# 1 Virtual Machine Abstract Syntax

| | | | |
|---|---|---|---|
| $e$ | ::= | $x$ | *expressions* |
| | \| | $\texttt{new}_s\ \tau\ \{x \Rightarrow \overline{d}\}$ | |
| | \| | $e.m(e)$ | |
| | \| | $e.f$ | |
| | \| | $e.f = e$ | |
| | \| | $\texttt{let}\ x = e\ \texttt{in}\ \ e$ | |
| | \| | $e : \tau$ | |
| | \| | $\mathscr{L}$ | |
| | \| | $e.\texttt{match}\ \overline{x : p.L \Rightarrow e}\texttt{else}\ e$ | |
| $\mathscr{L}$ | ::= | *string* | *literals* |
| | \| | *integer* | |
| | \| | *rational* | |
| $v$ | ::= | $x$ | *values* |
| $d$ | ::= | $\texttt{val}\ f : \tau = v$ | *declarations* |
| | \| | $\texttt{var}\ f : \tau = v$ | |
| | \| | $\texttt{def}\ m(\overline{x : \tau}) : \tau = e$ | |
| | \| | $\texttt{type}\ L = T$ | |
| | \| | $\texttt{delegate}\ \tau\ \texttt{to}\ f$ | |
| $T$ | ::= | $c$ | *type desc.* |
| | \| | $\texttt{extag}\ c$ | |
| | \| | $\texttt{datatag}\ \overline{p.L}\ c$ | |
| $c$ | ::= | $\tau$ | *case desc.* |
| | \| | $\texttt{case of}\ p.L\ \tau$ | |
| $\tau$ | ::= | $\tau\ \{\texttt{x} \Rightarrow \overline{\sigma}\}_s$ | *type* |
| | \| | $p.L$ | |
| | \| | $\top$ | |
| $p$ | ::= | $x$ | *paths* |
| | \| | $p.f$ | |
| $s$ | ::= | $\texttt{stateful} \mid \texttt{pure}$ | |
| $\sigma$ | ::= | $\texttt{val}\ f : \tau$ | *decl type* |
| | \| | $\texttt{var}\ f : \tau$ | |
| | \| | $\texttt{def}\ m : \Pi\overline{x{:}\tau}.\tau$ | |
| | \| | $\texttt{type}\ L = T$ | |
| | \| | $\texttt{type}_s\ L$ | |

Notes on semantics:

- Errors (e.g. nothing to do in the default case of a match) can be handled by calling a library function taking a string argument that reports an error (with the given string) and halts (optionally, in the debugger)
- As the type syntax for methods suggests, method types are dependent, so that the result type can depend on the argument type, and the types of later arguments can depend on earlier arguments. We need this to encode type parameters. Interestingly, first-class traits could encode this (treat a trait as a method, merge it with something that specifies the type) as could the ability to somehow create an object leaving the type abstract and then set the type (only once!)

Notes on encodings:

- encoding type parameters - see whiteboard picture from 6/2/2015
- FFI interface spec - see whiteboard picture from 5/27/2015
- encoding of tags to lower-level - see whiteboard picture from 6/2/2015

## 2 Virtual Machine Typing Rules

$\boxed{\Gamma \vdash e : \tau}$

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \text{ (T-Var)}$$

$$\frac{\Gamma,\ x : \tau \vdash_s \overline{d} : \overline{\sigma} \quad \{x \Rightarrow \overline{\sigma}\}_s <: \mathit{typepart}(\tau)}{\Gamma \vdash \texttt{new}_{\ s}\ \tau\ \{x \Rightarrow \overline{d}\} : \tau} \text{ (T-New)}$$

$$\frac{\Gamma \vdash e_1 : \{\overline{\sigma_e}\}_s \quad \texttt{def}\ \ m : \tau_2 \to \tau_1 \in \overline{\sigma_e} \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1.m(e_2) : \tau_1} \text{ (T-Meth)}$$

$$\frac{\Gamma \vdash e : \{\overline{\sigma}\}_s \quad \texttt{var}\ \ f : \tau \in \overline{\sigma}}{\Gamma \vdash e.f : \tau} \text{ (T-Field)} \qquad \frac{\Gamma \vdash e_1 : \{\overline{\sigma}\}_s \quad \texttt{var}\ \ f : \tau \in \overline{\sigma} \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1.f = e_2 : \tau} \text{ (T-Assign)}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{bind}\ x = e_1\ \texttt{in}\ e_2 : \tau_2} \text{ (T-Bind)} \qquad \frac{l : \tau \in \Sigma}{\Gamma \vdash l : \tau} \text{ (T-Loc)}$$

$$\frac{\Gamma \vdash l_1 : \{\overline{\sigma_e}\}_s \quad \texttt{def}\ \ m : \tau_2 \to \tau_1 \in \overline{\sigma_e} \quad \Gamma \vdash l_2 : \tau_2 \quad \Gamma \vdash e : \tau_1}{\Gamma \vdash l_1.m(l_2) \rhd e : \tau_1} \text{ (T-StackFrames)}$$

$$\frac{\Gamma \vdash e : \tau_1 \quad \tau_1 <: \tau_2}{\Gamma \vdash e : \tau_2} \text{ (T-Sub)}$$

$\boxed{\Gamma \vdash_s \overline{d} : \overline{\sigma}}$

$$\frac{\forall j,\ d_j \in \overline{d},\ \sigma_{ij} \in \overline{\sigma},\ \Gamma \vdash_s d_j : \sigma_{ij}}{\Gamma \vdash_s \overline{d}\ :\ \overline{\sigma}} \text{ (T-Decls)}$$

$\boxed{\Gamma \vdash_s d : \sigma}$

$$\frac{\Gamma_{\mathit{stateful}} = \{x : \{\overline{\sigma}\}_{\texttt{stateful}} \mid x : \{\overline{\sigma}\}_{\texttt{stateful}} \in \Gamma\} \quad \Gamma_{\mathit{pure}} = \Gamma \setminus \Gamma_{\mathit{stateful}} \quad \Gamma_{\mathit{pure}},\ y : \tau_1 \vdash e : \tau_2}{\Gamma \vdash_{\texttt{pure}} \texttt{def}\ m(y : \tau_1) : \tau_2 = e\ :\ \texttt{def}\ m : \tau_1 \to \tau_2} \text{ (DT-DefPure)}$$

$$\frac{\Gamma,\ x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash_{\texttt{stateful}} \texttt{def}\ m(x : \tau_1) : \tau_2 = e\ :\ \texttt{def}\ m : \tau_1 \to \tau_2} \text{ (DT-DefStateful)}$$

$$\frac{\Gamma \vdash x : \tau}{\Gamma \vdash_{\texttt{stateful}} \texttt{var}\ f : \tau = x\ :\ \texttt{var}\ f : \tau} \text{ (DT-Varx)} \qquad \frac{\Gamma \vdash l : \tau}{\Gamma \vdash_{\texttt{stateful}} \texttt{var}\ f : \tau = l\ :\ \texttt{var}\ f : \tau} \text{ (DT-Varl)}$$

$\boxed{\{\tau <: \tau}$

$$\frac{\tau \ \texttt{wf}}{\tau <: \tau} \ \text{(S-Refl)} \qquad \frac{\tau \ \texttt{wf}}{\tau <: \top} \ \text{(S-Top)}$$

$$\frac{\tau \ \{x \Rightarrow \overline{\sigma}\}_{\texttt{pure}} \ \ \texttt{wf}}{\tau \ \{x \Rightarrow \overline{\sigma}\}_{\texttt{pure}} \ <: \tau \ \{x \Rightarrow \overline{\sigma}\}_{\texttt{stateful}}} \ \text{(S-State)}$$

$$\frac{\sigma_j^{j\in 1..n} \textit{ is a permutation of } \sigma_j'^{j\in 1..n} \quad \tau \ \{x \Rightarrow \sigma_j^{j\in 1..n}\} \ \texttt{wf}}{\tau \ \{x \Rightarrow \sigma_j^{j\in 1..n}\}_s <: \tau \ \{x \Rightarrow \sigma_j'^{j\in 1..n}\}_s} \ \text{(S-Perm)}$$

$$\frac{\tau \ \{x \Rightarrow \sigma_j^{j\in 1..n+k}\}_s \ \texttt{wf} \quad \tau \ \{x \Rightarrow \sigma_j^{j\in 1..n}\}_s \ \texttt{wf}}{\tau \ \{x \Rightarrow \sigma_j^{j\in 1..n+k}\}_s <: \tau \ \{x \Rightarrow \sigma_j^{j\in 1..n}\}_s} \ \text{(S-Width)} \qquad \frac{\forall j, \ \sigma_j <: \sigma_j'}{\{\sigma_j^{j\in 1..n}\}_s <: \{\sigma_j^{j\in 1..n}\}_s} \ \text{(S-Depth)}$$

$\boxed{\sigma <: \sigma'}$

$$\frac{}{\sigma <: \sigma} \ \text{(S-Refl2)} \qquad \frac{\sigma_1 <: \sigma_2 \quad \sigma_2 <: \sigma_3}{\sigma_1 <: \sigma_3} \ \text{(S-Trans2)}$$

$$\frac{\tau_1' <: \tau_1 \quad \tau_2 <: \tau_2'}{\texttt{def} \ m : \tau_1 \to \tau_2 <: \texttt{def} \ m : \tau_1' \to \tau_2'} \ \text{(S-Def)} \qquad \frac{\tau <: \tau'}{\texttt{val} \ f : \tau <: \texttt{val} \ f : \tau'} \ \text{(S-Val)}$$

$$\frac{T \textit{ stateful} \Rightarrow (s = \texttt{stateful})}{\texttt{type} \ L = T <: \texttt{type}_s \ L} \ \text{(S-AbsType)} \qquad \frac{\tau <: \tau'}{\texttt{type}_{pure} \ L <: \texttt{type}_{stateful} \ L} \ \text{(S-Pure)}$$

$\boxed{\mathit{typepart}(\Gamma, T)}$

$$\frac{\mathit{typepart}(\Gamma, \tau) = \{\texttt{x} \Rightarrow \overline{\sigma'}\}}{\mathit{typepart}(\Gamma, \tau \ \{\texttt{x} \Rightarrow \overline{\sigma}\}_s) = \{\texttt{x} \Rightarrow \overline{\sigma'} \leftarrow \overline{\sigma}\}_s} \ \text{(Typepart-Refinement)}$$

$$\frac{\Gamma \vdash p : \tau \quad \Gamma \vdash \tau <: \{x \Rightarrow \texttt{type} \ L = T, \overline{\sigma}\}_s}{\mathit{typepart}(\Gamma, p.L) = \mathit{typepart}(\Gamma, [p/x]T)} \ \text{(Typepart-Member)}$$

$$\mathit{typepart}(\Gamma, \top) = \top$$

$$\mathit{typepart}(\Gamma, \texttt{extag} \ \ c) = \mathit{typepart}(\Gamma, c)$$

$$\mathit{typepart}(\Gamma, \texttt{datatag} \ \ \overline{p.L} \ c) = \mathit{typepart}(\Gamma, c)$$

$$\mathit{typepart}(\Gamma, \texttt{case of} \ \ p.L \ \tau) = \mathit{typepart}(\Gamma, \tau)$$

Note: *typepart* is not defined if the members of $T$ are not visible.

Note: need to define $T$ *stateful* judgment. Intuitively this just looks up the definition of $T$ (probably using $\Gamma$, not shown) to see if it is stateful.

Note: need to define $\overline{\sigma'} \leftarrow \overline{\sigma}$ judgment. This just takes the union of the declaration types, using the declarations on the right to override declarations with the same name on the left.

Note: need to define well-formedness, and specify when it must be checked. The refinement of a type must define members that are subtypes of corresponding members in the type being refined.