Haskell
Lecture 7

Laziness and Infinity

Lazy Evaluation

- *Haskell never evaluates expressions before they are actually needed.*
- *If an expression appears twice, it is evaluated only once*
- *For example, if an expression is given as an argument to a function, the function gets a thunk - the unevaluated expression (actually, the function gets a reference to the thunk).*
- *If there are two references to the thunk, when first the value is needed, the thunk is evaluated; the second time value is needed, it is already evaluated*

Thunk and Laziness

- *In lazy.hs, inf has the value of an infinite list, but not until the list is needed*
- *The function f has 2 arguments, but never uses the second argument.*
- *g has the result from calling f with 2 arguments, the second argument is inf, but this is never evaluated, instead, f has the thunk inf ([1..]) passed to it*

Lazy Evaluation Problem

- *Consider foldl (+) 0 [1..3]*
    - *evaluates as (((0 + 1) + 2) + 3)*
    - *all parts are thunks until value of function is actually needed*
- *Consider foldl (+) 0 [1..100000000] gives stack overflow exception*
- *Data.List module has foldl' which does not evaluate lazily, thunks are not built up. This gives result with above list.*

List of prime numbers

- *Infinite list of prime numbers can be defined.*
- *Use sieve of Eratosthenes*
- *Start with list of all integers 2 or larger*
  - *pick next value on list and remove all larger multiples of that number*
  - *continue process forever.*
  - *See sieve.hs*

Find if a prime is in list

- *elem 13 primes gives result of True*
- *elem 6 primes runs forever*
- *elem does not use fact that primes is ordered*
- *write function memberOf which uses this fact*

Fibonacci Numbers Again

- *Write a function which will return a tuple containing two consecutive fibonacci numbers.*
- *fibStep input is tuple (u,v), result is (v, u+v)*
- *fibPair is recursive with base case 0, and calls fibStep in each recursive call to fibPair to get the next step*
- *See fibStep.hs*

Lazy Fibonacci

- *Use zip to combine elements of a list with its tail to produce further elements of list*
- *See lazyFib.hs*

Lazy IO

- *hGetContents lazily reads an entire file (getContents is the same, but works on stdin)*
  - *must create a file handle to use hGetContents*
- *readFile and writeFile are shortcut functions to lazily read from and write to a file.*
- *See contentsRead.hs and fileRead.hs*

Lazy IO Continued

- *Lazy input: Input is not read until it is needed*
- *In preceding examples, the call to an output function requires processed data which requires the input*
- *The output is also lazy. The output functions write the data as it becomes available, and if nothing else in the program needs the data, the memory storing it can be freed at the time of writing.*

Interact

- *This is a shortcut to read from stdin and write to stdout lazily.*
- *interact has type (String -> String) -> IO ()*
- *interact.hs shows same processing as preceding examples, but files are stdin and stdout.*