Alternate Timelines for TidalCycles

Alex McLean
Then Try This
Penryn/Sheffield
alex@slab.org

ABSTRACT

The TidalCycles live coding environment has been developed since 2009, via several rewrites of its core representation. Rather than having fixed goals, this has been guided by use, motivated by the open aim to make music. This process can therefore be seen as a long-form improvisation, with insights into the nature of Tidal gained through the process of writing it, feeding back to guide the next steps of development. However, this brings the worrying notion that key insights will have been missed. It is well known that a beginner's mind is able to see much that an expert has become blind to. Running workshops are an excellent way to harvest new development ideas from beginners' minds, but the present paper explores a different technique - the rewrite.

This rewrite is an ongoing attempt to rewrite Tidal without reference to the existing codebase. It began with a two-hour public live stream[1] where I 'thought aloud' while rewriting its core representation of patterns as functions of time. The design of Tidal's innards has taken place over a decade, but this session could be viewed as replaying a condensed version of the thinking behind its design over a mere two hours. This points to a potentially useful research programme, where several people are invited to attempt a similar process of thinking aloud while remaking a core part of their system.

I will compare the work during this stream (and refinements in the days that followed) with the current 'mainline' codebase, identifying differences in terminology (i.e. the names of types, fields and functions) and implementation (e.g. of operation of the new functor, applicative and monad instances that control how patterns are combined). This rewrite was motivated by curiosity, but I outline the motivations that have emerged during the rewrite; the clarity from refactoring, the ideas to formalise Tidal's polymetric sequences underlying its 'mininotation', and more broadly ways to escape Haskell syntax with a custom parser.

[1] See https://www.youtube.com/watch?v=F2-evGtBnqQ for the live stream archive.