

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 the same as `_!_`, 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)
→ 2

head twos
→ head (2 : twos)
→ head (2 : 2 : twos)
→ head (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.)

Example: Fibonacci Sequence

- Well-known sequence:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

- Here is a Haskell program that mimics the mathematical definition:

```
fib 0    = 1
fib 1    = 1
fib n    = fib (n-1) + fib (n-2)
```

- Unfortunately, this program is *terribly inefficient* (perform the calculation to see this). Indeed, it has an *exponential blow-up*.
- Perhaps surprisingly, it is more efficient to create the *infinite stream* of Fibonacci numbers first, then select to the one we need.

Chasing One's Tail

- Notice in:

fibs \rightarrow 1 : 1 : add fibs (tail fibs)

that “tail fibs” starts right here .



- Introduce a *name* for that value so it can be *shared*:

fibs \rightarrow 1 : tf where tf = 1 : add fibs (tail fibs)
 \rightarrow 1 : tf where tf = 1 : add fibs tf

- Doing this again for the tail of the tail yields:

\rightarrow 1 : tf where tf = 1 : tf2
where tf2 = add fibs tf

- Finally, unfold add:

\rightarrow 1 : tf where tf = 1 : tf2
where tf2 = 2 : add tf tf2

Garbage Collection

- Because of sharing, exponential blowup is avoided.
- In a few more steps we have:

fibs → 1 : tf
 where tf = 1 : tf2
 where tf2 = 2 : tf3
 where tf3 = 3 : add tf2 tf3

- Now note that “tf” is only used in one place, and thus might as well be eliminated, yielding:

→ 1 : 1 : tf2
 where tf2 = 2 : tf3
 where tf3 = 3 : add tf2 tf3

- Think of this as “garbage collection” of names.

Stream Diagrams

- An alternative (perhaps better) way to depict sharing is *graphically using a stream diagram.*
- Another example: *client-server interactions.*

