

Javascript as an Intermediate Language for FRP

Subtitle Text, if any

Name1

Affiliation1

Email1

1. Introduction

Functional Reactive Programming (FRP) is an exciting and popular way to make interactive applications in a functional style. However, creating production level applications with FRP remains prohibitively difficult, especially for mobile platforms. Each mobile platform only supports particular languages and architectures, so many cross-compiling tools must be installed and managed. Many users are able to write FRP applications, but the lack of deployment options has kept this

We propose using Javascript as an intermediate language to eliminate most of the technical hurdles to publishing applications written with FRP, allowing more programmers to use FRP in production. We have developed a tool that automatically creates a mobile app package. It works for any for any mobile platform and takes an Elm program as input. Other languages like Haskell could easily be supported as well. The user can write exclusively in the FRP language, use a foreign function interface (FFI) to interface with lower level languages and hardware, or use FRP for only isolated pieces of the app.

2. Background

Although deployment is difficult, development of interactive apps can be faster and easier using FRP. One particular strength is that prototypes can be generated very quickly, and these prototypes are easily extended to full applications. This has been used in many interactive multimedia applications such as music, robotics, and games[1].

FRP implementations include many libraries in Haskell, the dedicated FRP language Elm, and the Sodium project which ports FRP to more language like Java and C. FRP often feels most natural in high-level functional languages like Haskell and Elm, but mobile development requires device specific languages (e.g Java for Android). Ideally, FRP programs could be cross-compiled to the target platform without significant interference in the development process.

For development on Apple's iOS platform, a cross-compiler exists for GHC that could provide a way to use a Haskell FRP library to develop iOS apps. Perez, in work with Keera Studios, has successfully cross-compiled Haskell to Android and published the

game *Magic Cookies*[2]. Although promising, the source code for this remains closed source and the setup has been difficult to reproduce due to many installation steps. Windows phone development remains largely unexplored. These however require that all tool chains be installed and managed separately - a large cost for an average user.

To simplify the process, we compile FRP programs to an intermediate language, then use a *platform agnostic* tool to compile the intermediate language to any target device desired. The intermediate language presented here is Javascript and platform agnostic tool is Phonegap. For Haskell, ghcjs can compile program to Javascript, while Elm is compiled to Javascript by default. Phonegap is a program developed to allow users to create apps for all platforms by writing Javascript. We instead send the generated (and incomprehensible) Javascript to Phonegap to create a package for any mobile device. Here Javascript is functioning as an intermediate language - the user does not need to look at it for the system to work.

Mobile development also relies on the use of APIs. Many platform specific APIs have similar Javascript implementation, which can be used via the FRP languages FFI. For APIs that require native code, Phonegap provides textplugins as a foreign function interface (FFI) from javascript to the lower level language. The Javascript plugin code can then be accessed via the FRP language through that FFI. Both ghcjs and Elm provide a javascript FFI to enable this chain.

Performance is a concern when using an intermediate language without optimizations. A common measure for performance in FRP systems is the frames per second (fps) that can be displayed, or the frame rate. Any number above 32 fps is generally acceptable, and our proof of concept app maintains a frame rate of ~150 fps on a Nexus 5 running Android 5.1.1. However, this is only for a 2D game, so performance measures for more complex systems will be needed in future work.

3. Results

This app uses FRP to read the touch interface native to the mobile hardware and control a game. The app also uses a plugin FFI to show ads, mixing javascript code with Elm. The source code and app files are available at github.com/santolucito/elm_games.

This approach is unique because it treats the high-level language of one tool as an intermediate language for another. It is unique in its simplicity because it leverages Phonegap to unify the workflow for all platforms - iOS, Android and Windows apps are all created from the same system. There is no need to install a different cross-compiler for each target device. This approach also works well in practice, providing good performance and access to low-level APIs when needed.

The development of FRP and functional languages in general relies on adoption by industry. Simplifying the workflow to bring FRP code to market will encourage future work in FRP.

References

[] P. Q. Smith, and X. Y. Jones. ...reference text...

```
main =  
    Signal.map2 scene Window.dimensions  
  
clickLocations =  
    Signal.foldp (::) [] Mouse.position  
  
scene (w,h) locs =  
    let drawPentagon (x,y) =  
        ngon 5 20  
        |> filled (hsla (toFloat  
        |> move (toFloat x - toF  
    in  
        layers  
        [ collage w h (List.map drawP
```