To: Matthias Felleisen
From: Jakob Hain and Kevin Zhang
Subject: Racket-Docs Implementation

We planned to build a language which:

1. Lets the user write documentation alongside Racket code.
2. Lets the user compile this documentation into a Scribble website.
3. Verifies examples in the documentation.
4. Adds weak type-checking using the documentation forms. For example, say the user documents that a method should be given 2 strings, and then runs the method with 3 integers. The resulting code should fail to compile, and the user should be notified that they violated the function type.
5. Resolves identifier references within the documentation. For example, in DrRacket, if the user hovers over a type inside a signature, the IDE should draw an arrow pointing to where the type was defined.
6. Overrides the reader, so that raw text (e.g. purpose statements) don't need to be quoted, they would automatically be parsed as a string.

We believe we accomplished the first three points. Using Racket-Docs's **define-docs** and **define-data** forms, a user can document functions, constants, macros, and data-definitions. In the documentation, they can add type signatures, purpose statements, and examples, and they can describe accumulators, generative recursion schemes, and side effects. The user can then compile this documentation, via a "Compile Docs" button added to the IDE, and Racket-Docs will generate a Scribble website containing all of the above information. Additionally, Racket-Docs will add assertions to the **test** submodule to check the examples within the documentation. Within **define-docs**, examples contain an "actual" expression with an "expected" output, and Racket-Docs will check that they are equal. Within **define-data**, Racket-Docs will check that the data examples' types match with the data definition.

We believe we partially accomplished point #4. Racket-Docs has weak type checking: if a function is called with wrongly-typed arguments, or defined with a wrongly-typed output, then Racket-Docs will raise a compile-time error. However, type-checking is slow: it takes Racket-Docs about 10 seconds to run **prototype-implemented.rkt**, which is 128 lines long, on a 2.3 GHz i7 2012 Macbook Pro. Furthermore, our language has only a few built-in typed functions, such as **cons**, **first**, and **rest**. Language users who wish to type other built-in functions will have to make documentation forms for them. This because Racket has thousands of other built-in functions; instead of typing those, we focused on improving the core type system.

We believe we partially accomplished point #5. DrRacket can infer where identifiers are bound when the identifiers are in types, examples, or accumulators. But it can't resolve "identifiers" within purpose statements and interpretations. This is because the purpose statements and interpretations are strings, so the "identifiers" themselves are also strings, and Racket can't resolve them. In order to allow identifiers spliced in purpose statements, we would need to override the reader (point #6), and we didn't accomplish this.

We didn't accomplish point #6. Purpose statements, interpretations, and other raw text forms need to be represented as strings. Initially, we believed that these extra quotes would be tedious, and adding them would make the syntax ugly. After some experimentation by documenting our own language, however, we realized that the quotes weren't a big deal, and implementing the core functions of our language (such as documentation forms, compiling docs, and type-checking) were more important than refining documentation strings.

We believe that the most difficult thing to accomplish was the type system. We initially tried to use an existing type system, such as Turnstile or Typed Racket, but both of them proved to be problematic. Turnstile didn't support literal expression types, such as **5** or **(cons 'foo Foobar)**. Typed Racket was too complicated, and we were unable to interpret the error messages it was providing. Finally, we built our own type system, which is based off the core implementation of Turnstile. We use syntax properties to store information about types, and override **#%app** and **#%datum**. We represent types differently however: types are thunks that return structs instead of syntax.