Chapter 22. Bird Meertens Formalism (BMF)
A Quick Tour

Zhenjiang Hu, Wei Zhang

School of Computer Science
Peking University
Email: huzj@pku.edu.cn

December 6, 2023
Outline

1. Running Example: Maximum Segment Sum Problem
2. Bird Meertens Formalism
We will explain the basic concepts of BMF by demonstrating how to develop a correct linear-time program.

### Maximum Segment Sum Problem
Given a list of numbers, find the maximum of sums of all consecutive sublists.

- $[-1, 3, 3, -4, -1, 4, 2, -1] \implies 7$
- $[-1, 3, 1, -4, -1, 4, 2, -1] \implies 6$
- $[-1, 3, 1, -4, -1, 1, 2, -1] \implies 4$
Outline

1. Running Example: Maximum Segment Sum Problem

2. Bird Meertens Formalism
   - Review: Functions and Lists
   - Structured Recursive Computation Patterns
   - Horner’s Rule
   - Application
BMF is a calculus of functions for *people* to derive programs from specifications:

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

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’s rule.
Question

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 \textbf{Horner’s rule}.

- How many \(\times\) are used in each side?
Question

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’s 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 \( \oplus \)?
Question

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’s 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?
Review: Functions

- A **function** $f$ that has source type $\alpha$ and target type $\beta$ is denoted by
  \[ f : \alpha \rightarrow \beta \]
  
  We shall say that $f$ takes arguments in $\alpha$ and returns results in $\beta$.

- **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

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

• Two functions $f$ and $g$ are **equivalence** iff

$$\forall x. f x = g x$$
Function composition is denoted by a centralized dot (\( \cdot \)). We have
\[
(f \cdot g) \, x = f \, (g \, x)
\]

Two functions \( f \) and \( g \) are equivalence iff
\[
\forall x. \ f \, x = g \, x
\]

Exercise
Show the following equation states that functional composition is associative.
\[
(f \cdot) \, (g \cdot) = ((f \cdot g) \cdot)
\]
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:

\[
\begin{align*}
(\oplus) \ a \ b &= a \oplus b \\
(a \oplus) \ b &= a \oplus b \\
(\oplus b) \ a &= a \oplus b
\end{align*}
\]
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:

\[
\begin{align*}
(\oplus) \ a \ b &= a \oplus b \\
(a \oplus) \ b &= a \oplus b \\
(\oplus b) \ a &= a \oplus b
\end{align*}
\]

**Exercise**

If $\oplus$ has type $\oplus : \alpha \times \beta \rightarrow \gamma$, then what are the types for $(\oplus)$, $(a \oplus)$ and $(\oplus b)$ for all $a$ in $\alpha$ and $b$ in $\beta$?
The identity element of $\oplus : \alpha \times \alpha \rightarrow \alpha$, if it exists, will be denoted by $id_\oplus$. Thus,

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

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

$$K a b = a$$
The identity element of $\oplus : \alpha \times \alpha \rightarrow \alpha$, if it exists, will be denoted by $id_\oplus$. Thus,

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

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

$$K a b = a$$

**Exercise**

What is the identity element of functional composition?
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 \(\alpha\).

Examples:
\[
[1, 2, 1] : [Int]
[[1], [1, 2], [1, 2, 1]] : [[Int]]
[] : [\alpha]
\]
List Constructors

- [] : [α] constructs an empty list.
- . : α → [α] 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

- \(([α], +, [])\) is a **monoid**.
- \(([α], +, [])\) is a **free monoid** generated by \(α\) under the assignment \([.] : α \rightarrow [α]\).
- \(([α]^+, +)\) is a **semigroup**.
List Functions: Homomorphisms

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

\[
\begin{align*}
h \, [] &= \text{id} \\
h \, [a] &= f \, a \\
h \, (x \, ++ \, y) &= h \, x \, \oplus \, h \, y
\end{align*}
\]

It defines a map from the monoid $(\alpha, +, [])$ to the monoid $(\beta, \oplus : \beta \to \beta \to \beta, \text{id}_\oplus : \beta)$. 

Zhenjiang Hu, Wei Zhang

Chapter 22. Bird Meertens Formalism (BMF)
A function $h$ defined in the following form is called **homomorphism**:

\[
\begin{align*}
  h \, [] & = \, \text{id}_\oplus \\
  h \, [a] & = \, f \, a \\
  h \, (x \oplus y) & = \, h \, x \oplus h \, y
\end{align*}
\]

It defines a map from the monoid $([\alpha], \oplus, [])$ to the monoid $(\beta, \oplus : \beta \to \beta \to \beta, \text{id}_\oplus : \beta)$.

Property: $h$ is **uniquely** determined by $f$ and $\oplus$. 
An Example: the function returning the length of a list.

\[
\begin{align*}
\# \ [\ ] & = 0 \\
\# \ [a] & = 1 \\
\# \ (x + y) & = \# \ x + \# \ y
\end{align*}
\]

Note that \((\mathbb{Int}, +, 0)\) 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 \( \ast \) (pronounced \textit{map}) takes a function on its left and a list on its right. Informally, we have

\[
f \ast [a_1, a_2, \ldots, a_n] = [f a_1, f a_2, \ldots, f a_n]
\]

Formally, \((f \ast)\) (or sometimes simply written as \(f \ast\)) is a homomorphism:

\[
f \ast (x + y) = (f \ast x) + (f \ast y)
\]
The operator \( \ast \) (pronounced map) takes a function on its left and a list on its right. Informally, we have

\[
f \ast [a_1, a_2, \ldots, a_n] = [f(a_1), f(a_2), \ldots, f(a_n)]
\]

Formally, \((f\ast)\) (or sometimes simply written as \(f\ast\)) is a homomorphism:

\[
\begin{align*}
f \ast [] &= [] \\
f \ast [a] &= [f(a)] \\
f \ast (x \ast y) &= (f \ast x) \ast (f \ast y)
\end{align*}
\]
Map

The operator \(*\) (pronounced \textit{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:

\[
egin{align*}
    f* [] &= [] \\
    f* [a] &= [f \, a] \\
    f* (x ++ y) &= (f* x) ++ (f* y)
\end{align*}
\]

Exercise

Prove the following \textit{map distributivity}.

\[
(f \cdot g)* = (f*) \cdot (g*)
\]
The operator $\oplus/ (\text{pronounced } \textit{reduce})$ 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:
Reduce

The operator \(/\) (pronounced \textit{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{align*}
\oplus/\[\] &= id_\oplus \quad \{ \text{if } id_\oplus \text{ exists } \} \\
\oplus/\[a\] &= a \\
\oplus/(x \mathbin{++} y) &= (\oplus/x) \oplus (\oplus/y)
\end{align*}
\]
Examples:

\[
\begin{align*}
\text{max} & : \ [\text{Int}] \to \text{Int} \\
\text{max} &= \uparrow/ \\
\text{where } a \uparrow b &= \text{if } a \leq b \text{ then } b \text{ else } a \\

\text{head} & : \ [\alpha]^+ \to \alpha \\
\text{head} &= \leftarrow/ \\
\text{where } a \leftarrow b &= a \\

\text{last} & : \ [\alpha]^+ \to \alpha \\
\text{last} &= \rightarrow/ \\
\text{where } a \rightarrow b &= b
\end{align*}
\]
$f^*$ and $\oplus/$ can be expressed as identities between functions.

Empty Rules

\[
\begin{align*}
  f^* \cdot K [] &= K [] \\
  \oplus/ \cdot K [] &= K \text{id}_{\oplus}
\end{align*}
\]

One-Point Rules

\[
\begin{align*}
  f^* \cdot [.] &= [.] \cdot f \\
  \oplus/ \cdot [.] &= \text{id}
\end{align*}
\]

Join Rules

\[
\begin{align*}
  f^* \cdot \oplus/ & = \oplus/ \cdot (f^*)^* \\
  \oplus/ \cdot \oplus/ & = \oplus/ \cdot (\oplus/)^*
\end{align*}
\]
Exercise

Any homomorphism $h$ can be defined in the following form:

$$h = \oplus \cdot f^*$$

for some functions $\oplus$ and $f$. 
Composition of two specific homomorphisms is a homomorphism.

\[
\oplus/ \cdot f^* \cdot ++ / \cdot g^*
\]

\[
= \{ \text{map promotion} \}
\]

\[
\oplus/ \cdot ++ / \cdot f^{**} \cdot g^*
\]

\[
= \{ \text{reduce promotion} \}
\]

\[
\oplus/ \cdot (\oplus/) \cdot f^{**} \cdot g^*
\]

\[
= \{ \text{map distribution} \}
\]

\[
\oplus/ \cdot (\oplus/ \cdot f^* \cdot g)^*
\]
Directed Reductions

We introduce two more computation patterns $\rightarrow$ (pronounced \textit{left-to-right reduce}) and $\leftarrow$ (\textit{right-to-left reduce}) which are closely related to $\div$. Informally, we have

\[
\oplus \rightarrow_e [a_1, a_2, \ldots, a_n] = ((e \oplus a_1) \oplus \cdots) \oplus a_n
\]
\[
\oplus \leftarrow_e [a_1, a_2, \ldots, a_n] = a_1 \oplus (a_2 \oplus \cdots \oplus (a_n \oplus e))
\]

Formally, we can define $\oplus \rightarrow_e$ on lists by two equations.

\[
\oplus \rightarrow_e [] = e
\]
\[
\oplus \rightarrow_e (x \mathbin{++} [a]) = (\oplus \rightarrow_e x) \oplus a
\]

Zhenjiang Hu, Wei Zhang

Chapter 22. Bird Meertens Formalism (BMF)
Directed Reductions

We introduce two more computation patterns $\rightarrow$ (pronounced left-to-right reduce) and $\leftarrow$ (right-to-left reduce) which are closely related to /. Informally, we have

$$\oplus \rightarrow_e [a_1, a_2, \ldots, a_n] = ((e \oplus a_1) \oplus \cdots) \oplus a_n$$
$$\oplus \leftarrow_e [a_1, a_2, \ldots, a_n] = a_1 \oplus (a_2 \oplus \cdots \oplus (a_n \oplus e))$$

Formally, we can define $\oplus \rightarrow_e$ on lists by two equations.

$$\oplus \rightarrow_e [] = e$$
$$\oplus \rightarrow_e (x ++ [a]) = (\oplus \rightarrow_e x) \oplus a$$

**Exercise:** Give a formal definition for $\oplus \leftarrow_e$. 
Directed Reductions without Seeds

\[ \oplus \rightarrow [a_1, a_2, \ldots, a_n] = ((a_1 \oplus a_2) \oplus \cdots) \oplus a_n \]

\[ \oplus \leftarrow [a_1, a_2, \ldots, a_n] = a_1 \oplus (a_2 \oplus \cdots \oplus (a_{n-1} \oplus a_n)) \]
Directed Reductions without Seeds

\[ \oplus \not\to [a_1, a_2, \ldots, a_n] = ((a_1 \oplus a_2) \oplus \cdots) \oplus a_n \]
\[ \oplus \not\leftarrow [a_1, a_2, \ldots, a_n] = a_1 \oplus (a_2 \oplus \cdots \oplus (a_{n-1} \oplus a_n)) \]

Properties:

\[ (\oplus \not\to) \cdot ([a] \not\leftarrow) = \oplus \not\to a \]
\[ (\oplus \not\leftarrow) \cdot (\not\leftarrow [a]) = \oplus \not\leftarrow a \]

Zhenjiang Hu, Wei Zhang
Chapter 22. Bird Meertens Formalism (BMF)
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:

\[\otimes \rightarrow_1 [a_1, a_2, \ldots, a_n]\]
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:

\[\circ \rightarrow_{1}[a_1, a_2, \ldots, a_n]\]

where \[a \circ b = (a \times b) + 1\]
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:

\[\odot \rightarrow_1 [a_1, a_2, \ldots, a_n]\]

where \(a \odot b = (a \times b) + 1\)

Exercise

Give the definition of \(\ominus\) such that the following holds.

\[\ominus \rightarrow [a_1, a_2, \ldots, a_n] = (((a_1 \times a_2 + a_2) \times a_3 + a_3) \times \cdots + a_{n-1}) \times a_n + a_n\]
The Special Homework Problem

Suppose \( f = \oplus \mapsto e = \otimes \leftarrow e \).

1. Prove that \( f \) is a homomorphism, i.e., there exists an associate operator \( \odot \) s.t.

\[
f(x \oplus y) = f(x)s \odot f(y)s.
\]

2. Implement in Haskell an algorithm to derive \( \odot \) from \( \oplus \) and \( \otimes \).
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 $\oplus\,\rangle\,\langle\,$ (pronounced left-accumulate) and $\langle\,\rangle\,\langle\,$ (right-accumulate) and are defined informally by

\[
\begin{align*}
\oplus\,\rangle\,\langle\, e[a_1, a_2, \ldots, a_n] &= \left[ e, e \oplus a_1, \ldots, ((e \oplus a_1) \oplus) \cdots \oplus a_n \right] \\
\langle\,\rangle\,\langle\, e[a_1, a_2, \ldots, a_n] &= \left[ a_1 \oplus (a_2 \oplus \cdots \oplus (a_n \oplus e)), \ldots, a_n \oplus e, e \right]
\end{align*}
\]
Formally, we can define $\oplus \triangleleft e$ on lists by two equations by

$$
\begin{align*}
\oplus \triangleleft e[] &= [e] \\
\oplus \triangleleft e([a] ++ x) &= [e] ++ (\oplus \triangleleft e \oplus a \times),
\end{align*}
$$

or

$$
\begin{align*}
\oplus \triangleleft e[] &= [e] \\
\oplus \triangleleft e(x ++ [a]) &= (\oplus \triangleleft e x) ++ [b \oplus a] \\
&\text{where } b = last(\oplus \triangleleft e x).
\end{align*}
$$
$$\bigoplus_{e} [a_1, a_2, \ldots, a_n]$$: can be evaluated with $n - 1$ calculations of $\bigoplus$.

**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. $\times \bigwedge_1 [1, 2, \ldots, n]$

2. $\text{fact} \times [0, 1, 2, \ldots, n]$ where $\text{fact} n = \text{product} [1..n]$. 
Relation between Reduce and Accumulate

\[ \oplus \to e = \text{last} \cdot \oplus \to / e \]

\[ \oplus \to / e = \otimes \to [e] \]

where \( x \otimes a = x \oplus [\text{last} \cdot x \oplus a] \)
Segments

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

$$x = u \leftrightarrow y \leftrightarrow v.$$ 

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 $[a_1, a_2, \ldots, a_n]$?
The function \textit{inits} returns the list of initial segments of a list, in increasing order of a list.

\[ \text{inits} \left[ a_1, a_2, \ldots, a_n \right] = [\emptyset, [a_1], [a_1, a_2], \ldots, [a_1, a_2, \ldots, a_n]] \]
The function \textit{inits} returns the list of initial segments of a list, in increasing order of a list.

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

\[
\text{inits} = (\# \#\emptyset) \cdot [\cdot]^*
\]
The function \texttt{tails} returns the list of final segments of a list, in decreasing order of a list.

\[
n\text{tails} [a_1, a_2, \ldots, a_n] = [[a_1, a_2, \ldots, a_n], [a_2, \ldots, a_n], \ldots, [a_n], [\,]]
\]
The function **tails** returns the list of final segments of a list, in decreasing order of a list.

\[
tails [a_1, a_2, \ldots, a_n] = [[a_1, a_2, \ldots, a_n], [a_2, \ldots, a_n], \ldots, [a_n], []]
\]

\[
tails = (+ \leftarrow []) \cdot [\cdot]^*
\]
Running Example: Maximum Segment Sum Problem
Bird Meertens Formalism

Review: Functions and Lists
Structured Recursive Computation Patterns
Horner’s Rule
Application

segs

segs = + + \cdot \text{tails} \ast \cdot \text{inits}

Exercise: Show the result of segs [1, 2].
Accumulation Lemma

\[(\oplus \# e) = (\oplus \rightarrow e) \cdot \text{inits}\]
\[(\oplus \#) = (\oplus \rightarrow) \cdot \text{inits}^+\]

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 Question: 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’s 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

\[ \oplus / \cdot \otimes / * \cdot \text{tails} = \odot \rightarrow e \]

where

\[ e = \text{id} \otimes \]
\[ a \odot b = (a \otimes b) \oplus e \]

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 \( c \).
Homework BMF 1-1

Prove the correctness of the Horner’s rule.

- Show that

\[(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)\]

is equivalent to

\[(\otimes c) \cdot \oplus/ = \oplus/ \cdot (\otimes c) \ast .\]

holds on all non-empty lists.

- Show that

\[f = \oplus/ \cdot \otimes/ \ast \cdot \text{tails}\]

satisfies the equations

\[f[\,] = e\]
\[f(x \text{ ++ } [a]) = f \times \otimes a\]
Generalizations of Horner’s Rule

Generalization 1:

\[ \oplus / \cdot \otimes / \ast \cdot \text{tails}^+ = \otimes / \ast \]

where

\[ a \odot b = (a \otimes b) \oplus b \]
Generalizations of Horner’s Rule

Generalization 1:

\[ \oplus/ \cdot \otimes/ \ast \cdot \text{tails}^+ = \odot \rightarrow \]

where

\[ a \odot b = (a \otimes b) \oplus b \]

Generalization 2:

\[ \oplus/ \cdot \left( \otimes/ \cdot f^* \right) \ast \cdot \text{tails} = \odot \rightarrow e \]

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

\[ mss = \uparrow / \cdot + / \ast \cdot \text{segs} \]
A Direct Solution

\[ mss = \uparrow / \cdot + / \ast \cdot \text{segs}. \]

Exercise

Write a Haskell program for this direct solution.
Calculating a Linear Algorithm using Horner’s Rule

\[ \text{mss} \]
\[ \begin{align*}
\text{mss} & \quad \{ \text{definition of mss} \} \\
& \uparrow/ \cdot +/ \ast \cdot \text{segs} \\
& = \quad \{ \text{definition of segs} \} \\
& \uparrow/ \cdot +/ \ast \cdot ++/ \cdot \text{tails} \ast \cdot \text{inits} \\
& \quad \{ \text{map and reduce promotion} \} \\
& \uparrow/ \cdot (\uparrow/ \cdot +/ \ast \cdot \text{tails}) \ast \cdot \text{inits} \\
& = \quad \{ \text{Horner’s rule with } a \circ b = (a + b) \uparrow 0 \} \\
& \uparrow/ \cdot \circ \# 0 \ast \cdot \text{inits} \\
& \quad \{ \text{accumulation lemma} \} \\
& \uparrow/ \cdot \circ \# 0
\]
Homework BMF 1-2

Code the derived linear algorithm for $mss$ in Haskell.
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

$$S = \oplus/ \cdot f \ast \cdot \text{segs}$$
$$T = \oplus/ \cdot f \ast \cdot \text{tails}$$

If $T$ can be expressed in the form $T = h \cdot \circ \rightarrow_e$, then we have

$$S = \oplus/ \cdot h \ast \cdot \circ \rightarrow_e$$
Homework BMF 1-3

Prove the segment decomposition theorem.