Full Proofs for Loci

Tobias Wrigstad, Filip Pizlo, Fadi Meawad, Lei Zhao, Jan Vitek

Purdue University

Abstract This note presents the full proofs for Loci [2]. There is one semantic change—we now allow `@Thread`, `@Context` and `@Shared` as class-level annotations[1], adding classes whose instances are always thread-local. We have also cleaned up the semantics by re-splitting rules that were combined to fit in within page limits, elaborated definitions, etc.

Note: *this techreport was posted with the explicit permission of Sophia Drossopoulou.*

1 A Formal Account of Loci

1.1 Syntax and Static Semantics

Loci's syntax is shown in Figure 1. For clarity, we use an explicit annotation `@Context`, instead of implicit annotations. Without loss of generality, we use a "named form," reminiscent of SSA but with mutable variables, where the results of field and variable accesses, method calls and instantiations must be immediately stored in a variable or field.

For simplicity, all rules have an implicit P on the left of the turnstile. We use the right-associative viewpoint-adaptation operator $\oplus$ to expand the `@Context` annotation thus:

$$\alpha_1 \oplus \alpha_2\ c = \begin{cases} \alpha_1\ c \text{ if } \alpha_2 = \texttt{@Context} \\ \alpha_2\ c \text{ otherwise} \end{cases} \qquad \alpha_1 \oplus \alpha_2 \oplus \alpha_3\ c = \alpha_1 \oplus (\alpha_2 \oplus \alpha_3\ c)$$

Class-Table For functions and relations g, g', define $g \bullet g'$ as:

$$(g \bullet g') = \begin{cases} g(x), \text{ if } g(x) \text{ is defined;} \\ g'(x) \text{ otherwise.} \end{cases}$$

Define $\bullet$ for tuples as:

$$(\overline{F}, \overline{M}) \bullet (\overline{F'}, \overline{M'}) = (\overline{F} \bullet \overline{F'}, \overline{M} \bullet \overline{M'})$$

[1] The old `@Thread` classes are now called `@Context`. Classes annotated `@Thread` now must always be thread-local.

$$
\begin{array}{lll}
P & ::= \overline{cd} & \textit{program} \\
C & ::= \alpha\ \texttt{class}\ c\ \texttt{extends}\ d\ \texttt{\{}\ \overline{F}\ \overline{M}\ \texttt{\}} & \textit{class declaration} \\
F & ::= \tau\, f & \textit{field} \\
M & ::= \tau\, m\texttt{(}\overline{\tau}\ \overline{x}\texttt{)}\ \texttt{\{}\ s\texttt{;return}\ y\ \texttt{\}} & \textit{method} \\
s & ::= s\texttt{;}s \mid \texttt{skip} \mid x = y.f \mid x = y \mid y.f = z \mid \tau\ x \mid & \textit{statement} \\
 & \quad x = \texttt{new}\ \tau\texttt{()} \mid x = y.m\texttt{(}\overline{z}\texttt{)} \mid x = \texttt{start}\ c\texttt{()} & \\
\tau & ::= \alpha\ c & \textit{type} \\
\alpha & ::= \texttt{@Thread} \mid \texttt{@Shared} \mid \texttt{@Context} & \textit{annotations} \\
 & & \\
E & ::= [] \mid E[x:\tau] & \textit{local type environment}
\end{array}
$$

Figure 1. Loci's syntax. c, d are class names, f, m are field and method names, and x, y, z are names of variables or parameters respectively, where $x \neq \textbf{this}$. For simplicity, we assume that names of classes, fields, methods and variables are unique. The special variable **ret** and **return** only appears in the dynamic syntax and semantics.

For a program P, define the class table CT as: $CT(c) = (\overline{F}, \overline{M}) \bullet CT(d)$ where $\texttt{class}\ c\ \texttt{extends}\ d\ \texttt{\{}\ \overline{F}\ \overline{M}\ \texttt{\}} \in P$ and $CT(\texttt{Object}) = (\epsilon, \epsilon)$. Also define $\epsilon(x) = \bot$. Now, lookup in combined tuples models lookup in inheritance hierarchies and lookups of non-existing elements return $\bot$ and $\bot(x) = \bot$.

We use a number of shorthands to access the class-table: $\mathsf{fields}(c) = \mathit{fst}(CT(c))$ and $\mathsf{method}(c) = \mathit{snd}(CT(c))$. Thus, $\mathsf{fields}(c.f)$ looks up the type of the (only, since we do not model overriding of fields) field f in the most specific supertype of c (including c). Furthermore define $\mathsf{mbody}(c.m)$ as $(\overline{x}, s\texttt{;return}\ y)$ and $\mathsf{mtype}(c.m)$ as $\overline{\tau} \to \tau$ when $\mathsf{methods}(c.m) = \tau\ m\texttt{(}\overline{\tau\ x}\texttt{)\{}\ s\texttt{;return}\ y\ \texttt{\}}$.

Lase, we also define a shorthand for retrieving then names of fields and methods of a class $\mathsf{names}(\overline{\tau}\ \overline{f}) = \overline{f}$ and $\mathsf{names}(\overline{\tau\ m\texttt{(}\cdots\texttt{)\{}\ \cdots\ \texttt{\}}}) = \overline{m}$.

Well-formed Programs By construction, all class hierarchies are rooted in `Object`. For simplicity, `Object` is an empty class in all P's. Furthermore, `Object` can be subclassed by classes with any thread-locality. We write $\cdots$ to denote an unimportant omission.

$$
\frac{\vdash cd \text{ for all } cd \in P}{\vdash P}\ (\textsc{wf-program}) \qquad \frac{}{\vdash \texttt{class Object \{ \}}}\ (\textsc{object})
$$

Modulo for `Object`, subclassing and overriding must preserve annotations. Overloading is not supported, nor is overriding fields.

$$
\frac{\begin{array}{c}\mathsf{fields}(d) = \overline{fd_2} \qquad \mathsf{methods}(d) = \overline{md_2} \\ \forall m \in \mathsf{names}(\overline{md_1}) \cap \mathsf{names}(\overline{md_2}).\ \mathsf{mtype}(c.m) = \mathsf{mtype}(d.m) \\ \mathsf{names}(\overline{fd_1}) \cap \mathsf{names}(\overline{fd_2}) = \emptyset \\ d \equiv \texttt{Object} \vee P(d) = \alpha_1\ \texttt{class}\ d\ \cdots \qquad \alpha_1 \vdash \overline{fd_1} \qquad \alpha_1\ c \vdash \overline{md_1}\end{array}}{\vdash \alpha_1\ \texttt{class}\ c\ \texttt{extends}\ d\ \texttt{\{}\ \overline{fd_1}\ \overline{md_1}\ \texttt{\}}}\ (\textsc{wf-class})
$$

(WF-FIELD) makes use of the viewpoint-adaptation operator $\oplus$ on annotations and types. This is similar to σ-substitution found in ownership types type systems. This makes the Loci type system treat a `@Context` variable as `@Thread` in a `@Thread` class, etc.

$$\frac{\vdash \alpha_1 \oplus \alpha_2\ c}{\alpha_1 \vdash \alpha_2\ c\ f}\ (\text{WF-FIELD}) \qquad \frac{\textbf{this} : \alpha\ c, \overline{x} : \alpha \oplus \overline{\tau} \vdash s; \textbf{return}\ y; E \qquad E \vdash y : \alpha \oplus \tau}{\alpha\ c \vdash \tau\ m(\overline{\tau}\ \overline{x})\{\ s; \textbf{return}\ y\ \}}\ (\text{WF-METHOD})$$

E is a standard local type environment that maps variables to types (Figure 1).

$$\frac{}{\vdash []}\ (\text{EMPTY-E}) \qquad \frac{\vdash E \qquad \vdash \tau \qquad x \notin dom(E)}{\vdash E[x : \tau]}\ (\text{WF-E})$$

Statements. The statements should be straightforward to follow for anyone familiar with Java. x, y, z are local variables and $x \neq \textbf{this}$.

$$\frac{E \vdash s_1; E_1 \qquad E_1 \vdash s_2; E_2}{E \vdash s_1; s_2; E_2}\ (\text{SEQUENCE}) \qquad \frac{\vdash E}{E \vdash \textbf{skip}; E}\ (\text{SKIP}) \qquad \frac{x : \tau \in E \qquad E \vdash y : \tau}{E \vdash x = y; E}\ (\text{ASSIGN})$$

$$\frac{\begin{array}{c}\vdash E \\ y : \alpha\ c \in E \qquad x : \tau \in E \\ \mathsf{fields}(c.f) = \tau' \qquad \vdash \alpha \oplus \tau' \leq \tau\end{array}}{E \vdash x = y.f; E}\ (\text{SELECT}) \qquad \frac{\begin{array}{c}y : \alpha\ c \in E \\ \mathsf{fields}(c.f) = \tau \\ E \vdash z : \alpha \oplus \tau\end{array}}{E \vdash y.f = z; E}\ (\text{UPDATE})$$

$$\frac{\begin{array}{c}\vdash E \qquad \vdash \tau \\ x \notin dom(E) \\ \textbf{this} : \alpha\ c \in E\end{array}}{E \vdash \tau\ x; E[x : \alpha \oplus \tau]}\ (\text{VAR-DECL})$$

(SELECT) and (UPDATE) apply $\oplus$ to the annotation on the target and the field to possibly expand `@Context`s. Note that (VAR-DECL) replaces `@Context` with the annotation of the current **this** (which may be `@Context`).

$$\frac{\begin{array}{c}\vdash E \\ x : \tau \in E \qquad \vdash \alpha\ c \leq \tau\end{array}}{E \vdash x = \textbf{new}\ \alpha\ c(); E}\ (\text{NEW}) \qquad \frac{\begin{array}{c}y : \alpha\ c \in E \qquad \mathsf{mtype}(c.m) = \overline{\tau} \to \tau' \\ E \vdash \overline{z} : \alpha \oplus \overline{\tau} \qquad x : \tau \in E \qquad \vdash \alpha \oplus \tau' \leq \tau\end{array}}{E \vdash x = y.m(\overline{z}); E}\ (\text{METHOD-CALL})$$

Similar to how Java deals with threads, the **start** operation only works on classes that have a 0-arity `run` method (denoted by ϵ parameter types). Finally, (VAR-SUB) allows subsumption for reading variables, but it not used in code.

$$\frac{\begin{array}{c}\mathsf{mtype}(c.\mathsf{run}) = \epsilon \to \_ \\ E \vdash x : \texttt{@Shared}\ d \qquad \vdash c \leq d\end{array}}{E \vdash x = \textbf{start}\ c(); E}\ (\text{FORK}) \qquad \frac{E \vdash y : \tau}{E \vdash \textbf{return}\ y : \tau; E}\ (\text{RETURN}) \qquad \frac{\begin{array}{c}\vdash E \\ x : \tau' \in E \qquad \vdash \tau' \leq \tau\end{array}}{E \vdash x : \tau}\ (\text{VAR-SUB})$$

$H ::= [] \mid H[\iota \mapsto (\rho\ c, F)]$	*heap*	$T ::= (S, \rho) \mid (\texttt{NPE}, \rho)$	*thread*
$F ::= [] \mid F[f \mapsto v]$	*fields*	$S ::= \epsilon \mid S\,\langle V, s\rangle$	*stack*
$v ::= \iota \mid \texttt{null}$	*value*	$V ::= [] \mid V[y \mapsto v]$	*stack frame*

Figure 2. Syntax for heaps, threads, stacks, frame, objects and values. For clarity, we split fields and stack frames into two different syntactic categories.

Subclassing, Types, and Subtypes. The following rules define subclass, subtype and well-formed type.

$$\frac{P(c) = \alpha\ \texttt{class}\ c\ \texttt{extends}\ d\ \cdots}{\vdash c \leq d}\ (\textsc{subclass-direct}) \qquad \frac{\vdash c \leq c' \qquad \vdash c' \leq d}{\vdash c \leq d}\ (\textsc{subclass-transitive}) \qquad \frac{\vdash c}{\vdash c \leq c}\ (\textsc{subclass-reflex})$$

$$\frac{P(c) = \alpha\ \texttt{class}\ c\ \cdots}{\vdash c}\ (\textsc{class}) \qquad \frac{\vdash c \qquad P(c) = \alpha_1\ \texttt{class}\ c\ \cdots \qquad \alpha = \alpha_1 \vee \alpha_1 = \texttt{@Context} \vee c = \texttt{Object}}{\vdash \alpha\ c}\ (\textsc{type}) \qquad \frac{\vdash \alpha\ c \qquad \vdash c \leq d}{\vdash \alpha\ c \leq \alpha\ d}\ (\textsc{subtype})$$

By (SUBTYPE), subtyping must preserve annotations. For brevity we write $\cdots$ to omit unimportant parts of code in P. Most importantly, though, `Object` may be use in both `@Shared`, `@Context` and `@Thread` types.

1.2 Dynamic Semantics

Loci's dynamic semantics is a small-step operational semantics. Syntax for heaps, fields, values, threads, stacks and stack frames can be found in Figure 2. A Loci configuration $(H; \overline{T})$ consists of a single heap H of locations mapped to objects tagged to denote to what heap(let) they belong to and a collection of threads. Each thread T has its own stack, plus a thread id denoted ρ. An instance of class c belonging to the thread ρ will have run-time type $\rho\ c$. We use ϱ to denote the shared heap. ρ and ϱ both belong to the syntactic category θ which is read as "heap id."

Thread-scheduling is modeled as a non-deterministic choice in (SCHEDULE). A configuration with a thread scheduled to run is denoted $(H; T; \overline{T})$. For convenience, we write $H(\iota.f) := v$ as a shorthand $H[\iota \mapsto (\theta\ c, V[f \mapsto v])]$ where $H(\iota) = (\theta\ c, V)$. We denote the look-up of a non-existent field $F(f) = \bot$ (where $\bot \neq v$), which can happen due to lazy creation of thread-local fields.

The initial configuration has the form

$$([\iota = (\rho\ \texttt{Object}, \epsilon)]; (\langle[\texttt{this} = \iota], s; \texttt{return}\ y\rangle, \rho); \epsilon)$$

i.e., there is only one thread on start-up, a single instance of `Object` on the heap, which the starting statements execute in.

There are two kinds of reduction rules. The first ($\rightsquigarrow$) is a "scheduling reduction" and takes a two-compartment configuration with $n+1$ idle threads to a three-compartment configuration with n idle threads and one active thread.

The second reduction ($\rightarrow$) is the regular reduction step and goes in the inverse direction of a scheduling reduction. Effectively, $(H; \epsilon)$ is a final configuration.

$$\frac{}{(H; \overline{T}, T, \overline{T'}) \rightsquigarrow (H; T; \overline{T}, \overline{T'})} \text{(SCHEDULE)}$$

The rules (FINISHED-THREAD) and (DEAD-THREAD) remove threads that are fully reduced (normally or abnormally) from the system. `NPE` is a special error configuration denoting a crashed thread due to a null-pointer dereference error.

$$\frac{}{(H; (\langle V, \textbf{return}\ x\rangle, \rho); \overline{T}) \rightarrow (H; \overline{T})} \text{(FINISHED-THREAD)} \qquad \frac{}{(H; (\texttt{NPE}, \rho); \overline{T}) \rightarrow (H; \overline{T})} \text{(DEAD-THREAD)}$$

We use the special variable **`ret`** to capture return values. The only assignment to **`ret`** is through a return which assigns the **`ret`** of the underlying stack frame.

$$\frac{F(y) = v \qquad T = (S\ \langle V'[\textbf{ret} \mapsto v], s'\rangle, \rho)}{(H; (S\ \langle V', s'\rangle, \langle V, \textbf{return}\ y\rangle, \rho); \overline{T}) \rightarrow (H; T, \overline{T})} \text{(RETURN)}$$

$$\frac{F(y) = v \qquad T = (S\ \langle V[x = v], s\rangle, \rho)}{(H; (S\ \langle V, x = y; s\rangle, \rho); \overline{T}) \rightarrow (H; T, \overline{T})} \text{(ASSIGN)}$$

$$\frac{T = (S\ \langle V[x \mapsto \texttt{null}], s\rangle, \rho)}{(H; (S\ \langle V, \tau\ x; s\rangle, \rho); \overline{T}) \rightarrow (H; T, \overline{T})} \text{(VAR-DECL)}$$

$$\frac{}{(H; (S\ \langle V, \textbf{skip}; s\rangle, \rho); \overline{T}) \rightarrow (H; (S\ \langle V, s\rangle, \rho), \overline{T})} \text{(SKIP)}$$

We model thread-local variables as zero or more variables indexed by the thread id ρ—a thread ρ accessing a `@Thread` field `f` returns the contents of the field $\texttt{f}_\rho$.

$$\frac{V(y) = \iota \quad H(\iota) = (\_\ c, F) \quad \mathsf{fields}(c.f) = \alpha\ d \quad \alpha \neq \texttt{@Thread} \quad F(f) = v}{(H; (S\ \langle V, x = y.f; s\rangle, \rho); \overline{T}) \rightarrow (H; (S\ \langle V[x = v], s\rangle, \rho), \overline{T})} \text{(SELECT-OTHER)}$$

$$\frac{V(y) = \iota \qquad H(\iota) = (\_\ c, F) \qquad \mathsf{fields}(c.f) = \texttt{@Thread}\ d \qquad F(f_\rho) = v}{(H; (S\ \langle V, x = y.f; s\rangle, \rho); \overline{T}) \rightarrow (H; (S\ \langle V[x = v], s\rangle, \rho), \overline{T})} \text{(SELECT-THREAD)}$$

$$\frac{V(y) = \iota \qquad H(\iota) = (\_\ c, F) \qquad \mathsf{fields}(c.f) = \texttt{@Thread}\ \_ \qquad F(f_\rho) = \bot}{(H; (S\ \langle V, x = y.f; s\rangle, \rho); \overline{T}) \rightarrow (H(\iota.f_\rho) := \texttt{null}; (S\ \langle V[x \mapsto \texttt{null}], s\rangle, \rho), \overline{T})} \text{(SELECT-FIRST-THREAD)}$$

The fields are created lazily on first read and are then given the value `null`. An alternative would be to create a copy for every thread in the system, but the above solution felt somewhat closer to the semantics of the `ThreadLocal` API, which calls `initialValue()` on the first read of a field by a particular thread.

(SELECT-NPE)
$$\frac{}{(H; (S\,\langle V[y \mapsto \texttt{null}], x = y.f; s\rangle, \rho); \overline{T}) \rightarrow (H; (\texttt{NPE}, \rho), \overline{T})}$$

Like reading, writing a `@Thread` field updates the copy of the field indexed by the current thread's id.

(UPDATE-OTHER)
$$\frac{V(y) = \iota \quad V(z) = v \quad H(\iota) = (\_\ c, \_) \quad \mathsf{fields}(c.f) = \alpha\ d \quad \alpha \neq \texttt{@Thread}}{(H; (S\,\langle V, y.f = z; s\rangle, \rho); \overline{T}) \rightarrow (H(\iota.f) := v; (S\,\langle V, s\rangle, \rho), \overline{T})}$$

(UPDATE-THREAD)
$$\frac{V(y) = \iota \qquad V(z) = v \qquad H(\iota) = (\_\ c, \_) \qquad \mathsf{fields}(c.f) = \texttt{@Thread}\ d}{(H; (S\,\langle V, y.f = z; s\rangle, \rho); \overline{T}) \rightarrow (H(\iota.f_\rho) := v; (S\,\langle V, s\rangle, \rho), \overline{T})}$$

(UPDATE-NPE)
$$\frac{}{(H; (S\,\langle V[y \mapsto \texttt{null}], y.f = z; s\rangle, \rho); \overline{T}) \rightarrow (H; (\texttt{NPE}, \rho), \overline{T})}$$

A new instance cannot be leaked on instantiation and can subsequently be placed either on the shared heap or in the current heaplet. This is decided by the annotation of the target variable for the instantiation.

(NEW-SHARED)
$$\frac{\iota \text{ is fresh} \qquad F = [f \mapsto \texttt{null} \mid \alpha\ f \in \mathsf{fields}(c) \wedge \alpha \neq \texttt{@Thread}] \qquad H' = H[\iota \mapsto (\varrho\ c, F)]}{(H; (S\,\langle V, x = \textbf{new}\ \texttt{@Shared}\ c\texttt{()}; s\rangle, \rho); \overline{T}) \rightarrow (H'; (S\,\langle V[x \mapsto \iota], s\rangle, \rho), \overline{T})}$$

(NEW-THREAD)
$$\frac{\iota \text{ is fresh} \qquad F = [f \mapsto \texttt{null} \mid \alpha\ f \in \mathsf{fields}(c) \wedge \alpha \neq \texttt{@Thread}] \qquad H' = H[\iota \mapsto (\rho\ c, F)]}{(H; (S\,\langle V, x = \textbf{new}\ \texttt{@Thread}\ c\texttt{()}; s\rangle, \rho); \overline{T}) \rightarrow (H'; (S\,\langle V[x \mapsto \iota], s\rangle, \rho), \overline{T})}$$

(NEW-CONTEXT)
$$\frac{H(V(\textbf{this})) = (\theta\ \_, \_) \qquad \iota \text{ is fresh} \qquad F = [f \mapsto \texttt{null} \mid \alpha\ f \in \mathsf{fields}(c) \wedge \alpha \neq \texttt{@Thread}] \qquad H' = H[\iota \mapsto (\theta\ c, F)]}{(H; (S\,\langle V, x = \textbf{new}\ \texttt{@Context}\ c\texttt{()}; s\rangle, \rho); \overline{T}) \rightarrow (H'; (S\,\langle V[x \mapsto \iota], s\rangle, \rho), \overline{T})}$$

If the target variable is `@Context`-annotated, the class is stored in the same heap or heaplet as the current **`this`**.

$$\frac{\begin{array}{c}V(y) = \iota \qquad H(\iota) = (\_\ c, \_) \qquad \mathsf{mbody}(c.m) = (\overline{x'}, s'\texttt{;return}\ y') \\ V(\overline{z}) = \overline{v} \qquad V' = \texttt{this} \mapsto \iota, \overline{x'} \mapsto \overline{v} \qquad S' = S\,\langle V, x = \texttt{ret}; s\rangle\end{array}}{(H; (S\,\langle V, x = y.m(\overline{z}); s\rangle, \rho); \overline{T}) \to (H; (S', \langle V', s'\texttt{;return}\ y'\rangle, \rho), \overline{T})}\ (\textsc{method-call})$$

An invocation $x = y.m()$ is rewritten into $x = \texttt{ret}$ and the method's body is executed on a new stack frame eventually assigning **ret** as the result of a return.

$$\frac{}{(H; (S\,\langle V[y \mapsto \texttt{null}], x = y.m(\overline{z}); s\rangle, \rho); \overline{T}) \to (H; (\texttt{NPE}, \rho), \overline{T})}\ (\textsc{method-call-npe})$$

For brevity, null-pointer exceptions kill the entire thread rather than propagate an error through the execution. The semantics is effectively the same.

$$\frac{\begin{array}{c}\iota, \rho' \text{ are fresh} \\ F = [f \mapsto \texttt{null} \mid \alpha\ f \in \mathsf{fields}(c) \wedge \alpha \neq \texttt{@Thread}] \qquad H' = H[\iota \mapsto (\varrho\ c, F)] \\ \mathsf{mbody}(c.\mathsf{run}) = (\epsilon, s'\texttt{;return}\ y') \qquad T = (\langle[\texttt{this} \mapsto \iota], s'\texttt{;return}\ y'\rangle, \rho')\end{array}}{(H; (S\,\langle V, x = \texttt{start}\ c(); s\rangle, \rho); \overline{T}) \to (H'; (S\,\langle V[x \mapsto \iota], s\rangle, \rho), \overline{T}, T)}\ (\textsc{fork})$$

The (FORK) operation adds a thread to the system and is a simplified union of Java's `new` and `start`. The new thread object is created on the shared area, forcing its thread-local data to be stored either on the stack of the `run` method, or in a thread-local field.

2 Well-Formedness Rules

We now present the rules for well-formed configurations, stacks and heaps. The store type Γ is defined $\Gamma ::= [] \mid \Gamma[\iota : t]$ where t is a run-time type on the form $\theta\ c$ where $\theta ::= \rho \mid \varrho$. For simplicity, we assume that names and ids of threads are unique.

Well-formed Store Type The rules for well-formed store type follow standard form, except that we distinguish between run-time and compile-time types. A run-time type in the store-type consists of a "heap id" θ denoting the heap to which an object belongs. Notably, `@Thread` classes and `@Shared` classes always live on some thread-local heap respective the shared heap. `@Context` classes always live on the same heap as the current **this**.

$$\frac{}{\vdash []}\ (\Gamma\textsc{-empty}) \qquad \frac{\vdash \Gamma \qquad (\vdash \texttt{@Context}\ c \vee \vdash \texttt{@Thread}\ c)}{\vdash \Gamma[\iota : \rho\ c]}\ (\textsc{wf-}\Gamma 1) \qquad \frac{\vdash \Gamma \qquad (\vdash \texttt{@Context}\ c \vee \vdash \texttt{@Shared}\ c)}{\vdash \Gamma[\iota : \varrho\ c]}\ (\textsc{wf-}\Gamma 2)$$

We use the function rtt to take static type to a run-time type. The function takes three parameters—the static type, the heap id of the current **this** and the current thread id.

$$\mathsf{rtt}(\alpha\ c, \theta, \rho) = \begin{cases} \theta\ c & \text{if } \alpha = \texttt{@Context} \\ \rho\ c & \text{if } \alpha = \texttt{@Thread} \\ \varrho\ c & \text{if } \alpha = \texttt{@Shared} \end{cases}$$

Well-formed Configuration An unscheduled configuration is well-formed if any scheduled configuration derived from it is well-formed. A scheduled configuration is well-formed if the heap and top-most frame of the scheduled thread are well-formed, plus the remainder of all unscheduled threads are well-formed.

$$\frac{\Gamma \vdash (H; T; \overline{T})}{\Gamma \vdash (H; T, \overline{T})}\ (\textsc{wf-unscheduled}) \qquad \frac{\Gamma \vdash H \qquad \Gamma \vdash (H; \overline{T}) \qquad \exists\ \overline{E}\ s.t.\ \Gamma; \overline{E}; \rho \vdash \overline{\langle V, s\rangle}}{\Gamma \vdash (H; (\overline{\langle V, s\rangle}, \rho); \overline{T})}\ (\textsc{wf-scheduled}) \qquad \frac{\Gamma \vdash (H; \overline{T})}{\Gamma \vdash (H; (\texttt{NPE}, \rho); \overline{T})}\ (\textsc{wf-npe})$$

Well-formed Stack Frame The key element in (WF-STACK-FRAME) is the extraction of the heap id of the current **this**, which is used in (WF-FRAME) to make sure that `@Context`-typed variables live on the correct heap.

$$\frac{\Gamma(V(\textbf{this})) = \theta\ c \qquad E \vdash s; E' \qquad \Gamma, E; \rho; \theta \vdash V}{\Gamma; E; \rho \vdash \langle V, s\rangle}\ (\textsc{wf-stack-frame})$$

Notably, there is always a **this** on the stack frame. The initial configuration has one, and (METHOD-CALL) always puts a **this** on the stack.

Well-formed Frame We make use of a run-time type function that given a set of heap ids, returns the correct heap id for an annotation.

$$\frac{\vdash \Gamma}{\Gamma; []; \rho; \theta \vdash []}\ (\textsc{frame-empty}) \qquad \frac{\Gamma \vdash v : \mathsf{rtt}(\tau, \theta, \rho) \qquad \Gamma; E; \rho; \theta \vdash V}{\Gamma; E[y : \tau]; \rho; \theta \vdash V[y \mapsto v]}\ (\textsc{wf-frame})$$

Well-formed Heap The rules for well-formed heap are mostly standard. The empty heap is well-formed, and adding a well-formed object to a well-formed heap results in a well-formed heap.

$$\frac{\vdash \Gamma}{\Gamma \vdash []}\ (\textsc{wf-heap-empty}) \qquad \frac{\Gamma \vdash H \qquad \Gamma(\iota) = \theta\ c \qquad E = \mathsf{fields}(c) \qquad \Gamma; E; \theta \vdash F}{\Gamma \vdash H[\iota \mapsto (\theta\ c, F)]}\ (\textsc{wf-heap})$$

Well-formed Fields The rules for well-formed fields are standard, modulo that (FIELD-THREADS) introduces zero or more thread-local fields for a single field declaration.

$$\frac{\vdash \Gamma}{\Gamma; []; \theta \vdash []}\ (\text{FIELDS-EMPTY}) \qquad \frac{\Gamma \vdash \overline{v} : \overline{\rho}\ c \qquad \Gamma; E; \theta \vdash F}{\Gamma; E[f : \texttt{@Thread}\ c]; \theta \vdash F[\overline{f_\rho \mapsto v}]}\ (\text{FIELD-THREAD})$$

$$\frac{\alpha \neq \texttt{@Thread} \qquad \Gamma \vdash v : \mathsf{rtt}(\alpha\ c, \theta, \bot) \qquad \Gamma; E; \theta \vdash F}{\Gamma; E[f : \alpha\ c]; \theta \vdash F[f \mapsto v]}\ (\text{FIELD-OTHER})$$

We use $\bot$ as third argument to rtt in (FIELD-OTHER), since it will never be used as $\alpha \neq \texttt{@Thread}$.

Well-formed Values The well-formed values rules map static types to run-time types. As usual, `null` can have any type.

$$\frac{\vdash c}{\Gamma \vdash \texttt{null} : \theta\ c}\ (\text{TYPE-NULL}) \qquad \frac{\Gamma(\iota) = \theta\ d \qquad \vdash d \leq c}{\Gamma \vdash \iota : \theta\ c}\ (\text{TYPE-VALUE}) \qquad \frac{\vdash \tau \leq \tau'}{\vdash \mathsf{rtt}(\tau, \theta, \rho) \leq \mathsf{rtt}(\tau', \theta, \rho)}\ (\text{SUBTYPE})$$

$$\frac{\vdash \alpha\ c \qquad \alpha = \texttt{@Thread}}{\vdash \mathsf{rtt}(\alpha\ c, \_, \rho) = \mathsf{rtt}(\alpha\ c, \_, \rho)}\ (\text{EQ-TYPE-THREAD}) \qquad \frac{\vdash \alpha\ c \qquad \alpha = \texttt{@Shared}}{\vdash \mathsf{rtt}(\alpha\ c, \_, \_) = \mathsf{rtt}(\alpha\ c, \_, \_)}\ (\text{EQ-TYPE-SHARED})$$

$$\frac{\vdash \alpha\ c \qquad \alpha = \texttt{@Context}}{\vdash \mathsf{rtt}(\alpha\ c, \theta, \_) = \mathsf{rtt}(\alpha\ c, \theta, \_)}\ (\text{EQ-TYPE-CONTEXT}) \qquad \frac{\vdash \texttt{@Context}\ c}{\vdash \mathsf{rtt}(\texttt{@Context}\ c, \rho, \_) = \mathsf{rtt}(\texttt{@Thread}\ c, \_, \rho)}\ (\text{EQ-TYPE-CT})$$

$$\frac{\vdash \texttt{@Context}\ c}{\vdash \mathsf{rtt}(\texttt{@Context}\ c, \varrho, \_) = \mathsf{rtt}(\texttt{@Shared}\ c, \_, \_)}\ (\text{EQ-TYPE-CS})$$

3 Proofs

3.1 Helper Lemmas

We use a couple of standard lemmas defined below.

Lemma 1. *Extension.*

1. *If* $E \vdash s; E'$, $x \notin \mathrm{dom}(E')$, *and* $\vdash \tau$, *then* $E[x{:}\tau] \vdash s; E'[x{:}\tau]$.
2. *If* $\Gamma \vdash H$ *and* $\Gamma' \supseteq \Gamma$, *then* $\Gamma' \vdash H$.

3. *If* $\Gamma; E; \rho; \theta \vdash V$ *and* $\Gamma' \supseteq \Gamma$, *then* $\Gamma'; E; \rho; \theta \vdash V$.
4. *If* $\Gamma \vdash (H; \overline{T})$, $\Gamma' \supseteq \Gamma$ *and* $\Gamma' \vdash H'$ *then* $\Gamma \vdash (H'; \overline{T})$.

Proof. Follows by straightforward derivation on the well-formedness rules. □

Lemma 2. *Well-formed Construction.*

1. *If* $E \vdash s; E'$, *then* $\vdash E$ *and* $\vdash E'$.
2. *If* $E \vdash y : \tau$, *then* $\vdash E$ *and* $\vdash \tau$.
3. *If* $\vdash \tau \leq \tau'$, *then* $\vdash \tau$ *and* $\vdash \tau'$.
4. *If* $\Gamma \vdash H$, *then* $\vdash \Gamma$.

Proof. Follows by straightforward derivation on the well-formedness rules. □

Lemma 3. *Lookup.*

1. *If* $\Gamma; E; \rho; \theta \vdash V$ *and* $E \vdash y : \tau$, *then* $\Gamma \vdash V(y) : \mathsf{rtt}(\tau, \theta, \rho)$.
2. *If* $\Gamma; E; \rho; \theta \vdash V$ *and* $y : \tau \in E$, *then* $\Gamma \vdash V(y) : \mathsf{rtt}(\tau, \theta, \rho)$.

Proof. Follows by straightforward derivation on the well-formedness rules. □

3.2 Invariants

Informally, Loci enforces the following property:

> *A thread* ρ *can only access objects in heaplet* ρ *or on the shared heap* ϱ.

We formulate this in two theorems, the first of which says that pointers in variables on a stack frame in a thread ρ either point to objects in ρ or in ϱ, and the second that evaluating a field access in thread ρ results in a pointer to either an object in ρ or in ϱ (or is a `null`-pointer).

Theorem 1. *Local variables point into shared heap or current heaplet. If* $\Gamma; E \vdash (H; (S\ \langle V, s\rangle, \rho); \overline{T})$, *then* $\forall \iota \in rng(V).\ \mathsf{tid}(H, \iota) \in \{\varrho, \rho\}$.

Proof. Follows by straightforward induction on s. The non-trivial cases are $s = x = y.f; s'$, which is covered by Theorem 2 and $s = x = \texttt{new}\ \alpha\ c\texttt{()}; s'$.

NB. Theorems 1 and 2 are proven co-inductively.

Trivially, the initial stack-frame satisfies the theorem as $\texttt{this} \mapsto \iota$ and $H = [\iota \mapsto \rho\ \texttt{Object}]$. clearly is empty. We now prove that all statements that modify local variables satisfy the theorem under the hypothesis that all existing stack variables satisfy the theorem.

$s'; s''$	Follows immediately from induction hypothesis.
$\texttt{skip}; s'$	Immediate since no variables are changed.
$x = y.f; s'$	Follows immediately from Theorem 2.
$x = y; s'$	Follows immediately from induction hypothesis.
$y.f = z; s'$	Immediate since no variables are changed.
$\tau\ x; s'$	Immediate since x is initialised with $\texttt{null} \neq \iota$.

$x = \texttt{new}\ \alpha\ c\texttt{()}; s'$ By (NEW-$*$), a newly instantated object ι will have thread id θ s.t. $\theta \in \{\rho, \varrho, \theta'\}$. The first two cases are immediate—ρ is the current thread id and ϱ is id of the global shared heap. For the third case, $\theta' = \mathsf{tid}(H(V(\texttt{this})))$, which is in $\{\varrho, \rho\}$ from the induction hypothesis.

$x = y.m(\overline{z}); s'$ Follows immediately from the induction hypothesis since only local variables can be returned.

$x = \texttt{start}\ c\texttt{()}; s'$ Follows immediately from (FORK) as a forked object is always shared.

$\texttt{return}\ y$ Follows immediately from the induction hypothesis as all frames on a stack belong to the same thread.

□

Theorem 2. *Field accesses yield pointers to shared heap or current heaplet. Let s be a field access $x = y.f$. If $\Gamma; E \vdash (H; (S\,\langle V, s; s'\rangle, \rho); \overline{T})$, $(H; (S\,\langle V, s; s'\rangle, \rho); \overline{T}) \rightarrow (H'; (S'\,\langle V', s'\rangle, \rho), \overline{T'})$, and $V'(x) = \iota$, then $\mathsf{tid}(H', \iota) \in \{\varrho, \rho\}$.*

Proof. The proof is by derivation on $\Gamma; E \vdash (H; (S\,\langle V, x = y.f\texttt{;}s'\rangle, \rho); \overline{T})$. The key points are the threading of ρ through the derivation, and rules for well-formed values. Note that $H' = H$ by (SELECT). For brevity, we omit facts in our derivations that are never relied on subsequently.

1. By (SCHEDULED), (a) $\Gamma \vdash H$ and (b) $\Gamma; E; \rho \vdash \langle V, x = y.f; s'\rangle$ for some E.
2. By 1.b) and (WF-STACK-FRAME), (a) $E \vdash x = y.f; s'; E'$ and (b) $\Gamma; E; \rho; \theta \vdash V$ where $\theta = \Gamma(V(\texttt{this}))$. By Theorem 1, (c) $\theta \in \{\varrho, \rho\}$.
3. By 2.a) and (SELECT) via (SEQUENCE), (a) $\mathsf{fields}(c.f) = \tau$ where $y : \alpha\ c \in E$.
4. By 2.b, 3.a) and (WF-FRAME), (a) $V(y) = \iota$ s.t. $\Gamma \vdash \iota : t$ s.t. $t = \mathsf{rtt}(\alpha\ c, \theta, \rho)$. (N.B. by (SELECT), $V(y) \neq \texttt{null}$ to be able to take the step.)
5. By 4.a) and (TYPE-VALUE), $\Gamma(\iota) = \theta'\ d.$ s.t. $\vdash d \leq c$.
6. By (SELECT-$*$), $H(\iota) = (d\ \theta', F)$ for some F.
7. By 1.a, 5., 6.), and (WF-HEAP), $\Gamma; \mathsf{fields}(d); \theta' \vdash F$.
8. Let $\tau = \alpha'\ c'$. By (SELECT-$*$), there are two cases, $\alpha' = \texttt{@Thread}$, $\alpha' \neq \texttt{@Thread}$. We continue in this order disregarding the cases when the field is `null`, i.e., v below is some ι.
   - By (SELECT-THREAD), $F(f_\rho) = v$. By (FIELD-THREAD), $\Gamma \vdash v : \rho\ c'$. Thus, by 1.a) and (WF-HEAP), $\mathsf{tid}(H, v) = \rho$.
   - By (SELECT-OTHER), $F(f) = v$. By (FIELD-OTHER), $\Gamma \vdash v : \theta''\ c'$ s.t. $\theta'' = \mathsf{rtt}(\alpha'\ c', \theta', \bot)$. As $\alpha \neq \texttt{@Thread}$, $\theta'' \in \{\theta', \varrho\}$. By 1.c), 4.a), 5.) and def. of rtt, $\theta' \in \{\varrho, \rho\}$. Thus, by 1.a) and (WF-HEAP), $\mathsf{tid}(H, v) \in \{\varrho, \rho\}$.

□

Thread-Locality Corollary Based on the theorems above, we formulate a "thread-locality corollary" that says that we can kill any thread and garbage collect any object belonging to the thread, without affecting the program. To this end, Figure 3 defines an "garbage collection" operation $|_\rho$ on H.

$$
\begin{array}{l|l}
[]|_\rho = [] & F[f_\rho \mapsto v]|_\rho = F|_\rho \\
H[\iota \mapsto (\rho\ c, F))]|_\rho = H|_\rho & F[f \mapsto v]|_\rho = F|_\rho[f \mapsto v] \\
H[\iota \mapsto (\rho'\ c, F))]|_\rho = H|_\rho[\iota \mapsto (\rho'\ c, F|_\rho)] & F[f_{\rho'} \mapsto v]|_\rho = F|_\rho[f_{\rho'} \mapsto v] \quad \rho \neq \rho'
\end{array}
$$

Figure 3. Garbage collection operator $|_\rho$ that removes all objects belonging to thread ρ, along with thread-local fields pointing to any such object.

Corollary 1. *Thread-Locality. If* $\Gamma \vdash (H; (S, \rho), T, \overline{T})$ *and* $(H; T; (S, \rho), \overline{T}) \rightarrow (H'; T'; (S, \rho), \overline{T})$*, then* $\Gamma' \vdash (H|_\rho; \overline{T})$ *and* $(H|_\rho; T; \overline{T}) \rightarrow (H|'_\rho; T'; \overline{T})$ *for some* $\Gamma' \subseteq \Gamma$*.*

Proof. Proof by negation-as-failure using the results of Theorems 1 and 2. The only way in which the execution could be altered by killing thread ρ is if a local variable on the top-most stack frame in T points to an object in ρ, or if a field access in T could result in a ρ-owned object. That this cannot be the case follows directly from Theorems 1 and 2. □

3.3 Type Soundness

We prove type soundness in the standard fashion of progress plus preservation [1]. In this context, preservation means that reduction does not invalidate the store typing.

Theorem 3. *Preservation.*

1. *If* $\Gamma \vdash (H; T; \overline{T})$*, and* $(H; T; \overline{T}) \rightarrow (H'; \overline{T})$*, then there exists a* Γ' *s.t.* $\Gamma' \vdash (H'; \overline{T})$*.*
2. *If* $\Gamma \vdash (H; T; \overline{T})$*, and* $(H; T; \overline{T}) \rightarrow (H'; T', \overline{T})$*, then there exists a* Γ' *s.t.* $\Gamma' \vdash (H'; T', \overline{T})$*.*

Proof. The proof of the first case is immediate from (FINISHED-THREAD) and (DEAD-THREAD), which are the only steps that remove threads. The proof of the second case is straightforward by structural induction on s when $T = (S\langle V, s\rangle)$. There are no surprising cases.

return y
1. By (WF-SCHEDULED),
   (a) $\Gamma \vdash H$,
   (b) $\Gamma; \underline{E}; \rho \vdash \langle V, \textbf{return}\ y\rangle$ for some E,
   (c) $\Gamma; \overline{E}; \rho \vdash S$ for some $\overline{E}$, and
   (d) $\Gamma \vdash (H, \overline{T})$
2. By 1.b) and (WF-STACK-FRAME),
   (a) $E \vdash \textbf{return}\ y; E'$ and
   (b) $\Gamma; E; \rho; \theta \vdash V$ s.t.
   (c) $\theta\ \_ = \Gamma(V(\textbf{this}))$.
3. By 2.a) and (RETURN), $E \vdash y : \tau$.
4. By 2.b), 3.) and Lemma 3 (Lookup),

(a) $\Gamma \vdash v : t$ where
(b) $V(y) = v$, and
(c) $t = \mathsf{rtt}(\tau, \theta, \rho)$.

5. By induction hypothesis, $S = S'\langle V', s' \rangle$, thus
   (a) $\Gamma; \underline{E_1}; \rho \vdash \langle V', s' \rangle$ and
   (b) $\Gamma; \overline{E'}; \rho \vdash S'$ for
   (c) $\overline{E} = E_1, \overline{E'}$.
6. By 1.c), 5.) and (WF-STACK-FRAME),
   (a) $E_1 \vdash x = \mathbf{ret}; s'; E_2$ and
   (b) $\Gamma; E_1; \rho; \theta' \vdash V'$ where
   (c) $\overline{E} = \overline{E'}, E_1$ and
   (d) $\theta' \_ = \Gamma(V'(\mathbf{this}))$.
7. By 6.a) and (SEQUENCE),
   (a) $E_1 \vdash x = \mathbf{ret}; E_3$ and
   (b) $E_3 \vdash s'; E_2$.
8. By 7.a) and (ASSIGN),
   (a) $E_1 \vdash \mathbf{ret} : \alpha\ c$ and
   (b) $x : \alpha\ c \in E$.
9. By (METHOD-CALL), the method call that was rewritten into $x = \mathbf{ret}$, had a return type τ_r s.t. $\vdash \alpha_r \oplus \tau_r \leq \alpha\ c$. By (WF-METHOD), $\alpha_t \oplus \tau \leq \tau_r$ where α_r is the annotation on the receiver of the method call and α_t is the annotation of the current $\mathbf{this}$ in E. Thus, if $\tau = \alpha_y\ d$, then $\vdash d \leq c$ by (SUBCLASS-$*$). Notably, if $\alpha = \texttt{@Context}$, then $\alpha_y = \alpha_r = \alpha_t = \texttt{@Context}$ by the definition of $\oplus$. Thus, $\theta = \theta'$ and by 4.a,c), $\Gamma \vdash v : \mathsf{rtt}(\alpha\ c, \theta', \rho)$ If $\tau = \texttt{@Shared}\ d$ (or $\tau = \texttt{@Thread}\ d$), then $\alpha = \texttt{@Shared}$ (or $\texttt{@Thread}$) by def. of $\oplus$ and trivially, $\Gamma \vdash v : \mathsf{rtt}(\tau, \theta', \rho)$ since θ' is never "used" and an $\vdash d \leq c$, $\Gamma \vdash v : \mathsf{rtt}(\alpha\ c, \theta', \rho)$.
10. By 6.b), 9.) and (WF-FRAME), $\Gamma; E_1; \rho; \theta' \vdash V'[\mathbf{ret} \mapsto v]$
11. By 6.a,c,d), 10.). and (WF-STACK-FRAME), $\Gamma; E_1; \rho \vdash \langle V'[\mathbf{ret} \mapsto v], x = \mathbf{ret}; s' \rangle$.
12. By 1.a), 5.b,c), 11.) and (WF-SCHEDULED), $\Gamma \vdash (H; S\langle V'[\mathbf{ret} \mapsto v], x = \mathbf{ret}; s' \rangle; \overline{T})$.
13. By 12.) and (WF-UNSCHEDULED), $\Gamma \vdash (H; S\langle V'[\mathbf{ret} \mapsto v], x = \mathbf{ret}; s' \rangle; \overline{T})$.

$s'; s''$ Follows immediately by the induction hypothesis.

$\mathbf{skip}; s'$ Immediate.

$x = y.f; s'$

1. By (WF-SCHEDULED),
   (a) $\Gamma \vdash H$,
   (b) $\Gamma; \underline{E}; \rho \vdash \langle V, x = y.f; s' \rangle$ for some E,
   (c) $\Gamma; \overline{E}; \rho \vdash S$ for some $\overline{E}$, and
   (d) $\Gamma \vdash (H, \overline{T})$
2. By 1.b) and (WF-STACK-FRAME),
   (a) $E \vdash x = y.f; s'; E$ and
   (b) $\Gamma; E; \rho; \theta \vdash V$ where
   (c) $\theta \_ = \Gamma(V(\mathbf{this}))$
3. By 2.a) and (SEQUENCE),
   (a) $E \vdash x = y.f; E$ and

(b) $E \vdash s'; E'$.

4. By 3.a) and (SELECT),
   (a) $y : \alpha\ c \in E$,
   (b) $x : \tau \in E$,
   (c) $\mathsf{fields}(c.f) = \tau'$, and
   (d) $\vdash \alpha \oplus \tau' \leq \tau$.
5. By 2.b), 4.a) and Lemma 3 (Lookup),
   (a) $\Gamma \vdash v : \theta'\ \_$ where
   (b) $V(y) = v$, and
   (c) $\theta'\ \_ = \mathsf{rtt}(\alpha\ c, \theta, \rho)$.
6. If $v = \texttt{null}$, then the theorem is trivially satisfied by (SELECT-NPE)—by 1.d) and (WF-NPE) $\Gamma \vdash (H; (\texttt{NPE}, \rho); \overline{T})$. Otherwise, evaluation can proceed according to (SELECT-OTHER), (SELECT-THREAD) and (SELECT-THREAD-FIRST). We prove the first and omit the latter because they are similar and updating a field with `null` is covered by (UPDATE). Thus, by (SELECT-OTHER), $H(v) = (\theta'\ d, F)$ and $F(f) = v'$ s.t. $\vdash d \leq c$.
7. By 1.a), 4.c), 6.) and (WF-HEAP), $\Gamma; \mathsf{fields}(d); \theta' \vdash F$.
8. By 6.), 7.) and (FIELD-OTHER), $\Gamma \vdash v' : \mathsf{rtt}(\tau', \theta', \bot)$.
9. If $\tau' = \texttt{@Context}\ c'$. By 5.a) and def. of rtt, $\alpha = \texttt{@Context}$ implies $\theta = \theta'$. Thus, by 8.), $\Gamma \vdash v' : \mathsf{rtt}(\tau', \theta, \bot)$.
10. If $\tau' = \texttt{@Shared}\ c'$. By 5.a), 8.) and def. of rtt, $\Gamma \vdash v' : \mathsf{rtt}(\tau', \theta, \bot)$.
11. $\tau = \_\ c''$. By 4.d) and (SUBCLASS), $\vdash c' \leq c''$.
12. By 9.), 10.), 11.) and (TYPE-VALUE), $\Gamma \vdash v' : \mathsf{rtt}(\tau, \theta, \bot)$.
13. By 2.b), 12.) and (WF-FRAME), $\Gamma; E : \rho; \theta \vdash V[x \mapsto v']$.
14. By 2.c), 3.b), 13.) and (WF-STACK-FRAME), $\Gamma; E : \rho \vdash \langle V[x \mapsto v'], s' \rangle$.
15. By 1.a,c,d), 14.) and (WF-SCHEDULED), $\Gamma \vdash (H; (S\langle V[x \mapsto v'], s' \rangle, \rho); \overline{T})$.
16. By 15.) and (WF-UNSCHEDULED), $\Gamma \vdash (H; (S\langle V[x \mapsto v'], s' \rangle, \rho), \overline{T})$.

$x = y; s'$

1. By (WF-SCHEDULED),
   (a) $\Gamma \vdash H$,
   (b) $\Gamma; E; \rho \vdash \langle V, x = y; s' \rangle$ for some E,
   (c) $\Gamma; \overline{E}; \rho \vdash S$ for some $\overline{E}$, and
   (d) $\Gamma \vdash (H, \overline{T})$
2. By 1.b) and (WF-STACK-FRAME),
   (a) $E \vdash x = y; s'; E$ and
   (b) $\Gamma; E; \rho; \theta \vdash V$ where
   (c) $\theta\ \_ = \Gamma(V(\textbf{this}))$.
3. By 2.a) and (SEQUENCE),
   (a) $E \vdash x = y; E$ and
   (b) $E \vdash s'; E'$.
4. By 3.a) and (ASSIGN),
   (a) $x : \tau \in E$, and
   (b) $E \vdash y : \tau$.
5. By 4.b) and (VAR-SUB),
   (a) $y : \tau' \in E$ and
   (b) $\vdash \tau' \leq \tau$.

6. By (ASSIGN), $V(y) = v$. By 2.b), 5.a) and Lemma 3 (Lookup), $\Gamma \vdash v : \mathsf{rtt}(\tau', \theta, \rho)$.
7. By 5.b), 6.), and (SUBTYPE), $\Gamma \vdash v : \mathsf{rtt}(\tau, \theta, \rho)$.
8. By 2.b), 4.a), 7.) and (WF-FRAME), $\Gamma; E; \rho; \theta \vdash V[x \mapsto v]$.
9. By 3.b), 8.) and (WF-STACK-FRAME), $\Gamma; E; \rho \vdash \langle V[x \mapsto v], s' \rangle$.
10. By 1.a,c,d), 9.) and (WF-SCHEDULED), $\Gamma \vdash (H; (S\langle V[x \mapsto v], s' \rangle; \overline{T})$.
11. By 10.) and (WF-UNSCHEDULED), $\Gamma \vdash (H; (S\langle V[x \mapsto v], s' \rangle, \overline{T})$.

$y.f = z; s'$

1. By (WF-SCHEDULED),
   (a) $\Gamma \vdash H$,
   (b) $\Gamma; \underline{E}; \rho \vdash \langle V, y.f = z; s' \rangle$ for some E,
   (c) $\Gamma; \overline{E}; \rho \vdash S$ for some $\overline{E}$, and
   (d) $\Gamma \vdash (H, \overline{T})$
2. By 1.b) and (WF-STACK-FRAME),
   (a) $\Gamma; E; \rho; \theta \vdash V$ and
   (b) $E \vdash y.f = z; s'; E$ where
   (c) $\theta \_ = \Gamma(V(\mathbf{this}))$.
3. By 2.b) and (SEQUENCE),
   (a) $E \vdash y.f = z; E$ and
   (b) $E \vdash s'; E'$.
4. By 3.a) and (UPDATE),
   (a) $y : \alpha\ c \in E$,
   (b) $\mathsf{fields}(c.f) = \tau$, and
   (c) $E \vdash z : \alpha \oplus \tau$.
5. By induction hypothesis, either $V(y) = \mathtt{null}$ or $V(y) = \iota$. The former case proceeds by (UPDATE-NPE), which trivially satisfies the theorem by 1.a,d) and (WF-NPE)—$\Gamma \vdash (H; (\mathtt{NPE}, \rho), \overline{T})$. We continue with the more interesting case, $V(y) = \iota$. For this case, the evaluation can proceed by either (UPDATE-OTHER), (UPDATE-THREAD). We omit the second because it is a copy-and-patch proof of the former (similar to when $\alpha' = \mathtt{@Shared}$). Thus, by (UPDATE-OTHER),
   (a) $V(z) = v$,
   (b) $H(\iota) = (\theta'\ d, F)$,
   (c) $\mathsf{fields}(d.f) = \alpha'\ c'$.
6. By 2.a), 4.a), 5.), and Lemma 3 (Lookup), $\Gamma \vdash \iota : \mathsf{rtt}(\alpha\ c, \theta, \rho)$.
7. By 5.), 6.) and (TYPE-VALUE), $\Gamma(\iota) = \theta'\ d$ s.t. $\vdash d \leq c$.
8. By 1.a), 7.) and (WF-HEAP),
   (a) $\Gamma \vdash H'$ for $H = H'[\iota \mapsto \_]$.
   (b) $\Gamma; \mathsf{fields}(d); \theta' \vdash F$.
9. By 4.c) and (VAR-SUB),
   (a) $z : \tau' \in E$, and
   (b) $\vdash \tau' \leq \alpha \oplus \tau$.
10. By 2.a), 5.a), 9.a), and Lemma 3 (Lookup), $\Gamma \vdash v : \mathsf{rtt}(\tau', \theta, \rho)$.
11. $\tau' = \alpha''\ c''$. If $\alpha' = \mathtt{@Shared}$, then $\alpha'' = \mathtt{@Shared}$ by def. of $\oplus$. Then, by 10.) and (TYPE-EQ-SHARED), $\Gamma \vdash v : \mathsf{rtt}(\tau', \theta', \rho)$.

12. $\tau' = \alpha''\ c''$. If $\alpha' = \texttt{@Context}$, then $\alpha'' = \alpha$ by def. of $\oplus$. There are three cases: If $\alpha = \texttt{@Context}$, then $\theta = \theta'$ by 6.). Thus, $\Gamma \vdash v : \mathsf{rtt}(\tau', \theta', \rho)$ by 11.) If $\alpha = \texttt{@Shared}$, then by (TYPE-EQ-SHARED) $\mathsf{rtt}(\tau', \theta, \rho) = \mathsf{rtt}(\tau, \theta', \rho)$. If $\alpha = \texttt{@Thread}$, then by (TYPE-EQ-THREAD) $\mathsf{rtt}(\tau', \theta, \rho) = \mathsf{rtt}(\tau, \theta', \rho)$.
13. By 5.c), 8.), 12.) and (WF-FRAME), $\Gamma; \mathsf{fields}(d); \theta' \vdash F[f \mapsto v]$.
14. By 7.), 8.a), 13.) $\Gamma \vdash H(\iota.f) := v$.
15. By 2.a,c), 3.b) and (WF-STACK-FRAME), $\Gamma; E; \rho \vdash \langle V, s' \rangle$.
16. By 1.c,d), 14.), 15.) and (WF-SCHEDULED), $\Gamma \vdash (H(\iota.f) := v; (S\langle V, s' \rangle, \rho); \overline{T})$.
17. By 17.) and (WF-UNSCHEDULED), $\Gamma \vdash (H(\iota.f) := v; (S\langle V, s' \rangle, \rho); \overline{T})$.

$\tau\ x; s'$

1. By (WF-SCHEDULED),
   (a) $\Gamma \vdash H$,
   (b) $\Gamma; \underline{E}; \rho \vdash \langle V, \tau\ x; s' \rangle$ for some E,
   (c) $\Gamma; \overline{E}; \rho \vdash S$ for some $\overline{E}$, and
   (d) $\Gamma \vdash (H, \overline{T})$
2. By 1.b) and (WF-STACK-FRAME),
   (a) $E \vdash \tau\ x; s'; E'$ and
   (b) $\Gamma; E; \rho; \theta \vdash V$ where
   (c) $\theta\ \_ = \Gamma(V(\texttt{this}))$.
3. By 2.a) and (SEQUENCE),
   (a) $E \vdash \tau\ x; E''$ and
   (b) $E'' \vdash s'; E'$.
4. By 3.a) and (VAR-DECL),
   (a) $\vdash \tau$,
   (b) $x \notin dom(E)$,
   (c) $\texttt{this} : \alpha\ c \in E$, and
   (d) $E'' = E[x : \alpha \oplus \tau]$.
5. By 4.a) and (TYPE), $\vdash d$ where $\tau = \_\ d$.
6. By 5.) and (TYPE-NULL), $\Gamma \vdash \texttt{null} : \mathsf{rtt}(\alpha \oplus \tau, \theta, \rho)$.
7. By 2.b), 6.) and (WF-FRAME), $\Gamma; E''; \rho; \theta \vdash V[x \mapsto v]$.
8. By 2.c), 3.b), 4.d), 7.) and (WF-STACK-FRAME),
   (a) $\Gamma; E''; \rho \vdash \langle V[x \mapsto v], s' \rangle$.
9. By 1.a,c,d), 8.) and (WF-SCHEDULED), $\Gamma \vdash (H; S\langle V[x \mapsto v], s' \rangle, \rho; \overline{T})$
10. By 9.) and (WF-UNSCHEDULED), $\Gamma \vdash (H; S\langle V[x \mapsto v], s' \rangle, \rho, \overline{T})$

$x = \texttt{new}\ \alpha\ c\texttt{()}; s'$

1. By (WF-SCHEDULED),
   (a) $\Gamma \vdash H$,
   (b) $\Gamma; \underline{E}; \rho \vdash \langle V, x = \texttt{new}\ \alpha\ c\texttt{()}; s' \rangle$ for some E,
   (c) $\Gamma; \overline{E}; \rho \vdash S$ for some $\overline{E}$, and
   (d) $\Gamma \vdash (H, \overline{T})$
2. By 1.b) and (WF-STACK-FRAME),
   (a) $E \vdash x = \texttt{new}\ \alpha\ c\texttt{()}; s'; E$ and
   (b) $\Gamma; E; \rho; \theta \vdash V$ where
   (c) $\theta\ \_ = \Gamma(V(\texttt{this}))$.
3. By 2.a) and (SEQUENCE),
   (a) $E \vdash x = \texttt{new}\ \alpha\ c\texttt{()}; E$ and
   (b) $E \vdash s'; E'$.

4. By 3.a) and (NEW),
   (a) $\vdash E$,
   (b) $x : \tau \in E$, and
   (c) $\vdash \alpha\ c \leq \tau$.
5. Evaluation can proceed in three cases, (NEW-CONTEXT), (NEW-THREAD), and (NEW-SHARED). As they are extremely similar, we only prove the first, which is arguably the most complex.
6. By (NEW-SHARED),
   (a) $H(V(\texttt{this})) = (\theta'\ \_, \_)$, $\overline{\texttt{@Context}\ f_1}, \overline{\texttt{@Shared}\ f_2}, \overline{\texttt{@Thread}\ f_3}$.
   (b) $\mathsf{fields}(c) = \overline{\texttt{@Context}\ f_1}, \overline{\texttt{@Shared}\ f_2}, \overline{\texttt{@Thread}\ f_3}$.
7. By 1.a) and (WF-HEAP), $\theta' = \theta$.
8. By 5.), (FIELDS-EMPTY), (FIELD-OTHER), and (TYPE-NULL) $\Gamma; [][\overline{\texttt{@Context}\ f_1}][\overline{\texttt{@Shared}\ f_2}]; \theta \vdash [][\overline{f_1} \mapsto \overline{\texttt{null}}][\overline{f_2} \mapsto \overline{\texttt{null}}]$.
9. By 4.c) and (SUBTYPE), $\vdash \alpha\ c$.
10. By 9.), Theorem 1 ($\theta \in \{\varrho, \rho\}$), (WF-$\Gamma$1), and (WF-$\Gamma$2), $\vdash \Gamma[\iota : \theta\ c]$.
11. By 1.a), 6.), 7.), 10.) and (WF-HEAP), $\Gamma[\iota : \theta\ c] \vdash H'$ where $H' = H[\iota \mapsto (\theta\ c, [][\overline{f_1} \mapsto \overline{\texttt{null}}][\overline{f_2} \mapsto \overline{\texttt{null}}])]$.
12. By 10.) (TYPE-VALUE), $\Gamma \vdash \iota : \mathsf{rtt}(\alpha\ c, \theta, \rho)$.
13. By 2.b), 4.b,c), 11.) and (WF-FRAME), $\Gamma; E; \rho; \theta \vdash V[x \mapsto \iota]$.
14. By 10.), 13.), and Lemma 1 (Extension), $\Gamma[\iota : \theta\ c]; E; \rho; \theta \vdash V[x \mapsto \iota]$.
15. By 1.c) and Lemma 1 (Extension), $\Gamma[\iota : \theta\ c]; \overline{E}; \rho \vdash S$.
16. By 3.b), 15.) and (WF-STACK-FRAME), $\Gamma[\iota : \theta\ c]; E; \rho \vdash \langle V[x \mapsto \iota], s' \rangle$.
17. By 11.), 15.), 16.) and (WF-SCHEDULED), $\Gamma[\iota : \theta\ c](H'; (S\langle V[x \mapsto \iota], s' \rangle, \rho); \overline{T})$.
18. By 17.) and (WF-UNSCHEDULED), $\Gamma[\iota : \theta\ c](H'; (S\langle V[x \mapsto \iota], s' \rangle, \rho), \overline{T})$.

$x = y.m(\overline{z}); s'$

1. By (WF-SCHEDULED),
   (a) $\Gamma \vdash H$,
   (b) $\Gamma; E; \rho \vdash \langle V, x = y.m(\overline{z}); s' \rangle$ for some E,
   (c) $\Gamma; \overline{E}; \rho \vdash S$ for some $\overline{E}$, and
   (d) $\Gamma \vdash (H, \overline{T})$
2. By 1.b) and (WF-STACK-FRAME),
   (a) $E \vdash x = y.m(\overline{z}); s'; E$ and
   (b) $\Gamma; E; \rho; \theta \vdash V$ where
   (c) $\theta\ \_ = \Gamma(V(\texttt{this}))$.
3. By 2.a) and (SEQUENCE),
   (a) $E \vdash x = y.m(\overline{z}); E$ and
   (b) $E \vdash s'; E'$.
4. By 3.a) and (METHOD-CALL),
   (a) $y : \alpha\ c \in E$,
   (b) $\mathsf{mtype}(c.m) = \overline{\tau} \to \tau'$,
   (c) $E \vdash \overline{z} : \alpha \oplus \overline{\tau}$,
   (d) $x : \tau \in E$,
   (e) $\vdash \alpha \oplus \tau' \leq \tau$.
5. A method call can proceed in two ways, $V(y) = \texttt{null}$ and $V(y) = \iota$. In the former, (METHOD-CALL-NPE) trivially satisfies the theorem by 1.a,d) and (WF-NPE), $\Gamma \vdash (H; (\texttt{NPE}, \rho), \overline{T})$. We now continue with the more interesting second case. By (METHOD-CALL),

(a) $V(v) = \iota$,
(b) $H(\iota) = (\theta'\ d, \_)$,
(c) $\mathsf{mbody}(d.m) = (\overline{x'}, s''\texttt{; return } y'')$, and
(d) $V(\overline{z})\overline{v}$.

6. By 2.b), 4.a) and Lemma 3 (Lookup), $\Gamma \vdash \iota : \mathsf{rtt}(\alpha\ c, \theta, \rho)$.
7. By 3.b), 4.c,d) and Lemma 2 (Well-Formed Construction),
   (a) $\vdash E$,
   (b) $\vdash \alpha \oplus \overline{\tau}$ and
   (c) $\vdash \tau$.
8. By 4.a), 7.) and (WF-E), $\vdash \alpha\ c$.
9. By 7.), 8.), (EMPTY-E), and (WF-E), $\vdash E''$ where $E'' = [][\texttt{this} : \alpha\ c][\overline{x'} : \overline{\alpha \oplus \tau}]$.
10. By 2.b), 4.c) and Lemma 3 (Lookup), $\Gamma \vdash \overline{v} : \overline{\mathsf{rtt}(\alpha \oplus \tau, \theta, \rho)}$. For each v_i typed α_i _, if $\alpha_i \in \{\texttt{@Shared}, \texttt{@Thread}\}$, $\mathsf{rtt}(\alpha \oplus \tau, \theta, \rho) = \mathsf{rtt}(\alpha \oplus \tau, \theta', \rho)$ by (TYPE-EQ-SHARED/THREAD). If $\alpha_i = \texttt{@Context}$, then $\theta = \theta'$ by 6.) and def. of rtt. Thus, $\Gamma \vdash \overline{v} : \overline{\mathsf{rtt}(\alpha \oplus \tau, \theta', \rho)}$.
11. By 6.), 9.), 10.) and (WF-FRAME), $\Gamma; E''; \rho; \theta' \vdash V'$ where $V' = [][\texttt{this} \mapsto \iota][\overline{x} \mapsto \overline{v}]$.
12. By 5.c) and (WF-METHOD), $E'' \vdash s''\texttt{;return } y''; E_3$.
13. By 11.), 12.) and (WF-STACK-FRAME), $\Gamma; E''; \rho \vdash \langle V', s''\texttt{;return } y''\rangle$.
14. By 7.a,c), $\vdash E[\texttt{ret} : \tau]$. By (WF-E), $E \vdash \texttt{ret} : \tau$.
15. By 4.d), 14.), and (ASSIGN), $E[\texttt{ret} : \tau] \vdash x = \texttt{ret}; E[\texttt{ret} : \tau]$.
16. By 3.b), 15.), and Lemma 1 (Extension), $E[\texttt{ret} : \tau] \vdash s; E'[\texttt{ret} : \tau]$.
17. By 16.), E.), and (SEQUENCE), $E[\texttt{ret} : \tau] \vdash x = \texttt{ret;}s; E'[\texttt{ret} : \tau]$.
18. By, 2.b), (TYPE-NULL), and (WF-FRAME), $\Gamma; E[\texttt{ret} : \tau]; \rho; \theta \vdash V[\texttt{ret} \mapsto \texttt{null}]$.
19. By 2.c), 17.), 18.) and (WF-STACK-FRAME), $\Gamma; E[\texttt{ret} : \tau]; \rho \vdash \langle V[\texttt{ret} \mapsto \texttt{null}], s''\texttt{;return } y''\rangle$.
20. By 1.a,c,d), 13.), 19.), and (WF-SCHEDULED), $\Gamma \vdash (H; (S', \rho); \overline{T})$ where $S' = S\langle V[\texttt{ret} \mapsto \texttt{null}], x = \texttt{ret; } s'\rangle\langle V', s''\texttt{; return } y''\rangle$.
21. By 20.) and (WF-UNSCHEDULED), $\Gamma \vdash (H; (S', \rho), \overline{T})$

$x = \texttt{start } c\texttt{()}; s'$ The proof is very similar to (NEW) and (METHOD-CALL) and is therefore omitted. The new top-most frame is placed in a new thread with a fresh thread id.

□

Theorem 4. *Progress. If $\Gamma \vdash (H; T; \overline{T})$, then there exists a reduction such that $(H; T; \overline{T}) \to (H'; \overline{T'})$ where $\overline{T} \subseteq \overline{T'}$.*

Proof. The case where $T = (\texttt{NPE}, \rho)$ satisfies the theorem for $\overline{T'} = \overline{T}$. The rest of the proof is by structural induction on the shape of s when $T = S\langle V, s\rangle, \rho$. Essentially, a statement can get stuck in our system if variables, fields, classes or methods mentioned do not exist, or on an attempt at returning from the bottom stack frame.

$\texttt{return}\ y$ There are two sub-cases, $S = \epsilon$ and $S = S'\langle V', s'\rangle$. The first case follows immediately from (FINISHED-THREAD). We now prove the second—there exists a local variable y. By (WF-SCHEDULED), 1.) $\Gamma; E; \rho \vdash \langle V, \texttt{return}\ y\rangle$ for some E. By 1.) and (WF-STACK-FRAME), 2.) $E \vdash \texttt{return}\ y; E$ and 3.) $\Gamma; E; \rho; \theta \vdash V$ for some uninteresting θ. By 2.) and (RETURN), 4.) $E \vdash y : \tau$. By 4.) and (WF-E), 5.) $y\tau \in E$. By 3., 5.) and (WF-FRAME) $V(y) = v$.

$s'; s''$ Immediate from induction hypothesis.

$\texttt{skip}; s'$ Immediate.

$x = y.f; s'$ By (WF-SCHEDULED), 1.) $\Gamma \vdash H$, and 2.) $\Gamma; E; \rho \vdash \langle V, x = y.f; s'\rangle$ for some E. By 2.) and (WF-STACK-FRAME), 3.) $E \vdash x = y.f; s'; E'$, and 4.) $\Gamma; E; \rho; \theta \vdash V$ for some uninteresting θ. By 3.) and (SELECT), 5.) $x : \tau \in E$, $y : \alpha\ c \in E$, and $\mathsf{field}(c.f) = \alpha'\ d$. By 4., 5.) and (WF-FRAME), 6.) $V(y) = v$, and $\Gamma \vdash v : \theta'\ c'$ for some c' s.t. $\vdash c' \leq c$.

There are two sub-cases $v = \iota$ and $v = \texttt{null}$. The latter case is handled by (SELECT-NPE) which trivially satisfies the theorem for $\overline{T'}$=($\texttt{NPE}$,ρ),$\overline{T}$.

For the former, there are two sub-cases $\alpha = \texttt{@Thread}$ and $\alpha \neq \texttt{@Thread}$. The first case is uninteresting as (SELECT-FIRST-THREAD) allows taking a step even if a $\texttt{@Thread}$ field is undefined. Thus, we concentrate on the second.

By 1., 6.) and (WF-HEAP), 7.) $H(\iota) = (\theta'\ c', F)$, i.e., there is an object ι on the heap with fields F, and (8) $\Gamma; \mathsf{fields}(c'); \theta' \vdash F$. Clearly, $\mathsf{fields}(c) \subseteq \mathsf{fields}(c')$, so by 5.) and (FIELD-OTHER), $F(f) = v_3$—the field exists.

$x = y; s'$ Similar to $\texttt{return}\ y$ and therefore omitted.

$y.f = z; s'$ Similar to $x = y.f; s'$ and therefore omitted.

$\tau\ x; s'$ Similar to $\texttt{return}\ y$ and therefore omitted.

$x = \texttt{new}\ \alpha\ c\texttt{()}; s'$ By (WF-SCHEDULED), 1.) $\Gamma; E; \rho \vdash \langle V, x = \texttt{new}\ \alpha\ c\texttt{()}; s'\rangle$ for some E. By 1.) and (WF-STACK-FRAME), 2.) $E \vdash x = \texttt{new}\ \alpha\ c\texttt{()}; s'; E'$. By 2.) and (NEW), 3.) $x : \alpha\ d \in E$ and 4.) $\vdash c \leq d$. By 4.) and (CLASS) and (SUBCLASS-*), there is a class c in P, so $\mathsf{fields}(c) \neq \bot$.

Notably, the initial configuration has $V = [\texttt{this} \mapsto \iota]$ and $H = [\iota \mapsto (\rho\ \texttt{Object}, \_)]$ where ρ is the thread id of the first thread. Thus, as (METHOD-CALL) always puts a $\texttt{this}$ in the stack, looking up $H(V(\texttt{this}))$ always succeeds.

$x = y.m(\overline{z}); s'$ By (WF-SCHEDULED), 1.) $\Gamma; E; \rho \vdash \langle V, x = y.m(\overline{z}); s'\rangle$ for some E. By 1.) and (WF-STACK-FRAME), 2.) $E \vdash x = y.m(\overline{z}); s'; E'$, and 3.) $\Gamma; E; \rho; \theta \vdash V$ for some uninteresting θ. By 2.) and (METHOD-CALL), 4.) $x : \tau \in E$, 5.) $y : \alpha\ c \in E$, 6.) $\mathsf{mtype}(c.m) = \overline{\tau} \to \tau'$, 7.) $E \vdash \overline{z} : \alpha \oplus \overline{\tau}$. By 2., 5.) and (WF-FRAME), $V(y) = v$ s.t. $\Gamma \vdash v : \_\ d$ s.t. $\vdash d \leq c$. If $v = \texttt{null}$, (METHOD-CALL-NPE) satisfies the theorem for $\overline{T'} = (\texttt{NPE}, \rho), \overline{T}$. Clearly, $\mathsf{mbody}(c) \subseteq \mathsf{mbody}(d)$. Thus, by 6.) the method exists and so $\mathsf{mbody}(c.m) \neq \bot$. Showing the existance $\overline{z}$ in V is similar to y and therefore omitted.

$x = \texttt{start}\ c\texttt{()}; s'$ Forking a new thread is a copy-and-patch proof of (NEW) and (METHOD-CALL) and therefore omitted.

□

References

1. A. K. Wright and M. Felleisen. A syntactic approach to type soundness. *Inf. Comput.*, 115(1):38–94, 1994.
2. T. Wrigstad, F. Pizlo, F. Meawad, L. Zhao, and J. Vitek. Loci: Simple thread-locality for java. Submitted.