Programming Derivatives for Sorting

Zhenjiang Hu

National Institute of Informatics, Japan

APLAS Workshup, Suzhou, November 30, 2017

Fish and Bear Paw in Programming

Fish and Bear Paw in Programming

sequential programs

parallel programs

Fish and Bear Paw in Programming

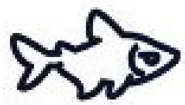
sequential programs
forward transformation

parallel programs
backward transformation

Fish and Bear Paw in Programming

sequential programs	parallel programs
forward transformation	backward transformation
non-incremental programs	incremental programs

Fish and Bear Paw in Programming



sequential programs	parallel programs
forward transformation	backward transformation
non-incremental programs	incremental programs
...	...

Fish and Bear Paw in Programming

sequential programs	parallel programs
forward transformation	backward transformation
non-incremental programs	incremental programs
...	...

We want to have both for different computation context, but it is difficult to maintain their consistency when one is changed.

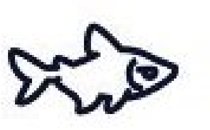
Fish and Bear Paw in Programming

Transformational Approach

$\Rightarrow$

Fish and Bear Paw in Programming

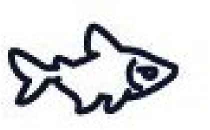
Transformational Approach



sequential programs $\Rightarrow$ parallelization parallel programs

Fish and Bear Paw in Programming

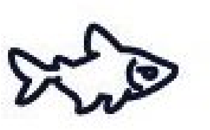
Transformational Approach



sequential programs	$\Rightarrow$ parallelization	parallel programs
forward transformation	bidirectionalozation	backward transformation

Fish and Bear Paw in Programming

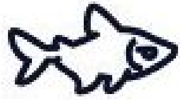
Transformational Approach



	$\Rightarrow$	
sequential programs	parallelization	parallel programs
forward transformation	bidirectionalozation	backward transformation
non-incremental programs	incrementalization	incremental programs

Fish and Bear Paw in Programming

Transformational Approach



	$\Rightarrow$	
sequential programs	parallelization	parallel programs
forward transformation	bidirectionalozation	backward transformation
non-incremental programs	incrementalization	incremental programs

However, some transformations are hard to be automated!

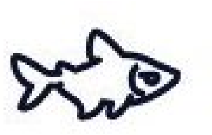
Fish and Bear Paw in Programming

Inverse Transformation

$\Leftarrow$

Fish and Bear Paw in Programming

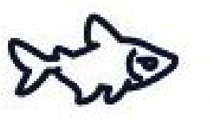
Inverse Transformation



sequential programs $\Leftarrow$ sequentialization parallel programs

Fish and Bear Paw in Programming

Inverse Transformation



sequential programs
forward transformation

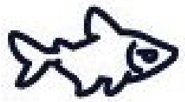
$\Leftarrow$

sequentialization
bidirectionalozation

parallel programs
backward transformation

Fish and Bear Paw in Programming

Inverse Transformation



sequential programs
forward transformation
non-incremental programs

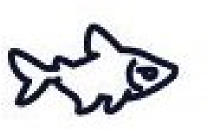
$\Leftarrow$
sequentialization
bidirectionalozation
???

Fish Bearpaw

parallel programs
backward transformation
incremental programs

Fish and Bear Paw in Programming

Inverse Transformation



sequential programs
forward transformation
non-incremental programs

$\Leftarrow$
sequentialization
bidirectionalozation
???

parallel programs
backward transformation
incremental programs

Example: Quicksort

Non-Incremental

```java
public class Quicksort  {
    private int[] numbers;
    private int number;
    public void sort(int[] values) {
        if (values ==null || values.length==0){
            return;
        }
        this.numbers = values;
        number = values.length;
        quicksort(0, number - 1);
    }
    private void quicksort(int low, int high) {
        int i = low, j = high;
        int pivot = numbers[low + (high-low)/2];

        while (i <= j) {
            while (numbers[i] < pivot) {
                i++;
            }
            while (numbers[j] > pivot) {
                j--;
            }
            if (i <= j) {
                exchange(i, j);
                i++;
                j--;
            }
        }
        if (low < j)
            quicksort(low, j);
        if (i < high)
            quicksort(i, high);
    }
```

Incremental

```cpp
template<typename I>
class inc_quick_sorter {
public:
  inc_quick_sorter(I i1, I i2) : first(i1), last(i2) {}

  class iterator {
  public:
    ...
    iterator& operator++() {
      ensure_sorted_at_current();
      ++current;
      return *this;
    }

    value_type& operator*() {
      ensure_sorted_at_current();
      return *current;
    }

  private:
    void ensure_sorted_at_current() {
      if (current == sort_end) {
        while (stack.back() - sort_end > sort_limit) {
          auto range_size = stack.back() - current;
          value_type pivot = *(current + (mt() % range_size));
          auto it = std::partition(
            current,
            stack.back(),
            [=](const value_type& v) { return v < pivot; });
          while (it == current) {
            pivot = *(current + (mt() % range_size));
            it = std::partition(
              current,
```

Can we specify and structure incremental programs well so that the non-incremental ones can be obtained for free automatically?

Can we specify and structure incremental programs well so that the non-incremental ones can be obtained for free automatically?

$$\Downarrow$$

Programming derivatives (for sorting)!

Outline

Differentiable Types

Definition (Differentiable Types)

A type τ is *differentiable* if there exists a delta type $\triangle_\tau$ with the following operators:

$$
\begin{array}{lr}
\ominus : \ \tau \to \tau \to \triangle_\tau & \{ \text{ partial diff operator } \} \\
\oplus : \ \tau \to \triangle_\tau \to \tau & \{ \text{ update operator } \} \\
0 : \ \triangle_\tau & \{ \text{ zero delta } \}
\end{array}
$$

satisfying the following properties:

$$
\begin{aligned}
t \ominus t &= 0 \\
t \oplus (s \ominus t) &= s \quad \text{if } s \ominus t \text{ is defined}
\end{aligned}
$$

Definition (Delta Composition)

Delta composition

$$\odot : \triangle_\tau \to \triangle_\tau \to \triangle_\tau$$

is defined by

$$t \oplus (d_1 \odot d_2) = (t \oplus d_1) \oplus d_2.$$

Example (Number)

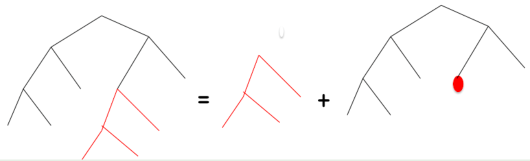
The type *Num* is differentiable by

$$\triangle_{Num} = Num$$
$$\ominus = -$$
$$\oplus = +$$
$$0 = 0$$

Note: $n \odot m = n + m$ and a minimum delta basis is $\{1\}$. Any non-zero delta can be represented as compositions of deltas in the minimal delta basis.

Example (Algebraic Data Types)

McBride Conor, The Derivative of a Regular Type is its Type of One-Hole Contexts, 2011.



$$
\begin{aligned}
t_1 \ominus t_2 &= \text{replace subtree } t_2 \text{ in } t_1 \text{ by a hole} \\
t \oplus d &= \text{fill in the hole in } d \text{ with } t \\
0 &= \text{the context with a single hole}
\end{aligned}
$$

Example (Delta List 1)

List $[a]_1$ is differentiable by

$$\begin{array}{lcl} \triangle_{[a]_1} & = & [a]_1 \\ (xs_1 +\!\!+\, xs_2) \ominus xs_2 & = & xs_1 \\ xs \oplus ds & = & ds +\!\!+\, xs \\ 0 & = & [] \end{array}$$

Note: $n \odot m = m +\!\!+\, n$ and a minimum delta basis is $\{[x] \mid x \leftarrow a\}$. Any non-zero delta can be represented as compositions of deltas in the minimal delta basis.

Example (Delta List 2)

List $[a]_2$ is differentiable by

$$
\begin{array}{lcl}
\triangle_{[a]_2} & = & ([a]_2, Num) \\
(xs_1 \mathbin{+\!\!+} xs \mathbin{+\!\!+} xs_2) \ominus (xs_1 \mathbin{+\!\!+} xs_2) & = & (xs, \# xs_1) \\
xs \oplus (ds, i) & = & take\ i\ xs \mathbin{+\!\!+} ds \mathbin{+\!\!+} drop\ i\ xs \\
0 & = & ([\,], 0)
\end{array}
$$

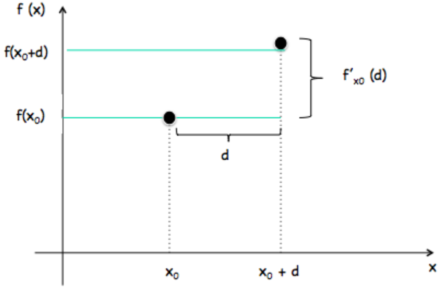
Note that a minimal delta basis is $\{([x], i) \mid x \leftarrow a, i \leftarrow Nat\}$.

Outline

Definition (Function Derivative)

A function $f : A \to B$ is differentiable if there exists a derivative $f' : A \to \triangle_A \to \triangle_B$ such that

$$f(x \oplus d) = f\ x \oplus f'_x\ d$$



Note: We sometimes write $f'\ x\ d$ instead of $f'_x\ d$.

Definition (Integral)

Given a derivative $f' : A \to \triangle_A \to \triangle_B$, its integral is $f : A \to B$ defined as follows:

$$f = \int_{(x_0, y_0)} f'$$

where

$$(\int_{(x_0, y_0)} f')\ x \quad = \quad y_0 \oplus f'\ x_0\ (x \mathbin{\hat{\ominus}} x_0)$$

with the boundary condition of $f\ x_0 = y_0$.

Definition (Integral)

Given a derivative $f' : A \to \triangle_A \to \triangle_B$, its integral is $f : A \to B$ defined as follows:

$$f = \int_{(x_0,y_0)} f'$$

where

$$(\int_{(x_0,y_0)} f')\ x \quad = \quad y_0 \ \oplus \ f'\ x_0\ (x \ \hat{\ominus}\ x_0)$$

with the boundary condition of $f\ x_0 = y_0$.

Lemma (Inversion)

$$(\int_{\_} f')' = f'$$

Example (sum')

$$\begin{array}{lcl} sum' & :: & [Num]_2 \rightarrow \triangle_{[Num]_2} \rightarrow \triangle_{Num} \\ sum'_{xs}\ ([x], i) & = & x \end{array}$$

Example (sort')

$$
\begin{array}{lcl}
sort' & :: & [a]_2 \rightarrow \triangle_{[a]_2} \rightarrow \triangle_{[a]_2} \\
sort'_{xs}\ ([x], i) & = & ([x], \#[y \mid y \leftarrow xs,\ y \leq x])
\end{array}
$$

Example (sort')

$$
\begin{array}{lcl}
sort' & :: & [a]_2 \rightarrow \triangle_{[a]_2} \rightarrow \triangle_{[a]_2} \\
sort'_{xs}\ ([x], i) & = & ([x], \#[y \mid y \leftarrow xs,\ y \leq x])
\end{array}
$$

An integral of $sort'$ with boundary conditions of $([\,],[\,])$ and $(xs, sort\ xs)$ gives the definition for $sort$:

$$
\begin{array}{lcl}
sort\ [\,] & = & [\,] \\
sort\ (x : xs) & = & sort\ xs \oplus sort'_{xs}\ ([x], 0)
\end{array}
$$

Note: $x : xs = xs \oplus ([x], 0)$.

Definition ($\lambda^{\triangle}$)

$$
\begin{array}{lcll}
e & := & c & \{ \text{ differentiable constant } \} \\
 & | & x & \{ \text{ variable } \} \\
 & | & \lambda x.e & \{ \text{ lambda } \} \\
 & | & e\ e & \{ \text{ application } \}
\end{array}
$$

Theorem

Any function $f = \lambda x.e$ (defined in $\lambda^{\triangle}$) is differentiable.

Proof.

We can prove that the derivative of $f = \lambda x.e$ is defined by

$$f'_x\ d = \frac{\partial e}{\partial x/d}$$

where

$$\begin{aligned}
\frac{\partial c}{\partial x/d} &= 0 \\
\frac{\partial x}{\partial x/d} &= d \\
\frac{\partial y}{\partial x/d} &= 0 \\
\frac{\partial(\lambda y.e)}{\partial x/d} &= \lambda y.\ \frac{\partial e}{\partial x/d} \\
\frac{\partial(r\ \ s)}{\partial x/d} &= (r'_s\ \frac{\partial s}{\partial x/d}) \odot (\frac{\partial r}{\partial x/d}\ (s \oplus \frac{\partial s}{\partial x/d}))
\end{aligned}$$

Example

Let $ss = \lambda x.sum\ (sort\ x)$. We can have

$$ss'\ x\ ([a], i) = a$$

by the following calculation:

Example

Let $ss = \lambda x.sum\ (sort\ x)$. We can have

$$ss'\ x\ ([a], i) = a$$

by the following calculation:

$$\begin{array}{ll}
& \frac{\partial sum\ (sort\ x)}{\partial x/([a],i)} \\
= & (sum'\ (sort\ x)\ \frac{\partial sort\ x}{\partial x/([a],i)}) \odot (\frac{\partial sum}{\partial x/([a],i)}\ (sort\ x \oplus \frac{\partial sort\ x}{\partial x/([a],i)})) \\
= & (sum'\ (sort\ x)\ (sort'\ x\ ([a], i))) \odot (0\ (sort\ x \oplus sort'\ x\ ([a], i))) \\
= & sum'\ (sort\ x)\ (sort'\ x\ ([a], i)) \\
= & sum'\ (sort\ x)\ ([a], \ldots) \\
= & a
\end{array}$$

Outline

Incremental Insert/Merge Sorting

Definition (Delta of Sorted Lists)

Sorted list $[a]_s$ is differentiable by

$$
\begin{array}{lcl}
\triangle_{[a]_s} & = & [a]_s \\
xs_1 \ominus xs_2 & = & xs_1 - xs_2 \quad (\text{if } Set\ xs_2 \subseteq Set\ xs_1) \\
xs \oplus ds & = & merge\ xs\ ds \\
0 & = & [\,]
\end{array}
$$

Example (msort')

$$\begin{array}{lcl} msort' & :: & [a]_1 \to \triangle_{[a]_1} \to \triangle_{[a]_s} \\ msort'_{xs}\ ds & = & msort\ ds \end{array}$$

Here *msort* is an integral of $msort'$:

$$msort\ (xs +\!\!+\ ys) = msort\ xs \oplus msort'_{xs}\ ys$$

with the boundary condition of $msort\ [\,] = [\,]$.

Example (msort')

$$\begin{array}{lcl} msort' & :: & [a]_1 \rightarrow \triangle_{[a]_1} \rightarrow \triangle_{[a]_s} \\ msort'_{xs}\ ds & = & msort\ ds \end{array}$$

Here *msort* is an integral of $msort'$:

$$msort\ (xs \mathbin{+\!\!+} ys) = msort\ xs \oplus msort'_{xs}\ ys$$

with the boundary condition of $msort\ [\,] = [\,]$.

This is how we code the merge sorting dynamically. Note that it is the *insert sorting* if $\#ds = 1$.

Can we code other sorting algorithms dynamically?

Can we code other sorting algorithms dynamically?

Definition (Delta of Binary Search Trees)

Binary search tree $BST\ a$ is differentiable by ...

$\Rightarrow$ similar to quick sorting.

Can we code other sorting algorithms dynamically?

Definition (Delta of Binary Search Trees)

Binary search tree $BST\ a$ is differentiable by ...

$\Rightarrow$ similar to quick sorting.

Definition (Delta of Heap Trees)

Heap tree $Heap\ a$ is differentiable by ...

$\Rightarrow$ similar to heap sorting.

Conclusion

- While automatic incrementalization is very difficult, automatic integration comes for free:
  - data derivatives
  - function derivatives
  - derivative composition

Conclusion

- While automatic incrementalization is very difficult, automatic integration comes for free:
  - data derivatives
  - function derivatives
  - derivative composition
- We demonstrate its power using the sorting algorithms.
  - Various incremental sorting algorithms can be declaratively and efficiently specified.

Conclusion

- While automatic incrementalization is very difficult, automatic integration comes for free:
  - data derivatives
  - function derivatives
  - derivative composition
- We demonstrate its power using the sorting algorithms.
  - Various incremental sorting algorithms can be declaratively and efficiently specified.

Key points: Changes are manipulable and compositional

Conclusion

- While automatic incrementalization is very difficult, automatic integration comes for free:
  - data derivatives
  - function derivatives
  - derivative composition
- We demonstrate its power using the sorting algorithms.
  - Various incremental sorting algorithms can be declaratively and efficiently specified.

Key points: Changes are manipulable and compositional

$\Rightarrow$ Towards Change-Oriented Programming!