Rate inference for flow fusion

Amos Robinson
PhD student at UNSW

December 15, 2013

Shortcut fusion is great, but...

- Relies on inlining - depends on compiler's mood
- 'Local' - only looks at a few combinators at a time
- User must inspect core to find out whether it all fused

Flow fusion is a more global transform

- Being implemented as a GHC compiler plugin
- Core operation fuses set of combinators into single loop, if possible
- I imagine most of you have seen Ben's talk on this

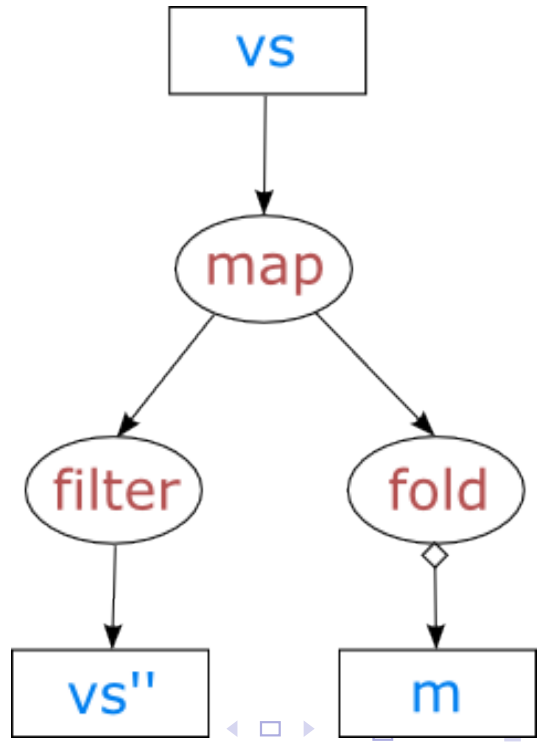
Rate inference schedules combinators into groups

- Each group becomes a single loop
- Aim to minimise number of loops and number of buffers
- This is what I'm talking about

Construct graph

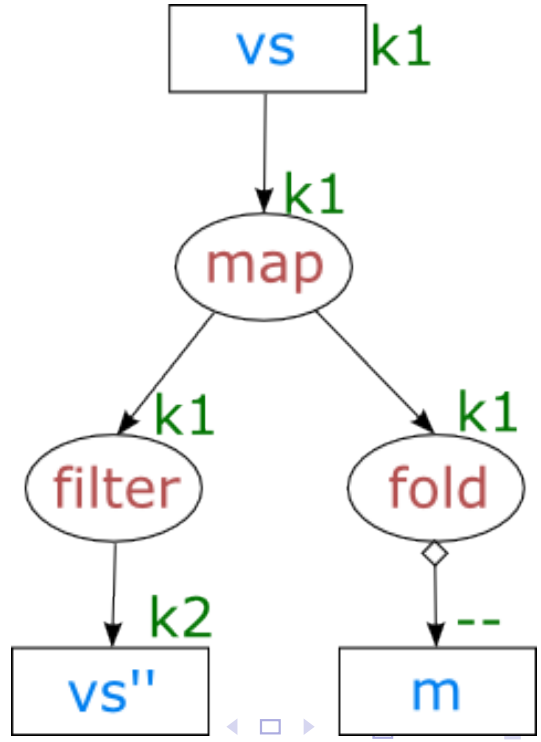
- Combinators are nodes
- Folds need all input before producing, so edge is *fusion-preventing*

```
filterMax (vs : Vector Int) =
 let vs' = map    (+1)    vs
     m   = fold     0 max vs'
     vs''= filter (>0)    vs'
 in (m, vs'')
```



Size/rate annotation

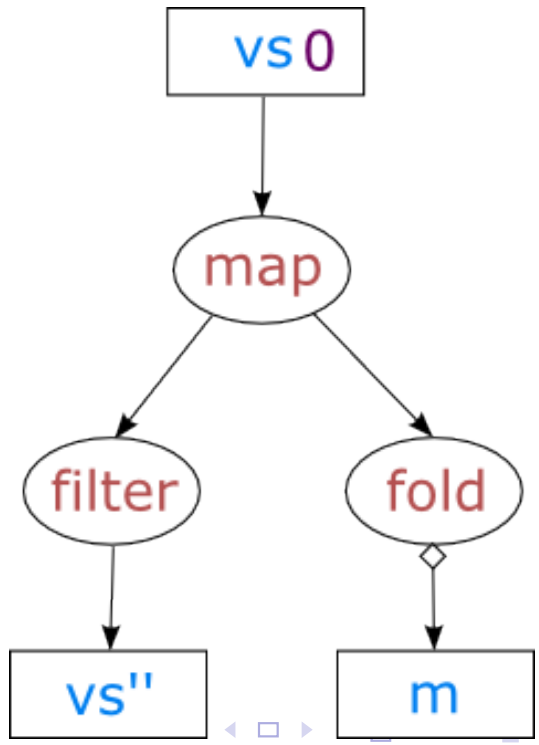
- Give each input fresh rate variable and propagate
- Filters are of *unknown* length
  with some upper bound



Scheduling

Finding a minimal schedule for this case is easy

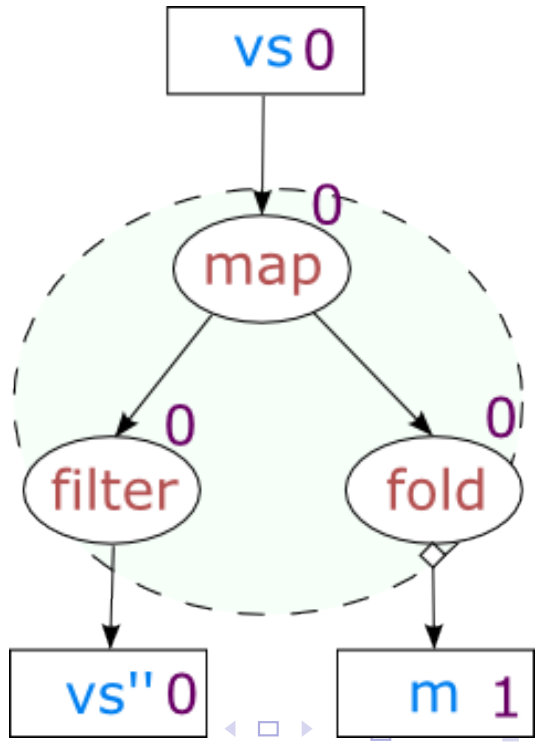
- Souces are 0
- $w(v) = \max_u(w(u) + \delta(u, v))$
- $\delta(edge) = 1$ if edge is fusion preventing



Scheduling

Finding a minimal schedule for this case is easy

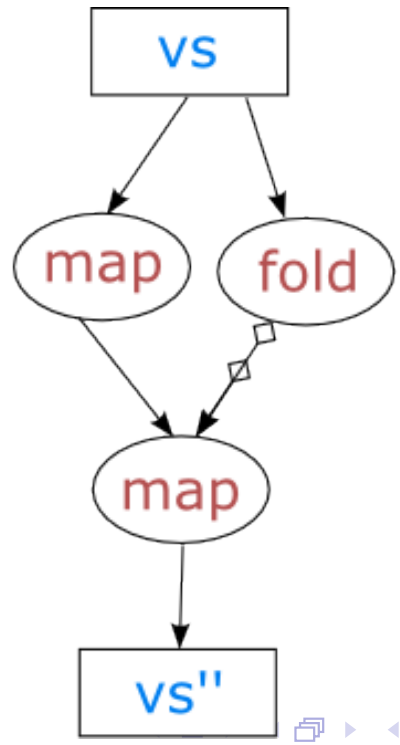
- Souces are 0
- $w(v) = \max_u(w(u) + \delta(u, v))$
- $\delta(edge) = 1$ if edge is fusion preventing



Minimal buffers

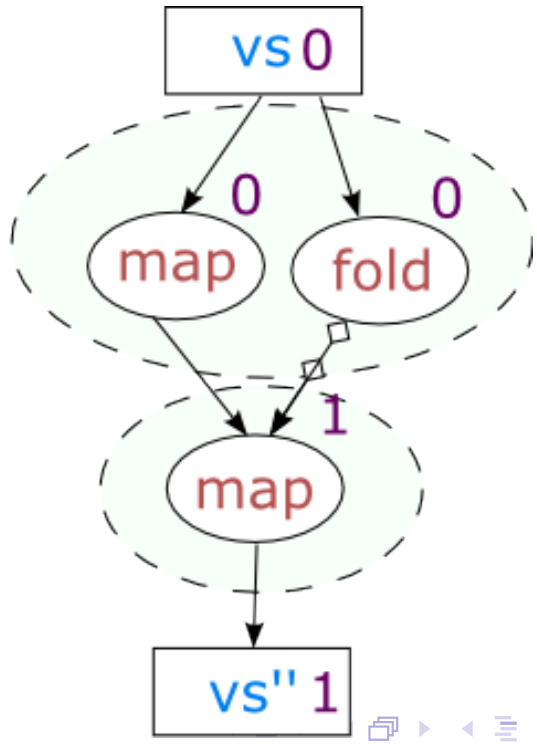
Some cases aren't quite as easy to schedule

```
normalise (vs : Vector Int) =
 let m   = fold      0 sum vs'
     vs' = map     (+1)    vs
     vs''= map     (/m)    vs'
 in (vs'')
```



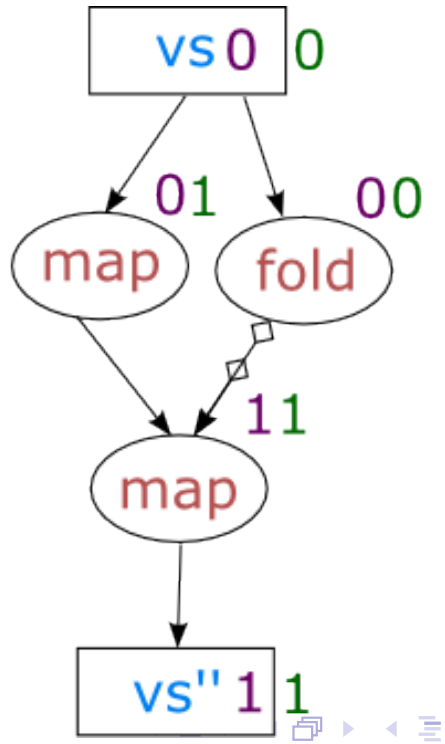
Minimal buffers

This scheduling requires a buffer for the first map's output.



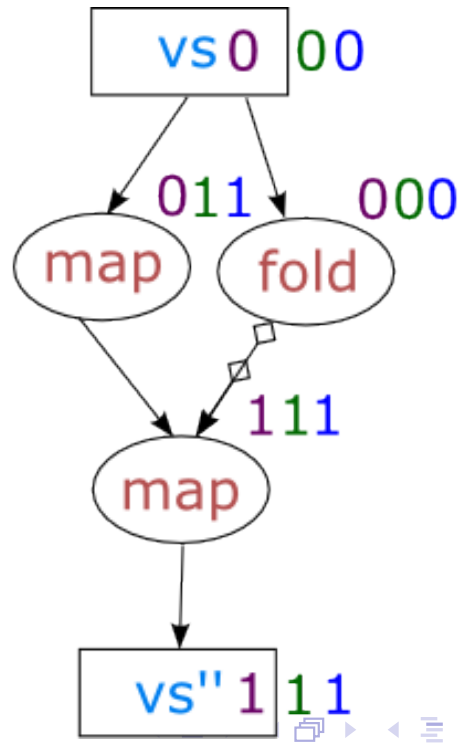
Minimal buffers

- The existing schedule is the *earliest*
- Working backwards, create a *latest* schedule
- $w(v) = \min_u(w(u) - \delta(u, v))$



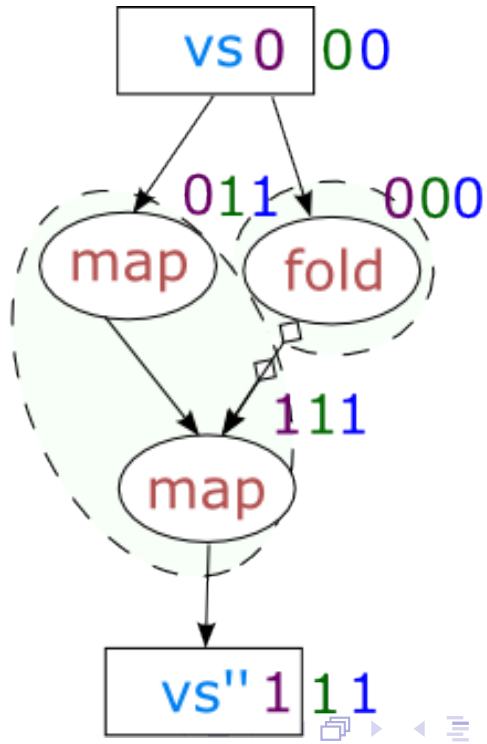
Minimal buffers

- The *optimal* schedule is somewhere in between
- The optimal schedule minimises edge crossings
- (In this case it is the same as the latest schedule)



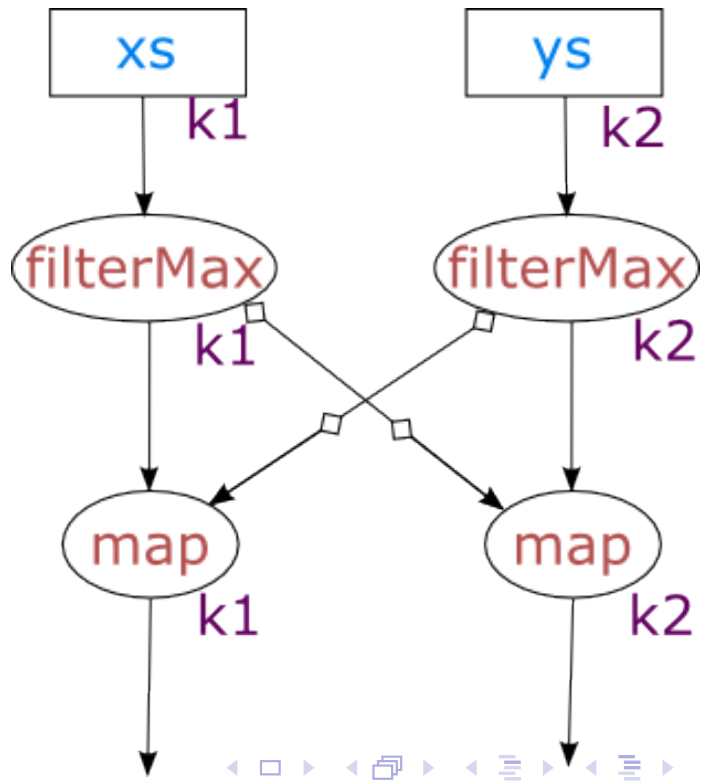
Minimal buffers

- The *optimal* schedule is somewhere in between
- The optimal schedule minimises edge crossings
- (In this case it is the same as the latest schedule)



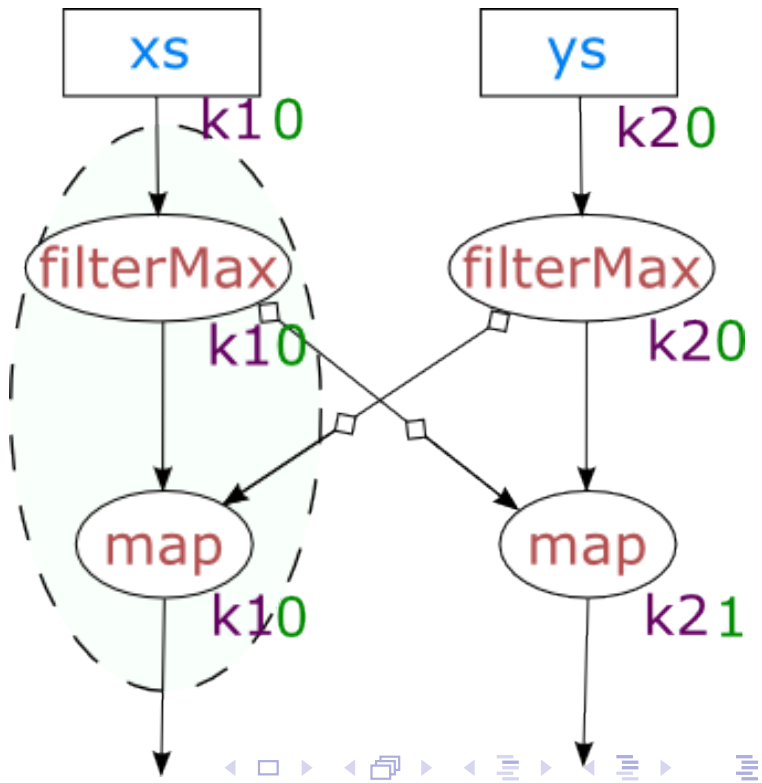
Mixing sizes

Combinators of different sizes cannot be fused



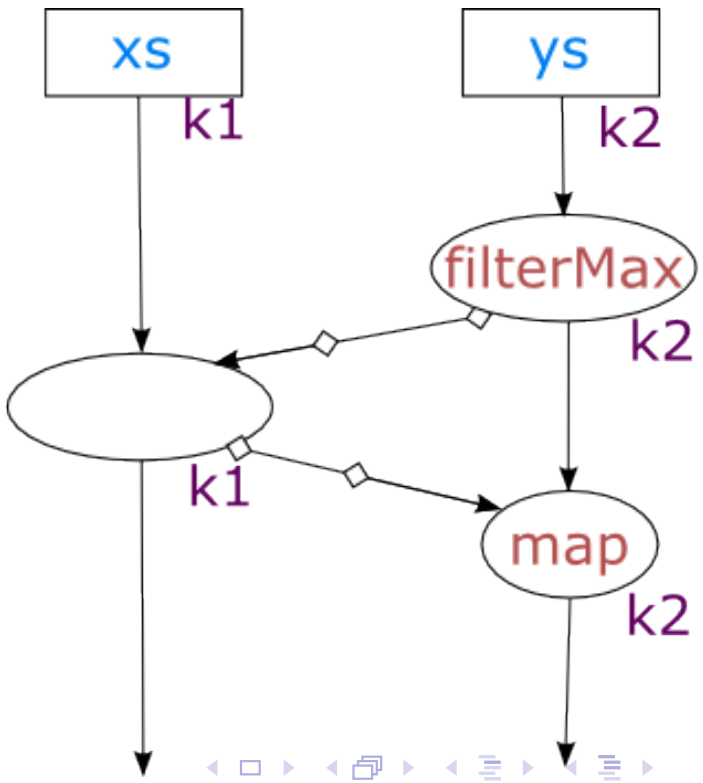
Mixing sizes

- Perform scheduling for each size variable separately
- With a slightly different δ function:
- $\delta(edge) = 1$ if edge is fusion preventing and source is same type



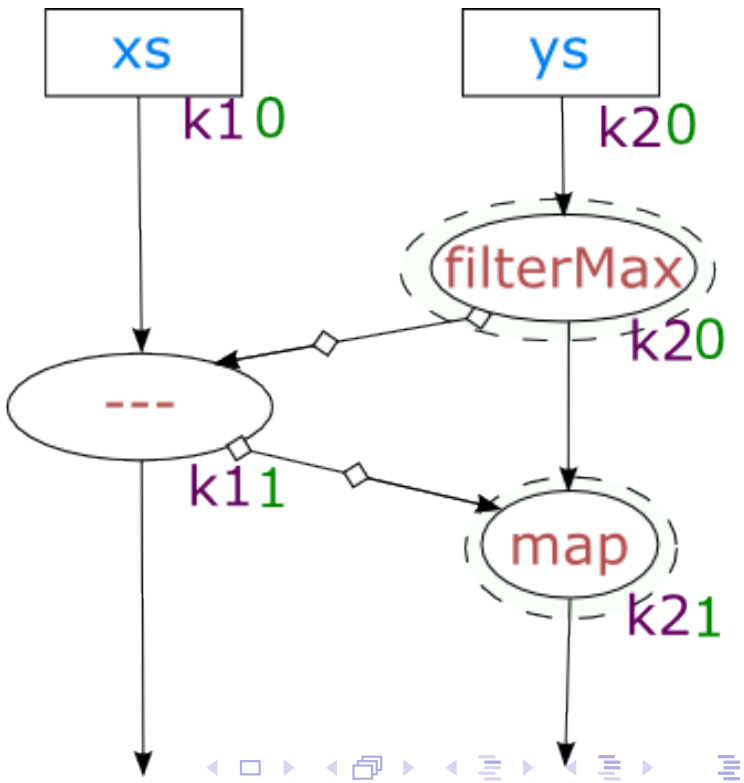
Mixing sizes

- Merge fusible nodes of given type together



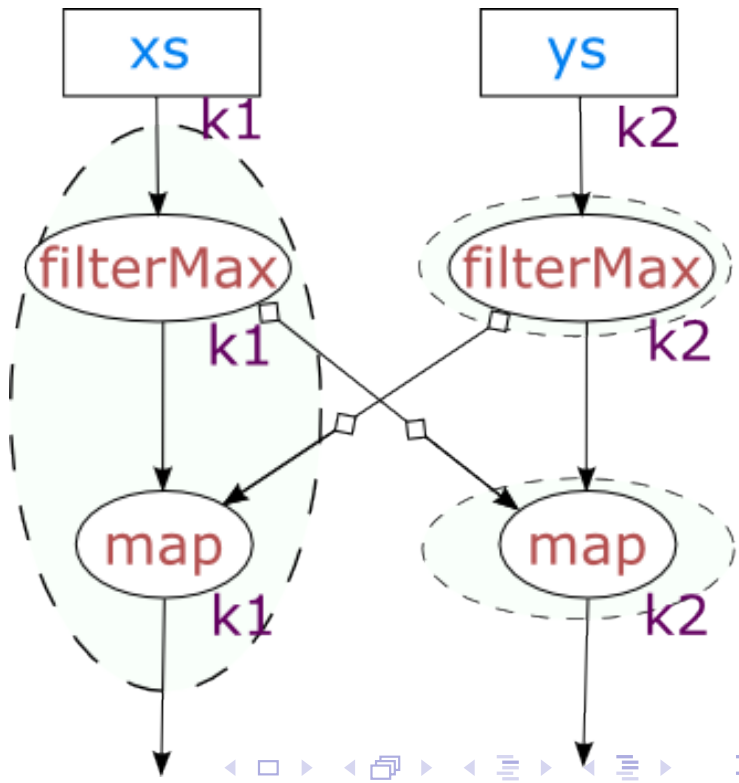
Mixing sizes

- Scheduling next type on merged graph



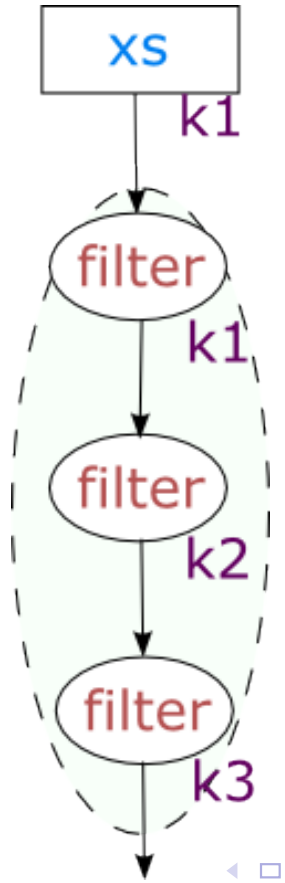
Mixing sizes

- After all types are done, we end up with



Filters

- Filters are special
- Despite being different sizes, they *can* be fused into their parents
- I'm still not sure about the best way to do this



The end

thanks

Size/rate annotation redux

It's a touch more complicated, but pretty boring:

- map2 (zipWith) requires inputs to be same rate
- filters are *skolem* and can't be constrained