

# Programming by Expression Refinement: the KMP Algorithm

Joseph M. Morris

Department of Computing Science  
University of Glasgow  
Glasgow G12 8QQ  
Scotland, UK

## 0 Introduction

We carry out a small exercise in programming by what might be called *expression refinement*. This is a style of formal programming in which we begin with an expression written in an expressive notation and regarded as a specification, and proceed to manipulate it into a constructive equivalent. This leads to programs making much use of recursive functions and less use of loops.

The exercise is to calculate a pattern-matching algorithm, and specifically the algorithm originally due to Knuth, Morris, and Pratt [0]. We begin, however, with a smaller problem, one that turns out to be both similar to, and a subproblem of, the larger problem. We prefer to start with this because it will give the reader a chance to become familiar in a simple setting with the style and notation we employ.

We briefly review some basic notation. Function application is denoted by an infix dot which has the highest operator precedence. Finite sequences are regarded as partial functions on an initial segment of the natural numbers: sequence  $x$  of length  $N$ ,  $N$  a natural, has elements  $x.0, x.1, \dots, x.(N-1)$ . The  $n$ -length prefix of  $x$ ,  $0 \leq n \leq N$ , is denoted by  $x|n$ . We denote " $x$  is a suffix of  $y$ " by  $x \geq y$ , and " $x$  is a proper suffix of  $y$ " by  $x > y$ , where  $y$  is another sequence.  $(k: P.k: f.k)$  denotes the bag (multiset) containing an occurrence of  $f.k$  for each  $k$  satisfying predicate  $P.k$  ( $k$  is a dummy variable).  $\max.B$  yields the maximum of bag  $B$  of integers and equals  $-\infty$  if  $B$  is empty. Infix **max** yields the maximum of its integer arguments. A useful law is

$$\max.(k: P.k: f.k) = \max.(k: P.k \wedge Q.k: f.k) \mathbf{max} \max.(k: P.k \wedge \neg Q.k: f.k)$$

where  $Q.k$  is another predicate; replacing an occurrence of the left-hand side of this equation with its right-hand side is called "range splitting". A similar sort of law, called "range disjunction", is

$$\max.(k: P.k \vee Q.k: f.k) = \max.(k: P.k: f.k) \mathbf{max} \max.(k: Q.k: f.k)$$

## 1 Maximal Prefix-Suffix Problem

We want to make a program that calculates for every non-empty prefix  $y$  of a given string  $x$  the longest string that is both a proper prefix and a proper suffix of  $y$ . We write down the formal specification and then we'll explain it; for any natural  $M$ :

MPS0:  $x$ : sequence( $M$ ) of char;  $q$ : sequence( $M+1$ ) of integer  
 $\{ x \in \text{sequence}(M) \text{ of char} \}$   
 $(\|n: 0 < n \leq M:$   
 $\quad q.n := \max.(k: 0 \leq k < n \wedge x|k > x|n: k)$   
 $)$

The names and types of global variables – here  $x$  and  $q$  – are given above the horizontal line. The specification proper is given below the line, and consists of the assumptions and a statement of the desired effect. The assumption is given as a so-called assert statement, and simply says that  $x$  has an initial value. The desired effect is here stated with a large concurrent assignment statement meaning "let  $q.n$  have the value of the max-term for each  $n$  in the range  $0 < n \leq M$ ".

We introduce some abbreviations:

D0:  $k \mathbf{xx} n \equiv x|k > x|n$  for all  $k$  and  $n$  satisfying  $0 \leq k, n \leq M$ .  
D1:  $mx.n = \max.(k: 0 \leq k < n \wedge k \mathbf{xx} n: k)$  for all  $n$  such that  $0 < n \leq M$ .

Obviously  $mx.n$  equals the right-hand side of the assignment in MPS0. We should begin by writing down the elementary properties of  $\mathbf{xx}$  and  $mx$  that follow immediately from D0 and D1.

L0:  $0 \mathbf{xx} n$  (0 < n ≤ M)  
L1:  $0 \leq mx.n < n$  (0 < n ≤ M)  
L2:  $mx.n \mathbf{xx} n$  (0 < n ≤ M)  
L3:  $k \mathbf{xx} n \equiv k \leq mx.n$  (0 ≤ k < n ≤ M)

We will be inventing some string-theory as we proceed, such as the preceding laws. The proofs of such laws are peripheral to the main theme, and so we'll omit them for brevity, but we expect the reader will not have much difficulty in convincing himself of their truth. We give  $\mathbf{xx}$  an operator precedence below the arithmetic and above the logical operators, so we can write  $k+1 \mathbf{xx} n+1$ , say, without brackets.

Looking at the specification we view the essence of our task as that of formulating  $mx.n$  constructively for  $0 < n \leq M$ , or equivalently,  $mx.(n+1)$  for  $0 \leq n < M$ :

$mx.(n+1)$   
="D1"  
 $\max.(k: 0 \leq k < n+1 \wedge k \mathbf{xx} n+1 : k)$

*Interlude.* The first step introduces one interesting term –  $k \mathbf{xx} n+1$ . We are on the lookout for a formulation of the original expression (i.e.  $mx.(n+1)$ ) in terms of  $mx.i$ 's for  $i \leq n$ , and that suggests trying to express  $k \mathbf{xx} n+1$  in terms of  $k \mathbf{xx} n$ . This is not hard to do, but it's more convenient to rewrite  $k+1 \mathbf{xx} n+1$ :

L4:  $k+1 \mathbf{xx} n+1 \equiv k \mathbf{xx} n \wedge x.k = x.n$  (0 ≤ k ≤ n < M)

We continue with a minor reshaping of the expression to accommodate L4.

*End.*

```

="range splitting with predicate k=0"
  max.(k: 0<k<n+1  $\wedge$  k xx n+1: k) max max.(k: k=0  $\wedge$  k xx n+1: k)
="one-point rule, L0"
  max.(k: 0<k<n+1  $\wedge$  k xx n+1: k) max 0
="change of dummy k to k+1"
  max.(k: 0 $\leq$ k<n  $\wedge$  k+1 xx n+1: k+1) max 0
="L4"
  max.(k: 0 $\leq$ k<n  $\wedge$  k xx n  $\wedge$  x.k = x.n: k+1) max 0 (i)
="We could contract the range of k via L3, but we must exclude n=0"
  if n=0  $\rightarrow$  0
  [] n>0  $\rightarrow$  max.(k: 0 $\leq$ k<n  $\wedge$  k xx n  $\wedge$  x.k = x.n: k+1) max 0
  fi
="L3; mx.n<n by L1"
  if n=0  $\rightarrow$  0
  [] n>0  $\rightarrow$  max.(k: 0 $\leq$ k $\leq$ mx.n  $\wedge$  k xx n  $\wedge$  x.k = x.n: k+1) max 0
  fi
="We cannot escape comparing a pair of characters. Examining the
max-expression with an eye on L2 suggests we might go about this by
isolating the case k=mx.n, by range splitting"
  if n=0  $\rightarrow$  0
  [] n>0  $\rightarrow$  max.(k: k=mx.n  $\wedge$  k xx n  $\wedge$  x.k = x.n: k+1)
    max max.(k: 0 $\leq$ k<mx.n  $\wedge$  k xx n  $\wedge$  x.k = x.n: k+1) max 0
  fi
="one-point, L2"
  if n=0  $\rightarrow$  0
  [] n>0  $\rightarrow$  if x.(mx.n)=x.n  $\rightarrow$  mx.n+1
    [] x.(mx.n) $\neq$ x.n  $\rightarrow$  - $\infty$ 
    fi max max.(k: 0 $\leq$ k<mx.n  $\wedge$  k xx n  $\wedge$  x.k = x.n: k+1) max 0
  fi
="distribution of terms over if...fi and simplification — mx.n+1 > 0 by L1,
and mx.n+1 > max.(....) as the largest element in the bag is at most mx.n"
  if n=0  $\rightarrow$  0
  [] n>0  $\rightarrow$  if x.(mx.n)=x.n  $\rightarrow$  mx.n+1
    [] x.(mx.n) $\neq$ x.n  $\rightarrow$ 
      max.(k: 0 $\leq$ k<mx.n  $\wedge$  k xx n  $\wedge$  x.k = x.n: k+1) max 0
    fi
  fi

```

*Interlude.* We are left with the difficult-looking max-term in the second guarded expression. Does it look like anything we've seen? It is similar to (i) – indeed it is (i) with the first  $n$  therein replaced with  $mx.n$ , and so we are invited to parametrise that occurrence on our way to completing a recursive definition of  $mx$ . We have to fulfil certain conditions before we parametrise. Firstly, we have to ensure that the substitution is "progressive", by which we mean that the substituted term is smaller than the term it replaces according to some well-founded ordering; otherwise nothing is gained. By L1 the substitution of  $mx.n$  for  $n$  is progressive, where the ordering is the usual  $\leq$  on the naturals. Secondly, we have to ensure that the substituted term has those properties of the replaced term that were used in the derivation, so that repeated such substitutions are valid inductively. Stepping through the derivation with the substitution in mind, we see that this is not the case: we get stuck where we apply L3 as that relies on the first two occurrences of  $n$  being one and the same, and we also fall foul of the application of L2 for the same reason. We conclude that there is no advantage in parametrising the first occurrence of  $n$  without including the second occurrence in the parametrisation. Evidently, the third occurrence of  $n$  in (i) is independent of the other two. If only that  $k$  **xx**  $n$  in the final inner max-term was  $k$  **xx**  $mx.n$ .....

L5:  $k \text{ xx } n \equiv k = mx.n \vee k \text{ xx } mx.n$  (0 ≤ k ≤ M, 0 < n ≤ M)

End.

= "L5"

```

if n=0 → 0
[] n>0 → if x.(mx.n)=x.n → mx.n+1
           [] x.(mx.n)≠x.n →
             max.(k: 0 ≤ k < mx.n ∧ k xx mx.n ∧ x.k = x.n: k+1) max 0
           fi
fi

```

Now the parametrisation goes through: we can replace the two indicated occurrences of  $n$  in (i) with any  $i$  satisfying  $0 \leq i \leq n$ , and the same derivation applies mutatis mutandis; we leave it to the reader to check this. It is now routine to extract a well-founded recursive definition. With fixed  $n$  satisfying  $0 \leq n < M$  define function  $f$  by

$$f.i = \max.(k: 0 \leq k < i \wedge k \text{ xx } i \wedge x.k = x.n: k+1) \text{ max } 0 \quad (0 \leq i < M)$$

We have shown – in the above derivation up to (i) – that  $mx.(n+1) = f.n$ . We have further shown – in the derivation from (i) and the above discussion – that for  $0 \leq i \leq n$ :

```

f.i = if i=0 → 0
      [] i>0 → if x.(mx.i)=x.n → mx.i+1
                [] x.(mx.i)≠x.n → f.(mx.i)
                fi
      fi

```

It follows from L1 and an easy inductive argument that in an evaluation of  $f.n$  the arguments of  $f$  are decreasing and bounded from below by 0, and so  $f$  is terminating. It further follows — taking a look at the text — that in an evaluation of  $f.n$  every argument  $i$  of  $f$  satisfies  $0 \leq i \leq n$ , and hence the evaluation refers only to those  $mx.i$ 's satisfying  $0 < i \leq n$ . This makes it attractive to evaluate the  $mx.n$ 's in order of increasing  $n$ , for then each  $mx.i$  that we need will be available in  $q.i$ . With loop invariant

$$0 \leq n \leq M \wedge (\forall i: 0 < i \leq n: q.i = mx.i)$$

the program is:

```

MPS:  [| n: integer
        ;n:= 0
        ;do n≠M
        → [| f: integer func =
              val i: integer
              ;if i=0 → 0
              [] i>0 → if x.(q.i)=x.n → q.i+1
                        [] x.(q.i)≠x.n → f.(q.i)
                        fi
              fi •
              q.(n+1):= f.n
        ] |
        ;n:= n+1

```

od  
 ]]

## 2 Knuth-Morris-Pratt

We want to compute all occurrences of string  $x$  of length  $M$  in another string  $y$  of length  $N$ ,  $M$  and  $N$  naturals:

KMP0:  $x$ : sequence( $M$ ) of char;  $y$ : sequence( $N$ ) of char;  $s$ : set of integer  
 $\{ x \in \text{sequence}(M) \text{ of char} \wedge y \in \text{sequence}(N) \text{ of char} \}$   
 $s := \{ n : 0 < n \leq N \wedge x \geq y|n : n-M \}$

The task is essentially one of determining the truth-value of terms  $x \geq y|n$  for  $0 < n \leq N$ . One way to evaluate  $x \geq y|n$  is to compute the length of the longest prefix of  $x$  that is a suffix of  $y|n$  – call this  $my.n$ , for then  $x \geq y|n \equiv my.n = M$ ; we will proceed with this. We should note that other choices were open to us: we might have opted to look at the longest common suffix of  $x$  and  $y|n$ , or the longest prefix common to  $x$  and the  $M$ -length tail of  $y|n$ , and so on, but we have made a choice that leads to the Knuth-Morris-Pratt algorithm. We define

D2:  $k \text{ xy } n \equiv x|k \geq y|n$  for  $k$  and  $n$  satisfying  $0 \leq k \leq M$ ,  $0 \leq n \leq N$ .  
 D3:  $my.n = \max.(k : 0 \leq k \leq M \wedge k \text{ xy } n : k)$  for all  $n$  such that  $0 \leq n \leq N$ .

KMP0 is refined by (not bothering to repeat the globals and assumptions):

KMP1:  $s := \{ n : 0 < n \leq N \wedge M = my.n : n-M \}$

$\text{xy}$  has the same operator precedence as  $\text{xx}$ . Let us write down the elementary properties that follow immediately from D2 and D3:

L6:  $0 \text{ xy } n$  ( $0 \leq n \leq N$ )  
 L7:  $0 \leq my.n \leq M$  ( $0 \leq n \leq N$ )  
 L8:  $my.n \text{ xy } n$  ( $0 \leq n \leq N$ )  
 L9:  $k \text{ xy } n \equiv k \leq my.n$  ( $0 \leq n \leq N$ ,  $0 \leq k \leq M$ )

To evaluate  $M = my.n$  is not at all difficult, but we should sense immediately that to evaluate it in isolation for every  $n$  is computationally expensive, and probably unnecessary. There is obviously some close relationship between successive  $my.i$ 's, and we should try to exploit this for the sake of computational efficiency. So we view the essence of the problem as one of formulating  $my.n$  constructively. It is clear that  $my.n$  is very similar indeed to  $mx.n$  of the previous section, and so we should expect our first approach to follow a similar developmental path. Where we apply the same heuristics as before we will not repeat the discussion. For  $n$  satisfying  $0 \leq n < N$ :

$my.(n+1)$   
 ="D3"  
 $\max.(k : 0 \leq k \leq M \wedge k \text{ xy } n+1 : k)$

*Interlude.* Analogous to L4 we have

L10:  $k+1 \text{ xy } n+1 \equiv k \text{ xy } n \wedge x.k = y.n$  ( $0 \leq k < M$ ,  $0 \leq n < N$ )

*End.*

="range splitting with  $k=0$ , L6"  
 $\max.(k: 0 < k \leq M \wedge k \mathbf{xy} \ n+1: k) \mathbf{max} \ 0$   
 ="change of dummy  $k$  to  $k+1$ "  
 $\max.(k: 0 \leq k < M \wedge k+1 \mathbf{xy} \ n+1: k+1) \mathbf{max} \ 0$   
 ="L10"  
 $\max.(k: 0 \leq k < M \wedge k \mathbf{xy} \ n \wedge x.k = y.n: k+1) \mathbf{max} \ 0$  (ii)

*Interlude.* We are proceeding very much as we did for the earlier problem. If things continue to go as previously then we should meet the expression (ii) but for some substitution. But there is a small obstacle in the way: here is the  $\mathbf{xy}$ -law analogous to L5

L11:  $k \mathbf{xy} \ n \equiv k = \mathbf{my}.n \vee k \mathbf{xx} \ \mathbf{my}.n \quad (0 \leq k \leq M, 0 \leq n \leq N)$

Now an application of L11 is different from L5 in this regard, that it replaces an  $\mathbf{xy}$ -term not with another term of its own kind but with an  $\mathbf{xx}$ -term. If we are to meet (ii) again but for a substitution, then at this point we should replace its  $\mathbf{xy}$ -term with an  $\mathbf{xx}$ -term – using L11 – and that we shall do. In practice one discovers the efficacy of this move by proceeding with the derivation much as for  $\mathbf{mx}$ , and meeting a subexpression the same as (ii) but for the expected substitution *and*  $\mathbf{xx}$  where (ii) has  $\mathbf{xy}$ . We could proceed along that path, make the discovery, and backtrack – but presumably the reader is happy to be spared the effort. *End.*

="L11"  
 $\max.(k: 0 \leq k < M \wedge (k = \mathbf{my}.n \vee k \mathbf{xx} \ \mathbf{my}.n) \wedge x.k = y.n: k+1) \mathbf{max} \ 0$  (iii)  
 ="let  $m$  abbreviate  $\mathbf{my}.n$ "  
 $\max.(k: 0 \leq k < M \wedge (k = m \vee k \mathbf{xx} \ m) \wedge x.k = y.n: k+1) \mathbf{max} \ 0$  (iv)  
 ="calculus and range disjunction"  
 $\max.(k: 0 \leq k < M \wedge k = m \wedge x.k = y.n: k+1) \mathbf{max}$   
 $\max.(k: 0 \leq k < M \wedge k \mathbf{xx} \ m \wedge x.k = y.n: k+1) \mathbf{max} \ 0$   
 ="max theory,  $m \geq 0$  by definition of  $m$  and L7"  
**if**  $m < M \wedge x.m = y.n \rightarrow m+1$   
 $\square \neg(m < M \wedge x.m = y.n) \rightarrow -\infty$   
**fi max**  
 $\max.(k: 0 \leq k < M \wedge k \mathbf{xx} \ m \wedge x.k = y.n: k+1) \mathbf{max} \ 0$   
 ="distribution of terms over **if**...**fi**; from L3 and L1 the maximum  $k$ -value in the bag of the middle term is at most  $m$ , and so the maximum value in the bag is no more than  $m+1$ ;  $0 < m+1$  by definition of  $m$  and L7"  
**if**  $m < M \wedge x.m = y.n \rightarrow m+1$   
 $\square \neg(m < M \wedge x.m = y.n) \rightarrow$   
 $\max.(k: 0 \leq k < M \wedge k \mathbf{xx} \ m \wedge x.k = y.n: k+1) \mathbf{max} \ 0$   
**fi**

*Interlude.* We are aiming to give the max-expression above the same shape as (iv). Obviously we think of applying L5, but for that we must exclude the case  $m = 0$ . We can do that via the obvious:

L12:  $\neg(k \mathbf{xx} \ 0)$  (0  $\leq k \leq M$ )

*End.*

="L12, max theory,  $m \geq 0$  by definition of  $m$  and L7"  
**if**  $m < M \wedge x.m = y.n \rightarrow m+1$   
 $\square \neg(m < M \wedge x.m = y.n) \rightarrow$   
**if**  $m = 0 \rightarrow 0$

```

    [] m>0 → max.(k: 0≤k<M ∧ k xx m ∧ x.k = y.n: k+1) max 0
  fi
fi
if m<M ∧ x.m=y.n → m+1
[] ¬(m<M ∧ x.m=y.n) →
  if m=0 → 0
  [] m>0 →
    max.(k: 0≤k<M ∧ (k=mx.m ∨ k xx mx.m) ∧ x.k = y.n: k+1) max 0
  fi
fi

```

The inner max-expression is (iv) with  $m$  replaced by  $mx.m$ , and so we are invited to make a parameter of  $m$ . Can we? In the derivation  $m$  denotes  $my.n$ , but looking back through the steps the only property of  $m$  we appealed to was  $m \geq 0$ . Now the substitution we propose for  $m$  (having established  $m > 0$ ) is  $mx.m$  which by L1 is also at least 0, and so on inductively. Moreover, L1 guarantees that this substitution is progressive. So we can extract the recursive function: with fixed  $n$  satisfying  $0 \leq n < N$  define function  $g$  by

$$g.i = \max.(k: 0 \leq k < M \wedge (k=i \vee k \mathbf{xx} i) \wedge x.k=y.n: k+1) \mathbf{max} 0 \quad (0 \leq i \leq M)$$

We have established – in the above derivation as far as (iii) – that for  $0 \leq n < N$ ,  $my.(n+1) = g.(my.n)$ . We have further shown – in the derivation from (iv) on, and in the discussion above – that

```

g.i = if i<M ∧ x.i=y.n → i+1
      [] ¬(i<M ∧ x.i=y.n) →
        if i=0 → 0 [] i>0 → g.(mx.i) fi
      fi

```

and moreover that this recursive definition is well-founded.

Function  $g$  is computationally very attractive: it defines  $my.(n+1)$  as a function depending only on  $my.n$  and the  $mx.i$ 's. So if we evaluate  $M=my.n$  in increasing order of  $n$  we need only keep one old  $my$ -value, as well as the  $mx.i$ 's which we can compute in advance. The invariant we need has a standard form

$$0 \leq n \leq N \wedge s = \{i: 0 < i \leq n \wedge M = my.i: i-M\} \wedge m = my.n \wedge (\forall i: 0 < i \leq M: q.i = mx.i)$$

The obvious initial values for  $n$ ,  $s$ , and  $m$  are 0,  $\emptyset$ , and 0, respectively –  $my.0=0$  follows easily from definitions D2 and D3. The obvious termination condition is  $n=N$ , but here we can make a minor optimisation. Observe that  $my.n$  grows no faster than  $n$ . Now a pattern match at position  $i$  of  $y$ ,  $n < i \leq N$  implies  $my.i=M$ , which is equivalent to  $my.i-my.n=M-my.n$ , which implies  $i-n \geq M-my.n$ , which implies  $N-n \geq M-my.n$ , which is equivalent to  $N-n \geq M-m$  – so when this becomes false we may terminate. The program is:

```

[] n,m: integer; q: sequence(M) of integer
;n:= 0 [] s:= ∅ [] m:= 0
;MPS
;do N-n ≥ M-m →
  [] g: integer func =
    val i: integer;
    if i<M ∧ x.i=y.n → i+1
    [] ¬(i<M ∧ x.i=y.n) →

```

```

        if i=0 → 0 [] i>0 → g.(q,i) fi
    fi •
    m:= g.m
  ]]
  ;n:= n+1
  ;if m=M → s:= s ∪ {n-M} [] m<M → skip fi
od
]]

```

The two conjunctions in the above code are "conditional", but a standard transformation will remove them if desired. A minor gain in efficiency can be got by converting the two recursive functions into loops, which is again a standard transformation, but one unlikely to be worth the trouble. Note that we didn't need to call upon L8 or L9.

### 3 Conclusion

Although the derivation above has of course been polished, it was arrived at pretty much as presented – with a few false moves here and there, but no great difficulties. My experience has been that developing the program by expression refinement is much easier than via invariant relations and loops alone. (See J. van der Woude [1], however, for a convincing derivation using loop invariants; indeed some of our inspiration has come from [1].) The most time-consuming part of the initial effort was experimenting with different notations. There is a basic choice: to use a string-based notation, or a notation based on indices of fixed strings – which is what was used eventually. The former seems on the face of it to be neater, and although it was succinct and malleable for parts of the derivation, in other parts it proved cumbersome. String indices, on the other hand, proved consistently workable, but they are tiresome to read – indeed the derivation may be accused of suffering from what has been called "indexitis". Indexitis apart, as an experiment in programming by expression refinement the outcome seems satisfactory, and the same approach has worked well on other problems. Experience suggests that programming by expression refinement is a more attractive basis for developing a fully formal practical programming methodology than one based solely on statement refinement.

### References

- [0] D. E. Knuth, J. H. Morris Jr, and V. R. Pratt, *Fast pattern matching in strings*, SIAM J. Comput. **6**, 323-350, 1977.
- [1] J. van der Woude, *Playing with patterns, searching for strings*, Computing Science Report 87/13, Eindhoven University of Technology, The Netherlands, 1987.