Verified Move

Zekun wang

March 5, 2023

Abstract

This is supposed to be a more reable version of the Coq formalization.

1 Definitions

1.1 Identifiers

$$\begin{array}{lll} & & \text{ModuleName} \\ & & \text{StructName} \\ f & \in & \text{FieldName} \\ x & \in & \text{VarName} \end{array}$$

1.2 Types and Kinds

$$\begin{array}{lllll} & & \text{Kind} & = & \text{resource } | \text{unrestricted} \\ & & \text{ModuleId} & = & \text{AccountAddr} \times \text{ModuleName} \\ s & \in & \text{StructID} & = & \text{ModuleID} \times \text{StructName} \\ & & \text{StructType} & = & \text{StructID} \\ & & \text{PrimitiveType} & = & \text{AccountAddr} \cup \text{Bool} \cup \text{Unsigned}\,64 \cup \text{Bytes} \\ a & \in & \text{AccountAddr} & & \\ b & \in & \text{Bool} & & \\ n & \in & \text{Unsigned}\,64 & & \\ \vec{b} & \in & \text{Bytes} & & \\ \tau & \in & \text{NonRefType} & = & \text{StructType} \times \text{Primitive} \\ & & \text{Type} & = & \tau \ | \& \,\text{mut}\, \tau \ | \& \tau \end{array}$$

1.3 Values

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

1.4 Memory

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

We write $M(l)$ to mean the value stored at l (if any) in memory M, where l is a local variable or a reference. We write $M[l \mapsto v]$ to mean the memory with l updated to have value v, and otherwise identical with M. We use $M \backslash x$ to mean 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	$=$	$\mathrm{RuntimeValue}^*$
l	$\in$	Location	$=$	$x.p \mid s.p \mid n.p$

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

2 Judgements

Judgement	**Meaning**
rq	reference r has mutability q
$M \triangleright t\mathrm{Fresh}$	tag t is fresh in M
$M \triangleright \quad \kappa\tau\{f_1 : \tau_1, \dots, f_n : \tau_n\}$	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
$\sigma \to \sigma'$	$\sigma \xrightarrow{i} \sigma'$ for some instruction i
$l : t \in \sigma$	$\sigma(l) = v$ and $\mathrm{tag}(v) = t$ for some value v

Table 1:

3 Operational Semantics

3.1 Local Instructions

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4 Resource Safety

Definition 1 *A local state σ is well-formed iff $p_1 p_2 \quad : \quad t \in \sigma$ implies $p_1 = p_2$. That is, resource tags are unique; a tag can appear at most once in σ.*

Proposition 1 *Small step evaluation preserves well-formedness. That is, if σ is well-formed and $\sigma \to \sigma'$, then σ' is well-formed.*

We would always assume that states are well-formed.

Theorem 1 (Local resource safety) *If $\sigma \to \sigma'$, then $\mathrm{tag}(\sigma) = \mathrm{tag}(\sigma') \vee \mathrm{tag}(\sigma) \cup \{t\} = \mathrm{tag}(\sigma') \vee \mathrm{tag}(\sigma) = \mathrm{tag}(\sigma') \cup \{t\}$. Further, the second case happens iff $\sigma = \langle M, S \rangle \xrightarrow{\text{pack}} \langle M, (v, t) : S \rangle = \sigma'$, and the third case happens iff $\sigma = \langle M, (v, t) : S \rangle \xrightarrow{\text{unpack}} \langle M, S \rangle = \sigma'$.*