

Functional Programming WS 2012/13

Harald Zankl (VO) Cezary Kaliszyk (PS) Thomas Sternagel (PS)

> Computational Logic Institute of Computer Science University of Innsbruck

> > week 12



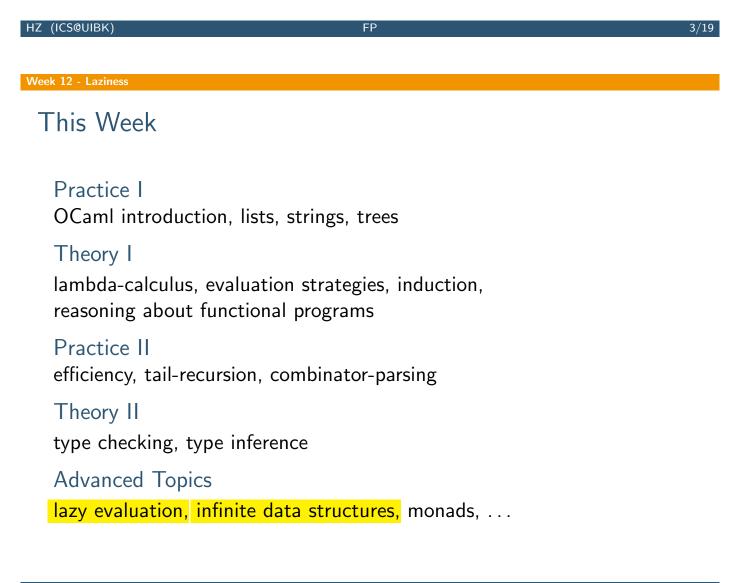


- prove that expression e has a type τ w.r.t. environment E
- formally: $E \vdash e : \tau$
- \blacktriangleright use the inference rules of ${\cal C}$ to do so

Summary of Weeks 10 & 11

Type Inference

- ▶ find most general type $\alpha_0 \sigma$ for expression *e* w.r.t. environment *E*
- formally: $E \triangleright e : \alpha_0$
- task is split into two parts:
 - 1. transform given type inference problem into unification problem
 - 2. solve the unification problem (result is substitution σ)



Motivation

Idea

Only compute values that are needed for the final result.

Example

In the program

```
let f1 x = x + 1 in
let f2 x = (* something non-terminating *) in
let x = read_int() in
Lst.hd(f1 x :: f2 x)
```

the value of 'f2 x' is not needed. Nevertheless, the whole program does not terminate.

```
HZ (ICS@UIBK)
                                    FP
                                                                        5/19
 Custom Lazy Lists – 1<sup>st</sup> Iteration
   Type
   type 'a llist = Nil | Cons of ('a * 'a llist)
   Example
                           ([])
    Nil
    Cons(1,Nil)
                           ([1])
    Cons(2,Cons(1,Nil)) ([2;1])
   Functions
   let hd = function Nil -> failwith "empty_list"
                     | Cons(x, ) \rightarrow x
   let rec from n = Cons(n,from(n+1))
```

Custom Lazy Lists – 1st Iteration (cont'd)

Problem

```
# hd(from 0);;
Stack overflow ...
```

Idea

- block computation of *tail*, until explicitly requested
- use unit function (i.e., of type unit -> ...)

```
FP
HZ (ICS@UIBK)
                                                                       7/19
 Custom Lazy Lists – 2<sup>nd</sup> Iteration
  Type
   type 'a llist = Nil | Cons of ('a * (unit -> 'a llist))
  Example
                                                   ([])
   Nil
                                                   ([1])
   Cons(1, fun () \rightarrow Nil)
   Cons(2, fun () -> Cons(1, fun () -> Nil))
                                                  ([2;1]))
  Functions
   let hd = function Nil
                            -> failwith "emptyulist"
                     | Cons(x, _) \rightarrow x
   let tl = function Nil -> failwith "empty_list"
                     | Cons(_,xs) -> xs ()
   let rec from n = Cons(n,fun() -> from(n+1))
```

Custom Lazy Lists – 2nd Iteration (cont'd)

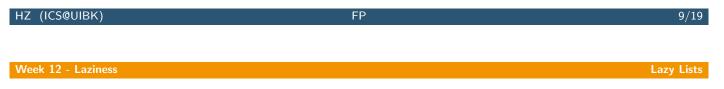
Now

```
# hd(from 0);;
- : int = 0
# hd(tl(from 0));;
```

-: int = 1

But

- strange that tail of llist is not llist itself
- use a mutually recursive type



Custom Lazy Lists – 3rd Iteration (module UnitList)

Type

```
type 'a cell = Nil | Cons of ('a * 'a llist)
and 'a llist = (unit -> 'a cell)
```

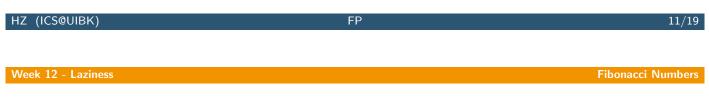
Example

```
fun () -> Nil ([])
fun () -> Cons(1,fun () -> Nil) ([1])
fun () -> Cons(2,fun () -> Cons(1,fun () -> Nil)) ([2;1])
```

Week 12 - Laziness

Lazy Lists

Custom Lazy Lists – 3rd Iteration (module UnitList cont'd)



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

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

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

Missing

- \blacktriangleright function to shift sequence to the left \rightarrow tl
- ▶ function to add two sequences $\rightarrow \text{zip}_with$ (+)

```
FP
HZ (ICS@UIBK)
                                                                  13/19
                                                           Fibonacci Numbers
 Implementation (in module UnitList)
   let tl xs = match xs() with Nil
                                          -> failwith "empty"
                              | Cons(_,xs) \rightarrow xs
   let rec zip_with f xs ys = fun() -> match (xs(),ys()) with
    | (Cons(x,xs),Cons(y,ys)) -> Cons(f x y,zip_with f xs ys)
    -> Nil
   let rec fibs =
    fun() -> Cons(0,fun() -> Cons(1, zip_with (+) fibs (tl fibs)))
  Example
     # to_list 10 fibs
     - : int list = [0; 1; 1; 2; 3; 5; 8; 13; 21; 34]
```

Fibonacci Numbers

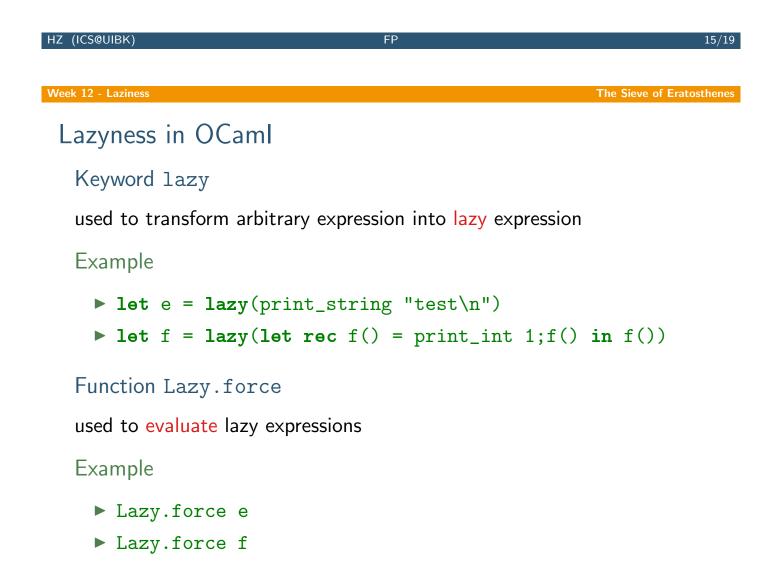
Problem

Lazy Enough?

- we defer computation (i.e., call-by-name evaluation) \checkmark
- \blacktriangleright we do not use memoization \times

Memoization

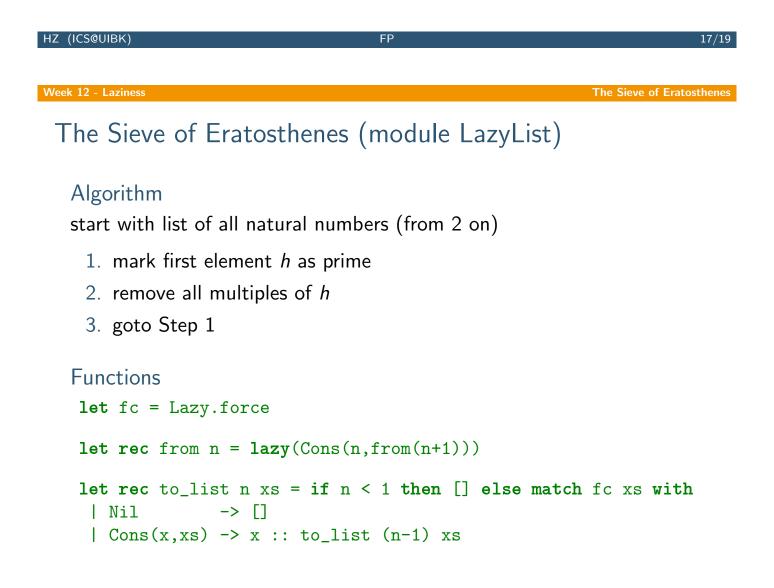
- prohibit recomputation of same expressions
- built-in in OCaml's support for lazyness



Lazy Lists Again (module LazyList)

```
Type
type 'a t = 'a cell Lazy.t
and 'a cell = Nil | Cons of ('a * 'a t)

Example
lazy Nil ([])
lazy (Cons(1, lazy Nil)) ([1])
lazy (Cons(2, lazy (Cons(1, lazy Nil)))) ([2;1])
```



Week 12 - Laziness

The Sieve of Eratosthenes (module LazyList cont'd)

HZ (ICS@UIBK)

FP

19/19