Leveling Up

A Survey of Functional Programming for Games

Chris ‘Topher’ Maraffi
topherm@soe.ucsc.edu
UC Santa Cruz

David Seagal
drseagal@soe.ucsc.edu
UC Santa Cruz
3 Types of Approaches

- Creating full games using Functional Programming.
- Using games as a pedagogical tool to teach game development and/or Functional Programming.
- Using Functional Programming for certain aspects of game development
Functional Programming and Games

- Functional Programming moving out of academia into commercial applications
- However, C++ still de facto language for videogame development
  - Efficiency
- Biggest proposed advantage of FP is writing cleaner code in fewer lines.
  - Paddleball: ~20 lines
  - Asteroids: ~220 lines
Most videogame attempts use Functional Reactive Programming over Functional Programming.

FRP adds elements of Reactive Programming useful for game development.

- Behaviors and events
- Values that vary over time

Allows easier implementation of an updatable game state.
The Challenge

- No switching of dynamic collections of reactive objects.
- Needed for adding/removing game objects.
- Lack of game examples.

I have to say I’m very sceptical about things like Fruit which rely on reactive animation, ever since I set our students an exercise implementing a simple space-invaders game in such a system, and had no end of a job producing an example solution. Things like getting an alien spaceship to move slowly downward, moving randomly to the left and right, and bouncing off the walls, turned out to be a major headache. Also I think I had to use “error” to get the message out to the outside world that the aliens had won. My suspicion is that reactive animation works very nicely for the examples constructed by reactive animation folk, but not for my examples.

- George Russell on the Haskell GUI list
Yampa developed in 2003 to address these issues.

In their paper “The Yampa Arcade” a Space Invaders-esque game is created as an example.
Yampa Space Invaders

- 3 game objects
  - Gun
  - Alien
  - Bullet
- Each object maintains its own game state
- Each object can be tested in isolation
Yampa Space Invaders

- Dynamic collection of game objects
- The route function determines how inputs are distributed
- The killOrSpawn determines whether to add/remove objects
In 2005 Mun Hon Cheong created Frag, a FPS built using Yampa.

HOpenGL used for graphics and IO.

Builds on structure established by Yampa Arcade.
FP Games and Pedagogy

- Students more motivated to program videogames; more individuality and creativity.
- FP suggested as middle ground between traditional game programming and script-based authoring.
- Can focus on problem solving without getting bogged down in CS concepts.
Not many papers on FP for games, so we compare two trajectories of FRP research that could impact game development.

Work of Conal Elliott (Microsoft) and Paul Hudak (Yale) on DSLs for reactive temporal media.

Developed out of their 1997 FRAN paper, which was later generalized as FRP by Hudak in 2000.
Two FRP Approaches to Reactive Media

- Both researchers have continued to independently revise their original FRP approach, now termed ‘Classic FRP’.
- Revisions >> make real-time implementations more efficient by addressing space leaks and latency.
- Motivation behind each approach is to provide FRP tools for media artists that are more intuitive and natural.
FRAN: The Birth of Classic FRP

- 1997 Functional Reactive Animation (FRAN) paper by Elliott & Hudak.
- Embedded DSL in Haskell that modeled reactive media = Behaviors that react to a stream of Event occurrences in continuous Time.
- Elliott continued developing Classic FRP while working at Microsoft Research from 1997-2001.
Classic FRP: DSL Approach

- Declarative *modeling* vs imperative *presentation* approach to temporal media.
- Modeling across the time dimension describes *what* an animation is, rather than *how* to present it.
- Continuous *stream* of time vs discrete *increments* of time == Continuous space (vector graphics) vs discrete space (pixel graphics).
- Scalability: Infinite size and resolution.
- Polymorphic *behaviors* and *events* which support a variety of media types.
Behaviors are time-varying, reactive values.

Events are sequences or streams of occurrences represented as a time-value pair.

First class values with a rich set of operators for composing an FRP program.

Static functions or values can be ‘lifted’ to behavior-value pairs:

\[ \text{Lift0} :: a \rightarrow \text{Behavior}\ a \]

Reactivity is in the form of mouse behavior, and keyboard or mouse-click events.

\[ \text{color} :: \text{Behavior}\ \text{Color} \]
\[ \text{color} = \text{red `until` (lbp -\rightarrow\ blue)} \]
Modern FRP 1: from Behaviors to Signals

- Represented streams of sampled behaviors = *signal processing*.
- Events = observations of physical signals outside of the FRP system.
- Yampa fixed memory problems in FRAN and FAL. “Use an arrow to plug a space leak”!!!
Yampa: Signals and Arrows

- DSL implemented as: virtual robot controller (2002); 2D arcade game (2003); 3D FPS (2005).
- Uses arrow notation (generalization of monad).
- Signals (formally behaviors):
  - not first-class values (hidden like the state monad).
  - Written in a point-free style (values not directly manipulated).
- Provides a set of arrow combinators to ‘wire’ signals.
- Deals with state through switch combinators.
Dynamic switching of game objects:

dpSwitch game example:

```
gameCore :: IL Object
    -> SF (GameInput, IL ObjOutput)
    (IL ObjOutput)
gameCore objs =
dpSwitch route
    objs
    (arr killOrSpawn >>> notYet)
    (\sfs’ f -> gameCore (f sfs’))
```
Elliott revisited (2009) FRP to modernize his original semantics, and deal with efficiency problems.

Influenced by Sperber’s Lulu FRP.

Restructured the Classic FRP functionality around standard type classes (monoids, functors, etc).

Hybrid ‘push’ (demand) and ‘pull’ (data) driven sampling fixed latency problems.

Added future values and improving values.

Latest implementation is called Reactive, and has been combined with the Fieldtrip functional 3D library.
FRP Comparison: Elliott vs Hudak

- Elliott’s pure FRP approach more *expressive*?
- Hudak’s hybrid FRP approach more *practical*?
- 8 Laws of Polymorphic Temporal Media = FRP provably sound.
- Both have a persistent focus on DSLs for the *arts* domain.
- Elliott’s graphics applications of DSLs for GUI fusion tools.
- Hudak’s music applications of DSLs for MUI virtual instruments.

Physical Model of a Flute

TV fusion subsumes function *composition*. 

R \rightarrow Region \quad R \rightarrow R \quad R \rightarrow Region
Burning Questions: Could FRP be a Game Changer?

- Declarative compositional approach a better fit for artists?
- Continuous time and behavior semantics more appropriate for streaming NUI games (Kinect)?
- Would FRP be more successful embedded in a commercial language such as F#?
  - More support for real-world applications?
  - Better integration with presentation systems?
  - Less bugs in the compiler?