Selective Lambda Lifting

Sebastian Graf, `sebastian.graf@kit.edu`

24 June 2019

Karlsruhe Institute of Technology

Motivation

```haskell
f a 0 = a
f a n = f (g (n `mod` 2)) (n-1)
  where
    g 0 = a
    g n = 1 + g (n-1)
```

Motivation

```haskell
f a 0 = a
f a n = f (g' a (n `mod` 2)) (n-1)

g' a 0 = a
g' a n = 1 + g' a (n-1)
```

Motivation

```haskell
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

```haskell
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?

When *not* to lift?

- Argument occurrences

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

$$\Big\Downarrow \text{LL } f$$

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

When to lift?

When *not* to lift?

- Argument occurrences

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

$$\Big\Downarrow \text{LL } f$$

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

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
```

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
```

$$\Big\Downarrow \text{LL } f$$

```
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?

When *not* to lift?

- Argument occurrences
- Closure growth
- Calling convention

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

$$\Big\Downarrow \text{ LL } f$$

```
f' a b c d = a*b*c*d*x*y*z;
in 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]
```

$\Downarrow$ LL *mapF*

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

When to lift?

```haskell
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?

When *not* to lift?

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

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

$\Downarrow$ 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
```

$$\Big\Downarrow \text{LL } f$$

```
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?

$$\Downarrow \text{LL } f$$

```
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
```

$$\Big\Downarrow \text{LL } f$$

```
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? ✗

```
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
```

$$\Big\Downarrow \text{LL } f$$

```
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
```

$$\Big\Downarrow \text{LL } f$$

```
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
```

$$\Downarrow \text{LL } f$$

```
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 $\mathbb{Z}_\infty$

```
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
```

$$\Downarrow \text{LL } f$$

```
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 $\mathbb{Z}_\infty$ ✓

```
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%
... and 98 more		
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%
... and 98 more		
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

Syntax

$$
\begin{array}{rrll}
\text{Variables} & f, g, x, y \in \mathrm{Var} & & \\
\text{Expressions} & e \in \mathrm{Expr} & ::= \ x & \text{Variable} \\
 & & ::= |f \ \overline{x} & \text{Function call} \\
 & & ::= |\mathbf{let}\ b\ \mathbf{in}\ e & \text{Recursive } \mathbf{let} \\
\text{Bindings} & b \in \mathrm{Bind} & ::= \ \overline{f = r} & \\
\text{Right-hand sides} & r \in \mathrm{Rhs} & ::= \ \lambda \overline{x} \to e & \\
\text{Programs} & p \in \mathrm{Prog} & ::= \ \overline{f\ \overline{x} = e;}\ e' &
\end{array}
$$

Figure 3: An STG-like untyped lambda calculus

Formally Estimating Closure Growth

$$\boxed{\text{cl-gr}^{\_}_{\_}(\_)\colon \mathcal{P}(\text{Var}) \to \mathcal{P}(\text{Var}) \to \text{Expr} \to \mathbb{Z}_\infty}$$

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

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

$$\boxed{\text{cl-gr-bind}^{\_}_{\_}(\_)\colon \mathcal{P}(\text{Var}) \to \mathcal{P}(\text{Var}) \to \text{Bind} \to \mathbb{Z}_\infty}$$

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

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

$$\boxed{\text{cl-gr-rhs}^{\_}_{\_}(\_)\colon \mathcal{P}(\text{Var}) \to \mathcal{P}(\text{Var}) \to \text{Rhs} \to \mathbb{Z}_\infty}$$

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

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

Using Closure Growth

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

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

is non-positive

Lambda Lifting

$$\boxed{\mathrm{lift}\_(\_) \colon \mathrm{Expander} \to \mathrm{Expr} \to \mathrm{Expr}}$$

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

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

where

$$\alpha' = \text{add-rqs}(bs, \alpha)$$

$$\boxed{\text{add-rqs}(\_,\_) \colon \mathrm{Bind} \to \mathrm{Expander} \to \mathrm{Expander}}$$

$$\text{add-rqs}(\overline{f = r}, \alpha) = \alpha\ \left[\overline{f \mapsto \mathrm{rqs}}\right]$$

where

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

$$\boxed{\mathrm{expand}\_(\_) \colon \mathrm{Expander} \to \mathcal{P}(\mathrm{Var}) \to \mathcal{P}(\mathrm{Var})}$$

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

$$\boxed{\text{lift-bind}\_(\_) \colon \mathrm{Expander} \to \mathrm{Bind} \to \mathrm{Bind}}$$

$$\text{lift-bind}_\alpha(\overline{f = \lambda \overline{x} \to e}) = \begin{cases} \overline{f = \lambda \overline{x} \to \mathrm{lift}_\alpha(e)} & f_1 \notin \mathrm{dom}\,\alpha \\ \overline{f = \lambda \alpha(f)\,\overline{x} \to \mathrm{lift}_\alpha(e)} & \text{otherwise} \end{cases}$$

Filler