PROP – Assignment 2

Joakim Berglund `jobe7147`
Billy Jaime Beltran `bibe1744`

December 2016

1 Macros and functions

In order to manipulate expressions in code you need to use macros, because unlike functions, macros receive their arguments as unevaluated code. In the case of the safe macro we had a need to work with unevaluated code as we were providing additional functionality during the evaluation. If we had a function, the expression would have first been evaluated before passing to it. This would make our *try-catch-esque* wrapper not do its job. Thus, it is naturally more elegant to use a compile-time macro at this point.

Macros changes the abstract syntax tree of the program during compile time, instead of during runtime [1]. This allows us to make it so that our *syntactic sugar* additions seem like native constructs.

There are however ways to turn these macros into functions. One way is to wrap the macro in a function. Another way would be to take strings as input, and then evaluate the strings in the functions. The first way would technically be a function, but containing a macro. The other way would be, maybe not more complex, but longer, more error prone and less clear. It would additionally require that anyone wanting to use these functions properly transform their code into strings/lists before sending them.

Our SQL macro takes several arguments in a way that provides a very similar syntax to the one usual in SQLs. If we had written an analogous *function*, we would have had to require all the arguments to be escaped with the quote symbol or encased as a big text string which is very unidiomatic for SQL and Clojure.

The equivalent function would have to handle the string input, parse it by breaking out the SQL keywords and their arguments, and only then proceed to evaluate the query. This leads to a lack of clarity for the end-developer as all they end up doing is throwing in a large string as the function argument. You're no longer working with SQL in Clojure, you're working with a string. All the advantages of defining a domain-specific language are erased. You lose any form of error prevention automatically provided for free in Clojure as the string is programmatically opaque. Thus, with a macro, instead of writing and manipulating strings, we gain the ability to manipulate actual code with our own code.

Macros make it possible to write code that would otherwise be unnecessarily complex. By using macros our safe and select macros are implemented in a clean way, instead of either parsing a string or quoting the input. If we had to write our functions using string parsing, the ability for the end-developers to examine the code and identify any possibly errors would be much hampered as they would have to traverse a large amount of string manipulation-related code before even reaching any syntactical analysis.

2 References

[1] E. Normand, *When to use a macro in clojure*, [Online; Accessed 13-December-2016], 2014. [Online]. Available: `http://www.lispcast.com/when-to-use-a-macro`.