Functional Programming

WS 2010/11

Christian Sternagel (VO)
Friedrich Neurauter (PS) Ulrich Kastlunger (PS)

Computational Logic
Institute of Computer Science
University of Innsbruck

January 19, 2011

Evaluation

LVA-Code

703017-0

Additional Questions

1. I can solve simple problems using Haskell.
2. The number of weekly exercises was to low.
3. The lecture notes (PDFs) were generally helpful.
4. There was too little theory.

Today's Topics

- Lazyness
- Fibonacci Numbers
- The Sieve of Eratosthenes

Lazyness

Motivation

Idea

- only compute values needed for final result
- avoid computing the same value twice (memoization)

Motivation

Idea

- only compute values needed for final result
- avoid computing the same value twice (memoization)

Example

- in the program

```
f1 x = x + 1
f2 x = f2 x -- nonterminating
main = do
  input <- getLine
  let i = read input
  print (head [f1 i, f2 i])
```

- the value of `f2 i` is not needed
- however, without lazyness the whole program would not terminate

Memory Leaks

Program (Tail Recursive)

```
length' acc []       =  acc
length' acc (_:xs)   =  length' (acc+1) xs
```

Memory Leaks

Program (Tail Recursive)

```
length' acc []       =  acc
length' acc (_:xs)   =  length' (acc+1) xs
```

Evaluation

```
length' 0 [1,2,3,4]
  =  length' (0+1) [2,3,4]
  =  length' (0+1+1) [3,4]
  =  length' (0+1+1+1) [4]
  =  length' (0+1+1+1+1) []
  =  (0+1+1+1+1)
```

Forcing Strictness

The `seq` Function

- type `seq :: a -> b -> b`
- `x `seq` y` first forces to evaluate `x` strictly
- and then returns `y`

Forcing Strictness

The seq Function

- type `seq :: a -> b -> b`
- `x `seq` y` first forces to evaluate `x` strictly
- and then returns `y`

Program (Still Tail Recursive)

```
length'' acc []       =  acc
length'' acc (_:xs)   =  acc `seq` length'' (acc+1) xs
```

Fibonacci Numbers

Recall

Definition (i-th Fibonacci number F_i)

$$F_i = \begin{cases} 0 & \text{if } i = 0 \\ 1 & \text{if } i = 1 \\ F_{i-1} + F_{i-2} & \text{otherwise} \end{cases}$$

Recall

Definition (i-th Fibonacci number F_i)

$$F_i = \begin{cases} 0 & \text{if } i = 0 \\ 1 & \text{if } i = 1 \\ F_{i-1} + F_{i-2} & \text{otherwise} \end{cases}$$

Sequence

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 ...

Idea

Visualization

starting at 0 0 1

Idea

Visualization

starting at 0	0	1
starting at 1	1	

Idea

Visualization

starting at 0	0	1
starting at 1	1	
$(+)$		

Idea

Visualization

starting at 0	0	1
starting at 1	1	
$(+)$		

Idea

Visualization

starting at 0	0	1
starting at 1	1	
$(+)$	1	

Idea

Visualization

starting at 0	0	1	1
starting at 1	1	1	
$(+)$	1		

Idea

Visualization

starting at 0	0	1	1
starting at 1	1	1	
$(+)$	1		

Idea

Visualization

starting at 0	0	1	1
starting at 1	1	1	
$(+)$	1	2	

Idea

Visualization

starting at 0	0	1	1	2
starting at 1	1	1	2	
$(+)$	1	2		

Idea

Visualization

starting at 0	0	1	1	2
starting at 1	1	1	2	
$(+)$	1	2		

Idea

Visualization

starting at 0	0	1	1	2
starting at 1	1	1	2	
$(+)$	1	2	3	

Idea

Visualization

starting at 0	0	1	1	2	3
starting at 1	1	1	2	3	
$(+)$	1	2	3		

Idea

Visualization

starting at 0	0	1	1	2	3
starting at 1	1	1	2	3	
$(+)$	1	2	3		

Idea

Visualization

starting at 0	0	1	1	2	3
starting at 1	1	1	2	3	
$(+)$	1	2	3	5	

Idea

Visualization

starting at 0	0	1	1	2	3	5
starting at 1	1	1	2	3	5	
$(+)$	1	2	3	5		

Idea

Visualization

starting at 0	0	1	1	2	3	5
starting at 1	1	1	2	3	5	
$(+)$	1	2	3	5		

Idea

Visualization

starting at 0	0	1	1	2	3	5
starting at 1	1	1	2	3	5	
$(+)$	1	2	3	5	8	

Idea

Visualization

starting at 0	0	1	1	2	3	5	8
starting at 1	1	1	2	3	5	8	
$(+)$	1	2	3	5	8		

Idea

Visualization

starting at 0	0	1	1	2	3	5	8
starting at 1	1	1	2	3	5	8	
$(+)$	1	2	3	5	8		

Idea

Visualization

starting at 0	0	1	1	2	3	5	8
starting at 1	1	1	2	3	5	8	
$(+)$	1	2	3	5	8	13	

Idea

Visualization

starting at 0	0	1	1	2	3	5	8	13
starting at 1	1	1	2	3	5	8	13	
$(+)$	1	2	3	5	8	13		

Idea

Visualization

starting at 0	0	1	1	2	3	5	8	13
starting at 1	1	1	2	3	5	8	13	
$(+)$	1	2	3	5	8	13		

Idea

Visualization

starting at 0	0	1	1	2	3	5	8	13
starting at 1	1	1	2	3	5	8	13	
$(+)$	1	2	3	5	8	13	21	

Idea

Visualization

starting at 0	0	1	1	2	3	5	8	13	21
starting at 1	1	1	2	3	5	8	13	21	
$(+)$	1	2	3	5	8	13	21		

Idea

Visualization

starting at 0	0	1	1	2	3	5	8	13	21
starting at 1	1	1	2	3	5	8	13	21	
$(+)$	1	2	3	5	8	13	21		

Idea

Visualization

starting at 0	0	1	1	2	3	5	8	13	21
starting at 1	1	1	2	3	5	8	13	21	
$(+)$	1	2	3	5	8	13	21	34	

Idea

Visualization

starting at 0	0	1	1	2	3	5	8	13	21	...
starting at 1	1	1	2	3	5	8	13	21	...	
$(+)$	1	2	3	5	8	13	21	34	...	

Idea

Visualization

starting at 0	0	1	1	2	3	5	8	13	21	...
starting at 1	1	1	2	3	5	8	13	21	...	
$(+)$	1	2	3	5	8	13	21	34	...	

Missing

- function to shift sequence to the left

Idea

Visualization

starting at 0	0	1	1	2	3	5	8	13	21	...
starting at 1	1	1	2	3	5	8	13	21	...	
$(+)$	1	2	3	5	8	13	21	34	...	

Missing

- function to shift sequence to the left
- function to add two sequences

Fibonacci Numbers in Haskell

```haskell
fibs :: [Integer]
fibs = 0 : 1 : zipWith (+) fibs (tail fibs)
```

The Sieve of Eratosthenes

The Sieve of Eratosthenes

Algorithm

start with list of all natural numbers (from 2 on)

1. mark first element x as prime
2. remove all multiples of x
3. go to Step 1

The Sieve in Haskell

```haskell
primes :: [Integer]
primes = sieve [2 ..]
  where
    sieve (x:xs) =
      x : sieve [y | y <- xs, y `mod` x /= 0]
```

Exercises (for January 29th)

1. Consider the λ-term $t = (\lambda xyz.\, x\ z\ (y\ z))\ (\lambda xy.\, x)\ (\lambda x.\, x)\ (\lambda x.\, x)$. Reduce t stepwise to NF using the leftmost innermost/outermost strategy.
2. Consider the Haskell type `data Tree = E | N Tree Int Tree` together with the functions:

```
preorder E           = []
preorder (N l x r) = x:(preorder l ++ preorder r)

sumTree E           = 0
sumTree (N l x r) = x + sumTree l + sumTree r

sum []      = 0
sum (x:xs) = x + sum xs
```

Prove by induction that `sum` (`preorder` t) $=$ `sumTree` t. You may use `sum` (xs `++` ys) $=$ `sum` xs $+$ `sum` ys.

Exercises (continued)

Consider the Haskell functions:

```
f x = if x `div` 2 == 0 then 0
                        else 1 + f (x `div` 2)
g x = if x < 2 then 1
               else g (x-1) + 2 * g (x-2)
```

3. Give a tail recursive variant of `f`.
4. Use tupling to implement a more efficient variant of `g`.

Exercises (continued)

5. Consider the typing environment:

$$
\begin{aligned}
E \quad = \{ & 1 :: \mathsf{Int}, 2 :: \mathsf{Int}, \mathsf{Cons} :: \alpha_0 \to \mathsf{List}(\alpha_0) \to \mathsf{List}(\alpha_0), \\
& \mathsf{head} :: \mathsf{List}(\alpha_0) \to \alpha_0, \mathsf{Nil} :: \mathsf{List}(\alpha_0), \\
& \mathsf{tail} :: \mathsf{List}(\alpha_0) \to \mathsf{List}(\alpha_0)\}
\end{aligned}
$$

   Prove the typing judgment
   $E \vdash \textbf{let}\ x = \mathsf{tail}\ (\mathsf{Cons}\ 1\ (\mathsf{Cons}\ 2\ \mathsf{Nil}))\ \textbf{in}\ \mathsf{head}\ x :: \mathsf{Int}$.

6. Solve the unification problem:

$$
\begin{aligned}
\alpha_3 \to \mathsf{List}(\alpha_3) \to \mathsf{List}(\alpha_3) &\approx \alpha_2 \to \alpha_1 \to \alpha_4; \\
\mathsf{Bool} &\approx \alpha_2; \\
\mathsf{List}(\alpha_0) &\approx \alpha_1; \\
\mathsf{List}(\alpha_0) &\approx \alpha_4
\end{aligned}
$$