1 Definitions

1.1 Identifiers

	ModuleName		
	StructName		
f	$\in$	FieldName	
x	$\in$	VarName	

1.2 Types and Kinds

		Kind	$=$	**resource**|**unrestredted**
		ModuleId	$=$	$\text{AccountAddr} \times \text{ModuleName}$
s	$\in$	StructID	$=$	$\text{ModuleID} \times \text{StructName}$
		StructType	$=$	StructID
		PrimitiveType	$=$	$\text{AccountAddr} \cup \text{Bool} \cup \text{Unsigned64} \cup \text{Bytes}$
a	$\in$	AccountAddr		
b	$\in$	Bool		
n	$\in$	Unsigned64		
$\vec{b}$	$\in$	Bytes		
τ	$\in$	NonRefType	$=$	$\text{StructType} \times \text{Primitive}$
		Type	$=$	$\tau \mid \&\text{mut}\,\tau \mid \&\tau$

1.3 Values

rv	$\in$	Resource	$=$	$\text{StructID} \times \text{Tag} \times \text{Value}^*$
		Struct	$=$	$\text{StructID} \times \text{UnrestrictedValue}^*$
		PrimitiveValue	$=$	$a \mid b \mid n \mid \vec{b}$
u	$\in$	UnrestrictedValue	$=$	$\text{Struct} \cup \text{PrimitiveValue}$
v	$\in$	Value	$=$	$\text{Resource} \cup \text{UnrestrictedValue}$
r	$\in$	Reference	$=$	$\text{Root} \times \text{Path} \times \text{Qualifier}$
		Root	$=$	$x \mid s$
p	$\in$	Path	$=$	f^*
q	$\in$	Qualifier	$=$	**mut** | **immut**
		RuntimeValue	$=$	$v \mid r$

1.4 Memory

M	$\in$	Memory	$=$	$\text{LocalMemory} \times \text{GlobalMemory}$
		LocalMemory	$=$	$\text{Var} \rightharpoonup \text{RuntimeVal}$
		GlobalMemory	$=$	$\text{AccountAddr} \rightharpoonup \text{Account}$
		Account	$=$	$\text{ModuleName} \rightharpoonup \text{Module}$
		Module	$=$	$\text{StructName} \rightharpoonup \text{StructSig}$
		StructSig	$=$	$\text{Kind} \times (\text{FieldName} \times \text{NonRefType})^*$

We write $M(l)$ to be the value stored at l in memory M, where l could be a local variable or a reference. We write $M[l \mapsto v]$ to be the memory with l updated to have value v, and otherwise identical with M. We use $M \setminus x$ to denote the memory with x removed, and otherwise identical with M.

1.5 Local Evaluation State

σ	$\in$	LocalState	$=$	$\langle M, S \rangle$
S	$\in$	LocalStack	$=$	RuntimeValue^*
l	$\in$	Location	$=$	$x.p \mid s.p \mid n.p$

We write $\sigma(l) = v$ if the (possibly runtime) value stored at l is v.

Definition 1. *A local state σ is tag-consistent if $\sigma(l_1) = rv_1$, $\sigma(l_2) = rv_2$ and $\text{tag}(rv_1) = \text{tag}(rv_2)$ implies that $l_1 = l_2$. That is, each resource value hold in σ has a unique tag.*

2 Judgements

Judgement	Meaning
$r\,q$	reference r has mutability q
$M \rhd t\,\mathbf{Fresh}$	tag t is fresh in M
$M \rhd \kappa\,\tau\{f_1\colon \tau_1, \ldots, f_n\colon \tau_n\}$	In memory M struct type τ has kind κ, field name f_i and field types τ_i
$\langle M, S\rangle \xrightarrow{i} \langle M', S'\rangle$	state $\langle M, S\rangle$ steps to $\langle M, S\rangle$ after executing instruction i

Table 1.

3 Operational Semantics

3.1 Local Instructions

$$\frac{M(x) = v \vee M(x) = r}{\langle M, S\rangle \xrightarrow{\mathrm{MvLoc}\langle x\rangle} \langle M \setminus x, M(x) :: S\rangle}\ \mathbf{MvLoc}$$

$$\frac{M(x) = u \vee M(x) = r}{\langle M, S\rangle \xrightarrow{\mathrm{CpLoc}\langle x\rangle} \langle M, M(x) :: S\rangle}\ \mathbf{CpLoc}$$

$$\frac{s = u \vee s = r}{\langle M, s :: S\rangle \xrightarrow{\mathrm{StLoc}\langle x\rangle} \langle M[x \mapsto s], S\rangle}\ \mathbf{StLoc}$$

$$\frac{M(x) = v}{\langle M, S\rangle \xrightarrow{\mathrm{BorrowLoc}\langle x\rangle} \langle M, \mathbf{ref}\,\langle x, [], \mathbf{mut}\rangle\rangle}\ \mathbf{BorrowLoc}$$

$$\frac{r = \mathbf{ref}\,\langle l, p, q\rangle}{\langle M, r :: S\rangle \xrightarrow{\mathrm{BorrowField}\langle f\rangle} \langle M, \mathbf{ref}\,\langle l, p :: f, q\rangle :: S\rangle}\ \mathbf{BorrowField}$$

$$\frac{r = \mathbf{ref}\,\langle l, p, q\rangle}{\langle M, r :: S\rangle \xrightarrow{\mathrm{FreezeRef}} \langle M, \mathbf{ref}\,\langle M, \mathbf{ref}\,\langle l, p, \mathbf{immut}\rangle\rangle\rangle}\ \mathbf{FreezeRef}$$

$$\frac{M(r) = u}{\langle M, r :: S\rangle \xrightarrow{\mathrm{ReadRef}} \langle M, u :: S\rangle}\ \mathbf{ReadRef}$$

$$\frac{r\,\mathbf{mut} \quad M(r) = u}{\langle M, v :: r :: S\rangle \xrightarrow{\mathrm{WriteRef}} \langle M[r \mapsto v], S\rangle}\ \mathbf{WriteRef}$$

$$\frac{s = u \vee s = r}{\langle M, s :: S\rangle \xrightarrow{\mathrm{Pop}} \langle M, S\rangle}\ \mathbf{Pop}$$

$$\frac{M \rhd \mathbf{resource}\,\tau\{f_1\colon \tau_1, \ldots, f_n\colon \tau_n\} \quad M \rhd t\,\mathbf{Fresh}}{\langle M, [v_i]_{i=1}^n :: S\rangle \xrightarrow{\mathrm{Pack}\langle\tau\rangle} \langle M, \langle \mathbf{resource}\,\tau\{f_1\colon v_1, \ldots, f_n\colon v_n\} :: S, t\rangle\rangle}\ \mathbf{PackR}$$

$$\frac{M \rhd \mathbf{unrestricted}\,\tau\{f_1\colon \tau_1, \ldots, f_n\colon \tau_n\}}{\langle M, [u_i]_{i=1}^n :: S\rangle \xrightarrow{\mathrm{Pack}\langle\tau\rangle} \langle M, \mathbf{resource}\,\tau\{f_1\colon u_1, \ldots, f_n\colon u_n\} :: S\rangle}\ \mathbf{PackU}$$

$$\frac{}{\langle M, \kappa\tau\{f_1\colon v_1, \ldots, f_n\colon v_n\} :: S\rangle \xrightarrow{\mathrm{Unpack}} \langle M, v_1 :: \cdots :: v_n :: S\rangle}\ \mathbf{Unpack}$$

$$\frac{}{\langle M, S\rangle \xrightarrow{\mathrm{LoadConst}\langle a\rangle} \langle M, a :: S\rangle}\ \mathbf{LoadConst}$$

$$\frac{|\mathrm{op}| = n}{\langle M, u_1 :: \cdots :: u_n :: S\rangle \xrightarrow{\mathrm{Op}} \langle M, \mathrm{op}(u_1, \ldots, u_n) :: S\rangle}\ \mathbf{StackOp}$$