1 Module types

To specify a module type, programmers would use the syntax of figure 1. It maps to the EBNF description given in figure 2, where X denotes an identifier, t a type and $\overline{X}$ a list of identifiers (more generally for all symbol A, $\overline{A}$ will denote a list of A). This formal syntax is translated to the one given in figure 3 by the inductive rules given in figure 4. This second syntax separates `param` declarations from the others[1] and puts them at the beginning of the module type definition.

```
defmoduletype B do
  [ $param X(X1, ..., Xn)
  | $opaque X(X1, ..., Xn)
  | $type X(X1, ..., Xn) = t
  | callback X : t
  ]*
end
```

Figure 1: Surface syntax proposal for module types

$$
\begin{aligned}
D_B^S ::= & \ \mathbf{param}\, X\overline{X} \mid \mathbf{type}\, X\overline{X} = T \mid \mathbf{opaque}\, X\overline{X} \mid \mathbf{callback}\, X : T \\
& \mid D_B^S; D_B^S \mid \epsilon \\
B^S ::= & \ \mathbf{defmoduletype}\, X\, \mathbf{do}\, D_B^S\, \mathbf{end}
\end{aligned}
$$

Figure 2: A first formal syntax for module types

$$
\begin{aligned}
D_B ::= & \ \mathbf{type}\, X\overline{X} = T \mid \mathbf{opaque}\, X\overline{X} \mid \mathbf{callback}\, X : T \mid D_B; D_B \mid \epsilon \\
B ::= & \ \mathbf{param}\, \overline{X\overline{X}}; D_B
\end{aligned}
$$

Figure 3: A second formal syntax for module types

2 Modules

Like with module types, we give a surface syntax for modules (figure 5), two formal syntaxes (figure 6 and figure 7) and the translation rules between them (figure 8). The `param` keyword allows modules to depend on types (the first kind of declaration) and to specify the arguments of module types (the second one).

[1] The translation to the core language will show how `param` declarations differ from the others.

$$\frac{D \hookrightarrow_B \mathbf{param}\,\overline{X\overline{Y}}; D_M \qquad D' \hookrightarrow_B \mathbf{param}\,\overline{X'\overline{Y'}}; D'_M \qquad X \cap X' = \emptyset}{D; D' \hookrightarrow_B \mathbf{param}\,\overline{X\overline{Y}}, \overline{X'\overline{Y'}}; (D_M; D'_M)}$$

$$\frac{}{\mathbf{param}\,X\overline{Y} \hookrightarrow_B \mathbf{param}\,X\overline{Y}; \epsilon} \qquad \frac{D \neq \mathbf{param}\,X\overline{Y} \qquad D \neq D^S_B; D^S_B}{D \hookrightarrow_B \mathbf{param}\,\emptyset; D}$$

Figure 4: Translation rules between the two formal syntaxes for module types

```
defmodule M do
  [ $param X(X1, ..., Xn)
  | $param X(X1, ..., Xn)=t
  | $type X(X1, ..., Xn)=t
  | $opaque X(X1, ..., Xn)=t
  | @behaviour B
  | def f(X1 : t1, ..., Xn : tn) : t = e
  | defp f(X1 : t1, ..., Xn : tn) : t = e
  ]*
end
```

Figure 5: Surface syntax proposal for modules

In the second syntax the parameters are put at the beginning, followed by module types declarations and function and type declarations. Module types are declared by a partial function from identifiers to records. For the translation rules we define the union of two such functions b and b' by the following equation:

$$(b \cup b')(X) = \begin{cases} b(X) & \text{if } X \notin \mathrm{dom}(b'), \\ b'(X) & \text{if } X \notin \mathrm{dom}(b), \\ \{\overline{X = T}; \overline{X' = T'}\} & \text{if } b(X) = \{\overline{X = T}\} \wedge b'(X) = \{\overline{X' = T'}\}, \end{cases}$$

and is undefined otherwise. When using this kind of union, we make sure that the tags $\overline{X_i}$ and $\overline{X'_i}$ are disjoint. The inductive rules of figure 8 define jugements of the form $\Gamma, \mathcal{B} \vdash D^S_M \hookrightarrow_M D_M$, meaning D^S_M can be translated to M knowing it should implement the module types in the set $\mathcal{B}$ and the possible module types, with their list of parameters, are given by Γ.

3 Programs

We can, now, define a program as a sequence of module types and module declarations, as done in figure 9. The translation rules are given in figure 10. They use the two preceding sets of rules while maintaining a context of module types.

$$
\begin{aligned}
D_M^S ::= & \ \mathbf{param}\, X\overline{X} \mid \ \mathbf{param}\, X\overline{X} = T \mid (\,\mathbf{type} \mid \mathbf{opaque}\,)X\overline{X} = T \\
& \mid \mathbf{behaviour}\, X \mid \ \mathbf{def(p)}\, X(\overline{X : T}) : T = E \mid D_M^S; D_M^S \mid \epsilon \\
M^S ::= & \ \mathbf{defmodule}\, X\, \mathbf{do}\, D_M^S\, \mathbf{end}
\end{aligned}
$$

Figure 6: A first formal syntax for modules

$$
\begin{aligned}
D_M ::= & \ (\,\mathbf{type} \mid \mathbf{opaque}\,)X\overline{X} = T \mid \ \mathbf{def(p)}\, X(\overline{X : T}) : T = E \mid D_M; D_M \mid \epsilon \\
M ::= & \ \mathbf{param}\, \overline{X\overline{X}};\ \mathbf{behaviour}\, X \mapsto \{\overline{X = T}\}; D_M
\end{aligned}
$$

Figure 7: A second formal syntax for modules

4 Core language

We translate the desugared syntaxes of the previous sections to 1ML's core language without `include` nor `where` and augmented with intersection types. The syntax of this language is given in figure 11.

Programs of the preceding section are translated to sequence of bindings. The module types are translated to functions from types to types, and modules are mapped to functions from types to records. The rules are given in figure 12.

We add sealings to `defp` declarations in order to have their return type is checked. Thus `m.s` can't be replaced by `list(int)` in the tests of the `stack_sharing.ex` example.

$$\frac{X \in \mathrm{dom}(\Gamma)}{\Gamma, \mathcal{B} \vdash \mathbf{behaviour}\, X \hookrightarrow_M \mathbf{param}\, \emptyset;\ \mathbf{behaviour}\, X \mapsto \{\};\ \epsilon}$$

$$\Gamma, \mathcal{B} \vdash \mathbf{param}\, X \hookrightarrow_M \mathbf{param}\, X;\ \mathbf{behaviour}\, \emptyset;\ \epsilon$$

$$\frac{\begin{array}{c}\Gamma, \mathcal{B} \vdash D \hookrightarrow_M \mathbf{param}\, P;\ \mathbf{behaviour}\, b;\ D_M \\ \Gamma, \mathcal{B} \cup \mathrm{dom}(b) \vdash D' \hookrightarrow_M \mathbf{param}\, P';\ \mathbf{behaviour}\, b';\ D'_M \\ P \cap P' = \emptyset \qquad \forall X \in \mathrm{dom}(b) \cap \mathrm{dom}(b').\mathrm{tags}(b(X)) \cap \mathrm{tags}(b'(X)) = \emptyset\end{array}}{\mathcal{B} \vdash D; D' \hookrightarrow_M \mathbf{param}\, PP';\ \mathbf{behaviour}\, b \cup b';\ (D_M; D'_M)}$$

$$\frac{\exists! B \in \mathcal{B}.X \in \Gamma(B)}{\Gamma, \mathcal{B} \vdash \mathbf{param}\, X = t \hookrightarrow_M \mathbf{param}\, \emptyset;\ \mathbf{behaviour}\, B \mapsto \{X = t\};\ \mathbf{type}\, X = t}$$

$$\frac{D \neq \mathbf{param}\, X \qquad D \neq \mathbf{param}\, X = t \qquad D \neq D^S_M; D^S_M}{\Gamma, \mathcal{B} \vdash D \hookrightarrow_M \mathbf{param}\, \emptyset;\ \mathbf{behaviour}\, \emptyset;\ D}$$

Figure 8: Translation rules between the two formal syntaxes for modules

$$\begin{aligned} P^S &::= B^S; P^S \mid M^S; P^S \mid \epsilon \\ P &::= X = B \mid X = M \mid P; P \mid \epsilon \end{aligned}$$

Figure 9: Two formal syntaxes for programs

$$\frac{D^S_B \hookrightarrow_B \mathbf{param}\, \overline{X}; D \qquad \Gamma \cup (B \mapsto \overline{X}) \vdash P^S \hookrightarrow_P P}{\Gamma \vdash \mathbf{defmoduletype}\, B\, \mathbf{do}\, D^S_B\, \mathbf{end}\,; P^S \hookrightarrow_P (B = \mathbf{param}\, \overline{X}; D); P}$$

$$\frac{\Gamma, \emptyset \vdash D^S_M \hookrightarrow_M M \qquad \Gamma \vdash P^S \hookrightarrow_P P}{\Gamma \vdash \mathbf{defmodule}\, X\, \mathbf{do}\, D^S_M\, \mathbf{end}\,; P^S \hookrightarrow_P X = M; P}$$

Figure 10: Translation rules between the two formal syntaxes for programs

$$\begin{aligned} T &::= E \mid \mathbf{bool()} \mid \{D\} \mid (X : T) \Rightarrow T \mid (X : T) \to T \\ &\quad \mid \mathbf{type} \mid = E \mid T \cap T \mid : X \\ D &::= X : T \mid D; D \mid \epsilon \\ E &::= X \mid \mathbf{true} \mid \mathbf{false} \mid \mathbf{if}\, E\, \mathbf{then}\, E\, \mathbf{else}\, E \mid \{N\} \mid E.X \\ &\quad \mid \mathbf{fun}\, (X : T) \Rightarrow E \mid XX \mid \mathbf{type}\, T \mid E :> T \mid : X \\ N &::= X = E \mid N; N \mid \epsilon \end{aligned}$$

Figure 11: Syntax of the core language

Behaviours

$$\frac{}{\textbf{type}\,X\overline{Y} = T \hookrightarrow_1^B X : \overline{(Y : \textbf{type})} \Rightarrow [= \textbf{type}\,T]}$$

$$\frac{}{\textbf{opaque}\,X\overline{Y} \hookrightarrow_1^B X : \overline{(Y : \textbf{type})} \Rightarrow \textbf{type}}$$

$$\frac{}{\textbf{callback}\,X : T \hookrightarrow_1^B X : T} \qquad \frac{}{\epsilon \hookrightarrow_1^B \epsilon} \qquad \frac{D_B \hookrightarrow_1^B D \qquad D'_B \hookrightarrow_1^B D'}{D_B; D'_B \hookrightarrow_1^B D; D'}$$

Figure 12: Translation rules to the core language

Modules

$$\frac{}{\textbf{def}\,X(\overline{Y : T}) : T' = E \hookrightarrow_1^M \left(X = \overline{\textbf{fun}\,Y : T \Rightarrow E}\right) : \left(X : \overline{(Y : T) \to T'}\right)}$$

$$\frac{}{\textbf{defp}\,X(\overline{Y : T}) : T' = E \hookrightarrow_1^M \left(X = \overline{\textbf{fun}\,Y : T \Rightarrow E} :> \overline{(Y : T) \to T'}\right) : \epsilon}$$

$$\frac{}{\begin{aligned}\textbf{type}\,X\overline{Y} = T \hookrightarrow_1^M X = \overline{\textbf{fun}\,Y : \textbf{type} \Rightarrow \textbf{type}\,T} :\\ X : \overline{(Y : \textbf{type})} \Rightarrow [= \textbf{type}\,T]\end{aligned}}$$

$$\frac{}{\begin{aligned}\textbf{opaque}\,X\overline{Y} = T \hookrightarrow_1^M X = \overline{\textbf{fun}\,Y : \textbf{type} \Rightarrow \textbf{type}\,T} :\\ X : \overline{(Y : \textbf{type})} \Rightarrow \textbf{type}\end{aligned}}$$

$$\frac{}{\epsilon \hookrightarrow_1^M \epsilon : \epsilon} \qquad \frac{D_M \hookrightarrow_1^M N : D \qquad D'_M \hookrightarrow_1^M N' : D'}{D_B; D'_B \hookrightarrow_1^M B; B'}$$

Figure 12: Translation rules to the core language

Programs

$$\frac{}{\overline{\epsilon \hookrightarrow_1 \epsilon}}$$

$$\frac{D_B \hookrightarrow_1^B D}{B = \mathbf{param}\,\overline{X\overline{Y}}; D_B \hookrightarrow_1^P B = \mathbf{fun}\,(p : \{\overline{X : \overline{(Y : \mathbf{type})} \Rightarrow \mathbf{type}}\}) \Rightarrow \mathbf{type}\,\{\overline{X : [= p.X]}; D\}}$$

$$\frac{\begin{array}{c} P = (M = \mathbf{param}\,\overline{X};\ \mathbf{behaviour}\,\overline{B \mapsto \{\overline{X_B = T_B}\}}; D_M) \\ D_M \hookrightarrow_1^M N : D \qquad \overline{T'_B = T_B[\overline{X \leftarrow p.X}]} \end{array}}{\begin{array}{l} P \hookrightarrow_1^P M = \mathbf{fun}\,(p : \{\overline{X : \mathbf{type}}\}) \Rightarrow \{\overline{X = p.X}; N\} :> \\ \qquad (p : \{\overline{X : \mathbf{type}}\}) \Rightarrow \{\overline{X : [= p.X]}; D\} \cap \bigcap \overline{B(\{\overline{X_B = \mathbf{type}\,T'_B}\})} \end{array}}$$

Figure 12: Translation rules to the core language