Project Title: Data-Driven Debugging for Transparent Research Outputs, Supervisor: Roly Perera

Student: Harleen Gulati

Summary of the project and its intended outcomes:

Pretty-printing takes expressions to a 'pretty' layout which can be read by a user. The project involved pretty-printing expressions in the Fluid programming language so users can see and interact with the code that produced their output.

The outcome was to ensure the pretty-printing functions had a 'round-tripping property' which will be discussed further later.

Progress made during internship/key steps:

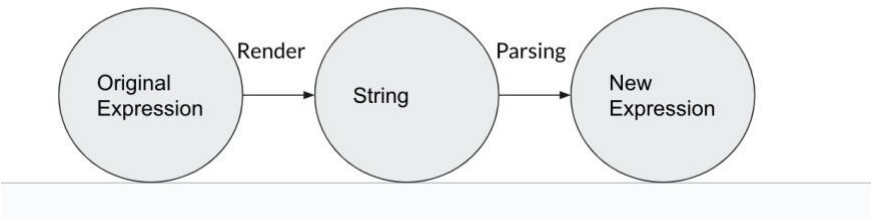
Pretty-printing takes expressions and converts them to a Doc (a data type which knows how to lay itself out prettily).

We first wrote the pretty-printing functions in Latex (using mathematical notation to write down a specification of the intended functionality – which we referred to as the formalism). We defined a function pretty which takes surface terms to a Doc. Some surface terms contained other entities such as 'qualifiers' or 'patterns' and so we had to write helper functions for these entities.

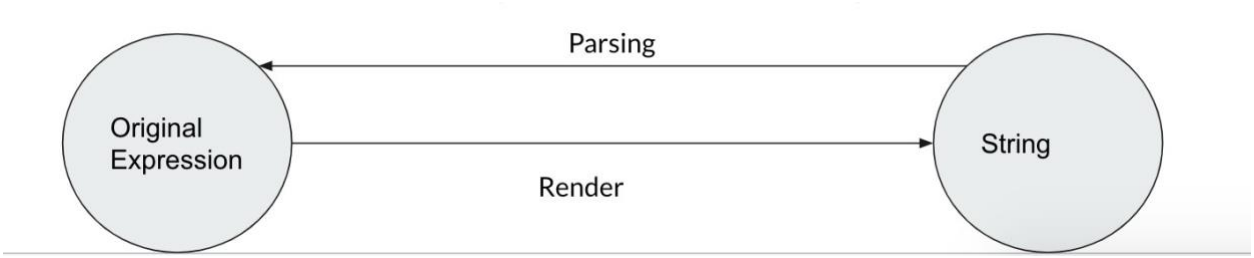
Once the formalism was ready, we moved onto implementing these pretty printing functions in PureScript.

Then we moved onto satisfying the round-tripping property. We had 66 tests, and each test had an associated fluid expression. The tests would only pass if the generated string was par sable (so we turn the expression to a String using render pretty s then we use the parser to turn the String back into an expression). Note 'render' is a function which takes a Doc to a String. So, now we had to make the tests pass (this involved adding brackets and white spaces in several places).

In more detail: we now had the following holding.



However, we now wanted the new expression to equal the original expression (this is the round-tripping property) so we added a further restriction into the tests to ensure this held. This led to tests failing again, but eventually we had all tests passing and the round-tripping property holding:



Next, we included the hillightIf function in our pretty-printing functions. For example, hillightIf can take a Boolean and a Doc and return another Doc (which given the Boolean True and a Doc, returns the Doc with underscores around it and given False returns the Doc as it was passed in). The idea here is to relate the output to the input (e.g., if we have the code : if x then print 5 else print 6 and the output is 5, we'd have underscores around x to show this is what led to 5 being printed). We thus added this function to our pretty printing functions.

For the round-tripping property to hold, we added brackets everywhere (e.g. if x then y - we had brackets around x and around y). Some brackets however were redundant (e.g. if True then print 5) but others were necessary (eg. if true then (let g in s)). So, to remove redundant brackets we assigned each expression an expression type (which we called Simple or Expression). The idea was that for expressions which needed further evaluation (binary applications, applications, let statements) we would call appropriate helper functions. These helper functions would evaluate individual components of the expression and if these components were Simple expressions, the brackets would not be needed otherwise we would add the brackets.

Then, we had to consider binary precedence. Initially we would add brackets around all binary expressions (so e.g., we'd have (5+(3+2)) however we only needed the brackets for cases like (5+3)*(9+2). So, we added a helper function for binary applications which takes individual components of a binary application and the precedence of the original binary application as input. If these individual components are themselves binary applications then we get their precedence and compare to the precedence of the original binary application. If the precedence of the component is larger, we do not require brackets otherwise we do.

Next steps for the project:

For the highlighting parts of the formalism, rather than using underscores we could use colors as next step and perhaps even use different colors to explain different parts of the output (e.g., if x then print hello if y then print bye so if the output is hello bye, we could highlight the hello and x with the same color and the bye and y with an alternative color).

Positive impact from the project and what was learnt from the experience:

This opportunity has had a huge positive impact from a confidence perspective. Talking to my supervisor and PhD student, discussing ideas and plans with them helped grow my communication and confidence skills.

A first proper introduction to the research industry was greatly appreciated, and has firmed my decision of entering research in the future.

Overall, this was a great chance to work with amazing and passionate individuals and I am thankful to have received this opportunity as well as the constant support, confidence and learning my supervisor and colleagues gave to me throughout this journey.