Discussion questions for *Template Meta-programming for Haskell*

Comp 150PLD

October 15, 2014

1 Check your understanding

1. What is *compile-time meta-programming*?
2. What can compile-time meta-programming be used for?
3. What is a splice and how is it notated in this paper?
4. What is a quasi-quotation and how is it notated?
5. How does the `gen` function work? How would you modify the definition of gen to support floats as an additional kind of format instructions?
6. Why can't we write the function `sel` using the quotation notation?
7. What is reification and why is it useful?
8. Explain what the problem is in the `cross2b` function and why `cross2a` doesn't have the same problem.
9. What services does the quotation monad provide?
10. Use Template Haskell data types to represent the apply function: `apply (f, x) = f x`.
11. Why is it illegal for meta-level operators like splices and quotations to appear in generated code?
12. Why is the code `f x = $(zipN x)` illegal at the top-level?
13. Explain how Haskell guarantees that well-typed programs cant "go wrong" at runtime. What do the three state B, C, and S in Figure 1 represent? What do the levels represent?
14. Explain how Haskell processes groups of declarations that contain splices and why it adopts this approach.
15. Why are declaration splices restricted to top-level?
16. Explain how the function `qIO :: IO a -> Q a` makes it so compiling your program can delete your entire file store.
17. What are *original names* and what are they used for?

2 Evaluation Questions

1. What are the advantages and disadvantages of having the language that executes at compile time be the same as the language that executes at run time?

2. What are the three layers of Template Haskell and how do they relate to each other? What are the advantages and disadvantages of using each layer?

3. What does the type of `sel` say about the generated code? What doesn't it say? Could the type be made more precise? Why or why not?

4. Template Haskell adopts the principle that *every occurrence of a variable is bound to the value that is lexically in scope at the occurrence site in the original source program, before any template expansion.* Give examples to illustrate why this property is tricky to guarantee. How do the designers of Template Haskell ensure this property?

Useful reference:
`https://hackage.haskell.org/package/template-haskell-2.8.0.0/docs/Language-Haskell-TH.html`