LINEARISATION: AN OPTIMISATION FOR NONLINEAR FUNCTIONAL PROGRAMS

Peter G. HARRISON

Department of Computing, Imperial College of Science and Technology, University of London, London SW7 2BZ, United Kingdom

Communicated by J. Darlington
Received April 1986
Revised May 1987

Abstract. Functional programming languages have great appeal from the point of view of both software design and amenability to formal reasoning, but to date they have suffered from poor performance when run on conventional computers. A promising solution to this problem may be provided by program transformation and several schemes have been proposed which can give quite impressive optimisations. However, these are at best only semi-automatic, requiring reasoning on behalf of the programmer to assist the transformation process. Part of the problem is that these schemes must take into account not only functions but also the objects to which they are applied in the defining expressions. By reasoning at the function level, the auxiliary domain of objects need not be considered explicitly, and transformations can be derived in terms of identities between functional expressions, rather than via sets of equations satisfied by objects from a certain class.

By expressing functional expressions in variable-free form, we use algebraic methods, based on the functional algebra of the language FP, to transform a certain class of nonlinear functions into linear form. A function in this class generates a reduction graph in the form of a balanced tree when applied to an argument, whereas a linear function generates a single-spine tree and so executes with a number of function calls which is linear in the size of its argument. Thus, for example, tail recursive functions form a small subset of the class of linear functions. Further optimisations include the tupling of functions which are defined by mutual recursion, and we identify conditions under which these are equivalent to a linear function. The compiler is able to detect if the conditions required by these transformation theorems are satisfied, and generate the appropriate optimised functions.

1. Introduction

The functional approach to problem solving has been gaining increasing appeal in recent years. The notion of a mapping which transforms the input to a problem into its solution is a logical starting point for deriving a specification. Moreover, expressing mappings successively in terms of mathematical compositions of other mappings, each with their own input and output objects, is a natural way of determining a complete definition of the solution. This procedure results in the type of hierarchical successive decomposition much beloved of software engineers.
Furthermore, the total absence of side effects makes functional languages semantically very attractive.

In [11], Backus took this viewpoint further and first advocated his functional style and language, FP, which provides the framework for the analysis of this paper. FP facilitates the manipulation of functions independently of any domain of objects, in contrast with the approach of repeatedly creating new objects from old ones in an auxiliary domain. It thus relates to a higher level of analysis than do the more common, object-oriented, functional languages and has its own functional algebra which simplifies reasoning about programs. This in turn can be used to provide a more formal basis for program transformation.

The main obstacle to the advancement of functional programming languages has been their poor run-time performance on conventional computers. This is primarily due to the large number of (mainly stack-based) manipulations required to preserve referential transparency in the languages. von Neumann computers execute instructions sequentially and are tailored towards supporting imperative languages with destructive assignment. An obvious alternative is to develop a radically different type of computer architecture, specifically tailored towards supporting functional languages [12, 18]. However, there is also a clear demand for efficient implementations of these languages on conventional machines, which are likely to remain widespread for the foreseeable future, whatever the impact of any new architectures. A route to improved performance is to transform recursively defined solutions into iterative ones. This may also benefit parallel architectures by providing a natural mechanism for achieving large-grain parallelism which many believe is fundamental to the whole issue of concurrent evaluation.

Previous work on optimisation has concentrated on recursion removal, which aims to transform recursive expressions into iterative form, i.e. into loops at the object level (in the imperative style), see for example [4, 5, 21]. The majority of this work has addressed linear functions. Broadly speaking, a linear function is one that generates a sequence of function calls which grows in a linear manner, and so executes in linear time with respect to the magnitude of its argument. For example, tail recursive functions (equivalent to loops) are linear, as is any function with a uni-directional or 'comb-shaped' reduction graph, such as factorial. However, a function with a balanced tree for its reduction graph is nonlinear—the Fibonacci function for example.

In the FP formalism, a linear function $f$ has the definition $f = p \rightarrow q; \ Hf$ for fixed functions $p, q$ and linear functional form $H$ defined, [2], by the property that for all functions $a, b, c, H(a \rightarrow b; c) = H.a \rightarrow Hb; Hc$ for some form $H_i$, called the predicate transformer of $H$. Functions defined by linear functional forms can be shown to possess an expansion theorem, [2], which may facilitate the subsequent derivation of loops at the object level [3]. The expansion theorem asserts that given object $x$ as argument, $f;x = (H'q);x$ where $i$ is an integer determined by $x$ and the predicate transformer. (Specifically, $i$ is the least integer such that $(H_i'p);x = T$.) Thus, for the application of $f$ to $x, f$ can be 'computed' iteratively in a loop on the domain
of functions, starting with $q$ in the ‘accumulator’ and applying $H$ to the accumulator $i$ times. Of course, in general, the increasing complexity of the representations of the sequence of functions $q, Hq, H^2q, \ldots$ renders this approach impractical, and further transformation is needed to derive an equivalent loop at the object level; see for example [16]. Further work in this area has identified a larger class of recursive program definitions which possess expansions and so has enabled more extensive algebraic reasoning [15, 23]. However, the resulting transformed definitions are difficult to map into loops at the object level automatically.

The theme of the transformations considered in this paper is to produce linear functions which are equivalent to functions of various nonlinear classes. The algebra of FP leads to a set of theorems which yield identities between functional expressions under conditions which relate only to their functional structure. The identities may then be used for function application to any set of arguments. This contrasts with corresponding work using an object-oriented approach, which yields equations satisfied by objects from certain classes. One of the simplest types of nonlinear functional forms in the function variable $f$ has more than one occurrence of $f$ and becomes linear if all but one are replaced by fixed functions, with corresponding predicate transformers which are independent of the choice of these fixed functions. This is precisely the class of ‘degenerate multilinear’ forms defined in the next section. A function which is defined in terms of such a form is shown to be equivalent to a linear definition if the predicate transformers of the form have certain properties. A significant class of functions satisfy these conditions, one example being the Fibonacci function. A similar approach enables a set of mutually recursive function definitions with multilinear defining expressions to be linearised, by considering the single function which is the FP construction of all the functions in the set.

Program transformation techniques have been studied in some depth by Darlington and Burstall [6, 8, 10], who give some quite impressive, semi-automatic optimisations. The main technique involves grouping together function references (‘tupling’) in function-defining expressions so that they may be executed in parallel, avoiding the otherwise exponential explosion in the number of calls at run time. In this way, many equivalent linear versions have been derived from nonlinear function definitions. The gains in efficiency are considerable, but the approach requires certain ‘Eureka’ steps in order to identify the right steps in the transformation process; the ‘where-abstractions’ and ‘folds’ in particular. An alternative method of optimising nonlinear functions is based on tabulation techniques [7], and memoisation [20, 22], which ‘remember’ the results of a function’s application to certain arguments by storing argument-result pairs in a table, and simply look up the result when the function is reapplied to some argument.

The transformational approach is readily expressed in FP since program transformations are inherently operations on functions, rather than on objects which tend to become a hindrance. It is well described by and developed in the algebra of functional forms. The main problem of the Darlington–Burstall methodology lies in the where-abstraction and folding stages (‘forced folding’ in [6]), viz. how to
select for folding not only functions but also the right formal parameters. The need for parameter selection originates in the object-oriented approach, and does not arise in the FP analysis, which derives general theorems relating to the linearisation of multilinear forms. The conditions for, and application of, these theorems may then be more easily automated.

In Section 2, we describe the means for linearising functions having defining expressions given by degenerate multilinear forms. This is based upon the FP representation of the heuristic tupling strategy, and yields a theorem that states the conditions which must hold in order to permit linearisation, together with the equivalent linear function definition. The conditions can be tested and the linear function generated automatically by the compiler. In Section 3 the question of a set of mutually recursively defined functions with the same domains is addressed by considering their functional construction within the FP framework. This in itself can improve execution efficiency since all of the defining expressions of the individual functions are evaluated together in a single call, so reducing the number of calls by at least some constant factor. However, considerably more optimisation is possible when the construction (or a mapping of it) is linearisable according to the methods described in Section 2. A second theorem gives the conditions for such linearisation and the equivalent linear function definition. By first expressing a degenerate multilinear function definition as a set of mutually recursive definitions, the methods described in Section 3 may also be applied to obtain the main result of Section 2. Section 4 takes this further, culminating in a theorem which when applied to FUSC ('obfuscate' of Dijkstra [13]) generates precisely the iterative version suggested by its inventor. In Section 5 the research presented here is put into perspective with a summary of related work, and the conclusions of the paper are laid out in Section 6.

Notation

\( \forall \) for all
\( \exists \) there exists
\( \not\exists \) there does not exist
\( \text{s.t.} \) such that
\( \in \) set membership
\( \not\in \) set non-membership
\( \top \) boolean value TRUE
\( \bot \) boolean value FALSE
\( \Rightarrow \) implies, as defined in [23]
\( : \) function application, e.g. \( +:(2, 3) = 5 \)
\( \circ \) function composition, e.g. \( (f \circ g): x = f:(g:x) \) \( \forall \) objects \( x \)
\( \bar{a} \) constant function, s.t. \( \bar{a}: x = a \) \( \forall \) objects \( x \not\in \bot, \bar{a}: \bot = \bot \)
\( f = p \Rightarrow q ; r \) \( f:x = \text{if } p:x = T \text{ then } q:x \text{ else if } p:x = F \text{ then } r:x \text{ else } \bot \) \( \forall \) objects \( x \)
\( \text{hd, hdr} \) primitive functions 'head', 'head right' as defined in [1]
\( \text{tl, tr} \) primitive functions 'tail', 'tail right' as defined in [1]
\( i \) \( \text{ith selector function, such that} \)
\( i:(x_1, \ldots, x_n) = x, \text{if } 1 \leq i \leq n, \)
\( \not\text{otherwise} \bot \)
\( \alpha \) 'apply to all' as defined in [1]
\( / \) 'insert' as defined in [1]
\( \text{len} \) function defined by \( \text{len}: x = T \text{ if } x \leq n, \)
\( =F \text{ if } x > n \)
\( +n \) function defined by \( (+n): x = x + n \) for number \( x \)
\( \text{an} \) function defined by \( \text{an}: x = a \times n \), for
infix arithmetic operator \( a \) and
number \( x \), e.g. \( \leq 1 = \leq 1 \)
2. Linearisation of degenerate multilinear function definitions

We first explain and define the term multilinear form. Informally, a multilinear form is a functional of several function variables, which is linear in each of its arguments. More precisely, it is linear in any argument when all of the others are fixed (i.e. non-variable), and moreover, the predicate transformer corresponding to each argument does not depend on any of the fixed values assigned to the others. Formally we have

**Definition 2.1.** $M$ is an $m$-multilinear functional form, and $M_i$ ($1 \leq i \leq m$) is the predicate transformer corresponding to the $i$th function variable argument if for all functions $f_1, f_2, \ldots, f_m, a, b, c$ and $1 \leq i \leq m$,

$$M(f_1, \ldots, f_{i-1}, a \rightarrow b; c, f_{i+1}, \ldots, f_m) = M_i(a) \rightarrow M(f_1, \ldots, f_{i-1}, b, f_{i+1}, \ldots, f_m);$$

$$M(f_1, \ldots, f_{i-1}, c, f_{i+1}, \ldots, f_m)$$

and if for some $f_1, \ldots, f_m$ with $f_i = \top$, $M(f_1, \ldots, f_m):x \neq \bot$, then $M_i a : x = T$ for all functions $a$.

In this section, we consider the function $f$ defined by $f = p \rightarrow q$; $Hf$ where $H$ is a degenerate multilinear form, i.e. where $Hf = M(f, f, \ldots, f)$ for some multilinear form $M$, defined as above. For example, the Fibonacci function, defined by $fib = le1 \rightarrow id; + \circ [fib \circ sub1, fib \circ sub2]$, is degenerate bilinear (2-multilinear) and has $M_1 a = a \circ sub1$ and $M_2 a = a \circ sub2$ in the notation of the definition.

The function $f$ has an expansion when $Hf$ is linear, e.g. when $M_1 = M_2 = \cdots = M_m$, easily seen by induction on $m$ [2]. When $Hf$ is not linear, we would like to find a function $f'$ which is equivalent to $f$, but has a defining equation in which the form $M$ is replaced by another multilinear form $M'$ which does have $M'_1 = M'_2 = \cdots = M'_m$, so yielding a linear definition. A multilinear form $M'$ with this property is called balanced. Actually, the original function is not equivalent to the linear function itself, but rather is easily extractable from it. Specifically, the original function $f$ can be obtained as the composition of a selector function with the linear function $f'$, and we will show that $f = 1 \circ f'$.

Our approach, then, is to transform degenerate multilinear functions into new functions which are linear; to be precise, defined by a certain class of balanced forms introduced in the next section. This contrasts with an alternative analysis which identifies classes of functions which possess expansions—taken by Williams [23], for example, who considered a class of degenerate bilinear (2-multilinear) functions. Thus many of Williams' functions can be linearised, a deeper comparison being made in Section 5. Function expansions are normally obtained as aids to formal reasoning, and do not necessarily induce efficient implementations.
2.1. The linearisation theorem

The main result of this section defines a transformation which converts a class of nonlinear functions, including the Fibonacci function, into equivalent linear versions. It is presented as a theorem below, the proof of which will use the following lemma.

**Lemma 2.2.** If $M$ is a balanced $m$-multilinear form with predicate transformers $M_1 = M_2 = \cdots = M_m$, and $a_1, a_2, \ldots, a_m$ are fixed strict functions, then the functional $H$, defined by $Hg = M(a_1 \circ g, a_2 \circ g, \ldots, a_m \circ g)$, is linear with predicate transformer $H_1 = M_1$.

**Proof.** Given functions $p, b, c$

\[
H(p \to b; c) = M(p \to a_1 \circ b; a_1 \circ c, p \to a_2 \circ b; a_2 \circ c, \ldots, p \to a_m \circ b; a_m \circ c)
\]

by FP laws

\[
= M_1 p \to M(a_1 \circ b, p \to a_2 \circ b; a_2 \circ c, \ldots, p \to a_m \circ b; a_m \circ c);
\]

\[
M(a_1 \circ c, p \to a_2 \circ b; a_2 \circ c, \ldots, p \to a_m \circ b; a_m \circ c)
\]

\[
= M_1 p \to M(a_1 \circ b, \ldots, a_m \circ b); M(a_1 \circ c, \ldots, a_m \circ c)
\]

by repeated application of linearity, using $M_1 = M_2 = \cdots = M_m$ and the law that $p \to (p \to q; r); s = p \to q; s$ for all functions $p, q, r, s$. Thus $H(p \to b; c) = H_1 p \to Hb; Hc$.

Now suppose $H \mathbb{I}: x \neq \mathbb{I}$ for some $x$, then $M(\mathbb{I}, \mathbb{I}, \ldots, \mathbb{I}) : x \neq \mathbb{I}$. But if for some $f_2, \ldots, f_m$, $M(\mathbb{I}, f_2, \ldots, f_m) : x \neq \mathbb{I}$ we have $M_1 p : x = T$ for all $p$ by definition of multilinear, so that $H_1 p : x = T$ and $H$ is linear. \(\square\)

The key to our linearisation techniques is expressed in Theorem 2.3 below, the use of which is illustrated by some examples in the next section. A slightly more general result can be derived with much heavier notation, and is given in the Appendix.

The general idea is to transform a nonlinear function with definition of the form

\[
f = p \to q; Hf
\]

where $H$ is degenerate multilinear, into one given in terms of a linear function $g$ defined by $g = p' \to q'; H'g$ where $H'$ is balanced. If $Hf = M(f, \ldots, f)$ where $M$ is $n$-multilinear, this may be achieved by first adjusting the formal function parameters in the body of $M$ so that we can write $M(f_1, \ldots, f_n) = M'(A_1f_1, \ldots, A_nf_n)$ for some forms $A_1f_1, \ldots, A_nf_n$ which render $M'$ balanced, i.e. make $M_1' = \cdots = M_n'$. But now we have $n$ functions $A_1f, \ldots, A_nf$ to consider rather than just one, and so we define the function $g = [A_1f, \ldots, A_nf]$ (we will actually be a little more general than this) so that $Hf = M'(1 \circ g, \ldots, n \circ g)$.

The recursive part of the equation defining $g$ may then be written as $[A_1M'(1 \circ g, \ldots, n \circ g), 2 \circ g, \ldots, n \circ g]$ which is not in general linear in $g$. However, now suppose that we can pick $A_1 = ID$ and that we have $A_2 = M_1'A_1, A_3 = M_1'A_2, \ldots, A_n = M_1'A_{n-1}$. Then the equation for $g$ has recursive part $H'g = [M'(1 \circ g, \ldots, n \circ g), M_1'(1 \circ g), \ldots, M_1'((n-1) \circ g)]$ where now $H'$ is linear with
predicate transformer $M'$. The original function $f$ is then given by the equation $f = 1 \circ g$.

$H'$ is a member of a class of linear forms defined as follows. A form $E$ is called \textit{equalised} if it is defined in terms of a balanced form $B_i$ of multilinearity $n_i$ as a construction of the form

$$E_g = [B(i_1 \circ g, \ldots, i_n \circ g), B_1(1 \circ g), \ldots, B_m(m \circ g)]$$

for some $m \geq 1$, where $1 \leq i_1 < \cdots < i_n \leq m + 1$, and in the above example, $m = n - 1$.

In Theorem 2.3, and in the paper as a whole, we will need functions which specify the domains on which given functions are defined and functions which filter out objects that do not satisfy some given predicate, such as membership of such a domain. These facilities are provided by the functionals \textit{dom} and $D$, defined as follows:

Given function $f$ and object $x$, let

$$\text{dom}_f : x = \begin{cases} T & \text{if } f(x) \neq \perp, \\ F & \text{otherwise}. \end{cases}$$

Similarly we will use in later sections the functional \textit{bool} to test when the result of a function's application is a boolean value. For function $f$ and object $x$ we therefore have

$$\text{bool}_f : x = \begin{cases} T & \text{if } f(x) \in \{T, F\}, \\ F & \text{otherwise}. \end{cases}$$

For predicate $p$, the filter, $D_p = p \rightarrow id$; $\perp$.

\textbf{Theorem 2.3.} Let $f = p \rightarrow q$: $M(f, f, \ldots, f)$ where $M(f_1, f_2, \ldots, f_m)$ is $m$-multilinear with predicate transformers $M_1, M_2, \ldots, M_m$ for $f_1, f_2, \ldots, f_m$ respectively. If

(a) there exists a strict linear form $M_0$ with predicate transformer $M_0$ such that

$$M_k = M_0^h \quad \text{for integers } 1 \leq h_1 < \cdots < h_m, 1 \leq k \leq m;$$

(b) there exists an $m$-multilinear form $M'$ with all its predicate transformers equal to $M_0$ such that

$$M(u_1, \ldots, u_m) = M'(M_0^{h_1-1}u_1, M_0^{h_2-1}u_2, \ldots, M_0^{h_m-1}u_m)$$

(if $M_0$ has an inverse, $M'(u_1, \ldots, u_m) = M(M_0 M_1^{-1}u_1, M_0 M_2^{-1}u_2, \ldots, M_0 M_m^{-1}u_m)$;

(c) $p \supset M_0^j p$ for integers $j \geq 0$,

then

(1) there exists a function $g$ such that

$$f = 1 \circ g \quad \text{and} \quad g = p \rightarrow q_0; Hg$$

where

$$Hw = [M''w, M_0(1 \circ w), \ldots, M_0((h_m - 1) \circ w)],$$

$$M''(w) = M'(h_1 \circ w, \ldots, h_m \circ w)$$
for function variable w, and

\[ q_0 = [q, M_0 q, \ldots, M_0^{b-1} q]; \]

(2) \( H \) is linear with predicate transformer \( H_1 = M_0 \), so the function \( g \) is linear;

(3) let the least fixed points of the equations \( f = p \to q; M f \ldots f \) and \( g = p \to q_0; H g \) be denoted by \( f^* \) and \( g^* \) respectively. Then

\[ f^* = 1 \circ g^* \]

provided that \( M, M_0 \) are monotonic and \( \text{dom}_p \supset M_0(\text{dom}_p) \).

The theorem therefore depends upon the ability to construct a functional form \( H \) which is equalised, its first component, \( M'' \), being balanced.

**Proof.** Let \( g = [f, M_0 f, \ldots, M_0^{b-1} f] \), then

\[ f = p \to q; M'(g) \quad \text{and} \quad g = [p \to q; M''(g), M_0 f, \ldots, M_0^{b-1} f] \]

where we abbreviate \( h_m \) by \( h \).

Now, \( p \supset f = q \), which is equivalent to the identity

\[ p \to f; \bot = p \to q; \bot. \]

Applying \( M_0 \) to both sides, noting that \( M_0 \) has predicate transformer \( M_0 \) by condition (a), we obtain, since \( M_0 \) is strict,

\[ M_0 p \to M_0 f; \bot = M_0 p \to M_0 q; \bot \]

i.e. \( M_0 p \supset M_0 f = M_0 q \). This is actually Lemma 2 of [23] which states that for linear form \( L \) with predicate transformer \( L_1 \), if \( a \supset b = c \) for functions \( a, b, c \) then \( L_1 \supset L b = L c \). By repeated application, we get

\[ M_0^k p \supset M_0^k f = M_0^k q \quad \text{for } k \geq 0. \]

But \( p \supset M_0^k p \) by condition (c). Thus, by the definition of \( g \),

\[ g = p \to [q, M_0 q, M_0^2 q, \ldots, M_0^{b-1} q]; [M''(g), M_0(1 \circ g), \ldots, M_0((h-1) \circ g)] \]

\[ = p \to q_0; H g. \]

Part (2) of the theorem follows immediately from Lemma 2.2, since all the predicate transformers of \( M' \) are equal to \( M_0 \), so that \( M'' \) is linear with predicate transformer \( M_0 \). Thus \( H \) is also linear with predicate transformer \( M_0 \), again using Lemma 2.2, and then a result of [2] which states that given linear forms \( L_1, L_2, \ldots, L_n \) each having predicate transformer \( L_1 \), their construction \([L_1, \ldots, L_n]\) is also linear and has predicate transformer \( L_1 \).

To prove part (3), we define the ordering \( \sqsubseteq \) on the domain of functions by \( f \sqsubseteq g \) iff \( f : x \sqsubseteq_{\text{obj}} g : x \) for all objects \( x \). A definition of \( \sqsubseteq_{\text{obj}} \) is assumed given, for example

\( x \sqsubseteq_{\text{obj}} y \) iff \( x = \bot \) or \( x = y \) in flat domain.
Now, by construction, \([f^*, \ldots, M_0^{h-1}f^*]\) satisfies \(g = p \rightarrow q_0; Hg\) and so is a fixed point. Thus, \(1 \circ g^* \subseteq f^*\) (which may be proved directly as shown below), and it is therefore sufficient to show that \(f^* \subseteq 1 \circ g^*\).

Let \(f_0 = g_0 = \bar{1}\) and define the ascending Kleene chains for \(f^*\) and \(g^*\) respectively by

\[
\begin{align*}
  f_{n+1} &= p \rightarrow q; M(f_n, \ldots, f_n) \\
  g_{n+1} &= p \rightarrow q_0; Hg_n \\
& \quad \text{for } n \geq 0.
\end{align*}
\]

We claim that for any non-negative integer \(n = ah + b\), where \(a, b\) are non-negative integers, \(b < h\),

\[
\text{dom}_p \supset j \circ g_{ah+b} \equiv M_0^{l-1}f_a \quad (b < j \leq h)
\]

\[
\equiv M_0^{l-1}f_{a+1} \quad (1 \leq j \leq b).
\]

Then we have in particular that \(f_a \subseteq 1 \circ g_{ah} \subseteq 1 \circ g^*\) for all \(a \geq 0\), so that \(f^* \subseteq 1 \circ g^*\) as required.

Now, for \(n = a = b = 0\), \(f_a = 1\) and the claim is trivially true \((b < 1)\).

Assume it is true for \(0 \leq ah + b \leq Ah + B\) for integers \(A, B \geq 0\) and \(B < h\). Writing \(N = Ah + B\),

\[
\begin{align*}
  g_{N+1} &= p \rightarrow q_0; [M'(h_1 \circ g_N, \ldots, h_m \circ g_N), M_0(1 \circ g_N), \ldots, M_0((h-1) \circ g_N)] \\
  &\equiv p \rightarrow q_0; [M'(M_0^{h-1}f_{A(h_1)}, \ldots, M_0^{h-1}f_{A(h_m)}), \\
  &\quad M_0f_{A(1)}, \ldots, M_0M_0^{h-2}f_{A(h-1)}]
\end{align*}
\]

where

\[
A(j) = \begin{cases} 
A + 1 & \text{if } j \leq B, \\
A & \text{if } j > B
\end{cases}
\]

by the inductive hypothesis, since \(M'\) and \(M_0\) are monotonic, and again using Lemma 2 of [23], here with the given assumption that \(\text{dom}_p \supset M_0 \text{dom}_p\).

Thus,

\[
\begin{align*}
  g_{N+1} \equiv p \rightarrow q_0; [M(f_A, \ldots, f_A), M_0f_{A(1)}, \ldots, M_0^{h-1}f_{A(h-1)}] \\
  = p \rightarrow q_0; [f_{A+1}, M_0f_{A(1)}, \ldots, M_0^{h-1}f_{A(h-1)}].
\end{align*}
\]

Now, for all \(k \geq 0\), \(p \supseteq M_0^kp \supseteq M_0^kq = M_0^kf\) as above, so that

\[
\begin{align*}
  g_{N+1} \equiv p \rightarrow [f_{A+1}, M_0f_{A(1)}, \ldots, M_0^{h-1}f_{A(h-1)}]; \\
  [f_{A+1}, M_0f_{A(1)}, \ldots, M_0^{h-1}f_{A(h-1)}]
\end{align*}
\]

and therefore

\[
\begin{align*}
  \text{dom}_p \supset j \circ g_{Ah+b+1} \equiv f_{A+1} \quad (j = 1) \\
  \equiv M_0^{j-1}f_{A(j-1)} \quad (2 \leq j \leq h),
\end{align*}
\]
i.e.

$$\text{dom}_p \supseteq j \circ g_{A_1 + B + 1} \equiv M_{0}^i f_{A_{i+1}} \quad (1 \leq j \leq B + 1)$$

$$\equiv M_{0}^i f_{A} \quad (B + 1 < j \leq h).$$

(Similarly we could claim $M_{0}^{i-1} f_n \equiv j \circ g_n$ for $1 \leq j \leq h$ to establish directly that $1 \circ g^* \equiv f^*$. The proof is much easier.) □

Notes. (i) If $M_0 a = a \circ e$ for some fixed function $e$ which has an inverse, the inverse of $M_0$ is given by $M_0^{-1} a = a \circ e^{-1}$. This is a sufficient condition for (b) of the theorem.

(ii) A special case of condition (a) is that for $2 \leq k \leq m$, $M_k = M_k^1$ for integers $i_k$ s.t. $1 \leq i_2 \leq i_3 \leq \cdots \leq i_m$. Then a sufficient condition for (a) is that $M_1$ is linear with predicate transformer equal to itself, i.e. $M_0 = M_1$.

(iii) Sufficient conditions for the existence of the integers $h_k$ are that each $M_k$ has an inverse, $M_k$ commutes with $M_{i_k}$ and $M_k = M_k^1$ for coprime integers $i_k \geq j_k \geq 1$ ($1 \leq k, l \leq m$). (Without loss of generality $1 \leq i_2 / j_2 \leq \cdots \leq i_m / j_m$, giving $1 \leq h_1 \leq \cdots \leq h_m$.) This follows since we can always find $M_{0k}$ and positive integers $r(k), s(k)$ for $1 \leq k \leq m$ such that $M_k = M_k^{s(k)}$ and $M_1 = M_0^{r(k)}$ using highest common factor arguments ($M_{0k} = M_1^{r(k)} M_k^{l(k)}$ for some integers $a, b$). Applying the same argument to $\{M_{0k} | 1 \leq k \leq m\}$ we can find $M_0$ and integers $n(k)$ such that $M_{0k} = M_0^{n(k)}$ so that $M_k = M_0^{n(k) + (k) r(k)}$ for $1 \leq k \leq m$.

(iv) For many multilinear forms $M_i$ each predicate transformer, $M_i$, is linear with its own predicate transformer equal to itself, as required by condition (a). It is shown in Lemma 1 of [15] that for all boolean functions $a, b, c$ and linear form $L$ with predicate transformer $L_i$, $L_i(a \rightarrow b; c):z = (L(a \rightarrow b; c):z$ for all objects $z$ for which there exist functions $f_z, g_z$ such that $(Lf_z):z \neq (Lg_z):z$.

(v) Condition (c) is sufficient (together with (a)), but the necessary condition is that $p \vdash M_0 \circ f = M_0 q$ for all integers $i > k \geq 0$. However, this condition is much harder to detect. In practice, condition (c) typically follows from $M_0^i p \vdash M_0^{i+1} p$ for all integers $i \geq 0$.

(vi) The linearity of $M_0$ in (a) can be relaxed to $M_0(a \rightarrow b; c) = M_0 a \rightarrow M_0 b; M_0 c$ whence $q_0$ becomes $[q, M_0 q, \ldots, M_0^{i-1} q]$.

2.2. Examples

Three examples are given to demonstrate linearisation by Theorem 2.3; first the classic Fibonacci function for which one predicate transformer is a power of the other (cf. note (ii) above), secondly a similar function for which the predicate transformers do not have inverses and thirdly one requiring a linear form $M_0$ which is not one of the predicate transformers.

2.2.1. (Fibonacci) $f = le1 \rightarrow \overline{1}; + o [f \circ \text{sub1}, f \circ \text{sub2}]$

In the notation of Theorem 2.3,

$$p = le1, \quad q = \overline{1}, \quad M(u, v) = + o [u \circ \text{sub1}, v \circ \text{sub2}].$$
Therefore \(M_1a = a \circ \text{sub}1\), \(M_2a = a \circ \text{sub}2 = M_1^2a\) and \(M_1\) is linear with predicate transformer equal to \(M_1\). \(M'(u, v) = + \circ [u \circ \text{sub}1, v \circ \text{sub}1]\), \(q_0 = [\bar{I}, \bar{I}]\) and we have \(M_1^k p = M_1^{k+1} p\) \((k \geq 0)\).

Thus, the theorem can be applied and yields \(M''g = M'(1 \circ g, 2 \circ g) = + \circ [1 \circ g \circ \text{sub}1, 2 \circ g \circ \text{sub}1] = + \circ [1, 2] \circ g \circ \text{sub}1 = + \circ g \circ \text{sub}1\). Thus,

\[
f = 1 \circ g \quad \text{where} \quad g = \text{le}1 \rightarrow [\bar{I}, \bar{I}]; [+1] \circ g \circ \text{sub}1.
\]

By application of the linear expansion theorem for the linear form \(H\) given by \(Hg = [+1] \circ g \circ \text{sub}1\), with \(H_1a = a \circ \text{sub}1\),

\[
g:x = [+1]^n \circ [\bar{I}, \bar{I}]:x - n \quad \text{for the least} \ n \ \text{s.t.} \ (H_1^* \text{le}1):x = T,
\]

\[
i.e.s.t. \ x - n \leq 1,
\]

\[
i.e.s.t. \ n = x - 1.
\]

Thus

\[
g:x = [+1]^n : (1, 1) = [+1]^{n-1}: (2, 1) = [+1]^{n-2}: (3, 2) = \cdots
\]

This reflects the usual way of implementing the Fibonacci iteration using an accumulator.

### 2.2.2. \(f = \text{null} \rightarrow \bar{0} ; + \circ [hd, + \circ [f \circ tl, f \circ tl o tl]]\)

In this example, the function \(tl\) has no inverse, but the functional \(M'\) of Theorem 2.3 can still be found, viz. \(M'(u, v) = + \circ [hd, + \circ [u \circ tl, v \circ tl]]\). Note that automatic detection of the defining expression for \(M'\) by the compiler is no more difficult in this case when \(M_1\) and \(M_2\) have no inverses because of the explicit composition of \(f\) with \(M_1d\) in the expression for \(Mff\). Moreover the same applies whenever \(M\) is a ‘simple’ form in each of its function variables, i.e. a single composition, construction or condition, or any functional composition of these (composed simple (bi)linear form [16]).

### 2.2.3. \(f = \text{le}1 \rightarrow \bar{I} ; + \circ [f \circ \text{sub}2, f \circ \text{sub}3]\)

For this example we need the most general form of condition (a) of Theorem 2.3, and find a linear form \(M_0\) with predicate transformer \(M_0\), and integers \(s > r > 1\) s.t. \(M_1 = M_0^s\), \(M_2 = M_0^r\).

Since \(M_1a = a \circ \text{sub}2\) and \(M_2a = a \circ \text{sub}3\), we have \(M_0a = a \circ \text{sub}1\). Thus we obtain \(M''g = + \circ [2, 3] \circ g \circ \text{sub}1\) and \(Hg = [+ \circ [2, 3], 1, 2] \circ g \circ \text{sub}1\), so that \(f\) may be defined by

\[
f = 1 \circ g \quad \text{where} \quad g = \text{le}1 \rightarrow [\bar{I}, \bar{I}, \bar{I}]; Hg.
\]

The assumptions required for Theorem 2.3 can be relaxed, permitting a more general result to be established. The result is much more complex notationally and unlikely to be widely applicable in practice in situations where Theorem 2.3 cannot be applied itself. It is given in the Appendix.
3. Linearisation of mutually recursive function definitions

A set of mutually recursively defined functions may be coalesced into a single function which is the construction of them all. It is this function which we attempt to make linear as per the previous section. It is first shown that, under appropriate assumptions, certain sets of functions can be linearised in this way. We then go on to show how such a set may be generalised, by relaxing these assumptions and by considering preliminary transformations of the defining equations to convert them into forms that do satisfy the assumptions. The collection of results so derived are assembled together in a theorem in Section 3.3, and examples are given in Section 3.4.

3.1. Basic result

To establish a linearisation result for a construction formed by a set of mutually recursively defined functions, we will need the following lemma. This states essentially that a multilinear form with non-distinct predicate transformers is equivalent to one with predicate transformers which are distinct—corresponding to each of the different ones in the original form.

**Lemma 3.1.** Let $M(f_1, f_2, \ldots, f_m)$ be $m$-multilinear with $M_{j_1} = M_{j_2} = \cdots = M_{j_n}$, $1 \leq j_1 < j_2 < \cdots < j_n \leq m$, $n \leq m$. If $J = \{j_1, j_2, \ldots, j_n\}$, let $u_1 = f_{j_1} = f_{j_2} = \cdots = f_{j_n}$ and $u_i = f_{j_i}$ ($2 \leq i \leq m - n + 1$) where

$$k_2 = \min_{h \in J} h, \quad k_i = \min_{h \in J, h > k_{i-1}} h$$

then the functional form $M'$ given by $M'(u_1, \ldots, u_{m-n+1}) = M(f_1, \ldots, f_m)$ is $(m - n + 1)$-multilinear with $M'_i = M_{j_i}$, $M'_i = M_{k_i}$ ($2 \leq i \leq m - n + 1$).

**Proof.** In the above notation, let $j = j_1, \ldots, j_n$. For $1 \leq i \leq m - n + 1$, let $u^i(v) = u_1, \ldots, u_{i-1}, v, u_{i+1}, \ldots, u_{m-n+1}$, and for $1 \leq j \leq m$, let $f^j(v) = f_1, \ldots, f_{j-1}, v, f_{j+1}, \ldots, f_m$. For $2 \leq i \leq m - n + 1$,

$$M'(u^i(a \to b; c)) = M(f^i(a \to b; c))
= M_{k_i} a \to M_{f^i}(b); M_{f^i}(c)$$

since $M$ is multilinear

$$= M'_{a \to M'_{u^i}(b); M'_{u^i}(c)},$$

$$M'u^1(a \to b; c) = M(f^1(a \to b; c))
= M_{k_1} a \to M_{f^1}(b); M_{f^1}(c)
= M'_{a \to M'_{u^1}(b); M'_{u^1}(c)}.$$}

Finally, if $x \neq \bot$ and $(M'u^1(\bot)) : x = \bot$, then $(M_{f^1}(\bot)) : x = \bot$ ($1 \leq i \leq m - n + 1, k_i = 1$).
Thus, $M_k a x = T$ for all $a$ since $M$ is linear in $f_k$, and so $M'_a x = T$ for all $a$. Hence, $M'$ is $(m - n + 1)$-multilinear. □

Notation. (i) If $f = f_1, \ldots, f_n$, let $M f = M(f_1, \ldots, f_n)$ for functional form $M$.
(ii) A functional form $M$ is $n$-degenerate $m$-multilinear if there exists an $m$-multilinear form $M^*$ such that $M(f_1, \ldots, f_n) = M^*(g_1, \ldots, g_m)$ where for $1 \leq j \leq m$, $g_j = f_i$ for some $i$, $1 \leq i \leq n$.

Thus any $m$-multilinear form is $m$-degenerate $m$-multilinear, and the Fibonacci function considered in Section 2 is 1-degenerate 2-multilinear.

The following proposition gives an equivalent linear function definition for a suitably restricted set of mutually recursively defined functions.

Proposition 3.2. For $1 \leq i \leq m$, let $f_i = p_i \rightarrow q_i$; $M f$ where $M_i$ is an $m$-degenerate $n_i$-multilinear form for some $n_i \geq 1$, $\text{dom}_{f_i} = \text{dom}_{f_i} = \cdots = \text{dom}_{f_m} = \text{dom}_{f}$, and $M f \circ D \text{dom} \neq 1$. Let $g = [f_1, f_2, \ldots, f_m]$ and $M_{g}^* = M(1 \circ g, 2 \circ g, \ldots, m \circ g)$. Then we have

(a) If $p_i \neq p_i$ for $1 \leq i \leq m$, then $g = p_i \rightarrow q_i$; $Ng$ where $q_0 = [q_1, \ldots, q_m]$ and $Ng = [M_{g}^* q, \ldots, M_{g}^* q]$.

(b) If $M f$ is $m$-degenerate $n$-multilinear, then $Ng$ is degenerate $n$-multilinear where $n = n_1 + \cdots + n_m$.

(c) If further for $1 \leq i \leq m$, $1 \leq j \leq n_i$, there exist integers $h(i, j) \geq 1$ and linear form $A$ with predicate transformer $A$ such that $M_{g}^* A = A^{h(i, j)}$, then $g$ is linearisable.

Notes. (i) Except for pathological cases where, for example, $A a = a \circ not$ so that $A A = ID$, condition (c) requires that every $M_{g}^* \neq ID$, but this restriction can be relaxed, see Section 3.3.

(ii) A sufficient condition for the existence of the $h(i, j)$ and the linear form $A$ is that for $1 \leq i, k \leq m$, $1 \leq j \leq n_i$, $1 \leq l \leq n_k$, there exist integers $n(ijkl) > 0$ s.t. $M_{g}^* n(ijkl) = M_{kl}^* n(klij)$, $M_{g}^*$ has an inverse, is commutative and has predicate transformer equal to $M_{g}^*$. The form $A$ is then found using highest common factor arguments, compare note (iii) after Theorem 2.3.

Proof. (a) $p_i = p_i$, so $p_i \supseteq f_i = q_i$ ($1 \leq i \leq m$) and $\neg p_i \supseteq f_i = M_{g}^* g$.

(b) Let $N'(g_{11}, \ldots, g_{1m_1}, \ldots, g_{l1}, \ldots, g_{lm_l}, \ldots, g_{m_1}, \ldots, g_{m_{m_m}}) = [M(1 \circ g_{11}, \ldots, n_1 \circ g_{1m_1}), \ldots, M(1 \circ g_{l1}, \ldots, n_l \circ g_{lm_l}), \ldots, M(1 \circ g_{m_1}, \ldots, n_m \circ g_{m_{m_m}})]$. Then $Ng = N'(g_{11}, \ldots, g_{m_{m_m}})$ with $g_{11} = g_{12} = \cdots = g_{m_{m_m}} = g$. $N'$ is $n$-multilinear with predicate transformer associated with $g_{q_i}$, $N_{q_i}^* g_i$ and $n = n_1 + \cdots + n_m$, the number of distinct $M_{g}^*$.

(c) Now follows immediately by Theorem 2.3. □

It also follows that $f_i = i \circ g^*$ where $g^*$ is the least fixed point of the equation $g = p_i \rightarrow q_0$; $Ng$ by an argument similar to that used to prove Theorem 2.3. The
proposition generalises to accommodate less stringent conditions on the predicates $p_i$ ($1 \leq i \leq m$) in the corollaries below, and further extension is given in the next two sections.

In the rest of this paper, we assume without loss of generality that the mutually recursive equations which define the functions $f_i$ ($1 \leq i \leq m$),

$$\{f_i = p_i \to q_i; M_if\} \quad \text{for} \quad 1 \leq i \leq m$$

are such that each function variable $f_j$ occurs at most once in each expression $M_if$ ($1 \leq i, j \leq m$). In particular, we will often assume that each $M_i$ is simply multilinear rather than $m$-degenerate $n_i$-multilinear (i.e. that $n_i = m$), the results derived under this assumption being easily generalised by methods analogous to those used to prove Proposition 3.2.

To justify this formally, suppose that $M_if$ contains 2 occurrences of the function variable $f_i$ ($1 \leq i \leq m$). Then we can define the new set of recursion equations, for the functions $g_0, \ldots, g_m$,

$$\{g_i = p_i \to q_i; L_ig\} \quad \text{for} \quad 0 \leq i \leq m$$

where $p_0 = p_1, q_0 = q_1, L_1 = M$, for $i \geq 2$ and $L_0g = L_1g$ is the expression obtained by replacing just one of the two occurrences of $g_i$ by $g_0$ in $M_1g$. Thus $g_0 = g_1$, and $g_i = f_i$ for $1 \leq i \leq m$. By repeating this procedure, a set of recursion equations of the required form will be obtained. To be fully rigorous, we should show that the least solutions of the equations for $f_1, \ldots, f_m$ are equal to the corresponding least solutions of the equations for $g_0, g_1, \ldots, g_m$. However, this is clearly true, as can be seen by considering the respective ascending Kleene chains for each set of equations, beginning with the zero-order approximations $f^0 = (\overline{1}, \ldots, \overline{1})$ and $g^0 = (\overline{1}, \overline{1}, \ldots, \overline{1})$, and defining $f^{i+1} = (\ldots, p_j \to q_j; M_jf^i, \ldots)$, $g^{i+1}$ similarly.

Moreover, the conditions stated for the results that will be derived are satisfied by the equations for $g$ iff they are satisfied by the equations for $f$. However, as a practical alternative, this approach is obviously less efficient than the method used in Proposition 3.2.

**Corollary 3.3.** If $p_1 \supset p_2 \supset \cdots \supset p_m$, $g = p_1 \to [q_1, \ldots, q_m];$

$$p_2 \to [M^n_1g, q_2, \ldots, q_m]; \ldots;$$

$$p_i \to [M^n_1g, \ldots, M^n_{i-1}g, q_i, \ldots, q_m]; \ldots; [M^n_1g, \ldots, M^n_mg].$$

**Proof.** Trivial, by FP laws. \(\square\)

**Corollary 3.4.** If $p_1 \supset p_2 \cdots \supset p_m$, and $p_i \supset q_i = M_if$ for $1 \leq i \leq m$, then $g = p_1 \to q_0; Ng$.

**Proof.** The general term in the expression given in Corollary 3.3 is

$$p_1 \to [M^n_1g, \ldots, M^n_{i-1}g, q_i, \ldots, q_m].$$
But by hypothesis, \( p_j \Rightarrow q_j = M_j f = M_j' g \), by definition of \( M_j'' \) \((1 \leq j \leq m)\), and since \( p_i \Rightarrow p_j \) for \( j \geq i \), the result follows. \( \square \)

The condition \( p_i \Rightarrow q_i = M_i f \) of Corollary 3.4 is rather obscure and extremely hard to check in practice, involving as it does the unknown \( f \). A more specific condition which is amenable to checking would be \( p_i \Rightarrow q_i = M_i q \). Sufficient conditions for this to render the recursion equations for \( f \) linearisable are given in Lemma A.1 of the Appendix.

A further generalisation of Proposition 3.2 relaxes the restriction that the predicates \( p_1, \ldots, p_m \) must be equal, without transforming the original equations, the latter approach being considered in the next subsection. We collate all the linearisations derived in this section and in the Appendix for mutual recursive function definitions in Theorem 3.8.

### 3.2. Preliminary transformations

We have identified two situations in which a set of mutually defined recursive functions, \( f_1, f_2, \ldots, f_m \), may be unsuitable for linearisation: when not all the predicates \( p_i \) are identical and when the defining multilinear forms, \( M_i \), in the 'else' parts on the right-hand sides have at least one predicate transformer equal to \( ID \). These problems can often be overcome using a transformation of each function definition of the type given in the following proposition. Moreover, this approach also leads to a significant extension of the domain of application of the linearisation theorem of Section 2. The following lemma will be required.

**Lemma 3.5.** If \( G, G_i \) are forms such that \( \forall a, b, c, G(a \rightarrow b; c) = G_i a \rightarrow Gb; Gc \) and \( G \) has an inverse, \( G^{-1} \), then
\[
\forall a, b, c, G^{-1}(a \rightarrow b; c) = G_i^{-1} a \rightarrow G^{-1} b; G^{-1} c
\]
provided \( G_i^{-1} \) exists.

**Proof.**
\[
G^{-1}(a \rightarrow b; c) = G^{-1}(G_i G_i^{-1} a \rightarrow G G^{-1} b; G G^{-1} c) \\
= G^{-1} G_i G_i^{-1} a \rightarrow G^{-1} b; G^{-1} c \\
= G_i^{-1} a \rightarrow G^{-1} b; G^{-1} c. \quad \square
\]

**Proposition 3.6.** Suppose we are given the mutually recursive defining equations for the function \( f_i, f_i = p_i \rightarrow q_i; M_i f \) \((1 \leq i \leq m)\), and functional forms \( T_i \) which are linear, with respective predicate transformers \( T_i t_i \), and have inverses. Then an equivalent set of function definitions is:

For \( 1 \leq i \leq m, f_i = p_i' \rightarrow q_i' \); \( M_i' f' \) where \( p_i' = T_i t_i p_i, q_i' = T_i t_i q_i \) and \( M_i'(u_1, \ldots, u_m) = T_i M_i(T_i^{-1} u_1, \ldots, T_i^{-1} u_m) \) for functions \( u_1, \ldots, u_m \), so that \( M_j' = T_i M_i T_j^{-1} \) for \( 1 \leq i, j \leq m \).
Proof. Define \( f'_i = T_i f_i \) and substitute \( f_i = T_i^{-1} f'_i \) in the expressions \( M_i f \). The expressions for the \( M'_i \) then follow using Lemma A.1 of the Appendix. \( \square \)

Corollary 3.7. Define \( T_i \) such that \( T_i p_i = p_1 \), for example \( T_i = T_1 \). Then if \( T_i \) is linear, the transformed equations for \( f'_i \) have the same predicates.

If for some \( i, k \), \( M_{ik} = ID \) \((1 \leq i, k \leq m)\), the linearisation process cannot be performed, since condition (a) of Theorem 2.3 cannot be satisfied. If \( i \neq k \), it may be possible to pre-transform the function definitions into a set such that no \( M_{ik} = ID \).

If \( i = k \), \( J_i = x \) unless \( p_i : x = T \).

Two methods of removal are given. The second is based on Proposition 3.6 and yields a more efficient set of function definitions, but does not guarantee to remove all identity predicate transformers simultaneously.

Pre-transformation 1. If \( M_{ik} = ID \) for some \( i, k \) \((1 \leq i, k \leq m)\), then we may write

\[
M'_i f = M_i(f_1, \ldots, f_{k-1}, p_k \rightarrow q_k; M_k f, f_{k+1}, \ldots, f_m)
\]

\[
= M_{ik} p_k \rightarrow M_i(f_1, \ldots, q_k, \ldots, f_m); M_i(f_1, \ldots, M_k f, \ldots, f_m).
\]

By Backus's theorems on linear forms [2] \( M'_i \) is linear in its \( k \)th argument, since \( M_i \) and \( p_k \rightarrow q_k \); \( M_k f \) is linear. Thus \( M'_{ik} \neq ID \) if \( M_{kk} \neq ID \).

Pre-transformation 2. Let \( K = \{ k \mid \exists j, 1 \leq j \leq m, M_{jk} = ID \} \). By Proposition 3.6 an equivalent set of function definitions is, for \( 1 \leq i \leq m \),

\[
f'_i = p'_i \rightarrow q'_i; M'_i f'
\]

where \( f'_i = T_i f_i, q'_i = T_i q_i, p'_i = T_i p_i \).

\( T_i \) may be any linear form which has an inverse. For example, provided \( M_i \) is linear and has an inverse, we may choose \( T_i = M_i \) for \( i \notin K \), and \( ID \) otherwise, or equivalently \( T_i = M_i^{-1} \) for \( i \in K \), \( ID \) otherwise.

These choices can be made automatically and ensure that if \( M_{ik} = ID, M'_{ik} \neq ID \). More generally, if the domain of \( f_i \) has a successor function, \( succ \) say, then we could choose one of the four possibilities \( T_i a = a \circ succ \) (or \( a \circ succ^{-1} \)) for \( i \notin K \) (or \( i \in K \)) and \( ID \) otherwise. Thus if the domain is the integers, we might have \( T_i a = a \circ add1 \). In fact often \( M_i^{-1} a = a \circ succ \).

This second method produces recursion equations which are more efficient in execution, due to the extra substitution of a right-hand side in Pre-transformation 1 which leads to a duplicated computation in each step in the linearised definitions. However, Transformation 2 cannot guarantee to remove all occurrences of identity
predicate transformers without introducing new ones. Consider for example

\[ f = eq \circ 0 \rightarrow \tilde{1}; \quad s \circ [g, f \circ s], \]
\[ g = eq \circ 0 \rightarrow \tilde{1}; \quad x \circ [g \circ s, f \circ s] \]

where \( s = sub 1 \) and we will write \( p = s^{-1} \), so \( s : n = n - 1 \) and \( p : n = n + 1 \). Writing

\[ f_1 = f, f_2 = g \text{ so that } M_{12} = ID, M_{11} a = M_{21} a = M_{22} a = a \circ s, \]

choosing \( T_2 a = a \circ p \) gives

\[ f' = eq \circ 0 \rightarrow \tilde{1}; \quad s \circ [g' \circ s, f' \circ s], \]
\[ g' = (eq - 1) \rightarrow \tilde{1}; \quad x \circ [g' \circ s, f'] \]

Hence, \( M_{12} = ID \) and attempting to remove this in the same way gives \( M_{12}' = ID \).

The problem is to find a suitable set of \( T_i \) such that \( T_i M_i T_j^{-1} \neq ID \) for any \( i, j \) (1 ≤ i, j ≤ m). If all the functionals \( T_i, M_i \) commute, as in the example, \( M_i = M_i \circ ID \), so the problem lies in having both \( T_i M_i T_j^{-1} \) and \( T_j M_j T_i^{-1} \neq ID \) simultaneously. In the above example, if \( T_i a = a \circ s^n \) and \( T_2 a = a \circ s^m \) for functions \( a \) and integers \( n, m > 0 \) (or equivalently < 0), we need \( M_{12} a \circ s^{n-m} \neq ID \) and \( M_{21} a \circ s^{n-m} \neq ID \).

Thus, \( M_{12} a = a \circ s^n \) and \( M_{21} = a \circ s^{m+1} \) and either \( m > n \) or \( m < n \). In this way we do indeed remove all the \( ID \)s but now there do not exist integers \( r \) such that \( M_{12}' = M_{21}' \), so the equations cannot be linearised by the method of Proposition 3.2.

Clearly there is interference between the transformations to remove predicate transformers equal to \( ID \) and to obtain equal predicates \( p_i \) in the defining expressions; a transformation which removes an \( ID \) predicate transformer will also cause the equality between the predicates \( p_i \) to be lost, and the converse may also occur.

3.3. Linearisation theorem for mutually recursive definitions

The results accumulated throughout Section 3, supported by the Appendix, are now assembled together in the following theorem.

**Theorem 3.8.** For 1 ≤ i, j ≤ m, let \( f_i = p_i \rightarrow q_i \); \( M_i f \) where \( M_i \) is an \( m \)-multilinear form with \( j \)-th predicate transformer \( M_{i j} \) corresponding to \( f_j \). Define the degenerate multilinear form \( N \) by \( N w = [M_i w, \ldots, M_m w] \) for function variable \( w \), where \( M_i g = M_i(1 \circ g, 2 \circ g, \ldots, m \circ g) \) for 1 ≤ i ≤ m, and suppose that \( dom_{f_i} = dom_{f_2} = \cdots = dom_{f_m} = dom \) and that \( M_i f \circ Ddom \neq \tilde{1} \).

If there exist linear form \( A \) with predicate transformer \( A \), and positive integers \( h(i, j) \) such that \( M_j = A^{h(i, j)} \), 1 ≤ i, j ≤ m, then

1. If \( p_i = p_i \) for 1 ≤ i ≤ m, \( g = p \rightarrow q \); \( Ng \) and may be linearised according to Theorem 2.3, viz. with \( p = p_1 \), \( q = [q_1, \ldots, q_m] \).
2. If (a) \( p_1 \supset p_2 \supset \cdots \supset p_m \), and
   - (b) \( p_i = A^{k(i)} p_i \) for positive integers \( k(i), 1 \leq i \leq m, \) such that \( 0 = k(1) \leq k(2) \leq \cdots \leq k(m) \), and
(c) \( \text{dom} \supset \text{dom}A_p \) for \( n \geq 0 \), i.e. for \( 1 \leq j \leq m \), if \( f_j : x \neq \bot \), then \( (A_p) : x \neq \bot \) for all \( i \geq 0 \),
then \( g = p \rightarrow q \); \( Ng \) where \( p = p_m \), \( q = [r_1, \ldots, r_m] \) and
\[
\begin{align*}
& r_i = p_i \rightarrow r_0; \quad \forall p \rightarrow r_t; \ldots; \quad A^{k(m)-1}p_i \rightarrow r_{t+1} \iff r_{t+1}
\end{align*}
\]
for fixed, known functions \( r_{ij} \), \( 1 \leq i \leq m \), \( 0 \leq j \leq k(m) \).

(3) If (a) \( p_i \supset p_2 \supset \cdots \supset p_m \), and
(b) \( p_i \supset q_i = M_iq \) and \( p_i \supset M_i p_k \) for \( 1 \leq i, j, k \leq m \),
then \( g = p \rightarrow q \); \( Ng \) where \( p = p_1 \) and \( q = [q_1, \ldots, q_m] \).

(4) If there exist integers \( k(i) \), such that \( 0 = k(1) \leq k(2) \leq \cdots \leq k(m) \), \( A^{k(i)}p_i = p_1 \) and for \( i > j \), either \( h(j, i) > k(i) - k(j) \) or \( M_{ij} \) is undefined (‘\( M \) is triangular’) (let \( g' = [A^{k(i)}f_1, \ldots, A^{k(m)}f_m] \), then \( g' = p' \rightarrow q' \); \( N'g' \) where \( N' \) is a linearisable degenerate multilinear form, and
\[
\begin{align*}
p' &= p_1 \quad \text{and} \\
q' &= [A^{k(i)}q_1, \ldots, A^{k(m)}q_m].
\end{align*}
\]

**Proof.** (1) Follows immediately from Proposition 3.2 and its corollaries.
(2) Follows from Lemma A.4.
(3) Follows from Corollary 3.4 and Lemma A.1.
(4) Define \( f_i' = A^{k(i)}f_i \) for \( 1 \leq i \leq m \). Then \( f_i' = p_i \rightarrow A^{k(i)}q_i \); \( M_i f_i' \) where \( M_i = A^{h(i) + k(i) - k(j)} \), by Proposition 3.6. Thus by hypothesis, \( M_i = A^n \), \( n > 0 \), if either \( i \leq j \), or if \( i > j \) and \( M_{ij} \) is defined. Thus \( g' = [f_1', \ldots, f_m'] \) may be linearised according to Theorem 2.3. \( \square \)

### 3.4. Examples

#### 3.4.1. Fibonacci re-visited (artificially complex version)

The following functions each compute the \( n \)th Fibonacci number (given their starting values):

\[
\begin{align*}
f_1 &= le1 \rightarrow \overline{1}; + \circ [f_1 \circ s, f_2 \circ s^2], \\
f_2 &= le1 \rightarrow \overline{1}; + \circ [f_2 \circ s, f_1 \circ s^2].
\end{align*}
\]

In the notation of Proposition 3.2, \( M_{11} a = M_{22} a = a \circ s \), \( M_{12} a = M_{21} a = -a \circ s^2 \).

Let \( g = [f_1, f_2] \), then
\[
g = le1 \rightarrow [\overline{1}, \overline{1}]; [M_1(1 \circ g, 2 \circ g), M_2(1 \circ g, 2 \circ g)]
\]
and so \( Ng \) is bilinear-degenerate. Now,
\[
Ng = [+ \circ [1 \circ u \circ s, 2 \circ v \circ s^2], + \circ [2 \circ u \circ s, 1 \circ v \circ s^2]] \quad \text{where} \quad u = v = g.
\]
The predicate transformers for \( N \) are \( N_1 a = a \circ s \) and \( N_2 a = a \circ s^2 \), so that \( N_2 = N_1^2 \).
Now let \( h = [N_1g, N_2g] \), and we have, by Theorem 2.3,

\[
h = le2 \rightarrow ([\bar{1}, \bar{1}], [\bar{1}, \bar{1}]);
\]

\[
[N_1(1 \circ 2 \circ h, 1 \circ 2 \circ h)], N_1(1 \circ h)]
\]

\[
= le2 \rightarrow ([\bar{1}, \bar{1}], [\bar{1}, \bar{1}]);
\]

\[
[N_1(1 \circ 2, 1 \circ 2)], N_1(1 \circ h).
\]

Proceeding as in the previous example, repeated composition of

\[
[[1 \circ 2, 2 \circ 2], + \circ [1 \circ 2, 1 \circ 2]], 1]
\]

shows how the Fibonacci series is computed in both parts of the construction.

### 3.4.2. List of factorials

\[
f_1 = le1 \rightarrow \bar{1}; \times \circ [id, f_1 \circ s],
\]

\[
f_2 = le0 \rightarrow \bar{nil}; ar \circ [f_2 \circ s, f_1].
\]

In the notation of Theorem 3.8, \( M_{21} = ID \), so a pre-transformation is needed. Substitution yields \( f_2' = le0 \rightarrow \bar{nil}; ar \circ [f_2 \circ s, x \circ [id, f_1 \circ s]] \) which works but involves the duplicated multiplication.

Alternatively, transformation using \( T_1a = a \circ p, T_2 = ID \) gives

\[
f_1' = le0 \rightarrow \bar{1}; \times \circ [p, f_1 \circ s],
\]

\[
f_2' = le0 \rightarrow \bar{nil}; ar \circ [f_2' \circ s, f_1' \circ s].
\]

Then, \( M_1' = M_2' = M_2' \), so that \( g = [f_1', f_2'] \) is given by

\[
g = le0 \rightarrow ([\bar{1}, \bar{nil}]; [\circ \circ [p, 1 \circ g \circ s], ar \circ [2, 1] \circ g \circ s].
\]

\( (M_1' \) is undefined. In the notation of Proposition 3.2 we have \( p_1' = p_2' \), so no further transformation is required. In any case \( p_2' \circ p_1' \circ p_1 = M_1(f_1, f_2) = M_1q, \) so the result holds anyway by either of Corollaries 3.3 and 3.4.)

Thus,

\[
g : 0 = \langle 1, \text{nil} \rangle,
\]

\[
g : 1 = [\circ \circ [p, 1 \circ \bar{1}, \bar{nil}] \circ s], ar \circ [2, 1] \circ [\bar{1}, \bar{nil}] \circ s); 1 = [\circ \circ [p, \bar{1}], ar \circ [\bar{nil}, \bar{1}]]: 1 = \langle 2, \langle 1 \rangle \rangle,
\]

\[
g : 2 = [\circ \circ [p, 1 \circ \bar{1}, \bar{nil}, \bar{1}] \circ s], ar \circ [2, 1] \circ [\circ \circ [p, \bar{1}],
\]

\[
ar \circ [\bar{nil}, \bar{1}] \circ s); 2 = [\circ \circ [p, \circ \circ [id, \bar{1}]], ar \circ [ar \circ [\bar{nil}, \bar{1}], \circ \circ [id, \bar{1}])); 2 = \langle 3 \times 2 \times 1, ar : \langle \langle 1 \rangle, 2 \times 1 \rangle = \langle 3!, \langle 11, 2! \rangle \rangle.
\]
3.4.3. ‘Out of phase pairs’

The following functions compute the values of a ‘shifted’ Fibonacci number and a list of its values:

\[ f_1 = \text{le}0 \rightarrow \text{nil}; \quad a \circ [f_1 \circ s, f_2 \circ s]. \]

\[ f_2 = \text{le}4 \rightarrow \text{id}; \quad + \circ [f_2 \circ s, f_5 \circ s \circ s]. \]

The functions would clearly be linearisable if the predicates \text{le}4 and \text{le}0 were not different, since all predicate transformers are of the form \( M_i a = a \circ s \) or \( a \circ s \circ s \) \((i, j = 1, 2)\). However, applying (b) of Theorem 3.8 (or Lemma A.4) gives

\[ f_1 = \text{le}4 \rightarrow q; \quad a \circ [f_1 \circ s, f_2 \circ s], \]

\[ f_2 = \text{le}4 \rightarrow \text{id}; \quad + \circ [f_2 \circ s, f_5 \circ s \circ s] \]

where

\[ q = \{\text{le}0 \rightarrow \text{null}; \text{le}1 \rightarrow \langle 0 \rangle; \text{le}2 \rightarrow \langle 0, 1 \rangle; \text{le}3 \rightarrow \langle 0, 1, 2 \rangle; \langle 0, 1, 2, 3 \rangle\}. \]

These functions now satisfy condition (a) of Proposition 3.2 and hence are easily made linear. Correctness and automatic computation of the ‘then’ part of \( f_1 \) are secured by Lemma A.4.

3.4.4. Simple parallel evaluation of functions applied to the same domain

Not all mutually recursive function definitions yield a degenerate multilinear construction via Theorem 3.8. However, a degree of parallelism still can often be achieved, gaining an increase in efficiency by some factor. Consider the domain of objects consisting of the set of regular binary trees, \( B \) say, and suppose we have the primitive functions

\[ lt : B \rightarrow B \quad \text{(left sub-tree)}, \]

\[ rt : B \rightarrow B \quad \text{(right sub-tree)}, \]

\[ \text{istip} : B \rightarrow \{T, F, \perp\} \quad \text{(predicate to test for a leaf)}, \]

\[ v : B \rightarrow \text{Integers} \quad \text{(value in a tip)}. \]

These functions could be defined through Abstract Data types in a more fully defined version of FP, e.g. [14], such as the extended definition of [19].

Consider the functions

\[ f_1 = \text{istip} \rightarrow v; + \circ [f_1 \circ rt, f_1 \circ lt], \]

\[ f_2 = \text{istip} \rightarrow v; \times \circ [f_2 \circ rt, f_2 \circ lt]. \]

Since \([f \circ rt, f \circ lt]\) is not linear in \( f \), only (a) of Proposition 3.2 can be used, giving that \( g = [f_1, f_2] \) is defined by

\[ g = \text{istip} \rightarrow [v, v]; + \circ [1 \circ g \circ rt, 1 \circ g \circ lt], \times \circ [2 \circ g \circ rt, 2 \circ g \circ lt]. \]

\( g \) computes both functions at once, halving the execution time corresponding to separate evaluation of \( f_1 \) and \( f_2 \).
4. A second look at degenerate multilinear defining forms

Another way of expressing the defining equation of a degenerate multilinear function is to give equations defining one function for each variable of the multilinear form concerned, with no degeneracy of the arguments. For example, if $f = p \rightarrow q; Gf$ where $Gf$ is bilinear, an equivalent definition for $f$ is given by $f_1$ or $f_2$, defined by

\[
\begin{align*}
    f_1 &= p \rightarrow q; Gf_1, \\
    f_2 &= p \rightarrow q; f_1.
\end{align*}
\]

This system of mutually recursive function definitions is bilinear in $f_1, f_2$, but has a predicate transformer equal to $ID$ ($M_2$, in Proposition 3.2). Hence a pre-transformation is necessary for application of Theorem 3.8. (Here another one than substitution! This would merely recast the problem in the degenerate multilinear form of Theorem 2.3). For a linear form $T$, with inverse $T^{-1}$ and predicate transformer $T$, let this transformation be

\[
\begin{align*}
    f'_i &= T f_i = Tp \rightarrow Tq; TG(T^{-1} f'_1, f'_2) \quad \text{and} \quad f'_2 = f_2 = p \rightarrow q; T^{-1} f'_1.
\end{align*}
\]

In the following section we generalise this approach to $m$-multilinear forms, and derive sufficient conditions for linearity of the function formed by the construction of the $m$ newly defined (and transformed) functions, $[f'_1, \ldots, f'_m]$. In this way, we again obtain an equalised form and so provide an alternative derivation of Theorem 2.3 (with slightly more restrictive conditions). A generalisation then follows in Section 4.2 which permits a larger class of functions to be linearised. In the last section, two examples are given of the new results’ application—to the Fibonacci function (which does not require the extended result) and to Dijkstra’s ‘obfuscate function’ (which does) which is transformed into its standard iterative version.

4.1. Degenerate multilinear functions and mutual recursion

Using the above procedure repeatedly, $(m-1)$ times in the $m$-multilinear case, consider the function definition $f = p \rightarrow q; Mf \ldots f$ where $M(f_1, \ldots, f_m)$ is $m$-multilinear. We first define

\[
\begin{align*}
    f_1 &= p \rightarrow q; Mf_1, \\
    f_2 &= p \rightarrow q; f_1, \\
    f_3 &= p \rightarrow q; f_2, \\
    \vdots \\
    f_m &= p \rightarrow q; f_{m-1}.
\end{align*}
\]

After applying $m$ transformations, $f'_i = T_i f_i$, $1 \leq i \leq m$, using forms $T_1, \ldots, T_m$, each with an inverse and predicate transformer equal to itself, we get

\[
\begin{align*}
    f'_i &= T_i p \rightarrow T_i q; T_i M(T_i^{-1} f'_1, T_i^{-1} f'_2, \ldots, T_i^{-1} f'_m), \\
    f'_j &= T_i p \rightarrow T_i q; T_j T_i^{-1} f'_j, \quad (1 \leq j \leq m).
\end{align*}
\]
Rewriting these equations in the form \( f'_i = a \rightarrow b \); \( N_i f' \) for \( 1 \leq i \leq m \), \( N_i \) is \( m \)-multilinear with predicate transformers \( N_{ij} = T_i M_j T_j^{-1} \), \( N_{j,j-1} = T_j T_j^{-1} \), \( 2 \leq j \leq m \), \( N_{jk} \) undefined for \( k \neq j - 1 \).

To achieve linearity, we need to have all the defined \( N_{jk} \) equal, so that the construction \([f_1, \ldots, f_m] \) has a linear defining equation without further transformation. This requires \( M_j T_j^{-1} = A \) for some fixed functional \( A \), and \( T_j T_j^{-1} = T_j A \). Hence \( M_j \) has an inverse, \( M_j = M_j M_{j-1} \) (\( j = 2, 3, \ldots, m \)), and without loss of generality, \( A = ID \). Thus, for \( 1 \leq j \leq m \), \( T_j = M_j \) and \( M_j \) also has predicate transformer equal to itself.

(It will be noticed that this condition is very close to its parallel in Theorem 2.3, and in fact the condition on \( M_j \) can be relaxed to requiring that \( M_j = M'_j \) for \( 1 = i_1 < i_2 < \cdots < i_m, 1 \leq j \leq m \). This is achieved by adding dummy functions to give a set \( g_1, g_{i_1+1}, \ldots, g_{i_m-1}, g_i \) with \( f_j = g_j \) (\( 1 \leq j \leq m \)) and defining the form \( N_g \) such that \( N_i = M_j \) and the other \( N_k \) are undefined (\( 1 \leq k \leq i_m \)).

Thus if \( g = [f'_1, \ldots, f'_m] \) and \( M_i p \geq M'_i p \) for \( 1 \leq i \leq m \), we obtain, as in Theorem 2.3,

\[
g = M_1 p \rightarrow [M_1 q, \ldots, M'_m q]; [M_1 M'' g, M_i (1 \circ g), \ldots, M_1 ((m - 1) \circ g)].
\]

### 4.2. An extended linearisation theorem

Using the approach of the previous section, an additional class of degenerate multilinear recursion equations may be linearised, as given by the following:

**Theorem 4.1.** Let \( J = p \rightarrow q; a \rightarrow Hf; Kf \) for fixed functions \( p, q, a \), where \( Hf = Lu = L u_1, u_2, \ldots, u_m \), \( Kf = Mu = Mu_1 u_2 \ldots u_m \) with \( f = u_1 = u_2 = \cdots = u_m \) and \( L, M \) are \( m \)-multilinear forms. Let \( L \) and \( M \) have predicate transformers, corresponding to their \( i \)th function variables, respectively \( AC^{i-1} \) and \( BC^{i-1} \) (\( 1 \leq i \leq m \)), for some linear forms \( A, B, \) and \( C \). Assuming

(a) \( C \) is the predicate transformer of a linear form, \( T \), and has an inverse (typically \( C = T \)),

(b) \( A = CB \),

(c) \( C^2 A = AC \),

(d) \( C a = a \) and \( C (a \circ a) = a \),

(e) \( C' p \geq C^{i+1} p \) for \( i \geq 0 \),

then \( g \), defined as \([f, T_f, T^2 f, \ldots, T^{2m-3} f]\) is the least solution of \( g = p \rightarrow [q, T_q, \ldots, T^{2m-3} q]; N_g \) where the form \( N \) is linear. (The actual definition of \( N \) is given in the proof.)

Again we will appeal to the recurring theme of finding balanced forms to replace the nonlinear (degenerate multilinear) forms \( H \) and \( K \). Here we will find that the resulting form \( N \) is much more complicated than before, being defined as a conditional chain of balanced forms.
Proof. Let \( f_i = T^i f \) for \( 0 \leq i \leq 2m - 3 \) and define \( f = (f_0, f_1, \ldots, f_{2m-3}) \). Then we may write
\[
f_i = C^i p \to T^i q; \ C^i a \to L_i f; \ M_i f
\]
where for \( 0 \leq j \leq m - 2 \) the \((2m-2)\)-multilinear forms \( L_i, M_i \) are defined by
\[
L_{2j} f = T^{2j} L(T^{-j} f, T^{-(j+1)} f_{j+1}, \ldots, T^{-(j+m-1)} f_{j+m-1}),
\]
\[
L_{2j+1} f = T^{2j+1} L(T^{-j} f, T^{-(j+1)} f_{j+1}, \ldots, T^{-(j+m)} f_{j+m}),
\]
\[
M_{2j} f = T^{2j} M(T^{-j} f, T^{-(j+1)} f_{j+1}, \ldots, T^{-(j+m-1)} f_{j+m-1}),
\]
\[
M_{2j+1} f = T^{2j+1} M(T^{-j} f, T^{-(j+1)} f_{j+1}, \ldots, T^{-(j+m)} f_{j+m}).
\]
Therefore, \( L_i f = T^i Hf; \ M_i f = T^i Kf \) as required.

Let the \( k \)th predicate transformer of \( L_j \) and \( M_j \) be \( L_{jk} \) and \( M_{jk} \) respectively \((1 \leq k \leq 2m - 2)\), then we have, for \( 1 \leq i \leq m \), the defined predicate transformers: \( L_{2j, i+j-1} = C^{2j} (AC^{i-1}) C^{-(i+j-1)} = C^{2j} AC^{-j} = A \) by (c). Similarly, \( L_{2j+1, i+j} = C^{2j+1} (AC^{i-1}) C^{-(i+j)} = C^{2j+1} AC^{-j} = C^{-1} A = B \) by (b) and \( M_{2j, i+j-1} = B, \ M_{2j+1, i+j} = CB = A \). The other predicate transformers of \( L \) and \( M \) are undefined.

Hence, since \( C^{2i} a = a \) and \( C^{2i+1} = \neg \circ a \) \((0 \leq i \leq m - 2)\),
\[
g = p \to X_0 g; \ C p \to X_1 g; \ldots; \ C^i p \to X_i g; \ldots; \ C^{2m-3} p \to X_{2m-3} g; \ X_{2m-2} g
\]
where for \( 0 \leq i \leq m - 1 \),
\[
X_{2i} = a \to [L_0 g, M_1 g, \ldots, M_{2i-1} g, T^{2i} q, T^{2i+1} q, \ldots, T^{2m-3} q];
\]
\[
[M_0 g, L_1 g, \ldots, L_{2i-1} g, T^{2i} q, T^{2i+1} q, \ldots, T^{2m-3} q]
\]
and for \( 0 \leq i \leq m - 2 \),
\[
X_{2i+1} = a \to [L_0 g, M_1 g, \ldots, L_{2i} g, T^{2i+1} q, T^{2i+2} q, \ldots, T^{2m-3} q];
\]
\[
[M_0 g, L_1 g, \ldots, M_{2i} g, T^{2i+1} q, T^{2i+2} q, \ldots, T^{2m-3} q]
\]
where we define \( L_0 g = L_0 u \) with \( u_i = i \circ g \) \((0 \leq i, j \leq 2m - 3)\) and \( M_0 g \) similarly.

Thus the functional forms \( X_{2i}, X_{2i+1} \) have predicate transformers \( X_E, X_0 \) given by \( X_E z = a \to Az; \ Bz \) and \( X_0 z = a \to Bz; \ Az \) for functions \( z \).

Hence the equation for \( g \) is linear by the conditional linear form theorem [2], and \( f \) is given by its least fixed point by an argument similar to that used in the proof of Theorem 2.3. \( \square \)

4.3. Examples

4.3.1. The Fibonacci function

Using the approach of Section 4.1 (we do not need Theorem 4.1), we may define the Fibonacci function by
\[
f_1 = \text{le} 1 \to \text{le}; \ + \circ [f_1 \circ s, f_2 \circ s \circ s],
\]
\[
f_2 = \text{le} 1 \to \text{le}; f_1.
\]
Since the domain of $f_1, f_2$ is the integers, we choose a transformation $T$ given by $Ta = a \circ s$, which is actually the same as choosing $T = M_{11}$. Noting that $T_i = T$ and $T^{-1}a = a \circ p$, we obtain

$$f'_1 = le0 \to \overline{I}; + \circ [f'_1 \circ s, f'_2 \circ s],$$

$$f'_2 = le1 \to \overline{I}; f'_1 \circ s.$$

It is immediately seen that $[f'_1, f'_2]$ is linear and that the familiar transformation given previously follows.

4.3.2. Obfuscate (Dijkstra [13])

Theorem 4.1 may be used to transform ‘FUSC’ into iterative form. Denoting ‘divide by two’ by $d$ (as $p$ and $s$ above), ‘FUSC’ is defined by $FUSC = f$ where

$$f = le1 \to id; \text{even} \to f \circ d; + \circ [f \circ d \circ p, f \circ d \circ s].$$

It is easily seen that the conditions of Theorem 4.1 are satisfied with $Ta = Ca = a \circ s$, $Aa = a \circ d$, $Ba = a \circ d \circ p$, $m = 2$. Thus, if $g = [f, f \circ s]$, we have

$$g = le1 \to [id, s];$$

$$le2 \to (\text{even} \to [L_0g, s]; [M_0g, s]);$$

$$\text{even} \to [L_0g, M_1g]; [M_0g, L_1g]$$

where

$$Lu = u_0 \circ d, \quad Mu = + \circ [u_0 \circ d \circ p, u_1 \circ d \circ s]$$

and so

$$L_0g = 1 \circ g \circ d, \quad L_1g = 2 \circ g \circ p \circ d \circ s = 2 \circ g \circ d \circ p,$$

$$M_0g = + \circ g \circ d \circ p, \quad M_1g = + \circ g \circ d.$$

Thus the last branch of the definition for $g (> 2)$ becomes

$$\text{even} \to [1, +] \circ g \circ d; [+ , 2] \circ g \circ d \circ p.$$

This reflects precisely the iteration of Dijkstra, and since the function is readily recognisable as linear in this form, the corresponding loop in an imperative programming language could be generated by the compiler. In fact this is given as an example in [16].

5. Related work

5.1. Program transformation

The main work on optimisation of functional programs by transformation was initiated by Darlington and Burstall in [10], where a set of five program schema
were given. For each schema, an equivalent iterative implementation—a loop in an imperative programming language—was given, along with a sufficient set of conditions for the transformation. When expressed purely in terms of functions in FP, schemas 1 and 2 are the same, as are schemas 4 and 5. Moreover, schemas 1, 2, 4, 5 are already linear, and schema 3 is degenerate bilinear and so linearisable by Theorem 2.3.

In [6], more formal techniques are presented, where a function definition (or set of definitions) is mapped into an equivalent definition in which the number of function calls is reduced by an order of magnitude. Very briefly, the basis of the method consists of grouping together function references, unfolding, where-abstraction and folding ('forced folding'). For a degenerate multilinear function definition, this is equivalent to forming the construction of the functions resulting from the application of powers of a certain predicate transformer to the function (Section 2). In the case of mutual recursion, the defined (possibly pretransformed) functions themselves form the construction first (Section 3), before possible further transformation. The availability of general theorems at the function level of FP greatly assists automatic transformation, and avoids many of the problems encountered in the object-oriented forced folding operation. Thus a whole class of function transformations can be represented in a unified way in FP, and moreover optimised at compile time. Specifically, the classes of function linearisable by the methods of this paper include all of those in sections 1–7 in [6] and more besides, for example FUSC in Section 4. It might be pointed out here that the defining expressions of Section 7 in [6] in particular, need no transformation in FP since they are already functionally linear. In fact in the case of factorial, a closed form expression may be derived in FP, as well as the iterative version, using the Linear Expansion Theorem [2], and Theorem 1 of [23].

Other applications of the Darlington method are a little more ad hoc and involve more Eureka steps, but can yield quite impressive optimisations. The types of function-defining expressions concerned are

(a) composition of simple functions, e.g. the length composed with the catenation of two lists,
(b) 'unrunnable' functions, such as a test for one argument being a sublist of the other,
(c) implicit definitions, a generalisation of (b), such as the concrete representation of a function between abstract data types.

A sufficient condition for optimisation of type (a) definitions is easily obtained at the function level: suppose we have the function definitions

\[ f_1 = p_1 \rightarrow q_1; \ H_1 f_1 \quad \text{and} \quad f_2 = p_2 \rightarrow q_2; \ H_2 f_2 \]

and we require an expression for the composition \( g = f_1 \circ f_2 \). If we have \((H_1 u) \circ (H_2 v) = H(u \circ v)\) for function variables \(u, v\) and some form \(H\), and \(p_1 \circ H_2 f_2 \Rightarrow p_2\), then by the laws of the functional algebra we may easily show that \( g = p_2 \rightarrow f_1 \circ q_2; \ H g \).
In the example of the length of the catenation of two lists, \( p_1 = \text{null}, q_1 = 0, H_1 f = + \circ [\overline{1}, f \circ t], p_2 = \text{null} \circ 1, q_2 = 2, H_2 f = al \circ [hd \circ 1, f \circ [tl \circ 1, 2]] \). Thus, since \( \text{null} \circ al \circ [hd \circ 1, f_2 \circ [tl \circ 1, 2]] = \overline{p_2} \),

\[
(H_1 f_1) \circ (H_2 f_2) = + \circ [\overline{1}, f_1 \circ t] \circ al \circ [hd \circ 1, f_2 \circ [tl \circ 1, 2]]
\]

by definition of \( \overline{1} \) and the identity

\[
tl \circ al \circ [u, v] = v
\]

for functions \( u, v \).

\[
= Hg, \text{ defined by } + \circ [\overline{1}, g \circ [tl \circ 1, 2]].
\]

The simplicity of the result, and its application to a particular example, illustrates the value of reasoning at the function level rather than having to deal with the domain of objects, using the generally applicable laws and theorems of the FP algebra. However, true though this is, it might be pointed out that a more sensible FP function to compute the length of a sequence is \( f_1 = / + \circ \alpha \overline{1} \) and to catenate two sequences is \( f_2 = /al \circ ar \). Such definitions are not only elegant, but also easily implemented as loops directly at the object level without transformation.

Perhaps the most interesting cases are (b) and (c) which in their most general form require unification. Object-level transformation into recursive form has been performed in special cases, using primarily ad hoc methods with Eureka steps proliferating. However, more recent research conducted at the function-level has developed a theory of inverse functions based on an extended form of FP in which, semantically, functions are defined on powerdomains [17]. In other words, all functions and their inverses map sets to sets. Recursive inverse functions can then be generated in many cases where a functional language extended to provide logical relations—or a logic language—would require unification. A further important role for this theory is emerging in the transformation of data types, whereby the prospects for automation have been enhanced considerably.

5.2. Transformations that remove recursion

Another objective of transformation systems is recursion removal. This attempts to transform recursive functions into iterative form, i.e. to convert an expression consisting of a recursively defined function applied to an object, into a form which can be implemented with bounded storage in a loop in an imperative language. Although this is not the subject of the present paper, which aims to find equivalent, functionally linear definitions which may or may not be simply translated into loops at the object level, such optimisation is clearly important both here and in its own right.

In [4], Bauer and Wossner give a quite extensive set of transformations of linear functions into ‘repetitive’ (iterative) form. Kieburtz and Shultis [21] operate at the function level using FP, and derive equivalent tail-recursive functions expressed in terms of a ‘while’ (canonical iterative) combining form, for functions with certain defining expression structures. In particular they show by fixpoint induction that
the function \( f = p \rightarrow q; h \circ [i, f \circ j] \) is equivalent to \( f' = 1 \circ w \circ [g, id] \) where \( w = p \circ 2 \rightarrow id; w \circ [h' \circ [1, i \circ 2], f \circ 2] \). Here, \( h' \) is the associative dual with respect to pivot function \( g \): for example, if \( h \) is associative, then \( h' = h \) and \( g \) is its constant unit function. This result has also been derived in [3], using the linear expansion theorem, in a somewhat weaker but more easily understood form.

Many of these methods are essentially a mathematical formalisation of early work by Darlington and Burstall [10], referred to in Section 5.1. A rigorous presentation and justification of the techniques are given in [4] which exploits properties of associativity (e.g. in the factorial example), commutativity etc. to the full, providing a compendium of useful transformations into repetitive form, expressed at the object level. (They also give transformations for nonlinear functions such as FIB and FUSC by ‘functional embedding’, equivalent to the forced-folding step of Darlington and Burstall, and indeed a similar result is given for the FIB class of functions in [21].) An algebraic approach to recursion removal has also been followed by Bird [5], who uses a combination of function-level and object-level equations and reasoning to give a concise presentation which is often ingenious, but intended more as a programming methodology than as a scheme for automatic transformation.

However, all of the analyses described above are conducted at the object level, or relate to specific structures of function defining expressions, and so are not easy to apply generally to functions defined in terms of classes of functional forms. In the great majority of cases discussed above the recursive functions optimised are functionally linear. The problem of recursion removal for the general class of recursion equations defined in terms of linear forms is the subject of current research [16] which suggests that an equivalent iterative form can be found (probably mechanistically) for any function defined by a form which is the functional composition of the basic linear forms of composition, construction and condition. All of the examples considered in this section (and the rest of this paper) could then be converted into iterative form, using the methods of previous sections if the function concerned was not already linear.

5.3. Extended expansion formulae

Degenerate bilinear forms have also been studied by [23], culminating in the ‘overrun tolerant’ theorem (ORT), which defines a class of recursion equations with expansion formulae. The expansion of a recursively defined function, when one exists, is a non-recursive solution of the recursion equation. It therefore aids formal reasoning considerably, but also produces some optimisation for that class of functions. Given overrun-tolerant function definition \( f = p \rightarrow q; Hf \), the number of iterations, \( i \), required in the computation of the ‘result-function’, \( H'q \), is linear with respect to its argument as measured by the predicate transformer \( H_1 \), in that \( i = \min(n | H^n_1 p : x = T) \). However, as noted in the Introduction, a function-level loop and accumulator are inadequate for evaluation of object-valued expressions, but the further transformation necessary to derive object-level loops from ORT expansions is not obvious.
Degenerate bilinear function definitions of the type $f = p \rightarrow q; \; Gf$, where $G$ is bilinear may be linearised according to Theorem 2.3 provided the conditions on the predicate transformers $G_1$ and $G_2$ are met. Although these conditions are more stringent than those for the ORT, it is hard to conceive of a function which can be optimised but which does not satisfy them. Often it is possible to find a pretransformation of the defining expression, $Gf$, to secure the conditions, although the linearised version will probably be no more efficient; an example is given below. Similarly, the condition that $p \supset q = Hq$ in the ORT also appears at first sight to be a severe limitation. However, in all but pathological cases, it appears that this condition too can be secured by a pre-transformation, although mechanised derivation appears difficult and often the meaning of the function will be obscured. Such a transformation is applied to the Fibonacci function in [23], giving

$$\text{fib} = \text{id} \dashv 1; + \circ [\text{fib} \circ i, \text{fib} \circ j]$$

$$i = \text{id} \dashv 0; \text{sub1}$$

$$j = \text{id} \dashv 0; \text{sub2}.$$ 

Although 'functionally' more efficient than the original definition which makes $2^n$ function calls for argument $n$, it is not obvious how to determine an iterative implementation. In any case, because of the more complicated form of the else part, $+ \circ [\text{fib} \circ i, \text{fib} \circ j]$, it is clearly less efficient than the linearised version of Section 2, which also has simpler structure, reflecting the conventional imperative implementation using destructive assignment and two accumulators.

The problem is that it is not a simple matter for any of these pre-transformations to be performed automatically—and if this were possible, it may not be that the transformed function would run any faster. Consider the regular binary tree example and the function

$$f = \text{istip} \rightarrow v; + \circ [f \circ \text{lt}, f \circ \text{rt}].$$

In the notation of Theorem 2.3, $M_1 a = a \circ \text{lt}$ and $M_2 a = a \circ \text{rt}$ and so we cannot say $M_1$ or $M_2 = M_0$, for any $M_0, i$. However, since $M_1 \text{istip}$ and $M_2 \text{istip}$ always give the same result when applied to regular trees, we may consider $M_1 = M_2$ for the domain of this computation, so that $f$ may be considered linear. However, the complexity of computing $H'v$ is almost equivalent to that of computing $f$ directly by recursive function calls. The same applies to the ORT transformed $f$:

$$f = \text{istip} \rightarrow v; + \circ [f \circ i, f \circ j]$$

$$i = \text{istip} \rightarrow v; \text{lt}$$

$$j = \text{istip} \rightarrow 0; \text{rt}$$

which ensures $\text{istip} \supset v = + \circ [v \circ i, v \circ j]$ and also (it is easily seen by induction on the depth of the tree) $H_1 \text{istip} \supset H_{i+1} \text{istip}$ ($i \geq 0$).
The ORT version also applies to irregular trees, but the linearising method fails as it stands since $M_1$ and $M_2$ are not equal. In fact, rewriting the definition to introduce dummy structures in either subtree at each stage which has no effect on the result, may enable one to conclude that $M_1 = M_2$. Linearisation would then be possible, but the effort would not produce a more efficient function (in fact probably less in view of the extra housekeeping) and would surely not be automatable.

5.4. Tabulation and memoisation

Another way to optimise nonlinear functions is to tabulate their results against the arguments to which they are applied. Then, because purely functional expressions are referentially transparent, on subsequent applications of a function to the same argument value, the result can be simply looked up rather than recomputed. Work by Cohen in this area [7] is based on the observation that there will be identifiable patterns in the dependency (or reduction) graphs of a function’s application if certain functional relationships hold between the arguments of the recursive calls. Cohen considers functions of the form

$$f = p \Rightarrow a; M(f, f) \text{ where } M(f_1, f_2) = b \circ [id, f_1 \circ c, f_2 \circ d]$$

for fixed functions $p$, $a$, $b$, $c$, $d$. Thus in our notation, the predicate transformers of $M$ are defined by $M_1 x = x \circ c$ and $M_2 x = x \circ d$ for function variable $x$. He called the functions $c$ and $d$, which correspond to these predicate transformers, descent functions, and the ‘descent conditions’ imposed on them determine the patterns in the dependency graphs which imply some measure of redundancy, i.e. potential recomputation. Tabulation strategies are developed for a four-level hierarchy of descent conditions, the commutativity of $c$ and $d$ being the weakest.

The conditions at the second level, ‘common generator redundancy’, correspond roughly to those required for the application of our Theorem 2.3 to the definition of the function $f$. The power relationship given for the predicate transformers must hold (see below) and a ‘frontier condition’ is also necessary to identify the boundary between terminal and non-terminal nodes in the graph, corresponding to base-case and recursive calls to the function $f$ respectively. The latter condition may be expressed as the implication $p \Rightarrow OR \circ [M_{ip}, NOT \circ dom_{M_{ip}}] \ (i = 1, 2)$.

The optimisation first finds the highest common generator $g$ with the property that $c = g^m$ and $d = g^n$ (in the domain of the else part of the expression for $f$) for some integers $m$, $n$. A non-local array $ARR[0: max(m, n)]$ is then declared to hold the values of the results of applications of $f$ that might be needed at later stages of the computation, and a procedure $f'$ is synthesised which does not return a result but leaves the value of $f(g^i(x))$ in $ARR[i]$, $0 \leq i \leq max(m, n)$, as a side-effect. The expression $f(x) = f(g^i(x))$ is computed by first calling in turn (but not yet evaluating) $f(g^i(x)), \ldots, f(g^k(x))$ for some appropriate $k$ for which $p(g^k(x))$ is true. For object $x$, if $p(x)$ is true, $ARR[i]$ can be set to $a(g^i(x))$ for each $i$ in the range $[0, max(m, n)]$ because of the frontier condition. If $p(x)$ is false, $f'(g(x))$ is called to place $f(g^{m+1}(x))$ into $ARR[i]$ for each $i$, the array $ARR$ is then shifted one position up, i.e. $ARR[i]$
overwrites \( ARR[i + 1] \) \((0 \leq i < \max(m, n))\), and finally \( b(x, ARR[m], ARR[n]) = b(x, f(c(x)), f(d(x)) = f(x) \) is placed in \( ARR[0] \).

Tabulation techniques such as Cohen's are rather restricted in their application, and the more general technique of memoisation constructs a table of argument-result pairs at run time. In principle, therefore, memoisation can be applied to \textit{any} function, but in practice the memo table will be forever growing, increasing the overhead involved in a look-up and possibly exhausting the available storage. One of the concerns of memoisation is to provide an efficient comparison of arguments against the argument values stored in the table entries in order to minimise look-up time, but by far the most important research area is the management of the table to limit its growth. This has been considered by Khoshnevisan [20], one of whose results provides a table managing function which minimises the size of the table, whilst ensuring no recomputations of results, for various degenerate multilinear functions which include those satisfying the conditions of our Theorem 2.3. Using a function level analysis, he establishes in particular that if the predicate transformers have a highest common factor, \( C \) say, the maximum size of the memo table is the lowest common sum of the powers of \( C \) that make up those predicate transformers.

6. Conclusion

Compile-time optimisation of a wide range of functions is possible by use of the transformation theorems presented here, together with Backus's linear expansion theorem [2], and the automatic generation of loops at the object level for linear recursion [16]. This follows because the compiler must in any case parse the forms defining the functions to be transformed, and can therefore determine directly in the simpler cases if they are multilinear using the laws for functional combination of linear forms, together with knowledge of the primitive linear forms [2]. At the same time, the predicate transformers for the resultant multilinear forms can be computed from the same theorems.

In fact, any linear definition can be further transformed to run in logarithmic time with respect to the number of times round the loop. This sometimes produces another considerable gain in efficiency, e.g. the Fibonacci sequence, but also sometimes the halving of the number of cycles is accompanied by a doubling of the complexity within each cycle, e.g. factorial. This is easily seen by recalling that if \( f = p \rightarrow q; Hf \) where \( H \) is linear, then \( f : x = (H^n q) : x \), where \( n \) is the least integer such that \( (H^n p) : x = T \).

But \( H^{2m} q = (H^2)^m q \) and \( H^{2m+1} q = H(H^{2m} q) \). For Fibonacci, \( H^2 g \) can be simplified to give \( a + [\text{double} \cdot 1, 2], \text{id} \cdot g \cdot s^2 \), equivalent to the well-known matrix product. But for factorial, \( H^2 q \) cannot be simplified.

All function-defining expressions are formed by application, to a set of functions, of the following functionals: (a) construction, (b) conditional, (c) composition, (d) application. A large class of functions formed by (a) and (b) can be optimised by
the methods presented in this paper. Case (c) may be highly nonlinear and be impossible to linearise in many instances, for example if there is no bound on the storage needed. A class of cases involving composition of \( f \) with itself which is \( p-q \) distributive [23], and so possesses an expansion, is analysed in [15], but these results appear to be of little use in practical optimisation. However, at the same time, such functions tend to be somewhat obscure and rare in practice. Case (d) involves higher-order functions and would appear most difficult.

There is also one more category, requiring the operation of unification to execute a function definition. This occurs when the left-hand side of a function definition is itself a functional of the defined function, giving an implicit definition, and may be represented as an extended definition in FP [2]. As noted in the previous section, current research in this area is very promising and has already produced some noteworthy results.

Finally, the mathematical reasoning crucial to the analysis of this paper relies heavily on the existence of the functional algebra of FP. Although all of the conclusions are equally applicable to object-oriented functional languages, it is hard to see how such general statements could have been devised at the object level. Perhaps the most striking example of this is the transformation of FUSC to iterative form, in Section 4. Theorem 4.1 is rather specific to FUSC-like functions, but does make clear the benefits of reasoning in the language of functional programs, without the necessity of taking into account properties of some auxiliary domain.

Acknowledgment

This work began during my 1984 visit to IBM Research, San Jose. I am very grateful to John Backus, John Williams and Ed Wimmers for much stimulating discussion, and especially to Ed for his time and patience in working on the early ideas. I would also like to thank the referee for constructive criticism on the first manuscript, and positive technical suggestions.

Appendix

A.1. Generalisation of Theorem 2.3

Theorem 2.3, which gives equivalent linear definitions for a class of degenerate multilinear functions under certain conditions, may be generalised according to the following proposition.

Proposition A.1. If we replace condition \( (c) \) in Theorem 2.3 by

\[
(c) \quad \mathcal{M}_0^p \ni \bigotimes_{j=1}^h \text{dom}_{\text{mp}(j)} \quad \text{where } \text{mp}(j) = \mathcal{M}_0^j p
\]
(this is always true if all the predicates are total, for example), an equivalent definition for \( f^* \) is
\[
f^* = 1 \circ g^*
\]
where \( g^* \) is the least fixed point of the equation
\[
g = p_0 \rightarrow q_0; \ p' \rightarrow Hg; \ [M^n g, M_0(1 \circ g), \ldots, M_0((h-1) \circ g)]
\]
where
\[
p_0 = \bigwedge_{j=1}^{h} M_0^j p, \quad p' = \neg \bigwedge_{j=1}^{h} \neg M_0^j p = \bigvee_{j=1}^{h} M_0^j p,
\]
and \( H \) is a linear form with predicate transformer \( H_1 \), such that for all functions \( a \),
\((H_1 a) \circ \text{dom}_{p'} = (M_0 a) \circ \text{dom}_{p'}\). \( M^n, M' \), and \( q_0 \) are as defined in Theorem 2.3.

**Proof.** Again, defining \( g = [f, M_0 f, \ldots, M_0^{h-1} f] \) and expanding the construction, we obtain
\[
g = [p \rightarrow q; \ M'^n g, M_0 p \rightarrow M_0 q; \ M_0(1 \circ g), \ldots, M_0^{h-1} p
\rightarrow M_0^{h-1} q; \ M_0((h-1) \circ g)]
\]
since \( M_0^h f = M_0(k \circ g) \) for \( 1 \leq k \leq h \), and the predicate transformer of \( M_0 \) is \( M_0 \) by hypothesis.

Expansion in full with respect to each predicate, using the laws of FP, then gives \( 2^h \) terms.

Let \( S = \{(b_1, b_2, \ldots, b_h) | b_j \in \{T, F\}, 1 \leq j \leq h\} \) and let \( s_j \in S \) be \( s_j = (s_{j_1}, s_{j_2}, \ldots, s_{j_h}) \) \((0 \leq j \leq 2^h - 1)\). \( S \) can now be used to select subsets from \( h \) predicates. For \( 0 \leq j \leq 2^h - 1 \), let
\[
p_j = \bigwedge_{0 \leq k \leq h-1} M_0^k p
\]
and \( E_j = [e_1, e_2, \ldots, e_h] \)
where for \( 2 \leq k \leq h \),
\[
e_k = \begin{cases} 
M_0^{k-1} q & \text{if } s_{jk} = T, \\
M_0((k-1) \circ g) & \text{otherwise},
\end{cases}
\]
\( e_1 = \begin{cases} 
q & \text{if } s_{j1} = T, \\
M_0^n g & \text{otherwise}.
\end{cases}
\]
Thus \( s_0 = (T, T, \ldots, T) \) and \( s_2^{h-1} = (F, F, \ldots, F) \) respectively give \( p_0 \) as defined above, \( E_0 = q_0 \) and \( p_2^{h-1} = \neg p' \), \( E_2^{h-1} = [M^n g, M_0(1 \circ g), \ldots, M_0((h-1) \circ g)] \).

We may now write the defining equation for \( g \) as
\[
g = p_0 \rightarrow E_0; \ p_1 \rightarrow E_1; \ldots; \ p_2^{h-2} \rightarrow E_2^{h-2}; \ E_2^{h-1}
\rightarrow p_0 \rightarrow E_0; \ p' \rightarrow (p_1 \rightarrow E_1; \ldots; \ p_2^{h-3} \rightarrow E_2^{h-3}; \ E_2^{h-2}); \ E_2^{h-1}
\]
by the new condition (c).

Thus \( H \) is defined by
\[
Hg = p_1 \rightarrow E_1; \ldots; \ E_2^{h-2}.
\]
But each $E_j$ is linear, being a degenerate multilinear form with equal predicate transformers, $M_0$, and so $H$ is also linear, with predicate transformer $H_1$ such that
given function $a$,$$
H_1 a = M_0 a \quad \text{if } p_i \cdot x \neq \bot \text{ for } 1 \leq i \leq 2^h - 3.
$$
This follows by repeated application of the conditional linear form theorem [2] which states that if $H$ is defined by $Hf = p \rightarrow A_f$; $Bf$ where $p$ is a fixed function and $A$, $B$ are linear forms with predicate transformers $A_0$, $B_0$ respectively, then $H$ is linear with predicate transformer $H_1$ defined by $H_1 a = p \rightarrow A_0 a$; $B_0 a$.

Thus, $(H_1 a) \circ \text{Ddom}_p = (M_0 a) \circ \text{Ddom}_p$.

The proof that $f^* = 1 \circ g^*$ is similar to that in the proof of Theorem 2.3. □

Note. Since $E_{2^{h-1}}$ is linear with predicate transformer equal to $M_0$, the definition of $g$ may be written as $g = p_0 \rightarrow q_0$; $Lg$ where $L$ is linear with predicate transformer $L_0$ with the property that $(L_0 a) \circ \text{Ddom}_p = (M_0 a) \circ \text{Ddom}_p$. This again follows from the conditional linear form theorem.

**A.2. Generalisation of Proposition 3.2 (Section 3.1)**

Proposition 3.2 may be generalised in a number of ways, as summarised in Theorem 3.8. First we derive a sufficient condition for its second corollary to hold, which is in a form that can be checked mechanically. This result is given by the following lemma.

**Lemma A.1.** If $M_i$ is $m$-multilinear, given that $p_1 \supset p_2 \supset \cdots \supset p_m$ and that for $1 \leq i \leq m$, $p_i \supset M_{jk} p_k$ for $1 \leq j \leq k \leq m$, then $p_i \supset M_i q = M_i f$ for $1 \leq i \leq j \leq m$.

**Proof.** We show by induction on $k$ that for $1 \leq i \leq j \leq m$,

$p_i \supset M_i q = M_i (f_1, \ldots, f_k, q_{k+1}, \ldots, q_m)$.

**Base case:** $p_1 \supset q_1 = f_1$ and so by Lemma 2 of [23],

$M_j p_1 \supset M_j (q_1, q_2, \ldots, q_m) = M_j (f_1, q_2, \ldots, q_m)$ \quad \text{for } 1 \leq j \leq m.

Since by hypothesis $p_i \supset M_j p_i$, we have $p_i \supset M_j q = M_j (f_1, q_2, \ldots, q_m)$ as required.

**Inductive step:** Assume the result to be true for $k = n (1 \leq n < m)$. $p_{n+1} \supset q_{n+1} = f_{n+1}$ and so

$M_{j,n+1} p_{n+1} \supset M_j (f_1, \ldots, f_n, q_{n+1}, q_{n+2}, \ldots, q_m)$

$= M_j (f_1, \ldots, f_n, f_{n+1}, q_{n+2}, \ldots, q_m)$.

But by the inductive hypothesis, $p_i \supset M_j q = M_j (f_1, \ldots, f_n, q_{n+1}, \ldots, q_m)$. Thus, since

$p_i \supset M_{j,n+1} p_{n+1}$, we have $p_i \supset M_j q = M_j (f_1, \ldots, f_n, q_{n+1}, \ldots, q_m)$. □
Corollary. If $M_i$ is $m$-multilinear, $p_1 \supset p_2 \supset \cdots \supset p_m$, $p_i \supset q_i = M_i q$ and $p_i \supset M_{jk} p_k$ for $1 \leq i, j, k \leq m$, then $g = p_1 \rightarrow q_0$; $Ng$.

It is actually quite common to have $p_1 \supset \cdots \supset p_m$ and $p_1 \supset A^n p_i$ for all $n \geq 0$, where $M_{jk} = A^{h(j,k)}$ for linear form $A$ and integer $h(j,k) \geq 0, 1 \leq i, j, k \leq m$. Thus it is easy to determine if $p_1 \supset M_{jk} p_k$ for all $j$ and for $k \geq i$. For $k < i$, the same may be true for particular cases, for example when $M_{jk}$ is undefined.

A further generalisation of Proposition 3.2 relaxes the restriction that the predicates $p_1, \ldots p_m$ must be equal. The intuition behind this analysis is that if the ‘else part’ of the construction $g$ is linearisable, then by the linear expansion Theorem [2] only a discrete set of predicates need ever be tested, for example le0, le1, \ldots. Thus, if $p_1 \supset \cdots \supset p_m$, we can have $p_m$ as the predicate in the definition of each $f_i$, checking for $p_i, p_{i+1}, \ldots, p_{m-1}$ (and, in general, some other intervening predicates) explicitly in the ‘then part’. The formal analysis begins with some technical Lemmas.

Lemma A.2. Given the linear form $A$ with predicate transformer $A_i$, fixed functions $p$ and $q$, the linear form $H$ defined by

$$Hf = p \rightarrow q; Af$$

has predicate transformer $H_i$ such that

$$H_i a = p \rightarrow v; A_i a$$

for function $a$, where $v$ is any function with the property that if for object $x$, $p : x = T$, then $v : x \in \{T, F\}$, i.e. $p \supset \text{bool}_v$.

Proof. $H$ is linear by [2].

$$H(a \rightarrow b; c) = p \rightarrow q; (A_i a \rightarrow Ab; Ac) = p \rightarrow (v \rightarrow q; q); (A_i a \rightarrow Ab; Ac)$$

for any $v$ such that $p : x = T \supset v : x \in \{T, F\}$

$$=(p \rightarrow v; A_i a) \rightarrow (p \rightarrow q; Ab); (p \rightarrow q; Ac) \quad \text{by FP laws}$$

$$= H_i a \rightarrow Hb; Hc \quad \text{with $H_i$ as defined}. \quad \Box$$

Corollary 1. Given function $a$, if $p \supset \text{bool}_{A_i a}$, then $H_i a = A_i a \circ \text{Dbool}_p$.

Proof. Choose $v = A_i a$, then $p \supset \text{bool}_v$ and so $H_i a : x = A_i a : x$ if $p : x \in \{T, F\}$. \quad \Box

Corollary 2. Let $A$ be an $n$-multilinear form with predicate transformers $A_1, \ldots, A_n$, $B$ be an $(n + 1)$-multilinear form with predicate transformers $B_1, \ldots, B_{n+1}$, and $p$ be a fixed boolean function. If the $(n + 1)$-multilinear form $M$ is defined by

$$M(f_1, \ldots, f_{n+1}) = p \rightarrow A(f_1, \ldots, f_n); B(f_1, \ldots, f_{n+1}),$$

then the predicate transformer of $M$ corresponding to $f_{n+1}, M_{n+1}$, is given by

$$M_{n+1} a = B_{n+1} a \circ \text{Dbool}_p$$

if $p \supset \text{bool}_{B_{n+1} a}$.
Proof. By linearity in the \((n+1)\) argument, using Lemma A.2 and Corollary 1. □

Lemma A.3. Given \(m\)-multilinear form \(M\) with predicate transformers \(M_1, \ldots, M_m\) such that for some boolean-valued function \(d\) and function \(a, d \supset M_1a = M_2a = \cdots = M_ma, \) the form \(H\) defined by \(Hf = M(f, f, \ldots, f)\) has the property that
\[
H(a \rightarrow b, c) \circ Dd = (M_1a \rightarrow Hb; Hc) \circ Dd.
\]

Proof. We prove by induction on \(n\) that
\[
M(a \rightarrow b; c, \ldots, a \rightarrow b; c, f_{n+1}, \ldots, f_m) \circ Dd
\]
\[
= (M_1a \rightarrow M(b, \ldots, b, f_{n+1}, \ldots, f_m);
M(c, \ldots, c, f_{n+1}, \ldots, f_m)) \circ Dd
\]
for \(1 \leq n \leq m.\)

For \(n = 1\), the result is true by the definition of multilinear. Assume it is true for \(1 \leq n < k-1 < m.\) Then
\[
M(a \rightarrow b; c, \ldots, a \rightarrow b; c, f_k, \ldots, f_m) \circ Dd
\]
\[
= \{M_{k-1}a \rightarrow M(a \rightarrow b; c, \ldots, a \rightarrow b; c, b, f_k, \ldots, f_m); M(c, \ldots, c, a, f_k, \ldots, f_m)\} \circ Dd
\]
\[
= \{M_1a \rightarrow \{M_1a \rightarrow M(b, \ldots, b, f_k, \ldots, f_m); M(c, \ldots, c, b, f_k, \ldots, f_m)\};
M(c, \ldots, c, c, f_k, \ldots, f_m)\} \circ Dd
\]
by the given properties and the inductive hypothesis
\[
= \{M_1a \rightarrow M(b, \ldots, b, f_k, \ldots, f_m); M(c, \ldots, c, f_k, \ldots, f_m)\} \circ Dd.
\]

Lemma A.4. As in Proposition 3.2, for \(1 \leq i \leq m\) and multilinear forms \(M_i, \) let \(f_i = p_i \rightarrow q_i; M_if, \) and suppose that \(\text{dom}_{p_i} = \text{dom}_{f_i} = \cdots = \text{dom}_{f_m} = \text{dom} \) and \(M_if \circ D\text{dom} \neq \perp.\)

Suppose further that there exists a linear functional form \(A\) with predicate transformer \(A,\) and integers \(h(i, j)\) such that \(M_{ij} = A^{h(i, j)}(1 \leq i, j \leq m).\) Let \(g = [f_1, f_2, \ldots, f_m].\)

Assuming that
(i) \(p_1 \supset p_2 \supset \cdots \supset p_m, \)
(ii) \(p_i = A^{k(i)p_i}\) for positive integers \(k(i), 1 \leq i \leq m, \) such that \(0 = k(1) \leq k(2) \leq \cdots \leq k(m),\)
(iii) \(\text{dom} \supset \text{bool}_{p_i}\) for \(1 \leq i \leq m\) (i.e. the predicates are boolean-valued) so that for
\(1 \leq j \leq m, \) if \(f_i x \neq \perp, \) then \((A^{p_i})x \neq \perp\) for all \(i \geq 0.\)

Then \(g\) is linearisable (by Theorem 2.3) and may be defined by
\[
g = p_m \rightarrow [r_1, \ldots, r_m]; N_g
\]
where
\[
N_g = [M_1^{r_1}g, \ldots, M_m^{r_m}g],
M_i^{r_i}g = M_i(1 \circ g, 2 \circ g, \ldots, m \circ g) \quad (1 \leq i \leq m)
\]
and
\[ r_i - p_1 \rightarrow r_{i0}; \ A p_1 \rightarrow r_{i1}; \ldots; \ A^{k(m)-1} p_1 \rightarrow r_{i(k(m)-1)}; \ r_{i(k(m))} \]
for fixed, known functions \( r_{ij}, 1 \leq i \leq m, 0 \leq j \leq k(m) \).

**Notation.** Before giving the proof, we first recap and define some new notation.

\[ g = [f_1, \ldots, f_m] \]
so that by Corollary 3.3, since \( p_1 \Rightarrow \cdots \Rightarrow p_m \)

\[ g = p_1 \Rightarrow q_0; \ Lg \]
where \( q_0 = [q_0, \ldots, q_m] \) and

\[ Lg = p_2 \Rightarrow N_2g; \ p_3 \Rightarrow N_3g; \ldots; \ p_m \Rightarrow N_mg; \ N_{m+1}g, \]

\[ N_ig = [M_i^o g, \ldots, M_{i-1}^o g, q_i, \ldots, q_m] \quad (1 \leq i \leq m+1), \]

\[ M_ig = M_i(1 \circ g, \ldots, m \circ g). \]

Thus \( N_{m+1} = N \).

Now, for \( 0 \leq j \leq m \), define

\[ N_{i+1}g' = [M_1(1 \circ g_{11}, \ldots, m \circ g_{1m}), \ldots, \]

\[ M_j(1 \circ g_{j1}, \ldots, m \circ g_{jm}), q_{j+1}, \ldots, q_m] \]

where \( g' = (g_{11}, g_{12}, \ldots, g_{1m}, g_{21}, \ldots, g_{mm}). \)

Thus

\[ N_ig \equiv N_i'(g, g, \ldots, g) \quad (2 \leq i \leq m) \]

and

\[ N_{m+1}' = N' \]
(as defined in the proof of Proposition 3.2).

Similarly we define \( L'g' = p_2 \Rightarrow N'_2g'; \ldots; \ p_m \Rightarrow N'_mg'; \ N'_{m+1}g' \) and let the predicate transformer of \( L' \) associated with \( g' \) be \( L'_1 \).

Finally, let \( e = [g, Ag, \ldots, A^{h-1}g] \) where \( h = \max_{1 \leq i \leq j \leq m} h(i, j) \).

**Proof.** By a straightforward generalisation of Lemma A.2, Corollary 1,

\[ L'_0a = M_0a \circ Dbool_{p_0} \circ Dbool_{p_0-1} \circ \cdots \circ Dbool_{p_2}, \]
and so by condition (iii), substituting \( p_1 \) for \( a \) and \( A^{h(i,j)} \) for \( M_0 \),

\[ (L'_0p_1) \circ Ddom = A^{h(i,j)} p_1 \circ Ddom. \]

Now since \( L'(g, \ldots, g) = Lg \), by Theorem 2.3 we have

\[ g = 1 \circ e \quad \text{where } e = p_1 \Rightarrow q_{00}; \ He \]
and where
\[ q_{00} = [q_0, Aq_0, \ldots, A^{h-1}q_0], \]
\[ H e = [L^e, A(1 \circ e), A(2 \circ e), \ldots, A((h-1) \circ e)], \]
\[ L^e = L^e(h(1, 1) \circ e, \ldots, h(m, m) \circ e), \]
\[ L'e' = L^e(A^{h(1,1)-1}e_{11}, \ldots, A^{h(m,m)-1}e_{mm}) \]
where \( e' = (e_{11}, \ldots, e_{mm}). \)

Now, \( L'' \) has predicate transformer \( L''^\sigma \) associated with \( e_y \), such that
\[ (L''^\sigma p_1) \circ Ddom = Ap_1 \circ Ddom \]
and so
\[ L''(p_1 \rightarrow b; c) \circ Ddom = (Ap_1 \rightarrow L''b; L''c) \circ Ddom \]
for functions \( b, c \) by Lemma A.3.

Thus, we may apply the Linear Expansion Theorem [2] to get
\[ g = 1 \circ e \]
where
\[ dom \supset e = p_1 \rightarrow q_{00}; Ap_1 \rightarrow Hq_{00}; \ldots; A^n p_1 \rightarrow H^n q_{00}; He \]
for any positive integer \( n \). In particular, taking \( n = k(m) \),
\[ g = p_1 \rightarrow q_{00}; Ap_1 \rightarrow 1 \circ Hq_{00}; \ldots; p_m \rightarrow 1 \circ H^{k(m)}q_{00}; 1 \circ He. \]

But,
\[ \neg p_m \supset Lg = Ng \]
\[ \supset L''e = Ng \]
\[ \supset (1 \circ He) = Ng \]
and so
\[ g = p_m \rightarrow (p_1 \rightarrow q_{00}; Ap_1 \rightarrow 1 \circ Hq_{00}; \ldots; A^{k(m)-1}p_1 \rightarrow 1 \circ H^{k(m)-1}q_{00}; 1 \circ H^{k(m)}q_{00}); Ng \]
so that \( r_y = i \circ 1 \circ H'/q_{00}. \)

Note that \( p_k \supset r_y = q_i \) for \( k \leq i \), so that only \( (m - k + 1) \) tests are necessary in the evaluation of the \( k \)th component of \( g \).

References


