Haskell Internals

Ramkumar Ramachandra

FOSS.IN/2009

01 December 2009

Outline

1. A Gentle Introduction to Haskell
2. Part I: Thinking in Haskell
3. Part II: A peek into GHC
4. Conclusion

Why Haskell? What's in it for you?

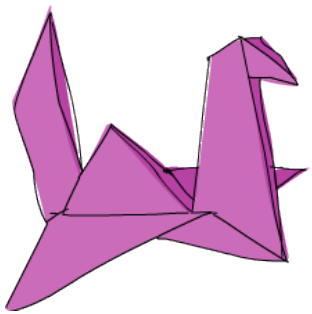
- Theoretical interest
- Ideas to apply in other places
- Real-world applications
- Concurrency: STM

Solve a simple problem imperatively

PE 5: What is the smallest number divisible by each of the numbers 1 to 20?

```
1  lcm_store = 1;
2  for(i = 1; i <= 20; i ++) {
3      lcm_store = lcm (lcm_store, i);
4  }
```

Re-think the problem in terms of folds

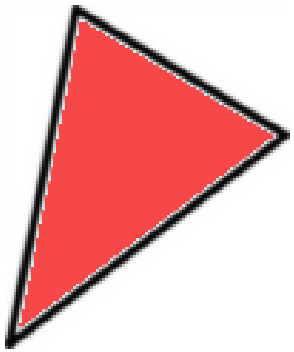


```
foldr :: (a -> b -> b) -> b -> [a] -> b
```

```
1  euler5 :: (Integral a) => a
2  euler5 = foldr lcm 1 [1..20]
3      where gcd a 0 = a
4            gcd a b = gcd b (a `mod` b)
5            lcm a b = (a*b) `div` gcd a b
```

Pick a more challenging problem

What is the first triangle number to have over 500 divisors?



```
10: 1,2,5,10
15: 1,3,5,15
21: 1,3,7,21
28: 1,2,4,7,14,28
```

```
28 = 2^2 + 7^1
(2+1) * (1+1) = 6 divisors
```

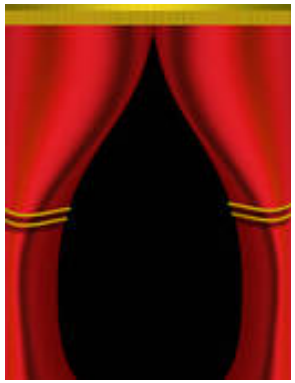
Solve it in Haskell

```
filter :: (a -> Bool) -> [a] -> [a]
```

```
map :: (a -> b) -> [a] -> [b]
```

```
1  euler12 :: (Integral a) => a
2  euler12 = head $ filter ((> 500) . n_divisors) triangleSeries
3      where triangleSeries = [div (n * (n + 1)) 2 | n <- [1..]]
4            n_divisors n = product . map ((+1) . length) . primeGroups $ n
5            primeGroups = group . (primeFactors n) . filterPrimes
6            filterPrimes n = filter (\x -> n `mod` x == 0) primes
```

Behind the scenes



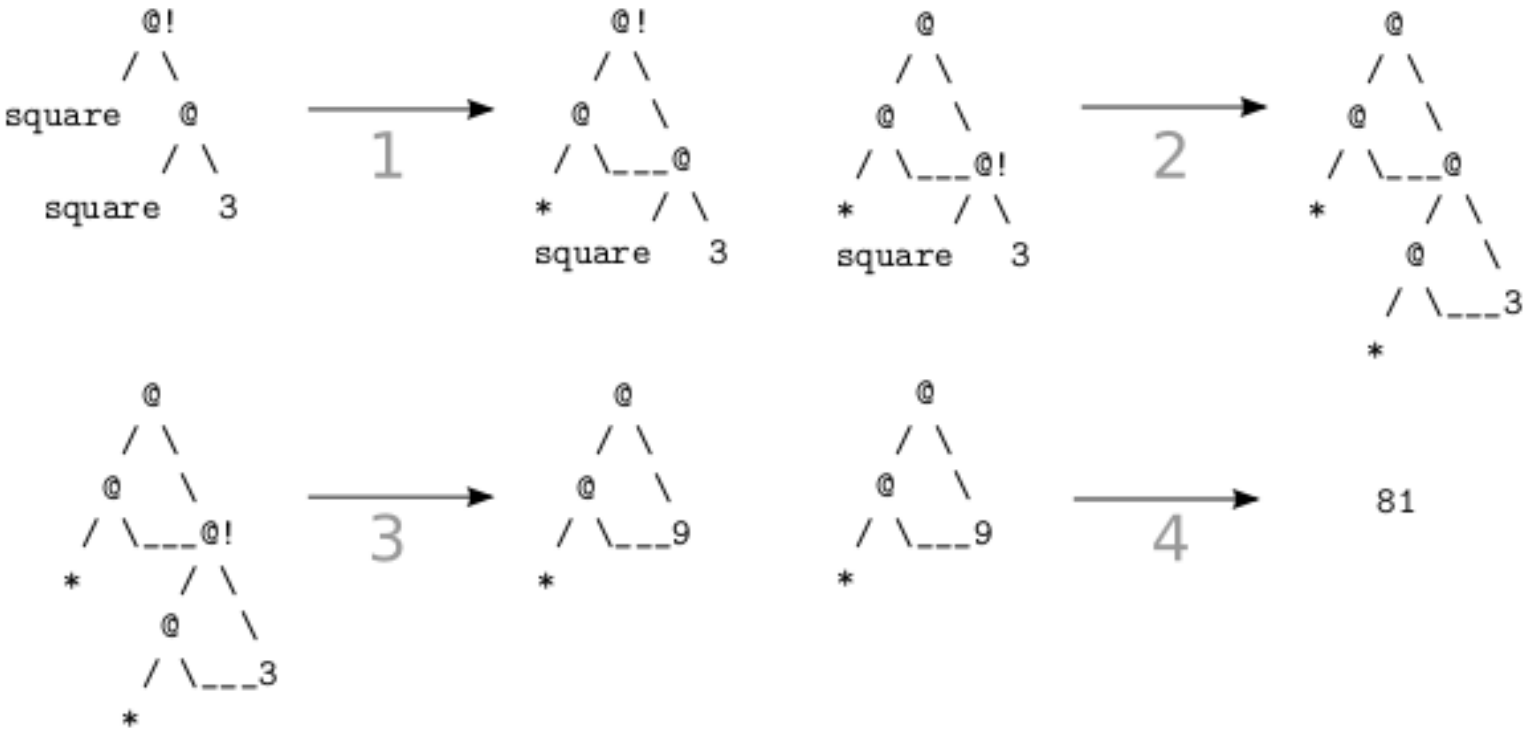
- Glasgow Haskell Compiler
- Parse everything into Core Language
- Use graph reduction
- Apply optimizations
- Compile Core Language into native code via GCC

What the core language looks like

- Local defintions
- Lexical closures provided by let/ letrec
- case for pattern matching
- Local function definitions (lambda abstractions)
- Structured data types provided by Pack

Apply graph reduction to the core language

```
square x = x * x ;
main = square (square 3)
```



Why bother with laziness

```
euler14 :: Integer
-- Stack overflow!
euler14 = foldl1 (pick_larger chain_length) l
    -- [2, 3 .. 999999]
    where chain_length = length . collatz_chain
```

```
euler14 = foldl1 (pick_larger snd) collatzip
    -- [(2,2),(3,8),(4,3),(5,6),(6,9),(7,17)]
    where collatzip = zip l chain_length
```

A deeper look into the compiler

- G-Machine compiler
- TIM compiler
- Parallel G-machine compiler
- Lambda lifter

References

[1] Augustsson, L., and Johnsson, T. Parallel graph reduction with the (v , g)-machine. In *FPCA '89: Proceedings of the fourth international conference on Functional programming languages and computer architecture* (New York, NY, USA, 1989), ACM, pp. 202–213.

[2] Johnsson, T. Efficient compilation of lazy evaluation. *SIGPLAN Not. 39*, 4 (2004), 125–138.

[3] Jones, S. L. P., and Lester, D. R. *The Implementation of Functional Programming Languages*. Prentice Hall, 1987.

[4] Jones, S. L. P., and Lester, D. R. *Impelementing Functional Languages: A Tutorial*. Prentice Hall, 1992.

[5] Terei, D. A. Low level virtual machine for glasgow haskell compiler, 2009. A Bachelor Thesis.

Contact information

Ramkumar Ramachandra
artagnon@gmail.com
http://artagnon.com
Indian Institute of Technology, Kharagpur
Presentation source available on http://github.com/artagnon/foss.in