Discussion questions for Haskell Quasiquoting

Scribes: Michael James, Sophia Panuthos, Louis Rassaby

1. Definitions:
   a. Object language: The language in which an object program is written in, which is the code that is manipulated by the meta-program
   b. Metalanguage: The language in which the meta-program is written, which is the program that manipulates the code written in the object language. The most obvious example is a compiler.
      i. In Template Haskell: the object language and the object are the same
      ii. In this paper: object language is C (mostly), meta-language is Haskell

2. Examples of situations where it is useful for the object and meta languages to be different
   a. EDSL
   b. If there’s a natural representation of the ideas not expressible in the metalanguage
   c. You can have different object languages talk to each other through the metalanguage

3. What is quasiquotation?
   a. Code that is, at compile time, translated into Haskell abstract syntax
   b. The quote is evaluated at compile time and used elsewhere in the Haskell program
   c. Splices happen automatically (different from Template Haskell paper)

4. What is anti-quotation?
   a. It escapes the object language to permit metalanguage insertion inside the quasiquote
   b. e.g., subst function
   c. Notation depends on the how the object language designer decides to implement it
   d. Implemented in the quasiquotation implementation, so the sigil (i.e., $) can be arbitrarily defined.
   e. In Template Haskell, quasiquotation is implemented automatically, but when the object language can be different from the meta-language, it cannot be automatically provided since the syntax may overlap with the object language
   f. The anti-quotation does not have to be pure Haskell, it can provide extra information (e.g., $ty::t$ $x$)

5. How is quasiquotation related to Template Haskell?
   a. Template Haskell provides the splicing infrastructure
   b. Quasiquotation is an extension of TH, you—the language designer—must write the parser, type checking (if you want), and anti-quotation
c. TH has nesting of quoting and antiquoting but quasiquoting does not make this easy
d. General syntax, and $ convention
e. Abstract syntax for Haskell borrowed from Template Haskell
f. TH can track the bound variables but QQ cannot

6. Explain what happens when the Haskell compiler encounters the add function... who is responsible for defining the syntax that defines the antiquotation
   a. Haskell will produce a function that is parameterized by the input. add :: Int -> (Int -> Int)
   b. antiquotation annotated with $ ... $

7. Figure 3:
   a. optimizing antiquated code by mapping operators so that the operation is applied in the metalanguage at compile time
   b. when f is evaluated it gets the folded constraints and does not have to perform the operation.

8. peep:
   a. The left hand side pattern matches on the assembly string using & for antiquotes and produces an abstract syntax for a pattern
   b. The right hand side produces abstract syntax for an expression
   c. The compiler splices the expression for where it encounters the pattern

9. Why is it important that quasiquotation parsers are passed line numbers?
   a. Error messages that are relative to the object code

10. Subst:
    a. recursively substitutes and uses quasi-quotation to make the code clear

11. lame and lamp?
    a. lambda expression and lambda pattern, for pattern matching
    b. maps to abstract structures instead of to haskell functions because of types

12. .. Skipped question during discussions ..
13. .. Skipped question during discussions ..
14. .. Skipped question during discussions ..
15. Explain why the type guarantees associated with quasiquotation can’t ensure generated the C code is type safe
    a. Haskell cannot type check C code.
    b. Anti-quotation introduces potential type holes
    c. Quasiquotation can guarantee syntactic correctness
    d. If you have a C type checker in Haskell and didn’t quote any C types, then potentially you could guarantee C type-safety

1. What extra work must the user do when using quasiquotes as in this paper instead of quasiquotes in the TH paper?
   a. You’ve got to write the parser and the evaluator

2. What are the Pros and Cons of implementing an EDSL by providing a function compile
a. Pros
   i. Easy to write an interpreter
   ii. Strings can come from outside the program, e.g. a real interpreter
   iii. Broader user base?

b. Cons
   i. Lose all compile-time guarantees, like type-safety
   ii. Can’t have anti-quotation or pattern-matching

3. Pros and Cons of a EDSL as a combinator library (e.g., slideshow)
   a. Pros
      i. No new syntax to learn
      ii. Some amount of pattern matching
      iii. Type-checking, if your type system in embeddable in Haskell
   b. Cons
      i. Sometimes very clunky syntax
      ii. You’ve got to build your own type checking
      iii. Pattern matching isn’t as expressive as it could be because you can’t evaluate on the left side of the equation (can’t match against the return value of a function)

4. Pros and Cons of an EDSL with quasiquotation over as a standalone language
   a. Pros (quasiquotation)
      i. Could use external libraries
   b. Cons (Standalone)
      i. A standalone language will be faster
      ii. Same Haskell user base issue
      iii. Have to build your own libraries