より清浄なStream Fusion

小林 友明　Oleg Kiselyov　(東北大学)

背景

- ストリーム：　逐次的に処理されるデータの系列（無限長を含む）
- stream fusion：　通常の関数型のストリーム処理において，処理中に生成されてしまう中間データ構造を除去すること
- strymonas[1]：　モジュール性・記述性の高い関数型のストリーム処理を表す見かけの「パイプライン」から，安全で実行効率の高い命令型のストリーム処理のコードを生成するDSL

問題点

元のstrymonasによって生成される，filterまたはflat-mapされたストリームの組に対してzipを行うストリーム処理のコードには，次の2つの問題がある

- 固定サイズの中間データ構造(option型)が必要 → 完璧なfusionにならない
- クロージャを含む → 正規形を満たさない(効率の良いLLVM IRへの自明なコンパイルが不可能)

解決方法

Strymonasによるストリーム処理のコード生成を以前よりも体系的に扱う

- 全てのprimitiveな操作(map, filter, etc.)に対して閉じているストリーム処理のIRを導入
- IRに対する操作の実行をストリーム処理の正規化と見做す

[1] Kiselyov, O. et al. “Stream fusion, to completeness.” POPL 2017.

```
    iota .<1>.
 |> map (fun e -> .<.~e * .~e>.)
 |> take .<10>.
 |> fold (fun z e -> .<.~z + .~e>.) .<0>.
```

↓コード生成

```
let lv1 = 10 - 1 in
let z = ref 0 in
let lv2 = ref 1 in
let ir = ref 0 in
let goon = ref (0 <= lv1) in
while ! goon do
  let i = !ir in
  if i > lv1
  then goon := false
  else
    (incr ir;
     let v = !lv2 in incr lv2;
     let t = v * v in z := !z + t)
  done;
!z
```

↓実行

$1^2 + 2^2 + 3^2 + \cdots + 10^2$

1. リファクタリング

```
type _ st_stream =
  | Init      : 's init * ('s -> 'a st_stream) ->
                'a st_stream
  | For       : 'a pull_array -> 'a st_stream
  | Unfold    : 'a emit       -> 'a st_stream
  | Stuttered : 'a option st_stream -> 'a st_stream
  | Nested    : 'b code st_stream * ('b code ->
                'a st_stream) -> 'a st_stream
  | Block     : 'a emit st_stream -> 'a st_stream
```

- 上記のIRを導入
- Stutteredは**continue**，Blockは**break**の付いたループを表現
- 上述の問題は，zip対象の両者がflat-mapである場合の生成コードを除いて解決済み（次が改善例）

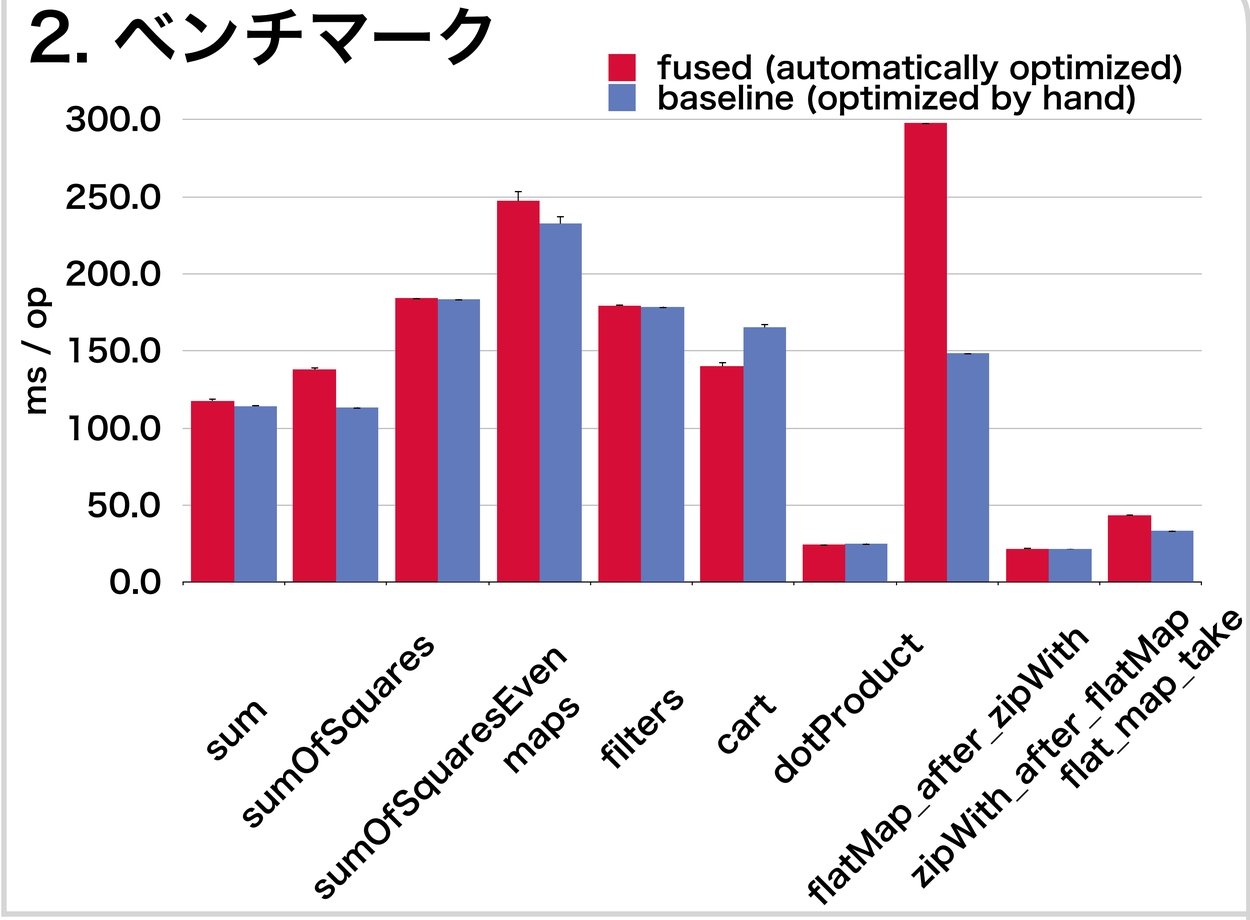
改善されるストリーム処理の例

```
(of_arr .<arr1>. |> filter (fun e -> .<.~e mod 2 = 1>.),
 of_arr .<arr2>. |> filter (fun e -> .<.~e mod 3 = 2>.))
|> uncurry @@ zip_with (fun x y -> .<.~x + .~y>.)
|> fold (fun z x -> .<.~z + .~x>.) .<0>.
```

A. 新実装による生成コード

```
let len1 = (Array.length arr1) - 1 in
let len2 = (Array.length arr2) - 1 in
let z = ref 0 in
let i2 = ref 0 in
let i1 = ref 0 in
let goon1 = ref (0 <= len1) in
while !goon1 do
  let i1' = ! i1 in
  if i1' > len1
  then goon1 := false
  else
    (incr i1;
     let e1 = Array.get arr1 i1' in
     if e1 mod 2 = 1
     then
       let goon2 = ref true in
       while !goon2 do
         if !i2 > len2
         then (goon2 := false;
               goon1 := false)
         else
           (let e2 = Array.get
                       arr2 (!i2) in
            if e2 mod 3 = 2
            then
              (incr i2;
               goon2 := false;
               z := !z + (e1 + e2))
            else (incr i2; ()))
       done
     else ())
done;
!z
```

2. ベンチマーク



B. 旧実装による生成コード

```
let z = ref 0 in
let i1 = ref 0 in
let curr = ref None in
let nadv = ref None in
let adv () =
  curr := None;
  while
    (!curr = None) &&
    ((!nadv <> None) ||
     (!i1 <=
      (Array.length arr1) - 1))
  do
    match !nadv with
    | Some adv' -> adv' ()
    | None ->
      let e1' = arr1.(!i1) in
      incr i1;
      if e1' mod 2 = 1
      then curr := Some e1'
  done in
  adv ();
let i2 = ref 0 in
let term = ref (!curr <> None) in
while !term && (!i2 <=
  (Array.length arr2) - 1)
do
  let e2 = arr2.(!i2) in
  incr i2;
  if e2 mod 3 = 2
  then
    (match !curr with
     | Some e1 ->
       adv ();
       term := !curr <> None;
       z := !z + (e1 + e2))
done;
!z
```