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

2007 年 12 月 10 日、17 日

The First Exercise Revisited

The Maximum Segment Sum (mss) Problem

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.

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

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

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

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

A Simple 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
$$f : \alpha \rightarrow \beta$$
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
$$(f \cdot g)\ x = f(g\ x)$$

Review: Notations on Functions

- 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: Given $(\oplus) : \alpha \rightarrow \beta \rightarrow \gamma$, give the types for $(a\oplus)$ and $(\oplus b)$?

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

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

Review: Notations on Functions

- 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

  $$K\ a\ b = a$$

Review: Lists

Lists are finite sequence of values of the same type. We use $[\alpha]$ to denote the type of lists whose elements have type α, and $[\alpha]^+$ to denote the type of non-empty lists whose elements have type α.

- Examples:
  $[1, 2, 1] : [\mathit{Int}]$
  $[[1], [1, 2], [1, 2, 1]] : [[\mathit{Int}]]$
  $[\,] : [\alpha]$

Review: Lists

- $[.] : \alpha \rightarrow [\alpha]$ 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 $[.] : \alpha \to [\alpha]$.
- $([\alpha]^+, +\!\!+)$ is a semigroup.

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):

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

Outline

List Functions as 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 structure-preserving map from the monoid $([\alpha], +\!\!+, [\,])$ to the monoid $(\beta, \oplus\colon \beta \to \beta \to \beta, id_{\oplus}\colon \beta)$.

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

List Functions as Homomorphisms

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

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

It is a structure-preserving map from the monoid $([\alpha], \mathbin{+\!\!+}, [\,])$ the monoid $(Int, +, 0)$.

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

Old 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:

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

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.

Reduce

Examples:

$$
\begin{array}{lcl}
max & : & [Int] \to Int \\
max & = & \uparrow / \\
 & & \text{where } a \uparrow b = \text{if } a \leq b \text{ then } b \text{ else } a
\end{array}
$$

$$
\begin{array}{lcl}
sum & : & [Int] \to Int \\
sum & = & +/
\end{array}
$$

$$
\begin{array}{lcl}
head & : & [\alpha]^+ \to \alpha \\
head & = & \lessdot / \quad \text{where } a \lessdot b = a
\end{array}
$$

$$
\begin{array}{lcl}
last & : & [\alpha]^+ \to \alpha \\
last & = & \gtrdot / \quad \text{where } a \gtrdot b = b
\end{array}
$$

The Homomorphism Lemma

The Homomorphism Lemma

A list function $h : [A] \to B$ is a homomorphism if and only if there exist f and $\oplus$ such that the following holds.

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

Proof

It suffices to prove that $\oplus / \cdot f*$ is a homomorphism to $(B, \oplus, id_{\oplus})$ with f on a singleton list, because of the uniqueness property of homomorphisms.

Promotion Rules

The equations defining $f*$ 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

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

Exercise: Prove the join rules.

An Example of Calculation

A composition of two specific homomorphisms is a homomorphism.

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

Outline

Directed Reductions (Folds)

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

$$
\begin{array}{lcl}
\oplus \not\rightarrow_e [a_1, a_2, \ldots, a_n] & = & (((e \oplus a_1) \oplus \cdots) \oplus a_{n-1}) \oplus a_n \\
\oplus \not\leftarrow_e [a_1, a_2, \ldots, a_n] & = & a_1 \oplus (a_2 \oplus (\cdots \oplus (a_n \oplus e)))
\end{array}
$$

Formally, we can define them as follows.

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

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 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 expressed by 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 a definition of $\ominus$ such that the following holds.

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

Outline

Accumulations (Scans)

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 accumulate) 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 them as follows.

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

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

Efficiency in Accumulations

$\oplus \not\!\!\!\!/\!\!\rightarrow_e [a_1, a_2, \ldots, a_n]$: 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

$$\begin{array}{lcl} \mathit{fact}\ 0 & = & 1 \\ \mathit{fact}\ (k+1) & = & k \times \mathit{fact}\ k. \end{array}$$

Relation between Reduce and Accumulate

$$\oplus \not\rightarrow_{e} = \mathit{last} \cdot \oplus \not\!\not\rightarrow_{e}$$

$$\oplus \not\!\not\rightarrow_{e} = \otimes \not\rightarrow_{[e]}$$
$$\text{where } x \otimes a = x +\!\!+ [\mathit{last}\ x \oplus a]$$

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.

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

Exercise: List all initial segments and final segments of $[1,2,3]$.
Exercise: How many segments of $[a_1, a_2, \ldots, a_n]$?

inits

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

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

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

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

$$(\oplus /\!\!/\!\!\!\rightarrow_e) = (\oplus \not\rightarrow_e) * \cdot inits$$

$$(\oplus /\!\!/\!\!\!\rightarrow) = (\oplus \not\rightarrow) * \cdot inits^+$$

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.

Outline

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

$$\oplus/ \cdot \otimes/ * \cdot tails = \odot \not\rightarrow_e$$

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.

Proof of Horner's Rule

The Horner's rule can be proved by the following two steps.

- Show that
$$(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$$
if and only if
$$(\otimes c) \cdot \oplus/ = \oplus/ \cdot (\otimes c) * .$$

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

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

Generalizations of Horner's Rule

Generalization 1:

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

where

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

Generalization 2:

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

where

$$e = id_{\otimes}$$

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

Outline

Application: MSS

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.

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

A Direct Solution

$$\mathit{mss} = \uparrow / \cdot +/ * \cdot \mathit{segs}$$

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

Calculating a Linear Algorithm

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

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 Theorem

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.

Segment Decomposition Theorem

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

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

Exercise: Prove the segment decomposition theorem.