Programming by Expression Refinement: the KMP Algorithm

Joseph M. Morris

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 [1]. 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 $x.0, x.1, \ldots, x.(N-1)$. 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). $\text{max} . B$ yields the maximum of bag $B$ of integers and equals $-\infty$ if $B$ is empty. Infix $\text{max}$ yields the maximum of its integer arguments. A useful law is

$$\text{max}. (k : P.k : f.k) = \text{max}. (k : P.k \land Q.k : f.k) \text{ max } \text{max}. (k : P.k \land \neg Q.k : f.k)$$

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".
similar sort of law, called “range disjunction”, is
\[
\max (k : P.k \lor Q.k : f.k) = \\
\max (k : P.k : f.k) \max \max (k : Q.k : f.k)
\]

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
\[
\{ x \in \text{sequence}(M) \text{ of char} \}
\]
\[
(|n : 0 < n \leq M : \\
q.n := \max (k : 0 \leq k < n \land x|k > x|n : k)
\}
\]

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: \(k xx n \equiv x|k > x|n\) for all \(k\) and \(n\) satisfying \(0\leq k, n\leq M\).
D1: \(mx.n = \max (k : 0 \leq k < n \land k xx n : k)\)
\[\text{for all } n \text{ such that } 0 < n \leq M.\]

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

L0: \(0 xx n\) \((0 < n \leq M)\)
L1: \(0 \leq mx.n < n\) \((0 < n \leq M)\)
L2: \(mx.n xx n\) \((0 < n \leq M)\)
L3: \(k xx n \Rightarrow k \leq mx.n\) \((0 \leq k < n \leq M)\)

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 \(xx\) an operator