## 1 Behaviours

To specify a behaviour, 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 behaviour definition.

```
defbehaviour M do
  [ $param X
  | $opaque X X1 ... Xn
  | $type X X1 ... Xn = t
  | callback X : t
  ]*
end
```

Figure 1: Surface syntax proposal for behaviours

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

Figure 2: A first formal syntax for behaviours

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

Figure 3: A second formal syntax for behaviours

## 2 Modules

Like with behaviours, 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 behaviours (the second one).

In the second syntax the parameters are put at the beginning, followed by behaviour declarations and function and type declarations. Behaviours are

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

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

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

Figure 4: Translation rules between the two formal syntaxes for behaviours

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

Figure 5: Surface syntax proposal for behaviours

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:

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

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 behaviours in the set $\mathcal{B}$ and the possible behaviours, with their list of parameters, are given by $\Gamma$.

## 3 Programs

We can, now, define a program as a sequence of behaviour 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 behaviours.

## 4 Core language

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

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

Figure 6: A first formal syntax for modules

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

Figure 7: A second formal syntax for modules

Programs of the preceding section are translated to sequence of bindings. The behaviours 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.

## References

[1] Andreas Rossberg. “1ML–Core and modules united”. In: *Journal of Functional Programming* 28 (2018), e22.

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

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

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

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

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

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

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

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

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

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

Figure 11: Syntax of the core language

***Behaviours***

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

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

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

***Modules***

$$\frac{}{\textbf{def}\, X(\overline{Y : T}) : T' = E \hookrightarrow_1^M \big(X = \overline{\textbf{fun}\, Y : T \Rightarrow} E\big) : \Big(X : \overline{(Y : T) \rightarrow} T'\Big)}$$

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

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

***Programs***

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

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

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

Figure 12: Translation rules to the core language