Dependently Typed Web Client Applications
FRP in Agda in HTML5

Alan Jeffrey
Alcatel-Lucent Bell Labs

Abstract. In this paper, we describe a compiler back end and library for
web client application development in Agda, a dependently typed func-
tional programming language. The compiler back end targets ECMA-
Script (also known as JavaScript), and so is executable in a browser.
The library is an implementation of Functional Reactive Programming
(FRP) using a constructive variant of Linear-time Temporal Logic (LTL)
as its type system.

1 Introduction

Client-side applications are typically model-view-controller architectures, and
often include features such as imperative state, concurrency and continuation-
passing. These features can result in code which is difficult to reason about,
debug and maintain. In this paper, we propose adapting Functional Reactive
Programming (FRP) [13] to the setting of a pure, dependently typed, functional
programming language, Agda [1].

Figure 1 shows some simple applications running in a browser. What is inter-
esting about these applications is that they are written in Agda, and compiled to
ECMAScript [7]. We have developed a compiler back end, foreign function inter-
face, and library bindings for FRP, and for HTML5 [15] Document Object Model
(DOM) node and event bindings. The compiler extensions have been released
as part of Agda 2.3.0, and the libraries are released under an MIT License [4].

Novel features of the compiler and libraries include:

– Interoperability with ECMAScript idioms. The compiler makes use of com-
mon ECMAScript idioms, to simplify the use of existing ECMAScript li-
braries in Agda. For example, the Visitor and Observer patterns [14] are
used to implement inductive datatypes and notification.

– Singleton analysis for type erasure. We perform a static analysis that con-
servatively approximates singleton types (which have only one inhabitant
at run time). Any term of singleton type is replaced by the singleton value
at compile time. In particular, we regard Set as having singleton value null,
which allows many type-level computations to be eliminated.

– View patterns in the FFI. We support a ECMAScript Foreign Function In-
terface (FFI) which, as well as providing bindings for constants and func-
tions, also allows inductive datatypes in Agda to be bound to any ECMA-
Script type. A variant of view patterns [24] allows pattern-matching to be
compiled to any ECMAScript conditional, for example an Agda boolean type can be compiled to ECMAScript native booleans, without any additional support from the Agda compiler.

- **Linear-time Temporal Logic (LTL) types for FRP.** The semantics of FRP is defined in terms of signals, which are time-dependent values. In previous work [16], we showed that signals can be typed using time-dependent types, using the combinators of LTL [23], such that any FRP program is a proof of an LTL tautology.

- **Resource reclamation of FRP signals.** The FRP implementation makes use of techniques from *self-adjusting computation* [8], where signals form a dataflow graph, making use of notifications whenever a signal value changes. We are recording the creation time of each signal in its type, and so can maintain time-sensitive invariants which allow resource reclamation of irrelevant signals, even when the garbage collector regards the signal as still live.

- **Inference of DOM node locations.** A difficult problem in GUI libraries for functional languages is the binding of event listeners to GUI components. In an OO language, binding makes use of object identity, which violates referential transparency since components with identical definitions may have different event streams. In a functional language, this could be modeled by a name creation mechanism [22] or nondeterminism [19], but such models are not compatible with Agda’s semantics. We provide a novel form of location inference, which supports the creation of DOM event streams from DOM nodes without violating referential transparency.

Agda is used throughout this paper, but we expect the results would apply to other dependently typed languages, such as Coq [3] or Epigram [5].

Thanks to Sebastian Bocq for detailed comments on this paper.

2 Compiling Agda to ECMAScript

We first consider the design of the ECMAScript back end for the Agda compiler, which is included in Agda 2.3. The compiler translates a dependently typed λ-calculus with inductive datatypes and records into an untyped λ-calculus with records. The interesting features of the compiler are its treatment of singleton