Jarlang

Chris Bailey, Nick Laine, Andrew Johnson

Supervised by: Scott Owens

Project Description

The central aim of this project is to write an Erlang to ES6 JavaScript transpiler, enabling solutions to problems in JavaScript to be expressed as Erlang code. Jarlang also enables websites to unify the languages used in their stack, utilising Jarlang-transpiled Erlang on the frontend, and either Erlang or Jarlang-transpiled Erlang (running via Node.js) on the backend.

Our transpiler is written in Erlang, the same language as our source language. The motivation behind this is so that in time we can bootstrap the Jarlang transpiler and transpile it to JavaScript, thus enabling us to compile or interpret raw Erlang in a browser environment independent of any Erlang packages installed on a host's system. We also intend to implement as much of Erlang as possible in JavaScript, including function overloading, modules, tail recursion and ultimately – and more ambitiously – emulating Erlang-style concurrency using JavaScript's event loop.

The bulk of our code is written in Erlang and utilises the Erlang compiler to generate Core Erlang, an intermediate language used in Erlang's compilation process. We then use built-in functions to get the Core Erlang abstract syntax tree which is then translated into a valid JavaScript abstract syntax tree via a series of transformation steps. Valid JavaScript is then derived from this abstract syntax tree via the employment of existing JavaScript projects such as escodegen.

Results

The Jarlang transpiler is capable of successfully transpiling relatively simple Erlang source code to JavaScript. One of the main issues Jarlang has is the lack of a standard library, thus more complex code may not run after being transpiled. In time, this issue will be resolved by simply implementing the necessary Erlang language features (of particular importance, Erlang's data types) in JavaScript, so that we can transpile more of Erlang's standard library to utilise in transpiled code. We have succeeded in implementing the majority of Erlang's data types, however they have yet to be integrated into the transpiled code.

In addition to this, we have implemented actor-style concurrency in JavaScript. Our success in implementing this has eliminated many of the initial concerns we harbored for this project, however this has raised other concerns such as garbage collection of completed processes. We have decided that this is outside the scope of this project.