Bird Meertens Formalisms (BMF)

Zhenjiang Hu

National Institute of Informatics

May 2, 2011

BMF

BMF is a calculus of functions for *people* to derive programs from specifications:

- a range of concepts and notations for defining functions over lists;
- a set of algebraic laws for manipulating functions.

A Problem

Consider the following simple identity:

$$(a_1 \times a_2 \times a_3) + (a_2 \times a_3) + a_3 + 1 = ((1 \times a_1 + 1) \times a_2 + 1) \times a_3 + 1$$

This equation generalizes in the obvious way to n variables $a_1, a_2, \ldots, a_n$, and we will refer to it as Horner'e rule.

A Problem

Consider the following simple identity:

$$(a_1 \times a_2 \times a_3) + (a_2 \times a_3) + a_3 + 1 = ((1 \times a_1 + 1) \times a_2 + 1) \times a_3 + 1$$

This equation generalizes in the obvious way to n variables $a_1, a_2, \ldots, a_n$, and we will refer to it as Horner'e rule.

- How many $\times$ are used in each side?

A Problem

Consider the following simple identity:

$$(a_1 \times a_2 \times a_3) + (a_2 \times a_3) + a_3 + 1 = ((1 \times a_1 + 1) \times a_2 + 1) \times a_3 + 1$$

This equation generalizes in the obvious way to n variables $a_1, a_2, \ldots, a_n$, and we will refer to it as Horner'e rule.

- How many $\times$ are used in each side?
- Can we generalize $\times$ to $\otimes$, $+$ to $\oplus$? What are the essential constraints for $\otimes$ and $\oplus$?

A Problem

Consider the following simple identity:

$$(a_1 \times a_2 \times a_3) + (a_2 \times a_3) + a_3 + 1 = ((1 \times a_1 + 1) \times a_2 + 1) \times a_3 + 1$$

This equation generalizes in the obvious way to n variables $a_1, a_2, \ldots, a_n$, and we will refer to it as Horner'e rule.

- How many $\times$ are used in each side?
- Can we generalize $\times$ to $\otimes$, $+$ to $\oplus$? What are the essential constraints for $\otimes$ and $\oplus$?
- Do you have suitable notation for expressing the Horner's rule concisely?

Functions

- A function f that has source type α and target type β is denoted by

$$f : \alpha \rightarrow \beta$$

  We shall say that f takes arguments in α and returns results in β.

- Function application is written without brackets; thus $f\ a$ means $f(a)$. Function application is more binding than any other operation, so $f\ a \otimes b$ means $(f\ a) \otimes b$.
- Functions are curried and applications associates to the left, so $f\ a\ b$ means $(f\ a)\ b$ (sometimes written as $f_a\ b$.

- Function composition is denoted by a centralized dot $(\cdot)$. We have

$$(f \cdot g)\ x = f(g\ x)$$

Exercise: Show the following equation state that functional compositon is associative.

$$(f\cdot) \cdot (g\cdot) = ((f \cdot g)\cdot)$$

- Binary operators will be denoted by $\oplus$, $\otimes$, $\odot$, etc. Binary operators can be sectioned. This means that $(\oplus)$, $(a\oplus)$ and $(\oplus a)$ all denote functions. The definitions are:

$$\begin{aligned}(\oplus)\ a\ b &= a \oplus b\\(a\oplus)\ b &= a \oplus b\\(\oplus b)\ a &= a \oplus b\end{aligned}$$

Exercise: If $\oplus$ has type $\oplus : \alpha \times \beta \to \gamma$, then what are the types for $(\oplus)$, $(a\oplus)$ and $(\oplus b)$ for all a in α and b in β?

- The identity element of $\oplus : \alpha \times \alpha \to \alpha$, if it exists, will be denoted by $id_\oplus$. Thus,

$$a \oplus id_\oplus = id_\oplus \oplus a = a$$

  Exericise: What is the identity element of functional composition?

- The constant values function $K : \alpha \to \beta \to \alpha$ is defined by the equation

$$K \ a \ b = a$$

Lists

- Lists are finite sequence of values of the same type. We use the notation $[\alpha]$ to describe the type of lists whose elements have type α.
  - Examples:
    $[1, 2, 1] : [\mathrm{Int}]$
    $[[1], [1, 2], [1, 2, 1]] : [[\mathrm{Int}]]$
    $[\,] : [\alpha]$

List Data Constructors

- $[\,] : [\alpha]$ constructs an empty list.
- $[.] : \alpha \rightarrow [\alpha]$ maps elements of α into singleton lists.

$$[.]\ a = [a]$$

- The primitive operator on lists is concatenation $(+\!\!+)$.

$$[1] +\!\!+ [2] +\!\!+ [1] = [1, 2, 1]$$

Concatenation is associative:

$$x +\!\!+ (y +\!\!+ z) = (x +\!\!+ y) +\!\!+ z$$

Algebraic View of Lists

- $([\alpha], +\!\!+, [\,])$ is a monoid.
- $([\alpha], +\!\!+, [\,])$ is a free monoid generated by α under the assignment $[.] : \alpha \to [\alpha]$.
- $([\alpha]^+, +\!\!+)$ is a semigroup.

List Functions: Homomorphisms

A function *h* defined in the following form is called homomorphism:

$$\begin{array}{lcl} h\ [\,] & = & id_{\oplus} \\ h\ [a] & = & f\ \ a \\ h\ (x \mathbin{+\!\!+} y) & = & h\ x \oplus h\ y \end{array}$$

It defines a map from the monoid $([\alpha], \mathbin{+\!\!+}, [\,])$ to the monoid $(\beta, \oplus : \beta \to \beta \to \beta, id_{\oplus} : \beta)$.

List Functions: Homomorphisms

A function h defined in the following form is called homomorphism:

$$\begin{array}{lcl} h\ [\,] & = & id_{\oplus} \\ h\ [a] & = & f\ \ a \\ h\ (x +\!\!\!+\ y) & = & h\ x \oplus h\ y \end{array}$$

It defines a map from the monoid $([\alpha], +\!\!\!+, [\,])$ to the monoid $(\beta, \oplus : \beta \rightarrow \beta \rightarrow \beta, id_{\oplus} : \beta)$.

Property: h is uniquely determined by f and $\oplus$.

An Example: the function returning the length of a list.

$$\begin{array}{lcl} \#\,[\,] & = & 0 \\ \#\,[a] & = & 1 \\ \#\,(x +\!\!+\, y) & = & \#\,x + \#\,y \end{array}$$

Note that $(\mathrm{Int}, +, 0)$ is a monoid.

Bags and Sets

- A bag is a list in which the order of the elements is ignored. Bags are constructed by adding the rule that $+\!\!+$ is commutative (as well as associative):

$$x +\!\!+ y = y +\!\!+ x$$

- A set is a bag in which repetitions of elements are ignored. Sets are constructed by adding the rule that $+\!\!+$ is idempotent (as well as commutative and associative):

$$x +\!\!+ x = x$$

Map

The operator $*$ (pronounced map) takes a function on Its left and a list on its right. Informally, we have

$$f * [a_1, a_2, \ldots, a_n] = [f\ a_1, f\ a_2, \ldots, f\ a_n]$$

Formally, $(f*)$ (or sometimes simply written as $f*$) is a homomorphism:

$$\begin{array}{lcl} f * [\,] & = & [\,] \\ f * [a] & = & [f\ a] \\ f * (x +\!\!+\, y) & = & (f * x) +\!\!+\, (f * y) \end{array}$$

Map Distributivity: $(f \cdot g)* = (f*) \cdot (g*)$

Reduce

The operator / (pronounced reduce) takes an associative binary operator on Its left and a list on its right. Informally, we have

$$\oplus/[a_1, a_2, \ldots, a_n] = a_1 \oplus a_2 \oplus \cdots \oplus a_n$$

Formally, $\oplus/$ is a homomorphism:

$$\begin{array}{lcll} \oplus/[\,] & = & id_\oplus & \{ \text{ if } id_\oplus \text{ exists } \} \\ \oplus/[a] & = & a & \\ \oplus/(x +\!\!+ y) & = & (\oplus/x) \oplus (\oplus/y) & \end{array}$$

Examples:

$$
\begin{array}{lcl}
\mathrm{max} & : & [\mathrm{Int}] \to \mathrm{Int} \\
\mathrm{max} & = & \uparrow / \\
 & & \textsf{where } a \uparrow b = \textsf{if } a \leq b \textsf{ then } b \textsf{ else } a \\
\mathrm{head} & : & [\alpha]^+ \to \alpha \\
\mathrm{head} & = & \lessdot / \\
 & & \textsf{where } a \lessdot b = a \\
\mathrm{last} & : & [\alpha]^+ \to \alpha \\
\mathrm{last} & = & \gtrdot / \\
 & & \textsf{where } a \gtrdot b = b
\end{array}
$$

Promotion

$f*$ and $\oplus/$ can be expressed as identities between functions.

Empty Rules

$$\begin{array}{rcl} f * \cdot K\ [] & = & K\ [] \\ \oplus/ \cdot K\ [] & = & id_{\oplus} \end{array}$$

One-Point Rules

$$\begin{array}{rcl} f * \cdot [\cdot] & = & [\cdot] \cdot f \\ \oplus/ \cdot [\cdot] & = & id \end{array}$$

Join Rules

$$\begin{array}{rcl} f * \cdot +\!\!+ / & = & +\!\!+ / \cdot (f*)* \\ \oplus/ \cdot +\!\!+ / & = & \oplus/.(\oplus/)* \end{array}$$

An Example of Calculation

$$
\begin{array}{ll}
 & \oplus/ \cdot f* \cdot +\!\!+/ \cdot g* \\
= & \quad \{ \text{ map promotion } \} \\
 & \oplus/ \cdot +\!\!+/ \cdot f** \cdot g* \\
= & \quad \{ \text{ reduce promotion } \} \\
 & \oplus/ \cdot (\oplus/)* \cdot f** \cdot g* \\
= & \quad \{ \text{ map distribution } \} \\
 & \oplus/ \cdot (\oplus/ \cdot f* \cdot g)*
\end{array}
$$

Directed Reductions

We introduce two more computation patterns $\not\rightarrow$ (pronounced left-to-right reduce) and $\not\leftarrow$ (right-to-left reduce) which are closely related to $/$. Informally, we have

$$
\begin{array}{rcl}
\oplus \not\rightarrow_e [a_1, a_2, \ldots, a_n] & = & ((e \oplus a_1) \oplus \cdots) \oplus a_n \\
\oplus \not\leftarrow_e [a_1, a_2, \ldots, a_n] & = & a_1 \oplus (a_2 \oplus \cdots \oplus (a_n \oplus e))
\end{array}
$$

Formally, we can define $\oplus \not\rightarrow_e$ on lists by two equations.

$$
\begin{array}{rcl}
\oplus \not\rightarrow_e [\,] & = & e \\
\oplus \not\rightarrow_e (x +\!\!+ [a]) & = & (\oplus \not\rightarrow_e x) \oplus a
\end{array}
$$

Exercise: Give a formal definition for $\oplus \not\leftarrow_e$.

Directed Reductions without Seeds

$$
\begin{array}{rcl}
\oplus \not\rightarrow [a_1, a_2, \ldots, a_n] & = & ((a_1 \oplus a_2) \oplus \cdots) \oplus a_n \\
\oplus \not\leftarrow [a_1, a_2, \ldots, a_n] & = & a_1 \oplus (a_2 \oplus \cdots \oplus (a_{n-1} \oplus a_n))
\end{array}
$$

Properties:

$$
\begin{array}{rcl}
(\oplus \not\rightarrow) \cdot ([a] +\!\!+) & = & \oplus \not\rightarrow_a \\
(\oplus \not\leftarrow) \cdot (+\!\!+ [a]) & = & \oplus \not\leftarrow_a
\end{array}
$$

An Example Use of Left-Reduce

Consider the right-hand side of Horner's rule:

$$(((1 \times a_1 + 1) \times a_2 + 1) \times \cdots + 1) \times a_n + 1$$

This expression can be written using a left-reduce:

$$\begin{array}{l} \odot \not\rightarrow_1 [a_1, a_2, \ldots, a_n] \\ \quad \text{where } a \odot b = (a \times b) + 1 \end{array}$$

Exercise: Give the definition of $\ominus$ such that the following holds.

$$\ominus \not\rightarrow [a_1, a_2, \ldots, a_n] = (((a_1 \times a_2 + a_2) \times a_3 + a_3) \times \cdots + a_{n-1}) \times a_n + a_n$$

Accumulations

With each form of directed reduction over lists there corresponds a form of computation called an accumulation. These forms are expressed with the operators $\not\!\!\!\!\rightarrow$ (pronounced left-accumualte) and $\not\!\!\!\!\leftarrow$ (right-accumulate) and are defined informally by

$$
\begin{array}{lcl}
\oplus \not\!\!\!\!\rightarrow_e [a_1, a_2, \ldots, a_n] & = & [e, e \oplus a_1, \ldots, ((e \oplus a_1) \oplus) \cdots \oplus a_n] \\
\oplus \not\!\!\!\!\leftarrow_e [a_1, a_2, \ldots, a_n] & = & [a_1 \oplus (a_2 \oplus \cdots \oplus (a_n \oplus e)), \ldots, a_n \oplus e, e]
\end{array}
$$

Formally, we can define $\oplus\!\not\!\!\!\!\!\not\!\!\rightarrow_e$ on lists by two equations by

$$
\begin{array}{lcl}
\oplus\!\not\!\!\!\!\!\not\!\!\rightarrow_e [\,] & = & [e] \\
\oplus\!\not\!\!\!\!\!\not\!\!\rightarrow_e ([a] \mathbin{+\!\!+} x) & = & [e] \mathbin{+\!\!+} (\oplus\!\not\!\!\!\!\!\not\!\!\rightarrow_{e \oplus a} x),
\end{array}
$$

or

$$
\begin{array}{lcl}
\oplus\!\not\!\!\!\!\!\not\!\!\rightarrow_e [\,] & = & [e] \\
\oplus\!\not\!\!\!\!\!\not\!\!\rightarrow_e (x \mathbin{+\!\!+} [a]) & = & (\oplus\!\not\!\!\!\!\!\not\!\!\rightarrow_e x) \mathbin{+\!\!+} [b \oplus a] \\
 & & \text{where } b = \mathrm{last}(\oplus\!\not\!\!\!\!\!\not\!\!\rightarrow_e x).
\end{array}
$$

Efficiency in Accumulate

$\oplus \not\!\!\!\!\!\rightarrow_e [a_1, a_2, \ldots, a_n]$: can be evaluated with $n-1$ calculations of $\oplus$.

Exercise: Consider computation of first $n+1$ factorial numbers: $[0!, 1!, \ldots, n!]$. How many calculations of $\times$ are required for the following two programs?

1. $\times \not\!\!\!\!\!\rightarrow_1 [1, 2, \ldots, n]$
2. $\mathrm{fact} * [0, 1, 2, \cdots, n]$ where $\mathrm{fact}\ 0 = 1$ and $\mathrm{fact}\ k = 1 \times 2 \times \cdots \times k$.

Relation between Reduce and Accumulate

$$\oplus \not\to_e = \text{last} \cdot \oplus\!\not\!\!\!/\!\!\to_e$$

$$\oplus\!\not\!\!\!/\!\!\to_e = \otimes \not\to_{[e]}$$

$$\text{where } x \otimes a = x +\!\!+ [\text{last } x \oplus a]$$

Segments

A list y is a segment of x if there exists u and v such that

$$x = u +\!\!+\, y +\!\!+\, v.$$

If $u = []$, then y is called an initial segment.
If $v = []$, then y is called an final segment.

An Example:

$$\text{segs } [1,2,3] = [[],[1],[1,2],[2],[1,2,3],[2,3],[3]]$$

Exercise: How many segments for a list $[a_1, a_2, \ldots, a_n]$?

inits

The function inits returns the list of initial segments of a list, in increasing order of a list.

$$\text{inits } [a_1, a_2, \ldots, a_n] = [[\,], [a_1], [a_1, a_2], \ldots, [a_1, a_2, \ldots, a_n]]$$

$$\text{inits} = (+\!\!+ \not\!\!\rightarrow_{[]}) \cdot [\cdot]*$$

tails

The function tails returns the list of final segments of a list, in decreasing order of a list.

$$\text{tails}\ [a_1, a_2, \ldots, a_n] = [[a_1, a_2, \ldots, a_n], [a_2, \ldots, a_n], \ldots, [a_n], [\,]]$$

$$\text{tails} = (+\!\!+ \not\leftarrow\!\!\!\!/\!\!\!/_{[]}) \cdot [\cdot]*$$

segs

$$segs = +\!\!+ \; / \cdot \mathrm{tails} * \cdot \mathrm{inits}$$

Exercise: Show the result of $\mathrm{segs}\ [1, 2]$.

Accumulation Lemma

$$(\oplus \not\!\!\!/\!\!\!\rightarrow_e) = (\oplus \not\rightarrow_e) * \cdot \mathrm{inits}$$
$$(\oplus \not\!\!\!/\!\!\!\rightarrow) = (\oplus \not\rightarrow) * \cdot \mathrm{inits}^+$$

The accumulation lemma is used frequently in the derivation of efficient algorithms for problems about segments.

> *On lists of length n, evaluation of the LHS requires $O(n)$ computations involving $\oplus$, while the RHS requires $O(n^2)$ computations.*

The Problem: Revisit

Consider the following simple identity:

$$(a_1 \times a_2 \times a_3) + (a_2 \times a_3) + a_3 + 1 = ((1 \times a_1 + 1) \times a_2 + 1) \times a_3 + 1$$

This equation generalizes in the obvious way to n variables $a_1, a_2, \ldots, a_2$, and we will refer to it as Horner'e rule.

- Can we generalize $\times$ to $\otimes$, $+$ to $\oplus$? What are the essential constraints for $\otimes$ and $\oplus$?
- Do you have suitable notation for expressing the Horner's rule concisely?

Horner's Rule

The following equation

$$\oplus/ \cdot \otimes/ * \cdot \text{tails} = \odot \not\rightarrow_{e}$$

$$\begin{aligned} &\text{where} \\ &\quad e = id_{\otimes} \\ &\quad a \odot b = (a \otimes b) \oplus e \end{aligned}$$

holds, provided that $\otimes$ distributes (backwards) over $\oplus$:

$$(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$$

for all a, b, and c.

Exercise: Prove the correctness of the Horner's rule.
Hints:

- Show that
$$(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$$
is equivalent to
$$(\otimes c) \cdot \oplus/ = \oplus/ \cdot (\otimes c) * .$$

- Show that
$$f = \oplus/ \cdot \otimes/ * \cdot \mathrm{tails}$$
satisfies the equations
$$\begin{array}{lcl} f\ [\,] & = & e \\ f\ (x +\!\!+ [a]) & = & f\ x \odot a \end{array}$$

Generalizations of Horner's Rule

Generalization 1:

$$\oplus/ \cdot \otimes/ * \cdot \mathrm{tails}^{+} = \odot \not\rightarrow$$

where

$$a \odot b = (a \otimes b) \oplus b$$

Generalizations of Horner's Rule

Generalization 1:

$$\oplus/ \cdot \otimes/ * \cdot \mathrm{tails}^{+} = \odot \not\rightarrow$$

where

$$a \odot b = (a \otimes b) \oplus b$$

Generalization 2:

$$\oplus/ \cdot (\otimes/ \cdot f*) * \cdot \mathrm{tails} = \odot \not\rightarrow_{e}$$

where

$$e = id_{\otimes}$$

$$a \odot b = (a \otimes f \ \ b) \oplus e$$

The Maximum Segment Sum (mss) Problem

Compute the maximum of the sums of all segments of a given sequence of numbers, positive, negative, or zero.

$$mss\ [3, 1, -4, 1, 5, -9, 2] = 6$$

A Direct Solution

$$mss = \uparrow / \cdot +/ * \cdot segs$$

Calculating a Linear Algorithm using Horner's Rule

$$
\begin{array}{ll}
 & mss \\
= & \quad \{ \text{ definition of } mss \ \} \\
 & \uparrow / \cdot +/ * \cdot segs \\
= & \quad \{ \text{ definition of } segs \ \} \\
 & \uparrow / \cdot +/ * \cdot +\!\!+ / \cdot tails * \cdot inits \\
= & \quad \{ \text{ map and reduce promotion } \} \\
 & \uparrow / \cdot (\uparrow / \cdot +/ * \cdot tails) * \cdot inits \\
= & \quad \{ \text{ Horner's rule with } a \odot b = (a + b) \uparrow 0 \ \} \\
 & \uparrow / \cdot \odot \not\rightarrow_0 * \cdot inits \\
= & \quad \{ \text{ accumulation lemma } \} \\
 & \uparrow / \cdot \odot \not\!\not\rightarrow_0
\end{array}
$$

A Program in Haskell

Exercise: Code the derived linear algorithm for *mss* in your favorite programming language.

Segment Decomposition

The sequence of calculation steps given in the derivation of the *mss* problem arises frequently. The essential idea can be summarized as a general theorem.

Theorem (Segment Decomposition)

Suppose S and T are defined by

$$S = \oplus/ \cdot f * \cdot segs$$
$$T = \oplus/ \cdot f * \cdot tails$$

If T can be expressed in the form $T = h \cdot \odot \not\rightarrow_e$, then we have

$$S = \oplus/ \cdot h * \cdot \odot \#\!\!\!\not\rightarrow_e$$