

Using Dataflow Optimization Techniques with a Monadic Intermediate Language

Justin Bailey
jgbailey@codeslower.com

Portland State University

May 7, 2012

Introduction

- ▶ Compiling Functional Programming Languages
- ▶ Dataflow Analysis
- ▶ Three-Address Code

Introduction

MIL

Dataflow Analysis

Uncurrying

Conclusion

Appendix

Dataflow Analysis for Functional Languages?

- ▶ Can we apply functional-language specific optimizations?
- ▶ Can we implement traditional dataflow-based optimizations?

Using Dataflow
Optimization
Techniques with a
Monadic
Intermediate
Language

Justin Bailey

Introduction

MIL

Dataflow Analysis

Uncurrying

Conclusion

Appendix

Monadic Programming to the Rescue!

- ▶ Simple control-flow
- ▶ Separate side-effecting computation from “pure” values
- ▶ Higher-order functions

MIL: A Monadic Intermediate Language

- ▶ Monadic: Haskell's **do** notation.
- ▶ Monadic: Segregated side-effects.
- ▶ Dataflow: Basic blocks.
- ▶ Dataflow: Block scope.
- ▶ Dataflow: Based on three-address code.

Contributions

- ▶ Applied the *dataflow algorithm* to a functional language.
- ▶ Implemented *uncurrying*, using the dataflow algorithm.
- ▶ Thorough exposition of the HOOPL library.

Goals of MIL

- ▶ Simplicity
- ▶ Allocation & Other Side-Effects
- ▶ Higher-Order Functions

Simplicity

$$\frac{(b * c + d)}{2}$$

Using Dataflow
Optimization
Techniques with a
Monadic
Intermediate
Language

Justin Bailey

Introduction

MIL

Goals of MIL

Related Work

Dataflow Analysis

Uncurrying

Conclusion

Appendix

Simplicity

$$\frac{(b * c + d)}{2}$$

► Three-Address Code

```
1  t1 := b * c;  
2  t2 := t1 + d;  
3  t3 := t2 / 2;
```

Simplicity

$$\frac{(b * c + d)}{2}$$

► MIL

```
1  t1 <- mul*(b, c)
2  t2 <- add*(t1, d)
3  t3 <- div*(t2, 2)
```

Side-Effects

Using Dataflow
Optimization
Techniques with a
Monadic
Intermediate
Language

Justin Bailey

Introduction

MIL

Goals of MIL

Related Work

Dataflow Analysis

Uncurrying

Conclusion

Appendix

Side-Effects

- ▶ Closures

```
t1 <- k {x}
```

Side-Effects

- ▶ Closures

```
t1 <- k {x}
```

Side-Effects

- ▶ Closures

```
t1 <- k {x}
```

- ▶ Data values

```
t2 <- Cons x xs
```

Side-Effects

- ▶ Closures

```
t1 <- k {x}
```

- ▶ Data values

```
t2 <- Cons x xs
```

Side-Effects

- ▶ Closures

```
t1 <- k {x}
```

- ▶ Data values

```
t2 <- Cons x xs
```

- ▶ Primitives

```
t3 <- add*(x, y)
```


Sufficiency

- ▶ Higher-Order Functions
- ▶ Primitives
- ▶ Data Values

Using Dataflow
Optimization
Techniques with a
Monadic
Intermediate
Language

Justin Bailey

Introduction

MIL

Goals of MIL

Related Work

Dataflow Analysis

Uncurrying

Conclusion

Appendix

Related Work

- ▶ MLj: Benton, Kennedy, & Russell¹
- ▶ Continuation-Passing Style²
- ▶ Administrative-Normal Form: Flanagan, Sabry, Duba, and Felleisen³

¹“Compiling Standard ML to Java Bytecodes” (1998).

²See Appel’s “Compiling with Continuations” (1992).

³“The Essence of Compiling with Continuations” (1993).

Dataflow Analysis

- ▶ Due to Kildall's "A Unified Approach to Global Program Optimization" (1973)
- ▶ Widely applied to imperative programming languages

Using Dataflow
Optimization
Techniques with a
Monadic
Intermediate
Language

Justin Bailey

Introduction

MIL

Dataflow Analysis

Fundamentals

Hoopl

Uncurrying

Conclusion

Appendix

Typical Dataflow Optimizations

- ▶ Dead-code Elimination
- ▶ Constant Folding
- ▶ Lazy Code Motion
- ▶ For more, see Muchnick's "Advanced compiler design and implementation" (1997)

Fundamentals: CFGs & Basic Blocks

Using Dataflow
Optimization
Techniques with a
Monadic
Intermediate
Language

Justin Bailey

Introduction

MIL

Dataflow Analysis

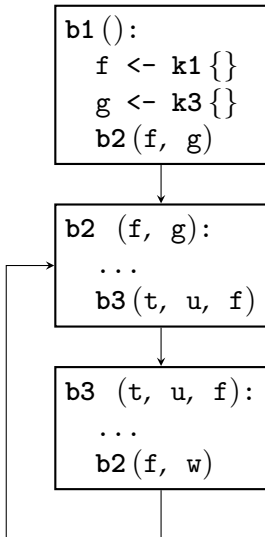
Fundamentals

Hoop!

Uncurrying

Conclusion

Appendix



Facts

Using Dataflow
Optimization
Techniques with a
Monadic
Intermediate
Language

Justin Bailey

Introduction

MIL

Dataflow Analysis

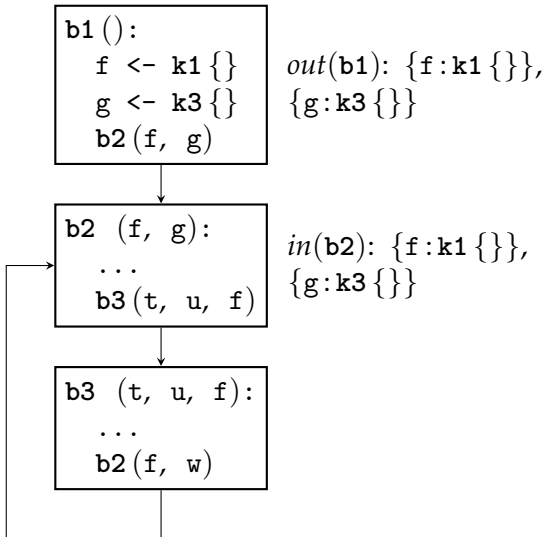
Fundamentals

Hoop!

Uncurrying

Conclusion

Appendix



Iteration

Using Dataflow
Optimization
Techniques with a
Monadic
Intermediate
Language

Justin Bailey

Introduction

MIL

Dataflow Analysis

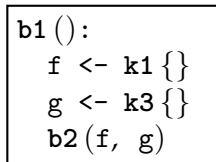
Fundamentals

HoopI

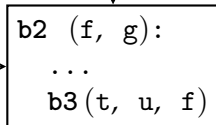
Uncurrying

Conclusion

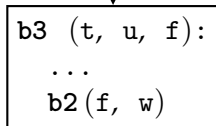
Appendix



$out(b1): \{f:k1 \{\}\},$
 $\{g:k3 \{\}\}$



$in(b2): \{f:k1 \{\}\},$
 $\{g:k3 \{\}\}$



$out(b3):$
 $\{f:k1 \{\}\},$
 $\{g:k4 \{v\}\}$

Iteration

Using Dataflow
Optimization
Techniques with a
Monadic
Intermediate
Language

Justin Bailey

Introduction

MIL

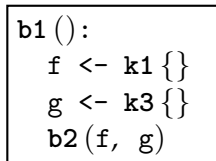
Dataflow Analysis

Fundamentals
HoopI

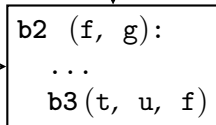
Uncurrying

Conclusion

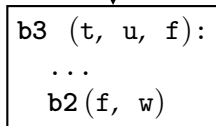
Appendix



$out(b1): \{f:k1 \{\}\},$
 $\{g:k3 \{\}\}$



$in(b2): \{f:k1 \{\}\},$
 $\{g:\top\}$



$out(b3):$
 $\{f:k1 \{\}\},$
 $\{g:k4 \{v\}\}$

HOOPL: A Haskell Library for Dataflow Analysis

- ▶ See “Hoopl: A Modular, Reusable Library for Dataflow Analysis and Transformation” by Ramsey, Dias, and Peyton Jones (2010)
- ▶ Used in the Glasgow Haskell Compiler
- ▶ Based on “Composing Dataflow Analyses and Transformations” by Lerner, Grove, and Chambers (2002)

Uncurrying

- ▶ Partial Application
- ▶ Uncurrying *map*

Partial Application

$map\ f\ xs = \mathbf{case\ } xs\ \mathbf{of}$

$Cons\ x\ xs' \rightarrow Cons\ (f\ x)\ (map\ f\ xs')$

$Nil \rightarrow Nil$

$toList\ x = Cons\ x\ Nil$

Partial Application

mkLists = map toList

map f xs = case xs of

Cons x xs' → Cons (f x) (map f xs')

Nil → Nil

toList x = Cons x Nil

Partial Application

mkLists = map toList

main1 ns = map toList ns

main2 ns = mkLists ns

*map f xs = **case xs of***

Cons x xs' → Cons (f x) (map f xs')

Nil → Nil

toList x = Cons x Nil

Uncurrying *map*

main ns = map toList ns

map f xs = case xs of

Cons x xs' →

Cons (f x) (map f xs')

Nil → Nil

toList x = Cons x Nil

Uncurrying *map*

main ns = *map toList ns*

map f xs = **case xs of**

Cons x xs' →

Cons (f x) (map f xs')

Nil → *Nil*

toList x = *Cons x Nil*

```
1  main (ns):  
2    v227 <- k203 {}  
3    v228 <- k219 {}  
4    v229 <- v227 @ v228  
5    v229 @ ns
```

Uncurrying *map*

main (ns):

main ns = *map toList ns*

map f xs = **case xs of**

Cons x xs' →

Cons (f x) (map f xs')

Nil → *Nil*

toList x = *Cons x Nil*

```
1  main (ns):  
2    v227 <- k203 {}  
3    v228 <- k219 {}  
4    v229 <- v227 @ v228  
5    v229 @ ns
```


Uncurrying *map*

main ns = map toList ns

map f xs = case xs of

Cons x xs' →

Cons (f x) (map f xs')

Nil → Nil

toList x = Cons x Nil

1 **toList** (x):

2 v221 <- **Cons1** {}

3 v222 <- v221 @ x

4 v223 <- Nil

5 v222 @ v223

Uncurrying *map*

toList(x):

main ns = map toList ns

map f xs = case xs of

Cons x xs' →

Cons (f x) (map f xs')

Nil → Nil

toList x = Cons x Nil

1 **toList(x):**

2 *v221 <- Cons1 {}*

3 *v222 <- v221 @ x*

4 *v223 <- Nil*

5 *v222 @ v223*

Uncurrying *map*

main ns = map toList ns

map f xs = **case xs of**

Cons x xs' →

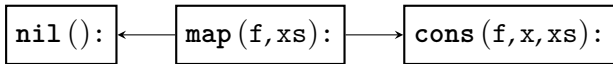
Cons (f x) (map f xs')

Nil → Nil

toList x = Cons x Nil

```
1  map (f, xs):  
2      case xs of  
3          Nil -> nil ()  
4          Cons x xs ->  
5              cons (f, x, xs)
```

Uncurrying *map*



main ns = map toList ns

map f xs = case xs of

Cons x xs' →

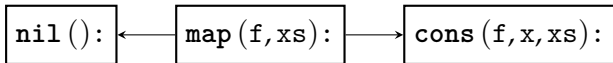
Cons (f x) (map f xs')

Nil → Nil

toList x = Cons x Nil

```
1 map (f, xs):  
2   case xs of  
3     Nil -> nil ()  
4     Cons x xs ->  
5       cons (f, x, xs)
```

Uncurrying *map*



main ns = map toList ns

map f xs = case xs of

Cons x xs' →

Cons (f x) (map f xs')

Nil → Nil

toList x = Cons x Nil

1 `cons (f, x, xs):`

2 `v209 <- Cons1 {}`

3 `v210 <- f @ x`

4 `v211 <- v209 @ v210`

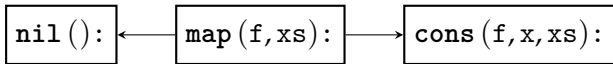
5 `v212 <- k203 {}`

6 `v213 <- v212 @ f`

7 `v214 <- v213 @ xs`

8 `v211 @ v214`

Uncurrying *map*



main ns = map toList ns

1 `nil (): Nil`

map f xs = case xs of

Cons x xs' →

Cons (f x) (map f xs')

Nil → Nil

toList x = Cons x Nil

Uncurrying *map*

`main (ns):`

`toList (x):`

`nil ():`

`map (f, xs):`

`cons (f, x, xs):`

$main\ ns = map\ toList\ ns$

$map\ f\ xs = \mathbf{case\ xs\ of}$

$Cons\ x\ xs' \rightarrow Cons\ (f\ x)\ (map\ f\ xs')$

$Nil \rightarrow Nil$

$toList\ x = Cons\ x\ Nil$

Uncurrying *map*

`main (ns):`

`toList (x):`

`nil ():`

`map (f, xs):`

`cons (f, x, xs):`

```
1  main (ns):  
2    v227 <- k203 {}  
3    v228 <- k219 {}  
4    v229 <- v227 @ v228  
5    v229 @ ns  
6    k203 {} f: k204 {f}  
7    k204 {f} xs: map (f, xs)  
8    k219 {} x: toList (x)
```


Uncurrying *map*

`main (ns):`

`toList (x):`

`nil ():`

`map (f, xs):`

`cons (f, x, xs):`

```
1  main (ns): ← {ns: T}
2    v227 <- k203 {} ← {v227: k203 {}}
3    v228 <- k219 {} ← {v228: k219 {}}
4    v229 <- v227 @ v228 ← {v229: T}
5    v229 @ ns
6  k203 {f} f: k204 {f}
7  k204 {f} xs: map (f, xs)
8  k219 {x} x: toList (x)
```

Uncurrying *map*

`main (ns):`

`toList (x):`

`nil ():`

`map (f, xs):`

`cons (f, x, xs):`

```
1  main (ns): ← {ns: T}
2    v227 <- k203 {} ← {v227: k203 {}}
3    v228 <- k219 {} ← {v228: k219 {}}
4    v229 <- v227 @ v228 ← {v229: T}
5    v229 @ ns
6  k203 {f} f: k204 {f}
7  k204 {f} xs: map (f, xs)
8  k219 {x} x: toList (x)
```

Uncurrying *map*

`main (ns):`

`toList (x):`

`nil ():`

`map (f, xs):`

`cons (f, x, xs):`

```
1  main (ns): ← {ns: T}
2    v227 <- k203 {} ← {v227: k203 {}}
3    v228 <- k219 {} ← {v228: k219 {}}
4    v229 <- k203 {} @ v228
5    v229 @ ns
6  k203 {f} f: k204 {f}
7  k204 {f} xs: map (f, xs)
8  k219 {x} x: toList (x)
```

Uncurrying *map*

`main (ns):`

`toList (x):`

`nil ():`

`map (f, xs):`

`cons (f, x, xs):`

```
1  main (ns): ← {ns: T}
2    v227 <- k203 {} ← {v227: k203 {}}
3    v228 <- k219 {} ← {v228: k219 {}}
4    v229 <- k203 {} @ v228
5    v229 @ ns
6    k203 {f: k204 {f}}
7    k204 {f} xs: map (f, xs)
8    k219 {x: toList (x)}
```

Uncurrying *map*

`main (ns):`

`toList (x):`

`nil ():`

`map (f, xs):`

`cons (f, x, xs):`

```
1  main (ns): ← {ns: T}
2    v227 <- k203 {} ← {v227:k203 {}}
3    v228 <- k219 {} ← {v228:k219 {}}
4    v229 <- k204 {v228}
5    v229 @ ns
6    k203 {f} f: k204 {f}
7    k204 {f} xs: map (f, xs)
8    k219 {x} x: toList (x)
```

Uncurrying *map*

`main (ns):`

`toList (x):`

`nil ():`

`map (f, xs):`

`cons (f, x, xs):`

```
1  main (ns): ← {ns: T}
2    v227 <- k203 {} ← {v227:k203 {}}
3    v228 <- k219 {} ← {v228:k219 {}}
4    v229 <- k204 {v228} ← {v229:k204 {v228}}
5    v229 @ ns
6  k203 {f} f: k204 {f}
7  k204 {f} xs: map (f, xs)
8  k219 {x} x: toList (x)
```

Uncurrying *map*

`main (ns):`

`toList (x):`

`nil ():`

`map (f, xs):`

`cons (f, x, xs):`

```
1  main (ns): ← {ns: T}
2    v227 <- k203 {} ← {v227: k203 {}}
3    v228 <- k219 {} ← {v228: k219 {}}
4    v229 <- k204 {v228} ← {v229: k204 {v228}}
5    v229 @ ns
6    k203 {f} k204 {f}
7    k204 {f} xs: map (f, xs)
8    k219 {x} x: toList (x)
```

Uncurrying *map*

`main (ns):`

`toList (x):`

`nil ():`

`map (f, xs):`

`cons (f, x, xs):`

```
1  main (ns): ← {ns: T}
2    v227 <- k203 {} ← {v227: k203 {}}
3    v228 <- k219 {} ← {v228: k219 {}}
4    v229 <- k204 {v228} ← {v229: k204 {v228}}
5    k204 {v228} @ ns
6    k203 {f} f: k204 {f}
7    k204 {f} xs: map (f, xs)
8    k219 {x} x: toList (x)
```


Uncurrying *map*

`main (ns):`

`toList (x):`

`nil ():`

`map (f, xs):`

`cons (f, x, xs):`

```
1  main (ns): ← {ns: T}
2    v227 <- k203 {} ← {v227: k203 {}}
3    v228 <- k219 {} ← {v228: k219 {}}
4    v229 <- k204 {v228} ← {v229: k204 {v228}}
5    k204 {v228} @ ns
6    k203 {f} f: k204 {f}
7    k204 {f} xs: map (f, xs)
8    k219 {x} x: toList (x)
```

Uncurrying *map*

`main (ns):`

`toList (x):`

`nil ():`

`map (f, xs):`

`cons (f, x, xs):`

```
1  main (ns): ← {ns: T}
2    v227 <- k203 {} ← {v227: k203 {}}
3    v228 <- k219 {} ← {v228: k219 {}}
4    v229 <- k204 {v228} ← {v229: k204 {v228}}
5    map (v228, ns)
6    k203 {f} f: k204 {f}
7    k204 {f} xs: map (f, xs)
8    k219 {x} x: toList (x)
```

Uncurrying *map*

`main (ns):`

`toList (x):`

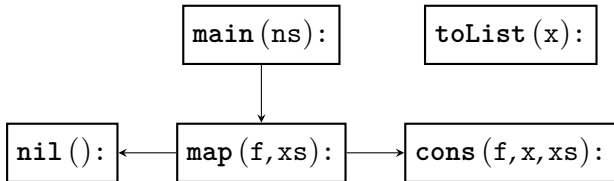
`nil ():`

`map (f, xs):`

`cons (f, x, xs):`

```
1  main (ns): ← {ns: T}
2    v227 <- k203 {} ← {v227: k203 {}}
3    v228 <- k219 {} ← {v228: k219 {}}
4    v229 <- k204 {v228} ← {v229: k204 {v228}}
5    map (v228, ns)
6  k203 {f} f: k204 {f}
7  k204 {f} xs: map (f, xs)
8  k219 {x} x: toList (x)
```

Uncurrying *map*



```
1 main (ns): ← {ns: T}
2 #1227 / KH / #203 / {}
3 v228 ← k219 {} ← {v228: k219 {}}
4 #1229 / KH / #204 / {} / #1228
5 map (v228, ns)
6 k203 {} f: k204 {f}
7 k204 {f} xs: map (f, xs)
8 k219 {} x: toList (x)
```

Uncurrying *map*

Using Dataflow
Optimization
Techniques with a
Monadic
Intermediate
Language

Justin Bailey

Introduction

MIL

Dataflow Analysis

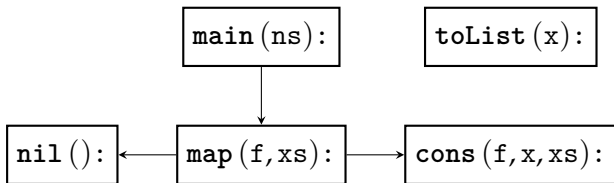
Uncurrying

Uncurrying *map*

Related Work

Conclusion

Appendix



```
1 main (ns): ← {ns: T}
2   v228 ← k219 {} ← {v228: k219 {}}
3   map (v228, ns)
4   k203 {} f: k204 {f}
5   k204 {f} xs: map (f, xs)
6   k219 {} x: toList (x)
```

Uncurrying *map*

Using Dataflow
Optimization
Techniques with a
Monadic
Intermediate
Language

Justin Bailey

Introduction

MIL

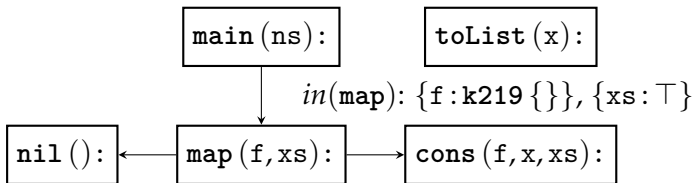
Dataflow Analysis

Uncurrying

Uncurrying *map*
Related Work

Conclusion

Appendix



```
1 main (ns):  
2   v228 <- k219 {}  
3   map (v228, ns)  
4 k203 {} f: k204 {f}  
5 k204 {f} xs: map (f, xs)  
6 k219 {} x: toList (x)
```

Uncurrying *map*

Using Dataflow
Optimization
Techniques with a
Monadic
Intermediate
Language

Justin Bailey

Introduction

MIL

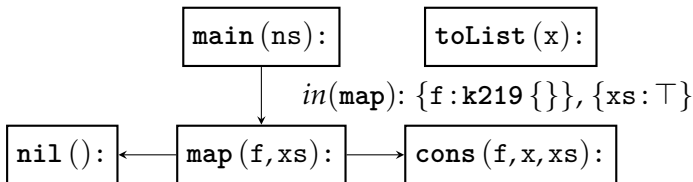
Dataflow Analysis

Uncurrying

Uncurrying *map*
Related Work

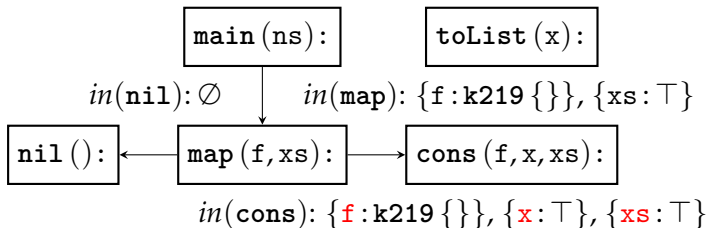
Conclusion

Appendix



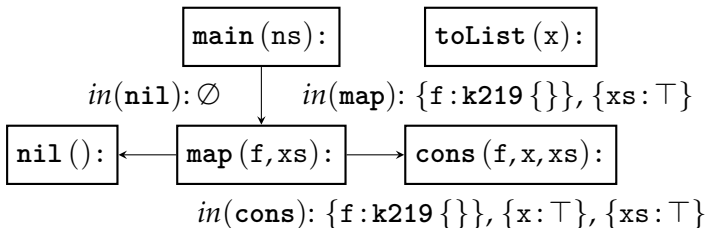
```
1  map(f, xs): ← {f:k219 {}}, {xs:T}
2    case xs of
3      Nil -> nil()
4      Cons x xs -> cons(f, x, xs)
                               ← {x:T}, {xs:T}
```

Uncurrying *map*



```
1  map (f, xs):  ← {f: k219 {}, {xs: T}
2    case xs of
3      Nil -> nil ()
4      Cons x xs -> cons (f, x, xs)
                               ← {x: T}, {xs: T}
```


Uncurrying *map*



```
1  cons (f, x, xs):  
2    v209 <- Cons1 {}  
3    v210 <- f @ x  
4    v211 <- v209 @ v210  
5    v212 <- k203 {}  
6    v213 <- v212 @ f  
7    v214 <- v213 @ xs  
8    v211 @ v214
```

Uncurrying *map*

```
1  cons (f, x, xs): ← {f:k219 {}}, {x:⊤}, {xs:⊤}
2  v209 <- Cons1 {}
3  v210 <- f @ x
4  v211 <- v209 @ v210
5  v212 <- k203 {}
6  v213 <- v212 @ f
7  v214 <- v213 @ xs
8  v211 @ v214
```

Uncurrying *map*

```
1  cons (f, x, xs): ← {f:k219 {}}, {x:T}, {xs:T}
2    v209 ← Cons1 {} ← {v209:Cons1 {}}
3    v210 ← f @ x
4    v211 ← v209 @ v210
5    v212 ← k203 {}
6    v213 ← v212 @ f
7    v214 ← v213 @ xs
8    v211 @ v214
```

Uncurrying *map*

```
1  cons (f, x, xs): ← {f:k219 {}}, {x:⊤}, {xs:⊤}
2    v209 ← Cons1 {} ← {v209:Cons1 {}}
3    v210 ← f @ x
4    v211 ← v209 @ v210
5    v212 ← k203 {}
6    v213 ← v212 @ f
7    v214 ← v213 @ xs
8    v211 @ v214
9  k219 {} x: toList (x)
```

Uncurrying *map*

```
1  cons (f, x, xs): ← {f:k219 {}}, {x:T}, {xs:T}
2    v209 <- Cons1 {} ← {v209:Cons1 {}}
3    v210 <- k219 {} @ x
4    v211 <- v209 @ v210
5    v212 <- k203 {}
6    v213 <- v212 @ f
7    v214 <- v213 @ xs
8    v211 @ v214
9  k219 {} x: toList (x)
```

Uncurrying *map*

```
1  cons (f, x, xs): ← {f:k219 {}}, {x:T}, {xs:T}
2    v209 <- Cons1 {} ← {v209:Cons1 {}}
3    v210 <- toList (x)
4    v211 <- v209 @ v210
5    v212 <- k203 {}
6    v213 <- v212 @ f
7    v214 <- v213 @ xs
8    v211 @ v214
9  k219 {} x: toList (x)
```

Uncurrying *map*

```
1  cons (f, x, xs): ← {f:k219 {}}, {x:T}, {xs:T}
2  v209 <- Cons1 {} ← {v209:Cons1 {}}
3  v210 <- toList (x) ← {v210:T}
4  v211 <- v209 @ v210
5  v212 <- k203 {}
6  v213 <- v212 @ f
7  v214 <- v213 @ xs
8  v211 @ v214
9  Cons1 {} a2: Cons2 {a2}
10 Cons2 {a2} a1: Cons a2 a1
```

Uncurrying *map*

```
1  cons (f, x, xs): ← {f:k219 {}}, {x:T}, {xs:T}
2  v209 <- Cons1 {} ← {v209:Cons1 {}}
3  v210 <- toList (x) ← {v210:T}
4  v211 <- Cons1 {} @ v210
5  v212 <- k203 {}
6  v213 <- v212 @ f
7  v214 <- v213 @ xs
8  v211 @ v214
9  Cons1 {} a2: Cons2 {a2}
10 Cons2 {a2} a1: Cons a2 a1
```


Uncurrying *map*

```
1  cons (f, x, xs): ← {f:k219 {}}, {x:T}, {xs:T}
2  v209 <- Cons1 {} ← {v209:Cons1 {}}
3  v210 <- toList (x) ← {v210:T}
4  v211 <- Cons2 {v210}
5  v212 <- k203 {}
6  v213 <- v212 @ f
7  v214 <- v213 @ xs
8  v211 @ v214
9  Cons1 {} a2: Cons2 {a2}
10 Cons2 {a2} a1: Cons a2 a1
```

Uncurrying *map*

```
1  cons (f, x, xs): ← {f:k219 {}}, {x:⊤}, {xs:⊤}
2  v209 <- Cons1 {} ← {v209:Cons1 {}}
3  v210 <- toList (x) ← {v210:⊤}
4  v211 <- Cons2 {v210} ← {v211:Cons2 {v210}}
5  v212 <- k203 {} ← {v212:k203 {}}
6  v213 <- v212 @ f ← {v213:⊤}
7  v214 <- v213 @ xs ← {v214:⊤}
8  v211 @ v214
9  k203 {f} f: k204 {f}
10 k204 {f} xs: map (f,xs)
```

Uncurrying *map*

```
1  cons (f, x, xs): ← {f:k219 {}}, {x:⊤}, {xs:⊤}
2  v209 <- Cons1 {} ← {v209:Cons1 {}}
3  v210 <- toList (x) ← {v210:⊤}
4  v211 <- Cons2 {v210} ← {v211:Cons2 {v210}}
5  v212 <- k203 {} ← {v212:k203 {}}
6  v213 <- k203 {} @ f ← {v213:⊤}
7  v214 <- v213 @ xs ← {v214:⊤}
8  v211 @ v214
9  k203 {} f: k204 {f}
10 k204 {f} xs: map (f,xs)
```

Uncurrying *map*

```
1  cons (f, x, xs): ← {f:k219 {}}, {x:⊤}, {xs:⊤}
2  v209 <- Cons1 {} ← {v209:Cons1 {}}
3  v210 <- toList (x) ← {v210:⊤}
4  v211 <- Cons2 {v210} ← {v211:Cons2 {v210}}
5  v212 <- k203 {} ← {v212:k203 {}}
6  v213 <- k204 {f} ← {v213:k204 {f}}
7  v214 <- v213 @ xs ← {v214:⊤}
8  v211 @ v214
9  k203 {} f: k204 {f}
10 k204 {f} xs: map (f, xs)
```

Uncurrying *map*

```
1  cons (f, x, xs): ← {f:k219 {}}, {x:⊤}, {xs:⊤}
2  v209 <- Cons1 {} ← {v209:Cons1 {}}
3  v210 <- toList (x) ← {v210:⊤}
4  v211 <- Cons2 {v210} ← {v211:Cons2 {v210}}
5  v212 <- k203 {} ← {v212:k203 {}}
6  v213 <- k204 {f} ← {v213:k204 {f}}
7  v214 <- v213 @ xs ← {v214:⊤}
8  v211 @ v214
9  k203 {} f: k204 {f}
10 k204 {f} xs: map (f, xs)
```

Uncurrying *map*

```
1  cons (f, x, xs): ← {f:k219 {}}, {x:⊤}, {xs:⊤}
2  v209 <- Cons1 {} ← {v209:Cons1 {}}
3  v210 <- toList (x) ← {v210:⊤}
4  v211 <- Cons2 {v210} ← {v211:Cons2 {v210}}
5  v212 <- k203 {} ← {v212:k203 {}}
6  v213 <- k204 {f} ← {v213:k204 {f}}
7  v214 <- k204 {f} @ xs ← {v214:⊤}
8  v211 @ v214
9  k203 {} f: k204 {f}
10 k204 {f} xs: map (f, xs)
```

Uncurrying *map*

```
1  cons (f, x, xs): ← {f:k219 {}}, {x:⊤}, {xs:⊤}
2  v209 <- Cons1 {} ← {v209:Cons1 {}}
3  v210 <- toList (x) ← {v210:⊤}
4  v211 <- Cons2 {v210} ← {v211:Cons2 {v210}}
5  v212 <- k203 {} ← {v212:k203 {}}
6  v213 <- k204 {f} ← {v213:k204 {f}}
7  v214 <- map (f, xs) ← {v214:⊤}
8  v211 @ v214
9  k203 {} f: k204 {f}
10 k204 {f} xs: map (f, xs)
```

Uncurrying *map*

```
1  cons (f, x, xs): ← {f:k219 {}}, {x:⊤}, {xs:⊤}
2  v209 <- Cons1 {} ← {v209:Cons1 {}}
3  v210 <- toList (x) ← {v210:⊤}
4  v211 <- Cons2 {v210} ← {v211:Cons2 {v210}}
5  v212 <- k203 {} ← {v212:k203 {}}
6  v213 <- k204 {f} ← {v213:k204 {f}}
7  v214 <- map (f, xs) ← {v214:⊤}
8  v211 @ v214
9  Cons1 {} a2: Cons2 {a2}
10 Cons2 {a2} a1: Cons a2 a1
```


Uncurrying *map*

```
1  cons (f, x, xs): ← {f:k219 {}}, {x:⊤}, {xs:⊤}
2  v209 <- Cons1 {} ← {v209:Cons1 {}}
3  v210 <- toList (x) ← {v210:⊤}
4  v211 <- Cons2 {v210} ← {v211:Cons2 {v210}}
5  v212 <- k203 {} ← {v212:k203 {}}
6  v213 <- k204 {f} ← {v213:k204 {f}}
7  v214 <- map (f, xs) ← {v214:⊤}
8  Cons2 {v210} @ v214
9  Cons1 {} a2: Cons2 {a2}
10 Cons2 {a2} a1: Cons a2 a1
```

Uncurrying *map*

```
1  cons (f, x, xs): ← {f:k219 {}}, {x:⊤}, {xs:⊤}
2  v209 <- Cons1 {} ← {v209:Cons1 {}}
3  v210 <- toList (x) ← {v210:⊤}
4  v211 <- Cons2 {v210} ← {v211:Cons2 {v210}}
5  v212 <- k203 {} ← {v212:k203 {}}
6  v213 <- k204 {f} ← {v213:k204 {f}}
7  v214 <- map (f, xs) ← {v214:⊤}
8  Cons v210 v214
9  Cons1 {} a2: Cons2 {a2}
10 Cons2 {a2} a1: Cons a2 a1
```

Uncurrying *map*

```
1  cons (f, x, xs):
2    v209 <- Cons1 {}
3    v210 <- f @ x
4    v211 <- v209 @ v210
5    v212 <- k203 {}
6    v213 <- v212 @ f
7    v214 <- v213 @ xs
8    v211 @ v214
```

Uncurrying *map*

```
1  cons (f, x, xs):
2    v209 <- Cons1 {}
3    v210 <- toList (x)
4    v211 <- Cons2 {v210}
5    v212 <- k203 {}
6    v213 <- k204 {f}
7    v214 <- map (f, xs)
8    Cons v210 v214
```

Uncurrying *map*

```
1  cons (f, x, xs):  
2      #209/KH/Cons1/N  
3      v210 <- toList (x)  
4      #211/KH/Cons2/N209/N  
5      #212/KH/203/N  
6      #213/KH/204/N  
7      v214 <- map (f, xs)  
8      Cons v210 v214
```

Uncurrying *map*

```
1  cons (f, x, xs):  
2      v210 <- toList (x)  
3      v214 <- map (f, xs)  
4      Cons v210 v214
```

Uncurrying *map*

`main (ns):`

`toList (x):`

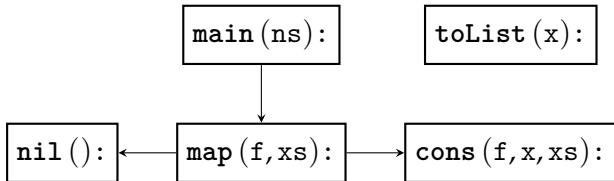
`nil ():`

`map (f, xs):`

`cons (f, x, xs):`

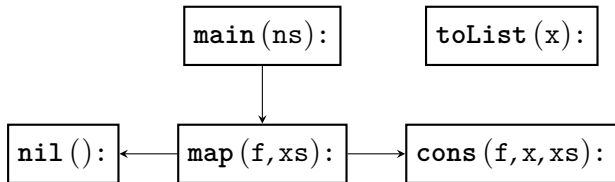
```
1  main (ns):  
2    v227 <- k203 {}  
3    v228 <- k219 {}  
4    v229 <- v227 @ v228  
5    v229 @ ns
```

Uncurrying *map*



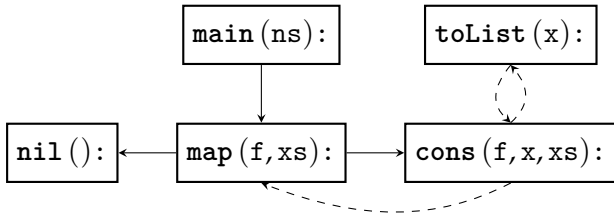
```
1  main (ns):  
2  #227/KH/#203/R/}  
3  v228 <- k219 {  
4  #229/KH/#227/Q/#228  
5  map (v228, ns)
```


Uncurrying *map*



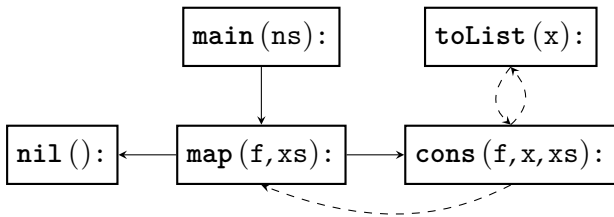
```
1  cons (f, x, xs):
2    v209 <- Cons1 {}
3    v210 <- f @ x
4    v211 <- v209 @ v210
5    v212 <- k203 {}
6    v213 <- v212 @ f
7    v214 <- v213 @ xs
8    v211 @ v214
```

Uncurrying *map*



```
1  cons (f, x, xs):  
2  #2109/KH/0013A/Y  
3  v210 <- toList (x)  
4  #211N/KH/N209/Q/N2A0  
5  #2112/KH/1203/Y  
6  #211B/KH/N212/Q/A  
7  v214 <- map (f, xs)  
8  Cons v210 v214
```

Uncurrying *map*



- ▶ One closure and two applications eliminated from **main**.
- ▶ Link from **main** to **map**.
- ▶ Two closures and four function applications from **cons**.
- ▶ Link from **cons** to **map** and **toList**.

Related Work

- ▶ Appel: Uncurrying by pattern matching; “Compiling with Continuations” (1992)
- ▶ Tarditi: Uncurrying in four passes; “Design and Implementation of Code Optimizations for a Type-Directed Compiler for Standard ML” (1996)
- ▶ Tolmach & Oliva: Automatic uncurrying; “From ML to Ada: Strongly-typed Language Interoperability via Source Translation” (1998)

Conclusion

- ▶ Monadic Optimizations
- ▶ Contributions

Using Dataflow
Optimization
Techniques with a
Monadic
Intermediate
Language

Justin Bailey

Introduction

MIL

Dataflow Analysis

Uncurrying

Conclusion

Monadic Optimizations

Contributions

Appendix

Optimizing using the Monad Laws

- ▶ *Left-Unit*

$$\mathbf{do} \{ x \leftarrow \text{return } y; m \} \equiv \mathbf{do} \{ [y \mapsto x] m \}$$

- ▶ *Right-Unit*

$$\mathbf{do} \{ x \leftarrow m; \text{return } x \} \equiv \mathbf{do} \{ m \}$$

- ▶ *Associativity*

$$\mathbf{do} \{ x \leftarrow \mathbf{do} \{ y \leftarrow m; n \}; o \} \equiv \mathbf{do} \{ y \leftarrow m; x \leftarrow n; o \}$$

- ▶ From Wadler's "Monads for Functional Programming" (1995)

Dataflow Analysis & MIL

- ▶ High-level functional programming; low-level details.
- ▶ Structured for dataflow analysis.
- ▶ Implemented other optimizations; for example, dead-code elimination.

Using Dataflow
Optimization
Techniques with a
Monadic
Intermediate
Language

Justin Bailey

Introduction

MIL

Dataflow Analysis

Uncurrying

Conclusion

Monadic Optimizations

Contributions

Appendix

Uncurrying

- ▶ Implemented using the dataflow algorithm
- ▶ Able to uncurry across blocks and loops (with some caveats)
- ▶ Complete implementation described

HOOPL

- ▶ Thorough description of the library
- ▶ Simple, but complete, example implementation given

Using Dataflow
Optimization
Techniques with a
Monadic
Intermediate
Language

Justin Bailey

Introduction

MIL

Dataflow Analysis

Uncurrying

Conclusion

Monadic Optimizations

Contributions

Appendix

Questions?

Source code and paper available at
<http://mil.codeslower.com>, or email me at
jgbailey@codeslower.com.

Using Dataflow
Optimization
Techniques with a
Monadic
Intermediate
Language

Justin Bailey

Introduction

MIL

Dataflow Analysis

Uncurrying

Conclusion

Monadic Optimizations

Contributions

Appendix

Related Work: Appel

$f\ x = g\ x$

$g\ x\ y = x + y$

$main\ a\ b = f\ a\ b$

```
1  main(a, b): b208(a, b)
2  b208(x, y):
3      v210 <- plusclo1 {x}
4      v210 @ y
5  plusclo1 {a2} a1: plus*(a2, a1)
6  f(): k212 {}
7  k212 {} x: k207 {x}
8  g(): k206 {}
9  k206 {} x: k207 {x}
10 k207 {x} y: b208(x, y)
```

Related Work: Tarditi

$g\ x\ y\ z = x + y + z$

$h\ x\ y = g\ x\ y$

$main\ s\ t\ u = h\ s\ t\ u$

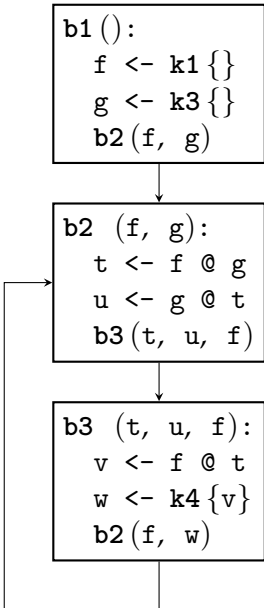
```
1  main(s, t, u): b216(s, t, u)
2  b216(x, y, z):
3      v219 <- plusclo1 {x}
4      v220 <- v219 @ y
5      v221 <- plusclo1 {v220}
6      v221 @ z
7  plusclo1 {a2} a1: plus*(a2, a1)
8  g {}: k213 {}
9  k213 {} x: k214 {x}
10 k214 {x} y: k215 {x, y}
11 h {}: k207 {}
12 k207 {} x: k208 {x}
13 k208 {x} y: k215 {x, y}
14 k215 {x, y} z: b216(x, y, z)
```

Original Program

```
b1 ():  
  f <- k1 {}  
  g <- k3 {}  
  b2(f, g)
```

```
b2 (f, g):  
  t <- f @ g  
  u <- g @ t  
  b3(t, u, f)
```

```
b3 (t, u, f):  
  v <- f @ t  
  w <- k4 {v}  
  b2(f, w)
```



Initial Facts

Using Dataflow
Optimization
Techniques with a
Monadic
Intermediate
Language

Justin Bailey

Introduction

MIL

Dataflow Analysis

Uncurrying

Conclusion

Appendix

Uncurrying Comparisons

Example: Uncurrying with
Loops

Future Work

```
b1 ():  
  f <- k1 {}  
  g <- k3 {}  
  b2(f, g)
```

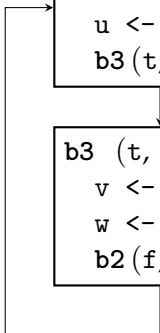
$out(b1): \{f:k1 \{\}\},$
 $\{g:k3 \{\}\}$

```
b2 (f, g):  
  t <- f @ g  
  u <- g @ t  
  b3(t, u, f)
```

$in(b2): \{f:k1 \{\}\},$
 $\{g:\top\}$

```
b3 (t, u, f):  
  v <- f @ t  
  w <- k4 {v}  
  b2(f, w)
```

$out(b3):$
 $\{g:k4 \{v\}\}$



Facts After Iteration

Using Dataflow
Optimization
Techniques with a
Monadic
Intermediate
Language

Justin Bailey

Introduction

MIL

Dataflow Analysis

Uncurrying

Conclusion

Appendix

Uncurrying Comparisons

Example: Uncurrying with
Loops

Future Work

```
b1 ():  
  f <- k1 {}  
  g <- k3 {}  
  b2(f, g)
```

$out(b1): \{f:k1 \{\}\},$
 $\{g:k3 \{\}\}$

```
b2 (f, g):  
  t <- f @ g  
  u <- g @ t  
  b3(t, u, f)
```

$in(b2): \{f:k1 \{\}\},$
 $\{g:\top\}$

```
b3 (t, u, f):  
  v <- f @ t  
  w <- k4 {v}  
  b2(f, w)
```

$in(b3): \{t:k2 \{g\}\},$
 $\{u:\top\}, \{f:k1 \{\}\}$

$out(b3):$
 $\{f:k1 \{\}\},$
 $\{g:k4 \{v\}\}$



Rewrite

Using Dataflow
Optimization
Techniques with a
Monadic
Intermediate
Language

Justin Bailey

Introduction

MIL

Dataflow Analysis

Uncurrying

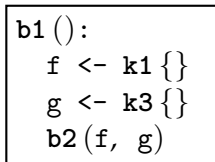
Conclusion

Appendix

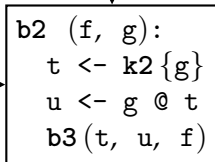
Uncurrying Comparisons

Example: Uncurrying with
Loops

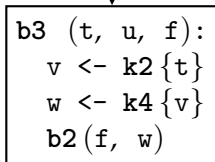
Future Work



$out(b1): \{f:k1 \{\}\},$
 $\{g:k3 \{\}\}$



$in(b2): \{f:k1 \{\}\},$
 $\{g:\top\}$



$out(b3):$
 $\{f:k1 \{\}\},$
 $\{g:k4 \{v\}\}$

$in(b3): \{t:k2 \{g\}\},$
 $\{u:\top\}, \{f:k1 \{\}\}$

Future Work

- ▶ Eliminating Thunks
- ▶ “Push Through Cases”

Using Dataflow
Optimization
Techniques with a
Monadic
Intermediate
Language

Justin Bailey

Introduction

MIL

Dataflow Analysis

Uncurrying

Conclusion

Appendix

Uncurrying Comparisons

Example: Uncurrying with
Loops

Future Work