Dependent Types for Real-time Constraints

William Blair
wdblair@cs.bu.edu
Boston University

Hongwei Xi
hwxi@cs.bu.edu
Boston University

Synopsis

Synchronous programming languages emerged in the 1980s as tools to implement reactive systems. These systems interact with events from their physical environment and often must do so under strict time constraints. In this talk, we use ATS, a statically typed functional language with a highly expressive type system, to support real-time primitives in an experimental synchronous language called Prelude. We show that these primitives and their verification requirements may be expressed using dependent types in ATS, and then modify the Prelude compiler to produce ATS code for type checking. This modified Prelude compiler thus discharges part of its proof obligations to ATS. Whereas ATS is typically used as a general purpose programming language, we want to demonstrate that it can also be used to support certain advanced features in languages equipped with less expressive types.

1 Talk Description

Building software that must work reliably is an undeniably challenging process. Yet, engineers can still construct reliable systems that operate under strict temporal requirements. Recently, companies have adopted model-based methods for designing real-time software. In this approach, code is automatically generated from high level models that express the behavior of a system. Synchronous programming languages represent a movement towards this paradigm and were first introduced in the 1980s. In synchronous programs, tasks communicate through data flows, and it is assumed that values from flows are computed instantaneously at every step of a logical clock.

Despite this simplification, some systems are still difficult to be expressed in such a high level environment. If a system consists of multi-rate periodic tasks, the advantage of having a global logical clock is no longer useful since synchronization must now be done manually by the programmer. This process can be tedious and error prone with little support given by the compiler for finding subtle timing errors. As a part of his PhD thesis, Julien Forget developed Prelude, a synchronous language that features primitives called clocks that describe the real-time behavior of tasks in a multi-rate periodic system. Synchronizing flows with different clocks is still left to the programmer, but the language features primitives to directly modify real-time clocks, such as oversampling and undersampling. Prelude's typechecker then automatically checks that all communicating tasks are synchronous. The advantage of this system is that the language offers the ability to easily adjust the temporal behavior of flows and verifies that communication is synchronous.

In this talk, we will describe how these real-time primitives can be captured in ATS. ATS is a call-by-value functional programming language with a core of ML-style. The type system of ATS is rooted in the Applied Type System framework, which gives the language its name. In ATS, both dependent types (of DML-style) and linear types are supported. For this work, we use dependent types to express real-time clocks available in Prelude, and we develop ATS functions that capture the behavior of all clock transformation operators in its clock calculus. We also show an example in which ATS code can be automatically generated from compiling Prelude programs. We argue that typechecking these ATS programs verifies synchronization of flows in the original Prelude programs. Though ATS is typically used as a source language, we show hereby that it can also be used as a target language for the purpose of supporting advanced features in a host language equipped with less expressive types. In addition, our use of dependent types to reason about temporal characteristics in a system should be interesting in its own regard.