Generic Sorting By Insertion

Jorge Sousa Pinto

PURe Cafe 11-03-04

1. List Homomorphisms

- "Sorting" is a list homomorphism:

$$\text{isort}\ (l_1 +\!\!+ l_2) = (\text{isort}\ l_1) \odot (\text{isort}\ l_2)$$

- can be calculated leftwards or rightwards:

```
isort = foldr insert []

isort = foldl insert' []
        where insert' l x = insert x l
```

[Bird, 1987]

- note that an associative operator exists s.t.

$$\text{insert } x\ l = [x] \odot l$$

- since this is the unique function that can be calculated in these two ways, we call it simply

$$hom\ (\odot)\ (\backslash x \rightarrow [x])\ [\,]$$

2. Sorting by Insertion

We consider a container type constructor C with operations

$$\mathsf{ist} :: a \to \mathrm{C}\ a \to \mathrm{C}\ a$$

$$\mathsf{tl} :: \mathrm{C}\ a \to [a]$$

verifying equations

$$\mathsf{tl}\ \varepsilon = [\,] \tag{1}$$

$$\mathsf{tl}\ (\mathsf{ist}\ x\ c) = \mathsf{insert}\ x\ (\mathsf{tl}\ c) \tag{2}$$

we define a function

$$\mathsf{gisort} = \mathsf{tl} \circ (\mathsf{foldr}\ \mathsf{ist}\ \varepsilon)$$

this is a sorting algorithm since

$$\mathsf{gisort} = \mathsf{foldr}\ \mathsf{insert}\ [\,]$$

Proof: straightforward application of foldr fusion law

3. Examples of sorting by Insertion

We instantiate tl as follows for node-labeled and leaf-labeled binary trees:

$$\mathsf{tlBTree} = \mathsf{foldBTree}\ (\backslash y\ l\ r \to [y] \odot l \odot r)\ []$$

$$\mathsf{tlLTree} = \mathsf{foldLTree}\ (\odot)\ t\ \text{where}\ t\ \mathsf{Nothing} = [\,]$$
$$t\ (\mathsf{Just}\ x) = [x]$$

using a heap as container:

```
isth :: (Ord a) => a -> Heap a -> Heap a
isth x Empty = Node x Empty Empty
isth x (Node y l r) | x<y = Node x (isth y r) l
                    | otherwise = Node y (isth x r) l

isorth = tlBTree . (foldr isth Empty)
```

using a bst as container:

```
istq :: (Ord a) => a -> BTree a -> BTree a
istq x Empty = Node x Empty Empty
istq x (Node y l r) | x<y = Node y (istq x l) r
                    | otherwise = Node y l (istq x r)

isortq = tlBTree . (foldr istq Empty)
```

using a leaf-tree as container:

```
istm :: (Ord a) => a -> LTree a -> LTree a
istm x (Leaf Nothing)  = Leaf (Just x)
istm x (Leaf (Just y)) = Branch (Leaf (Just x)) (Leaf (Just y))
istm x (Branch l  r)   = Branch (istm x r) l

isortm = tlLTree . (foldr istm (Leaf Nothing))
```

recall equations to be proved:

$$\mathsf{tl}\ \varepsilon = [\,] \qquad (1)$$

$$\mathsf{tl}\ (\mathsf{ist}\ x\ c) = \mathsf{insert}\ x\ (\mathsf{tl}\ c) \qquad (2)$$

(1) is straightforward for these examples

Example proof (outline)

$$\mathsf{tlLTree} \circ (\mathsf{istm}\ x) = (\mathsf{insert}\ x) \circ \mathsf{tlLTree}$$

1. paramorphism fusion on lhs:

$$\mathsf{tlLTree} \circ (\mathsf{istm}\ x) = \mathsf{paraLTree}\ a\ b$$

$$b\ \mathsf{Nothing} = [x]$$
$$b\ (\mathsf{Just}\ y) = [x] \odot [y]$$
$$a\ l\ l'\ r\ r' = r' \odot (\mathsf{tlLTree}\ l)$$

2. express the fold tlLTree as a paramorphism, then apply (strong) paramorphism fusion to prove

$$(\mathsf{insert}\ x) \circ \mathsf{tlLTree} = \mathsf{paraLTree}\ a\ b$$

4. Generic Sorting Accumulations

Specification:

$$\mathsf{gisort}_t :: [a] \to \mathrm{C}\ a \to [a]$$
$$\mathsf{gisort}_t\ l\ y = (\mathsf{isort}\ l) \odot (\mathsf{tl}\ y)$$

can be rewritten as

$$\mathsf{gisort}_t = (\oplus) \circ \mathsf{isort}_t$$

where $s \oplus y = s \odot (\mathsf{tl}\ y)$.

We apply fusion again to get a higher-order fold:

$$\text{foldr ist}' \text{ tl} = (\oplus) \circ (\text{foldr insert } [\,])$$

where

$$\text{ist}' \; x \; f \; y = f \; (\text{ist} \; x \; y)$$

or with explicit recursion

```
gisort_t :: (Ord a) => [a] -> C a -> [a]
gisort_t []     y      = tl y
gisort_t (x:xs) y = gisort_t xs (ist x y)
```

5. Sorting Hylos

- properties of the intermediate structures easily proved by induction:
  - foldr isth Empty and
    foldr istm (Leaf Nothing)
    generate *balanced* trees.
  - foldr isth Empty generates heaps...
  - foldr istq Empty generates BSTs...

this allows to refine tlBTree:

for heaps, tlBTRee becomes

```
catahsort :: (Ord a) => Heap a -> [a]
catahsort = foldBTree (\y a b -> y:(merge a b)) []
```

for BSTs, tlBTRee becomes

```
cataqsort :: (Ord a) => BTree a -> [a]
cataqsort = foldBTree (\y a b -> a++y:b) []
```

for leaf-trees, tlLTRee = catamsort

Overall, the intermediate structures constructed by folding over the unsorted list:

- have the same characteristics
- take the same asymptotic time to build

as those constructed by unfolding the trees, as in the standard sorting hylomorphisms.

The algorithms are in this sense equivalent.

To sum up, the specification of "sorting" is the list homomorphism $hom\ (\odot)\ (\backslash x \to [x])\ [\,]$. The efficiency of computing it directly leftwards or rightwards can be improved by using an intermediate tree structure (constructed either by folding or unfolding). The tree can be converted to the result type of the homomorphism by folding, using the parameters $(\odot), (\backslash x \to [x]), [\,]$ of the homomorphism. This fold may be optimized depending on properties of the intermediate structures.

This not only shows an interesting common structure of the divide-and-conquer algorithms, but it also proposes a design principle that can be applied to other list homomorphisms.