Congruences for Incremental Datatype Migration

Christoph Reichenbach (Lund U),
Evan Chang (CU Boulder),
Amer Diwan (Google)

Why Incremental Datatype Migration?

vector: α `list` $\rightarrow$ α `vector`

Why Incremental Datatype Migration?

vector: α `list` $\rightarrow$ α `vector`

```
(* module A *)
fun derivatives(f, depth) =
  let val results : real list list = ...
  in  results
  end
fun integrals(f, depth) =
  let val results : real list list = ...
  in  results
  end
```

Why Incremental Datatype Migration?

vector: α `list` $\rightarrow$ α `vector`

```
(* module A *)
fun derivatives(f, depth) =
  let val results : real list list = ...
  in  results
  end
fun integrals(f, depth) =
  let val results : real list list = ...
  in  results
  end
```

```
(* module AutoFit *)
val deriv = A.derivatives (...)
fun get(d, x) =
  List.nth(List.nth(deriv, d), x)
```

Why Incremental Datatype Migration?

vector: α `list` $\rightarrow$ α `vector`

```
(* module A *)
fun derivatives(f, depth) =
   let val results : real list list = ...
   in vector (map vector results )
   end
fun integrals(f, depth) =
   let val results : real list list = ...
   in vector (map vector results )
   end
```

```
(* module AutoFit *)
val deriv = A.derivatives (...)
fun get(d, x) =
   List.nth(List.nth(deriv, d), x)
```

Why Incremental Datatype Migration?

vector: α `list` $\rightarrow$ α `vector`

```
(* module A *)
fun derivatives(f, depth) =
  let val results : real list list = ...
  in vector (map vector results )
  end
fun integrals(f, depth) =
  let val results : real list list = ...
  in vector (map vector results )
  end
```

```
(* module AutoFit *)
val deriv = A.derivatives (...)
fun get(d, x) =
    Vector.get(Vector.get(deriv, d), x)
```

Why Incremental Datatype Migration?

vector: α `list` $\rightarrow$ α `vector`

```
(* module A *)
fun derivatives(f, depth) =
  let val results : real list list = ...
  in vector (map vector results )
  end
fun integrals(f, depth) =
  let val results : real list list = ...
  in vector (map vector results )
  end
```

```
(* module AutoFit *)
val deriv = A.derivatives (...)
fun get(d, x) =
    Vector.get(Vector.get(deriv, d), x)
```

Challenges:

- Redundancy / Custom abstractions

Why Incremental Datatype Migration?

vector: α `list` $\rightarrow$ α `vector`

```
(* module A *)
fun derivatives(f, depth) =
   let val results : real list list = ...
   in vector (map vector results )
   end
fun integrals(f, depth) =
   let val results : real list list = ...
   in vector (map vector results )
   end
```

```
(* module DrawDerivatives *)
fun plotAll(...) =
  Plot.draw: real list → unit
  Plot.draw (A.derivatives ...)
```

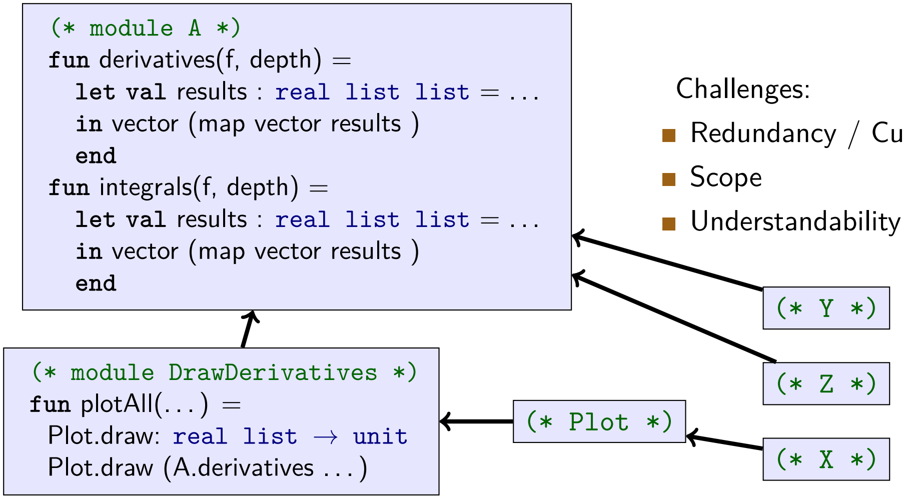
```
(* module AutoFit *)
val deriv = A.derivatives (...)
fun get(d, x) =
    Vector.get(Vector.get(deriv, d), x)
```

Challenges:

- Redundancy / Custom abstractions
- Scope

Why Incremental Datatype Migration?

vector: α `list` $\rightarrow$ α `vector`



```
(* module A *)
fun derivatives(f, depth) =
  let val results : real list list = ...
  in vector (map vector results )
  end
fun integrals(f, depth) =
  let val results : real list list = ...
  in vector (map vector results )
  end
```

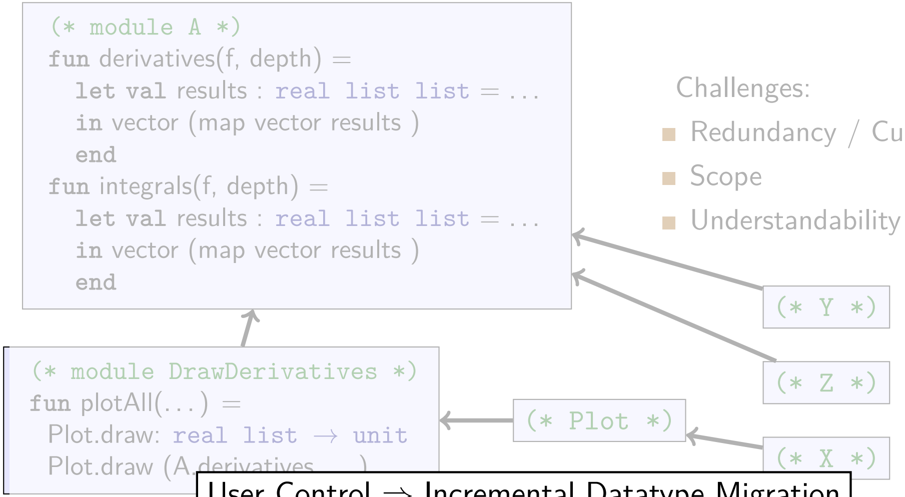
```
(* module DrawDerivatives *)
fun plotAll(...) =
  Plot.draw: real list → unit
  Plot.draw (A.derivatives ...)
```

Challenges:

- Redundancy / Custom abstractions
- Scope
- Understandability

Why Incremental Datatype Migration?

vector: α `list` $\rightarrow$ α `vector`



```
(* module A *)
fun derivatives(f, depth) =
  let val results : real list list = ...
  in vector (map vector results )
  end
fun integrals(f, depth) =
  let val results : real list list = ...
  in vector (map vector results )
  end
```

```
(* module DrawDerivatives *)
fun plotAll(...) =
 Plot.draw: real list → unit
 Plot.draw (A.derivatives ... )
```

Challenges:

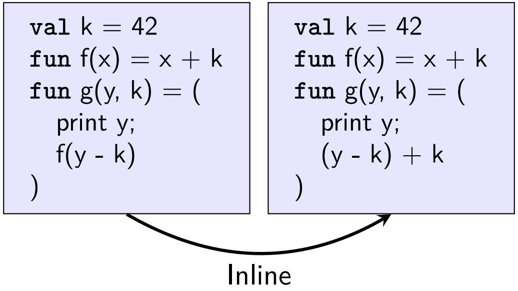
- Redundancy / Custom abstractions
- Scope
- Understandability

User Control ⇒ Incremental Datatype Migration

Program Metamorphosis for SML

```
val k = 42
fun f(x) = x + k
fun g(y, k) = (
   print y;
   f(y - k)
)
```

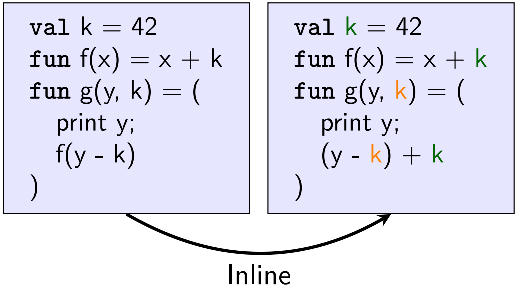
Program Metamorphosis for SML



```
val k = 42
fun f(x) = x + k
fun g(y, k) = (
   print y;
   f(y - k)
)
```

```
val k = 42
fun f(x) = x + k
fun g(y, k) = (
   print y;
   (y - k) + k
)
```

Program Metamorphosis for SML

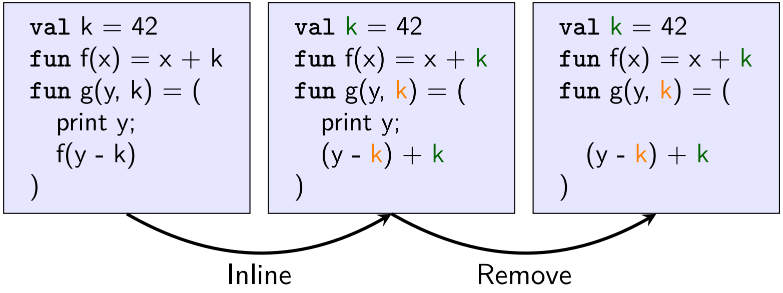


```
val k = 42
fun f(x) = x + k
fun g(y, k) = (
   print y;
   f(y - k)
)
```

```
val k = 42
fun f(x) = x + k
fun g(y, k) = (
   print y;
   (y - k) + k
)
```

- Error: Name Capture

Program Metamorphosis for SML



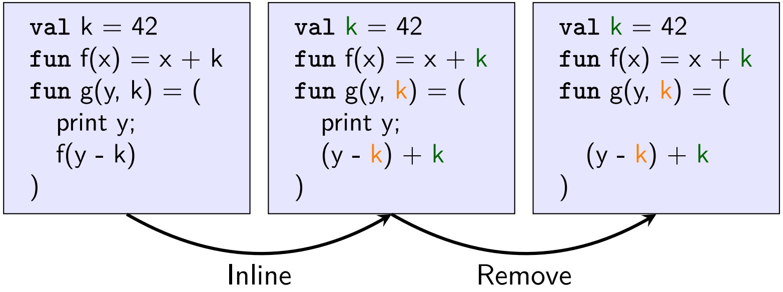
```
val k = 42
fun f(x) = x + k
fun g(y, k) = (
   print y;
   f(y - k)
)
```

```
val k = 42
fun f(x) = x + k
fun g(y, k) = (
   print y;
   (y - k) + k
)
```

```
val k = 42
fun f(x) = x + k
fun g(y, k) = (

   (y - k) + k
)
```

- Error: Name Capture

Program Metamorphosis for SML



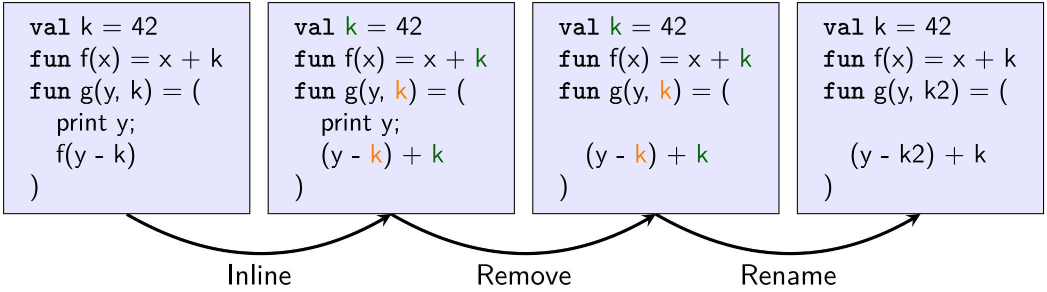
```
val k = 42
fun f(x) = x + k
fun g(y, k) = (
   print y;
   f(y - k)
)
```

```
val k = 42
fun f(x) = x + k
fun g(y, k) = (
   print y;
   (y - k) + k
)
```

```
val k = 42
fun f(x) = x + k
fun g(y, k) = (

   (y - k) + k
)
```

- Error: Name Capture
- Error: Side effect removed

Program Metamorphosis for SML



```
val k = 42
fun f(x) = x + k
fun g(y, k) = (
   print y;
   f(y - k)
)
```

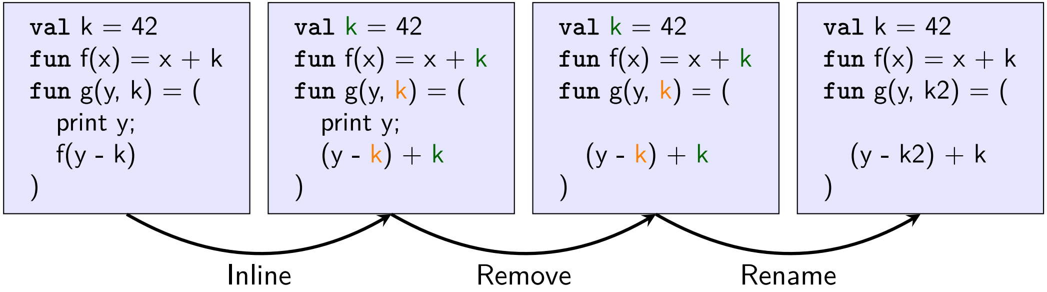
```
val k = 42
fun f(x) = x + k
fun g(y, k) = (
   print y;
   (y - k) + k
)
```

```
val k = 42
fun f(x) = x + k
fun g(y, k) = (

   (y - k) + k
)
```

```
val k = 42
fun f(x) = x + k
fun g(y, k2) = (

   (y - k2) + k
)
```

- Error: Name Capture
- Error: Side effect removed

Program Metamorphosis for SML



```
val k = 42
fun f(x) = x + k
fun g(y, k) = (
   print y;
   f(y - k)
)
```

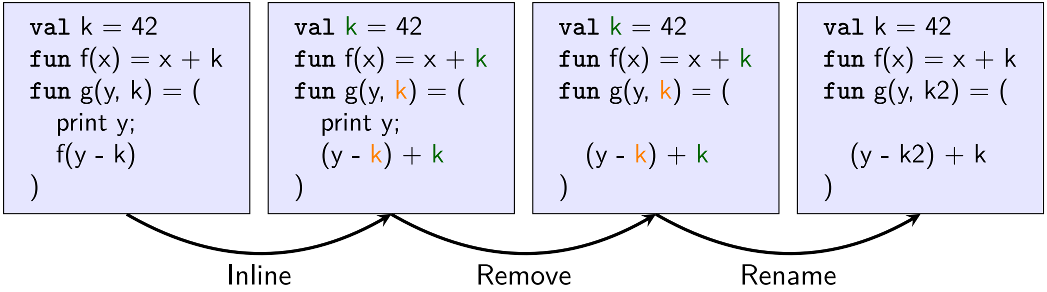
```
val k = 42
fun f(x) = x + k
fun g(y, k) = (
   print y;
   (y - k) + k
)
```

```
val k = 42
fun f(x) = x + k
fun g(y, k) = (

   (y - k) + k
)
```

```
val k = 42
fun f(x) = x + k
fun g(y, k2) = (

   (y - k2) + k
)
```

- Error: Side effect removed

Program Metamorphosis for SML



```
val k = 42
fun f(x) = x + k
fun g(y, k) = (
   print y;
   f(y - k)
)
```

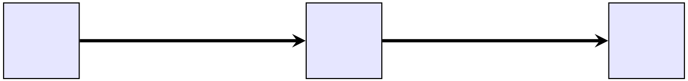
```
val k = 42
fun f(x) = x + k
fun g(y, k) = (
   print y;
   (y - k) + k
)
```

```
val k = 42
fun f(x) = x + k
fun g(y, k) = (

   (y - k) + k
)
```

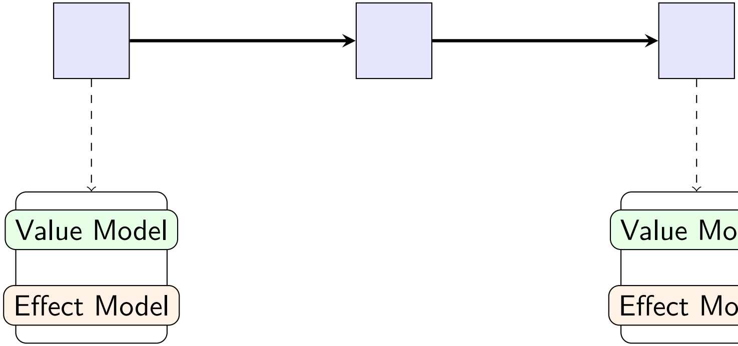
```
val k = 42
fun f(x) = x + k
fun g(y, k2) = (

   (y - k2) + k
)
```

- User confirmed: Side effect removed

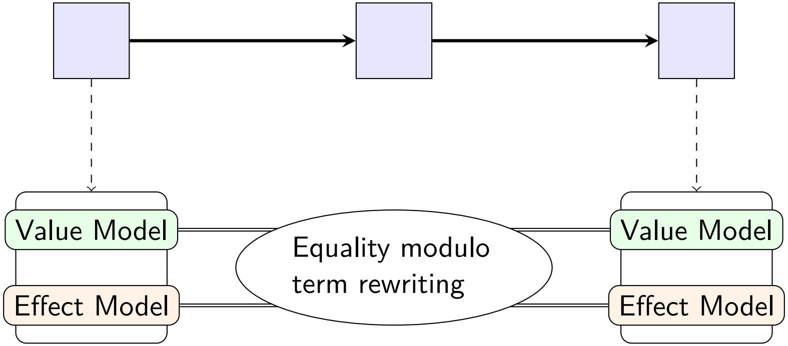
Program Metamorphosis: Implementation



Program Metamorphosis: Implementation



Program Metamorphosis: Implementation



Algorithmic Optimisation in Program Metamorphosis

Algorithmic Optimisation in Program Metamorphosis

- Extend term rewriting with user-defined axioms

$$\frac{V = \mathrm{vector}(L)}{Vector.sub(V, i) = \mathrm{List.nth}(L, i)}$$

Algorithmic Optimisation in Program Metamorphosis

- Extend term rewriting with user-defined axioms
- Use *congruences* ($\mathrel{\overset{\triangleleft}{\triangleright}}$) to increase abstraction

$$\frac{}{\mathrm{vector}(L) \mathrel{\overset{\triangleleft}{\triangleright}} L}$$

$$\frac{V \mathrel{\overset{\triangleleft}{\triangleright}} L}{Vector.sub(V, i) = \mathrm{List.nth}(L, i)} \qquad \frac{V_1 \mathrel{\overset{\triangleleft}{\triangleright}} L_1 \quad V_2 \mathrel{\overset{\triangleleft}{\triangleright}} L_2}{\mathrm{Vector.concat}[V_1, V_2] \mathrel{\overset{\triangleleft}{\triangleright}} L_1 @ L_2}$$

Splitting the Congruence

- Challenge: Can't put entire program into term rewriter

Splitting the Congruence

- Challenge: Can't put entire program into term rewriter

$\texttt{val}\ \ell = [1, 2, 3]$: `int list`

List.nth(ℓ, 1)

$\Rightarrow$ expect ℓ: `int list`

Splitting the Congruence

- Challenge: Can't put entire program into term rewriter

$\texttt{\textbf{val}}\ \ell = \text{vector}\ [1, 2, 3]\text{:}\ \texttt{int vector}$

$\text{List.nth}(\ell, 1)$

$\Rightarrow \text{expect}\ \ell\text{:}\ \texttt{int list}$

Splitting the Congruence

- Challenge: Can't put entire program into term rewriter

$\texttt{\textbf{val}}\ \ell = \triangleleft(\text{vector } [1, 2, 3])$: `int vector`

List.nth(ℓ, 1)

$\Rightarrow$ expect ℓ: `int list`

Splitting the Congruence

- Challenge: Can't put entire program into term rewriter
- Rewrite *with transformation obligation*: $\triangleleft(x) \equiv$ 'the vector equivalent to the list x'

val $\ell = \triangleleft$(vector [1, 2, 3]): `⊲|(int vector)`

List.nth(ℓ, 1)

$\Rightarrow$ expect ℓ: `int list`

Splitting the Congruence

- Challenge: Can't put entire program into term rewriter
- Rewrite *with transformation obligation*: $\triangleleft(x) \equiv$ 'the vector equivalent to the list x'
- Push obligation through type system

`val` $\ell = \triangleleft$(vector [1, 2, 3]): `⊲|(int vector)`

Type conflict

List.nth(ℓ, 1)

$\Rightarrow$ expect ℓ: `int list`

Splitting the Congruence

- Challenge: Can't put entire program into term rewriter
- Rewrite *with transformation obligation*: $\triangleleft(x) \equiv$ 'the vector equivalent to the list x'
- Push obligation through type system

$\texttt{val}\ \ell = \triangleleft(\text{vector } [1, 2, 3])$: `◁(int vector)`

$\text{List.nth}(\ell, 1)$

$\Rightarrow$ expect ℓ: `int list`

Splitting the Congruence

- Challenge: Can't put entire program into term rewriter
- Rewrite *with transformation obligation*: $\triangleleft(x) \equiv$ 'the vector equivalent to the list x'
- Push obligation through type system

$\texttt{\textbf{val}}\ \ell = \triangleleft(\text{vector } [1, 2, 3])$: `⊲|(int vector)`

Vector.sub($\triangleright\ell$, 1)

$\Rightarrow$ expect ℓ: `int vector`

Splitting the Congruence

- Challenge: Can't put entire program into term rewriter
- Rewrite *with transformation obligation*: $\triangleleft(x) \equiv$ 'the vector equivalent to the list x'
- Push obligation through type system
- Complementary obligations cancel out: $⧐(⧏(\tau)) = \tau$

$$\texttt{\textbf{val}}\ \ell = \triangleleft(\text{vector } [1, 2, 3]) \text{: } ⧏(\texttt{int vector})$$

Vector.sub($\triangleright\ell$, 1)

$\Rightarrow$ expect ℓ: $⧏(\texttt{int vector})$

$$\boxed{\textbf{Typechecks!}}$$

Conclusions and WIP

- Congruences allow reasoning over datatype migration
- Incrementalise reasoning by pushing transformation obligations into type system
- Applicable to many transformation challenges
  - List $\leftrightarrow$ Vector
  - Replacing (stateful) hashmap implementations
  - linear space to log space or quadratic space (requires preconditions)
- TODO:
  - Improve accuracy for locations to transform using recent related work
  - Suggest transformations