

Selective Lambda Lifting

Sebastian Graf, sebastian.graf@kit.edu

24 June 2019

Karlsruhe Institute of Technology

Motivation

$f\ a\ 0 = a$

$f\ a\ n = f\ (g\ (n \bmod 2))\ (n-1)$

where

$g\ 0 = a$

$g\ n = 1 + g\ (n-1)$

Motivation

$$f(a, 0) = a$$

$$f(a, n) = f(g'(a, (n \bmod 2)), (n-1))$$

$$g'(a, 0) = a$$

$$g'(a, n) = 1 + g'(a, (n-1))$$

Motivation

```
f :: [Int] -> [Int] -> Int -> Int
f a b 0 = a
f a b 1 = b
f a b n = f (g n) a (n `mod` 2)
  where
    g 0 = a
    g 1 = b
    g n = n : h
      where
        h = g (n-1)
```

Motivation

```
f :: [Int] -> [Int] -> Int -> Int
f a b 0 = a
f a b 1 = b
f a b n = f (g' a b n) a (n `mod` 2)
```

```
g' a b 0 = a
g' a b 1 = b
g' a b n = n : h
  where
    h = g' a b (n-1)
```

Closure Conversion vs. Lambda Lifting

- Codegen strategies: turn local functions into global functions and auxiliary heap allocations
- Closure Conversion: References to free variables lowered as fields accesses on a closure record containing all FVs
- Lambda Lifting: Convert free variables into parameters, supplied as additional arguments at call sites

```
let f = \a b -> a*x+b*y
in f 4 2
```

$\xrightarrow{\text{CC } f}$

```
data EnvF = EnvF
  { x :: Int
    , y :: Int }
f' env a b =
  a*x env + b*y env;
let f = (f', EnvF x y)
in (fst f) (snd f) 4 2
```

Closure Conversion vs. Lambda Lifting

- Codegen strategies: turn local functions into global functions and auxiliary heap allocations
- Closure Conversion: References to free variables lowered as fields accesses on a closure record containing all FVs
- Lambda Lifting: Convert free variables into parameters, supplied as additional arguments at call sites

```
let f = \a b -> a*x+b*y  
in f 4 2
```

$$\xRightarrow{\text{LL } f}$$

```
f' x y a b = a*x + b*y;  
f' x y 4 2
```

When to lift?

When to lift?

When *not* to lift?

When to lift?

```
let f = \a b c -> a*x + b*y + z  
in g 5 x f
```

When *not* to lift?

- Argument occurrences

When to lift?

```
let f = \a b c -> a*x + b*y + z  
in g 5 x f
```

When *not* to lift?

- Argument occurrences

\Downarrow LL f

```
f' a b c = a*x + b*y + z;  
g 5 x (f' x y)
```

When to lift?

```
let f = \a b c -> a*x + b*y + z
in g 5 x f
```

When *not* to lift?

- Argument occurrences

\Downarrow LL f

```
f' a b c = a*x + b*y + z;
let f = f' x y
in g 5 x f
```

When to lift?

```
let f = \a b -> a*x + b*y
      g = \d -> f d d + x
in g 5
```

When *not* to lift?

- Argument occurrences
- Closure growth

When to lift?

When *not* to lift?

- Argument occurrences
- Closure growth

```
let f = \a b -> a*x + b*y
      g = \d -> f d d + x
in g 5
```

\Downarrow LL f

```
f' x y a b = a*x + b*y;
let g = \d -> f x y d d + x
in g 5
```

When to lift?

```
let f = \a b c d -> a*b*c*d*x*y*z  
in f 1 2 3 4
```

When *not* to lift?

- Argument occurrences
- Closure growth
- Calling convention

When to lift?

```
let f = \a b c d -> a*b*c*d*x*y*z  
in f 1 2 3 4
```

When *not* to lift?

- Argument occurrences
- Closure growth
- Calling convention

⇓ LL *f*

```
f' x y z a b c d = a*b*c*d*x*y*z;  
f' x y z 1 2 3 4
```


When to lift?

When *not* to lift?

- Argument occurrences
- Closure growth
- Calling convention
- Known calls to FVs

```
let f = \x -> 2*x
    mapF = \xs -> case xs of
        []      -> []
        x:xs'   -> f x : mapF xs'
in mapF [1..n]
```

When to lift?

When *not* to lift?

- Argument occurrences
- Closure growth
- Calling convention
- Known calls to FVs

```
let f = \x -> 2*x
    mapF = \xs -> case xs of
        []      -> []
        x:xs'   -> f x : mapF xs'
in mapF [1..n]
    ↓ LL mapF
mapF' f []      = [];
mapF' f (x:xs') = f x : mapF' f xs'
let f = \x -> 2*x
in mapF' f [1..n]
```

When to lift?

```
let p = (,) x y  
in fst p + snd p
```

When *not* to lift?

- Argument occurrences
- Closure growth
- Calling convention
- Known calls to FVs
- Sharing

When to lift?

```
let p = (,) x y
in fst p + snd p
```

When *not* to lift?

- Argument occurrences
- Closure growth
- Calling convention
- Known calls to FVs
- Sharing

```
                ↓↓ LL p
p x y = (,) x y
fst (p x y) + snd (p x y)
```

Estimating Closure Growth

Estimating Closure Growth

```
let f = \a b -> a*x + b*y  
      g = \d -> f d d + x  
in g 5
```

\Downarrow LL f

```
f' x y a b = a*x + b*y;  
let g = \d -> f x y d d + x  
in g 5
```

Estimating Closure Growth

```
let f = \a b -> a*x + b*y
      g = \d -> f d d + x
in g 5
```

- Closure alloc minus syntactic call sites?

\Downarrow LL f

```
f' x y a b = a*x + b*y;
let g = \d -> f x y d d + x
in g 5
```

Estimating Closure Growth

- Closure alloc minus syntactic call sites?

```
let f = \a b -> a*x + b*y
g = \d ->
  let h = \e -> f e e
  in h x
in g 1 + g 2 + g 3
```

LL f

```
f' x y a b = a*x + b*y;
let g = \d ->
  let h = \e -> f' x y e e
  in h x
in g 1 + g 2 + g 3
```


Estimating Closure Growth

- Closure alloc minus syntactic call sites? **X**

```
let f = \a b -> a*x + b*y
g = \d ->
  let h = \e -> f e e
  in h x
in g 1 + g 2 + g 3
```

LL f

```
f' x y a b = a*x + b*y;
let g = \d ->
  let h = \e -> f' x y e e
  in h x
in g 1 + g 2 + g 3
```

Estimating Closure Growth

- Closure alloc minus syntactic call sites? ✗
- Don't lift multi-shot occurrences?

```
let f = \a b -> a*x + b*y
g = \d ->
  let h = \e -> f e e
  in h x
in g 1 + g 2 + g 3
```

⇓ LL f

```
f' x y a b = a*x + b*y;
let g = \d ->
  let h = \e -> f' x y e e
  in h x
in g 1 + g 2 + g 3
```

Estimating Closure Growth

- Closure alloc minus syntactic call sites? ✗
- Don't lift multi-shot occurrences? ✗

```
let f = \a b -> a*x + b*y
g = \d ->
  let h1 = \e -> f e e
      h2 = \e -> f e e+x*y
  in h1 d + h2 d
```

```
in g 1 + g 2 + g 3
```

⇓ LL f

```
f' x y a b = a*x + b*y;
```

```
let g = \d ->
  let h1 = \e -> f' x y e e
      h2 = \e -> f' x y e e+x*y
  in h1 d + h2 d
in g 1 + g 2 + g 3
```

Estimating Closure Growth

- Closure alloc minus syntactic call sites? ✗
- Don't lift multi-shot occurrences? ✗
- Cost model in \mathbb{Z}_∞

```
let f = \a b -> a*x + b*y
g = \d ->
  let h1 = \e -> f e e
      h2 = \e -> f e e+x*y
  in h1 d + h2 d
```

```
in g 1 + g 2 + g 3
```

\Downarrow LL f

```
f' x y a b = a*x + b*y;
```

```
let g = \d ->
  let h1 = \e -> f' x y e e
      h2 = \e -> f' x y e e+x*y
  in h1 d + h2 d
in g 1 + g 2 + g 3
```

Estimating Closure Growth

- Closure alloc minus syntactic call sites? ✗
- Don't lift multi-shot occurrences? ✗
- Cost model in \mathbb{Z}_∞ ✓

```
let f = \a b -> a*x + b*y
g = \d ->
  let h1 = \e -> f e e
      h2 = \e -> f e e+x*y
  in h1 d + h2 d
```

```
in g 1 + g 2 + g 3
```

\Downarrow LL f

```
f' x y a b = a*x + b*y;
```

```
let g = \d ->
  let h1 = \e -> f' x y e e
      h2 = \e -> f' x y e e+x*y
  in h1 d + h2 d
in g 1 + g 2 + g 3
```

Evaluation – Against Baseline

Program	Bytes allocated	Runtime
cryptarithm1	-2.8%	-8.0%
grep	-6.7%	-4.3%
lambda	-0.0%	-13.5%
mate	-8.4%	-3.1%
minimax	-1.1%	+3.8%
n-body	-20.2%	-0.0%
queens	-18.0%	-0.5%
<i>... and 98 more</i>		
Min	-20.2%	-13.5%
Max	0.0%	+3.8%
Geometric Mean	-0.9%	-0.7%

Figure 1: GHC baseline vs. late lambda lifting

Evaluation – Closure Growth Heuristic

Program	Bytes allocated	Runtime
eliza	-2.6%	+2.4%
grep	-7.2%	-3.1%
integrate	+0.4%	+4.1%
lift	-4.1%	-2.5%
paraffins	+17.0%	+3.7%
prolog	-5.1%	-2.8%
wheel-sieve1	+31.4%	+3.2%
<i>... and 98 more</i>		
Min	-7.2%	-3.1%
Max	+31.4%	+4.8%
Geometric Mean	+0.4%	-0.0%

Figure 2: Late lambda lifting with vs. without closure growth heuristic

Variables	$f, g, x, y \in \text{Var}$	
Expressions	$e \in \text{Expr} ::= x$	Variable
	$::= f \bar{x}$	Function call
	$::= \text{let } b \text{ in } e$	Recursive let
Bindings	$b \in \text{Bind} ::= \overline{f = r}$	
Right-hand sides	$r \in \text{Rhs} ::= \lambda \bar{x} \rightarrow e$	
Programs	$p \in \text{Prog} ::= \overline{f \bar{x} = e; e'}$	

Figure 3: An STG-like untyped lambda calculus

Formally Estimating Closure Growth

$$\boxed{\text{cl-gr}_{-}^{\varphi}(\cdot): \mathcal{P}(\text{Var}) \rightarrow \mathcal{P}(\text{Var}) \rightarrow \text{Expr} \rightarrow \mathbb{Z}_{\infty}}$$

$$\text{cl-gr}_{\varphi^{-}}^{\varphi^{+}}(x) = 0 \quad \text{cl-gr}_{\varphi^{-}}^{\varphi^{+}}(f \bar{x}) = 0$$

$$\text{cl-gr}_{\varphi^{-}}^{\varphi^{+}}(\text{let } bs \text{ in } e) = \text{cl-gr}_{\varphi^{-}}^{\varphi^{+}}(\text{bind}(bs) + \text{cl-gr}_{\varphi^{-}}^{\varphi^{+}}(e))$$

$$\boxed{\text{cl-gr-bind}_{-}^{\varphi}(\cdot): \mathcal{P}(\text{Var}) \rightarrow \mathcal{P}(\text{Var}) \rightarrow \text{Bind} \rightarrow \mathbb{Z}_{\infty}}$$

$$\text{cl-gr-bind}_{\varphi^{-}}^{\varphi^{+}}(\overline{f = r}) = \sum_i \text{growth}_i + \text{cl-gr-rhs}_{\varphi^{-}}^{\varphi^{+}}(r_i) \quad \nu_i = |\text{fvs}(f_i) \cap \varphi^{-}|$$

$$\text{growth}_i = \begin{cases} |\varphi^{+} \setminus \text{fvs}(f_i)| - \nu_i, & \text{if } \nu_i > 0 \\ 0, & \text{otherwise} \end{cases}$$

$$\boxed{\text{cl-gr-rhs}_{-}^{\varphi}(\cdot): \mathcal{P}(\text{Var}) \rightarrow \mathcal{P}(\text{Var}) \rightarrow \text{Rhs} \rightarrow \mathbb{Z}_{\infty}}$$

$$\text{cl-gr-rhs}_{\varphi^{-}}^{\varphi^{+}}(\lambda \bar{x} \rightarrow e) = \text{cl-gr}_{\varphi^{-}}^{\varphi^{+}}(e) * [\sigma, \tau] \quad n * [\sigma, \tau] = \begin{cases} n * \sigma, & n < 0 \\ n * \tau, & \text{otherwise} \end{cases}$$

$$\sigma = \begin{cases} 1, & e \text{ entered at least once} \\ 0, & \text{otherwise} \end{cases} \quad \tau = \begin{cases} 0, & e \text{ never entered} \\ 1, & e \text{ entered at most once} \\ 1, & \text{RHS bound to a thunk} \\ \infty, & \text{otherwise} \end{cases}$$

Using Closure Growth

Allow to lift **let** $\overline{g = \lambda \bar{x} \rightarrow e}$ **in** e' when

$$\text{cl-gr}_{\{\bar{g}\}}^{\alpha'(g_1)}(\text{let } \overline{g = \lambda \alpha'(g_1) \bar{x} \rightarrow e} \text{ in } e') - \sum_i 1 + |\text{fvs}(g_i) \setminus \{\bar{g}\}|$$

is non-positive

Lambda Lifting

$$\boxed{\text{lift_}(_): \text{Expander} \rightarrow \text{Expr} \rightarrow \text{Expr}}$$

$$\text{lift}_\alpha(x) = \begin{cases} x, & x \notin \text{dom } \alpha \\ x \ \alpha(x), & \text{otherwise} \end{cases} \quad \text{lift}_\alpha(f \ \bar{x}) = \text{lift}_\alpha(f) \ \bar{x}$$

$$\text{lift}_\alpha(\text{let } bs \text{ in } e) = \begin{cases} \text{lift}_{\alpha'}(e), & bs \text{ is to be lifted as } \text{lift_bind}_{\alpha'}(bs) \\ \text{let lift_bind}_\alpha(bs) \text{ in lift}_\alpha(e) & \text{otherwise} \end{cases}$$

where

$$\alpha' = \text{add_rqs}(bs, \alpha)$$

$$\boxed{\text{add_rqs}(_, _): \text{Bind} \rightarrow \text{Expander} \rightarrow \text{Expander}}$$

$$\text{add_rqs}(\overline{f = r}, \alpha) = \alpha \ [\overline{f} \mapsto \text{rqs}]$$

where

$$\text{rqs} = \bigcup_i \text{expand}_\alpha(\text{fvs}(r_i)) \setminus \{ \overline{f} \}$$

$$\boxed{\text{expand_}(_): \text{Expander} \rightarrow \mathcal{P}(\text{Var}) \rightarrow \mathcal{P}(\text{Var})}$$

$$\text{expand}_\alpha(V) = \bigcup_{x \in V} \begin{cases} \{x\}, & x \notin \text{dom } \alpha \\ \alpha(x), & \text{otherwise} \end{cases}$$

$$\boxed{\text{lift_bind_}(_): \text{Expander} \rightarrow \text{Bind} \rightarrow \text{Bind}}$$

$$\text{lift_bind}_\alpha(\overline{f = \lambda \bar{x} \rightarrow e}) = \begin{cases} \overline{f = \lambda \bar{x} \rightarrow \text{lift}_\alpha(e)} & f_1 \notin \text{dom } \alpha \\ \overline{f = \lambda \alpha(f) \bar{x} \rightarrow \text{lift}_\alpha(e)} & \text{otherwise} \end{cases}$$

