Abstract

Concurrent programming is notoriously difficult, due to needing to reason not only about the sequential progress of any algorithms, but also about how information moves between concurrent agents. What if programmers were able to reason about their concurrent programs and statically verify both sequential and concurrent guarantees about those programs’ behaviour? That would likely reduce the number of bugs and defects in concurrent systems.

I propose a system combining dependent types, in the form of the Idris programming language, and the Actor model, in the form of Erlang and its runtime system, to create a system which allows me to do exactly this. By expressing these concurrent programs and properties in Idris and by being able to compile Idris programs to Erlang, I can produce statically verified concurrent programs.

Given that these programs are generated for the Erlang runtime system, I also produce verified, flexible Idris APIs for Erlang/OTP behaviours.

What’s more, with this description of the Idris code generation interface, it is now much easier to write code generators for the Idris compiler, which will bring dependently-typed programs to even more platforms and situations. 1, for one, welcome our new dependently-typed supervisors.

1. Idris to Erlang Compiler

My compiler uses the existing Idris code generation system. There are three possible Idris intermediate representations that I could generate code from: a high-level IR with lambdas and laziness, a functionalised IR with only fully-applied functions, and an applicative-normal form IR. In my case, we used the defunctionalised IR to generate Erlang source code from, according to the following translation, \$E \langle \text{expression} \rangle \$:

\[^{\text{variable}}\] turns IR variables into valid Erlang variable names; \[^{\text{quote}}\] \$\text{constant}\$; \[^{\text{expression}}\] \$\text{expression}\$; \[^{\text{constructor}}\] creates constructors; \[^{\text{name}}\] \$\text{name}\$ turns IR names into valid Erlang atoms; \[^{\text{quote}}\] \$\text{expression}\$\$\$ turns primitive operators into their Erlang equivalent.

Foreign Calls

In Idris, the type of IO a is in fact parameterised by the types that the foreign function interface uses. So for instance, IO a is the type of computations that use C-like types in their foreign functions. Erlang doesn’t have a type system like C’s, so instead we defined our new computation type, EIO a which is parameterised by the types that Erlang uses, so we can make foreign calls to Erlang code.

2. Concurrency in Idris

Of course, our Idris to Erlang compiler is uninteresting unless we can use it to write concurrent programs.

The system I settled on implements the Actor model - Erlang’s underlying concurrency model - as a type Process 1 a, where i is the type of messages that process can receive (something I like to refer to as the language of a process), and n is the type of what the computation creates.

We can then embed important details about how spawning, sending and receiving work into the types of the functions that do so, for instance:

```
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
ENDORSEMENT
```

For instance, this program will only run concurrently (it uses some basic constructs on top of Process to provide RPC functionality):

```
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Conclusion

I have shown that Idris is an entirely adequate programming language for concurrent programming. With the new Erlang code generation system that I have built, we can now write and run safe, flexible actor-based programs which conform to statically proven guarantees.

I have built an Idris to Erlang compiler, which supports concurrent Idris code. Along the way I have documented exactly how to produce a compiler and how my compiler works.

I also had devised a way to model the Actor model in Idris’ dependent types in a lightweight way such that we can run the resultant concurrent programs on Erlang, including doing inter-process RPC.

I looked into a heavily dependently-typed model of the three main Erlang/OTP behaviours, which I finished, but are not executable.

I wanted to look into modelling other concurrency calculi in Idris, but was unable to do so, and likewise I was unable to devise a Hoare-like logic for Erlang and its runtime system.

I hope that the work in my dissertation paves the way for future research into concurrent programming with Idris and other dependently-typed programming languages.