Advanced Programming Handout 11

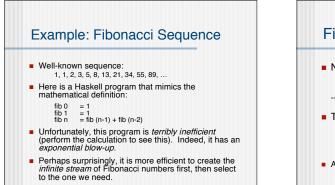
Programming With Streams (SOE Chapter 14)

Streams

- A *stream* is an infinite sequence of values.
- We could define a special data type for them: data Stream a = a :^ Stream a
- but in practice it's easier to use conventional lists, ignoring [], so that we can reuse the many operations on lists.
- Streams are often defined recursively, such as: twos = 2 : twos
- By calculation: twos → 2 : twos → 2 : 2 : twos → 2 : 2 : 2 : twos → .
 This calculation does not terminate - yet it is not
- This calculation does not terminate yet it is not the same as _l_, in that it yields useful information.
- [Another example: numsfrom n = n : numsfrom (n+1)]

Lazy Evaluation

- Two ways to calculate "head twos": head twos → head (2: twos) → head (2: 2: twos) → head (2: 2: 2: twos) → head (2: 2: 2: 2: twos)
- One strategy terminates, the other doesn't.
- Normal order calculation guarantees finding a terminating sequence if one exists.
- Normal order calculation: always choose the outermost calculation (e.g.: unfolding "head" above instead of unfolding "twos").
- Also called *lazy evaluation*, or *non-strict* evaluation.
- (In contrast to *eager* or *strict* evaluation.)



Note this relationship: fibs = 1 + 1 + 2 + 3 + 5 + 13 + 2 + 13 + 1 + 2 + 3 + 5 + 13 + 2 + 13 + 13 + 13 + 13 + 55 + 13 + 13 + 13	ïbs, cont'd
This is easily transcribed into Haskell: fibs = 1 : 1 : add fibs (tail fibs) tail (tail fibs) where add = zipWith (+) And then finally:	fibs 1 1 2 3 5 8 13 21 34
where add = zipWith (+) And then finally:	This is easily transcribed into Haskell:
	where add = zipWith (+) And then finally:

Chasing One's	Tail
 Notice in: fibs 1 : 1: add fibs (tail that "tail fibs" starts right h Introduce a name for that fibs 1 : if where if = 1 tift where if = 1 Doing this again for the ta 1 : if where if = 1 Finally, unfold add: 1 : if where if = 1 	here . value so it can be <i>shared</i> : add fibs (tail fibs) add fibs tf iil of the tail yields: 1 : tf2 where tf2 = add fibs tf

