Maximum Segment Sum, Monadically

Jeremy Gibbons
University of Oxford
WG2.1, May 2011

1. Maximum segment sum

$$mss \quad = \quad maximum \cdot map\, sum \cdot segs$$

where *segs* computes the contiguous segments of a list:

$$\begin{array}{lcl} segs & = & concat \cdot map\, inits \cdot tails \\ tails & = & foldr\, f\, [[\,]] \quad \textbf{where}\ f\, x\, xss = (x : head\, xss) : xss \\ inits & = & foldr\, g\, [[\,]] \quad \textbf{where}\ g\, x\, xss = [\,] : map\, (x :)\, xss \end{array}$$

and *sum* computes the sum of a list:

$$sum \quad = \quad foldr\, (+)\, 0$$

and *maximum* the maximum of a nonempty list:

$$maximum \quad = \quad foldr_1\, max$$

1.1. Calculation using standard properties

$$
\begin{array}{ll}
 & mss \\
= & \quad \{\text{definition of } mss\} \\
 & maximum \cdot map\, sum \cdot segs \\
= & \quad \{\text{definition of } segs\} \\
 & maximum \cdot map\, sum \cdot concat \cdot map\, inits \cdot tails \\
= & \quad \{\text{naturality of } concat\} \\
 & maximum \cdot concat \cdot map\,(map\, sum) \cdot map\, inits \cdot tails \\
= & \quad \{maximum \cdot concat = maximum \cdot map\, maximum\} \\
 & maximum \cdot map\, maximum \cdot map\,(map\, sum) \cdot map\, inits \cdot tails \\
= & \quad \{\text{functors}\} \\
 & maximum \cdot map\,(maximum \cdot map\, sum \cdot inits) \cdot tails
\end{array}
$$

Then *Scan Lemma*: write $maximum \cdot map\, sum \cdot inits$ in the form $foldr\, h\, e$.

1.2. Horner's Rule

Classically,

$$\sum_{i=0}^{n-1} a_i x^i = a_0 + x \times (a_1 + x \times (a_2 + \cdots + x \times a_{n-1}))$$

In Haskell:

$$sum \cdot map\, product \cdot inits = foldr\, (\oplus)\, e$$

where $e = 1$ and $u \oplus z = e + u \times z$

Works for any *semiring*: associative "addition" and "multiplication"; a unit for "multiplication"; "multiplication" distributes over "addition".

Eg for max and $+$:

$$maximum \cdot map\, sum \cdot inits = foldr\, (\oplus)\, e$$

where $e = 0$ and $u \oplus z = e\, `max`\, (u + z)$. Moreover, $\oplus$ takes constant time, so this gives a linear-time algorithm for mss.

2. Datatype-generically

A datatype-generic version of the maximum segment sum problem and solution: Bird, de Moor and Hoogendijk, *Generic functional programming with types and relations*, JFP 1996.

It's clear what "maximum" and "sum" mean generically, but not so clear what "segment" means for nonlinear datatypes; the point of the paper is basically to resolve that issue.

2.1. DGP tail segments

A "tail" of a data structure is just a subterm—of the same type.

The key idea for *tails*: *every node of the original list is labelled with the subterm of the original list rooted at that node*.

Need a "labelled variant" of a datatype, in which *every node has precisely one label*.

Labelled variant of $\mathsf{T}\,\alpha = \mu(\mathsf{F}\,\alpha)$ is $\mathsf{L}\,\alpha = \mu(\mathsf{G}\,\alpha)$, where $\mathsf{G}\,\alpha\,\beta = \alpha \times \mathsf{F}\,1\,\beta$.

Then

$$\mathit{subterms}_{\mathsf{F}} = \mathit{fold}_{\mathsf{F}}(\mathit{in}_{\mathsf{G}} \cdot \mathit{fork}(\mathit{in}_{\mathsf{F}} \cdot \mathsf{F}\,\mathit{id}\,\mathit{root}, \mathsf{F}\,!\,\mathit{id})) :: \mathsf{T}\,\alpha \to \mathsf{L}(\mathsf{T}\,\alpha)$$

where $\mathit{root} = \mathit{fst} \cdot \mathit{in}_{\mathsf{G}}^{-1} = \mathit{fold}_{\mathsf{G}}\,\mathit{fst} :: \mathsf{L}\,\alpha \to \alpha$ returns the root label of a labelled data structure, and $!_{\alpha} :: \alpha \to 1$ is the unique arrow to the unit type.

2.2. DGP Scan Lemma

$$
\begin{array}{lcl}
\mathit{scan}_{\mathsf{F}} & :: & (\mathsf{F}\,\alpha\,\beta \to \beta) \to \mathsf{T}\,\alpha \to \mathsf{L}\,\beta \\
\mathit{scan}_{\mathsf{F}}\,\phi & = & \mathsf{L}\,(\mathit{fold}_{\mathsf{F}}\,\phi) \cdot \mathit{subterms}_{\mathsf{F}} \\
 & = & \mathit{fold}_{\mathsf{F}}(\mathit{in}_{\mathsf{G}} \cdot \mathit{fork}(\phi \cdot \mathsf{F}\,\mathit{id}\,\mathit{root}, \mathsf{F}\,!\,\mathit{id}))
\end{array}
$$

The label for each node can be constructed from the root labels of each of the child trees.

2.3. DGP initial segment

An "initial segment" of a data structure is obtained by replacing some subterms by the empty structure.

For arbitrary F, define $\mathsf{U}\,\alpha = \mu(\mathsf{H}\,\alpha)$ where $\mathsf{H}\,\alpha\,\beta = 1 + \mathsf{F}\,\alpha\,\beta$.

(Bird &co insist that T supports empty structures: $\mathsf{F}\,\alpha\,\beta = 1 + \mathsf{F}'\,\alpha\,\beta$ for some F'. This isn't necessary.)

2.4. DGP initial segments

For lists, *inits* returned another list: there are as many initial segments as there are tail segments. That's not true generically.

We'll just use a collection monad M: with *MonadPlus*-like structure, via $\mathit{mplus} :: \mathsf{M}\,\alpha \times \mathsf{M}\,\alpha \to \mathsf{M}\,\alpha$. However, no *mzero* yielding an empty collection.

Then

$$\mathit{prune} = \mathit{fold}_{\mathsf{F}}(\mathsf{M}\,\mathit{in}_{\mathsf{H}} \cdot \mathit{opt}\,\mathit{Nothing} \cdot \mathsf{M}\,\mathit{Just} \cdot \delta_2) :: \mathsf{T}\,\alpha \to \mathsf{M}(\mathsf{U}\,\alpha)$$

Here, *opt* supplies a new alternative for a nondeterministic computation:

$$\mathit{opt}\,a\,mx = \mathit{return}\,a\,`\mathit{mplus}`\,mx$$

and $\delta_2 :: (\mathsf{F}\,\alpha)\mathsf{M} \dot{\to} \mathsf{M}(\mathsf{F}\,\alpha)$ distributes shape functor F over monad M.

2.5. DGP distributivity

- $(\mathsf{F}\,\alpha)$-algebra (β, f) ("DGP product")
- M-algebra (β, k) ("collection sum")

such that

$$\begin{tikzcd}
\mathsf{F}\,\alpha\,(\mathsf{M}\,\beta) \arrow[r, "\delta_2"] \arrow[d, "\mathsf{F}\,id\,k"'] & \mathsf{M}(\mathsf{F}\,\alpha\,\beta) \arrow[r, "\mathsf{M}\,f"] & \mathsf{M}\,\beta \arrow[d, "k"] \\
\mathsf{F}\,\alpha\,\beta \arrow[rr, "f"'] & & \beta
\end{tikzcd}$$

For example, if $f :: \mathsf{F}\,\mathbb{Z}\,\mathbb{Z} \to \mathbb{Z}$ adds all the integers in an F-structure, and $k :: \mathsf{M}\,\mathbb{Z} \to \mathbb{Z}$ finds the maximum of a non-empty integer collection, then f distributes over k.

2.6. Algebras for a MonadPlus

Note that (β, k) is an algebra for the monad M:

- $k :: \mathsf{M}\,\beta \to \beta$
- $k \cdot return = id$
- $k \cdot join = k \cdot \mathsf{M}\,k$

We will stick to "reductions": $k = \oplus/$ for associative binary operator $\oplus :: \beta \times \beta \to \beta$; then we also have distribution over choice

$$\oplus/(x \text{`}mplus\text{`}\ y) = (\oplus/x) \oplus (\oplus/y)$$

No empty collections in M, so we do not need unit for $\oplus$.

2.7. DGP Horner's Rule

We also need the "product" b of an empty structure. Then let $g = \mathit{maybe}\, b\, f :: \mathsf{H}\, \alpha\, \beta \to \beta$, so that $\mathit{fold}_{\mathsf{H}}(g) :: \mathsf{U}\, \alpha \to \beta$.

Straightforward to show that

$$\oplus/ \cdot \mathsf{M}(\mathit{fold}_{\mathsf{H}}(\mathit{maybe}\, b\, f)) \cdot \mathit{prune} = \mathit{fold}_{\mathsf{F}}((b\oplus) \cdot f)$$

using fold fusion.

(Curiously, it doesn't seem to matter what value is chosen for b.)

2.8. The fusion condition

$$
\begin{array}{ll}
 & \oplus/ \cdot \mathsf{M}(\mathit{fold}_{\mathsf{H}}(g)) \cdot \mathit{prune} \cdot \mathit{in}_{\mathsf{F}} \\
= & \quad \{\text{evaluation for } \mathit{prune}\text{; functors; evaluation for } \mathit{fold}_{\mathsf{H}}\} \\
 & \oplus/ \cdot \mathsf{M}(g \cdot \mathsf{H}\,\mathit{id}\,(\mathit{fold}_{\mathsf{H}}(g))) \cdot \mathit{opt}\,\mathit{Nothing} \cdot \mathsf{M}\,\mathit{Just} \cdot \delta_2 \cdot \mathsf{F}\,\mathit{id}\,\mathit{prune} \\
= & \quad \{\mathsf{M}\,h \cdot \mathit{opt}\,a = \mathit{opt}\,(h\,a) \cdot \mathsf{M}\,h\} \\
 & \oplus/ \cdot \mathit{opt}\,(g\,\mathit{Nothing}) \cdot \mathsf{M}(g \cdot \mathsf{H}\,\mathit{id}\,(\mathit{fold}_{\mathsf{H}}(g))) \cdot \mathsf{M}\,\mathit{Just} \cdot \delta_2 \cdot \mathsf{F}\,\mathit{id}\,\mathit{prune} \\
= & \quad \{\text{functors; } \mathit{Just} :: \mathsf{F}\,\alpha \dot{\to} \mathsf{H}\,\alpha\} \\
 & \oplus/ \cdot \mathit{opt}\,(g\,\mathit{Nothing}) \cdot \mathsf{M}\,g \cdot \mathsf{M}(\mathit{Just} \cdot \mathsf{F}\,\mathit{id}\,(\mathit{fold}_{\mathsf{H}}(g))) \cdot \delta_2 \cdot \mathsf{F}\,\mathit{id}\,\mathit{prune} \\
= & \quad \{\text{functors; } \delta_2 :: (\mathsf{F}\alpha)\mathsf{M} \dot{\to} \mathsf{M}(\mathsf{F}\alpha)\} \\
 & \oplus/ \cdot \mathit{opt}\,(g\,\mathit{Nothing}) \cdot \mathsf{M}(g \cdot \mathit{Just}) \cdot \delta_2 \cdot \mathsf{F}\,\mathit{id}\,(\mathsf{M}(\mathit{fold}_{\mathsf{H}}(g))) \cdot \mathsf{F}\,\mathit{id}\,\mathit{prune} \\
= & \quad \{g = \mathit{maybe}\,b\,f\} \\
 & \oplus/ \cdot \mathit{opt}\,b \cdot \mathsf{M}\,f \cdot \delta_2 \cdot \mathsf{F}\,\mathit{id}\,(\mathsf{M}(\mathit{fold}_{\mathsf{H}}(g))) \cdot \mathsf{F}\,\mathit{id}\,\mathit{prune} \\
= & \quad \{\oplus/ \cdot \mathit{opt}\,b = (b\oplus) \cdot \oplus/\} \\
 & (b\oplus) \cdot \oplus/ \cdot \mathsf{M}\,f \cdot \delta_2 \cdot \mathsf{F}\,\mathit{id}\,(\mathsf{M}(\mathit{fold}_{\mathsf{H}}(g))) \cdot \mathsf{F}\,\mathit{id}\,\mathit{prune} \\
= & \quad \{\text{distributivity: } \oplus/ \cdot \mathsf{M}\,f \cdot \delta_2 = f \cdot \mathsf{F}\,\mathit{id}\,(\oplus/)\text{; functors}\} \\
 & (b\oplus) \cdot f \cdot \mathsf{F}\,\mathit{id}\,(\oplus/ \cdot \mathsf{M}(\mathit{fold}_{\mathsf{H}}(g)) \cdot \mathit{prune})
\end{array}
$$

2.9. DGP MSS

Given

- traversable shape bifunctor F
- collection monad M
- distributive law $\delta_2 :: \mathsf{F}\,\alpha\,(\mathsf{M}\,\alpha) \to \mathsf{M}(\mathsf{F}\,\alpha\,\alpha)$, hence $\mathit{contents}_{\mathsf{L}} :: \mathsf{L}\,\alpha \to \mathsf{M}\,\alpha$
- $(\mathsf{F}\,\alpha)$-algebra (β, f), and additional element $b :: \beta$
- M-algebra $(\beta, \oplus/)$

such that f and $\oplus$ take constant time, and f distributes over $\oplus/$, then

$$\oplus/ \cdot \mathsf{M}(\mathit{fold}_{\mathsf{H}}(\mathit{maybe}\,b\,f)) \cdot \mathit{segs}$$

can be computed in linear time, where

$$\mathit{segs} = \mathit{join} \cdot \mathsf{M}\,\mathit{prune} \cdot \mathit{contents}_{\mathsf{L}} \cdot \mathit{subterms} :: \mathsf{T}\,\alpha \to \mathsf{M}(\mathsf{U}\,\alpha)$$

2.10. The calculation

$$
\begin{array}{ll}
 & \oplus/ \cdot \mathsf{M}(\mathit{fold}_{\mathsf{H}}(\mathit{maybe}\, b\, f)) \cdot \mathit{segs} \\
= & \quad \{\text{definition of } \mathit{segs}\} \\
 & \oplus/ \cdot \mathsf{M}(\mathit{fold}_{\mathsf{H}}(\mathit{maybe}\, b\, f)) \cdot \mathit{join} \cdot \mathsf{M}\, \mathit{prune} \cdot \mathit{contents}_{\mathsf{L}} \cdot \mathit{subterms} \\
= & \quad \{\text{naturality of } \mathit{join} :: \mathsf{MM} \dot{\to} \mathsf{M}; \text{ functors}\} \\
 & \oplus/ \cdot \mathit{join} \cdot \mathsf{M}(\mathsf{M}(\mathit{fold}_{\mathsf{H}}(\mathit{maybe}\, b\, f)) \cdot \mathit{prune}) \cdot \mathit{contents}_{\mathsf{L}} \cdot \mathit{subterms} \\
= & \quad \{\oplus/ \text{ is an } \mathsf{M}\text{-algebra; functors}\} \\
 & \oplus/ \cdot \mathsf{M}(\oplus/ \cdot \mathsf{M}(\mathit{fold}_{\mathsf{H}}(\mathit{maybe}\, b\, f)) \cdot \mathit{prune}) \cdot \mathit{contents}_{\mathsf{L}} \cdot \mathit{subterms} \\
= & \quad \{\text{naturality of } \mathit{contents} :: \mathsf{L} \dot{\to} \mathsf{M}\} \\
 & \oplus/ \cdot \mathit{contents}_{\mathsf{L}} \cdot \mathsf{L}(\oplus/ \cdot \mathsf{M}(\mathit{fold}_{\mathsf{H}}(\mathit{maybe}\, b\, f)) \cdot \mathit{prune}) \cdot \mathit{subterms} \\
= & \quad \{\text{Horner's Rule}\} \\
 & \oplus/ \cdot \mathit{contents}_{\mathsf{L}} \cdot \mathsf{L}(\mathit{fold}_{\mathsf{F}}((b\oplus) \cdot f)) \cdot \mathit{subterms} \\
= & \quad \{\text{Scan Lemma}\} \\
 & \oplus/ \cdot \mathit{contents}_{\mathsf{L}} \cdot \mathit{scan}_{\mathsf{F}}((b\oplus) \cdot f)
\end{array}
$$

3. Monads versus relations

Bird &co give essentially this calculation, but in terms of relations

$$\mathit{prune} :: \mathsf{T}\,\alpha \rightsquigarrow \mathsf{U}\,\alpha$$

instead of monads

$$\mathit{prune} :: \mathsf{T}\,\alpha \to \mathsf{M}(\mathsf{U}\,\alpha)$$

Relations and set-valued functions isomorphic, so equivalent?

They discuss MSS, of course, but not other semirings—in particular, not $(\mathbb{Z}, +, \times)$. Why not?

We use distribution of $\oplus/$ over choice: $\oplus/(x \text{`}\mathit{mplus}\text{`}\, y) = (\oplus/x) \oplus (\oplus/y)$. For the set monad, this requires $\oplus$ to be idempotent.

Horner's Rule does work for the $(\mathbb{Z}, +, \times)$ semiring in the bag monad, which models collections where multiplicity of results matters.

Relational programming is monadic programming with just one monad!