Lazy Lists

Functional Programming SS2007

Stefan Blom

Date: 2007/05/21 15:34:42
Revision: 1.1

1 The difference between lists and lazy lists

The file `lazyList.ml`, contains the module `LazyList`, which contains the lazy list iterator function `liter`.

```
let rec liter p l = match force l with
  | Nil -> ()
  | Cons(x,xs) -> p x; liter p xs
;;
```

The lazy list iterator, computes the entire list and applies the proecude `p` while it computes elements. For example, if we define

```
let rec lrange n k =
  lazy (if n <= k then Cons(n,lrange (n+1) k) else Nil)
;;
let rec range n k =
  if n <= k then n::(range (n+1) k) else []
;;
let p x = Printf.printf "%d\n" x ;;
```

then we get

```
# liter p (lrange 1 5);;
1
2
3
4
5
- : unit = ()
```

If we do this with normal lists then we get

```
# List.iter p (range 1 5);;
1
2
3
4
5
- : unit = ()
```

But the computation itself is in a very different order:

Normal		Lazy	
output	state	output	state
	`iter p (range 1 3)`		`liter p (lrange 1 3)`
	`iter p (1::(range 2 3))`		`liter p (Cons(1,lrange 2 3))`
	`iter p (1::2::(range 3 3))`	1	`liter p (lrange 2 3)`
	`iter p (1::2::3::(range 4 3))`		`liter p (Cons(2,lrange 3 3))`
	`iter p (1::2::3::[])`	2	`liter p (lrange 3 3)`
1	`iter p (2::3::[])`		`liter p (Cons(3,lrange 4 3))`
2	`iter p (3::[])`	3	`liter p (lrange 4 3)`
3	`iter p ([])`		`liter p (Nil)`
	`()`		`()`

2 Infinite lists

Normal lists must be finite, because they are stored explicitly. Lazy lists are stored in a partially evaluated form and can therefore represent infinite lists. For example, we can define the list of all prime numbers using the sieve of Eratosthenes.

```
let rec lseq n = lazy(Cons(n,lseq (n+1)));;

let rec sieve l = lazy ( match force l with
  | Cons(p,xs) -> Cons(p,sieve (lfilter (fun x->(x mod p)<>0) xs))
);;

let primes = sieve (lseq 2);;
```

Try running `liter p primes`!