Semantic Type Soundness for System Capless

Yichen Xu
Autumn 2025

This document drafts semantic type soundness for System Capless. We first formally define System Capless, then sketch logical type soundness proof for it.

1 Definitions of System Capless

The following sections define System Capless.

1.1 Syntax

x, y, z			**Term Variable**
T, U			**Type Variable**
c			**Capture Variable**
S, R	$:=$		**Shape Type**
		$\top$	Top
		X	Type Variable
		$(x : T) \to E$	Function
		$[X <: S] \to E$	Type Function
		$[c <: B] \to E$	Capture Function
		Unit	Unit
		Capability	Capability
S, R	$:=$	$S^\wedge C$	**Shape Type**
E, F	$:=$		**Existential Type**
		$\exists c.T$	existential type
		T	capturing type
θ	$:=$		**Capture**
		x	variable
		c	capability
C	$:=$	$\{\theta_1, ..., \theta_n\}$	**Capture Set**
B	$:=$	$* \mid C$	**Capture Bound**

s, t, u	$:=$		**Term**
		a	answer
		$x\,y$	application
		$x[S]$	type application
		$x[C]$	capture application
		$\text{let } x = t \text{ in } u$	let
		$\text{unpack } t \text{ as } \langle c, x \rangle \text{ in } u$	unpack
a	$:=$		**Answer**
		x	variable
		v	value
v	$:=$		**Value**
		$()$	Unit
		$\lambda(x : T).t$	Function
		$\lambda[X <: S].t$	Type Function
		$\lambda[c <: B].t$	Capture Function
		$\langle C, x \rangle$	Packing
Γ	$:=$		**Type Context**
		$[]$	
		$\Gamma, x : T$	
		$\Gamma, X <: S$	
		$\Gamma, c <: B$	
Σ	$:=$	$\cdot \mid \Sigma, x \mapsto v \mid \Sigma, x \mapsto \mathbf{cap}$	**Store**

Figure 1: Syntax of System Capless.

Figure 1 defines the syntax of System Capless. It is an extension of System $\mathsf{CC}_{<:\Box}$.

1.2 Type System

Figure 2 defines the type system of System Capless.

1.3 Operational Semantics

Figure 3 defines the small-step evaluation relation, $\Sigma \mid s \xrightarrow{C} \Sigma' \mid s'$, for System Capless. This evaluation relation is indexed by a capability set C, restricting the program from using capabilities outside C during evaluation. We write $\Sigma \mid s \xrightarrow[*]{C} \Sigma' \mid s'$ for the reflexive, transitive closure of $\Sigma \mid s \xrightarrow{C} \Sigma' \mid s'$, with all C being all capability sets along the trace unioned together. In other words, given $\Sigma_1 \mid t_1 \xrightarrow[1]{C} \Sigma_2 \mid t_2 \xrightarrow[2]{C} ... \xrightarrow[n]{C} \Sigma_{n+1} \mid t_{n+1}$, we have $\Sigma_1 \mid t_1 \xrightarrow[*]{C_1 \cup C_2 \cup ... \cup C_n} \Sigma_{n+1} \mid t_{n+1}$.

Figure 4 defines a big-step evaluation relation. $\Sigma \mid t \overset{C}{\twoheadrightarrow} Q$ means that for any possible evaluation $\Sigma \mid t \xrightarrow[*]{C} \Sigma' \mid a$, the resulting configuration satisfies the postcondition Q.

Typing $C; \Gamma \vdash t : E$

$$\frac{x : S\wedge C \in \Gamma}{\{x\}; \Gamma \vdash x : S\wedge\{x\}} \quad \text{(var)}$$

$$\frac{C_1; \Gamma \vdash e : T_1 \quad \Gamma \vdash C_2, T_2 \text{ wf} \quad \Gamma \vdash T_1 <: T_2 \quad \Gamma \vdash C_1 <: C_2}{C_2; \Gamma \vdash e : T_2} \quad \text{(sub)}$$

$$\frac{}{\{\}; \Gamma \vdash () : \mathsf{Unit}} \quad \text{(var)}$$

$$\frac{C; (\Gamma, x : T) \vdash t : E}{\{\}; \Gamma \vdash \lambda(x : T)t : ((x : T) \to E)\wedge(C \setminus \{x\})} \quad \text{(abs)}$$

$$\frac{C; (\Gamma, X <: S) \vdash t : E}{\{\}; \Gamma \vdash \lambda[X <: S]t : ([X <: S] \to E)\wedge C} \quad \text{(tabs)}$$

$$\frac{C; (\Gamma, c <: B) \vdash t : E \quad \Gamma \vdash C \text{ wf}}{\{\}; \Gamma \vdash \lambda[c <: B]t : ([X <: S] \to E)\wedge C} \quad \text{(cabs)}$$

$$\frac{C; \Gamma \vdash x : ((z : T) \to E)\wedge C_f \quad C; \Gamma \vdash y : T}{C; \Gamma \vdash x\, y : [z := x]E} \quad \text{(app)}$$

$$\frac{C; \Gamma \vdash x : ([X <: S] \to E)\wedge C_f}{C; \Gamma \vdash x[S] : [X := S]E} \quad \text{(tapp)}$$

$$\frac{C; \Gamma \vdash x : ([c <: B] \to E)\wedge C_f \quad \Gamma \vdash C <: B}{C; \Gamma \vdash x[C] : [c := C]E} \quad \text{(capp)}$$

$$\frac{C; \Gamma \vdash x : [c := C]T}{C; \Gamma \vdash \langle C, x \rangle : \exists c.T} \quad \text{(pack)}$$

$$\frac{C; \Gamma \vdash t : T \quad C; (\Gamma, x : T) \vdash u : U \quad \Gamma \vdash C, U \text{ wf}}{C; \Gamma \vdash \text{let } x = t \text{ in } u : U} \quad \text{(let)}$$

$$\frac{C; \Gamma \vdash t : \exists c.T \quad C; (\Gamma, c <: *, x : T) \vdash u : U \quad \Gamma \vdash (C \setminus \{x\}), U \text{ wf}}{C \setminus \{x\}; \Gamma \vdash \text{unpack } t \text{ as } \langle c, x \rangle \text{ in } u : U} \quad \text{(unpack)}$$

Subcapturing $\Gamma \vdash C_1 <: C_2, \Gamma \vdash C <: B$

$$\frac{C_1 \subseteq C_2}{\Gamma \vdash C_1 <: C_2} \quad \text{(sc-subset)}$$

$$\frac{\Gamma \vdash C_1 <: C_2 \quad \Gamma \vdash C_2 <: C_3}{\Gamma \vdash C_1 <: C_3} \quad \text{(sc-trans)}$$

$$\frac{\Gamma \vdash C_1 <: C \quad \Gamma \vdash C_2 <: C}{\Gamma \vdash C_1 \cup C_2 <: C} \quad \text{(sc-union)}$$

$$\frac{x : S\wedge C \in \Gamma}{\Gamma \vdash \{x\} <: C} \quad \text{(sc-var)}$$

$$\frac{c <: C \in \Gamma}{\Gamma \vdash \{c\} <: C} \quad \text{(sc-cvar)}$$

$$\frac{}{\Gamma \vdash C <: *} \quad \text{(sc-bound)}$$

Subtyping $\Gamma \vdash E_1 <: E_2$

$$\frac{}{\Gamma \vdash S <: \top} \quad \text{(top)}$$

$$\frac{}{\Gamma \vdash S <: S} \quad \text{(refl)}$$

$$\frac{\Gamma \vdash S_1 <: S_2 \quad \Gamma \vdash S_2 <: S_3}{\Gamma \vdash S_1 <: S_3} \quad \text{(trans)}$$

$$\frac{X <: S \in \Gamma}{\Gamma \vdash X <: S} \quad \text{(tvar)}$$

$$\frac{(\Gamma, c <: *) \vdash T_1 <: T_2}{\Gamma \vdash \exists c.T_1 <: \exists c.T_2} \quad \text{(exists)}$$

$$\frac{\Gamma \vdash T_2 <: T_1 \quad (\Gamma, x : T_2) \vdash E_1 <: E_2}{\Gamma \vdash (x : T_1) \to E_1 <: (x : T_2) \to E_2} \quad \text{(fun)}$$

$$\frac{\Gamma \vdash S_2 <: S_1 \quad (\Gamma, X <: S_2) \vdash E_1 <: E_2}{\Gamma \vdash [X <: S_1] \to E_1 <: [X <: S_2] \to E_2} \quad \text{(tfun)}$$

$$\frac{\Gamma \vdash B_2 <: B_1 \quad (\Gamma, c <: B_2) \vdash E_1 <: E_2}{\Gamma \vdash [c <: B_1] \to E_1 <: [c <: B_2] \to E_2} \quad \text{(cfun)}$$

Figure 2: Type System of System Capless.

Proposition 1.3.1: Given $\Sigma \mid t \overset{C}{\twoheadrightarrow} Q$, there exist Σ' and a such that $\Sigma \sqsubset \Sigma' \wedge Q(a)(\Sigma')$.

Proposition 1.3.2: Given $\Sigma \mid t \overset{C}{\twoheadrightarrow} Q$, for any Σ' and a such that $\Sigma \mid t \xrightarrow[*]{C} \Sigma' \mid a$, we have $Q(a)(\Sigma')$.

$$\Sigma \mid x\,y \xrightarrow{\{\}} \Sigma \mid [z := y]t \qquad \text{if } \Sigma(x) = \lambda(z : T)t \qquad \text{(e-apply)}$$

$$\Sigma \mid x\,y \xrightarrow{\{x\}} \Sigma \mid () \qquad \text{if } \Sigma(x) = \mathbf{cap} \text{ and } \Sigma(y) = () \qquad \text{(e-invoke)}$$

$$\Sigma \mid x[S] \xrightarrow{\{\}} \Sigma \mid [X := \top]t \qquad \text{if } \Sigma(x) = \lambda[X <: S']t \qquad \text{(e-tapply)}$$

$$\Sigma \mid x[C] \xrightarrow{\{\}} \Sigma \mid [c := \{\}]t \qquad \text{if } \Sigma(x) = \lambda[c <: B]t \qquad \text{(e-capply)}$$

$$\Sigma \mid \text{let } x = t \text{ in } u \xrightarrow{C} \Sigma' \mid \text{let } x = t' \text{ in } u \qquad \text{if } \Sigma \mid t \xrightarrow{C} \Sigma' \mid t' \qquad \text{(e-ctx1)}$$

$$\Sigma \mid \text{unpack } t \text{ as } \langle c, x\rangle \text{ in } u \xrightarrow{C} \Sigma' \mid \text{unpack } t' \text{ as } \langle c, x\rangle \text{ in } u \qquad \text{if } \Sigma \mid t \xrightarrow{C} \Sigma' \mid t' \qquad \text{(e-ctx2)}$$

$$\Sigma \mid \text{let } x = y \text{ in } t \xrightarrow{\{\}} \Sigma \mid [x := y]t \qquad \text{(e-rename)}$$

$$\Sigma \mid \text{let } x = v \text{ in } t \xrightarrow{\{\}} (\Sigma, x \mapsto v) \mid t \qquad \text{(e-lift)}$$

$$\Sigma \mid \text{unpack } \langle c', x'\rangle \text{ as } \langle c, x\rangle \text{ in } u \xrightarrow{\{\}} \Sigma \mid [c := c'][x := x']u \qquad \text{(e-unpack)}$$

Figure 3: Operational Semantics of System Capless.

Proposition 1.3.3: If $\forall \Sigma' \forall a, \left(\Sigma \mid t \xrightarrow[*]{C} \Sigma' \mid a\right) \to Q(a)(\Sigma')$, then $\Sigma \mid t \overset{C}{\twoheadrightarrow} Q$.

Proposition 1.3.1, Proposition 1.3.2, and Proposition 1.3.3 establish the equivalence between small-step evaluation and big-step evaluation.

2 Semantic Type Soundness

2.1 Type Denotation

The types are interpreted into predicates. The interpretation is done under a type environment ρ, which maps type variables to predicates of the type $\mathrm{CaptureSet} \to \mathrm{Term} \to \mathrm{Heap} \to \mathrm{Prop}$ (representing the denotation function for shape types parameterized by capability sets); term variables to capability sets; and capture variables to capability sets.

$$\frac{Q(a)(\Sigma)}{\Sigma \mid a \overset{C}{\twoheadrightarrow} Q} \ \text{(bs-ans)} \qquad \frac{\Sigma(x) = \lambda(z : T)t \quad \Sigma \mid [z := y]t \overset{C}{\twoheadrightarrow} Q}{\Sigma \mid x\,y \overset{C}{\twoheadrightarrow} Q} \ \text{(bs-apply)}$$

$$\frac{\Sigma(x) = \lambda[X <: S]t \quad \Sigma \mid [X := \top]t \overset{C}{\twoheadrightarrow} Q}{\Sigma \mid x[S'] \overset{C}{\twoheadrightarrow} Q} \ \text{(bs-tapply)} \qquad \frac{\Sigma(x) = \lambda[c <: B]t \quad \Sigma \mid [c := \{\}]t \overset{C}{\twoheadrightarrow} Q}{\Sigma \mid x[C'] \overset{C}{\twoheadrightarrow} Q} \ \text{(bs-capply)}$$

$$\frac{\begin{array}{c}\Sigma(x) = \mathbf{cap} \quad \Sigma(y) = () \\ Q(())(\Sigma) \quad x \in C\end{array}}{\Sigma \mid x\,y \overset{C}{\twoheadrightarrow} Q} \ \text{(bs-invoke)}$$

$$\frac{\begin{array}{c}\Sigma \mid t \overset{C}{\twoheadrightarrow} Q' \\ \left(\forall v \forall \Sigma', \Sigma \sqsubseteq \Sigma' \to Q'(v)(\Sigma') \to (\Sigma', x \mapsto v) \mid u \overset{C}{\twoheadrightarrow} Q\right) \\ \left(\forall z \forall \Sigma', \Sigma \sqsubseteq \Sigma' \to Q'(z)(\Sigma') \to \Sigma' \mid [x := z]u \overset{C}{\twoheadrightarrow} Q\right)\end{array}}{\Sigma \mid \text{let } x = t \text{ in } u \overset{C}{\twoheadrightarrow} Q} \ \text{(bs-let)}$$

$$\frac{\begin{array}{c}\Sigma \mid t \overset{C}{\twoheadrightarrow} Q' \\ \left(\forall C' \forall z \forall \Sigma', \Sigma \sqsubseteq \Sigma' \to Q'(\langle C', z\rangle)(\Sigma') \to \Sigma \mid [x := z][c := \{\}]u \overset{C}{\twoheadrightarrow} Q\right)\end{array}}{\Sigma \mid \text{unpack } t \text{ as } \langle c, x\rangle \text{ in } u \overset{C}{\twoheadrightarrow} Q} \ \text{(bs-unpack)}$$

Figure 4: Big-Step Evaluation for System Capless.

Given predicates P and Q of type $\text{CaptureSet} \to \text{Term} \to \text{Heap} \to \text{Prop}$, we write $P \Rightarrow Q$ for the logical implication between them: $P \Rightarrow Q$ iff $\forall C \forall t \forall \Sigma, P(C)(t)(\Sigma) \to Q(C)(t)(\Sigma)$.

We write $\Sigma_1 \sqsubset \Sigma_2$ for subsumption between stores: $\Sigma_1 \sqsubset \Sigma_2$ iff $\forall x, \Sigma_1(x) = e \to \Sigma_2(x) = e$. Here e can be either a value v or a capability $\mathbf{cap}$.

We write $\mathcal{C}$ for a capture set that contains only capabilities in the store Σ, i.e. $\mathcal{C} = \{x_1, ..., x_n\}$ and $\forall i, \Sigma(x_i) = \mathbf{cap}$. The store is inferred from the context in which $\mathcal{C}$ is used.

We write $\Sigma(t)$ for resolving a term t in the store Σ. Basically, if t is a variable x, then $\Sigma(t) = \Sigma(x)$; otherwise, $\Sigma(t) = t$.

We write $[\![S]\!]_{\rho,\cdot}$ as a shorthand for $\lambda\mathcal{C}.[\![S]\!]_{\rho,\mathcal{C}}$.

We first define the denotation of capture sets and capture bounds, which maps them to sets of capabilities.

$$
\begin{aligned}
[\![\{\}]\!]_\rho &= \{\} \\
[\![\{x\}]\!]_\rho &= \rho(x) \\
[\![\{c\}]\!]_\rho &= \rho(c) \\
[\![C_1 \cup C_2]\!]_\rho &= [\![C_1]\!] \cup [\![C_2]\!] \\
[\![*]\!]_\rho &= \mathbb{N}
\end{aligned}
$$

Type denotations are defined as follows. The denotation function now acts on shape types and takes a capability set as an additional parameter.

$$
\begin{aligned}
[\![\top]\!]_{\rho,\mathcal{C}} &= \lambda t.\lambda\Sigma.\ \mathsf{True} \\
[\![\mathsf{Unit}]\!]_{\rho,\mathcal{C}} &= \lambda t.\lambda\Sigma.\ t = () \\
[\![\mathsf{Capability}]\!]_{\rho,\mathcal{C}} &= \lambda t.\lambda\Sigma.\exists z.z \in [\![C]\!]_\rho \wedge \Sigma(z) = \mathbf{cap} \\
[\![X]\!]_{\rho,\mathcal{C}} &= \rho(X)([\![C]\!]_\rho) \\
[\![(x : T) \to E]\!]_{\rho,\mathcal{C}} &= \lambda t.\lambda\Sigma.\exists T_0 t_0, \Sigma(t) = \lambda(z : T_0)t_0 \wedge \\
&\qquad \forall z \forall \Sigma', \Sigma \sqsubset \Sigma' \to [\![T]\!]_\rho(z)(\Sigma') \to [\![E]\!]^e_{\left(\left[x := [\![C_T]\!]_\rho\right]\rho\right),\mathcal{C}}([x := z]t_0)(\Sigma') \\
&\qquad \text{where } T = S_T \wedge C_T \\
[\![[X <: S] \to E]\!]_{\rho,\mathcal{C}} &= \lambda t.\lambda\Sigma.\exists S_0 t_0, \Sigma(t) = \lambda[X <: S_0]t_0 \wedge \\
&\qquad \forall P \forall \Sigma', \Sigma \sqsubset \Sigma' \to \left(P \Rightarrow [\![S]\!]_{\rho,\cdot}\right) \to [\![E]\!]^e_{([X := P]\rho),\mathcal{C}}([x := \top]t_0)(\Sigma') \\
[\![[c <: B] \to E]\!]_{\rho,\mathcal{C}} &= \lambda t.\lambda\Sigma.\exists B_0 t_0, \Sigma(t) = \lambda[c <: B_0]t_0 \wedge \\
&\qquad \forall C_0 \forall \Sigma', \Sigma \sqsubset \Sigma' \to \left(C_0 \subseteq [\![B]\!]_\rho\right) \to [\![E]\!]^e_{([c := C_0]\rho),\mathcal{C}}([x := \{\}]t_0)(\Sigma') \\
[\![S \wedge C]\!]_\rho &= [\![S]\!]_{\rho,[\![C]\!]_\rho} \\
[\![\exists c.T]\!]_\rho &= \lambda t.\lambda\Sigma.\exists\mathcal{C}, [\![T]\!]_{[c := \mathcal{C}]\rho}(t)(\Sigma) \\
[\![E]\!]^e_{\rho,\mathcal{C}} &= \lambda t.\lambda\Sigma.\Sigma \mid t \overset{\mathcal{C}}{\twoheadrightarrow} [\![E]\!]_\rho
\end{aligned}
$$

Then, we need to define semantic typing for contexts $(\Gamma, \rho) \vDash \Sigma$.

$$
\begin{aligned}
([], \rho) \vDash \Sigma &:= \mathsf{True} \\
((\Gamma, x : S \wedge C), \rho) \vDash \Sigma &:= [\![S]\!]_{\rho,[\![C]\!]_\rho}(x)(\Sigma) \wedge \rho(x) = [\![C]\!]_\rho \wedge (\Gamma, \rho) \vDash \Sigma \\
((\Gamma, X <: S), \rho) \vDash \Sigma &:= \left(\rho(X) \Rightarrow [\![S]\!]_{\rho,\cdot}\right) \wedge (\Gamma, \rho) \vDash \Sigma \\
((\Gamma, c <: B), \rho) \vDash \Sigma &:= \left(\rho(c) \subseteq [\![B]\!]_\rho\right) \wedge (\Gamma, \rho) \vDash \Sigma
\end{aligned}
$$

Finally, we can define semantic typing:

$$
C; \Gamma \vDash t : T := \forall\rho\forall\Sigma, (\Gamma, \rho) \vDash \Sigma \to [\![T]\!]^e_\rho(t)(\Sigma)
$$

Theorem 2.1.1 (Fundamental Theorem of Semantic Type Soundness): If $C; \Gamma \vdash t : T$ then $C; \Gamma \vDash t : T$. That is, syntactic typing implies semantic typing.