The Derivation of a Hierarchy of Algorithms for Pattern Matching on Arrays

Johan Jeuring*
CWI
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands
(jt@cwi.nl)

1 Introduction

The pattern-matching problem can be posed on all 'structured' data types: given a pattern and a subject, determine an occurrence of the pattern in the subject. For example, if P is a u by v rectangular two-dimensional array, and S is an m by n array of the same type, the problem is to find a pair (i, j) such that for all k and l such that $u \geq k \geq 1$ and $v \geq l \geq 1$

$$(1) \quad S[i-u+k, j-v+l] \;=\; P[k, l] \; .$$

This description of the two-dimensional pattern-matching problem is taken from Baker [2]. For higher dimensional arrays similar definitions of matching can be given. This paper derives a hierarchy of algorithms for pattern matching on arrays in the Bird-Meertens calculus for program transformation. In this calculus, both specifications and algorithms are functions, and a few high-level theorems are used as transformation rules. An algorithm is derived from its specification by means of a calculation which typically consists of a sequence of equalities, each an instantiation of a high-level theorem or a definition. Aspects of the Bird-Meertens calculus can be found in [4], [5], [9], [14], [15], and [13].

The laws we use in the derivation are derived from the definition of the data type *array*. The data type *array* can be defined in various ways. For a specific problem a suitable definition has to be chosen, and this choice may differ for different problems. Here, as in Jeuring [8], we define arrays as nested lists. The definition of the data type *array* given in [8] is slightly modified and as a result the definition of several functions given in [8], such as the function that returns all subarrays of an array (a subarray of an array is the extension of the notion of substring or segment on lists to arrays), are shorter. Other definitions of the data type array have been given by Bird [5], and Mullin [16].

*This research has been supported by the Dutch organisation for scientific research under project-nr. NF 62.518.

Pattern matching algorithms on arrays have been described before. The algorithms for pattern matching from Knuth et al. [12] and Aho and Corasick [1] can be used in an algorithm for two-dimensional pattern matching. This algorithm is described by Bird [3] and Baker [2]. Baker also notices the existence of (but does not give) a hierarchy of algorithms for pattern matching on arrays. Karp, Miller and Rosenberg [11] give an algorithm for finding repeated occurrences of square subarrays. This algorithm can be adjusted to deal with pattern matching. Although this algorithm is well suited for parallel implementation, see Crochemore and Rytter [7], it is inefficient compared with the aforementioned sequential algorithms.

This paper is organised as follows. Section 2 defines the data type *array*, and gives some laws. Section 3 specifies the pattern-matching problem in terms of subarrays and Section 4 sketches the derivation of a hierarchy of efficient algorithms for pattern matching on arrays. For reasons of space, most of the proofs are omitted; these proofs can be found in [10].

2 The data type array

This section defines the data type *array*. The data type *array* is an example of a hierarchical data type: data types of which the nth component ($n > 1$) is expressed in terms of the ($n-1$)th component, see Jeuring [10]. We give a definition of the data type *zero-dimensional array*, denoted by $A\star_0$, of the data type *one-dimensional array*, denoted by $A\star$, and we define the data type *n-dimensional array* ($n > 1$) in terms of the data type ($n-1$)*-dimensional array*.

Elements of the data type *zero-dimensional array*, which is denoted by $A\star_0$, are scalars: elements of simple types like *bool* and *nat* (the data type natural numbers). The data type *one-dimensional array* over base type A, denoted by $A\star$, is the data type *snoc-list* which is informally defined as follows (for a formal definition the reader is referred to Malcolm [13]). The empty list $\square$ is a snoc-list, so $\square : A\star$, and if x is a snoc-list and a is an element of type A, then $x \mathbin{-\!\!\!<} a$ is a snoc-list, so $\mathbin{-\!\!\!<} : A\star \times A \to A\star$. The list with consecutive elements 1, 2 and 8, formally $\square \mathbin{-\!\!\!<} 1 \mathbin{-\!\!\!<} 2 \mathbin{-\!\!\!<} 8$ is written $[1, 2, 8]$. The data type *snoc-list* is defined as an initial algebra in the category of algebras with two operators $\phi : B \times A \to B$ and $\psi : B$. By definition, an initial algebra satisfies the property that each homomorphism from an initial algebra to another algebra is unique. Such a unique homomorphism satisfies nice calculational properties. In the case of *snoc-list*, we have that, given an operator $\oplus : B \times A \to B$ and a value $e : B$, there exists a unique function $h : A\star \to B$ that satisfies

$$\begin{aligned} h\,\square &= e \\ h\,(x \mathbin{-\!\!\!<} a) &= (h\,x) \oplus a \;. \end{aligned}$$

Function h is called a *left-reduction*, and it is denoted by $\oplus\!\not\rightarrow\! e$. If operator $\oplus : A \times A \to A$ with unit e is associative we write $\oplus/$ for $\oplus\!\not\rightarrow\! e$, so $+/\,[1, 2, 8] = 11$, and we call $\oplus/$ a reduction. We give some examples of left-reductions.

- Given a function $f : A \rightarrow B$, the *map* $f\star : A\star \rightarrow B\star$ takes a list and applies f to each element in the list, so $f\star[1,2,8] = [f\,1, f\,2, f\,8]$. We have

  $$f\star \quad = \quad (\mathbin{-\!\!<} \cdot id \times f)\not\rightarrow\Box \ ,$$

  where operator $\times$ is defined by $(f \times g)\,(a,b) = (f\ a, g\ b)$.

- Given a predicate $p : A \rightarrow bool$, the *filter* $p\triangleleft : A\star \rightarrow A\star$ takes a list and retains the elements that satisfy p in the list, so $odd\triangleleft[1,2,8] = [1]$. We have $p\triangleleft = \oplus\not\rightarrow\Box$, where operator $\oplus$ is defined by

  $$x \oplus a \quad = \quad \begin{array}{ll} x \mathbin{-\!\!<} a & \text{if } p\ a \\ x & \text{otherwise} \end{array} .$$

  The composition of functions $\otimes/ \cdot p\triangleleft$ equals the composition of functions $\otimes/ \cdot p?_{\otimes}\star$, where $p?_{\otimes}$ is defined by

  $$p?_{\otimes}\ x \quad = \quad \begin{array}{ll} x & \text{if } p\ x \\ \nu_{\otimes} & \text{otherwise} \end{array} ,$$

  where $\nu_{\otimes}$ is the unit of operator $\otimes$. Thus we can rewrite the composition of a reduction and a filter into the composition of a reduction and a map. Function $p?_{\otimes}$ itself is a left-reduction, the definition of which is omitted, provided predicate p satisfies $p\,\Box$ and $p\,(x \mathbin{-\!\!<} a) \ \Rightarrow\ p\ x$.

- The operator $\mathbin{+\!\!\!+\!\!<}$, which concatenates two lists, is defined in terms of a left-reduction by

  $$x \mathbin{+\!\!\!+\!\!<} y \quad = \quad (\mathbin{-\!\!<}\not\rightarrow x)\ y\ .$$

  Operator $\mathbin{+\!\!\!+\!\!<}$ is associative and has unit $\Box$.

- The function which returns the length of a list is denoted by # and defined by

  $$\# \quad = \quad ((+1) \cdot \ll)\not\rightarrow 0\ ,$$

  where $\ll$ is the left-projection, i.e., $a \ll b = a$, or, equivalently, $\ll (a,b) = a$.

Operator zip, denoted by $\Upsilon : (A\star \times A\star) \rightarrow (A \times A)\star$, takes two equal-length lists, and 'zips' these lists together. For example, $[1,2,3]\ \Upsilon\ [4,5,6] = [(1,4),(2,5),(3,6)]$.

A theorem that is often used in the derivation of algorithms is the so-called Fusion on *snoc-list* Theorem. This theorem describes the condition for 'fusing' the composition of a function and a left-reduction into a left-reduction.

(2) Theorem (Fusion on *snoc-list***)** *Let $\oplus : A \times C \rightarrow A$, $\otimes : B \times C \rightarrow B$. Suppose $f : A \rightarrow B$ satisfies for all a, and for all x in the image of $\oplus\not\rightarrow e$, $f\,(x \oplus a) = (f\ x) \otimes a$. Then*

$$f \cdot (\oplus\not\rightarrow e) \quad = \quad \otimes\not\rightarrow(f\ e)\ .$$

Function me_1 returns a singleton list containing the length of a snoc-list, that is $me_1\, x = [\#\, x]$, or, equivalently, $me_1 = \tau \cdot \#$, where $\tau = (\square \mathbin{+\!\!<})$. Using Fusion on *snoc-list*, observing that $[x{+}1] = (+1)\star [x]$, we obtain

$$me_1 \quad = \quad ((+1)\star \cdot \ll) \not\to [0]\ .$$

A two-dimensional array is modelled by a list of lists. Intuitively, only a list of equal-length lists is a proper two-dimensional array, but we forget about this restriction to simplify the development of theory. We discuss this restriction in more detail later. Hence, the elements of the data type *two-dimensional array*, denoted by $A\star_2$, are the elements of the data type $A\star\star$.

In general, the data type *n-dimensional array* ($n > 1$), denoted by $A\star_n$, is defined as the data type *snoc-list* of $(n{-}1)$-dimensional arrays.

(3) Definition *The data type* zero-dimensional array*, denoted by $A\star_0$, is any simple data type like* bool *and* nat*. The data type* one-dimensional array*, denoted by $A\star$, is the data type* snoc-list*. Let $A\star_{n-1}$ be the data type* $(n{-}1)$-dimensional array*. Then $A\star_n = A\star_{n-1}\star$ is the data type* n-dimensional array.

Left-reductions on n-dimensional arrays are defined as follows.

(4) Definition (Hierarchical Left-reduction) *The hierarchical left-reduction denoted by $(\oplus_n, \ldots, \oplus_1, f) \not\to_n (e_n, \ldots, e_1) : A\star_n \to B$ is defined as follows. If $n = 0$, then define $(\oplus_n, \ldots, \oplus_1, f) \not\to_n (e_n, \ldots, e_1) = f$, and for $n \geq 1$ define*

$$(\oplus_n, \ldots, \oplus_1, f) \not\to_n (e_n, \ldots, e_1) \quad = \quad \oplus_n \not\to e_n \cdot ((\oplus_{n-1}, \ldots, \oplus_1, f) \not\to_{n-1} (e_{n-1}, \ldots, e_1))\star\ .$$

The following theorem is one of the most important results for the derivation of hierarchies of algorithms on arrays: it characterises hierarchical left-reductions.

(5) Theorem (Characterisation of Hierarchical Left-reductions)

$$f_n \quad = \quad (\otimes_n, \ldots, \otimes_1, f_0) \not\to_n (e_n, \ldots, e_1)$$

if and only if for all m with $n \geq m \geq 1$,

$$\begin{array}{lcl} f_m\, \square & = & e_m \\ f_m\, (x \mathbin{+\!\!<} a) & = & (f_m\, x) \otimes_m (f_{m-1}\, a)\ , \end{array}$$

for some family of values e_m and some family of operators $\otimes_m$.

Function me_2 returns, when applied to a proper two-dimensional array (a proper array is an array in the usual meaning of the word, and hence a proper two-dimensional array is a list of equal-length lists), a list consisting of two elements: the height and the width of the array. When applied to a non-proper two-dimensional array it returns some arbitrary value. Function me_2 is a component of the family of functions $me_n : A{\star}_n \rightarrow nat{\star}$, the components of which return the measure of a proper n-dimensional array as a list and some arbitrary value when applied to a non-proper array. The family of functions me_n is an example of a hierarchical left-reduction. Define function me_0 by $me_0\ a = \square$. The family of functions me_n is characterised by the equalities

$$\begin{aligned} me_m\,\square &= [0] \\ me_m\,(x \mathbin{+\!\!\!<} a) &= (me_{m-1}\ a) \mathbin{+\!\!\!<} ((lt\ me_m\ x) + 1)\ , \end{aligned}$$

for all m with $n \geq m \geq 1$, where function lt returns the last element of a list. Hence, if we define for m with $n \geq m \geq 1$, $e_m = [0]$, and

$$(6) \qquad x \otimes_m a \ = \ a \mathbin{+\!\!\!<} ((lt\ x) + 1)\ ,$$

then, applying Characterisation of Hierarchical Left-reductions, me_n is defined by

$$(7) \qquad me_n \ = \ (\otimes_n, \ldots, \otimes_1, \square^{\bullet}) \not\rightarrow_n (e_n, \ldots, e_1)\ ,$$

where function $a^{\bullet}$ is defined by $a^{\bullet}\ b = a$ for all b.

For the definition of the family of functions $propar_n : A{\star}_n \rightarrow bool$ (for 'proper array') we have to develop some more theory. The mth component of the family of functions $propar_n$ determines, when applied to an element of $A{\star}_m$, whether or not the element is a proper m-dimensional array. An example of a component of this family of functions is the function $propar_2$, which, given a lists of lists, determines whether or not it is a list of equal-length lists. The family of functions $propar_n$ is defined by means of the family of functions me_n. The tuple of functions $propar_n \vartriangle me_n$ (function $f \vartriangle g$ is defined by $(f \vartriangle g)\ a = (f\ a, g\ a)$) is a hierarchical left-reduction. To prove this fact, we use the following notions and theorem.

We say that the family of functions f_n is catamorphic modulo the family of functions g_n if there exists a family of operators $\oplus_n$ such that for all m with $n \geq m \geq 1$

$$f_m\,(x \mathbin{+\!\!\!<} a) \ = \ (f_m\ x, g_m\ x) \oplus_m (f_{m-1}\ a, g_{m-1}\ a)\ .$$

If the family of functions g_n is catamorphic modulo the family of functions f_n too, we call the families of functions f_n and g_n mutumorphisms. For mutumorphisms f_n and g_n it is easy to prove the following theorem.

(8) Theorem (Hierarchical Mutumorphisms) *Suppose the families of functions f_n and g_n are catamorphic modulo each other, that is, there exist families of operators $\oplus_n$ and $\ominus_n$ such that for all m with $n \geq m \geq 1$*

$$\begin{aligned} f_m\,(x \mathbin{+\!\!\!<} a) &= (f_m\ x, g_m\ x) \oplus_m (f_{m-1}\ a, g_{m-1}\ a) \\ g_m\,(x \mathbin{+\!\!\!<} a) &= (f_m\ x, g_m\ x) \ominus_m (f_{m-1}\ a, g_{m-1}\ a)\ . \end{aligned}$$

Then $f_n \vartriangle g_n$ is a hierarchical left-reduction:

$$f_n \vartriangle g_n \quad = \quad (\odot_n, \ldots, \odot_1, f_0 \vartriangle g_0) \not\rightarrow_n (e_n, \ldots, e_1) \ ,$$

where for all m with $n \geq m \geq 1$, $e_m = (f_m \,\Box, g_m \,\Box)$, and operator $\odot_m$ is defined by

$$(x, y) \odot_m (a, b) \quad = \quad ((x, y) \oplus_m (a, b), (x, y) \ominus_m (a, b)) \ .$$

The families of functions $propar_n$ and me_n are mutumorphisms. For all m with $n \geq m \geq 1$ we have

$$me_m\,(x \twoheadleftarrow a) \quad = \quad (me_m\,x, propar_m\,x) \oplus_m (me_{m-1}\,a, propar_{m-1}\,a) \ ,$$

where operator $\oplus_m$ is defined by

$$(x, y) \oplus_m (a, b) \quad = \quad x \otimes_m a \ ,$$

and

$$propar_m\,(x \twoheadleftarrow a) \quad = \quad (me_m\,x, propar_m\,x) \ominus_m (me_{m-1}\,a, propar_{m-1}\,a) \ ,$$

where operator $\ominus_m$ is defined by

$$(x, y) \ominus_m (a, b) \quad = \quad \begin{array}{ll} false & \text{if } y = false \ \vee \ b = false \ \vee \ it\,x \neq a \\ true & \text{otherwise} \end{array} \ ,$$

where function it returns all but the last elements of a nonempty list, so $it\,(x \twoheadleftarrow a) = x$. (Function tl returns all but the first elements of a list, so $tl\,([a] \mathbin{+\!\!\!+\!\!\!<} x) = x$, and function hd returns the first element of a list, so $hd\,([a] \mathbin{+\!\!\!+\!\!\!<} x) = a$.) Applying Hierarchical Mutumorphisms we obtain

$$me_n \vartriangle propar_n \quad = \quad (\odot_n, \ldots, \odot_1, \Box^\bullet \vartriangle true^\bullet) \not\rightarrow_n (e_n, \ldots, e_1) \ ,$$

where operator $\odot$ is defined by

$$(x, y) \odot_m (a, b) \quad = \quad ((x, y) \oplus_m (a, b), (x, y) \ominus_m (a, b)) \ .$$

Another example of a hierarchical left-reduction is the function *size*, denoted by $\#_n$, which returns the number of elements in an n-dimensional array.

(9) $\quad \#_n \quad = \quad (+, \ldots, +, 1^\bullet) \not\rightarrow_n (0, \ldots, 0) \ .$

One would expect that the following equality holds on the domain of proper arrays. Let $\times$ be multiplication of natural numbers.

(10) $\quad \#_n \quad = \quad \times / \cdot me_n \ .$

This equality can be proved with the following theorem, called Array Fusion, which lists the conditions that have to be satisfied in order to fuse the composition of a family of functions with a hierarchical left-reduction into a hierarchical left-reduction. It is often applied in calculations, see Jeuring [8]. Its proof is an application of Characterisation of Hierarchical Left-reductions, Theorem 5.

(11) Theorem (Array Fusion) *Suppose that the family of functions f_n and the families of operators $\otimes_n$ and $\oplus_n$ satisfy for m with $n \geq m \geq 1$,*

$$f_m\,(x \otimes_m a) \;=\; (f_m\,x) \oplus_m (f_{m-1}\,a)\;,$$

for all x in the image of $(\otimes_m,\ldots,\otimes_1,g_0)\not\rightarrow_n(e_m,\ldots,e_1)$, and for all a in the image of $(\otimes_{m-1},\ldots,\otimes_1,g_0)\not\rightarrow_n(e_{m-1},\ldots,e_1)$. Then

$$f_n \cdot (\otimes_n,\ldots,\otimes_1,g_0)\not\rightarrow_n(e_n,\ldots,e_1) \;=\; (\oplus_n,\ldots,\oplus_1,f_0\cdot g_0)\not\rightarrow_n(f_n\,e_n,\ldots,f_1\,e_1)\;.$$

Since for all m with $n \geq m \geq 1$ we have

$$
\begin{array}{ll}
 & \times/\,(x \otimes_m a) \\
= & \quad \text{definition of } \otimes_m \text{ (6)} \\
 & \times/\,(a \mathbin{+\!\!\!<} ((\mathit{lt}\,x)+1)) \\
= & \quad \text{definition of reduction} \\
 & (\times/\,a) \times ((\mathit{lt}\,x)+1) \\
= & \quad \text{definition of } \times \\
 & ((\times/\,a) \times (\mathit{lt}\,x)) + (\times/\,a) \\
= & \quad a = \mathit{it}\,x \text{ on the domain of proper arrays} \\
 & ((\times/\,\mathit{it}\,x) \times (\mathit{lt}\,x)) + (\times/\,a) \\
= & \quad \text{definition of reduction} \\
 & (\times/\,x) + (\times/\,a)\;,
\end{array}
$$

and since $\times/\cdot\square^\bullet = 1^\bullet$ because 1 is the unit of $\times$, apply Array Fusion to obtain equality (10).

3 The specification

In this section we give a specification as a family of functions for pattern matching on arrays.

A subarray of a two-dimensional array is a contiguous rectangle within the array. A subarray of an n-dimensional array is the straightforward extension of the notion of subarray on *two-dimensional array* to the data type *n-dimensional array*. The family of functions suba_n returns all subarrays of an n-dimensional array as a snoc-list. Given two arrays, the operator $\uparrow_{\#_n}$: $A\star_n \times A\star_n \to A\star_n$ returns an array with maximal size. Given an N-dimensional array P, the pattern-matching problem pm_N requires finding an occurrence of P in an N-dimensional array, or, if there are no such occurrences, the longest prefix of P (function *inits* returns all prefixes of an array as a snoc-list, including the array itself) occurring in the given array. Recall that array P is a list of $(N-1)$-dimensional arrays, so function *inits* may be applied to P.

(12) $pm_N \;=\; \uparrow_{\#_N}/ \cdot (\in inits\ P)\triangleleft \cdot suba_N$.

The function pm_1 is the specification of pattern matching on *snoc-list*. Bird et al. [6] start with this specification in their derivation of the pattern-matching algorithm of Knuth, Morris and Pratt [12]. For the purpose of applying the theory developed in the previous subsections, we want to specify the pattern-matching problem as a family of functions instead of a function defined just on N-dimensional arrays, that is, we want the functions $\uparrow_{\#_N}/$, $(\in inits\ P)$, and $suba_N$ to be dimension-dependent. Replace $\uparrow_{\#_N}/$ by $\uparrow_{\#_n}/$ and $suba_N$ by $suba_n$. This leaves function $(\in inits\ P)$ to be replaced by a dimension-dependent equivalent. Define the family of functions $list_n$ by

$$\begin{array}{lrcl} & list_n & : & A\star_N\star \rightarrow A\star_{N-n}\star \\ (13) & list_0 & = & id \\ & list_n & = & +\!\!\!\!\!+\!\!\!\!\!\!<\!/ \cdot list_{n-1}\ , \end{array}$$

that is, $list_n = (+\!\!\!\!\!+\!\!\!\!\!\!<, \ldots, +\!\!\!\!\!+\!\!\!\!\!\!<, id)\not\to_n(\square, \ldots, \square)$. Note that $list_{N-n} : A\star_N\star \rightarrow A\star_n\star$. The family of functions pm_n is specified by

(14) $pm_n \;=\; \uparrow_{\#_n}/ \cdot (\in +\!\!\!\!\!+\!\!\!\!\!\!</\ inits\star\ list_{N-n}\,[P])\triangleleft \cdot suba_n$.

Function *inits* is defined on the data type *n-dimensional array* for $n \geq 1$; on the data type *zero-dimensional array* we assume function *inits* to be the identity function. Function $+\!\!\!\!\!+\!\!\!\!\!\!</$ is not defined on the data type $A\star$ if A is not a data type of the form $B\star$ for some B, and in this case we extend its definition by defining it to be the identity function on *snoc-list*. Consider the following example. Let $P = [[2,3],[5,4]]$ be a two-dimensional array pattern. Then the family of functions pm_n consists of three components:

$$\begin{array}{lcl} pm_2 & = & \uparrow_{\#_2}/ \cdot (\in [\square, [[2,3]], [[2,3],[5,4]]])\triangleleft \cdot suba_2 \\ pm_1 & = & \uparrow_{\#_1}/ \cdot (\in [\square, [2], [2,3], \square, [5], [5,4]])\triangleleft \cdot suba_1 \\ pm_0 & = & \uparrow_{\#_0}/ \cdot (\in [2,3,4,5])\triangleleft \cdot suba_0\ . \end{array}$$

Function pm_1 is a specification of the problem of pattern matching with probably more than one pattern. Abbreviate the predicate $\in +\!\!\!\!\!+\!\!\!\!\!\!</\ inits\star\ list_{N-n}\,[P]$ to q_n. Note that $q_N = (\in inits P)$, and that $q_n(x +\!\!\!\!\!< a) \;\Rightarrow\; q_n\ x \;\wedge\; q_{n-1}\ a$. Abbreviate the list $+\!\!\!\!\!+\!\!\!\!\!\!</\ inits\star\ list_{N-n}\,[P]$ to Q_n. A function $\delta_x\ a$ satisfying $q_n(x +\!\!\!\!\!< a) = q_n\ x \;\wedge\; \delta_x\ a$ is defined by

(15) $\delta_x\ a \;=\; a \in (hd \cdot (\#\, x \hookrightarrow))\star(x \in inits)\triangleleft\, Q_n$.

Consider the function pm_2. Let the two-dimensional pattern P have height h. Then the elements of the set $inits\,P$ all have height h, and each element of $suba_2$ with height not equal to h does not match with any of the elements of $inits\ P$. Therefore, all elements of $suba_2$ with height not equal to h can immediately be discarded. This argument can be repeated for the higher-dimensional functions pm_n, with height replaced by me_{n-1}-value. We define a family of functions $subm_n$ the elements of which enumerate only subarrays of a given measure, and we replace $suba_n$ by $subm_n$ in the specification of the pattern-matching problem.

Let the family of functions $subm_n$ enumerate all subarrays of its argument with me_{n-1}-value equal to me_{n-1}-value of the pattern. For example, if the pattern is a two-dimensional array of height 4, then $subm_2$ enumerates all subarrays of height 4. The family of functions $subm_n$ can be defined elegantly using an auxiliary family of functions $taim_n$ which enumerates the 'tails' with me_{n-1}-value equal to me_{n-1}-value of the pattern. Consider for example two-dimensional arrays. The two-dimensional subarrays of height 4 of a two-dimensional array $x \mathbin{+\!\!<} a$ consist of the two-dimensional subarrays of height 4 of x, together with all subarrays of height 4 of $x \mathbin{+\!\!<} a$ containing four contiguous elements from the one-dimensional array a. All contiguous elements of height 4 from a are obtained by applying $subm_1$ to a. All subarrays of height 4 of $x \mathbin{+\!\!<} a$ containing a contiguous part of a are obtained by appending the elements of $subm_1\, a$ to the 'corresponding' (of the same height and occurring at the same position) tails of x. The tails of x are obtained by means of the function $taim_2$. Informally, we define a subarray y of an array x to end in x if y, when drawn in two dimensions, occurs at the right end of x. The function $taim_2$ applied to an array x returns a list of lists, in which each list consists of subarrays of equal height (4 in the example) of x, ending at the same position in x. Any two arrays occurring in one of the lists of $taim_2$ are of equal height, but of different breadth. The lists in the list of lists are enumerated in the same order of height as the subarrays in $subm_1$. Let P be an N-dimensional array with measure p, with $(\neq 0) \triangleleft p = p$. Define the family of functions $stam_n$ by

$$(16) \quad stam_n \;=\; subm_n \vartriangle taim_n$$

Apply Hierarchical Mutumorphisms to obtain a hierarchical left-reduction for $stam_n$. Define $stam_0$ by $\tau \vartriangle (\tau \cdot \tau)$. Characterise the family of functions $subm_n$ by

$$\begin{array}{lll} & subm_n & : \; A\star_n \rightarrow A\star_n\star \\ (17) & subm_n\, \square & = \; \square \\ & subm_n\,(x \mathbin{+\!\!<} a) & = \; (subm_n\, x) \mathbin{+\!\!+\!\!<} (\mathbin{+\!\!+\!\!<}/\; taim_n\,(x \mathbin{+\!\!<} a))\;, \end{array}$$

and family of functions $taim_n$ by

$$\begin{array}{lll} & taim_n & : \; A\star_n \rightarrow A\star_n\star\star \\ (18) & taim_n\, \square & = \; \square \\ & taim_n\,(x \mathbin{+\!\!<} a) & = \; (taim_n\, x) \oslash (((= (n-1) \rightharpoonup p) \cdot me_{n-1}) \triangleleft subm_{n-1}\, a)\;, \end{array}$$

where function $n \rightharpoonup$ takes the first n elements of a list, and is defined by

$$\begin{array}{lll} 0 \rightharpoonup x & = & \square \\ (n+1) \rightharpoonup x & = & \begin{array}{ll} [hd\, x] \mathbin{+\!\!+\!\!<} (n \rightharpoonup (tl\, x)) & \text{if } x \neq \square \\ \square & \text{otherwise}\;, \end{array} \end{array}$$

and operator $\oslash$ is defined by

$$(19)\quad x \oslash a \;=\; \begin{array}{ll} (\tau\cdot\tau)\star a & \text{if } x = \Box \\ x\,\Upsilon_{\odot}\,a & \text{otherwise} \end{array} ,$$

where operator $\odot$ is defined by

$$(20)\quad x \odot a \;=\; (\mathbin{-\!\!\!<} a)\star x \mathbin{+\!\!\!+\!\!\!<} [[a]]\;.$$

Abbreviate $((= n \rightharpoonup p)\cdot me_n)$ to p_n, and note that $p_0 = true^{\bullet}$. From these equations it follows that the families of functions $subm_n$ and $taim_n$ are mutumorphisms. Applying Hierarchical Mutumorphisms we obtain

$$(21)\quad stam_n \;=\; (\otimes_n,\ldots,\otimes_1,\tau \vartriangle (\tau\cdot\tau))\not\rightarrow\!\!\!\!/_{n}(e_n,\ldots,e_1)\;,$$

where the family of values e_n is defined by $e_m = (\Box,\Box)$, and the family of operators $\otimes_n$ is defined by

$$\begin{array}{llcl} & \otimes_m & : & (A\star_m\star \times A\star_m\star\star)\times(A\star_{m-1}\star\times A\star_{m-1}\star\star)\to A\star_m\star\times A\star_m\star\star \\ (22) & (x,y)\otimes_m(a,b) & = & (x \mathbin{+\!\!\!+\!\!\!<} (\mathbin{+\!\!\!+\!\!\!<}/\,z), z) \\ & & & \textbf{where } z = y \oslash (p_{m-1}\triangleleft\, a)\;, \end{array}$$

for all m with $n \geq m \geq 1$. Specification (14) is transformed into

$$(23)\quad pm_n \;=\; \uparrow_{\#_n}/\cdot\, q_n\triangleleft\cdot subm_n\;.$$

4 The derivation

This section sketches the derivation of a hierarchy of efficient algorithms for pattern-matching on arrays.

The derivation of a hierarchical left-reduction for the pattern-matching problem specified in (23) is structured as follows. First, we rewrite the specification to a form to which Array Fusion can be applied. Then we try to apply Array Fusion, but the condition of this theorem cannot be satisfied. However, the derivation suggests to tuple with an extra function. Applying Hierarchical Mutumorphisms we show that the resulting tuple of functions is a hierarchical left-reduction that can be implemented as an efficient program.

Since $subm_n = \ll\cdot\, stam_n$, we derive for arbitrary family of functions g_n:

$$\begin{array}{ll} & pm_n \\ = & \quad\text{definition of } pm_n \\ & \uparrow_{\#_n}/\cdot\, q_n\triangleleft\cdot subm_n \\ = & \quad\text{definition of } stam_n \\ & \uparrow_{\#_n}/\cdot\, q_n\triangleleft\cdot \ll\cdot\, stam_n \\ = & \quad\text{law for } \times \\ & \ll\cdot\,(\uparrow_{\#_n}/\cdot\, q_n\triangleleft)\times g_n\cdot stam_n\;. \end{array}$$

Abbreviate the product of functions $(\uparrow_{\#_n}/\cdot q_n\triangleleft)\times g_n$ to j_n. Array Fusion is now applied to the expression $j_n\cdot stam_n$. We obtain

$$j_n\cdot stam_n \quad = \quad (\ominus_n,\ldots,\ominus_1,j_0\cdot stam_0)\not\to_n(e_n,\ldots,e_1)\ ,$$

provided

$$\begin{array}{lll} (24)\ \ e_m & = & j_m\,(\square,\square) \\ (25)\ \ j_m\,((x,y)\otimes_m(a,b)) & = & (j_m\,(x,y))\ominus_m(j_{m-1}\,(a,b))\ , \end{array}$$

for all m with $n\geq m\geq 1$. In the synthesis of a family of operators $\ominus_n$ satisfying the above equation a suitable definition of a family of functions g_n appears. An operator $\ominus_m$ satisfying condition (25) is synthesised as follows.

$$\begin{array}{ll}
 & j_m\,((x,y)\otimes_m(a,b)) \\
= & \quad \text{definition of } j_m \text{ and } \otimes_m,\ z=y\oslash(p_{m-1}\triangleleft a) \\
 & ((\uparrow_{\#_m}/\cdot q_m\triangleleft)\times g_m)\,(x+\!\!\!+\!\!\!<(+\!\!\!+\!\!\!</\,z),z) \\
= & \quad \text{definition of } \times \\
 & (\uparrow_{\#_m}/\ q_m\triangleleft(x+\!\!\!+\!\!\!<(+\!\!\!+\!\!\!</\,z)),g_m\ z) \\
= & \quad \text{filter and reduction on } \textit{snoc-list} \\
 & ((\uparrow_{\#_m}/\ q_m\triangleleft x)+\!\!\!+\!\!\!<(\uparrow_{\#_m}/\ q_m\triangleleft+\!\!\!+\!\!\!</\,z),g_m\ z) \\
= & \quad \text{Fusion on } \textit{snoc-list} \\
 & ((\uparrow_{\#_m}/\ q_m\triangleleft x)+\!\!\!+\!\!\!<(\uparrow_{\#_m}/\,(\uparrow_{\#_m}/\cdot q_m\triangleleft)\star\ z),g_m\ z)\ .
\end{array}$$

In this last expression we distinguish three subexpressions:

$$\begin{array}{l} \uparrow_{\#_m}/\ q_m\triangleleft x \\ (\uparrow_{\#_m}/\cdot q_m\triangleleft)\star\ z \\ g_m\ z\ , \end{array}$$

where $z=y\oslash(p_{m-1}\triangleleft a)$. In view of the form of the desired expression, equation (25), the first subexpression $\uparrow_{\#_m}/\ q_m\triangleleft x$, which equals $\ll j_m\,(x,y)$, need not be developed any further. The subexpressions $(\uparrow_{\#_m}/\cdot q_m\triangleleft)\star z$ and $g_m\ z$, where $z=y\oslash(p_{m-1}\triangleleft a)$, have to be expressed in terms of $g_m\ y$ and $\uparrow_{\#_{m-1}}/\ q_{m-1}\triangleleft a$. Hence a reasonable choice for g_m seems to be

$$g_m \quad = \quad (\uparrow_{\#_m}/\cdot q_m\triangleleft)\star\ .$$

We proceed the synthesis of an operator $\ominus_m$ satisfying (25) with distinguishing the two cases in the definition of operator $\oslash$, which occurs in the expression $(\uparrow_{\#_m}/\cdot q_m\triangleleft)\star(y\oslash(p_{m-1}\triangleleft a))$. In this omitted derivation we use the fact that the family of predicates q_n satisfies for all m with $n\geq m\geq 1$, $q_m\,(x+\!\!<a)\ \Rightarrow\ q_m\,x\ \wedge\ q_{m-1}\,a$. For a family of predicates satisfying this implication there exists a hierarchical left-reduction such that

$$q_n?_{\uparrow_{\#_n}} \quad = \quad (\oplus_n,\ldots,\oplus_1,q_0?_{\uparrow_{\#_0}})\not\to_n(\square,\ldots,\square)\ ,$$

where operator $\oplus_m$ is defined by

$$x \oplus_m a \quad = \quad \begin{array}{ll} x \mathbin{+\!\!\!<} a & \text{if } x \neq \omega_m \;\wedge\; a \neq \omega_{m-1} \;\wedge\; q_m\,(x \mathbin{+\!\!\!<} a) \\ \omega_m & \text{otherwise} \;, \end{array}$$

where ω_m is the unit of operator $\uparrow_{\#_m}$, for all m with $n \geq m \geq 1$. The case distinction in the definition of operator $\oslash$ gives the following equation.

$$\begin{array}{rcl} j_m\,((x,y) \otimes_m (a,b)) & = & (\ll j_m\,(x,y) \mathbin{+\!\!\!+\!\!\!<} \uparrow_{\#_m}/\; s, s) \\ & & \quad \textbf{where } s = \begin{array}{ll} (\Box\oplus_m)\star\, k_{m-1}\, a & \text{if } y = \Box \\ y \;\Upsilon_{\Phi_m}\; k_{m-1}\, a & \text{otherwise} \;, \end{array} \end{array}$$

where the the family of operators Φ_n is given in the final description of the algorithm, and the family of functions k_n is defined by

$$k_m \quad = \quad q_m ?_{\uparrow_{\#_m}} \star \cdot\, p_m \triangleleft \cdot\, subm_m \;.$$

It follows that we cannot apply Array Fusion to $j_m \cdot stam_m$, but also that $j_m \cdot stam_m$ is catamorphic modulo k_m. We have

$$j_m\, stam_m\,(x \mathbin{+\!\!\!<} a) \quad = \quad (j_m\, stam_m\, x, k_m\, x) \ominus_m (j_{m-1}\, stam_{m-1}\, a, k_{m-1}\, a) \;,$$

where the family of operators $\ominus_n$ is defined by

$$(26) \quad \begin{array}{rcl} ((x,y),z) \ominus_m ((a,b),c) & = & (x \mathbin{+\!\!\!+\!\!\!<} \uparrow_{\#_m}/\; s, s) \\ & & \quad \textbf{where } s = \begin{array}{ll} (\Box\oplus_m)\star\, c & \text{if } y = \Box \\ y \;\Upsilon_{\Phi_m}\; c & \text{otherwise} \;, \end{array} \end{array}$$

for all m with $n \geq m \geq 1$.

If we can also show that k_m is catamorphic modulo $j_m \cdot stam_m$, that is, if we can exhibit a family of operators $\circleddash_n$ satisfying

$$(27) \quad k_m\,(x \mathbin{+\!\!\!<} a) \quad = \quad (j_m\, stam_m\, x, k_m\, x) \circleddash_m (j_{m-1}\, stam_{m-1}\, a, k_{m-1}\, a) \;,$$

for all m with $n \geq m \geq 1$, then we can apply the Hierarchical Mutumorphisms Theorem to show that

$$(j_m \cdot stam_m) \vartriangle k_m \quad = \quad (\odot_n, \ldots, \odot_1) \nrightarrow_n (e_n, \ldots, e_1)$$

where the family of operators $\odot_n$ is defined by

$$(x,y) \odot_m (a,b) \quad = \quad ((x,y) \ominus_m (a,b), (x,y) \circleddash_m (a,b)) \;,$$

and the family of values e_n is defined by

$$e_m \quad = \quad (j_m\,(\Box,\Box), k_m \Box) \;,$$

for all m with $n \geq m \geq 1$. A family of operators $\obslash_n$ satisfying equation (27) is defined by

$$(28)\quad ((x,y),z) \obslash_m ((a,b),c) \;=\; z \mathbin{+\!\!\!\!\prec} \mathbin{+\!\!\!\!\prec}/\, t$$

$$\textbf{where}\quad t = \begin{cases} (\tau \cdot (\Box \oplus_m)) \star c & \text{if } y = \Box \ \wedge\ lp_m = 1 \\ \Box^{\bullet} \star c & \text{if } y \neq \Box \ \wedge\ lp_m \neq 1 \\ y \ \Upsilon_{\circeq_m}\ c & \text{otherwise} \end{cases} ,$$

where $lp_m = lt\,(m \rightharpoonup p)$, for all m with $n \geq m \geq 1$. The definition of the family of operators $\circeq_n$ is given in the final description of the algorithm.

The final result of the derivation of a hierarchy of efficient algorithms for pattern matching looks as follows. This hierarchy of algorithms is given merely for completeness' sake. After the description of the hierarchy of algorithms we give a final optimisation of it. The first two components of the following hierarchical left-reduction correspond to the function $j_m \cdot stam_m$, the third component to the function k_m, and the fourth component to the function #, which is also needed to construct a hierarchical left-reduction that can be implemented as an efficient algorithm.

We have

$$(29)\quad pm_n \;=\; \gg \cdot \ll \cdot (\ominus_n, \ldots, \ominus_1, h) \not\rightarrow_n (e_n, \ldots, e_1) ,$$

where h is defined by $(q_0?_{\upharpoonright\#_0} \vartriangle (\tau \cdot q_0?_{\upharpoonright\#_0})) \vartriangle (\tau \cdot q_0?_{\upharpoonright\#_0}) \vartriangle 1^{\bullet}$, e_m is defined by $((\omega_m, \Box), \Box, 0)$, and operator $\ominus_m$ is defined by

$$(x,y,z) \ominus_m (a,b,c) \;=\; ((x,y) \oslash (a,b), (x,y) \obslash (a,b), z{+}1)$$

where the families of operators $\oslash_n$ and $\obslash_n$ are defined respectively in (26) and (28). The families of operators $\obar_n$ and $\circeq_n$ are defined by

$$s \obar_m c \;=\; (s \,\Phi_m\, c) \upharpoonright_{\#_m} (\Box \oplus_m c) ,$$

where

$$s \,\Phi_m\, c \;=\; \begin{cases} s \mathbin{+\!\!\!\!\prec} c & \text{if } c \neq \omega_{m-1} \ \wedge\ \delta_s\, c \\ (lpt_m\ tl\ s) \,\Phi_m\, c & \text{if } c \neq \omega_{m-1} \ \wedge\ \neg\delta_s\, c \ \wedge\ s \neq \Box \\ \omega_m & \text{otherwise} \end{cases} ,$$

where lpt_m is defined by

$$lpt_m \;=\; \upharpoonright_{\#}/ \cdot q_m \triangleleft \cdot tails ,$$

where function *tails* returns the tail segments of a list in descending order of length, for example $tails\,[1,2,3] = [[1,2,3],[2,3],[3],\Box]$, and operator $\circeq_m$ is defined by $s \circeq_m c = \Box$ if $z < lp_m - 1$, and if $z \geq lp_m - 1$, then

$$s \circeq_m c \;=\; \begin{cases} [\omega_m] & \text{if } \# s < lp_m - 1 \\ [s \oplus_m c] & \text{if } \# s = lp_m - 1 \\ (lpt_m\ tl\ s) \circeq_m c & \text{if } \# s = lp_m \end{cases} ,$$

for all m with $n \geq m \geq 1$.

The two remaining problems that have to be addressed are the computation of $\delta_s\, c$ and the computation of $lpt\ tl\, s$.

Concerning the computation of these values we have the following. Function δ takes as arguments an m-dimensional array s and an $(m-1)$-dimensional array c, and checks whether $s \twoheadleftarrow c$ is an element of Q_m, that is, whether $q_m\,(s \twoheadleftarrow c)$ holds. Matching an m-dimensional array is expensive and superfluous, since it can be avoided. We explain the situation for two-dimensional arrays. Consider the set of two-dimensional array patterns Q_2 defined by $\# / \; inits * \; list_{N-2}\,\{P\}$. The two-dimensional pattern-matching algorithm with the set of patterns Q_2 uses the one-dimensional pattern-matching algorithm with the set of patterns Q_1. The columns of the two-dimensional arrays in Q_2 are one-dimensional arrays in Q_1. Assign a unique identifier, a natural number say, to each element in Q_1. Let the one-dimensional pattern-matching algorithm return the unique identifier instead of the one-dimensional array in case of a match, and replace the columns in the set of patterns Q_2 by the unique identifiers of the columns. Thus the two-dimensional pattern-matching problem has been reduced to a one-dimensional pattern-matching algorithm. This idea is easily generalised to n-dimensional arrays. We do not give the definitions of the functions involved in working out this idea. We have reduced the problem of computing $\delta_s\, c$ for an m-dimensional aray s and an $(m-1)$-dimensional array c to computing $\delta_s\, c$ for a one-dimensional array s and an element c. The efficient computation of this value, and of the value $lpt\ tl\, s$, in the context of pattern-matching on lists with probably more than one pattern, is discussed in my forthcoming thesis [10].

The hierarchical left-reduction (29) we have derived can be implemented in some language. The straightforward implementation is essentially a hierarchical version of the pattern-matching algorithm of Aho and Corasick [1]. It requires time $O(\#_n\, P + \#_n S)$ on an array S. The n-dimensional component of the hierarchical left-reduction applies the $(n-1)$-dimensional component of the hierarchical left-reduction to the $(n-1)$-dimensional arrays in the n-dimensional array, and applies the pattern-matching algorithm of Aho and Corasick to the resulting list.

References

[1] A.V. Aho and M.J. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

[2] T.P. Baker. A technique for extending rapid exact-match string matching to arrays of more than one dimension. *SIAM J. Computing*, 7(4):533–541, 1978.

[3] R.S. Bird. Two dimensional pattern matching. *Information Processing Letters*, 6(5):168–170, 1977.

[4] R.S. Bird. An introduction to the theory of lists. In M. Broy, editor, *Logic of Programming and Calculi of Discrete Design*, volume F36 of *NATO ASI Series*, pages 5–42. Springer–Verlag, 1987.

[5] R.S. Bird. Lectures on constructive functional programming. In M. Broy, editor, *Constructive Methods in Computing Science*, volume F55 of *NATO ASI Series*, pages 151–216. Springer–Verlag, 1989.

[6] R.S. Bird, J. Gibbons, and G. Jones. Formal derivation of a pattern matching algorithm. *Science of Computer Programming*, 12:93–104, 1989.

[7] M. Crochemore and W. Rytter. Parallel computations on strings and arrays. In C. Choffrut and T. Lengauer, editors, *7th Annual Symposium on Theoretical Aspects of Computer Science*, LNCS 415, pages 109–125, 1990.

[8] J. Jeuring. The derivation of hierarchies of algorithms on matrices. In B. Möller, editor, *Constructing Programs from Specifications*, pages 9–32. North-Holland, 1991.

[9] J. Jeuring. The derivation of on-line algorithms, with an application to finding palindromes. To appear in Algorithmica, 1992.

[10] J. Jeuring. *Theories for Algorithm Calculation*. PhD thesis, Utrecht University, 1993. To appear.

[11] R. Karp, R.E. Miller, and A. Rosenberg. Rapid identification of repeated patterns in strings, trees and arrays. In *Proceedings 4^{th} Annual ACM Symposium on Theory of Computing*, pages 125–136, 1972.

[12] D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6:323–350, 1978.

[13] G. Malcolm. Data structures and program transformation. *Science of Computer Programming*, 14:255–279, 1990.

[14] L. Meertens. Algorithmics—towards programming as a mathematical activity. In J.W. de Bakker, M. Hazewinkel, and J.K. Lenstra, editors, *Proceedings of the CWI Symposium on Mathematics and Computer Science*, volume 1 of *CWI Monographs*, pages 289–334. North–Holland, 1986.

[15] L. Meertens. Paramorphisms. Technical Report CS-R9005, CWI, 1990. To appear in Formal Aspects of Computing.

[16] L.M.R. Mullin. *A Mathematics of Arrays*. PhD thesis, Syracuse University, Syracuse, New York, 1988.