To: Matthias Felleisen, boss
From: Alex Knauth
Milo Turner
Subject: Implementing Module / Functor Systems with Macros

Completed Aspects of the Proposal

For our module language, we have implemented mods, sigs, opaque types, functors, pi signatures, `where`-refinements of opaque types, and submodules.

Incomplete Aspects

In the core language, we didn't get to implemeneting polymorphism or newtypes. This was mainly because the core language was not as big of a focus as the module language. There are some issues with type aliases that need to be resolved, in particular recursive type aliases currently can put the type checker into an infinite loop.

In the module language, we only implemented function application with module-path arguments. This restriction requires the programmer to manually convert complex module expressions into an ANF-like form, using submodules to bind intermediate modules. The reason for this restriction was that it allowed us to avoid deciding equality between non-path module expressions. This bypasses undecidabilty issues that arise from the dependently-typed nature of module functors. This trade-off was also made by *A Modular Module System*.

We decided to implement submodules instead of a $\texttt{let}^{\text{m}}$ form, but we still may implement this form in the future. Additionally, some aspects of submodules remain to be implemented, such as the ability to specify opaque submodules specifications in sigs, and `where`-refinements of those opaque submodules.

Implementation Difficulties

Signature matching ended up being more complicated than we anticipated. Matching `sig`s required building environments from signatures; our implementation has undergone many iterations and changes in data representation. There were many consequences of particular representations that we did not initially take into account.

One of the trickier algorithms was qualifying module componenents as we select them using `#%dot`. Bugs in the qualifying algorithm tended to be subtle, and it was often difficult to come up with good edge cases which reveal when names are incorrectly qualified.

The two problems above are both instances of the more general problem of managing environments and scope. Similar problems surfaced when converting elements of environments into signatures, and performing capture-avoiding substitution during functor application and `where`-refinement.

Future Plans

In addition to aspects of submodules mentioned earlier, we have plans to add other module system features that were not part of the original proposal. For instance, we would like to create a module form that allow types and functions from other Racket libraries to be used.

We also plan to modify our module implementation so that it is decoupled further from the core language. This would allow us to determine what interface we need Hackett to fulfil, so that we can extend that language with a module system like the one presented here.