CS653: Functional Programming
2017-18 II^{nd} Semester

Haskell to Enriched λ Calculus

Amey Karkare
karkare@cse.iitk.ac.in

http://www.cse.iitk.ac.in/~karkare/cs653
Department of CSE, IIT Kanpur

Agenda

- Translating Haskell to Enriched λ calculus
  - Examples
  - Translation of List Comprehension
- Compiling enriched λ calculus

Example 1

```
length [] = 0
length (x:xs) = 1 + length xs
```

push patterns to the right

```
length l = case l of
                  [] -> 0
                  (x:xs) -> 1 + length xs
```

replace multiple alts by (pattern, _) pairs

```
length l = case l of
                  [] -> 0
                  _ -> case l of
                           (x:xs) - 1 + length xs
                           _ -> error "insufficient patterns"
```

0-ary constructor rule

```
length l = if (l==[]) then 0
           else case l of
                  (x:xs) - 1 + length xs
                  _ -> error "insufficient patterns"
```

refutable pattern matching

```
length l = if (l==[]) then 0
           else (UNPACK_SUM_:  (λxλxs -> 1 + length xs) l)
                 [] ERROR
```

remove recursion

```
length = Y (λlenλl.if (l==[]) then 0
                   else (UNPACK_SUM_:  (λxλxs -> 1 + len xs) l)
                         [] ERROR)
```

Example 2

```
data Bintree a = Node (Bintree a) (Bintree a) | Leaf a
reflect (Leaf n) = Leaf n
reflect (Branch t1 t2) = Branch(reflect t1)(reflect t2)
```

---

push patterns to the right

```
reflect t = case t of
              (Leaf n)->Leaf n
              (Branch t1 t2)->Branch(reflect t1)(reflect t2)
```

---

replace multiple alts by (pattern, _) pairs

```
reflect t = case t of
              (Leaf n)->Leaf n
              _ ->case t of
                     (Branch t1 t2)->Branch(reflect t1)(reflect t2)
                     _->error
```

---

refutable pattern matching (twice)

```
reflect t = UNPACK_SUM_Leaf (λn->Leaf n) t
        [] UNPACK_SUM_Branch(λt1λt2->Branch(reflect t1)(reflect t2)) t
        [] error
```

---

remove recursion

```
reflect = Y (λrλt->UNPACK_SUM_Leaf (λn->Leaf n) t
                   [] UNPACK_SUM_Branch (λt1λt2->Branch (r t1) (r t2)) t
                   [] error)
```

Example 3

```
f ~(x,y) = x
```

push patterns to the right

```
f z = case z of
        ~(x,y)->x
```

refutable pattern matching involving product constructors

f z = UNPACK_PROD_pair (λxλy->x) z

simplify

f = UNPACK_PROD_pair (λxλy->x)

Example 4

```
let          -- actually a letrec
    length [] = 0
    length (x:xs) = 1 + length xs
in length [1,2,3]
```

---

remove patterns and recursion

```
let
    length = Y (λlenλl->if (l == []) then 0
                   else UNPACK_SUM_:  (λxλxs->1 + len xs) l)
                        [] error
in length [1,2,3]
```

---

translate to a lambda

```
(λlength->length [1,2,3])
    (Y (λlen λl->(if (l == []) then 0
              else UNPACK_SUM_:  (λxλxs->1 + len xs) l)
                  [] error))
```

Example 5

```
let             -- actually a letrec
    x = 1:y
    y = 2:x
in x
```

convert to single definition letrec

```
letrec
    (x,y) = (1:y, 2:x)
in x
```

remove recursion

```
letrec
    (x,y) = (λ~(x,y)->(1:y, 2:x)) (x, y)
in x

let
    (x,y) = Y (λ~(x,y)->(1:y, 2:x))
in x
```

convert let into case

```
case Y (λ~(x,y)->(1:y, 2:x)) of
    (x,y)->x
```

refutable pattern matching with product constructors

UNPACK_PROD_Pair `(λx'λy'->x') (Y (λ~(x,y)->(1:y, 2:x)))`

Example 5

Thus,

```
let
    x = 1:y
    y = 2:x
in x
```

is translated to:

UNPACK_PROD_Pair $(\lambda x'\lambda y'\text{->}x')$ $(Y\ (\lambda\sim(x,y)\text{->}(1:y,\ 2:x)))$

To see that it works, denote $\lambda\sim(x,y)\text{->}(1:y,\ 2:x)$ as F. Then,

$$
\begin{aligned}
& Y\ F \\
&= F\ (Y\ F) \\
&= \lambda\sim(x,y)\text{->}(1:y,\ 2:x)\ (Y\ F) \\
&= (\lambda x'\lambda y'\text{->}(1:y',\ 2:x'))\ (\mathrm{SEL}^1_{pair}(Y\ F))\ (\mathrm{SEL}^2_{pair}(Y\ F)) \\
&= (1:(\mathrm{SEL}^2_{pair}(Y\ F)),\ 2:(\mathrm{SEL}^1_{pair}(Y\ F)))
\end{aligned}
$$

Therefore

UNPACK_PROD_Pair $(\lambda x'\lambda y'\text{->}x)$ $(Y\ (\lambda\sim(x,y)\text{->}(1:y,\ 2:x)))$

```
= (1:(case Y F of (x,y)->y))
= (1:2:(case Y F of (x,y)->x))
= (1:2:1:(case Y F of (x,y)->y))
...
```

Handling List Comprehensions

Example LC-1

```
[sqr x | x <- [1,2,3] ]
```

Example LC-2

```
[x + y | x <- [1,2,3], y <- [4,5] ]
```

Example LC-3

```
[sqr x | x <- [1,2,3], odd x ]
```

Example LC-4

```
[sqr x | (2, x) <- [(1,4), (2, 5) , (3, 6)] ]
```

Rules for De-sugaring List Comprehension

- `[e | ] = [e]`
- `[e | v<-[], Qs] = []`
- `[e | v<-(H:T), Qs]`
  `= [e | Qs][H/v] ++ [e | v<-T, Qs]`
- `[e | gd, Qs] = if gd then [e | Qs] else [ ]`

Reducing the number of “unpack” functions