A parsimonious module system

TODO

1 Introduction

TODO: BLABLA Intro

We propose an *expressive*, *syntactic* and *parsimonious* module system. By expressive, we mean that we support a rich subset of features that have so far not been formalized together and allow the user for rich manipulation of modules, notably transparent ascriptions, module aliases, enrichment constraints and applicative functors. Unlike work like **?**, our rules are defined *syntactically*, directly on the concrete syntax. This makes the rule easier to grasp, make deriving an inference algorithm almost immediate, and ensure that error messages and typing information are easier to surface to the user. Finally, this system is *parsimonious*, in that it does the minimum amount of expansion required by using ideas similar to explicit substitutions for module operations. Module types can be quite large, and limiting the expansion can simplify error messages and improve the speed of the typechecker.

2 Motivations

TODO: Explain functor aliasing.

TODO: Explain transparent ascriptions, with constraints

TODO: Explain avoidance problem

TODO: Add Examples

TODO: Motivate incrementality

3 An ML module calculus

We now introduce our module calculus. First, let us defined some conventions: lowercase meta-variables such as x, e or t are used for the core language; capitalized meta-variables such as X, M, S represent modules; calligraphic meta-variables such as $\mathcal{X}$, $\mathcal{M}$, $\mathcal{S}$ represent module types.

3.1 Syntax

The syntax is defined in Figure 1. The module expression language contains paths P, which might be variables X or qualified accesses $P.X$, parameterized modules i.e. *functors* $((X : \mathcal{M}) \to M)$, module type annotations i.e. *ascriptions*, both opaque $(M : \mathcal{M})$ and transparent $(M <: \mathcal{M})$, and finally structures $\mathtt{struct}_X\ S\ \mathtt{end}$. Structures contains a list of declarations, noted D, which can be value, type, module or module type bindings. Structures are also annotated with a “Self” variable, which represent the object through which other field in the module should be accessed.

In our core calculus, module types are stratified. $\mathcal{M}$ represent module types with operations, while $\mathcal{M}^\circ$ are raw module types without initial operations (i.e., in head normal form). Raw module types can be (qualified) module type variables $\mathcal{X}$ or $\mathcal{P}.\mathcal{X}$, functor types $(\mathcal{X} : \mathcal{M}_1) \to \mathcal{M}_2$, signatures $\mathtt{sig}_X\ \mathcal{S}\ \mathtt{end}$, and singletons $(= \mathcal{P})$. As with structures, signatures are a list of declarations noted $\mathcal{D}$, and a “Self” variable. Allowed operations strengthening a module by a path, noted $\mathcal{M}/\mathcal{P}$, enriching a module with additional constraints, noted $\mathcal{M}_{[\mathcal{C}]}$, and locally binding a module name, noted $\mathtt{let}\ X : \mathcal{M}\ \mathtt{in}\ \mathcal{M}'$.

As described before, there are two kinds of paths. The first, P, is mostly used for *dynamic* components, and is composed of a list of modules of the form $A.B.C$. The second, $\mathcal{P}$, is mostly used for *static* components such as types, and can additionally contain functor applications and transparent ascriptions, for instance $F(X).Y$ or $(X <: \mathcal{M}).Y$

Finally, the core language is left largely undefined in our calculus, except for qualified accesses.

$\mathcal{M}[X \mapsto Y]$ denotes the substitution of variable X by variable Y in $\mathcal{M}$. We only substitute variables by other variables.

Path

$$\begin{array}{rl}
P ::= & X \mid P.X \\
\mathcal{P} ::= & X \mid \mathcal{P}.X \mid \mathcal{P}_1(\mathcal{P}_2) \mid (\mathcal{P} <: \mathcal{M})
\end{array}$$

Module Expressions

$$\begin{array}{rlr}
M ::= & P & \text{(Variables)} \\
& \mid (M : \mathcal{M}) & \text{(Opaque Ascription)} \\
& \mid (M <: \mathcal{M}) & \text{(Transparent Ascription)} \\
& \mid M_1(M_2) & \text{(Functor application)} \\
& \mid (X : \mathcal{M}) \to M & \text{(Functor)} \\
& \mid \mathtt{struct}_X\ S\ \mathtt{end} & \text{(Structure)}
\end{array}$$

Structures

$$\begin{array}{rlr}
S ::= & \varepsilon \mid D; S & \\
D ::= & \mathtt{let}\ x = e & \text{(Values)} \\
& \mid \mathtt{type}\ t = \tau & \text{(Types)} \\
& \mid \mathtt{module}\ X = M & \text{(Modules)} \\
& \mid \mathtt{module\ type}\ \mathcal{X} = \mathcal{M} & \text{(Module types)}
\end{array}$$

Core language

$$\begin{array}{rlr}
e ::= & P.x & \text{(Qualified variable)} \\
& \mid \ldots & \text{(Other expressions)} \\
\tau ::= & \mathcal{P}.t & \text{(Qualified type)} \\
& \mid \ldots & \text{(Other types)}
\end{array}$$

Module types

$$\begin{array}{rlr}
\mathcal{M}^\circ ::= & \mathcal{X} \mid \mathcal{P}.\mathcal{X} & \text{(Variables)} \\
& \mid (= \mathcal{P}) & \text{(Alias)} \\
& \mid (X : \mathcal{M}_1) \to \mathcal{M}_2 & \text{(Functor)} \\
& \mid \mathtt{sig}_X\ \mathcal{S}\ \mathtt{end} & \text{(Signature)} \\
\mathcal{M} ::= & \mathcal{M}/\mathcal{P} & \text{(Strengthening)} \\
& \mid \mathtt{let}\ X : \mathcal{M}\ \mathtt{in}\ \mathcal{M}' & \text{(Let)} \\
& \mid \mathcal{M}_{[\mathcal{C}]} & \text{(Enrichment)} \\
& \mid \mathcal{M}^\circ &
\end{array}$$

Enrichment

$$\begin{array}{rl}
\mathcal{C} ::= & P.t = \tau \\
& \mid P : \mathcal{M}
\end{array}$$

Signatures

$$\begin{array}{rlr}
\mathcal{S} ::= & \varepsilon \mid \mathcal{D}; \mathcal{S} & \\
\mathcal{D} ::= & \mathtt{val}\ x : \tau & \text{(Values)} \\
& \mid \mathtt{type}\ t = \tau & \text{(Types)} \\
& \mid \mathtt{type}\ t & \text{(Abstract types)} \\
& \mid \mathtt{module}\ X : \mathcal{M} & \text{(Modules)} \\
& \mid \mathtt{module\ type}\ \mathcal{X} = \mathcal{M} & \text{(Module types)}
\end{array}$$

Environments

$$\Gamma ::= \mathcal{S}$$

Figure 1: Module language

3.2 Notations

Let us now define a few notations that we will use in the rest of this article. We will then describe how these various construction interact on specific features.

We define the following type judgements on modules. Note that the subtyping judgement also *returns* a module type, which is the result of the transparent ascription between the two modules.

$\Gamma \vdash M : \mathcal{M}$ **:** The module M is of type $\mathcal{M}$ in Γ. See Figure 2.

$\Gamma \vdash \mathcal{M} <: \mathcal{M}' \rightsquigarrow \mathcal{M}_r$ **:** The module type $\mathcal{M}$ is a subtype of $\mathcal{M}'$ in environment Γ, and their ascription is $\mathcal{M}_r$. See Figure 3.

$(\!|\mathcal{M}|\!)_\Gamma = \mathcal{M}^\circ$ **:** The module type with operations $\mathcal{M}$ can be reduced to a module type without operations $\mathcal{M}^\circ$ in environment Γ. See Figures 4 to 6. **TODO: Change bracket style ...**

$$\textsc{ModVar}\ \frac{P \in \Gamma}{\Gamma \vdash P : (= P)} \qquad \frac{\Gamma \vdash M : \mathcal{M}' \quad \Gamma \vdash \mathcal{M}' <: \mathcal{M} \rightsquigarrow \_}{\Gamma \vdash (M : \mathcal{M}) : \mathcal{M}} \qquad \frac{\Gamma \vdash M : \mathcal{M}' \quad \Gamma \vdash \mathcal{M}' <: \mathcal{M} \rightsquigarrow \mathcal{M}_r}{\Gamma \vdash (M <: \mathcal{M}) : \mathcal{M}_r}$$

$$\frac{\Gamma \vdash M_f : (X : \mathcal{M}_a) \to \mathcal{M}_r \quad Y \text{ fresh} \quad \Gamma \vdash M : \mathcal{M} \quad \Gamma \vdash \mathcal{M} <: \mathcal{M}_a \rightsquigarrow \mathcal{M}_c}{\Gamma \vdash M_f(M) : \texttt{let } Y : \mathcal{M}_c \texttt{ in } \mathcal{M}_r[X \mapsto Y]}$$

$$\frac{\Gamma \vDash \mathcal{M} \quad X \notin \Gamma \quad \Gamma; \texttt{module } X : \mathcal{M} \vdash M : \mathcal{M}'}{\Gamma \vdash (X : \mathcal{M}) \to M : (X : \mathcal{M}) \to \mathcal{M}'} \qquad \frac{\Gamma \vdash_X S : \mathcal{S}}{\Gamma \vdash \texttt{struct}_X\ S\ \texttt{end} : \texttt{sig}_X\ \mathcal{S}\ \texttt{end}} \qquad \frac{}{\Gamma \vdash_Y \varepsilon : \varepsilon}$$

$$\frac{\Gamma \vdash e : \tau \quad x \notin \Gamma(Y) \quad \Gamma; \texttt{val } Y.x : \tau \vdash_Y S : \mathcal{S}}{\Gamma \vdash_Y (\texttt{let } x = e; S) : (\texttt{val } x : \tau; \mathcal{S})} \qquad \frac{\Gamma \vdash M : \mathcal{M} \quad X \notin \Gamma(Y) \quad \Gamma; \texttt{module } Y.X : \mathcal{M} \vdash_Y S : \mathcal{S}}{\Gamma \vdash_Y (\texttt{module } X = M; S) : (\texttt{module } X : \mathcal{M}; \mathcal{S})}$$

$$\frac{\Gamma \vDash \tau \quad t \notin \Gamma(Y) \quad \Gamma; \texttt{type } Y.t = \tau \vdash_Y S : \mathcal{S}}{\Gamma \vdash_Y (\texttt{type } t = \tau; S) : (\texttt{type } t = \tau; \mathcal{S})} \qquad \frac{\Gamma \vDash \mathcal{M} \quad \mathcal{X} \notin \Gamma(Y) \quad \Gamma; \texttt{module type } Y.\mathcal{X} = \mathcal{M} \vdash_Y S : \mathcal{S}}{\Gamma \vdash_Y (\texttt{module type } \mathcal{X} = \mathcal{M}; S) : (\texttt{module type } \mathcal{X} = \mathcal{M}; \mathcal{S})}$$

Figure 2: Module typing rules – $\Gamma \vdash M : \mathcal{M}$

In our calculus, environments are simply signature on which we can "query" fields. To properly model our module calculus, we need to carefully consider how these environments are accessed. For this purpose, we consider the following operations on environments, which are defined in Figure 7

$\Gamma(P)$ **:**	Lookup the path P in Γ to a module or module type.
$\text{normalize}_\Gamma(\mathcal{P})$ **:**	Normalizes the path $\mathcal{P}$ in Γ to a canonical path.
$\text{resolve}_\Gamma(\mathcal{P})$ **:**	Resolve the path $\mathcal{P}$ in Γ to its shape (i.e., either an arrow or a signature).
$\text{shape}_\Gamma(\mathcal{M})$ **:**	Resolve the module $\mathcal{M}$ in Γ to its shape (i.e., either an arrow or a signature).

3.3 Structures and qualified accesses

3.4 Subtyping and transparent ascriptions

3.5 Functors

3.6 Module operations

$$\frac{\mathrm{normalize}_\Gamma(\mathcal{P}) = \mathrm{normalize}_\Gamma(\mathcal{P}')}{\Gamma \vdash \mathcal{P} = \mathcal{P}'} \qquad \frac{\Gamma \vdash (\!|\mathcal{M}|\!)_\Gamma <: (\!|\mathcal{M}'|\!)_\Gamma \rightsquigarrow \mathcal{M}''}{\Gamma \vdash \mathcal{M} <: \mathcal{M}' \rightsquigarrow \mathcal{M}''} \qquad \frac{\Gamma \vdash \Gamma(\mathcal{P}) <: \mathcal{M}^\circ \rightsquigarrow \mathcal{M}}{\Gamma \vdash \mathcal{P} <: \mathcal{M}^\circ \rightsquigarrow \mathcal{M}}$$

$$\frac{\Gamma \vdash \mathcal{M}^\circ <: \Gamma(\mathcal{P}) \rightsquigarrow \mathcal{M}}{\Gamma \vdash \mathcal{M}^\circ <: \mathcal{P} \rightsquigarrow \mathcal{M}} \qquad \frac{\Gamma \vdash \mathcal{P} = \mathcal{P}' \qquad \Gamma \vdash \mathrm{resolve}_\Gamma(\mathcal{P}) <: \mathcal{M} \rightsquigarrow \mathcal{M}'' \qquad \Gamma \vdash \mathcal{M}'' <: \mathcal{M}' \rightsquigarrow \_}{\Gamma \vdash (= (\mathcal{P} <: \mathcal{M})) <: (= (\mathcal{P}' <: \mathcal{M}')) \rightsquigarrow (= (\mathcal{P}' <: \mathcal{M}'))}$$

$$\frac{\Gamma \vdash \mathcal{P} = \mathcal{P}' \qquad \Gamma \vdash \mathrm{resolve}_\Gamma(\mathcal{P}) <: \mathcal{M} \rightsquigarrow \mathcal{M}'' \qquad \Gamma \vdash \mathcal{M}'' <: \mathrm{resolve}_\Gamma(\mathcal{P}) \rightsquigarrow \_}{\Gamma \vdash (= (\mathcal{P} <: \mathcal{M})) <: (= \mathcal{P}') \rightsquigarrow (= \mathcal{P}')}$$

$$\frac{\Gamma \vdash \mathcal{P} = \mathcal{P}' \qquad \Gamma \vdash \mathcal{M}'' <: \mathcal{M}' \rightsquigarrow \_}{\Gamma \vdash (= \mathcal{P}) <: (= (\mathcal{P}' <: \mathcal{M}')) \rightsquigarrow (= (\mathcal{P}' <: \mathcal{M}'))} \qquad \frac{\Gamma \vdash \mathcal{P} = \mathcal{P}'}{\Gamma \vdash (= \mathcal{P}) <: (= \mathcal{P}') \rightsquigarrow (= \mathcal{P}')}$$

$$\frac{}{\Gamma \vdash (= \mathcal{P}) <: \mathcal{M}^\circ \rightsquigarrow (= (\mathcal{P} <: \mathcal{M}^\circ))} \qquad \frac{\Gamma \vdash \mathcal{M}'_a <: \mathcal{M}_a \rightsquigarrow \_ \qquad \Gamma, \texttt{module}\ X : \mathcal{M}'_a \vdash \mathcal{M}_r <: \mathcal{M}'_r \rightsquigarrow \mathcal{M}''_r}{\Gamma \vdash (X : \mathcal{M}_a) \to \mathcal{M}_r <: (X : \mathcal{M}'_a) \to \mathcal{M}'_r \rightsquigarrow (X : \mathcal{M}'_a) \to \mathcal{M}''_r}$$

$$\frac{\mathcal{B} = \mathcal{S} \times \mathcal{S}'[X' \mapsto X] \qquad \forall \mathcal{D}_x, \mathcal{D}'_x \in \mathcal{B}, \quad \Gamma; (\texttt{module}\ X = \texttt{sig}_X\ \mathcal{S}\ \texttt{end}) \vdash \mathcal{D}_x <: \mathcal{D}'_x \rightsquigarrow \mathcal{D}''_x}{\Gamma \vdash \texttt{sig}_X\ \mathcal{S}\ \texttt{end} <: \texttt{sig}_{X'}\ \mathcal{S}'\ \texttt{end} \rightsquigarrow \texttt{sig}_X\ \overline{\mathcal{D}''_x}\ \texttt{end}}$$

$$\frac{\Gamma \vdash \mathcal{M}_1 <: \mathcal{M}_2 \rightsquigarrow \mathcal{M}}{\Gamma \vdash (\texttt{module}\ X : \mathcal{M}_1) <: (\texttt{module}\ X : \mathcal{M}_2) \rightsquigarrow (\texttt{module}\ X : \mathcal{M})}$$

$$\frac{\Gamma \vdash \mathcal{M}_1 <: (\texttt{sig}\ \ \texttt{end}) \rightsquigarrow \mathcal{M}}{\Gamma \vdash (\texttt{module}\ X : \mathcal{M}_1) <: \emptyset \rightsquigarrow (\texttt{module}\ X : \mathcal{M})}$$

$$\frac{\Gamma \vdash \mathcal{M}_1 <: \mathcal{M}_2 \rightsquigarrow \mathcal{M}}{\Gamma \vdash (\texttt{module type}\ \mathcal{X} = \mathcal{M}_1) <: (\texttt{module type}\ \mathcal{X} = \mathcal{M}_2) \rightsquigarrow (\texttt{module type}\ \mathcal{X} = \mathcal{M})}$$

$$\frac{}{\Gamma \vdash (\texttt{module type}\ \mathcal{X} = \mathcal{M}) <: \emptyset \rightsquigarrow (\texttt{module type}\ \mathcal{X} = \mathcal{M})}$$

$$\frac{\Gamma \vdash \tau_1 <: \tau_2}{\Gamma \vdash (\texttt{val}\ x : \tau_1) <: (\texttt{val}\ x : \tau_2) \rightsquigarrow (\texttt{val}\ x : \tau_2)} \qquad \frac{}{\Gamma \vdash (\texttt{val}\ x : \tau_1) <: \emptyset \rightsquigarrow \emptyset}$$

$$\frac{}{\Gamma \vdash (\texttt{type}\ t) <: (\texttt{type}\ t) \rightsquigarrow (\texttt{type}\ t)} \qquad \frac{\Gamma \vdash \tau_1 <: \tau_2}{\Gamma \vdash (\texttt{type}\ t = \tau_1) <: (\texttt{type}\ t = \tau_2) \rightsquigarrow (\texttt{type}\ t = \tau_1)}$$

$$\frac{\Gamma \vdash t <: \tau}{\Gamma \vdash (\texttt{type}\ t) <: (\texttt{type}\ t = \tau) \rightsquigarrow (\texttt{type}\ t = \tau)} \qquad \frac{}{\Gamma \vdash (\texttt{type}\ t = \tau) <: (\texttt{type}\ t) \rightsquigarrow (\texttt{type}\ t = \tau)}$$

$$\frac{}{\Gamma \vdash (\texttt{type}\ t = \tau) <: \emptyset \rightsquigarrow (\texttt{type}\ t = \tau)} \qquad \frac{}{\Gamma \vdash (\texttt{type}\ t) <: \emptyset \rightsquigarrow (\texttt{type}\ t)}$$

Figure 3: Module subtyping rules – $\Gamma \vdash \mathcal{M} <: \mathcal{M}' \rightsquigarrow \mathcal{M}_r$

$$
\begin{aligned}
\llparenthesis \mathcal{X}/\mathcal{P} \rrparenthesis_\Gamma &\to \llparenthesis \Gamma(\mathcal{X})/\mathcal{P} \rrparenthesis_\Gamma \\
\llparenthesis \mathcal{P}'.\mathcal{X}/\mathcal{P} \rrparenthesis_\Gamma &\to \llparenthesis \Gamma(\mathcal{P}'.\mathcal{X})/\mathcal{P} \rrparenthesis_\Gamma \\
\llparenthesis (= \mathcal{P}')/\mathcal{P} \rrparenthesis_\Gamma &\to (= \mathcal{P}') \\
\llparenthesis ((X : \mathcal{M}) \to \mathcal{M}')/\mathcal{P} \rrparenthesis_\Gamma &\to (X : \mathcal{M}) \to \mathcal{M}'/\mathcal{P}(X) \\
\llparenthesis \mathtt{sig}_X\ \mathcal{S}\ \mathtt{end}/\mathcal{P} \rrparenthesis_\Gamma &\to \mathtt{sig}_X\ \llparenthesis \mathcal{S}/\mathcal{P} \rrparenthesis_\Gamma\ \mathtt{end} \\
\\
\llparenthesis \mathtt{type}\ t = \tau; \mathcal{S}/\mathcal{P} \rrparenthesis_\Gamma &\to \mathtt{type}\ t = \mathcal{P}.t; \llparenthesis \mathcal{S}/\mathcal{P} \rrparenthesis_\Gamma \\
\llparenthesis \mathtt{type}\ t; \mathcal{S}/\mathcal{P} \rrparenthesis_\Gamma &\to \mathtt{type}\ t = \mathcal{P}.t; \llparenthesis \mathcal{S}/\mathcal{P} \rrparenthesis_\Gamma \\
\llparenthesis \mathtt{module}\ X : \mathcal{M}; \mathcal{S}/\mathcal{P} \rrparenthesis_\Gamma &\to \mathtt{module}\ X : (= \mathcal{P}.X); \llparenthesis \mathcal{S}/\mathcal{P} \rrparenthesis_\Gamma \\
\llparenthesis \mathtt{module\ type}\ \mathcal{X} = \mathcal{M}; \mathcal{S}/\mathcal{P} \rrparenthesis_\Gamma &\to \mathtt{module\ type}\ \mathcal{X} = \mathcal{M}; \llparenthesis \mathcal{S}/\mathcal{P} \rrparenthesis_\Gamma \\
\\
\llparenthesis \mathcal{M}/\mathcal{P}/\mathcal{P}' \rrparenthesis_\Gamma &\to \llparenthesis \mathcal{M}/\mathcal{P} \rrparenthesis_\Gamma \\
\llparenthesis \mathcal{M}_{[\mathcal{C}]}/\mathcal{P} \rrparenthesis_\Gamma &\to \llparenthesis \llparenthesis \mathcal{M}_{[\mathcal{C}]} \rrparenthesis_\Gamma/\mathcal{P} \rrparenthesis_\Gamma
\end{aligned}
$$

Figure 4: Module strengthening operation – $\mathcal{M}/\mathcal{P}$

$$
\begin{aligned}
\llparenthesis \mathcal{X}_{[\mathcal{C}]} \rrparenthesis_\Gamma &\to \llparenthesis \Gamma(\mathcal{X})_{[\mathcal{C}]} \rrparenthesis_\Gamma \\
\llparenthesis \mathcal{P}.\mathcal{X}_{[\mathcal{C}]} \rrparenthesis_\Gamma &\to \llparenthesis \Gamma(\mathcal{P}.\mathcal{X})_{[\mathcal{C}]} \rrparenthesis_\Gamma \\
\llparenthesis (= \mathcal{P})_{[\mathcal{C}]} \rrparenthesis_\Gamma &\to (= \mathcal{P}) && \text{when } \llparenthesis \text{resolve}_\Gamma(\mathcal{P})_{[\mathcal{C}]} \rrparenthesis_\Gamma \to \_ \\
\llparenthesis ((X : \mathcal{M}) \to \mathcal{M}')_{[\mathcal{C}]} \rrparenthesis_\Gamma &\to \mathtt{fail} \\
\llparenthesis \mathtt{sig}_X\ \mathcal{S}\ \mathtt{end}_{[\mathcal{C}]} \rrparenthesis_\Gamma &\to \mathtt{sig}_X\ \llparenthesis \mathcal{S}_{[\mathcal{C}]} \rrparenthesis_\Gamma\ \mathtt{end} \\
\\
\llparenthesis \mathtt{type}\ t = \tau; \mathcal{S}_{[t=\tau']} \rrparenthesis_\Gamma &\to \mathtt{type}\ t = \tau'; \mathcal{S} && \text{when } \Gamma \vdash \tau' <: \tau \\
\llparenthesis \mathtt{module}\ X : \mathcal{M}; \mathcal{S}_{[X:\mathcal{M}']} \rrparenthesis_\Gamma &\to \mathtt{module}\ X : \mathcal{M}'; \mathcal{S} && \text{when } \Gamma \vdash \mathcal{M}' <: \mathcal{M} \rightsquigarrow \_ \\
\llparenthesis \mathtt{module}\ X : \mathcal{M}; \mathcal{S}_{[X.P.t=\tau]} \rrparenthesis_\Gamma &\to \mathtt{module}\ X : \llparenthesis \mathcal{M}_{[P.t=\tau]} \rrparenthesis_\Gamma; \mathcal{S} \\
\llparenthesis \mathtt{module}\ X : \mathcal{M}; \mathcal{S}_{[X.P:\mathcal{M}']} \rrparenthesis_\Gamma &\to \mathtt{module}\ X : \llparenthesis \mathcal{M}_{[P:\mathcal{M}']} \rrparenthesis_\Gamma; \mathcal{S} \\
\llparenthesis (\mathcal{D}; \mathcal{S})_{[\mathcal{C}]} \rrparenthesis_\Gamma &\to \mathcal{D}; \llparenthesis \mathcal{S}_{[\mathcal{C}]} \rrparenthesis_\Gamma && \text{when } id(\mathcal{D}) \text{ is not a prefix of } id(\mathcal{C}) \\
\\
\llparenthesis (\mathcal{M}/P)_{[\mathcal{C}]} \rrparenthesis_\Gamma &\to \llparenthesis \llparenthesis \mathcal{M}/P \rrparenthesis_{\Gamma\,[\mathcal{C}]} \rrparenthesis_\Gamma \\
\llparenthesis \mathcal{M}_{[\mathcal{C}']\,[\mathcal{C}]} \rrparenthesis_\Gamma &\to \llparenthesis \llparenthesis \mathcal{M}_{[\mathcal{C}']} \rrparenthesis_{\Gamma\,[\mathcal{C}]} \rrparenthesis_\Gamma
\end{aligned}
$$

Figure 5: Enrichment operation – $\mathcal{M}_{[\mathcal{C}]}$

$$
\begin{aligned}
(\!|\texttt{let}\ X : (=\mathcal{P})\ \texttt{in}\ \mathcal{M}|\!)_\Gamma &\to \mathcal{M}[X \mapsto \mathcal{P}] \\
(\!|\texttt{let}\ X : \mathcal{M}\ \texttt{in}\ \mathcal{M}'|\!)_\Gamma &\to \mathcal{M}' && \texttt{when}\ X \notin FV(\mathcal{M}')
\end{aligned}
$$

TODO: Provide something more elaborate

Figure 6: Subst operation – $\texttt{let}\ X : \mathcal{M}\ \texttt{in}\ \mathcal{M}'$

$$
\frac{(\texttt{module}\ X : \mathcal{M}) \in \Gamma}{\Gamma(X) = \mathcal{M}}
\qquad
\frac{\mathrm{shape}_\Gamma(\Gamma(\mathcal{P})) = \texttt{sig}_Y\ \mathcal{S}_1; \texttt{module}\ X : \mathcal{M}; \mathcal{S}_2\ \texttt{end}}{\Gamma(\mathcal{P}.X) = \mathcal{M}[Y \mapsto \mathcal{P}]}
$$

$$
\frac{(\texttt{module type}\ \mathcal{X} = \mathcal{M}) \in \Gamma}{\Gamma(\mathcal{X}) = \mathcal{M}}
\qquad
\frac{\mathrm{shape}_\Gamma(\Gamma(\mathcal{P})) = \texttt{sig}_Y\ \mathcal{S}_1; \texttt{module type}\ \mathcal{X} = \mathcal{M}; \mathcal{S}_2\ \texttt{end}}{\Gamma(\mathcal{P}.\mathcal{X}) = \mathcal{M}[Y \mapsto \mathcal{P}]}
$$

$$
\frac{\Gamma(\mathcal{P}) = \mathcal{M} \qquad \Gamma \vdash \mathcal{M} <: \mathcal{M}' \leadsto \mathcal{M}_r}{\Gamma((\mathcal{P} <: \mathcal{M}')) = \mathcal{M}_r}
\qquad
\frac{\mathrm{shape}_\Gamma(\Gamma(\mathcal{P}_f)) = (X : \mathcal{M}_a) \to \mathcal{M}_r \qquad \Gamma \vdash (=\mathcal{P}_a) <: \mathcal{M}_a \leadsto \mathcal{M}}{\Gamma(\mathcal{P}_f(\mathcal{P}_a)) = \texttt{let}\ Y : \mathcal{M}\ \texttt{in}\ \mathcal{M}_r[X \mapsto Y]}
$$

(a) Lookup rules – $\Gamma(P) = \mathcal{M}$

$$
\mathrm{normalize}_\Gamma(\mathcal{P}) = \begin{cases} \mathrm{normalize}_\Gamma(\mathcal{P}') & \text{when } \Gamma(\mathcal{P}) = (=\mathcal{P}') \\ \mathcal{P} & \text{otherwise} \end{cases}
$$

$$
\mathrm{resolve}_\Gamma(\mathcal{P}) = \Gamma(\mathrm{normalize}_\Gamma(\mathcal{P})) /\ \mathrm{normalize}_\Gamma(\mathcal{P})
$$

(b) Path normalization and resolution

$$
\begin{aligned}
\mathrm{shape}_\Gamma(\mathcal{M}) &= \mathrm{shape}_\Gamma((\!|\mathcal{M}|\!)_\Gamma) \\
\mathrm{shape}_\Gamma((=\mathcal{P})) &= \mathrm{shape}_\Gamma(\mathrm{resolve}_\Gamma(\mathcal{P})) \\
\mathrm{shape}_\Gamma(\texttt{sig}_X\ \mathcal{S}\ \texttt{end}) &= \texttt{sig}_X\ \mathcal{S}\ \texttt{end} \\
\mathrm{shape}_\Gamma((X : \mathcal{M}) \to \mathcal{M}') &= (X : \mathcal{M}) \to \mathcal{M}'
\end{aligned}
$$

(c) Shape resolution – $\mathrm{shape}_\Gamma(\mathcal{M}) = \mathcal{M}'$

Figure 7: Environment access, normalization and resolution