A Case Study on

Proving Transformations Correct

Stephen Fitzpatrick
Kestrel Institute
Palo Alto, California, USA

Peter Kilpatrick & Maurice Clint
Department of Computer Science
The Queen's University of Belfast

Introduction

- Overview of application: data-parallel conversion
- Notation and functions used
- Strategy for conversion
- Transformations applied to an example
- Example proofs

Motivation

Functional form:

```
fun times(U:real vector, V:real vector):real vector
      = map(U, V, *)
fun sum(U:real vector):real
      = fold(+, 0, U)
fun innerproduct(U:real vector, V:real vector):real
      = sum(times(U, V))
fun mmmult(A:real matrix, B:real matrix):real matrix
      = generate([shape(A, 1), shape(B, 2)],
        λ[i,j]·innerproduct(row(A,i), column(B,j)))
```

Fortran Plus Enhanced form (DAP array processor):

```
   C = 0
   do 10 k = 1, m
   C = C + matc(A( ,k),l)*matr(B(k, ),n)
10 continue
```

Intermediate Forms

SML

$\Downarrow$

λ-calculus

$\Downarrow$

Unfolded

$\Downarrow$

Simplified

$\Downarrow$

Array Form

$\Downarrow$

Common Sub-expressions Eliminated

$\Downarrow$

Fortran Plus Enhanced

Primitive Array Functions

shape: α array $\rightarrow$ shape
shape: α array $\times$ integer $\rightarrow$ integer
element: α array $\times$ index $\rightarrow$ α
A@i $\stackrel{def}{=}$ element(A,i)

$$\text{shape}\left(\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}\right) = [2, 3]$$

$$\text{shape}\left(\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, 1\right) = 2$$

$$\text{element}\left(\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, [2, 3]\right) = 6$$

generate: shape $\times$ (index $\rightarrow \alpha$) $\rightarrow \alpha$ array
reduce: ($\alpha \times \alpha \rightarrow \alpha$) $\times\ \alpha \times$ shape $\times$ (index $\rightarrow \alpha$) $\rightarrow \alpha$

$$\text{generate([2,2], } \lambda\text{[i,j]}\cdot\text{if i=j then 1 else 0)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\text{reduce(+, 0, shape(A), } \lambda\text{[i,j]}\cdot\text{A@[i,j])} = 45$$

$$\text{where } A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Array Form Functions

map(A: α array, B: β array, f: $\alpha \times \beta \to \gamma$) $\to \gamma$ array

$\stackrel{def}{=}$ generate(shape(A), λi·f(A@i, B@i))

ϵ(f:$\alpha \times \beta \to \gamma$) $\to$ (α array $\times$ β array $\to \gamma$ array)

$\stackrel{def}{=}$ λX,Y·generate(shape(Y), λi·f(X@i, Y@i))

fold(r:$\alpha \times \alpha \to \alpha$, r0: α, A: α array) $\to \alpha$

$\stackrel{def}{=}$ reduce(r, r0, shape(A), λi·A@i) row(A:α array, i:integer) $\rightarrow$ α array

$\stackrel{def}{=}$ generate(shape(A, 2), λ[j]$\cdot$A@[i, j])

expandrows(n:integer, U:α array) $\rightarrow$ α array

$\stackrel{def}{=}$ generate([n] $\times$ shape(U), λ[i,j]$\cdot$U@[j])

$$\text{expandrows(3, [1, 2, 3])} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$$

The Transformational Strategy

- Automatic application of local transformations
- Exhaustive application (fixed-point iteration)
- Distribution laws push instances of generate down into expressions creating instances of Array Form functions: e.g.

  generate(S, λi·A@i + B@i)
  $\Rightarrow$ map(generate(S, λi·A@i), generate(S, λi·B@i), +)

- Base cases: e.g.

  generate(S, λi·A@i) $\Rightarrow$A

  (where A has shape S)
- Some additional transformations to optimize parallelism

Worked Example: Matrix-Matrix Multiplication

Assume real matrices A, B of shapes [l,m], [m,n] respectively.

mmmult(A, B)
$\Rightarrow$ *unfolding and simplification*
generate([l,n], λ[i,j]·reduce(+, 0, [m], λ[k]·A@[i,k]*B@[k,j]))

- *Transformation:* generate-reduce *swap*

$$\begin{aligned} & \text{generate(S, } \lambda \text{i} \cdot \text{reduce(r, r0, T, } \lambda \text{j} \cdot \text{e))} \\ \equiv\ & \text{reduce(} \epsilon \text{(r), generate(S, } \lambda \text{i} \cdot \text{r0), T, } \lambda \text{j} \cdot \text{generate(S, } \lambda \text{i} \cdot \text{e))} \end{aligned}$$

where r, r0 *and* T *are independent of* i
and S *is independent of* j

---

generate([l,n], λ[i,j]·reduce(+, 0, [m], λ[k]·A@[i,k]*B@[k,j]))
$\Rightarrow$
reduce(ϵ(+), generate([l,n], λ[i,j]·0), [m],
λ[k]·generate([l,n], λ[i,j]·A@[i,k]*B@[k,j]))

- *Transformation: Propagation through scalar functions*

  generate(S, λi·f(a, b))
$\equiv$ map(generate(S, λi·a), generate(S, λi·b), f)
  where f *is a scalar function for which*
  elementwise application is supported by map

---

reduce(ϵ(+), generate([l,n], λ[i,j]·0), [m],
  λ[k]·generate([l,n], λ[i,j]·A@[i,k]*B@[k,j]))
$\Rightarrow$
reduce(ϵ(+), generate([l,n], λ[i,j]·0), [m],
 λ[k]·map(generate([l,n], λ[i,j]·A@[i,k]),
  generate([l,n], λ[i,j]·B@[k,j]),
  *))

- *Transformation: Generated expression independent of one index*

  generate([m,n], λ[i,j]$\cdot$e)
$\equiv$ expandrows(m, generate([n], λ[j]$\cdot$e))
    where e *is independent of* i

```
reduce(ε(+), generate([l,n], λ[i,j]·0), [m],
  λ[k]·map(generate([l,n], λ[i,j]·A@[i,k]),
    generate([l,n], λ[i,j]·B@[k,j]),
    *))
⇒
reduce(ε(+), generate([l,n], λ[i,j]·0), [m],
  λ[k]·map(expandcols(n, generate([l], λ[i]·A@[i,k])),
    expandrows(l, generate([n], λ[j]·B@[k,j])),
    *))
```

- *Transformation: Base case — row of a matrix*

generate([n], λ[j]·A@[r, j])
$\equiv$ row(A, r)
where A *has shape* [m,n]
and A *and* r *are independent of* j

```
reduce(ε(+), generate([l,n], λ[i,j]·0), [m],
  λ[k]·map(expandcols(n, generate([l], λ[i]·A@[i,k])),
    expandrows(l, generate([n], λ[j]·B@[k,j])),
    *))
⇒
reduce(ε(+), generate([l,n], λ[i,j]·0), [m],
  λ[k]·map(expandcols(n, column(A,k)),
    expandrows(l, row(B,k)),
    *))
```

Example Proofs

Axiomatic Definitions

$$\text{shape}(\text{generate}(S, \lambda i \cdot g)) \equiv S$$

$$\forall i' \in S: \ \text{element}(\text{generate}(S, \lambda i \cdot g), i') \equiv \lambda i \cdot g\,(i')$$

$$\text{reduce}(r, r0, \emptyset, \lambda i \cdot g) \equiv r0$$

$$\text{reduce}(r, r0, S+i', \lambda i \cdot g) \equiv r(\lambda i \cdot g\,(i'), \text{reduce}(r, r0, S, \lambda i \cdot g))$$

where $i' \notin S$ *and* $\emptyset$ *denotes the empty set (of indices)*

- *Lemma: Shape of an elementwise application*

shape(ϵ(f)(A, B)) $\equiv$ shape(A) $\equiv$ shape(B)

- *Lemma: Element of an elementwise application*

    element(ϵ(f)(A, B), i)
$\equiv$ f(element(A, i), element(B, i)) $\equiv$ f(A@i, B@i)

---

element(ϵ(f)(A, B), i$'$)
 = *definition of* ϵ
element(λX, Y·generate(shape(Y), λi·f(X@i, Y@i)) (A, B), i$'$)
 = β*-reduce*
element(generate(shape(B), λi·f(A@i, B@i)), i$'$)
 = *element of* generate
λi·f(A@i, B@i) (i$'$)
 = β*-reduce*
f(A@i$'$, B@i$'$)

- *Lemma: Element of an ϵ-reduction*

  element(reduce(ϵ(r), R0, S, λi$\cdot$g), j)
$\equiv$ reduce(r, element(R0, j), S, λi$\cdot$element(g, j))
  where S *is independent of* j

Proof is by induction on S.

Base Step: $\emptyset$

element(reduce(ϵ(r), R0, $\emptyset$, λi$\cdot$g), j)
 = *reduction over empty set*
element(R0, j)

reduce(r, element(R0, j), $\emptyset$, λi$\cdot$element(g, j))
 = *reduction over empty set*
element(R0, j)

Inductive Step: S+i’

Assume the lemma holds for shape S.

Consider shape S+i’ where i’ $\notin$ S.

Left side:

element(reduce(ϵ(r), R0, S+i$'$, λi·g), j)

= *reduction over set inclusion*

element(ϵ(r)(λi·g (i$'$), reduce(ϵ(r), R0, S, λi·g)), j)

= *element of an elementwise application*

r(element(λi·g (i$'$), j), element(reduce(ϵ(r), R0, S, λi·g), j))

= *by induction hypothesis*

r(element(λi·g (i$'$), j), reduce(r, element(R0, j), S, λi·element(g, j)))

Right side:

reduce(r, element(R0, j), S+i′, λi·element(g, j))

= *reduction over set inclusion*

r(λi·element(g, j) (i′), reduce(r, element(R0, j), S, λi·element(g, j)))

= *move λ-binding into* element*)*

r(element(λi·g (i′), j), reduce(r, element(R0, j), S, λi·element(g, j)))

=

Left side

Hence, by induction, the lemma holds for all shapes.

- *Transformation:* generate-reduce *swap*

  generate(S, λi·reduce(r, r0, T, λj·e))

$\equiv$ reduce(ϵ(r), generate(S, λi·r0), T, λj·generate(S, λi·e))

where r*,* r0 *and* T *are independent of* i
and S *is independent of* j

Same Shapes

shape(generate(S, λi·reduce(r, r0, T, λj·e)))
= *shape of* generate
S

shape(reduce(ϵ(r), generate(S, λi·r0), T, λj·generate(S, λi·e)))
= *shape of an ϵ-reduction*
shape(generate(S, λi·r0))
= *shape of* generate
S

Same Elements

Consider an arbitrary element i'.

Left side:

element(generate(S, λi·reduce(r, r0, T, λj·e)), i′)

= *element of* generate

λi·reduce(r, r0, T, λj·e) (i′)

= *since* r*,* r0 *and* T *are independent of* i*,*
move binding into reduction

reduce(r, r0, T, λi·(λj·e) (i′))

= *move binding of* i *into abstraction of* j

reduce(r, r0, T, λj·(λi·e (i′)))

Right side:

element(reduce(ϵ(r), generate(S, λi·r0), T, λj·generate(S, λi·e)), i$'$)

= *element of an ϵ-reduction*

reduce(r, element(generate(S, λi·r0), i$'$), T,
λj·element(generate(S, λi·e), i$'$))

= *element of* generate

reduce(r, λi·r0 (i$'$), T, λj·(λi·e (i$'$)))

= *since* r0 *is independent of* i

reduce(r, r0, T, λj·(λi·e (i$'$)))

=

Left side

Hence, arrays have same shapes and same elements, and so are equal.

Conclusion

- Conversion to Array Form is a significant step
- Proof of correctness simplified by:
  - the intermediate forms
  - the transformational style: short, local, simple rewrites
  - the simple semantics of the pure, functional form