The tale of an eventually periodic sequence

0. Introduction

This note is devoted to a programming problem Wim Feijen shared with us while we were in Eindhoven. Wim promised us that the problem admits a very beautiful solution. Here we present a solution which we feel lives up to that qualification.

1. Some terminology

Before presenting the problem we introduce some terminology.

To say that a sequence is **iteratively defined** means that each of its elements (after the first) is some fixed function of the preceding element. In symbols, to say that a sequence g is iteratively defined means that there exists some function f such that:

$$g.(n+1) \;=\; f.(g.n) \qquad \text{for all } n \qquad .$$

To say that a sequence is **periodic** means that the sequence is composed of a finite pattern (the **period**) which repeats indefinitely. In symbols, to say that g is periodic means that there exists a positive number p such that:

$$g.(n+p) \;=\; g.n \qquad \text{for all } n \qquad .$$

The smallest such p is called the **period length**.

To say that a sequence is **eventually periodic** means that the sequence is periodic from some point onwards. In symbols, to say that g is eventually periodic means that there exists a number N and positive number p such that:

$$g.(n+p) \;=\; g.n \qquad \text{for all } n \;,\quad N \le n \qquad .$$

For the smallest such N, the interval $[0..N)$ is called the **non-periodic segment** of the sequence, and $[N..\infty)$ is called the **periodic segment**.

Please note that the particular formalizations we have chosen above do not represent design decisions; they are not necessarily the properties we will use in our solution to the problem. We have included them only to sharpen and clarify the English descriptions.

2. The problem

Given an iteratively defined, eventually periodic sequence, we are asked to compute the number of elements in its initial non-periodic segment. In what follows, we call the given sequence g.

3. Observations about the iterative definition of g

We begin with two observations about the iterative definition of g .

First: While designing the program, for simplicity's sake we prefer to reason about g in the usual way, with indices. But since an iteratively defined sequence is generated by a function, ultimately we wish to implement g in terms of its generating function. We reconcile these conflicting concerns by observing that this refinement can be easily implemented, provided our program makes small increments to g's indices. (For example, letting f be g's generating function and assuming $x = g.n$, increment $n := n+1$ would correspond to $x := f.x$.) We therefore adopt the constraint of small increments as our first design decision.

Second: Though we are given that g is iteratively defined and eventually periodic, the postcondition refers to periodicity alone. Thus we might wonder whether we can solve the problem without using the iterative definition of g . However, we cannot: Given an arbitrary eventually periodic sequence, we would have to inspect infinitely many of its elements in order to determine whether a particular element is in the periodic segment or not. Therefore the iterative definition of g is a necessary precondition.

Moreover, the above consideration suggests that the iterative definition of g will allow us to draw conclusions about periodicity by inspecting finitely many elements of g . Indeed, this is the only use we make of the iterative definition of g , and in order to prepare the reader for this use, we present in advance the only two proof steps which rely on this property. They are:

" n is in the periodic segment of g "

$\Leftarrow$ { g is iteratively defined }

" $g.n$ occurs twice in g "

and:

" m is a multiple of the period length "

$\Leftarrow$ { g is iteratively defined }

" some element of g repeats at a distance of m " .

We hope that with some reflection, the reader will find these proof steps unobjectionable.

* *
*

We now proceed with the design of the program in question.

4. The postcondition, and a coarse-grained solution

Recall that we wish to compute the number of elements in the non-periodic segment of g , so let us introduce a natural variable n for this purpose. How might we formalize the postcondition? Rather than commit ourselves prematurely to a particular formalization of periodicity, we simply let T be the number we wish to compute, and formalize the postcondition as $n = T$.

Given our constraint of small increments, possibly the simplest way to establish $n = T$ is with a so-called "linear search" . This yields the following program:

$$
\begin{array}{ll}
|[& \textbf{var } n : \text{nat} ; \\
 & n := 0 \\
 & \{ \text{ inv: } n \le T \ \} \\
\ ; & \textbf{do } \neg(n = T) \ \rightarrow \ n := n+1 \ \textbf{od} \\
 & \{ \ n = T \ \} \\
]| & \qquad ,
\end{array}
$$

which is well-known to be totally correct. Though correct, this program is unacceptable in its current form, because the guard $\neg(n = T)$ cannot be evaluated without knowing T . Thus our next aim is to refine this guard and eliminate T .

5. Refining the guard

We wish to refine the guard $\neg(n = T)$ in the above program fragment, with an eye towards eliminating T . For simplicity's sake, we put the negation aside and focus on refining subexpression $n = T$.

First we observe that by the reflexivity and antisymmetry of $\le$, $n = T$ equivales:

$$n \le T \ \wedge \ T \le n \qquad .$$

Thus, thanks to the invariant $n \le T$, we may rewrite $n = T$ equivalently as $T \le n$. In this way, though we have not succeeded in eliminating T , we have disentangled the expression, which may make it easier to refine.

We now aim to rewrite $T \le n$ as an expression that is easily evaluated, for instance an expression in g and the program variables. A little pondering reveals the simple equivalence:

$$
\begin{array}{ll}
 & T \le n \\
\equiv & \quad \{ \text{ the non-periodic segment is } [0..T) \ \} \\
 & \text{“ } n \text{ is in the periodic segment of } g \text{ ”} \qquad .
\end{array}
$$

This rewrite eliminates T and introduces g, but the expression it yields is still not easily evaluated, and hence we refine further.

The only other property left to use is that g is iteratively defined:

" n is in the periodic segment of g "
$\Leftarrow$ { g is iteratively defined — see Section 3 }
" $g.n$ occurs twice in g "
$\Leftarrow$ { one possible formalization, assuming $1 \leq m$ }
$g.n = g.(n+m)$.

Thus we have proved:

(0) $(T \leq n \Leftarrow g.n = g.(n+m)) \Leftarrow 1 \leq m$.

Expression $g.n = g.(n+m)$ is indeed easily evaluated. However, in the interest of simplicity we wish to rewrite the guard equivalently, and hence we investigate sufficient conditions for the converse condition:

$T \leq n \Rightarrow g.n = g.(n+m)$.

Writing this equivalently as:

" n is in the periodic segment of g "
$\Rightarrow$ { ??? }
$g.n = g.(n+m)$,

we see that a simple sufficient condition is that m is a multiple of the period length. Thus, letting P be the period length and letting $\sqsubseteq$ be 'divides', we have:

(1) $(T \leq n \Rightarrow g.n = g.(n+m)) \Leftarrow P \sqsubseteq m$.

From (0) and (1) we obtain the desired equivalence:

(2) $(T \leq n \equiv g.n = g.(n+m)) \Leftarrow 1 \leq m \wedge P \sqsubseteq m$.

We may summarize the results of this section in the following calculation:

$n = T$
$\equiv$ { invariant $n \leq T$ }
$T \leq n$
$\equiv$ { (2) and $1 \leq m \wedge P \sqsubseteq m$ }
$g.n = g.(n+m)$.

6. Implementing the refinement

In order to implement the refinement, we need to put $1 \leq m \ \wedge \ P \sqsubseteq m$ (the antecedent of (2)) in the context of the guard. We do this by establishing the condition as a loop invariant: Because m does not appear in our program text, the condition is not falsified by the loop, and therefore we only need to establish it initially. Accordingly, we introduce a new program variable m , and modify our program as follows:

$$
\begin{array}{ll}
|[& \textbf{var } n, m : \text{nat} ; \\
& \text{“ Establish } \ 1 \leq m \ \wedge \ P \sqsubseteq m \text{ ”} \\
& \{ \ 1 \leq m \ \wedge \ P \sqsubseteq m \ \} \\
; & n := 0 \\
; & \textbf{do } \ \neg(\, g.n \, = \, g.(n+m) \,) \ \rightarrow \ n := n+1 \ \textbf{ od} \\
& \{ \ n = T \ \} \\
]| & \qquad .
\end{array}
$$

In the remainder of the exposition, we focus on refining “ Establish... ” .

7. Establishing $1 \leq m \ \wedge \ P \sqsubseteq m$

A very standard way to establish a conjunction like $1 \leq m \ \wedge \ P \sqsubseteq m$ is to split the conjuncts between the invariant and the negation of the guard of a loop. The conjunct $1 \leq m$ makes an ideal invariant, leaving $\neg(P \sqsubseteq m)$ for the guard:

$$
\begin{array}{ll}
& m := 1 \\
& \{ \ \text{inv: } \ 1 \leq m \ \} \\
; & \textbf{do } \ \neg(P \sqsubseteq m) \ \rightarrow \ m := m+1 \ \textbf{ od} \\
& \{ \ 1 \leq m \ \wedge \ P \sqsubseteq m \ \} \qquad .
\end{array}
$$

This program fragment terminates because $m \leq P$ is a loop invariant, as the reader may verify. Because the guard $\neg(P \sqsubseteq m)$ is not easily evaluated, we refine it.

8. Refining the guard, again

As before, we wish to rewrite the guard $\neg(P \sqsubseteq m)$ in terms of g and the program variables. Towards this end we calculate:

$P \sqsubseteq m$

$\equiv$ { translation }

" m is a multiple of the period length "

$\Leftarrow$ { g is iteratively defined — see Section 3 }

" some element of g repeats at a distance of m "

$\Leftarrow$ { one possible formalization }

$g.n \;=\; g.(n+m)$.

Thus we conclude:

(3) $\quad P \sqsubseteq m \quad \Leftarrow \quad g.n \;=\; g.(n+m)$.

Also as before, we would like an equivalent guard, and so we investigate sufficient conditions for the converse of (3) :

$P \sqsubseteq m \quad \Rightarrow \quad g.n \;=\; g.(n+m)$.

But here the work has already been done for us, since by predicate calculus, (1) can be written equivalently as:

(1) $\quad (\; P \sqsubseteq m \;\; \Rightarrow \;\; g.n \;=\; g.(n+m) \;) \quad \Leftarrow \quad T \leq n$.

Thus from (1) and (3) we may conclude:

(4) $\quad (\; P \sqsubseteq m \;\; \equiv \;\; g.n \;=\; g.(n+m) \;) \quad \Leftarrow \quad T \leq n$,

yielding the desired equivalence.

Via (4) , we can rewrite guard $\neg(P \sqsubseteq m)$ as $\neg(\, g.n \;=\; g.(n+m)\,)$, provided we can place $T \leq n$ in the context of $P \sqsubseteq m$. We deal with this obligation in the next section.

9. Implementing the refinement, again

Our aim now is to implement refinement (4), by putting $T \le n$ in the context of $P \sqsubseteq m$. Unfortunately, the approach of Section 6 will not help us here, because trying to establish $T \le n$ initially will send us in circles, as the reader can verify.

The only other way to put $T \le n$ in the context of $P \sqsubseteq m$ is locally: that is, in the guard itself. Following this approach, the guard would become either:

$$T \le n \quad \wedge \quad \neg(P \sqsubseteq m)$$

or:

$$\neg(\ T \le n \quad \wedge \quad P \sqsubseteq m\ ) \qquad . \tag{5}$$

We choose (5) as our guard, because on account of invariant $1 \le m$ and all our previous results, we have:

$$\begin{array}{ll} & T \le n \quad \wedge \quad P \sqsubseteq m \\ \equiv & \quad \{\ \ 1 \le m\ ,\ \ (0)\ ,\ \ (1)\ ,\ \ (3)\ ,\ \text{and predicate calculus}\ \} \\ & g.n \ = \ g.(n+m) \qquad . \end{array}$$

(This might be considered the fundamental theorem of iteratively defined, eventually periodic sequences.)

By changing the guard to (5), we weaken it. This does not affect the partial correctness of our program, since $1 \le m$ is still a loop invariant; however, termination needs to be re-addressed. This brings us to the final part of our argument.

10. Termination

We prove termination by proving that the guard $\neg(\ T \le n\ \wedge\ P \sqsubseteq m\ )$ is eventually **false**, as follows:

Because of assignment $m := m+1$ in the loop body, predicate $P \sqsubseteq m$ is **true** periodically: every m iterations, to be precise. Thus a simple way to guarantee termination is to ensure that $T \le n$ becomes stably **true**.

The heuristics of Widening/Weakening suggest that this can be accomplished by putting any increment to n in the loop: indeed, then $T \le n$ will become stably **true** after at most T iterations. For convenience, we initialize n with $n := 0$, and increment it with $n := n+1$.

This completes the correctness argument, and we are now ready to present the solution.

11. A solution in terms of g

Our final program, in terms of g , is:

$$
\begin{array}{l}
|[\ \ \textbf{var}\ n, m : \text{nat}\ ; \\
\quad n, m := 0, 1 \\
\quad \{\ \text{inv:}\ \ 1 \le m\ \ \} \\
;\ \textbf{do}\ \ \neg(\, g.n = g.(n+m)\,)\ \ \rightarrow\ \ n, m := n+1,\, m+1\ \ \textbf{od} \\
\quad \{\ 1 \le m\ \ \wedge\ \ P \sqsubseteq m\ \} \\
;\ n := 0 \\
;\ \textbf{do}\ \ \neg(\, g.n = g.(n+m)\,)\ \ \rightarrow\ \ n := n+1\ \ \textbf{od} \\
\quad \{\ n = T\ \} \\
]| \qquad .
\end{array}
$$

12. An implementation of g

As promised, we implement g in terms of its generating function, which we call f . For convenience we let $c = g.0$. By choosing variables x and y to take on the roles of $g.n$ and $g.(n+m)$ respectively, and by choosing the appropriate invariants for x and y , we obtain the following program:

$$
\begin{array}{l}
|[\ \ \textbf{var}\ x, y\ \ ;\ \ n, m : \text{nat}\ \ ; \\
\quad n, m := 0, 1\ ;\ \ x, y := c, f.c \\
\quad \{\ \text{inv:}\ \ x = g.n\ \ \wedge\ \ y = g.(n+m)\ \ \wedge\ \ n+1 = m\ \ \} \\
;\ \textbf{do}\ \ \neg(x = y)\ \ \rightarrow\ \ n, m := n+1,\, m+1\ ;\ \ x, y := f.x,\, f.(f.y)\ \ \textbf{od} \\
\quad \{\ x = g.n\ \ \wedge\ \ n+1 = m\ \} \\
;\ n := 0\ ;\ \ x, y := c,\, f.x \\
\quad \{\ \text{inv:}\ \ x = g.n\ \ \wedge\ \ y = g.(n+m)\ \ \} \\
;\ \textbf{do}\ \ \neg(x = y)\ \ \rightarrow\ \ n := n+1\ ;\ \ x, y := f.x,\, f.y\ \ \textbf{od} \\
\quad \{\ n = T\ \} \\
]| \qquad .
\end{array}
$$

The correctness proofs for the new assertions are left as an exercise for the industrious reader. They do not involve much more than the Axiom of Assignment.

13. The final program

Finally, we project this program onto the variables and statements which are relevant for the computation of the desired postcondition:

$$
\begin{array}{l}
|[\quad \textbf{var}\ x, y\ ;\quad n : \text{nat}\ ; \\
\quad\ x, y := c, f.c \\
\ ;\ \textbf{do}\ \ \neg(x = y)\ \ \rightarrow\ \ x, y := f.x,\ f.(f.y)\ \ \textbf{od} \\
\ ;\ n := 0\ ;\ \ x, y := c,\ f.x \\
\ ;\ \textbf{do}\ \ \neg(x = y)\ \ \rightarrow\ \ n := n{+}1\ ;\ \ x, y := f.x,\ f.y\ \ \textbf{od} \\
\quad\ \{\ n = T\ \} \\
]| \qquad .
\end{array}
$$

Mumbai, India and Santa Cruz, CA ; 5 July 2007

Apurva Mehta
apurva@mathmeth.com

Jeremy Weissmann
jeremy@mathmeth.com