Jim Shargo
Ryan Cassidy
Implementing “Embedding Dynamic Dataflow in a Call By Value Language”

Cooper and Krishnamurthi present a functional reactive programming language embedded in Racket (then known as PLT Scheme). This language, called FrTime, introduces the concept of ‘signal values’ to Racket. Signal values vary over time, and are updated via a hierarchical dataflow dependency graph. FrTime uses a push-driven evaluation strategy to update signals that are dependant on other signals. FrTime has mathematically complicated rules for dynamically reconfiguring the graph when there are dynamic dependencies. These dynamic dependencies occur, for example, when the program applies a signal as a function, or placing a signal in the test clause of an if expression. The dependants of these cases are considered dynamic because signals that might be created by their evaluation must be updated when, for example, the function signal changes or a different branch of the expression gets explored. If they were not updated, they could cause inappropriate runtime errors.

The reductions for constructing a FrTime dataflow map and subsequently updating it take up about an entire printed page of mathematics within the paper. In our proposal for this project, our main questions involved developing an understanding of these large reduction relations, and what they really were doing. Furthermore, we also did not fully grasp the differences between different signal types which are used in this reduction math. Since signals are the building-blocks of the semantics of FrTime, we sought to fully understand the differences and similarities between them. These questions underlie our main objective, which is to understand the dynamic update system.

To answer these questions, we constructed a model of FrTime using PLT Redex. We defined two languages in our Redex model: ‘FrTime’ for use by modeling the front-end of the language; and ‘FrTime-Semantics’, an extension of FrTime that contains the data structures used in the back-end of the language. The two reduction relations presented in the paper which we aimed to understand are named ->construction and ->update. Modeling these relations is the core of our project and our code. The ->construction relation creates the dataflow graph for a FrTime program. This graph is the core of FrTime programs. It provides a path for the later computation of programs by hierarchically ordering the computation of signal values.

Once we understood the basics of constructing the signal graph (through implementing ->construction), the nuances between the distinct types of signals became clear, and enabled us to implement the dynamic update system within FrTime via implementing ->update. Once the main signal map is created with ->construction, it is augmented with a context comprising of external signal events, and then updated

continuously with the ->update relation. This ->update relation updates values in the graph over time, providing ‘dynamic data’.

Our final question involved developing an understanding of the specific signal data types used in this graph. Some signal types are less complex than others. **lift** signals ‘wrap’ primitive operations that depend on signals in a signal value to ensure they can be reevaluated as those signals changes. **const** signals represent constant values, and **delay** signals represent values that a signal had in the past. In the dependency graph, changes to base signal values may require a significant rebuild of the dataflow map to make sure evaluation takes place correctly. This is accomplished with **dynamic** signals, which contain a lambda to compute the value of the signal when evaluated, and **fwd** signals, which serve as roots in the graph for the section of the graph potentially modified by a dynamic signal.

The Redex model we designed provided valuable insight into the inner workings of this language. We formed an understanding and intuition for the interplay of the different signals in the graph created by ->construction. This understanding provided us with satisfactory answers to our initial questions.