Coq formalization

Billy Bai

February 2024

1 Whole setting

$$
\begin{array}{rcl}
P & ::= & \overline{S} \\
ct & ::= & \overline{ClassDecl_i} \\
\text{ClassDecl}_i & ::= & \text{class } C_i \ \{ \ \overline{F_n} \ K \ \overline{M_j} \ \} \\
F_i & ::= & f_i : \ Tf_i; \\
K & ::= & C \ (\overline{p_i : Tf_i}) \ : \text{ret} : C_i : \ \{ \ \overline{\text{this}.f_i := p_i;} \ \text{ret} := this \ \} \\
M_j & ::= & \text{def } m_j(\text{this} : C_i, \ \overline{x_k : T_k}) \ : \text{ret} : T_r : \ \{S\} \\
e & ::= & x \mid c \mid x.f \\
S & ::= & \text{skip} \mid x := y \mid x := y.f \mid x.f := y \mid x := y.m_k(\overline{z}) \\
 & \mid & \text{var } x : T := e \ \textsf{in} \ S \mid \text{var } x : C_i := \text{new } \rho \ C(\overline{y}) \ \textsf{in} \ S \mid \textsf{if} \ e \ \text{then} \ S_1 \ \textsf{else} \ S_2 \\
 & \mid & \textsf{while} \ e \ \textsf{do} \ S \mid S_1; \ S_2 \\
T & ::= & \text{Bool} \mid \text{C}_i \mid TUnit \\
c & ::= & True \mid False \\
\Gamma & ::= & \overline{x_i : T_i}
\end{array}
$$

$$
\begin{array}{rcl}
v & ::= & c \mid \&l \\
c & ::= & True \mid False \\
l & \in & \mathbb{N} \\
h & ::= & \overline{l_i := (T, fs)} \\
fs & ::= & \overline{fv_k} \\
fv_i & ::= & v \\
\sigma & ::= & \overline{x_i := (T_i, v_i)}
\end{array}
$$

$\boxed{WF(ct)}$

$$\frac{}{\mathrm{WF}(\emptyset)}\;(\textsc{wf-ct-nil}) \qquad \frac{\mathrm{WF}(ct) \quad n = length(ct) \quad ClassDecl_k = \text{class } C_i \; \{ \; \overline{F_n} \; K \; \overline{M_j} \; \}}{\mathrm{WF}(ClassDecl_k :: ct)}\;(\textsc{wf-ct-nil})$$

$\boxed{WF_ct(T)}$

$$\frac{}{\mathrm{WF}_{ct}(Bool)}\;(\textsc{wf-t-Bool}) \qquad \frac{\mathrm{ClassDecl}_i \in ct}{\mathrm{WF}_{ct}(C_i)}\;(\textsc{wf-t-class})$$

Figure 1: Definition of well-formness.

$\boxed{\Gamma \vdash e : \; T}$

$$\frac{}{\Gamma \vdash c : \mathrm{Bool}}\;(\text{T-}\textsc{c}) \qquad \frac{\Gamma(x) = T \quad \mathrm{WF}_{ct}(T)}{\Gamma \vdash x : T}\;(\text{T-}\textsc{var})$$

$$\frac{\Gamma(x) = \mathrm{C}_i \quad \mathrm{lookup}_{ct}(C_i, f) = T \quad \mathrm{WF}_{ct}(T)}{\Gamma \vdash x.f : T}\;(\text{T-}\textsc{facc})$$

Figure 2: Type rules for expressions.

$$\boxed{\Gamma \vdash s :\ TUnit}$$

$$\text{(T-SKIP)}\quad \frac{}{\Gamma \vdash \mathit{skip} : \mathrm{TUnit}} \qquad \text{(T-ASSIGN)}\quad \frac{\Gamma \vdash x : T \quad \Gamma \vdash y : T}{\Gamma \vdash x := y : \mathrm{TUnit}} \qquad \text{(T-LOAD)}\quad \frac{\Gamma \vdash x : T \quad \Gamma \vdash y.f : T}{\Gamma \vdash x := y.f : \mathrm{TUnit}}$$

$$\text{(T-STORE)}\quad \frac{\Gamma \vdash x.f : T \quad \Gamma \vdash y : T}{\Gamma \vdash x.f := y : \mathrm{TUnit}}$$

$$\text{(T-METHODCALL)}\quad \frac{\begin{array}{c}\mathrm{ClassDecl}_i = \mathrm{Class}\ C_i\{\ \overline{F_n}\ K\ \overline{M_j}\} \quad M_k = \mathrm{def}\ m_k(\mathrm{this} : C_i,\ \overline{x_k : T_k}) : \mathrm{ret} : T_r : \{S\} \\ \Gamma \vdash x : T_r \quad \Gamma \vdash y : C_i \quad \Gamma \vdash \overline{z} : \overline{T_k} \quad \Gamma(\overline{z}) \neq None \\ \Gamma \vdash ret[y/this, \overline{z}/\overline{x_k}] : T_r \quad \Gamma \vdash S[y/this, \overline{z}/\overline{x_k}] : \mathrm{TUnit}\end{array}}{\Gamma \vdash x := y.m_k(\overline{z}) : \mathrm{TUnit}}$$

$$\text{(T-LETTERM)}\quad \frac{FreeVar^s(S,\Gamma) = x \quad \Gamma \vdash e : T \quad \Gamma, x \mapsto T \vdash S : \mathrm{TUnit}}{\Gamma \vdash var\ x : T := e\ in\ S : \mathrm{TUnit}}$$

$$\text{(T-LETNEW)}\quad \frac{\begin{array}{c}\mathrm{ClassDecl}_i = \mathrm{Class}\ C_i\{\ \overline{F_n}\ K\ \overline{M_j}\} \quad K = C\ (\overline{p_i : Tf_i})\ : \mathrm{ret} : C_i :\ \{\overline{\mathrm{this}.f_i := p_i};\ \mathrm{ret} := this\} \\ FreeVar^s(S,\Gamma) = x \quad \Gamma \vdash \overline{y} : \overline{Tf_i} \quad \Gamma, x \mapsto C_i \vdash S : \mathrm{TUnit}\end{array}}{\Gamma \vdash var\ x : T := new\ C_i(\overline{y})\ in\ S : \mathrm{TUnit}}$$

$$\text{(T-IF)}\quad \frac{\Gamma \vdash e : Bool \quad \Gamma \vdash S_1 : \mathrm{TUnit} \quad \Gamma \vdash S_2 : \mathrm{TUnit}}{\Gamma \vdash if\ e\ then\ S_1\ else\ S_2 : \mathrm{TUnit}}$$

$$\text{(T-LOOP)}\quad \frac{\Gamma \vdash e : Bool \quad \Gamma S while\ e\ do\ S : \mathrm{TUnit}}{\Gamma \vdash while\ e\ do\ S : \mathrm{TUnit}} \qquad \text{(T-SEQ)}\quad \frac{\Gamma \vdash S_1 : \mathrm{TUnit} \quad \Gamma \vdash S_2 : \mathrm{TUnit}}{\Gamma \vdash S_1; S_2 : \mathrm{TUnit}}$$

Figure 3: Type rules for statements.

$$\boxed{h;\ \sigma \vdash e \rightsquigarrow (T, v)}$$

$$\text{(S-C)}\ \frac{}{h;\ \sigma \vdash c \rightsquigarrow (Bool, c)} \qquad \text{(S-ADD)}\ \frac{h(l) = (C_i, fs)}{h;\ \sigma \vdash \&l \rightsquigarrow (C_i, \&l)}$$

$$\text{(S-ADDFACC)}\ \frac{h(l) = (C_i, fs) \quad fs(f) = (T, v) \quad Value(v)}{h;\ \sigma \vdash \&l.f \rightsquigarrow (T, v)} \qquad \text{(S-VAR)}\ \frac{\sigma(x) = (T, v) \quad Value(v)}{h;\ \sigma \vdash x \rightsquigarrow (T, v)}$$

$$\text{(S-VFACC)}\ \frac{\sigma(x) = (C_i, \&l) \quad h(l) = (C_i, fs) \quad fs(f) = (T, v) \quad Value(v)}{h;\ \sigma \vdash x.f \rightsquigarrow (T, v)}$$

Figure 4: Operational semantics for expressions.

$$\boxed{h;\ \sigma \vdash v : T}$$

$$\text{(R-C)}\ \frac{}{h;\ \sigma \vdash c : Bool} \qquad \text{(R-ADD)}\ \frac{\text{ClassDecl}_i \in ct \quad h(l) = (C_i, fs)}{h;\ \sigma \vdash \&l : C_i}$$

Figure 5: Runtime value type.

$$\boxed{(h;\sigma;S) \rightsquigarrow (h';\sigma')}$$

$$\text{(S-SKIP)}\quad \frac{}{(h;\sigma;skip) \rightsquigarrow (h;\ \sigma)}$$

$$\text{(S-ASSIGN)}\quad \frac{h;\ \sigma \vdash y \rightsquigarrow (T,v) \quad \sigma(x) \neq None}{(h;\sigma;x := y) \rightsquigarrow (h;\ \sigma[x \mapsto (T,v)])}$$

$$\text{(S-LOAD)}\quad \frac{h;\ \sigma \vdash y.f \rightsquigarrow (T,v) \quad \sigma(x) \neq None}{(h;\sigma;x := y.f) \rightsquigarrow (h;\ \sigma[x \mapsto (T,v)])}$$

$$\text{(S-STORE)}\quad \frac{\begin{array}{c} h;\ \sigma \vdash y \rightsquigarrow (T,v) \quad \sigma(x) = (C_i, \&l) \\ h(l) = (C_i, fs) \quad fs(f) = (T,v') \quad Value(v') \end{array}}{(h;\sigma;x.f := y) \rightsquigarrow (h[l \mapsto (C_i, fs[f \mapsto (T,v)])]);\ \sigma)}$$

$$\text{(S-METHODCALL)}\quad \frac{\begin{array}{c} \sigma(y) = (C_i, \&l) \quad \text{ClassDecl}_i = \text{Class } C_i\{\ \overline{F_n}\ K\ \overline{M_j}\} \quad M_k = \text{def } m_k(\text{this} : C_i,\ \overline{x_k : T_k}) : \text{ret} : T_r : \{S\} \\ \sigma(x) \neq None \quad \sigma(z) \neq None \quad (h;\sigma;S[y/this, \overline{z}/\overline{x_k}]) \rightsquigarrow (h',\sigma') \quad h';\ \sigma' \vdash ret \rightsquigarrow (T_r, v_r) \end{array}}{(h;\sigma;x := y.m_k(\overline{z})) \rightsquigarrow (h';\ \sigma'[x \mapsto (T_r, v_r)])}$$

$$\text{(S-LETTERM)}\quad \frac{h;\ \sigma \vdash e \rightsquigarrow (T,r) \quad (h;\sigma[x \mapsto (T,r)];S) \rightsquigarrow (h';\ \sigma'[x \mapsto (T,r')]) \quad FreeVar^r(S,\sigma) = x}{(h;\sigma;var\ x : T := e\ in\ S) \rightsquigarrow (h';\ \sigma' - x)}$$

$$\text{(S-LETNEW)}\quad \frac{\begin{array}{c} \sigma(\overline{y}) \neq None \quad h;\ \sigma \vdash \overline{y} \rightsquigarrow O(= \overline{(T,v)}) \\ \text{ClassDecl}_i = \text{Class } C_i\{\ \overline{F_n}\ K\ \overline{M_j}\} \quad O.1 = \overline{F_n} \quad l = fresh(h) \\ (h[l \mapsto (C_i, O)];\sigma[x \mapsto \&l], S) \rightsquigarrow (h';\sigma;) \quad FreeVar^r(S,\sigma) = x \end{array}}{(h;\sigma;var\ x : T := new\ C_i(\overline{y})\ in\ S) \rightsquigarrow (h';\ \sigma' - x)}$$

$$\text{(S-IFTRUE)}\quad \frac{\begin{array}{c} h;\ \sigma \vdash e \rightsquigarrow (Bool, True) \\ (h;\sigma;S_1) \rightsquigarrow (h';\ \sigma') \\ FreeVar^r(S_1,\sigma) = \emptyset \end{array}}{(h;\sigma;if\ e\ then\ S_1\ else\ S_2) \rightsquigarrow (h';\ \sigma')}$$

$$\text{(S-IFFALSE)}\quad \frac{\begin{array}{c} h;\ \sigma \vdash e \rightsquigarrow (Bool, False) \\ (h;\sigma;S_2) \rightsquigarrow (h';\ \sigma') \\ FreeVar^r(S_2,\sigma) = \emptyset \end{array}}{(h;\sigma;if\ e\ then\ S_1\ else\ S_2) \rightsquigarrow (h';\ \sigma')}$$

$$\text{(S-LOOPTRUE)}\quad \frac{\begin{array}{c} h;\ \sigma \vdash e \rightsquigarrow (Bool, True) \quad (h;\sigma;S) \rightsquigarrow (h';\ \sigma') \\ (h';\sigma';while\ e\ do\ S) \rightsquigarrow (h'';\ \sigma'') \end{array}}{(h;\sigma;while\ e\ do\ S) \rightsquigarrow (h'';\ \sigma'')}$$

$$\text{(S-LOOPFALSE)}\quad \frac{h;\ \sigma \vdash e \rightsquigarrow (Bool, False)}{(h;\sigma;while\ e\ do\ S) \rightsquigarrow (h;\ \sigma)}$$

$$\text{(S-SEQUENCE)}\quad \frac{(h;\sigma;S_1) \rightsquigarrow (h';\ \sigma') \quad (h';\sigma';S_2) \rightsquigarrow (h'';\ \sigma'')}{(h;\sigma;S_1;S_2) \rightsquigarrow (h'';\ \sigma'')}$$

Figure 6: Operational semantics for statements.

$\boxed{\textit{StoreOK}\ \Gamma\ \sigma\ h\ ct}$

$$\frac{dom(\Gamma) = dom(\sigma) \quad \mathrm{WF}(ct) \qquad \forall x \in dom(\Gamma),\ \Gamma \vdash x : T \implies \exists v, \sigma(x) = (T, v) \land Value(v) \land h;\ \sigma \vdash v : T}{\textit{StoreOK}\ \Gamma\ \sigma\ h\ ct}$$

$\boxed{\textit{HeapOK}\ \Gamma\ \sigma\ h\ ct}$

$$\frac{\forall o \in dom(h), h(o) = (C_i, fs) \implies \mathrm{ClassDecl}_i = \mathrm{Class}\ C_i\{\overline{F_n}\ K\ \overline{M_j}\} \land \overline{F_n} = \overline{f_i : Tf_i} \land fs.1 = \overline{Tf_i} \land h; \sigma \vdash fs.2 : fs.1}{\textit{HeapOK}\ \Gamma\ \sigma\ h\ ct}$$

$\boxed{\textit{HeapStoreOK}\ \Gamma\ \sigma\ h\ ct}$

$$\frac{\forall x \in dom(\Gamma), \Gamma(x) = C_i \implies \exists l, \sigma(x) = (C_i, \&l) \land h(l) = (C_i, fs) \land \mathrm{ClassDecl}_i = \mathrm{Class}\ C_i\{\overline{F_n}\ K\ \overline{M_j}\} \land \overline{F_n} = \overline{f_i : Tf_i} \land \overline{Tf_i} = fs.1 \land h; \sigma \vdash fs.2 : \overline{Tf_i}}{\textit{HeapStoreOK}\ \Gamma\ \sigma\ h\ ct}$$

$\boxed{\textit{CtxOK}\ \Gamma\ \sigma\ h\ ct}$

$$\frac{\textit{StoreOK}\ \Gamma\ \sigma\ h\ ct \land \textit{HeapOK}\ \Gamma\ \sigma\ h\ ct \land \textit{HeapStoreOK}\ \Gamma\ \sigma\ h\ ct}{\textit{CtxOK}\ \Gamma\ \sigma\ h\ ct}$$

Figure 7: Definition of safety properties.

$\boxed{\text{Type Safety}}$

$$\textit{CtxOK}\ \Gamma\ \sigma\ h\ ct \land \Gamma \vdash S : \mathrm{TUnit} \implies \exists \sigma'\ h', (h;\ \sigma, S) \rightsquigarrow (h';\ \sigma') \land \textit{CtxOK}\ \Gamma\ \sigma'\ h'\ ct$$

Figure 8: Type safety theorem (progress).