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 prohibitivly difficult, especially for mobile platforms. Each mobile platform only supports particular languages and archetictures, 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 circle to only the most dedicated FRP researchers.

Using Javascript as an intermediate language eliminates most of the technical hurdles to publishing applications written with FRP, allowing more programmers to use FRP in production. We have developed a tool chain to automatically create a mobile app package for any mobile platform from an Elm program, though other languages like Haskell could easily be supported as well. The user can write exclusively in the FRP language, use a forgein function interface (FFI) to interface with lower level languages and hardware, or use FRP for only isolated pieces of the app.

2. Background

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

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 succesfully cross-compiled Haskell to Android and published the game *Magic Cookies*[]. 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 seperatly - 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.

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 in our proof of concept app we were able to acheive a frame rate of XX fps. However, this is only for a 2D game, so performance measures for more complex system will be needed in future work.

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

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.

3. Results

We developed a proof of concept app using Elm and have provided the source code and app files at This app uses FRP to read the touch interface native to the mobile hardware and control a game. The app also uses a plugin to show ads, mixing javascript code with Elm. It can run at ?? frames per second on an Nexus 5 running Android verison X.XX. pass in (fps 200) as the time signal to a display of "toString (1000/t)" to meausre actual FPS - capped at 200 on laptop.

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

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