Programming and Reasoning

Anne Kaldewaij

April 1, 1999

Abstract

A programming problem and a solution thereof are presented. It is based on the co-operative work in our course on Programming and Reasoning, a unique project between the fields of logic and of program derivation. Just for fun.

Contents

1 Introduction

About three years ago, Johan came to me with the idea to set up a joint lecture for first year undergraduate students. A project in which logic and program derivation would be introduced in such a way, that the combination thereof would lead to a coherent and inspiring series of lectures. And so we did.

The outcome was quite successful, it even resulted in an Award for "best lecture" in 1997. However, the most successful part for me was the close cooperation between us, and the way in which we learned from each other. The appreciation of the lectures by the students was particularly due to our own enthousiasm, which was rooted in this close cooperation.

A collaboration between logicians and programmers is quite rare. The different ways in which programmers use logic in program derivation and logicians use semantics to study programs (amongst many other things) seems usually to lead to highly debated differences of insight. Not with us, however, since we were interested in what the other one was doing, and how we could benefit from it.

In this little note, I will treat a small programming exercise of the kind that Johan likes. Which means that it is not about numbers or about "calculations". It is about recognition.

The problem originates from the Eindhoven programming group. I don't know its inventor, but E.W. Dijkstra might be a good guess. Parts of the development of the solution to it rely on logic, of course.
Although it may be caracterized as "just for fun", which it is anyway, the presented problem and its solution represent some of the flavour of the lectures mentioned above.

2 Recognizing h-sequences

The problem is stated as follows.
An *h-sequence* is either a single 0 or it is a 1 followed by two h-sequences. A grammer of h-sequences may be given by

$$\langle h \rangle ::= 0 \mid 1\langle h \rangle\langle h \rangle$$

Examples of h-seqences are

0
100
10100
110010100
110100100

The programming problem is to determine for a given array $x[0..N)$ of N elements, $N \geq 0$, wether x is an h-sequence or not. For the sake of convenience, we assume that sequence x contains zeroes and ones only.

The way in which people look at this problem depends very much on their background. I have no idea what a "typical logician" would think of. An average computer scientist might think of a parser and try to develop a recursive function, that follows the pattern of the definition above. A mathematician might try to characterize h-sequences in a more manageable way.
Some typical properties of h-sequences are

- the length is at least one
- it ends on a 0
- if the length is greater than one, it starts with a 1
- the lenght is odd
- the number of zeroes is greater than the number of ones

Indeed, it is possible to extend this list of properties in such a way that a complete (and sound) characterization of h-sequences is obtained, which might then be used to derive a program for recognition.

We will not do so, because, as far as I know, the resulting programs are not too nice and the corresponding a posteriori correctness proofs are quite cumbersome. In the next section, we show a solution, that does not use properties like the ones above.

3 A Solution

From the definition of h-sequences,

$$\langle h \rangle ::= 0 \mid 1\langle h \rangle\langle h \rangle$$

we infer that the only h-sequence starting with 0 is the one-element sequence consisting of 0 alone. Hence, if $x[0]=0$, the seqence $x[0..N)$ is an h-sequence if and only if $N=1$.
If, on the other hand, $x[0]=1$, the definition yields, that $x[0..N)$ is an h-sequence if and only if $x[1..N)$ is the concatenation of 2 h-sequences. We generalize this last equivalence by replacing constants 1 and 2 by program variables k and m. Their meaning is given by predicate P, defined as

$$x[0..N) \text{ is an h-sequence } \Leftrightarrow \ x[k..N) \text{ is the concatenation of } m \text{ h-sequences}$$

For $k=0$ and $m=1$, predicate P is obviously true, since it then reads

$$x[0..N) \text{ is an h-sequence } \Leftrightarrow \ x[0..N) \text{ is the concatenation of 1 h-sequences}$$

This means that, when we use P as an *invariant* of a repetition, it is established by the assignments $k := 0\,;\ m := 1$.

We now concentrate on the right-hand side of the equivalence:

$$x[k..N) \text{ is the concatenation of } m \text{ h-sequences}$$

Its value (true or false) is easily computed when $k=N \vee m=0$ holds. Indeed, for $k=N$, the right-hand side reads the empty sequence is the concatenation of m h-sequences

which is true for $m = 0$ and false otherwise.

For $m = 0$ it reads

$x[k..N)$ is the concatenation of 0 h-sequences

which is true for $k = N$ (the empty sequence) and false otherwise. This leads to

$$k \neq N \wedge m \neq 0$$

as condition for a repetition. From the analysis above, we conclude

$$\begin{array}{lcl} P \wedge k = N & \Rightarrow & x \text{ is h-sequence} \Leftrightarrow m = 0 \\ P \wedge m = 0 & \Rightarrow & x \text{ is h-sequence} \Leftrightarrow k = N \end{array}$$

For $k < N$, element $x[k]$ can be inspected.
If $x[k] = 0$, the first element of $x[k..N)$ forms an h-sequence, and

$x[k..N)$ is the concatenation of m h-sequences

is equivalent to

$x[k+1..N)$ is the concatenation of $m-1$ h-sequences

If $x[k] = 1$, we have

$x[k..N)$ is the concatenation of m h-sequences

is equivalent to

$x[k+1..N)$ is the concatenation of $m+1$ h-sequences

From these equivalences it follows how invariant P is maintained when k is replaced by $k+1$. The corresponding program reads

```
var k,m: integer;
k:=0; m:=1;
{invariant:P}
do k≠N ∧ m≠0
  →  if x[k]=1 → m:=m+1 [] x[k]=0 → m:=m-1 fi;
     k:=k+1
od;
if k=N → ish:=m=0 [] m=0 → ish:=k=N fi
{ish = x is an h-sequence}
```

But we are not completely satisfied. The last selection statement is of the form

$$\text{if B} \rightarrow \text{ish:=C} \;[\!]\; \text{C} \rightarrow \text{ish:=B fi}$$

Thanks to Logic and the rules of the selection, we know that this selection may be replaced by the much simpler

$$\text{ish:=B}\wedge\text{C}$$

The final program is obtained by this replacement.

```
var k,m: integer;
k:=0; m:=1;
do k≠N ∧ m≠0
  →  if x[k]=1 → m:=m+1 [] x[k]=0 → m:=m-1 fi;
     k:=k+1
od;
ish:= k=N∧m=0
```

4 Concluding Remarks

When all steps in the derivation of the program presented in the previous section are carried out in detail, most of it is pure logic. Formal proofs of the invariance and of termination require knowledge and experience in calculating with predicates.

For those who have tried to solve the problem themselves (as the first year students did), the solution presented is a real surprise. It might be called a toy program, and indeed, it is not meant to solve a large class of todays software problems.

These small problems from logic and programming, however, serve as a very good start of one's education as a professional in computer science.

Moreover, it are these kinds of problems and their solutions that are liked by us. And by many others, fortunately.

It has been a privilege to share my ideas with Johan and to have experienced his way of conveying a variety of logics and applications thereof. I hope we will be able to continue further cooperation in education and research.

I apologize for always being so busy with the less exciting stuff.