1 Syntax

We first introduce the syntax of a MiniML program. A program can consist of a set of modules, consisting of an interface and module body and uniquely identified with a module *name*. The interface and body are represented as a set of declarations Δ and of definitions d respectively. The program is concluded by a single naked expression, functioning as the main entry point of the program.

$$
\begin{aligned}
\text{Program} &::= \overline{Mod};\ e \\
\text{Module } Mod &::= \{\Delta, \overline{d}\}^{name} \\
\\
\text{Declaration } \Delta\ &::= \emptyset \\
&\mid (id : \tau), \Delta \\
\\
\text{Definition } d\ &::= \emptyset \\
&\mid (id = e : \tau), d \\
\\
\text{Expression } e\ &::= num\ n \mid false \mid true \\
&\mid id \\
&\mid name.id \\
&\mid this.id \\
&\mid e_1 e_2 \\
&\mid (e_1, e_2) \\
&\mid \lambda(p : \tau)\ .\ e \\
&\mid let\ p\ =\ e_1\ in\ e_2 \\
&\mid letrec\ p\ =\ e_1\ in\ e_2 \\
&\mid if(e_1)\ then\ e_2\ else\ e_3 \\
\\
\text{Pattern } p\ &::= id \\
&\mid (p, p) \\
\\
\text{Type } \tau\ &::= nat \\
&\mid bool \\
&\mid \tau_1 \to \tau_2 \\
&\mid \tau_1 \times \tau_2 \\
&\mid \alpha
\end{aligned}
$$

2 Type System

We now introduce the Type System for MiniML. First, we introduce the concept of a type-scheme. The concept of a type-scheme, sometimes called

polytype, introduces polymorphism by making use of the type variable α in the definition of τ, and quantifying it with the $\forall$ quantifier. This allows any concrete types τ to 'match' to the type variable.
This concept of a type-scheme will later be used to provide let-polymorphism. Note that the definition of a Type-Scheme assures that the resulting type-tcheme is in *prenex normal form*, i.e. a string of quantifiers concluded by a quantifier-free ending.

$$\begin{aligned} \text{Type-Scheme } \sigma \; &::= \tau \\ &\mid \forall \alpha . \sigma \end{aligned}$$

We also introduce the notion of a context. The context simply is a **set** allowing for lookups. It contains type assumptions, with representation $x : \sigma$, meaning x has type σ, as well as mappings from module name to module definition

$$\begin{aligned} \text{Context } \Gamma &::= \emptyset \\ &\mid \Gamma, (x : \sigma) \\ &\mid \Gamma, (name : \{\Delta, d\}^{name} \end{aligned}$$

The following four relations are the typing judgements. They convey the meaning that an expression or other part of the syntax is well-typed in the context Γ. The typing of a module body and it's definitions generates a new typing context for the module. In this resulting context, the declarations must be well-typed.

$$\begin{aligned} \text{Typing } &::= \Gamma \vdash e : \sigma \\[1em] \text{ModuleTyping } &::= \Gamma \vdash Mod \\ \text{DefinitionTyping } &::= \Gamma \vdash d \rightarrow \Gamma' \\[1em] \text{DeclarationTyping } &::= \Gamma \vdash \Delta \end{aligned}$$

2.1 Rules

$$\Gamma \vdash true : bool \tag{T-True}$$

$$\Gamma \vdash false : bool \tag{T-False}$$

$$\Gamma \vdash num\ n : nat \tag{T-Num}$$

$$\frac{\sigma \geq \tau \qquad id : \sigma \in \Gamma}{\Gamma \vdash id : \tau} \tag{T-Mono}$$

$$\frac{\Gamma \vdash e_1 : \tau_2 \to \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1 e_2 : \tau_1} \tag{T-App}$$

$$id : \sigma \to \emptyset, (id : \sigma) \tag{T-BuildContext1}$$

$$\frac{p_1 : \sigma_1 \to \Gamma_1 \qquad p_2 : \sigma_2 \to \Gamma_2}{(p_1, p_2) : \sigma_1 \times \sigma_2 \to \Gamma_1 \cup \Gamma_2} \tag{T-BuildContext2}$$

$$\frac{p : \tau_2 \to \Gamma_2 \qquad \Gamma_2 \cup \Gamma_1 \vdash e : \tau_1}{\Gamma_1 \vdash \lambda(p : \tau).e : \tau_2 \to \tau_1} \tag{T-Fun}$$

$$\frac{\Gamma \vdash e_1 : bool \qquad \Gamma \vdash e_2 : \tau \qquad \Gamma \vdash e_3 : \tau}{\Gamma \vdash if\ e_1\ then\ e_2\ else\ e_3 : \tau} \tag{T-IfThenElse}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2} \tag{T-Pair}$$

$$\frac{\Gamma \vdash e_2 : \tau_2 \quad \sigma = gen(\Gamma, \tau) \quad p : \sigma \to \Gamma_2 \quad \Gamma \cup \Gamma_2 \vdash e_1 : \tau}{\Gamma \vdash let\ p\ =\ e_2\ in\ e_1 : \tau} \tag{T-Let}$$

$$\frac{\Gamma \vdash let\ p\ =\ fix\ (\lambda p.e_2)\ in\ e_1 : \tau}{\Gamma \vdash letrec\ p\ =\ e_2\ in\ e_1 : \tau} \tag{T-Letrec}$$

$$\frac{\Gamma \vdash e : \tau \to \tau}{\Gamma \vdash fix(e) : \tau} \tag{T-Fix}$$

$$\frac{\sigma \geq \tau \qquad \mathit{this.id} : \sigma \in \Gamma}{\Gamma \vdash \mathit{this.id} : \tau} \qquad \text{(T-ModVarThis)}$$

$$\frac{\Gamma \vdash \{\Delta, d\}^{name} \qquad id : \tau \in \Delta}{\Gamma \vdash name.id : \tau} \qquad \text{(T-ModVarOther)}$$

$$\frac{\emptyset \vdash d \to \Gamma' \qquad \Gamma' \vdash \Delta}{\Gamma \vdash \{\Delta, d\}^{name}} \qquad \text{(T-Module)}$$

$$\frac{(x : \tau) \in \Gamma \qquad \Gamma \vdash \Delta}{\Gamma \vdash (x : \tau), \Delta} \qquad \text{(T-ModInterface)}$$

$$\frac{(x : \tau), \Gamma \vdash d \to \Gamma' \qquad \Gamma \vdash e : \tau}{\Gamma \vdash (x = e : \tau), d \to (x : \tau), \Gamma'} \qquad \text{(T-ModBody)}$$

$$\frac{Wellformednessof \Gamma}{\Gamma \vdash \emptyset} \qquad \text{(T-EmptySet)}$$

3 Operational semantics

$$\begin{aligned} \text{Value } v ::= & \; num\ n \mid true \mid false \\ & |(v, v) \\ & |\lambda p.e \end{aligned}$$

$$\begin{aligned} \text{Environment } E \; ::= & \; \cdot \\ & \mid E : (id = v) \\ & \mid E : (\mathit{this.id} = v) \end{aligned}$$

$$\text{Evaluation } ::= e \to e'$$

3.1 Rules

$$if\ true\ then\ e_1\ else\ e_2 \rightarrow e_1 \qquad \text{(E-IfTrue)}$$

$$if\ false\ then\ e_1\ else\ e_2 \rightarrow e_2 \qquad \text{(E-IfFalse)}$$

$$\frac{e_1 \rightarrow e_1'}{if\ e_1\ then\ e_2\ else\ e_3 \rightarrow if\ e_1'\ then\ e_2\ else\ e_3} \qquad \text{(E-IfThenElse)}$$

$$\frac{e_1 \rightarrow e_1'}{(e_1, e_2) \rightarrow (e_1', e_2)} \qquad \text{(E-PairLeft)}$$

$$\frac{e_2 \rightarrow e_2'}{(e_1, e_2) \rightarrow (e_1, e_2')} \qquad \text{(E-PairRight)}$$

$$\frac{e_1 \rightarrow e_1'}{let\ p\ =\ e_1\ in\ e_2 \rightarrow let\ x\ =\ e_1'\ in\ e_2} \qquad \text{(E-Let)}$$

$$let\ x\ =\ v\ in\ e \rightarrow [x \mapsto v]e \qquad \text{(E-LetV)}$$

$$letrec\ p =\ e_1\ in\ e_2 \rightarrow let\ p\ = fix(\lambda p.e_1)\ in\ e_2 \qquad \text{(E-LetRec)}$$

$$\frac{e \rightarrow e'}{fix(e) \rightarrow fix(e')} \qquad \text{(E-Fix)}$$

$$fix(\lambda(p.e)) \rightarrow [p \mapsto (fix(\lambda(p.e))]e \qquad \text{(E-FixRec)}$$

$$let\ (p_1, p_2)\ =\ (e_1, e_2)\ in\ e_3 \rightarrow let\ p_1\ =\ e_1\ in\ (let\ p_2\ =\ e_2\ in\ e_3) \qquad \text{(E-PatternMatch)}$$

$$\frac{e_1 \rightarrow e_1'}{e_1 e_2 \rightarrow e_1' e_2} \qquad \text{(E-App1)}$$

$$\frac{e_2 \rightarrow e_2'}{v\ e_2 \rightarrow v\ e_2'} \qquad \text{(E-App2)}$$

$$(\lambda x.e)\ v \rightarrow [x \mapsto v]e \qquad \text{(E-Lambda)}$$

$$(\lambda(p_1, p_2).e_3)\ (e_1, e_2) \rightarrow (\lambda p_1.(\lambda p_2.e_3)\ e_2)\ e_1 \qquad \text{(E-MatchLambda)}$$

$$\frac{\{\Delta, d\}^M \qquad (x = e' : \tau) \in d \qquad e = [this.y \mapsto M.y]e' \;\forall(this.y \in e')}{M.x \to e} \quad \text{(E-ModVar)}$$