構成的アルゴリズム論の基本概念

胡 振江
東京大学 計数工学科

2006 年度

Constructive Algorithmics

Zhenjiang HU
University of Tokyo

The First Exercise Revisited

The Maximum Segment Sum (mss) Problem

Try you best to design an *efficient* and *correct* program to compute the maximum of the sums of all segments of a given sequence of numbers, positive, negative, or zero.

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

Reference

R.S. Bird: *Lecture Notes on Constructive Functional Programming*, *Technical Monograph PRG-69*, ISBN 0-902928-51-1, 1988.

Subject

A *calculus* of functions for deriving programs from their specifications:

- A range of concepts and notations for defining *functions* over various data types (including lists, trees, and arrays);
- A set of *algebraic laws* (rules, lemmas, theorems) for manipulating functions;
- A framework for *constructing new calculation rules* to capture principles of programming.

Outline

- *Basic Concepts*
- *Deriving Programs for Manipulating Lists*
  - Homomorphism: Join Lists
  - Left Reduction: Snoc Lists
- *Deriving Programs for Manipulating Arrays (Matrices)*
- *Deriving Programs for Manipulating Trees*
- *Categorical Aspects of Constructive Algorithmics*

Basic Concepts

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?

Review: Notations on Functions

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

$$f : \alpha \to \beta$$

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

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

- 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:

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

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?

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

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

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

$$a \oplus id_{\oplus} = id_{\oplus} \oplus a = a$$

Exericise: What is the identity element of functional composition?

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

$$K\ a\ b = a$$

Review: 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:
    $[1, 2, 1] : [Int]$
    $[[1], [1, 2], [1, 2, 1]] : [[Int]]$
    $[\,] : [\alpha]$

- $[.] : \alpha \to [\alpha]$ maps elements of α into singleton lists.

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

- The primitive operator on lists is *concatenation*, denoted by $+\!\!+$.

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

Concatenation is associative:

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

Exercise: What is the identity for concatenation?

- **Algebraic View of Lists**:
  - $([\alpha], +\!\!+, [\,])$ is a *monoid*.
  - $([\alpha], +\!\!+, [\,])$ is a *free monoid* generated by α under the assignment $[.] : \alpha \rightarrow [\alpha]$.
  - $([\alpha]^+, +\!\!+)$ is a *semigroup*.

List Functions: Homomorphisms

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

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

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

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 $(Int, +, 0)$ 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 lts 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: $(f \cdot g)* = (f*) \cdot (g*)$

Exercise: Prove the map distributivity.

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:

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

If $\oplus$ is commutative as well as associative, then $\oplus/$ can be applied to bags; and if $\oplus$ is also idempotent, then $\oplus/$ can be applied to sets.

Examples:

$$
\begin{aligned}
max &: [Int] \to Int \\
max &= \uparrow / \\
&\quad \text{where } a \uparrow b = \text{if } a \le b \text{ then } b \text{ else } a \\
head &: [\alpha]^{+} \to \alpha \\
head &= \lessdot / \\
&\quad \text{where } a \lessdot b = a \\
last &: [\alpha]^{+} \to \alpha \\
last &= \gtrdot / \\
&\quad \text{where } a \gtrdot b = b
\end{aligned}
$$

Promotion

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

Empty Rules

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

One-Point Rules

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

Join Rules

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

Exercise: Prove the join rules.

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

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

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

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

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

Directed Reductions without Seeds

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

Properties:

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

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:

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

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 $\leftarrow\!\!\!\!\not\!\!/$ (*right-accumulate*) and are defined informally by

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

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

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

or

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

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. $\mathit{fact} * [0, 1, 2, \cdots, n]$ where $\mathit{fact}\ 0 = 1$ and $\mathit{fact}\ k = 1 \times 2 \times \cdots \times k$.

Relation between Reduce and Accumulate

$$\oplus \not\to_e = last \cdot \oplus \not\!\!\to_e$$

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

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

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:

$$segs\ [1,2,3] = [[],[1],[1,2],[2],[1,2,3],[2,3],[3]]$$

Exercise: List all initial segments and final segments in the above example.

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.

$$inits\ [a_1, a_2, \ldots, a_n] = [[\,], [a_1], [a_1, a_2], \ldots, [a_1, a_2, \ldots, a_n]]$$

$$inits = (\mathbin{+\!\!+} \not\!\!\!\!\!\!\twoheadrightarrow_{[]}) \cdot [\cdot]*$$

tails

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

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

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

segs

$$segs = +\!\!+ / \cdot tails * \cdot inits$$

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

Accumulation Lemma

$$
\begin{aligned}
(\oplus \mathbin{/\!\!\!/\!\!\!\rightarrow}_e) &= (\oplus \not\rightarrow_e) * \cdot \mathit{inits} \\
(\oplus \mathbin{/\!\!\!/\!\!\!\rightarrow}) &= (\oplus \not\rightarrow) * \cdot \mathit{inits}^+
\end{aligned}
$$

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

$$\oplus/ \cdot \otimes/ * \cdot tails = \odot \not\rightarrow_e$$
$$\text{where}$$
$$e = id_{\otimes}$$
$$a \odot b = (a \otimes b) \oplus e$$

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.

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

  $$(\otimes c) \cdot \oplus/ = \oplus/ \cdot (\otimes c)*$$

  holds on all non-empty lists.

- Show that

  $$f = \oplus/ \cdot \otimes/ * \cdot \mathit{tails}$$

  satisfies the equations

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

Generalizations of Horner's Rule

Generalization 1:

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

where

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

Generalization 2:

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

where

$$e = id_{\otimes}$$

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

Application

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

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

Exercise: How many steps are required in the above direct solution?

Calculating a Linear Algorithm using Horner's Rule

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

A Program in Haskell

```
mss = foldl1 (max) . scanl odot 0
  where a `odot` b = (a + b) `max` 0
```

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 grequently. The essential idea can be summarized as a general theorem.

Theorem 1 (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*

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

Exercise: Prove the segment decomposition theorem.