1 Background and Motivation

1.1 Reactive Systems

Reactive systems are a broad class of computer systems whose defining element is the continued interaction between the system and its environment. Reactive systems are important because of their many applications in embedded devices, robots, hardware circuits, GUIs, interactive multimedia, video games, and more. Reactive systems are also considered among the most difficult types of systems to design correctly, and writing a reactive program is a highly error-prone process. In an ideal world, we would only describe the desired properties of the program in a standard specification language, such as a temporal logic, and let the computer generate a correct-by-construction program from that specification.

1.2 Functional Reactive Programming

Functional Reactive Programming (FRP), is a programming paradigm for expressing reactive, streaming computations in a functional language. It creates a clean modularization between the control structures and the data transformations in a reactive system. The fundamental idea of FRP is to extend the classic building blocks of functional programming with the abstraction of a signal to describe time-varying values:

\[ \text{Signal } a :: \text{Time} \rightarrow a \]

which produces values of some arbitrary type \( a \) over time. The type \( a \) can be an input from the world, such as the current position of the mouse, or an output type, such as some text that should be rendered to the screen. Signals are also used internally to manipulate values over time, such as when the position of the mouse should be rendered to the screen.
1.3 Temporal Stream Logic

Temporal Stream Logic (TSL) was introduced by Professor Piskac’s group as the first specification language for functional reactive programs. The logic is based on linear-time temporal logic (LTL), the most commonly used specification language for reactive programs. TSL extends LTL with a specification of how output signals are computed from input signals. This specification allows easy specification of the behavior of reactive systems.

1.4 Program Synthesis

Professor Piskac’s group has also developed a framework for synthesis of reactive systems from TSL specification to runnable FRP programs. The immediate advantage of synthesis over manual programming is that if the synthesis succeeds, there is a guarantee that the constructed program satisfies the specification. In this way, synthesis can also serve as a check on manually written specifications, where a failed synthesis may indicate an incorrectly written specification. Finally, synthesis is desirable as a way to make programs easier to inspect and understand, even by readers who may not be familiar with the particular FRP libraries or syntax.

In the current system for FRP synthesis from TSL specifications, the user provides a TSL specification over a set of predicate and function terms. This specification is then synthesized into a Term Annotated Mealy Machine (TAMM), which is then translated into an FRP using the Haskell Yampa library\(^1\).

While the current system successfully synthesizes FRP programs, there is a number of possible improvements to explore. First, rather than synthesizing using the TAMM intermediate, a semantic representation of the program, we can attempt synthesis by converting the TSL specification to a boolean circuit. Preliminary tests additionally indicate that synthesized programs may be much less efficient than handwritten programs. Exploring reasons why the synthesized program performs so much worse than handwritten programs may give us insights into optimization and improve the applicability and utility of the synthesis procedure. Finally, given a particular TSL specification, the question of whether or not this specification is realizable is undecidable. Hence, the synthesis framework is a nondeterministic process. Analyzing the specifications for which synthesis does not succeed may offer interesting results.

2 Proposed Work

My proposed work for this project will take place in three broad categories: AIGER translation, benchmarking of synthesized programs, and exploration of optimization for synthesized FRP programs.

\(^1\)https://wiki.haskell.org/Yampa
2.1 AIGER Translation

AIGER is a format, library and set of utilities for And-Inverter Graphs (AIGs)\(^2\). In essence, AIGER specifies the logic of a program as a boolean circuit, and can be used to formulate structural SAT and model checking problems. The AIGER format is used widely in the SYNTCOMP reactive synthesis competition\(^3\).

For this part of the project, I will implement a tool to translate AIGER specifications into FRP programs. I will adapt existing LTL to AIGER synthesis along with the current TSL to LTL reduction tool in order to implement a tool that completes the entire process of translating a TSL specification into an FRP program.

2.2 Benchmarking

For this part of the project, I will obtain benchmarks to compare the TAM\(-\)synthesized FRP programs, AIGER-synthesized FRP programs, and handwritten programs. Based on benchmarking results, we will better optimize the synthesis procedure and synthesized programs and further evaluate the applicability of FRP synthesis.

2.3 Optimization

For this part of the project, I will explore reasons for why synthesized programs perform worse than handwritten programs and look for possible optimizations in both the synthesis procedure and resulting synthesized programs. I will also explore the theoretical optimal performances of programs based on our FRP restrictions.

3 Deliverables

The culmination of this project will result in the following deliverables:

1. AIGER to FRP translation tool
2. FRP Synthesis Benchmarks
   a. TAM\(-\)synthesis benchmarks
   b. AIGER-synthesis benchmarks
   c. handwritten FRP benchmarks
3. Optimized FRP synthesis tool
4. Any supplemental correctness proofs

---

\(^2\)http://fmv.jku.at/aiger/
\(^3\)http://www.syntcomp.org/