Verified Move

BY ZEKUN WANG

Abstract

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

1 Definitions

1.1 Identifiers

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

1.2 Types and Kinds

$$
\begin{array}{rcll}
 & \text{Kind} & = & \textbf{resource} \mid \textbf{unrestricted} \\
 & \text{ModuleId} & = & \text{AccountAddr} \times \text{ModuleName} \\
s & \in \text{StructID} & = & \text{ModuleID} \times \text{StructName} \\
 & \text{StructType} & = & \text{StructID} \\
 & \text{PrimitiveType} & = & \text{AccountAddr} \cup \text{Bool} \cup \text{Unsigned64} \cup \text{Bytes} \\
a & \in \text{AccountAddr} & & \\
b & \in \text{Bool} & & \\
n & \in \text{Unsigned64} & & \\
\vec{b} & \in \text{Bytes} & & \\
\tau & \in \text{NonRefType} & = & \text{StructType} \times \text{Primitive} \\
 & \text{Type} & = & \tau \mid \&\text{mut}\,\tau \mid \&\tau
\end{array}
$$

1.3 Values

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

1.4 Memory

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

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 $M \setminus x$ to mean the memory with x removed, and otherwise identical with M.

1.5 Local Evaluation State

$$
\begin{array}{rcll}
\sigma & \in & \text{LocalState} & = & \langle M, S \rangle \\
S & \in & \text{LocalStack} & = & \text{RuntimeValue}^* \\
l & \in & \text{Location} & = & x.p \mid s.p \mid n.p
\end{array}
$$

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

2 Judgements

Judgement	Meaning
rq	reference r has mutability q
$M \rhd t$ **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
$\sigma \rightarrow \sigma'$	$\sigma \xrightarrow{i} \sigma'$ for some instruction i
$l \colon t \in \sigma$	$\sigma(l) = v$ and $\text{tag}(v) = t$ for some value v

Table 1.

3 Operational Semantics

3.1 Local Instructions

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4 Resource Safety

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

Proposition 2. *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 3. (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 $\sigma = \langle M, (v, t) :: S\rangle \xrightarrow{\text{unpack}} \langle M, S\rangle = \sigma'$.*