten thus:

$$ms \;=\; zip\cdot(0 \triangleright ms\lfloor 1)\cdot((1{-})\bullet ms) \quad ,$$

and, indeed, this definition can also be obtained by means of a direct derivation from ms 's specification.

2 productivity issues

From the introduction we recall the definition of function zip :

$$(4) \qquad zip\cdot(b\triangleright x)\cdot y \;=\; b \triangleright zip\cdot y\cdot x \quad .$$

As a function of its first parameter, *zip* is $(+0)$ -productive, and as a function of its second parameter, *zip* even is $(+1)$ -productive: the function's value more critically depends on x than on y.

If we now substitute $(1-)\bullet x$ for x and $x\lfloor 1$ for y we obtain a function F, say, defined thus:

$$F{\cdot}x \;=\; zip{\cdot}((1-)\bullet x){\cdot}(x\lfloor 1) \quad .$$

The addition of $(1-)\bullet$ does not affect the function's productivity, and the substitution of $x\lfloor 1$ for x reduces productivity (in the second argument) by one. As a result, function F is $(+0)$ -productive.

Now, in terms of this function F, our sequence ms is a fixed point of function G defined by:

$$G{\cdot}x \;=\; 0 \triangleright F{\cdot}x \quad .$$

Function G is $(+1)$ -productive; hence, its fixed point ms is an infinite list indeed.

3 epilogue

Although this is quite a nice exercise, the Thue-Morse sequence poses no particular difficulties whatsoever, nor does it require any special techniques or formalisms. The derivation of recursive definition (15) from its specification (13) requires no more than application of the, by now standard, techniques – like the "$\triangleright$-trick" – for listification of a function. Of course, treating infinite lists as functions on the naturals, with some additional structure, simplifies matters (somewhat).

In addition, properties like (14) can be proved directly from the sequence's *specification*, hence, its recursive definition plays no part in this proof.

Eindhoven, 31 may 2005

Rob R. Hoogerwoord
department of mathematics and computing science
Eindhoven University of Technology
postbus 513
5600 MB Eindhoven