# Ownership Domains Revisited

**Radu Vanciu** **Marwan Abi-Antoun**

May 2012

Department of Computer Science
Wayne State University
Detroit, MI 48202

**Abstract**

This technical report revisits the Ownership Domains type system by Aldrich and Chambers, which appeared at ECOOP 2004, and contributes the proofs for Preservation, Progress, Heap Link Soundness, and Expression Link Soundness. We also discuss relaxing some of the restrictions of the original type system, to make it possible to add ownership types to legacy code that was written without ownership in mind.

**Keywords:** Ownership Domains, Featherweight Domain Java, soundness proof, Heap Link Soundness, Expression Link Soundness

# Contents

# List of Figures

## List of Symbols

# 1 Ownership Domains Revisited

This paper revisits the Ownership Domains type system by Aldrich and Chambers [2]. The original paper cites an unpublished companion technical report [3], which contains an informal, partial proof. This technical report revises the formal system, and contributes a full, formal, soundness proof.

For completeness, we reproduce here the Featherweight Domain Java (FDJ) type system, incorporating corrections from the errata[1] and making additional changes listed below. The differences between the type system discussed in this report and the original paper are:

- We added field write expression and the corresponding static and dynamic semantics adapted from [1].
- We did some renaming in the FDJ syntax:
  - $d$ is a name of a domain instead of $x$.
  - $x$ ranges over variable names;
  - $p$ ranges over domains;
- We added a few things:
  - added `shared`, and $\ell_0$ as a dummy parent of `shared`.
  - we used $\theta$ to denote the receiver in the dynamic semantics;
  - we added $\Delta$ as the union of all links between runtime domains;
- We added explicit `that`/`this` substitution in type checking rules (Fig. 6) and auxiliary judgements (Fig. 9). Adapted from [5].
- We added well formed environment and types rules to replace the implicit check $T \neq \texttt{ERROR} \Longrightarrow \Gamma; \Sigma; n_{this} \models n_{this} \rightarrow owner(T)$. Adapted from [4].
- We check for well-formed types in the static typechecking rules and *ClsOK* and *MethOK*. The additional checks are *highlighted*. In *ClsOK*, we check that the types of the fields are well-formed. *MethOK* verifies that type of the arguments and the type of the return expression are well-formed.
- We check for well-formed types in Preservation, Progress, and Expression Link Soundness theorems.
- We changed *T-DynamicLink* so that it requires the link to be in $\Delta$ instead of $links(\Sigma[\ell])$.
- We changed *T-Assumptions*, and *T-Store* to use the set of all the links $\Delta$. We avoided using $\models$; instead, we used set inclusion to avoid any confusion.
- We changed *T-Context*, so that it explicitly check for well formed types.
- We added additional lemmas which helped us in the proof. The lemmas provide the reader with additional insights that may be useful in the process of adding annotations.

**Outline.** We do not repeat the motivation or examples from the original paper, and delve straight into the formal system (Section 2). Next, we state and prove type soundness and link soundness for Featherweight Domain Java (Section 3). Finally, we discuss some extensions (Section 4) to relax some of the restrictions of the original type system, to make it easier to add ownership types to legacy code that was written without ownership in mind.

[1] Errata available at: `http://www.cs.cmu.edu/~aldrich/papers/ownership-domains-errata.html`

$$
\begin{array}{rcl}
CL & ::= & \texttt{class}\ C\texttt{<}\overline{\alpha},\overline{\beta}\texttt{>}\ \texttt{extends}\ C'\texttt{<}\overline{\alpha}\texttt{>} \\
 & & \texttt{assumes}\ \overline{\gamma} \to \overline{\delta}\ \{\ \overline{D}\ \overline{L}\ \overline{F}\ K\ \overline{M}\ \} \\
D & ::= & [\texttt{public}]\ \texttt{domain}\ d; \\
L & ::= & \texttt{link}\ p \to p'; \\
F & ::= & T\ f; \\
K & ::= & C(\overline{T'}\ \overline{f'}, \overline{T}\ \overline{f})\ \{\texttt{super}(\overline{f'});\ \overline{\texttt{this}.f = f};\} \\
M & ::= & T_R\ m(\overline{T}\ \overline{x})\ T_{\texttt{this}}\ \{\ \texttt{return}\ e;\ \} \\
e & ::= & x \\
 & | & \texttt{new}\ C\texttt{<}\overline{p}\texttt{>}(\overline{e}) \\
 & | & e.f \\
 & | & e.f = e' \\
 & | & e.m(\overline{e}) \\
 & | & (T)e \\
 & | & \ell \\
 & | & \ell \triangleright e \\
 & | & \texttt{error} \\
z & ::= & \texttt{this} \mid \texttt{that} \mid x \\
n & ::= & z \mid v \\
p & ::= & \alpha \mid n.d \mid \texttt{shared} \\
T & ::= & C\texttt{<}\overline{p}\texttt{>} \mid \texttt{ERROR} \\
v, \ell, \ell_0, \theta & \in & \mathit{locations} \\
S & ::= & \ell \mapsto C\texttt{<}\overline{\ell'.d}\texttt{>}(\overline{v}) \\
\Gamma & ::= & x \mapsto T \\
\Sigma & ::= & \ell \mapsto T \\
\Delta & ::= & \{\ell_1.d_1 \to \ell_2.d_2\}
\end{array}
$$

Figure 1: Featherweight Domain Java Abstract Syntax.

# 2 Formal System

## 2.1 Syntax

Figure 1 shows the syntax of FDJ.

- $C$ ranges over class names;
- $T$ ranges over types;

- $f$ ranges over fields;
- $v$ ranges over values;
- $e$ ranges over expressions;
- $x$ ranges over variable names; $d$ over domain names;
- $n$ ranges over values and variable names;
- $p$ ranges over domains;
- $S$ ranges over stores;
- $\ell$ range over locations in the store;
- $n_{this}$ is used to represent the name or the value of `this`;
- $\theta$ is the location corresponding to the value of `this`;
- $\alpha$, $\beta$, $\gamma$, and $\delta$ range over formal domain parameters;
- $m$ ranges over method names.
- As a shorthand, an overbar is used to represent a sequence.
- A store S maps locations $\ell$ to their contents: the class of the object, the actual ownership domain parameters, and the values stored in its fields.
- $S[\ell]$ denotes the store entry for $\ell$;
- $S[\ell, i]$ to denote the value in the $i$th field of $S[\ell]$.
- Adding an entry for location $\ell$ to the store is abbreviated $S[\ell \mapsto C{<}\overline{p}{>}(\overline{v})]$.
- $\ell \triangleright e$ represents a method body $e$ executing with a receiver $\ell$. Result of the computation is a location $\ell$, which is sometimes referred to as a value $v$.
- The set of variables $z$ includes the variable `this` of type $T_{\texttt{this}}$ used to refer to the current context, and `that` used to refer to the context of a receiver.

$$\frac{\ell \notin domain(S) \qquad S' = S[\ell \mapsto C\!<\!\overline{\ell'.d}\!>\!(\overline{v})]}{S;\theta \vdash \texttt{new}\ C\!<\!\overline{\ell'.d}\!>\!(\overline{v}) \mapsto \ell, S'}\ \textit{R-New}$$

$$\frac{S[\ell] = C\!<\!\overline{\ell'.d}\!>\!(\overline{v}) \qquad fields(C\!<\!\overline{\ell'.d}\!>) = \overline{T}\ \overline{f}}{S;\theta \vdash \ell.f_i \mapsto v_i, S}\ \textit{R-Read}$$

$$\frac{S[\ell] = C\!<\!\overline{\ell'.d}\!>\!(\overline{v}) \qquad fields(C\!<\!\overline{\ell'.d}\!>) = \overline{T}\ \overline{f} \qquad S' = S[\ell \mapsto C\!<\!\overline{\ell'.d}\!>\!([v/v_i]\overline{v})]}{S;\theta \vdash \ell.f_i = v \mapsto v,\ S'}\ \textit{R-Write}$$

$$\frac{S[\ell] = C\!<\!\overline{\ell'.d}\!>\!(\overline{v}) \qquad mbody(m, C\!<\!\overline{\ell'.d}\!>) = (\overline{x}, e_0)}{S;\theta \vdash \ell.m(\overline{v}) \mapsto \ell \triangleright [\overline{v}/\overline{x}, \ell/\texttt{this}]e_0, S}\ \textit{R-Invk}$$

$$\frac{S[\ell] = C\!<\!\overline{\ell'.d}\!>\!(\overline{v}) \qquad C\!<\!\overline{\ell'.d}\!> <: T}{S;\theta \vdash (T)\ell \mapsto \ell, S}\ \textit{R-Cast}$$

$$\frac{S[\ell] = C\!<\!\overline{\ell'.d}\!>\!(\overline{v}) \qquad C\!<\!\overline{\ell'.d}\!> \not<: T}{S;\theta \vdash (T)\ell \mapsto \texttt{error}, S}\ \textit{E-Cast}$$

$$\frac{}{S;\theta \vdash \ell \triangleright v \mapsto v, S}\ \textit{R-Context}$$

Figure 2: Dynamic Semantics

- Fixed class table $CT$ mapping classes to their definitions.
- A program, then, is a tuple $(CT, S, e)$ of a class table, a store, and an expression.
- The store type $\Sigma$ maps locations $\ell$ to a runtime type $T$.
- $\Delta$ is the union of all links between runtime domains;
- When necessary, we distinguish between a static type $C\!<\!\overline{p}\!>$ and a runtime type $C\!<\!\overline{\ell'.d}\!>$.

## 2.2 Dynamic semantics

The evaluation relation, defined by the dynamic semantics given in Figure 2, is of the form $S;\theta \vdash e \mapsto e', S'$, read "In the context of store $S$, and receiver $\theta$, expression $e$ reduces to expression $e'$ in one step, producing the new store $S'$. We write $\mapsto^*$ for the reflexive, transitive closure of $\mapsto$. Most of the rules are standard; the interesting features are how they track ownership domains.

The *R-New* rule reduces an object creation expression to a fresh location. The store is extended at that location to refer to a class with the specified ownership parameters, with the fields set to the values passed to the constructor.

The *R-Read* and *R-Write* rules look up the receiver in the store and identifies the $i$th field. The result of *R-Read* is the value at field position $i$ in the store. *R-Write* changes the store by replacing the value at the field position $i$ with the value $v$ of the right-hand side of the field write expression, and returns $v$. In the original rules, *R-Read* and *R-Write* call the $fields$ auxiliary judgements to denote that $f_i$ is the field at position $i$. Since, in the latest version of $fields$ we added `that/this` substitution, and `this` is not encountered in the dynamic semantics, we removed this call. As an alternative, we could still call $fields$, but we assume that `that/this` substitution is ignored if the argument is a runtime type.

As in Java (and FJ), the *R-Cast* rule checks that the cast expression is a subtype of the cast type. Note, however, that in FDJ this check also verifies that the ownership domain parameters match, doing an extra run-time check that is not present in Java. If the run-time check in the cast rule fails, however, then the cast reduces to the `error` expression, following the cast error rule *E-Cast*. This rule shows how the formal system models the exception that is thrown by the full language when a cast fails.

The method invocation rule *R-Invk* looks up the receiver in the store, then uses the *mbody* helper function (defined in Figure 9) to determine the correct method body to invoke. The method invocation is replaced with the appropriate method body. In the body, all occurrences of the formal method parameters and `this` are replaced with the actual arguments and the receiver, respectively. Here, the capture-avoiding substitution of values $\overline{v}$ for variables $\overline{x}$ in $e$ is written $[\overline{v}/\overline{x}]e$. Execution of the method body continues in the context of the receiver location.

When a method expression reduces to a value, the *R-Context* rule propagates the value outside of its method context and into the surrounding method expression. The full definition of FDJ also includes congruence rules that allow reduction to proceed within an expression in the order of evaluation defined by Java. For example, the congruence rule for the field read expression states that an expression $e.f$ reduces to $e'.f$ whenever $e$ reduces to $e'$.

$$\frac{S;\theta \vdash e_i \mapsto e_i', S'}{S;\theta \vdash C{<}\overline{\ell'.d}{>}(v_{1..i-1}, e_i, e_{i+1..n}) \mapsto C{<}\overline{\ell'.d}{>}(v_{1..i-1}, e_i', e_{i+1..n}), S'} \; \textit{RC-New}$$

$$\frac{S;\theta \vdash e \mapsto e', S'}{S;\theta \vdash e.f_i \mapsto e'.f_i, S'} \; \textit{RC-Read}$$

$$\frac{S;\theta \vdash e \mapsto e', S'}{S;\theta \vdash e.f_i = e_{arg} \mapsto e'.f_i = e_{arg}, S'} \; \textit{RC-RecvWrite}$$

$$\frac{S;\theta \vdash e \mapsto e', S'}{S;\theta \vdash v.f_i = e \mapsto v.f_i = e', S'} \; \textit{RC-ArgWrite}$$

$$\frac{S;\theta \vdash e \mapsto e', S'}{S;\theta \vdash e.m(\overline{e}) \mapsto e'.m(\overline{e}), S'} \; \textit{RC-RecvInvk}$$

$$\frac{S;\theta \vdash e_i \mapsto e_i', S'}{S;\theta \vdash v.m(v_{1..i-1}, e_i, e_{i+1..n}) \mapsto v.m(v_{1..i-1}, e_i', e_{i+1..n}), S'} \; \textit{RC-ArgInvk}$$

$$\frac{S;\theta \vdash e \mapsto e', S'}{S;\theta \vdash (T)e \mapsto (T)e', S'} \; \textit{RC-Cast}$$

$$\frac{S;\ell \vdash e \mapsto e', S'}{S;\theta \vdash \ell \triangleright e \mapsto \ell \triangleright e', S'} \; \textit{RC-Context}$$

Figure 3: Congruence and Error Rules

$$\frac{}{\emptyset;\Sigma;n_{this} \vdash \Diamond} \; \textit{T-Env-Empty}$$

$$\frac{\Gamma;\Sigma;n_{this} \vdash T \qquad x \notin dom(\Gamma)}{\Gamma, x:T;\Sigma;n_{this} \vdash \Diamond} \; \textit{T-Env-X}$$

$$\frac{\Gamma;\Sigma;n_{this} \models n_{this} \rightarrow owner(T) \qquad \Gamma;\Sigma;n_{this} \models assumptions(T)}{\Gamma;\Sigma;n_{this} \vdash T} \; \textit{T-Type}$$

Figure 4: Well Formed Environment and Types Rules

Figure 3 shows the congruence rules that allow reduction to proceed within an expression in the order of evaluation defined by Java. For example, the rule RC-Read states that an expression $e.f$ reduces to $e'.f$ whenever $e$ reduces to $e'$. The congruence rule RC-Context shows the semantics of the $\ell \triangleright e$ construct: evaluation of the expression e occurs in the context of the receiver $\ell$ instead of the receiver $\theta$.

## 2.3 Well Formed Environment and Types

The well formed rules come in two forms: $\Gamma; \Sigma; n_{this} \vdash \Diamond$ and $\Gamma; \Sigma; n_{this} \vdash T$. The first form of the rule is read: "Given the type environment $\Gamma$, the store type $\Sigma$, and a name for the current object $n_{this}$, the type environment is well formed". The second form is similar, except that the conclusion is "the type $T$ is well formed" (Fig. 4 ).

According to *T-Env-Empty* and *T-Env-X*, an empty type environment is well formed by definition, while a non-empty one is well formed if whenever $\Gamma$ is extended with a new variable $x$, the type $T$ of $x$ is well formed. Next, according to *T-Type*, a given type $T$ is well formed if $n_{this}$ has permission to access the owner domain of $T$, and verifies that the assumptions that $T$ makes of its domain parameters are justified based on the current typing environment. *T-Type* uses the link permission rules defined in Figure 8. We use $\Gamma; \Sigma; n_{this} \models assumptions(C\overline{<p>})$ as a shorthand for $\forall\ (p_1 \rightarrow p_2) \in assumptions(C\overline{<p>}),\ \Gamma; \Sigma; n_{this} \models p_1 \rightarrow p_2$. To avoid duplication, we also use *T-Type* to check that dynamic types $C\overline{<\ell'.d>}$ are well formed.

$$\frac{CT(C) = \texttt{class}\ C\overline{<\alpha, \beta>}\ \texttt{extends}\ C'\overline{<\alpha>} \ldots}{C\overline{<p, p'>} <: C'\overline{<p>}}\ \textit{Subtype-Class}$$

$$\frac{}{T <: T}\ \textit{Subtype-Reflex}$$

$$\frac{T <: T' \qquad T' <: T''}{T <: T''}\ \textit{Subtype-Trans}$$

$$\frac{}{\texttt{ERROR} <: T}\ \textit{Subtype-Error}$$

Figure 5: Subtyping Rules

## 2.4 Typing Rules

FDJ's subtyping rules are given in Figure 5. Subtyping is derived from the immediate subclass relation given by the `extends` clauses in the class table $CT$. The subtyping relation is reflexive and transitive, and it is required that there be no cycles in the relation (other than self-cycles due to reflexivity). The `ERROR` type is a subtype of every type.

$$\frac{\begin{array}{c}\Gamma;\Sigma;n_{this} \models assumptions(C\!<\!\overline{p}\!>) \qquad \Gamma;\Sigma;n_{this} \vdash \overline{e} : \overline{T'} \\ fields(C\!<\!\overline{p}\!>) = \overline{T}\ \overline{f} \qquad \overline{T'} <: \overline{T} \qquad \Gamma;\Sigma;n_{this} \vdash n_{this}{:}T_{this} \\ owner(C\!<\!\overline{p}\!>) \in (domains(T_{this}) \cup owner(T_{this})) \\ \Gamma;\Sigma;n_{this} \vdash \overline{T'} \qquad \Gamma;\Sigma;n_{this} \vdash T_{this}\end{array}}{\Gamma;\Sigma;n_{this} \vdash \texttt{new}\ C\!<\!\overline{p}\!>(\overline{e}) : C\!<\!\overline{p}\!>}[\text{T-New}]$$

$$\frac{\Gamma;\Sigma;n_{this} \vdash e_0 : T_0 \qquad fields(T_0) = \overline{T}\ \overline{f} \qquad \Gamma;\Sigma;n_{this} \vdash T_0}{\Gamma;\Sigma;n_{this} \vdash e_0.f_i : [e_0/\texttt{that}]T_i}[\text{T-Read}]$$

$$\frac{\begin{array}{c}\Gamma;\Sigma;n_{this} \vdash e_0 : T_0 \qquad fields(T_0) = \overline{T}\ \overline{f} \qquad T_i \in \overline{T} \\ \Gamma;\Sigma;n_{this} \vdash e_R : T_R \qquad T_R <: [e_0/\texttt{that}]T_i \\ \Gamma;\Sigma;n_{this} \vdash T_0 \qquad \Gamma;\Sigma;n_{this} \vdash T_R\end{array}}{\Gamma;\Sigma;n_{this} \vdash e_0.f_i = e_R : T_R}[\text{T-Write}]$$

$$\frac{\begin{array}{c}\Gamma;\Sigma;n_{this} \vdash e_0 : T_0 \qquad \Gamma;\Sigma;n_{this} \vdash \overline{e} : \overline{T_a} \\ mtype(m,T_0) = \overline{T} \to T_R \qquad mbody(m,T_0) = (\overline{x}, e_R) \qquad \overline{T_a} <: [\overline{e}/\overline{x}, e_0/\texttt{that}]\overline{T} \\ \Gamma;\Sigma;n_{this} \vdash T_0 \qquad \Gamma;\Sigma;n_{this} \vdash \overline{T_a}\end{array}}{\Gamma;\Sigma;n_{this} \vdash e_0.m(\overline{e}) : [\overline{e}/\overline{x}, e_0/\texttt{that}]T_R}[\text{T-Invk}]$$

$$\frac{\Gamma(x) = C\!<\!\overline{p}\!>}{\Gamma;\Sigma;n_{this} \vdash x : C\!<\!\overline{p}\!>}[\text{T-Var}] \qquad \frac{\Sigma(\ell) = C\!<\!\overline{p}\!>}{\Gamma;\Sigma;n_{this} \vdash \ell : C\!<\!\overline{p}\!>}[\text{T-Loc}]$$

$$\frac{\Gamma;\Sigma;n_{this} \vdash e : T' \qquad \Gamma;\Sigma;n_{this} \vdash T}{\Gamma;\Sigma;n_{this} \vdash (T)e : T}[\text{T-Cast}]$$

$$\frac{\Gamma;\Sigma;n_{this} \vdash \Sigma[\ell] \qquad \Gamma;\Sigma;\ell \vdash e : T \qquad \Gamma;\Sigma;\ell \vdash T}{\Gamma;\Sigma;n_{this} \vdash \ell \triangleright e : T}[\text{T-Context}]$$

Figure 6: Typechecking rules.

### 2.5 Typechecking Rules

Typing judgments, shown in Figure 6, are of the form $\Gamma;\Sigma;n_{this}\vdash e : T$, read, "In the type environment $\Gamma$, store typing $\Sigma$, and receiver instance $n_{this}$ of type $T_{this}$, expression $e$ has type $T$."

The *T-Var* rule looks up the type of a variable in $\Gamma$. The *T-Loc* rule looks up the type of a location in $\Sigma$. The object creation rule verifies that any assumptions that the class being instantiated makes about its domain parameters are justified based on the current typing environment. It also checks that the parameters to the constructor have types that match the types of that class's fields. Finally, it verifies that the object being created is part of the same domain as $n_{this}$ or else

$$\frac{\begin{array}{c}\overline{M}\ OK\ in\ C \qquad fields(C'\!<\!\overline{\alpha}\!>) = \overline{T'}\ \overline{g} \qquad \overline{L}\ OK\ in\ C\!<\!\overline{\alpha},\overline{\beta}\!> \\ \overline{F} = \overline{T}\ \overline{f} \qquad \{\texttt{this} : C\!<\!\overline{\alpha},\overline{\beta}\!>\};\ \emptyset;\ \texttt{this} \vdash \overline{T} \\ K = C(\overline{T'}\ \overline{g},\ \overline{T}\ \overline{f})\ \{\ super(\overline{g});\ this.\overline{f} = \overline{f};\ \}\end{array}}{\texttt{class}\ C\!<\!\overline{\alpha},\overline{\beta}\!>\ \texttt{extends}\ C'\!<\!\overline{\alpha}\!>\ \texttt{assumes}\ \overline{\gamma} \to \overline{\delta}\ \{\ \overline{D};\ \overline{L};\ \overline{F};\ K\ \overline{M};\}\ OK}\ ClsOK$$

$$\frac{\begin{array}{c}CT(C) = \texttt{class}\ C\!<\!\overline{\alpha},\overline{\beta}\!>\ \texttt{extends}\ C'\!<\!\overline{\alpha}\!>\ldots \\ override(m, C'\!<\!\overline{\alpha}\!>, \overline{T} \to T_R) \\ \{\overline{x} : \overline{T};\ \texttt{this} : C\!<\!\overline{\alpha},\overline{\beta}\!>\};\ \emptyset;\ \texttt{this} \vdash e : T'_R \\ \{\overline{x} : \overline{T};\ \texttt{this} : C\!<\!\overline{\alpha},\overline{\beta}\!>\};\ \emptyset;\ \texttt{this} \vdash T'_R \\ \{\overline{x} : \overline{T};\ \texttt{this} : C\!<\!\overline{\alpha},\overline{\beta}\!>\};\ \emptyset;\ \texttt{this} \vdash T'_R \qquad T'_R <: T_R \\ \{\overline{x} : \overline{T};\ \texttt{this} : C\!<\!\overline{\alpha},\overline{\beta}\!>\};\ \emptyset;\ \texttt{this} \vdash \overline{T}\end{array}}{T_R\ m(\overline{T}\ \overline{x})\ \{\ \texttt{return}\ e;\ \}\ OK\ in\ C}\ MethOK$$

$$\frac{\begin{array}{c}\{p_1, p_2\} \cap domains(C\!<\!\overline{\alpha}\!>) \neq \emptyset \\ p_1 \notin domains(C\!<\!\overline{\alpha}\!>) \Longrightarrow (\texttt{this} : C\!<\!\overline{\alpha}\!>; \emptyset; \texttt{this} \models p_1 \to owner(C\!<\!\overline{\alpha}\!>)) \\ p_2 \notin domains(C\!<\!\overline{\alpha}\!>) \Longrightarrow (\texttt{this} : C\!<\!\overline{\alpha}\!>; \emptyset; \texttt{this} \models \texttt{this} \to p_2)\end{array}}{\texttt{link}\ p_1 \to p_2\ OK\ in\ C\!<\!\overline{\alpha}\!>}\ LinkOK$$

$$\frac{\forall \ell \in domain(\Sigma)\ \ (links(\Sigma[\ell])) \subseteq \Delta}{\Sigma_\Delta\ OK}\ T\text{-}Assumptions$$

$$\frac{\begin{array}{c}domain(S) = domain(\Sigma) \qquad S[\ell] = C\!<\!\overline{\ell'.d}\!>(\overline{v}) \iff \Sigma[\ell] = C\!<\!\overline{\ell'.d}\!> \\ fields(\Sigma[\ell]) = \overline{T}\ \overline{f} \Longrightarrow (S[\ell, i] = \ell'') \wedge (\Sigma[\ell''] <: T_i) \\ \Sigma_\Delta\ OK \\ (S[\ell, i] = \ell'') \Longrightarrow (owner(\Sigma[\ell]) \to owner(\Sigma[\ell''])) \in \Delta\end{array}}{\Sigma_\Delta \vdash S}\ T\text{-}Store$$

Figure 7: Class, Method and Store Typing.

is part of the domains declared by $n_{this}$.

The typing rule for `error` assigns it the type `ERROR`. The rule for field reads looks up the declared type of the field using the $fields$ function defined in Figure 9. The cast rule simply checks that the expression being cast is well-typed; a run-time check will determine if the value that comes out of the expression matches the type of the cast.

Rule *T-Invk* looks up the invoked method's type using the *mtype* function defined in Figure 9, and verifies that the actual argument types are subtypes of the method's argument types. Finally, the *T-Context* typing rule for an executing method checks the method's body in the context of the new receiver $\ell$.

Figure 7 shows the rules for typing classes, declarations within classes, and the store. The typing rules for classes and declarations have the form "class C is OK," and "method/link declaration is OK in C." The class rule checks that the methods and links in the class are well-formed, and that types of fields in the class are well formed.

The rule for methods checks that the method body is well typed, and uses the *override* function (defined in Figure 9) to verify that methods are overridden with a method of the same type. It also verifies that type of the arguments and the type of the return expression are well formed.

The link rule verifies that one of the two domains in the link declaration was declared locally, preventing a class from linking two external domains together. The rule also ensures that if the declaration links an internal and an external domain, there is a corresponding linking relationship between `this` and the external domain.

The store typing rule ensures that the store type gives a type to each location in the store's domain that is consistent with the classes and ownership parameters in the actual store. For every value in a field in the store, the type of the value must be a subtype of the declared type of the field. The check $\Sigma_\Delta \; OK$, defined by the *T-Assumptions* rule, creates $\Delta$ as the union of all links between runtime domains based on actual link declarations in the source code. Finally, the last check verifies link soundness for the store: if object $\ell$ refers to object $\ell''$ in it's $i$th field, then the link declarations implied by the store type $\Sigma$ imply that owner domain of $\Sigma[\ell]$ has permission to access the owner domain of $\Sigma[\ell'']$.

$$\Delta = \bigsqcup_{\ell \in domain(\Sigma)} (links(\Sigma[\ell]))$$

Figure 8 shows the rules for determining whether an object named by $n$ or a domain $p$ has permission to access another domain $p'$. These rules come in two forms: $\Gamma; \Sigma; n_{this} \models n \rightarrow p$ and $\Gamma; \Sigma; n_{this} \models p \rightarrow p'$. The first form of rule is read, "Given the type environment $\Gamma$, the store type $\Sigma$, and a name for the current object $n_{this}$, the object named by $n$ has permission to access domain $p$." The second form is similar, except that the conclusion is that any object in domain $p$ has permission to access domain $p'$. The two forms allow us to reason about access permission both on a per-object basis and on a per-domain basis.

$$\frac{(\ell_1.d_1 \to \ell_2.d_2) \in \Delta}{\Gamma; \Sigma; n_{this} \models \ell_1.d_1 \to \ell_2.d_2}\ \textit{T-DynamicLink}$$

$$\frac{\Gamma; \Sigma; n_{this} \vdash n_{this} : T \qquad (p_1 \to p_2) \in \mathit{linkdecls}(T)}{\Gamma; \Sigma; n_{this} \models p_1 \to p_2}\ \textit{T-DeclaredLink}$$

$$\frac{}{\Gamma; \Sigma; n_{this} \models n \to n.d}\ \textit{T-ChildRef} \qquad \frac{}{\Gamma; \Sigma; n_{this} \models p \to p}\ \textit{T-SelfLink}$$

$$\frac{\Gamma; \Sigma; n_{this} \vdash n : T \qquad \Gamma; \Sigma; n_{this} \models \mathit{owner}(T) \to p}{\Gamma; \Sigma; n_{this} \models n \to p}\ \textit{T-LinkRef}$$

$$\frac{\Gamma; \Sigma; n_{this} \vdash n : T \qquad \Gamma; \Sigma; n_{this} \models p \to \mathit{owner}(T) \qquad \mathit{public}(d)}{\Gamma; \Sigma; n_{this} \models p \to n.d}\ \textit{T-PublicLink}$$

$$\frac{\Gamma; \Sigma; n_{this} \vdash n : T \qquad \Gamma; \Sigma; n_{this} \models n_s \to \mathit{owner}(T) \qquad \mathit{public}(d)}{\Gamma; \Sigma; n_{this} \models n_s \to n.d}\ \textit{T-PublicRef}$$

$$\frac{}{\Gamma; \Sigma; n_{this} \models p \to \texttt{shared}}\ \textit{T-LinkShared}$$

$$\frac{}{\Gamma; \Sigma; n_{this} \models n \to \texttt{shared}}\ \textit{T-LinkSharedRef}$$

Figure 8: Link Permission Rules

The *T-DynamicLink* rule can be used to conclude that two domains are linked if there is an object in the store that explicitly linked them. The *T-DeclaredLink* rule allows the type system to rely on any links that are declared or assumed in the context of the class of $n_{this}$. The *T-ChildRef* rule states that any object named by $n$ has permission to access one of its own domains $n.d$. The *T-SelfLink* rule states that every domain can access itself. The *T-LinkRef* rule allows the object named by $n$ to access a domain if the owner of $n$ can access that domain. The *T-PublicLink* and *T-PublicRef* rules allow objects and domains to access the public domain of some object in a domain they already have access to. *T-ChildLink* rule states that any public domain $d$ of an object named by $n$ has access to a domain $p$ if $d$ is explicitly linked to $p$ based on the links declared in the context of the class of $n$. An object in `shared` may be aliased globally, but may not alias non-`shared` references. Since no class declares the `shared` domain, we use a dummy location $\ell_0$ to refer to it, as in $\ell_0.d_{\texttt{shared}}$. *T-LinkShared* and *T-LinkSharedRef* rules state that any object and any domain has permission to access `shared`.

$$CT(C) = \texttt{class}\ C\texttt{<}\overline{\alpha},\overline{\beta}\texttt{>}\ \texttt{extends}\ C'\texttt{<}\overline{\alpha}\texttt{>}\ \texttt{assumes}\ \overline{\gamma} \to \overline{\delta}\ \{\ \overline{D};\ \overline{L};\ \overline{F};\ K\ \overline{M};\ \}$$

$$\frac{(\texttt{public domain}\ d) \in \overline{D}}{public(d)}\ \textit{Aux-Public}$$

$$\frac{\overline{D} = \overline{\texttt{public}_{opt}\ \texttt{domain}\ d_C} \qquad domains(C'\texttt{<}\overline{p}\texttt{>}) = \overline{d'}}{domains(C\texttt{<}\overline{p},\overline{p'}\texttt{>}) = \overline{this.d_C}, \overline{d'}}\ \textit{Aux-Domains}$$

$$\frac{}{domains(\texttt{Object<}\overline{\alpha_0}\texttt{>}) = \emptyset}\ \textit{Aux-Domains-Obj}$$

$$\frac{\texttt{class}\ C\texttt{<}\overline{\alpha}\texttt{>}}{params(C) = \overline{\alpha}}\ \textit{Aux-Params}$$

$$\frac{T = C\texttt{<}\overline{p}\texttt{>}}{tparams(T) = \overline{p}}\ \textit{Aux-TParams}$$

$$\frac{\overline{L} = \overline{\texttt{link}\ \overline{p_c} \to \overline{p'_c}} \qquad links(C'\texttt{<}\overline{p}\texttt{>}) = \overline{p_s} \to \overline{p'_s}}{links(C\texttt{<}\overline{p},\overline{p'}\texttt{>}) = ([\texttt{that/this}][\overline{p}/\overline{\alpha},\overline{p'}/\overline{\beta}]\ (\overline{p_c} \to \overline{p'_c})),\ \overline{p_s} \to \overline{p'_s}}\ \textit{Aux-Links}$$

$$\frac{assumptions(C'\texttt{<}\overline{p}\texttt{>}) = \overline{p_s} \to \overline{p'_s}}{assumptions(C\texttt{<}\overline{p},\overline{p'}\texttt{>}) = ([\texttt{that/this}][\overline{p}/\overline{\alpha},\overline{p'}/\overline{\beta}]\ (\overline{\gamma} \to \overline{\delta})), \overline{p_s} \to \overline{p'_s}}\ \textit{Aux-Assume}$$

$$\frac{\overline{F} = \overline{T}\ \overline{f} \qquad fields(C'\texttt{<}\overline{p}\texttt{>}) = \overline{T'}\ \overline{f'}}{fields(C\texttt{<}\overline{p},\overline{p'}\texttt{>}) = ([\texttt{that/this}][\overline{p}/\overline{\alpha},\overline{p'}/\overline{\beta}]\ \overline{T}\ \overline{f}), \overline{T'}\ \overline{f'}}\ \textit{Aux-Fields}$$

$$\frac{}{fields(\texttt{Object<}\overline{\alpha_0}\texttt{>}) = \emptyset}\ \textit{Aux-Fields-Obj}$$

$$\frac{}{linkdecls(C\texttt{<}\overline{p}\texttt{>}) = links(C\texttt{<}\overline{p}\texttt{>}) \cup assumptions(C\texttt{<}\overline{p}\texttt{>})}\ \textit{Aux-LinkDecls}$$

$$\frac{}{owner(C\texttt{<}\overline{p}\texttt{>}) = p_1}\ \textit{Aux-Owner}$$

$$\frac{(T_R\ m(\overline{T}\ \overline{x})\ \{\ \texttt{return}\ e;\ \}) \in \overline{M}}{mtype(m, C\texttt{<}\overline{p},\overline{p'}\texttt{>}) = [\texttt{that/this}][\overline{p}/\overline{\alpha},\overline{p'}/\overline{\beta}]\ \overline{T} \to T_R}\ \textit{Aux-MType1}$$

$$\frac{m\ is\ not\ defined\ in\ \overline{M}}{mtype(m, C\texttt{<}\overline{p},\overline{p'}\texttt{>}) = mtype(m, C'\texttt{<}\overline{p}\texttt{>})}\ \textit{Aux-MType2}$$

$$\frac{(T_R\ m(\overline{T}\ \overline{x})\ \{\ \texttt{return}\ e;\ \}) \in \overline{M}}{mbody(m, C\texttt{<}\overline{p}\texttt{>}) = [\texttt{that/this}][\overline{p}/\overline{\alpha}]\ (\overline{x},\ e)}\ \textit{Aux-MBody1}$$

$$\frac{m\ is\ not\ defined\ in\ \overline{M}}{mbody(m, C\texttt{<}\overline{p},\overline{p'}\texttt{>}) = mbody(m, C'\texttt{<}\overline{p}\texttt{>})}\ \textit{Aux-MBody2}$$

$$\frac{(mtype(m, C\texttt{<}\overline{p}\texttt{>}) = \overline{T'} \to T') \Longrightarrow (\overline{T} = \overline{T'} \wedge T = T')}{override(m, C\texttt{<}\overline{p}\texttt{>}, \overline{T} \to T)}\ \textit{Aux-Override}$$



Figure 9: Auxiliary Definitions

Figure 9 shows the definitions of many auxiliary functions used earlier in the semantics. These definitions are straightforward and in many cases are derived directly from rules in Featherweight Java. The Aux-Public rule checks whether a domain is public. The next few rules define the *domains*, *links*, *assumptions*, and $fields$ functions by looking up the declarations in the class and adding them to the declarations in superclasses. The *linkdecls* function just returns the union of the *links* and *assumptions* in a class, while the *owner* function just returns the first domain parameter (which represents the owning domain in our formal system).

The *mtype* function looks up the type of a method in the class; if the method is not present, it looks in the superclass instead. The *mbody* function looks up the body of a method in a similar way. Finally, the *override* function verifies that if a superclass defines method $m$, it has the same type as the definition of $m$ in a subclass.

# 3 Ownership Domains Properties

In this section, we state type soundness and link soundness for Featherweight Domain Java.

**Lemma 1 (Lemma)**

*If* $mtype(m, C'\!<\!\overline{p}\!>) = \overline{T} \to T_R$ *then* $mtype(m, C\!<\!\overline{p}, \overline{p'}\!>) = \overline{T} \to T_R$ *for all* $C\!<\!\overline{p}, \overline{p'}\!> <: C'\!<\!\overline{p}\!>$.

**Proof:** By induction on the derivation of $C\!<\!\overline{p}, \overline{p'}\!> <: C'\!<\!\overline{p}\!>$ and $mtype(m, C'\!<\!\overline{p}\!>)$. ■

**Lemma 2 (Substitution Lemma)**

*If*

$\Gamma, \overline{x} : \overline{T_f}; \Sigma; \texttt{this} \vdash e : T$,

$\Gamma, \overline{x} : \overline{T_f}; \Sigma; \texttt{this} \vdash T$,

$\Gamma; \Sigma; \texttt{that} \vdash \overline{v} : \overline{T_a}$ *where* $\overline{T_a} <: \overline{T_f}$,

$\Gamma; \Sigma; \texttt{that} \vdash \overline{T_a}$

$\Gamma; \Sigma; \texttt{that} \vdash \texttt{that} : T_{that}$

$\Gamma; \Sigma; \texttt{that} \vdash T_{that}$

*then*

$\Gamma; \Sigma; \texttt{that} \vdash [\overline{v}/\overline{x}, \texttt{that}/\texttt{this}]e : T'$ *for some* $T' <: [\overline{v}/\overline{x}, \texttt{that}/\texttt{this}]T$,

$\Gamma; \Sigma; \texttt{that} \vdash T'$

**Proof:** By induction on the typing rules.

■

**Lemma 3 (Weakening Lemma)**

*If* $\Gamma \vdash e : T$, *then* $\Gamma, x : T_x \vdash e : T$.

**Proof:** By induction on the typing rules. ■

**Lemma 4 (Store Lemma)**

*If* $fields(C\!<\!\overline{\ell'.d}\!>) = \overline{T}\ \overline{f}$ *and* $S[\ell] = C\!<\!\overline{\ell'.d}\!>(\overline{v})$ *and* $\emptyset; \Sigma; \theta \vdash \overline{v} : \Sigma[\overline{v}]$

*then* $\Sigma[\overline{v}] <: \overline{T}$ *and* $assumptions(\Sigma[\overline{v}]) \in \Delta$.

**Proof:** Based on rules T-New and R-New.

$$\frac{\begin{array}{c}\Gamma;\Sigma;n_{this} \models assumptions(C\!<\!\overline{p}\!>) \qquad \Gamma;\Sigma;n_{this} \vdash \overline{e} : \overline{T'} \\ fields(C\!<\!\overline{p}\!>) = \overline{T}\ \overline{f} \qquad \overline{T'} <: \overline{T} \qquad \Gamma;\Sigma;n_{this} \vdash n_{this}{:}T_{this} \\ owner(C\!<\!\overline{p}\!>) \in (domains(T_{this}) \cup owner(T_{this})) \\ \Gamma;\Sigma;n_{this} \vdash \overline{T'} \qquad \Gamma;\Sigma;n_{this} \vdash T_{this}\end{array}}{\Gamma;\Sigma;n_{this} \vdash \texttt{new}\ C\!<\!\overline{p}\!>(\overline{e}) : C\!<\!\overline{p}\!>}\ \textit{T-New}$$

$$\frac{\ell \notin domain(S) \qquad S' = S[\ell \mapsto C\!<\!\overline{p}\!>(\overline{v})]}{S;\theta \vdash \texttt{new}\ C\!<\!\overline{p}\!>(\overline{v}) \mapsto \ell, S'}\ \textit{R-New}$$

Take $n_{this} = \theta$

| | |
|---|---|
| $S[\ell] = C\!<\!\overline{p}\!>(\overline{v})$ | By hypothesis |
| $fields(C\!<\!\overline{p}\!> = \overline{T}\ \overline{f}$ | By hypothesis |
| $\emptyset;\Sigma;\theta \vdash \overline{v} : \Sigma[\overline{v}]$ | By hypothesis |
| $\emptyset;\Sigma;\theta \vdash \texttt{new}\ C\!<\!\overline{p}\!>(\overline{v}) : C\!<\!\overline{p}\!>$ | By T-New |
| $\Sigma[\overline{v}] <: \overline{T}$ | By subderivation of T-New |
| $\emptyset;\Sigma;\theta \vdash \Sigma[\overline{v}]$ | By subderivation of T-New |
| $\emptyset;\Sigma;\theta \models assumptions(\Sigma[\overline{v}])$ | By subderivation of T-New |

This concludes the proof.

■

**Lemma 5 (Method Lemma)**

*If* $mtype(m, C\!<\!\overline{p}, \overline{p'}\!>) = \overline{T} \to T_R$

*and* $mbody(m, C\!<\!\overline{p}, \overline{p'}\!>) = (\overline{x}, e_R)$

*then for some* $C'<\overline{p}>$ *with* $C<\overline{p},\overline{p'}> <: C'<\overline{p}>$,

*there exists* $T_0 <: T_R$ *such that* $\overline{x} : \overline{T}, \texttt{this} : C'<\overline{p}> \vdash e_R : T_0$.

**Proof:** By induction on *mtype*.

■

**Lemma 6 (Field's Owner Lemma. NEW!!!)**

*If*

$\texttt{class}\ C<\overline{\alpha},\overline{\beta}>\ \texttt{extends}\ C'<\overline{\alpha}>\ \texttt{assumes}\ \overline{\gamma} \rightarrow \overline{\delta}\ \{\ \overline{D};\ \overline{L};\ \overline{F};\ K\ \overline{M};\}$ *OK,*

*and* $T_i\ f_i \in \overline{F}$

*then*

$owner(T_i) \in \{\overline{\alpha},\overline{\beta}\} \cup \{d \mid d \in domains(C<\overline{\alpha},\overline{\beta}>)\} \cup$

$\cup \{f_j.d_j \mid T_j\ f_j \in \overline{F}, d_j \in domains(T_j), public(d_j)\}$

**Proof:** By ClsOK. ■

**Lemma 7 (Assumptions Owner Lemma. NEW!!!)**

*If*

$\texttt{class}\ C<\overline{\alpha},\overline{\beta}>\ \texttt{extends}\ C'<\overline{\alpha}>\ \texttt{assumes}\ \overline{\gamma} \rightarrow \overline{\delta}\ \{\ \overline{D};\ \overline{L};\ \overline{F};\ K\ \overline{M};\}$ *OK,*

$T_i\ f_i \in \overline{F}$,

*and* $owner(T_i) = \alpha_j \mid \alpha_j \in \{\overline{\alpha},\overline{\beta}\}$

*then* $\alpha_1 \rightarrow \alpha_j \in assumptions(C<\overline{\alpha},\overline{\beta}>)$

**Proof:**

| | |
|---|---|
| $\{\texttt{this} : C<\overline{\alpha}, \overline{\beta}>\};\ \emptyset;\ \texttt{this} \vdash \overline{T}$ | By subderivation of ClsOK |
| $\{\texttt{this} : C<\overline{\alpha}, \overline{\beta}>\};\ \emptyset;\ \texttt{this} \models \texttt{this} \rightarrow owner(\overline{T})$ | By subderivation of T-Type |
| $\{\texttt{this} : C<\overline{\alpha}, \overline{\beta}>\};\ \emptyset;\ \texttt{this} \models owner(C<\overline{\alpha}, \overline{\beta}>) \rightarrow \alpha_j$ | By inversion of T-LinkRef |
| $\{\texttt{this} : C<\overline{\alpha}, \overline{\beta}>\};\ \emptyset;\ \texttt{this} \models \alpha_1 \rightarrow \alpha_j$ | By subdereivation of Aux-Owner |
| $\alpha_1 \rightarrow \alpha_j \in assumptions(C<\overline{\alpha}, \overline{\beta}>) \cup links(C<\overline{\alpha}, \overline{\beta}>)$ | By subderivation of T-DeclaredLink |
| $\alpha_1 \rightarrow \alpha_j \in assumptions(C<\overline{\alpha}, \overline{\beta}>)$ | By LinkOK since $owner(\overline{T}) \notin domains(C<\overline{\alpha}, \overline{\beta}>)$ |

■

**Theorem 8 (Assumptions Theorem. NEW!!!)**

*If*

$\Gamma; \Sigma; n_{this} \vdash \texttt{new}\ C<\overline{p}>(\overline{e}) : C<\overline{p}>$,

$\Gamma; \Sigma; n_{this} \vdash n_{this}{:}T_{this}$,

$p_i \rightarrow p_j \in assumptions(C<\overline{p}>), p_i, p_j \in \overline{p}$

*then*

$p_i \rightarrow p_j \in linkdecls(T_{this})$

**Proof:**

| | |
|---|---|
| $\Gamma; \Sigma; n_{this} \models assumptions(C<\overline{p}>)$ | By subderivation of T-New |
| $assumptions(C<\overline{p}> \in linkdecls(T_{this})$ | By subderivation of T-DeclaredType |
| $p_i \rightarrow p_j \in assumptions(C<\overline{p}>)$ | By hypothesis |
| $p_i \rightarrow p_j \in linkdecls(T_{this})$ | By set inclusion transitivity |

■

This theorem implies that the assumptions can be evaluated locally. This is important from the implementation point of view. It means that an analysis does not have to parse the whole program in order to evaluate the assumptions of a class $C$; it is enough to evaluate the classes where $C$ is instantiated.

**Lemma 9 (Subtype Lemma. NEW!!!)**

*If*

$\Gamma; \Sigma; n_{this} \vdash e : T'$,

$\Gamma; \Sigma; n_{this} \vdash T'$,

$\Gamma; \Sigma; n_{this} \vdash n_{this}{:}T_{this}$,

$\forall T.$ *such that* $T' <: T$

*then*

$\Gamma; \Sigma; n_{this} \vdash T$,

**Proof:** By induction over the derivation of $\emptyset, \Sigma, n_{this} \vdash e : T'$.

| | | |
|---|---|---|
| $\emptyset; \Sigma; n_{this} \vdash e : T'$ | By induction hypothesis | (1) |
| $\emptyset; \Sigma; n_{this} \vdash T'$ | By induction hypothesis | (2) |
| $\emptyset; \Sigma; n_{this} \models n_{this} \rightarrow owner(T')$ | Since (2), by subderivation of T-Type | (3) |
| $T' <: T$ | by hypothesis | (4) |
| $owner(T') = owner(T)$ | By Subtype-Class | (5) |
| $\emptyset; \Sigma; n_{this} \models n_{this} \rightarrow owner(T)$ | Since $owner(T') = owner(T)$ | (6) |
| $\emptyset; \Sigma; n_{this} \models assumptions(T')$ | Since (2), by subderivation of T-Type | (7) |
| $assumptions(T') \in linkdecls(T_{this})$ | By subderivation of T-DeclaredLink | (8) |
| $assumptions(T) \subseteq assumptions(T')$ | By subderivation of Aux-Assume since $T' <: T$ | (9) |
| $assumptions(T) \subseteq linkdecls(T_{this})$ | By transitivity of set inclusion | (10) |
| $\emptyset; \Sigma; n_{this} \models assumptions(T)$ | By inversion of T-DeclaredLink | (11) |
| $\emptyset; \Sigma; n_{this} \vdash T$ | Since (6) and (11), by inversion of T-Type | (12) |

■

Based on the Subtype Lemma, if a type $T'$ is well formed then the base types $T$ is also well formed. However, the vice-versa is not true in general, since there are additional assumptions of $T'$ that are not satisfied. The additional requirements are described in the Type-Strengthening Lemma:

**Lemma 10 (Super Type Lemma. NEW!!!)**

*If*

$\Gamma; \Sigma; \theta \vdash e : T$,

$\Gamma; \Sigma; \theta \vdash T$,

$\Sigma_\Delta \vdash S$,

$S; \theta \vdash e \mapsto \ell \triangleright e', S$

$\Gamma; \Sigma; \theta \vdash \ell \triangleright e' : T'$

$T' <: T$,

*then*

$\Gamma; \Sigma; \theta \vdash T'$

**Proof:** By induction over the derivation of $\emptyset, \Sigma, \theta \vdash e : T$.

| | | |
|---|---|---|
| $\emptyset; \Sigma; \theta \vdash e : T$ | By induction hypothesis | (1) |
| $\emptyset; \Sigma; \theta \vdash T$ | By induction hypothesis | (2) |
| $\emptyset; \Sigma; \theta \models \theta \rightarrow owner(T)$ | Since (2), by subderivation of T-Type | (3) |
| $\emptyset; \Sigma; \theta \models assumptions(T)$ | Since (2), by subderivation of T-Type | (4) |
| $assumptions(T) \subseteq \Delta$ | by subderivation of T-DynamicLink | (5) |
| $T' <: T$ | by hypothesis | (6) |
| $owner(T') = owner(T)$ | By Subtype-Class | (7) |
| $\emptyset; \Sigma; \theta \models \theta \rightarrow owner(T')$ | Since $owner(T') = owner(T)$ | (8) |

| | | |
|---|---|---|
| $\Gamma; \Sigma; \theta \vdash \ell \triangleright e' : T'$ | By induction hypothesis | (9) |
| $\Gamma; \Sigma; \theta \vdash \Sigma[\ell]$ | By subderivation of T-Context | (10) |
| $\Gamma; \Sigma; \ell \vdash e' : T'$ | By subderivation of T-Context | (11) |
| $\Gamma; \Sigma; \ell \vdash T'$ | By subderivation of T-Context | (12) |
| $\Gamma; \Sigma; \theta \models \mathit{assumptions}(\Sigma[\ell])$ | Since (10), by subderivation of T-Type | (13) |
| $\mathit{assumptions}(\Sigma[\ell]) \in \Delta$ | By subderivation of T-DynamicLink | (14) |
| $\Gamma; \Sigma; \ell \models \mathit{assumptions}(T')$ | Since (12), by subderivation of T-Type | (15) |
| $\mathit{assumptions}(T') \in \mathit{linkdecls}(\Sigma[\ell])$ | By subderivation of T-DeclaredLink | (16) |
| $\mathit{linkdecls}(\Sigma[\ell]) = \mathit{links}(\Sigma[\ell]) \cup \mathit{assumptions}(\Sigma[\ell])$ | By Aux-LinkDecls | (17) |
| $\mathit{assumptions}(T') \in \mathit{links}(\Sigma[\ell]) \cup \mathit{assumptions}(\Sigma[\ell])$ | | (18) |
| $\mathit{links}(\Sigma[\ell]) \subseteq \Delta$ | By $\Sigma_\Delta$ $OK$ | (19) |
| $\mathit{assumptions}(\Sigma[\ell]) \subseteq \Delta$ | By (14) | (20) |
| $\mathit{assumptions}(T') \subseteq \Delta$ | | (21) |
| $\Gamma; \Sigma; \theta \models \mathit{assumptions}(T')$ | By inversion of T-DynamicLink | (22) |
| $\Gamma; \Sigma; \theta \vdash T'$ | Since (8) and (22), by inversion of T-Type | (23) |
| This proves the lemma. | | (24) |

Note: in the proof, we showed $\mathit{assumptions}(T) \in \mathit{assumptions}(T') \in \mathit{assumptions}(\Sigma[\theta']) \in \mathit{linkdecls}(\Sigma[\theta])$. At (18), $\mathit{assumptions}(T') \notin \mathit{links}(\Sigma[\theta'])$ because $\mathit{links}(\Sigma[\theta'])$ and $\mathit{assumptions}(\Sigma[\theta']$ are disjoint sets.

■

**Theorem 11 (Type Preservation, a.k.a. Subject Reduction)**

*If*

$\emptyset; \Sigma; \theta \vdash e : T,$

$\emptyset; \Sigma; \theta \vdash T,$

$\Sigma_\Delta \vdash S,$

*and* $S; \theta \vdash e \mapsto e', S'$

*then there exists* $\Sigma' \supseteq \Sigma,\ \Delta' \supseteq \Delta,$ *and* $T' <: T$ *such that:*

$\emptyset, \Sigma', \theta \vdash e' : T',$

$\emptyset, \Sigma', \theta \vdash T',$

*and* $\Sigma'_{\Delta'} \vdash S'.$

**Proof:** By induction over the derivation of $S \vdash e \mapsto e', S'$, with a case analysis on the outermost reduction rule used. We only cover a few rules.

**Case *R-New*:**. Then $e = \texttt{new}\ C{<}\overline{p}{>}(\overline{v}) \qquad e' = \ell$

$$\frac{\ell \notin \mathit{domain}(S) \qquad S' = S[\ell \mapsto C{<}\overline{p}{>}(\overline{v})]}{S; \theta \vdash \texttt{new}\ C{<}\overline{p}{>}(\overline{v}) \mapsto \ell, S'}\ \mathit{R\text{-}New}$$

To show:

$$\emptyset; \Sigma'; \theta \vdash e' : T' \tag{1}$$

$$\emptyset; \Sigma'; \theta \vdash T' \tag{2}$$

$$\Sigma'_{\Delta'} \vdash S' \tag{3}$$

By *T-New*, we have:

$$\frac{\begin{array}{c}\Gamma;\Sigma;n_{this} \models assumptions(C\!<\!\overline{p}\!>) \qquad \Gamma;\Sigma;n_{this} \vdash \overline{e} : \overline{T'} \\ fields(C\!<\!\overline{p}\!>) = \overline{T}\ \overline{f} \qquad \overline{T'} <: \overline{T} \qquad \Gamma;\Sigma;n_{this} \vdash n_{this}{:}T_{this} \\ owner(C\!<\!\overline{p}\!>) \in (domains(T_{this}) \cup owner(T_{this})) \\ \Gamma;\Sigma;n_{this} \vdash \overline{T'} \qquad \Gamma;\Sigma;n_{this} \vdash T_{this}\end{array}}{\Gamma;\Sigma;n_{this} \vdash \texttt{new}\ C\!<\!\overline{p}\!>(\overline{e}) : C\!<\!\overline{p}\!>}\text{[T-New]}$$

Take $\Sigma' = \Sigma[\ell \mapsto C\!<\!\overline{\ell'.d}\!>]$

| | |
|---|---|
| $\emptyset;\Sigma';\theta \vdash e : C\!<\!\overline{\ell'.d}\!>$ | By T-New |
| $\Sigma'[\ell] = C\!<\!\overline{\ell'.d}\!>$ | By definition of $\Sigma'$ |
| $\emptyset;\Sigma';\theta \vdash \ell : C\!<\!\overline{\ell'.d}\!>$ | By inversion of T-Loc |
| $\emptyset;\Sigma';\theta \vdash e' : C\!<\!\overline{\ell'.d}\!>$ | By $e' = \ell$ |
| Take $T' = T = C\!<\!\overline{\ell'.d}\!>$ | This proves (1) |

| | |
|---|---|
| $\Sigma[\theta] = T_{this}$ | By subderivation of T-New |
| $owner(C\!<\!\overline{\ell'.d}\!>) \in owner(\Sigma[\theta]) \cup domains(\Sigma[\theta])$ | By subderivation of T-New |

Subcase $owner(C<\overline{\ell'.d}>) = owner(\Sigma[\theta])$

| | |
|---|---|
| $\emptyset; \Sigma; \theta \models owner(\Sigma[\theta]) \rightarrow owner(\Sigma[\theta])$ | By T-SelfLink |
| $\emptyset; \Sigma; \theta \models \theta \rightarrow owner(\Sigma[\theta])$ | By T-LinkRef |
| $\emptyset; \Sigma; \theta \models \theta \rightarrow owner(C<\overline{\ell'.d}>)$ | By subcase hypothesis |
| $\emptyset; \Sigma; \theta \models assumptions(C<\overline{\ell'.d}>)$ | By subderivation of T-New |
| $\emptyset; \Sigma; \theta \vdash C<\overline{\ell'.d}>$ | By inversion of T-Type |
| $T' = T = C<\overline{\ell'.d}>$ This proves (2). | |

Subcase $owner(C<\overline{\ell'.d}>) \in domains(\Sigma[\theta])$.

Take $\theta.d \in domains(\Sigma[\theta])$, that is $owner(C<\overline{\ell'.d}>) = \theta.d$

| | |
|---|---|
| $\emptyset; \Sigma; \theta \models \theta \rightarrow \theta.d$ | By T-ChildRef |
| $\emptyset; \Sigma; \theta \models \theta \rightarrow owner(C<\overline{\ell'.d}>)$ | By subcase hypothesis |
| $\emptyset; \Sigma; \theta \models assumptions(C<\overline{\ell'.d}>)$ | By subderivation of T-New |
| $\emptyset; \Sigma; \theta \vdash C<\overline{\ell'.d}>$ | By inversion of T-Type |
| $T' = T = C<\overline{\ell'.d}>$ This proves (2). | |

$\Sigma_\Delta \vdash S$ — By i. h.

$domain(S) = domain(\Sigma)$ — By T-Store

$S[\ell_1] = C<\overline{\ell_1'.d}>(\overline{v}) \iff \Sigma[\ell_1] = C<\overline{\ell_1'.d}>$ — By T-Store

$fields(\Sigma[\ell_1]) = \overline{T}\ \overline{f} \Longrightarrow (S[\ell_1, i] = \ell_1'') \wedge (\Sigma[\ell_1''] <: T_i)$ — By T-Store

$\Sigma_\Delta\ OK$ — By T-Store

$(S[\ell_1, i] = \ell_1'') \Longrightarrow (owner(\Sigma[\ell_1]) \rightarrow owner(\Sigma[\ell_1'])) \in \Delta$ — By T-Store

$\forall \ell_2 \in domain(\Sigma)\ links(\Sigma[\ell_2]) \subseteq \Delta$ — By $\Sigma_\Delta\ OK$

$S' = S[\ell \mapsto C<\overline{\ell'.d}>(\overline{v})]$ — Sub-derivation of *R-New*

$domain(S') = domain(S) \cup \{\ell\}$

$domain(\Sigma') = domain(\Sigma) \cup \{\ell\}$

$domain(S') = domain(\Sigma')$ — by T-Store, $domain(S) = domain(\Sigma)$

$S'[\ell] = C<\overline{\ell'.d}>(\overline{v}) \iff \Sigma'[\ell] = C<\overline{\ell'.d}>$ — by T-Store and def. of S' and $\Sigma'$

To show (3):

$$fields(\Sigma'[\ell]) = \overline{T}\ \overline{f} \Longrightarrow (S'[\ell, i] = \ell'') \wedge (\Sigma'[\ell''] <: T_i) \tag{4}$$

$$\Sigma'_{\Delta'}\ OK \tag{5}$$

$$(S'[\ell, i] = \ell'') \Longrightarrow (owner(\Sigma[\ell]) \rightarrow owner(\Sigma'[\ell''])) \in \Delta' \tag{6}$$

$fields(\Sigma'[\ell]) = \overline{T}\ \overline{f}$ — By $\Sigma'[\ell] = C<\overline{\ell'.d}>$

$S'[\ell, i] = v_i\ \Sigma'[v_i] = \Sigma[v_i] = T_i' \Longrightarrow (S'[\ell, i] = v_i) \wedge (\Sigma'[v_i] <: T_i)$ — By Store Lemma

Take $\ell'' = v_i,$ this proves (4).

$\emptyset; \Sigma'; \theta \models assumptions(\Sigma'[\ell])$ By subderivation of T-New

$assumptions(\Sigma'[\ell]) \subseteq \Delta$ By subderivation of T-DynamicLink

Take $\Delta' = \Delta \cup (links(C<\overline{\ell'.d}>)$

$links(\Sigma'[\ell]) \in \Delta'$ Since $\Sigma'[\ell] = C<\overline{\ell'.d}>$

This proves (5).

$S'[\ell, i] = v_i$ By $S' = S[\ell \mapsto C<\overline{p}>(\overline{v})]$

$\Sigma[v_i] = \Sigma'[v_i]$ By $\Sigma' = \Sigma[\ell \mapsto C<\overline{\ell'.d}>]$

$owner(\Sigma'[v_i]) = owner(T_i)$ by $\Sigma'[v_i] <: T_i$

$owner(\Sigma'[v_i]) \in owner(\Sigma'[\ell]) \cup domains(\Sigma'[\ell]) \cup$

$\cup \{\ell'_j.d_j \in \overline{\ell'.d}\} \cup \{v_k.d_k \mid v_k \in \overline{v}, v_k \neq v_i, public(d_k)\}$ By $owner(v_i) = owner(T_i)$ and ClsOK

Subcase: $owner(\Sigma'[v_i]) \in owner(\Sigma'[\ell])$

$\emptyset; \Sigma'; \theta \models owner(\Sigma'[\ell]) \rightarrow owner(\Sigma'[\ell])$ By T-SelfLink

$\emptyset; \Sigma'; \theta \models owner(\Sigma'[\ell]) \rightarrow owner(\Sigma'[v_i])$ By subcase hypothesis

$owner(\Sigma'[\ell]) \rightarrow owner(\Sigma'[v_i]) \in \Delta'$ By subderivation of T-DynamicLink

Take $\ell'' = v_i$. This proves (6)

Subcase: $owner(\Sigma'[v_i]) \in domains(\Sigma'[\ell])$

Take $\ell.d \in domains(\Sigma'[\ell])$

$\emptyset; \Sigma'; \theta \models \ell \to \ell.d$ By T-ChildLink

$\emptyset; \Sigma'; \theta \models owner(\Sigma'[\ell]) \to \ell.d$ By subderivation of T-LinkRef

$\emptyset; \Sigma'; \theta \models owner(\Sigma'[\ell]) \to owner(\Sigma'[v_i])$ By subcase hypothesis

$owner(\Sigma'[\ell]) \to owner(\Sigma'[v_i]) \in \Delta'$ By subderivation of T-DynamicLink

Take $\ell'' = v_i$. This proves (6)

Subcase: $owner(\Sigma'[v_i]) \in \{\ell'_j.d_j \in \overline{\ell'.d}\}$

$owner(\Sigma'[\ell]) \to \ell'_j.d_j \in assumptions(\Sigma'[\ell])$ By Assumptions Owner Lemma

$\emptyset; \Sigma'; \theta \models assumptions(\Sigma'[\ell])$ By subderivation of T-New

$assumptions(\Sigma'[\ell]) \in \Delta'$ By subderivation of T-DynamicLink

$owner(\Sigma'[\ell]) \to \ell'_j.d_j \in \Delta'$ By transitivity of set inclusion

$owner(\Sigma'[\ell]) \to owner(\Sigma[v_i]) \in \Delta'$ By subcase hypothesis

Take $\ell'' = v_i$. This proves (6)

Subcase: $owner(\Sigma'[v_i]) \in \{v_k.d_k \mid v_k \in \overline{v}, v_k \neq v_i, public(d_k)\}$

Assume by T-Store: $\emptyset; \Sigma'; \theta \models owner(\Sigma'[\ell]) \to owner(\Sigma[v_k]), \forall k < i$

$\emptyset; \Sigma'; \theta \models owner(\Sigma'[\ell]) \to owner(\Sigma[v_k]) \quad public(d_k)$ By induction hypothesis

$\emptyset; \Sigma'; \theta \models owner(\Sigma'[\ell]) \to v_k.d_k$ By inversion of T-PublicLink

$\emptyset; \Sigma'; \theta \models owner(\Sigma'[\ell]) \to owner(\Sigma'[v_i])$ By subcase hypothesis

$owner(\Sigma'[\ell]) \to owner(\Sigma'[v_i]) \in \Delta'$ By subderivation of T-DynamicLink

Take $\ell'' = v_i$. This proves (6)

**Case *R-Read*:** Then $e = \ell.f_i \qquad e' = v_i$

$$\frac{S[\ell] = C\overline{<\ell'.d>}(\overline{v}) \qquad fields(C\overline{<\ell'.d>}) = \overline{T}\ \overline{f}}{S; \theta \vdash \ell.f_i \mapsto v_i, S}\ \textit{R-Read}$$

To show:

$$\emptyset; \Sigma'; \theta \vdash e' : T' \tag{7}$$

$$\emptyset; \Sigma'; \theta \vdash T' \tag{8}$$

$$\Sigma'_{\Delta'} \vdash S' \tag{9}$$

By Rule T-Read, we have:

$$\frac{\Gamma; \Sigma; \theta \vdash e_0 : T_0 \qquad fields(T_0) = \overline{T}\ \overline{f} \qquad \Gamma; \Sigma; \theta \vdash T_0}{\Gamma; \Sigma; \theta \vdash e_0.f_i : [e_0/\texttt{that}]T_i}[\text{T-READ}]$$

$S[\ell] = C\overline{<p>}(\overline{v})$ Sub-derivation of R-Read

$S' = S,$ Take $\Sigma' = \Sigma,\ \Delta' = \Delta$

$\Sigma_\Delta \vdash S$ By i. h.

This proves (9).

| | |
|---|---|
| $\emptyset; \Sigma; \theta \vdash \ell : T_0$ | Take $\ell = e_0$ |
| $S[\ell] = C<\overline{p}>(\overline{v}) \iff \Sigma[\ell] = C<\overline{p}>$ | By T-Store |
| $T_0 = C<\overline{p}>$ | By T-Store |
| $fields(T_0) = \overline{T}\ \overline{f}$ | By subderivation of T-Read |
| $\emptyset; \Sigma; \theta \vdash \ell.f_i : [\ell/\texttt{that}]T_i$ | By T-Read |
| $\emptyset; \Sigma; \theta \vdash \overline{v} : \Sigma[\overline{v}],$ where $\Sigma[\overline{v}] <: \overline{T}$ | By Store Lemma |
| $assumptions(\Sigma[\overline{v}]) \subseteq \Delta$ | By Store Lemma |
| $v_i \in \overline{v}$ | |
| $\emptyset; \Sigma; \theta \vdash v_i : \Sigma[v_i]$ | By T-Loc |
| Take $T = [\ell/\texttt{that}]T_i \quad T' = \Sigma[v_i] \quad T' <: T$ | |
| This proves (7) | |

| | |
|---|---|
| $\emptyset; \Sigma; \theta \vdash \ell.f_i : T$ | By induction hypothesis. |
| $\emptyset; \Sigma; \theta \vdash T$ | By induction hypothesis. |
| $\emptyset; \Sigma; \theta \models \theta \rightarrow owner(T)$ | By subderivation of T-Type |
| $\emptyset; \Sigma; \theta \models \theta \rightarrow owner(\Sigma[v_i])$ | $owner(\Sigma[v_i]) = owner(T)$ |
| $assumptions(\Sigma[v_i]) \subseteq \Delta$ | By By Store Lemma |
| $\emptyset; \Sigma; \theta \models assumptions(\Sigma[v_i])$ | By inversion of T-DeclaredLink |
| $\emptyset; \Sigma; \theta \vdash \Sigma[v_i]$ | By inversion of T-Type |
| This proves (8) | |

**Case *R-Write*:** $e = \ell.f_i = v \quad e' = v$

$$\frac{S[\ell] = C{<}\overline{p}{>}(\overline{v}) \qquad S' = S[\ell \mapsto C{<}\overline{p}{>}([v/v_i]\overline{v})]}{S; \theta \vdash \ell.f_i = v \mapsto v,\ S'}\ \textit{R-Write}$$

To show:

$$\emptyset; \Sigma'; \theta \vdash e' : T' \tag{10}$$

$$\emptyset; \Sigma'; \theta \vdash T' \tag{11}$$

$$\Sigma'_{\Delta'} \vdash S' \tag{12}$$

By T-Write:

$$\frac{\Gamma; \Sigma; \theta \vdash e_0 : T_0 \qquad \mathit{fields}(T_0) = \overline{T}\ \overline{f} \qquad T_i \in \overline{T} \qquad \Gamma; \Sigma; \theta \vdash e_R : T_R \qquad T_R <: [e_0/\texttt{that}]T_i \qquad \Gamma; \Sigma; \theta \vdash T_0 \qquad \Gamma; \Sigma; \theta \vdash T_R}{\Gamma; \Sigma; \theta \vdash e_0.f_i = e_R : T_R}\ [\text{T-WRITE}]$$

$\emptyset; \Sigma; \theta \vdash \ell.f_i = v : T_R$ By T-Write

$\emptyset; \Sigma; \theta \vdash v : T_R$ By subderivation of T-Write $v = e_R$

Take $T = T' = T_R \quad T' <: T$

This proves (10)

$\emptyset; \Sigma; \theta \vdash \ell.f_i = v : T_R$ — By T-Write

$\emptyset; \Sigma; \theta \vdash v : T_R$ — By subderivation of T-Write $v = e_R$

$\emptyset; \Sigma'; \theta \vdash T_R$ — By Subderivation of T-Write

$\emptyset; \Sigma'; \theta \vdash \Sigma[v]$ — Since $\Sigma[v] = T_R$

This proves (11) Take $T' = T_R$

$S' = S[\ell \mapsto C<\overline{\ell'.d}>([v/v_i]\overline{v})]$ — By inversion of R-Write

Take $\Sigma' = \Sigma,\ \Delta = \Delta'$

$\Sigma_\Delta \vdash S$ — By i. h.

$domain(S) = domain(\Sigma)$ — By T-Store

$S[\ell_1] = C<\overline{\ell_1'.d}>(\overline{v}) \iff \Sigma[\ell_1] = C<\overline{\ell_1'.d}>$ — By T-Store

$fields(\Sigma[\ell_1]) = \overline{T}\ \overline{f} \Longrightarrow (S[\ell_1, i] = \ell_1'') \wedge (\Sigma[\ell_1''] <: T_i)$ — By T-Store

$\Sigma_\Delta\ OK$ — $\Sigma_\Delta \vdash S$

$(S[\ell_1, i] = \ell_1'') \Longrightarrow (owner(\Sigma[\ell_1]) \rightarrow owner(\Sigma[\ell_1''])) \in \Delta$ — By T-Store

$\forall \ell_2 \in domain(\Sigma)\ links(\Sigma[\ell_2]) \subseteq \Delta$ — By $\Sigma_\Delta\ OK$

$domain(S') = domain(S) = domain(\Sigma) = domain(\Sigma')$

$S'[\ell] = C<\overline{\ell'.d}>([v/v_i]\overline{v}) \iff \Sigma[\ell] = C<\overline{\ell'.d}>$ — Since only $\overline{v}$ changes

$fields(\Sigma'[\ell]) = [\ell/\texttt{that}][\texttt{that}/\texttt{this}][\overline{\ell'.d}/\overline{\alpha}]\overline{T}\ \overline{f} \Longrightarrow$

$(S'[\ell, i] = v) \wedge (\Sigma[v] <: [\ell/\texttt{that}][\texttt{that}/\texttt{this}][\overline{\ell'.d}/\overline{\alpha}]T_i)$ — By T-Write, since $\Sigma[v] = T_R$

$\Sigma'_{\Delta'} OK$ — Since $\Sigma' = \Sigma,\ \Delta' = \Delta$

$S'[\ell, i] = v$ — By $S' = S[\ell \mapsto C{<}\overline{p}{>}([v/v_i]\overline{v})]$

$\Sigma[v] = \Sigma'[v]$ — By By T-Store

$owner(\Sigma'[v]) = owner(T_i)$ — by $\Sigma'[v] <: T_i$

$owner(\Sigma'[v]) \in owner(\Sigma'[\ell]) \cup domains(\Sigma'[\ell]) \cup$

$\cup \{\ell'_j.d_j \in \overline{\ell'.d}\} \cup \{v_k.d_k \mid v_k \in \overline{v}, v_k \neq v, public(d_k)\}$ — By $owner(v) = owner(T_i)$ and ClsOK

Subcase: $owner(\Sigma'[v]) \in owner(\Sigma'[\ell])$

$\emptyset; \Sigma'; \theta \models owner(\Sigma'[\ell]) \to owner(\Sigma'[\ell])$ — By T-SelfLink

$\emptyset; \Sigma'; \theta \models owner(\Sigma'[\ell]) \to owner(\Sigma'[v])$ — By subcase hypothesis

$owner(\Sigma'[\ell]) \to owner(\Sigma'[v]) \in \Delta'$ — By subderivation of T-DynamicLink

Take $\ell'' = v$. This proves (12)

Subcase: $owner(\Sigma'[v]) \in domains(\Sigma'[\ell])$

Take $\ell.d \in domains(\Sigma'[\ell])$

$\emptyset; \Sigma'; \theta \models \ell \to \ell.d$ — By T-ChildLink

$\emptyset; \Sigma'; \theta \models owner(\Sigma'[\ell]) \to \ell.d$ — By subderivation of T-LinkRef

$\emptyset; \Sigma'; \theta \models owner(\Sigma'[\ell]) \to owner(\Sigma'[v_i])$ — By subcase hypothesis

$owner(\Sigma'[\ell]) \to owner(\Sigma'[v]) \in \Delta'$ — By subderivation of T-DynamicLink

Take $\ell'' = v$. This proves (12)

Subcase: $owner(\Sigma'[v]) \in \{\ell'_j.d_j \in \overline{\ell'.d}\}$

| | |
|---|---|
| $owner(\Sigma'[\ell]) \to \ell'_j.d_j \in assumptions(\Sigma'[\ell])$ | By Assumptions Owner Lemma |
| $\emptyset; \Sigma'; \theta \vdash \Sigma[\ell]$ | By subderivation of T-Write |
| $\emptyset; \Sigma'; \theta \models assumptions(\Sigma'[\ell])$ | By subderivation of T-Type |
| $assumptions(\Sigma'[\ell]) \in \Delta'$ | By subderivation of T-DeclaredLink |
| $owner(\Sigma'[\ell]) \to \ell'_j.d_j \in \Delta'$ | By transitivity of set inclusion |
| $owner(\Sigma'[\ell]) \to owner(\Sigma[v]) \in \Delta'$ | By subcase hypothesis |

Take $\ell'' = v$. This proves (12)

Subcase: $owner(\Sigma'[v]) \in \{v_k.d_k \mid v_k \in \overline{v}, v_k \neq v, public(d_k)\}$

Assume by T-Store: $\emptyset; \Sigma'; \theta \models owner(\Sigma'[\ell]) \to owner(\Sigma[v_k]), \forall k < i$

| | |
|---|---|
| $\emptyset; \Sigma'; \theta \models owner(\Sigma'[\ell]) \to owner(\Sigma[v_k]) \quad public(d_k)$ | By induction hypothesis |
| $\emptyset; \Sigma'; \theta \models owner(\Sigma'[\ell]) \to v_k.d_k$ | By inversion of T-PublicLink |
| $\emptyset; \Sigma'; \theta \models owner(\Sigma'[\ell]) \to owner(\Sigma'[v_i])$ | By subcase hypothesis |
| $owner(\Sigma'[\ell]) \to owner(\Sigma'[v]) \in \Delta'$ | By subderivation of T-DynamicLink |

Take $\ell'' = v$. This proves (12)

**Case *R-Invk*:** $e = \ell.m(\overline{v}) \quad e' = \ell \triangleright [\overline{v}/\overline{x}, \ell/\texttt{this}]e_R$

$$\frac{S[\ell] = C\!<\!\overline{p}\!>\!(\overline{v}) \qquad mbody(m, C\!<\!\overline{p}\!>) = (\overline{x}, e_0)}{S; \theta \vdash \ell.m(\overline{v}) \mapsto \ell \triangleright [\overline{v}/\overline{x}, \ell/\texttt{this}]e_0, S}\ \textit{R-Invk}$$

To show:

$$\emptyset; \Sigma'; \theta \vdash e' : T' \tag{13}$$

$$\emptyset; \Sigma'; \theta \vdash T' \tag{14}$$

$$\Sigma'_{\Delta'} \vdash S' \tag{15}$$

$S' = S$, Take $\Sigma' = \Sigma$, $\Delta' = \Delta$

$\Sigma_\Delta \vdash S$ By i. h.

This proves (15).

From *T-Invk*:

$$\frac{\begin{array}{c}\Gamma; \Sigma; \theta \vdash e_0 : T_0 \qquad \Gamma; \Sigma; \theta \vdash \overline{e} : \overline{T_a} \\ mtype(m, T_0) = \overline{T} \to T_R \qquad mbody(m, T_0) = (\overline{x}, e_R) \qquad \overline{T_a} <: [\overline{e}/\overline{x}, e_0/\texttt{that}]\overline{T} \\ \Gamma; \Sigma; \theta \vdash T_0 \qquad \Gamma; \Sigma; \theta \vdash \overline{T_a}\end{array}}{\Gamma; \Sigma; \theta \vdash e_0.m(\overline{e}) : [\overline{e}/\overline{x}, e_0/\texttt{that}]T_R}\ [\text{T-Invk}]$$

$\emptyset; \Sigma'; \theta \vdash \ell.m(\overline{v}) : T$ By induction hypothesis.

$\{\overline{x} : \overline{T},\ \texttt{this} : C\!<\!\overline{\alpha}, \overline{\beta}\!>\},\ \emptyset,\ \texttt{this} \vdash e_R : T_R \qquad T_R <: T$ By MethOK

$S[\ell] = C\!<\!\overline{p}, \overline{p'}\!>(\overline{v})$ By inversion of R-Invk

$mbody(m, C\!<\!\overline{p}, \overline{p'}\!>) = (\overline{x}, e_R)$ By inversion of R-Invk

$e_0 = \ell$

$\Sigma[\ell] = C\!<\!\overline{p}, \overline{p'}\!> = T_0$ T-Store

$\emptyset; \Sigma'; \theta \vdash e_0 : C\!<\!\overline{p}, \overline{p'}\!>$ By subderivation of T-Invk

$mtype(m, C\!<\!\overline{p}, \overline{p'}\!>) = \overline{T} \to T_R$ By subderivation of T-Invk

$\emptyset; \Sigma'; \theta \vdash \overline{v} : \overline{T_a}$ By subderivation of T-Invk

$\overline{T_a} <: [\overline{v}/\overline{x}, \ell/\texttt{this}]\ \overline{T}$ for some $\overline{T_a}$ and $\overline{T}$

There are some $D\!<\!\overline{p}\!>$ and $T'_R$ s.t. $T'_R <: T_R$ and $C\!<\!\overline{p}, \overline{p'}\!> <: D\!<\!\overline{p}\!>$

s.t. $\{\overline{x} : \overline{T}\}; \emptyset, \texttt{this} : D\!<\!\overline{p}\!> \vdash e_R : T'_R$ By Method Lemma

$\{\overline{x} : \overline{T}\}; \Sigma; \texttt{this} : D\!<\!\overline{p}\!> \vdash e_R : T'_R$ By Weakening

$\Gamma; \Sigma; \theta \vdash \overline{v} : \overline{T_a}$ and $\overline{T_a} <: [\overline{v}/\overline{x}, \ell/\texttt{this}]\ \overline{T}$ By Weakening

$\Gamma; \Sigma; \ell \vdash [\overline{v}/\overline{x}, \ell/\texttt{this}]e_R : T_S$ for some $T_S <: [\overline{v}/\overline{x}, \texttt{that}/\texttt{this}]T'_R$ By Substitution Lemma

$T_S <: T'_R$ and $[\overline{v}/\overline{x}, \texttt{that}/\texttt{this}]T'_R <: [\overline{v}/\overline{x}, \texttt{that}/\texttt{this}]T_R$ By above

$T_S <: [\overline{v}/\overline{x}, \texttt{that}/\texttt{this}]T_R$ By transitivity of $<:$

$\emptyset; \Sigma; \theta \vdash \Sigma[\ell]$ By subderivation of T-Invk

$\{\overline{x} : \overline{T};\ \texttt{this} : D\!<\!\overline{p}\!>\};\ \emptyset;\ \texttt{this} : D\!<\!\overline{p}\!> \vdash T'_R$ By MethOK

$\Gamma; \Sigma; \ell \vdash T_S$ By Substitution Lemma

$\Gamma; \Sigma; \ell \vdash [\overline{v}/\overline{x}, \ell/\texttt{this}]T'_R$ By Subtype Lemma

$\emptyset; \Sigma; \theta \vdash \ell \triangleright [\overline{v}/\overline{x}, \ell/\texttt{this}]e_R : T_S$ By inversion of T-Context

Take $T = T_R, T' = T_S$ Preservation

| | |
|---|---|
| $\emptyset; \Sigma; \theta \vdash \ell.m(v) : T$ | By induction hypothesis. |
| $\emptyset; \Sigma; \theta \vdash T$ | By induction hypothesis. |
| $S; \theta \vdash \ell.m(\overline{v}) \mapsto \ell \triangleright [\overline{v}/\overline{x}, \ell/\texttt{this}]e_0, S$ | By induction hypothesis. |
| $\emptyset; \Sigma; \theta \vdash \ell \triangleright [\overline{v}/\overline{x}, \ell/\texttt{this}]e_R : T'$ | By (13) |
| $T' <: T$ | By (13) |
| $\emptyset; \Sigma; \theta \vdash T'$ | By Super Type Lemma |

**Case *R-Context*:** $e = \ell \triangleright v$, $e' = v$

$$\frac{}{S; \theta \vdash \ell \triangleright v \mapsto v, S}\ \textit{R-Context}$$

To Show:

$$\emptyset; \Sigma'; \theta \vdash e' : T' \tag{16}$$

$$\emptyset; \Sigma'; \theta \vdash T' \tag{17}$$

$$\Sigma'_{\Delta'} \vdash S' \tag{18}$$

From *T-Context*:

$$\frac{\Gamma; \Sigma; n_{this} \vdash \Sigma[\ell] \qquad \Gamma; \Sigma; \ell \vdash e : T \qquad \Gamma; \Sigma; \ell \vdash T}{\Gamma; \Sigma; n_{this} \vdash \ell \triangleright e : T}[\text{T-Context}]$$

$S = S'$, Take $\Sigma = \Sigma', \Delta = \Delta'$

| | |
|---|---|
| $\Sigma_\Delta \vdash S$ | By induction hypothesis |

This proves (18).

| | |
|---|---|
| $\emptyset; \Sigma'; \theta \vdash e : C\langle\overline{\ell'.d}\rangle$ | By hypothesis |
| $\emptyset; \Sigma'; \ell \vdash v : C\langle\overline{\ell'.d}\rangle$ | By subderivation of T-Context |
| $\Sigma[v] = C\langle\overline{\ell'.d}\rangle$ | By subderivation of T-Loc |
| $\emptyset; \Sigma'; \theta \vdash C\langle\overline{\ell'.d}\rangle$ | By inversion of T-Loc |

This proves (16). Take $T' = T = C\langle\overline{\ell'.d}\rangle$

| | |
|---|---|
| $\emptyset; \Sigma'; \theta \vdash e : T$ | by induction hypothesis |
| $\emptyset; \Sigma; \theta \vdash T$ | by induction hypothesis |
| $\emptyset; \Sigma; \theta \vdash T'$ | Since $T' = T$ |

This proves (17)

■

**Theorem 12 (Progress)**

*If*

$\emptyset; \Sigma; \theta \vdash e : T$,

$\emptyset; \Sigma; \theta \vdash T$,

*and* $\Sigma_\Delta \vdash S$

— *i.e., e is closed and well-typed,*

*then*

*either e is a value, or*

*else* $S; \theta \vdash e \mapsto e', S'$.

**Proof:** By induction over the derivation of $\emptyset, \Sigma, \theta \vdash e : T$.

**Case *T-Var*:** Then $e = x$.

$$\frac{}{\Gamma; \Sigma; n_{this} \vdash x : \Gamma(x)} \; \textit{T-Var}$$

Not applicable. Variable is not a closed term.

**Case *T-Loc*:** Then $e = \ell$.

$$\frac{\Sigma[\ell] = C\langle\overline{\ell'.d}\rangle}{\emptyset; \Sigma; \theta \vdash \ell : C\langle\overline{\ell'.d}\rangle} \; \textit{T-Loc}$$

$e$ is a value.

**Case *T-New*:** Then $e = \texttt{new}\ C\langle\overline{p}\rangle(\overline{e})$.

$$\frac{\begin{array}{c}\emptyset; \Sigma; \theta \models \mathit{assumptions}(C\langle\overline{p}\rangle) \qquad \emptyset; \Sigma; \theta \vdash \overline{e} : \overline{T'} \\ \mathit{fields}(C\langle\overline{p}\rangle) = \overline{T}\ \overline{f} \qquad \overline{T'} <: \overline{T} \qquad \emptyset; \Sigma; \theta \vdash \theta{:}\Sigma[\theta] \\ \mathit{owner}(C\langle\overline{p}\rangle) \in (\mathit{domains}(\Sigma[\theta]) \cup \mathit{owner}(\Sigma[\theta])) \\ \emptyset; \Sigma; \theta \vdash \overline{T'} \qquad \emptyset; \Sigma; \theta \vdash T_{this}\end{array}}{\emptyset; \Sigma; \theta \vdash \texttt{new}\ C\langle\overline{p}\rangle(\overline{e}) : C\langle\overline{p}\rangle} \; [\text{T-New}]$$

**Subcase** $\overline{e} = \overline{v}$, that is $e = \texttt{new}\ C{<}\overline{p}{>}(\overline{v})$. Then *R-New* applies:

$$\frac{\ell \notin \mathit{domain}(S) \qquad S' = S[\ell \mapsto C{<}\overline{\ell'.d}{>}(\overline{v})]}{S;\theta \vdash \texttt{new}\ C{<}\overline{p}{>}(\overline{v}) \mapsto \ell, S'}\ \textit{R-New}$$

Take $e' = \ell$.

**Subcase** $e = \texttt{new}\ C{<}\overline{p}{>}(v_{1..i-1}, e_i, e_{i+1..n})$. Then, *RC-New* applies.

$$\frac{S;\theta \vdash e_i \mapsto e_i', S'}{S;\theta \vdash C{<}\overline{\ell'.d}{>}(v_{1..i-1}, e_i, e_{i+1..n}) \mapsto C{<}\overline{\ell'.d}{>}(v_{1..i-1}, e_i', e_{i+1..n}), S'}\ \textit{RC-New}$$

Take $e' = C{<}\overline{p}{>}(v_{1..i-1}, e_i', e_{i+1..n})$.

**Case *T-Read*:** Then $e = e_0.f_i$.

$$\frac{\emptyset;\Sigma;\theta \vdash e_0 : T_0 \qquad \mathit{fields}(T_0) = \overline{T}\ \overline{f} \qquad \emptyset;\Sigma;\theta \vdash T_0}{\emptyset;\Sigma;\theta \vdash e_0.f_i : [e_0/\texttt{that}]T_i}[\text{T-READ}]$$

**Subcase** $e_0 = \ell$, that is $e = \ell.f_i$. Then, *R-Read* applies.

$$\frac{S[\ell] = C{<}\overline{\ell'.d}{>}(\overline{v}) \qquad \mathit{fields}(C{<}\overline{\ell'.d}{>}) = \overline{T}\ \overline{f}}{S;\theta \vdash \ell.f_i \mapsto v_i, S}\ \textit{R-Read}$$

Take $e' = v_i$.

**Subcase** $e_0$ is not a value, that is $e = e_0'.f_i$. Then *RC-Read* applies.

$$\frac{S;\theta \vdash e_0' \mapsto e_0'', S'}{S;\theta \vdash e_0'.f_i \mapsto e_0''.f_i, S'}\ \textit{RC-Read}$$

Take $e' = e_0''.f_i$.

**Case *T-Write*:**. Then, $e = (e_0.f_i = e_R)$

$$\frac{\emptyset;\Sigma;\theta \vdash e_0 : T_0 \qquad fields(T_0) = \overline{T}\ \overline{f} \qquad T_i \in \overline{T} \qquad \emptyset;\Sigma;\theta \vdash e_R : T_R \qquad T_R <: [e_0/\texttt{that}]T_i \qquad \emptyset;\Sigma;\theta \vdash T_0 \qquad \emptyset;\Sigma;\theta \vdash T_R}{\emptyset;\Sigma;\theta \vdash e_0.f_i = e_R : T_R}\ [\text{T-WRITE}]$$

**Subcase** $e_0 = \ell$, and $e_R = v$, that is $e = (\ell.f_i = v)$. Then *R-Write* applies.

$$\frac{S[\ell] = C\langle\overline{\ell'.d}\rangle(\overline{v}) \qquad S' = S[\ell \mapsto C\langle\overline{\ell'.d}\rangle([v/v_i]\overline{v})]}{S;\theta \vdash \ell.f_i = v \mapsto v,\ S'}\ \textit{R-Write}$$

Take $e' = v$.

**Subcase** $e_0$ in not a value, that is $e = (e_0'.f_i = e_{arg})$. Then, *RC-RecvWrite* applies.

$$\frac{S;\theta \vdash e_0' \mapsto e_0'', S'}{S;\theta \vdash e_0'.f_i = e_{arg} \mapsto e_0''.f_i = e_{arg}, S'}\ \textit{RC-RecvWrite}$$

Take $e' = (e_0''.f_i = e_{arg})$.

**Subcase** $e_0 = v$ and $e_{arg} = e_{arg}'$, that is $e = (v.f_i = e_{arg}')$. Then, *RC-ArgWrite* applies.

$$\frac{S;\theta \vdash e_{arg}' \mapsto e_{arg}'', S'}{S;\theta \vdash v.f_i = e_{arg}' \mapsto v.f_i = e_{arg}'', S'}\ \textit{RC-ArgWrite}$$

Take $e' = (v.f_i = e_{arg}'')$.

**Case *T-Invk*:** Then, $e = e_0.m(\overline{e})$

$$\frac{\emptyset;\Sigma;\theta \vdash e_0 : T_0 \qquad \emptyset;\Sigma;\theta \vdash \overline{e} : \overline{T_a} \qquad mtype(m, T_0) = \overline{T} \to T_R \qquad mbody(m, T_0) = (\overline{x}, e_R) \qquad \overline{T_a} <: [\overline{e}/\overline{x}, e_0/\texttt{that}]\overline{T} \qquad \emptyset;\Sigma;\theta \vdash T_0 \qquad \emptyset;\Sigma;\theta \vdash \overline{T_a}}{\emptyset;\Sigma;\theta \vdash e_0.m(\overline{e}) : [\overline{e}/\overline{x}, e_0/\texttt{that}]T_R}\ [\text{T-INVK}]$$

**Subcase** $e_0 = \ell$ and $\overline{e} = \overline{v}$, that is $e = \ell.m(\overline{v})$. Then *R-Invk* applies.

$$\frac{S[\ell] = C{<}\overline{\ell'.d}{>}(\overline{v}) \qquad mbody(m, C{<}\overline{\ell'.d}{>}) = (\overline{x}, e_0)}{S; \theta \vdash \ell.m(\overline{v}) \mapsto \ell \triangleright [\overline{v}/\overline{x}, \ell/\texttt{this}]e_0, S} \; \textit{R-Invk}$$

Take $e' = \ell \triangleright [\overline{v}/\overline{x}, \ell/\texttt{this}]e_0$.

**Subcase** $e_0 = e_0'$, that is, $e_0$ is not a value $e = e_0'.m(\overline{e})$. Then, *RC-RecvInvk* applies.

$$\frac{S; \theta \vdash e_0' \mapsto e_0'', S'}{S; \theta \vdash e_0'.m(\overline{e}) \mapsto e_0''.m(\overline{e}), S'} \; \textit{RC-RecvInvk}$$

Take $e' = e_0''.m(\overline{e})$.

**Subcase** $e_0 = \ell$ and $\overline{e} = v_{1..i-1}, e_i, e_{i+1..n}$, that is $e_0$ is a value $e = \ell.m(v_{1..i-1}, e_i, e_{i+1..n})$. Then, *RC-ArgInvk* applies.

$$\frac{S; \theta \vdash e_i \mapsto e_i', S'}{S; \theta \vdash v.m(v_{1..i-1}, e_i, e_{i+1..n}) \mapsto v.m(v_{1..i-1}, e_i', e_{i+1..n}), S'} \; \textit{RC-ArgInvk}$$

Take $e' = \ell.m(v_{1..i-1}, e_i', e_{i+1..n})$.

**Case *T-Cast*:** Then $e = (T)e_0$

$$\frac{\emptyset; \Sigma; \theta \vdash e : T' \qquad \emptyset; \Sigma; \theta \vdash T}{\emptyset; \Sigma; \theta \vdash (T)e : T} [\text{T-CAST}]$$

**Subcase** $e_0 = \ell$, that is $e = (T)\ell$. Then *R-Cast* applies.

$$\frac{S[\ell] = C{<}\overline{\ell'.d}{>}(\overline{v}) \qquad C{<}\overline{\ell'.d}{>} <: T}{S; \theta \vdash (T)\ell \mapsto \ell, S} \; \textit{R-Cast}$$

Take $e' = \ell$.

**Subcase** $e_0$ is not a value, that is $e = (T)e_0'$. Then *RC-Cast* applies.

$$\frac{S; \theta \vdash e_0' \mapsto e_0'', S'}{S; \theta \vdash (T)e_0' \mapsto (T)e_0'', S'} \; \textit{RC-Cast}$$

Take $e' = (T)e_0''$.

**Case *T-Context*:**

$$\frac{\emptyset; \Sigma; \theta \vdash \Sigma[\ell] \qquad \emptyset; \Sigma; \ell \vdash e_0 : T \qquad \emptyset; \Sigma; \ell \vdash T}{\emptyset; \Sigma; \theta \vdash \ell \triangleright e_0 : T}[\text{T-CONTEXT}]$$

**Subcase** $e_0 = v$, that is $e = \ell \triangleright v$. By *R-Context*:

$$\frac{}{S; \theta \vdash \ell \triangleright v \mapsto v, S}\ \textit{R-Context}$$

Take $e' = v$.

**Subcase** $e_0$ is not a value, that is $e = \ell \triangleright e_0'$. Then *RC-Context* applies.

$$\frac{S; \ell \vdash e_0' \mapsto e_0'', S'}{S; \theta \vdash \ell \triangleright e_0' \mapsto \ell \triangleright e_0'', S'}\ \textit{RC-Context}$$

Take $e' = \ell \triangleright e_0''$.

This concludes the progress proof. ■

Together, Type Preservation and Progress imply that the type system for FDJ is sound. We also state a link soundness property for FDJ. First, we define link soundness for the heap: if one object refers to another, then it has permission to do so.

**Theorem 13 (Heap Link Soundness)**
*If* $\Sigma_\Delta \vdash S$ *and* $S[\ell, i] = \ell''$ *then* $owner(\Sigma[\ell]) \rightarrow owner(\Sigma[\ell'']) \in \Delta$.

**Proof:** This property is enforced by the store typing rule *T-Store*. ■

In practice, it is important that link soundness hold not only for field references in the system, but also for expressions in methods. The intuition behind expression link soundness is that if a method with receiver object $n_{this}$ is currently executing, it should only be able to compute with objects that $n_{this}$ has permission to access.

**Theorem 14 (Expression Link Soundness)**

*If* $\emptyset, \Sigma, n_{this} \vdash e : T$, *and* $\emptyset, \Sigma, n_{this} \vdash T$ *and* $T \neq$ `ERROR` *then* $\emptyset, \Sigma, n_{this} \models n_{this} \rightarrow owner(T)$.

**Proof:** This condition is enforced by $\emptyset, \Sigma, n_{this} \vdash T$. ■

As a result of link soundness, developers using ownership domains can be confident that the linking specifications are an accurate representation of run time aliasing in the system.

While extending Ownership Domains(OD), our goal is to minimize implicit rules and baking things in the type system system. Some of the problems we encountered were also mentioned in the OD errata[2].

**Well-formed Types.** In OD, for each rule of type $\Gamma, \Sigma, n_{this} \vdash e : T$, there is an implicit check $T \neq \texttt{ERROR} \Longrightarrow \Gamma, \Sigma, n_{this} \models n_{this} \rightarrow owner(T)$. Instead, in FOD, we introduce well-formed types, and the implicit condition above is now part of T-Type. Each rule $\Gamma, \Sigma, n_{this} \vdash e : T$, rules for well-formed classes and methods (ClsOK, and MethOK) check for well-formed types.

Inspired by SLOD, we consider the following rule for well-formed types ( T-Type-Alt):

*T-Type-Alt*

$$\frac{\Gamma; \Sigma; n_{this} \models n_{this} \rightarrow owner(T) \qquad \Gamma; \Sigma; n_{this} \models owner(T) \rightarrow tparams(T)}{\Gamma; \Sigma; n_{this} \vdash T}$$

One problem with T-Type-Alt is that we require the owner to have access to every domain parameter of $T$. Since domain parameters binds to actual domains, this requirement my lead to a proliferation of domain links, and we may end up requiring every domain to be connected to every domain at the top-level of the OOG. However, if one of the domain parameter is not used as an owner in the body of the class of $T$, this requirement might be unnecessary. Instead of including these restrictions in the type system, we relax the second condition of T-Type-Alt, and require that all the assumptions of the class of $T$ are satisfied. In this way, since developers control the assumptions, they can decide if a type is well-formed. The second condition of T-Type is similar to

[2]Errata available at: `http://www.cs.cmu.edu/~aldrich/papers/ownership-domains-errata.html`

the well-formed type rule in Matthew Smith's thesis [5, Fig. 4.17].

$$\frac{\Gamma; \Sigma; n_{this} \models n_{this} \to owner(T) \qquad \Gamma; \Sigma; n_{this} \models assumptions(T)}{\Gamma; \Sigma; n_{this} \vdash T} \; \textit{T-Type}$$

**Link Permission Rules.** One correction of the OD type system from OD errata suggests duplicating link permission rules [2, Fig. 13], omitting *T-DeclaredLink* from the new set of rules.

This solution leads to additional problems. First in *T-DynamicLink-orig*, the variable $\ell$ is a free variable, and the rules mixes variables from static and dynamic semantics, e.g., $\ell$, and $n_{this}$, where the substitution is somehow implicit. To clarify this confusion, we considered *T-DynamicLink-Alt1* instead of *T-DynamicLink-orig*, where the conclusion is written in terms of dynamic semantics.

Next, since *T-DynamicLink-orig* relies on *links* auxiliary judgement only, while the assumptions are ignored. *T-Assumption-Orig* [2, Fig. 12] requires assumptions of a type to be satisfied. But according to *T-DynamicLink-Alt1* [2, Fig. 12], the assumptions are satisfied if $assumpions(\Sigma[\ell]) \in links(\Sigma[\ell])$. Such a condition cannot be satisfied since $assumptions(\Sigma[\ell])$ and $links(\Sigma[\ell])$ are disjoint sets by definition. Replacing the premise in *T-DynamicLink-Alt1*, so that it uses $linkdecls(\Sigma[\ell])$ instead of $links(\Sigma[\ell])$ is not a solution, since in this case, the condition in *T-Assumption-Orig* is trivially true: $assumptions(\Sigma[\ell]) \in linkdecls(\Sigma[\ell]) = links(\Sigma[\ell]) \cup assumptions(\Sigma[\ell])$

$$\frac{\forall \ell \in domain(\Sigma)\ \emptyset, \Sigma, \ell \models assumptions(\Sigma[\ell])}{\Sigma OK} \; \textit{T-Assumption-Orig}$$

$$\frac{(d_1 \to d_2) \in links(\Sigma[\ell])}{\Gamma, \Sigma, n_{this} \models (d_1 \to d_2)} \; \textit{T-DynamicLink-Orig}$$

$$\frac{(\ell_1.d_1 \to \ell_2.d_2) \in links(\Sigma[\ell])}{\emptyset, \Sigma, \ell \models (\ell_1.d_1 \to \ell_2.d_2)} \; \textit{T-DynamicLink-Alt1}$$

$$\frac{(\ell_1.d_1 \to \ell_2.d_2) \in linkdecls(\Sigma[\ell])}{\emptyset, \Sigma, \ell \models (\ell_1.d_1 \to \ell_2.d_2)} \; \textit{T-DynamicLink-Alt2}$$

The solution is to rely on $\Delta$, where $\Delta$ is the union of all links between runtime domains. By using $\Delta$, we were able to preserve the original idea from OD where there are two link permission rules, *T-DeclaredLink* for static domains, and *T-DynamicLink* for runtime domains. We changed *T-DynamicLink* to use $\Delta$ instead of $links(\Sigma[\ell])$, and *T-Assumptions* to check that $assumptions(\Sigma[\ell]) \in \Delta$. Furthermore, given a store $S$, and $\Sigma$, when a new location $\ell$ is added to $S$, *T-Store* ensures that $links(\Sigma[\ell])$ is in $\Delta$. These changes address the problem in the OD errata, and avoid the confusion in using $\models$ with static and runtime domains.

**The `that`/`this` substitution.** In OD, the *T-Invk* rule substitutes `this` with the receiver $e_0$. It is not clear, however, that `this` refers to the receiver inside the body of the method, and not to the context where the method is invoked. To avoid the confusion, we adopt a solution proposed by Smith, where `that` refers to the receiver inside the body of a method or a class [5, Section 4.6.2.2]. Consequently, the `that`/`this` substitution is explicitly declared in the auxiliary judgments (Fig. 9), and the substitution $[e_0/\texttt{this}]$ from *T-Invk* becomes $[e_0/\texttt{that}]$ (Fig. 6). In addition, we also solved a related problem mentioned in the errata, that *T-Read* should have the substitution $[e_0/\texttt{that}]$ just like *T-Invk*.

# 4 Extensions

*T-New* formalizes the following rule:

- An object $o$ can only create objects in domains declared by $o$, or the `owner` domain of $o$.

We can relax this rule as:

- An object $o$ can only create objects in domains declared by $o$, or the `owner` domain of $o$, or a domain that `owner` has permissions to access as specified in the assumptions of the class.

To specify that owner has permission to access other domains, a developer can specify an assumption `owner ->` $\alpha$, where $\alpha$ is a domain parameter. During the evaluation the analysis substitutes both `owner` and $\alpha$ with actual domains. Also, $o$ can create objects in a domain $n.d$ of another object which $o$ has permissions to access. $d$ may be either a public domain and $o$ has permission to access the owner of $n$, or a private domain and a link declaration which provides the permissions.

In *T-New*, we extend the set that $owner(C<\overline{p}>)$ is part of. The preservation proof changes accordingly, and FDJ remains sound. We highlight the *added premisses* and the *removed premisses*. Note that, $\Gamma;\Sigma;n_{this} \models n_{this} \rightarrow owner(C<\overline{p}>)$ together with $\Gamma;\Sigma;n_{this} \models assumptions(C<\overline{p}>)$ implies $C<\overline{p}>$ is well formed according to *T-Type*. Proving this in preservation becomes trivial from subderivation of *T-New*

$$\frac{\begin{array}{c}\Gamma;\Sigma;n_{this} \models assumptions(C<\overline{p}>) \qquad \Gamma;\Sigma;n_{this} \vdash \overline{e} : \overline{T'} \\ fields(C<\overline{p}>) = \overline{T}\ \overline{f} \qquad \overline{T'} <: \overline{T} \qquad \Gamma;\Sigma;n_{this} \vdash n_{this}{:}T_{this} \\ \Gamma;\Sigma;n_{this} \models n_{this} \rightarrow owner(C<\overline{p}>) \\ owner(C<\overline{p}>) \in (domains(T_{this}) \cup owner(T_{this})) \\ \Gamma;\Sigma;n_{this} \vdash \overline{T'} \qquad \Gamma;\Sigma;n_{this} \vdash T_{this}\end{array}}{\Gamma;\Sigma;n_{this} \vdash \texttt{new}\ C<\overline{p}>(\overline{e}) : C<\overline{p}>}\ [\text{T-New}]$$

Preservation **Proof:**

**Case *R-New*:**. Then $e = \texttt{new}\ C<\overline{p}>(\overline{v}) \qquad e' = \ell$

$$\frac{\ell \notin domain(S) \qquad S' = S[\ell \mapsto C<\overline{p}>(\overline{v})]}{S;\theta \vdash \texttt{new}\ C<\overline{p}>(\overline{v}) \mapsto \ell, S'}\ \textit{R-New}$$

To show:

$$\emptyset; \Sigma'; \theta \vdash e' : T' \tag{1}$$

$$\emptyset; \Sigma'; \theta \vdash T' \tag{2}$$

$$\Sigma'_{\Delta'} \vdash S' \tag{3}$$

Take $\Sigma' = \Sigma[\ell \mapsto C\langle\overline{\ell'.d}\rangle]$

$\emptyset; \Sigma'; \theta \vdash e : C\langle\overline{\ell'.d}\rangle$ By T-New

$\Sigma'[\ell] = C\langle\overline{\ell'.d}\rangle$ By definition of $\Sigma'$

$\emptyset; \Sigma'; \theta \vdash \ell : C\langle\overline{\ell'.d}\rangle$ By inversion of T-Loc

$\emptyset; \Sigma'; \theta \vdash e' : C\langle\overline{\ell'.d}\rangle$ By $e' = \ell$

Take $T' = T = C\langle\overline{\ell'.d}\rangle$ This proves (1)

$\Sigma[\theta] = T_{this}$ By subderivation of T-New

$\emptyset; \Sigma; \theta \models assumptions(C\langle\overline{\ell'.d}\rangle)$ By subderivation of T-New

$\emptyset; \Sigma; \theta \models \theta \rightarrow owner(C\langle\overline{\ell'.d}\rangle)$ By subderivation of T-New

$\emptyset; \Sigma; \theta \vdash C\langle\overline{\ell'.d}\rangle$ By inversion of T-Type

$T' = T = C\langle\overline{\ell'.d}\rangle$ This proves (2).

**BEGIN: REMOVE**

$\Sigma[\theta] = T_{this}$ By subderivation of T-New

$owner(C<\overline{\ell'.d}>) \in owner(\Sigma[\theta]) \cup domains(\Sigma[\theta])$ By subderivation of T-New

Subcase $owner(C<\overline{\ell'.d}>) = owner(\Sigma[\theta])$

$\emptyset; \Sigma; \theta \models owner(\Sigma[\theta]) \rightarrow owner(\Sigma[\theta])$ By T-SelfLink

$\emptyset; \Sigma; \theta \models \theta \rightarrow owner(\Sigma[\theta])$ By T-LinkRef

$\emptyset; \Sigma; \theta \models \theta \rightarrow owner(C<\overline{\ell'.d}>)$ By subcase hypothesis

$\emptyset; \Sigma; \theta \models assumptions(C<\overline{\ell'.d}>)$ By subderivation of T-New

$\emptyset; \Sigma; \theta \vdash C<\overline{\ell'.d}>$ By inversion of T-Type

$T' = T = C<\overline{\ell'.d}>$ This proves (2).

Subcase $owner(C<\overline{\ell'.d}>) \in domains(\Sigma[\theta])$.

Take $\theta.d \in domains(\Sigma[\theta])$, that is $owner(C<\overline{\ell'.d}>) = \theta.d$

$\emptyset; \Sigma; \theta \models \theta \rightarrow \theta.d$ By T-ChildRef

$\emptyset; \Sigma; \theta \models \theta \rightarrow owner(C<\overline{\ell'.d}>)$ By subcase hypothesis

$\emptyset; \Sigma; \theta \models assumptions(C<\overline{\ell'.d}>)$ By subderivation of T-New

$\emptyset; \Sigma; \theta \vdash C<\overline{\ell'.d}>$ By inversion of T-Type

$T' = T = C<\overline{\ell'.d}>$ This proves (2).

**END: REMOVE**

The remainder of the proof does not change.

■

## Acknowledgements

The authors thank Jonathan Aldrich for providing us with many technical insights into the design of the Ownership Domains type system.

## References

[1] J. Aldrich. *Using Types to Enforce Architectural Structure*. PhD thesis, University of Washington, August 2003.

[2] J. Aldrich and C. Chambers. Ownership Domains: Separating Aliasing Policy from Mechanism. In *ECOOP*, 2004.

[3] J. Aldrich and C. Chambers. Ownership Domains: Separating Aliasing Policy from Mechanism. Carnegie Mellon Technical Report CMU-ISRI-04-110, Unpublished, April 2004.

[4] J. Schäfer and A. Poetzsch-Heffter. A Parameterized Type System for Simple Loose Ownership Domains. *Journal of Object Technology*, 6(5), 2007.

[5] M. Smith. *A Model of Effects with an Application to Ownership Types*. PhD thesis, May 2007.