On Mapped Reduction

Andrew Butterfield
Department of Computer Science
Trinity College, Dublin
Ireland

April 14, 1993

Keywords: Induction; List Operations; Reduction; Accumulation

Abstract

This article discusses the Mapped Reduction operator (Π) that was developed while modelling aspects of fault tolerant systems [But93b].

Two well-known operations on lists or sequences are found extensively throughout the type of discrete maths used in formal methods, and the range of functions provided in declarative languages. These are list mapping and list reduction. While these operations are very versatile, work on a model of Stable Storage [Lam81, But93b, But93a] using the techniques of the *VDM*♣ [Mac90, Mac91] produced a list operation that cannot be easliy expressed as combinations of mapping and reduction. The operator converts a list of the form:

$$< x_1, x_2, \ldots, x_{n-1}, x_n >$$

with the aid of an associative binary operator $\oplus$, to one of the form:

$$< x_1, x_1 \oplus x_2, \ldots, x_1 \oplus x_2 \oplus \cdots x_{n-1} \oplus x_n >$$

While the need for the operator arose in a specialized context, it seems sufficiently general to be worth examining in its own right. This paper examines the operator and looks at its key properties.

First, we briefly review mapping and reduction.

1.1 List Mapping

List mapping takes a function of type $X \to Y$ and a list of type $X^\star$, and returns a list of type $Y^\star$, simply by applying the function to every element of the list. Here we denote the mapping of function f over sequence σ by $f^\star\sigma$.

$$f^\star < x_1, x_2, \ldots, x_n > = < f(x_1), f(x_2), \ldots, f(x_n) >$$

We can give a formal definition of the * functor being used here:

(1) $\quad \_^\star \;:\; (X \to Y) \to X^\star \to Y^\star$

(2) $\quad f^\star\Lambda \triangleq \Lambda$

(3) $\quad f(^\star x : \sigma) \triangleq f(x) : f^\star\sigma$

Note that we use the infix "cons" operator $\_ : \_$ in preference to the more conventional, but lengthier sequence concatenation notation: $< \_ > \frown \_$.

1.2 List Reduction

List reduction comes in a variety of forms, but one of the simplest takes a binary operation of type $X \times X \to X$ and a list of type $X^\star$, and returns a item of type X, simply by using the operator in a pairwise fashion to reduce the list down to a single item. This can be performed with a leftwards or rightwards bias which determines from which end of the list the operator is first applied. Here we denote the rightwards reduction with operator $\oplus$ of sequence σ by ${}^{\oplus}/_r\sigma$

$${}^{\oplus}/_r < x_1, x_2, \ldots, x_{n-1}, x_n > = x_1 \oplus (x_2 \oplus \ldots \oplus (x_{n-1} \oplus x_n) \ldots)$$

Note that the operator is applied to the list elements from the right. We denote the leftwards reduction with operator $\oplus$ of sequence σ by ${}^{\oplus}/_l\sigma$

$${}^{\oplus}/_l < x_1, x_2, \ldots, x_{n-1}, x_n > = ((\ldots(x_1 \oplus x_2) \oplus \ldots \oplus x_{n-1}) \oplus x_n)$$

Note that the operator is applied to the list elements from the left. We can give a formal definition of the $/_r$ functor being used here:

(4) $\quad /_r \;:\; (X \times X \to X) \to X^\star \to X$

(5) $\quad {}^{\oplus}/_r\Lambda \triangleq \bot$

(6) $\quad {}^{\oplus}/_r < x > \triangleq x$

(7) $\quad {}^{\oplus}/_r(x : \sigma) \triangleq x \oplus {}^{\oplus}/_r\sigma$

We can give a formal definition of the leftwards equivalent $/_l$:

$$
\begin{array}{lrcl}
(8) & /_l & : & (X \times X \to X) \to X^\star \to X \\
(9) & {}^{\oplus}/_l \Lambda & \triangleq & \bot \\
(10) & {}^{\oplus}/_l <x> & \triangleq & x \\
(11) & {}^{\oplus}/_l (x : y : \sigma) & \triangleq & {}^{\oplus}/_l ((x \oplus y) : \sigma)
\end{array}
$$

These variants of reduction cannot handle empty sequences. Such handling requires either a more elaborate version of reduction, or rectricting its use to cases where an identity for $\oplus$ is known and can be used instead of $\bot$. Of course, if $\oplus$ is associative then the brackets are unneccessary, and the bias is not important, so we have:

(12) ${}^{\oplus}/_r = {}^{\oplus}/_l = {}^{\oplus}/$, **if** $\oplus$ is assoc.

1.3 Sections

A useful notation for converting (possibly partially applied) binary operators into functions is that of *sections*, drawn from functional programming languages like Miranda[1] [Tur85] and HASKELL[Com92]. Given a binary operator $\oplus$ of type $X \times X \to X$ they are defined as follows:

$$
\begin{array}{lrcl}
(13) & (\oplus) & : & X \times X \to X \\
(14) & (\oplus)(x, y) & \triangleq & x \oplus y \\
(15) & & & \\
(16) & (\_\oplus) & : & X \to X \\
(17) & (x\oplus)(y) & \triangleq & x \oplus y \\
(18) & & & \\
(19) & (\oplus\_) & : & X \to X \\
(20) & (\oplus y)(x) & \triangleq & x \oplus y
\end{array}
$$

Occaisionally we might use the so that $(\_ \oplus \_) = (\oplus)$, $(x \oplus \_) = (x\oplus)$ and $(\_ \oplus y) = (\oplus y)$.

1.4 Indexing and Slices

As well as the head and tail operations, we also have obvious mechanisms for accessing elements within a list (indexing) and for extracting sub-sequences

[1] Miranda is a trademark of Research Software Ltd.

(slicing).

$$
\begin{aligned}
&(21) && \_[\_] && : && X^\star \times \mathbf{N}_1 \to X \\
&(22) && \Lambda[i] && \triangleq && \bot \\
&(23) && \sigma[0] && \triangleq && \bot \\
&(24) && (x:\sigma)[1] && \triangleq && x \\
&(25) && (x:\sigma)[n+1] && \triangleq && \sigma[n]
\end{aligned}
$$

$$
\begin{aligned}
&(26) && \_[\_\ldots\_] && : && X^\star \times \mathbf{N}_1 \times \mathbf{N}_1 \to X^\star \\
&(27) && \Lambda[l\ldots h] && \triangleq && \Lambda \\
&(28) && \sigma[l\ldots 0] && \triangleq && \Lambda \\
&(29) && (x:\sigma)[1\ldots h+1] && \triangleq && x:(\sigma[1\ldots h]) \\
&(30) && (x:\sigma)[l+1\ldots h+1] && \triangleq && \sigma[l\ldots h]
\end{aligned}
$$

1.5 Some Useful Lemmas

The following lemmas will be useful for subsequent proofs. Their own proofs are left as exercises.

- Section Composition — if $\oplus$ is associative, then

  (31) $(x \oplus \_) \circ (y \oplus \_) = ((x \oplus y) \oplus \_)$

- Distributivity of Composition over Mapping

  (32) $f^\star \circ g^\star = (f \circ g)^\star$

- Selection after Mapping

  (33) $(f^\star\sigma)[n] = f(\sigma[n])$

While the introduction talked about mapped-reduction (II) using an associative operator, it is possible to have left- (II_l) and rightward (II_r) biased versions as well. The effects of both on a list $<x_1, x_2, \ldots, x_n>$ is illustrated below:

$$\text{II}_l^\oplus :< x_1, x_1 \oplus x_2, \ldots, (\ldots(x_1 \oplus x_2)\cdots \oplus x_{n-1}) \oplus x_n >$$

$$\amalg_r^{\oplus} :< x_1, x_1 \oplus x_2, \ldots, x_1 \oplus (x_2 \oplus \cdots (x_{n-1} \oplus x_n) \ldots) >$$

The definition of leftwards mapped reduction is as follows:

$$\amalg_l : X \times X \to X \to X^\star \to X^\star \tag{34}$$

$$\amalg_l^{\oplus} \Lambda \triangleq \Lambda \tag{35}$$

$$\amalg_l^{\oplus} < x > \triangleq < x > \tag{36}$$

$$\amalg_l^{\oplus} (x : y : \sigma) \triangleq x : (\amalg_l^{\oplus} ((x \oplus y) : \sigma)) \tag{37}$$

That for rightwards mapped reduction is:

$$\amalg_r : X \times X \to X \to X^\star \to X^\star \tag{38}$$

$$\amalg_r^{\oplus} \Lambda \triangleq \Lambda \tag{39}$$

$$\amalg_r^{\oplus} (x : \sigma) \triangleq x : ((x\oplus)^\star (\amalg_r^{\oplus} \sigma)) \tag{40}$$

In each case, we expect the outcome of $\amalg_x^{\oplus} \sigma$ (where $x \in \{l, r\}$) to be such that the nth element of the result is obtained by reducing the $[1 \ldots n]$ substring of σ with ${}^{\oplus}/_x$:

$$(\amalg_x^{\oplus} \sigma)[n] = {}^{\oplus}/_x (\sigma[1 \ldots n])$$

This is our basic sanity check, to confirm that the definitions given above are what we require.

Proposition $P_1(n, \sigma)$ For all n and non-null σ such that $1 \leq n \leq \text{len}\sigma$, we have:

$$(\amalg_r^{\oplus} \sigma)[n] = {}^{\oplus}/_r (\sigma[1 \ldots n])$$

Proof We prove this by double induction on n and σ.

The base case $P_1(1, \sigma)$ involves $n = 1$, and is shown for all lists such that $len\sigma \geq 1$. The list σ is therefore non-empty, and can be written as $z : \sigma'$. The details are fairly trivial.

For the inductive step, we assume the proposition is true for n and any sequence σ whose length is greater than of equal to n. We show that it would then hold for $n + 1$ given any list of the form $x : \sigma$, where x is arbitrary. We perform the proof by stating $P_1(n + 1, x : \sigma)$ and showing that its lhs and rhs reduce to the same value, using the hypothesis $P_1(n, \sigma)$ where required. We wish to show:

$$(\amalg_r^{\oplus} (x : \sigma))[n + 1] = {}^{\oplus}/_r ((x : \sigma)[1 \ldots n + 1])$$

We first reduce the lhs:

$$
\begin{array}{lll}
(\Pi_r^{\oplus}(x:\sigma))[n+1] & & \\
= \quad (x:(x\oplus)^{\star}(\Pi_r^{\oplus}\sigma))[n+1] & - - & \text{eval. } \Pi_r^{\oplus}(\ldots) \\
= \quad ((x\oplus)^{\star}(\Pi_r^{\oplus}\sigma))[n] & - - & \text{eval. } (x:\ldots)[n+1] \\
= \quad (x\oplus)(\Pi_r^{\oplus}\sigma)[n] & - - & \text{Select} \circ \text{Map Lemma} \\
= \quad x \oplus (\Pi_r^{\oplus}\sigma)[n] & - - & \text{by defn. of Section}
\end{array}
$$

We then reduce the rhs:

$$
\begin{array}{lll}
{}^{\oplus}/_r((x:\sigma)[1\ldots n+1]) & & \\
= \quad {}^{\oplus}/_r(x:(\sigma[1\ldots n+1])) & - - & \text{defn. of Slicing} \\
= \quad x \oplus ({}^{\oplus}/_r(\sigma[1\ldots n+1]) & - - & \text{defn. of } {}^{\oplus}/_r \\
= \quad x \oplus (\Pi_r^{\oplus}\sigma)[n] & - - & \text{by Hyp.}
\end{array}
$$

The conjunction of $P_1(1, z:\sigma')$ with the fact that the truth of $P_1(n+1, x:\sigma)$ is deducible from the hypothesis $P_1(n, seq)$, allows us to deduce $P_1(n,\sigma)$ ♣

Proposition $P_2(n,\sigma)$ For all non-null σ and n such that $1 \leq n \leq \text{len}\sigma$, we have:

$$(\Pi_l^{\oplus}\sigma)[n] = {}^{\oplus}/_l(\sigma[1\ldots n])$$

Proof We prove this by double induction on n and σ.

The base case $P_2(1,\sigma)$ involves $n = 1$, and is shown for all lists such that $len\sigma \geq 1$. The list σ is therefore non-empty, and can be written as $z:\sigma'$. As before, the details are fairly trivial.

For the inductive step, we assume $P_2(n,\sigma)$, for any non-null $\sigma = z:\sigma'$. We then show that given the particular *instance* of $P_2(n, z:\sigma')$ where $z = x \oplus y$, that we can deduce $P_2(n+1, x:y:\sigma')$. in other words, our hypothesis is $P_2(n, (x\oplus y):\sigma')$. We wish to show:

$$(\Pi_l^{\oplus}(x:\sigma))[n+1] = {}^{\oplus}/_l((x:\sigma)[1\ldots n+1])$$

We first reduce the lhs:

$$
\begin{array}{lll}
(\Pi_l^{\oplus}(x:\sigma'))[n+1] & & \\
= \quad (x:(\Pi_l^{\oplus}(x\oplus y):\sigma'))[n+1] & \text{- -} & \text{eval. } \Pi_l^{\oplus}(\ldots) \\
= \quad (\Pi_l^{\oplus}(x\oplus y):\sigma')[n] & \text{- -} & \text{defn. of Select} \\
= \quad {}^{\oplus}/_l(((x\oplus y):\sigma')[1\ldots n]) & \text{- -} & \text{by Hyp.}
\end{array}
$$

We then reduce the rhs:

$$
\begin{array}{lll}
{}^{\oplus}/_l((x:y:\sigma')[1\ldots n+1]) & & \\
= \quad {}^{\oplus}/_l(x:(y:\sigma'[1\ldots n])) & \text{- -} & \text{defn. of Slicing} \\
= \quad {}^{\oplus}/_l(x:y:(\sigma'[1\ldots n-1])) & \text{- -} & \text{defn. of Slicing, noting } n\geq 1 \\
= \quad {}^{\oplus}/_l((x\oplus y):(\sigma'[1\ldots n-1])) & \text{- -} & \text{defn. of } /_l \\
= \quad {}^{\oplus}/_l(((x\oplus y):\sigma')[1\ldots n])) & \text{- -} & \text{defn. of Slicing}
\end{array}
$$

The lhs and rhs are identical, and this gives us our desired result. ♣

In a similar fashion to reduction, if $\oplus$ is associative, we expect:

$$\Pi_l^{\oplus} = \Pi_r^{\oplus} = \Pi^{\oplus}$$

. Rather than perform a complete proof, we simply show that

$$\Pi_r^{\oplus}(x:y:\sigma) = x:(\Pi_r^{\oplus}(x\oplus y):\sigma)$$

when $\oplus$ is associative.

$$
\begin{array}{lll}
\Pi_r^{\oplus}(x:y:\sigma) & & \\
= \quad x:((x\oplus)^{\star}(\Pi_r^{\oplus}(y:\sigma))) & \text{- -} & \text{defn. of } \Pi_r \\
= \quad x:((x\oplus)^{\star}(y:((y\oplus)^{\star}(\Pi_r^{\oplus}(y:\sigma))))) & \text{- -} & \text{defn. of } \Pi_r \\
= \quad x:(x\oplus y):(((x\oplus)^{\star}\circ(y\oplus)^{\star})(\Pi_r^{\oplus}\sigma)) & \text{- -} & \text{defn. of } (x\oplus)^{\star} \text{ and } \circ \\
= \quad x:(x\oplus y):((((x\oplus)\circ(y\oplus))^{\star})(\Pi_r^{\oplus}\sigma)) & \text{- -} & \text{Distr. of Comp. and Map.} \\
= \quad x:(x\oplus y):((((x\oplus y)\oplus)^{\star})(\Pi_r^{\oplus}\sigma)) & \text{- -} & \text{Assoc. of } \oplus \\
= \quad x:(\Pi_r^{\oplus}((x\oplus y):\sigma)) & \text{- -} & \text{defn of } \Pi_r
\end{array}
$$

Inspection of the definition of Π_l will then show the desired identity. ♣

Left and right versions of a new operator (Π) that combines list mapping and reduction have been presented. The operators were shown correct by showing that they met the specification which states:

> The ith entry of the result must be the outcome of reducing the prefix of the input list up to and including that entry using the binary operator.

It has also been shown that the operators behave identically if the binary operation is associative.

[But93a] Andrew Butterfield. A *VDM*$^{\clubsuit}$ study of fault-tolerant stable storage — towards a computer engineering mathematics. In *FME'93: Industrial Strength Formal Methods, Odense, Denmark*, volume to appear of *Lecture Notes in Computer Science*, page ?? Formal Methods Europe, Springer Verlag, April 1993.

[But93b] Andrew Butterfield. The careful memory abstraction in stable storage. Technical Report to appear, Dept. of Comp. Science, Trinity College, Dublin, 1993.

[Com92] Haskell Committee. Report on the programming language haskell. Technical Report Version 1.2, Haskell Committee, March 1992.

[Lam81] B. W. Lampson. Atomic transactions. In *Distributed Systems, Architecture and Implementation: an Advanced Course*, volume 105 of *Lecture Notes in Computer Science*, chapter 11, pages 246–265. Springer Verlag, 1981.

[Mac90] Mícheál Mac an Airchinnigh. *Conceptual Models and Computing*. PhD thesis, Dept. of Comp. Sci,, Trinity College Dublin, Ireland, 1990.

[Mac91] Mícheál Mac an Airchinnigh. The irish school of vdm. In *VDM '91*, volume 552 of *Lecture Notes in Computer Science*. Springer Verlag, 1991.

[Tur85] D. A. Turner. Miranda: a non-strict functional language with polymorphic types. In *Functional Programming Languages and Computer Architecture*, volume 201 of *Lecture Notes in Computer Science*, pages 1–16. Springer Verlag, 1985.