

# Ownership Domains Revisited

Radu Vanciu

Marwan Abi-Antoun

Jonathan Aldrich

June 1, 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 easier to add ownership annotations 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

<b>1</b>	<b>Ownership Domains Revisited</b>	<b>4</b>
<b>2</b>	<b>Formal System</b>	<b>5</b>
2.1	Syntax . . . . .	5
2.2	Dynamic semantics . . . . .	8
2.3	Well Formed Environment and Types . . . . .	9
2.4	Typing Rules . . . . .	10
2.5	Typechecking Rules . . . . .	11
<b>3</b>	<b>Ownership Domains Properties</b>	<b>17</b>
<b>4</b>	<b>Extensions</b>	<b>47</b>

## List of Figures

1	Featherweight Domain Java Abstract Syntax. . . . .	5
2	Dynamic Semantics . . . . .	7
3	Congruence and Error Rules . . . . .	9
4	Well Formed Environment and Types Rules . . . . .	10
5	Subtyping Rules . . . . .	10
6	Typechecking rules. . . . .	11
7	Class, Method and Store Typing. . . . .	13
8	Link Permission Rules . . . . .	14
9	Auxiliary Definitions . . . . .	16

## List of Symbols

$CL$	class declaration . . . . .	5
$D$	domain declaration . . . . .	5
$L$	link declaration . . . . .	5
$F$	field declaration . . . . .	5
$K$	constructor declaration . . . . .	5
$M$	method declaration . . . . .	5
$\bar{v}$	overbar denotes sequence . . . . .	5
$\mapsto$	maps to . . . . .	5
$\rightarrow$	permission link . . . . .	5
$C$	class name . . . . .	6
$T$	static type . . . . .	6
$f$	field name . . . . .	6
$v$	value . . . . .	6
$e$	expression . . . . .	6
$x$	variable name . . . . .	6
$d$	domain name . . . . .	6
$n$	values or variable name . . . . .	6
$p$	domain . . . . .	6
$S$	store map . . . . .	6
$\ell$	location in store . . . . .	6
$n_{this}$	name or value of the current context . . . . .	6
$\theta$	location representing the current context . . . . .	6
$\alpha$	formal domain parameter . . . . .	6
$\beta$	formal domain parameter . . . . .	6
$\gamma$	formal domain parameter . . . . .	6
$\delta$	formal domain parameter . . . . .	6
$m$	method name . . . . .	6
$\triangleright$	change of context . . . . .	7
$z$	variable, including <b>this</b> . . . . .	7
<b>this</b>	reference to the current context . . . . .	7
<b>that</b>	reference to the context of a receiver . . . . .	7
$CT$	class table . . . . .	7
$\Sigma$	Store type . . . . .	7
$\ell'.d$	runtime domain . . . . .	7
$\mapsto$	an expression reduces to another expression . . . . .	8
$\mapsto^*$	reflexive, transitive closure of $\mapsto$ . . . . .	8
$\vdash \Diamond$	well formed environment . . . . .	9
$\vdash T$	well formed type . . . . .	9
$\Gamma$	type environment . . . . .	9
$<:$	subtyping relation . . . . .	10
$\vdash e : T$	typechecking relation . . . . .	11
$\Delta$	union of all links between runtime domains . . . . .	13
$\models n \rightarrow p$	link permission relation between an object and a domain . . . . .	13
$\models p \rightarrow p'$	link permission relation between two domains . . . . .	13

# 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<sup>1</sup> 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  $T \neq \text{ERROR} \implies \Gamma; \Sigma; n_{this} \models n_{this} \rightarrow \text{owner}(T)$ . 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  $\text{links}(\Sigma[\ell])$ .
- 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 changes the context from  $n_{this}$  to  $\ell$ .
- 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 annotations to legacy code that was written without ownership in mind.

---

<sup>1</sup>Errata available at: <http://www.cs.cmu.edu/~aldrich/papers/ownership-domains-errata.html>

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

Figure 1: Featherweight Domain Java Abstract Syntax.

## 2 Formal System

**Assumptions.** The model makes the following assumptions:

- It does not formalize `lent`, `unique`;

### 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  $S[\ell \mapsto C\langle\bar{p}\rangle(\bar{v})]$ .



$$\begin{array}{c}
\frac{\ell \notin \text{domain}(S) \quad S' = S[\ell \mapsto C\langle\overline{\ell'}.d\rangle(\overline{v})]}{S; \theta \vdash \mathbf{new} \ C\langle\overline{\ell'}.d\rangle(\overline{v}) \mapsto \ell, S'} \text{ R-New} \\
\\
\frac{S[\ell] = C\langle\overline{\ell'}.d\rangle(\overline{v}) \quad \text{fields}(C\langle\overline{\ell'}.d\rangle) = \overline{T} \ \overline{f}}{S; \theta \vdash \ell.f_i \mapsto v_i, S} \text{ R-Read} \\
\\
\frac{S[\ell] = C\langle\overline{\ell'}.d\rangle(\overline{v}) \quad \text{fields}(C\langle\overline{\ell'}.d\rangle) = \overline{T} \ \overline{f} \quad S' = S[\ell \mapsto C\langle\overline{\ell'}.d\rangle([v/v_i]\overline{v})]}{S; \theta \vdash \ell.f_i = v \mapsto v, S'} \text{ R-Write} \\
\\
\frac{S[\ell] = C\langle\overline{\ell'}.d\rangle(\overline{v}) \quad \text{mbody}(m, C\langle\overline{\ell'}.d\rangle) = (\overline{x}, e_0)}{S; \theta \vdash \ell.m(\overline{v}) \mapsto \ell \triangleright [\overline{v}/\overline{x}, \ell/\mathbf{this}]e_0, S} \text{ R-Invk} \\
\\
\frac{S[\ell] = C\langle\overline{\ell'}.d\rangle(\overline{v}) \quad C\langle\overline{\ell'}.d\rangle <: T}{S; \theta \vdash (T)\ell \mapsto \ell, S} \text{ R-Cast} \\
\\
\frac{S[\ell] = C\langle\overline{\ell'}.d\rangle(\overline{v}) \quad C\langle\overline{\ell'}.d\rangle \not<: T}{S; \theta \vdash (T)\ell \mapsto \mathbf{error}, S} \text{ E-Cast} \\
\\
\frac{}{S; \theta \vdash \ell \triangleright v \mapsto v, S} \text{ R-Context}
\end{array}$$

Figure 2: Dynamic Semantics

- $\ell \triangleright e$  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  $T_{\mathbf{this}}$  used to refer to the current context, and **that** used to refer to the context of a receiver.
- 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  $C\langle\overline{p}\rangle$  and a runtime type  $C\langle\overline{\ell'}.d\rangle$ .

## 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  $\bar{v}$  for variables  $\bar{x}$  in  $e$  is written  $[\bar{v}/\bar{x}]e$ . Execution of the method body continues in the context of the receiver location.

$$\begin{array}{c}
\frac{S; \theta \vdash e_i \mapsto e'_i, S'}{S; \theta \vdash C \langle \overline{\ell'}.d \rangle (v_{1..i-1}, e_i, e_{i+1..n}) \mapsto C \langle \overline{\ell'}.d \rangle (v_{1..i-1}, e'_i, e_{i+1..n}), S'} \text{ RC-New} \\
\\
\frac{S; \theta \vdash e \mapsto e', S'}{S; \theta \vdash e.f_i \mapsto e'.f_i, S'} \text{ RC-Read} \\
\\
\frac{S; \theta \vdash e \mapsto e', S'}{S; \theta \vdash e.f_i = e_{arg} \mapsto e'.f_i = e_{arg}, S'} \text{ RC-RecvWrite} \\
\\
\frac{S; \theta \vdash e \mapsto e', S'}{S; \theta \vdash v.f_i = e \mapsto v.f_i = e', S'} \text{ RC-ArgWrite} \\
\\
\frac{S; \theta \vdash e \mapsto e', S'}{S; \theta \vdash e.m(\overline{e}) \mapsto e'.m(\overline{e}), S'} \text{ RC-RecvInvk} \\
\\
\frac{S; \theta \vdash e_i \mapsto e'_i, S'}{S; \theta \vdash v.m(v_{1..i-1}, e_i, e_{i+1..n}) \mapsto v.m(v_{1..i-1}, e'_i, e_{i+1..n}), S'} \text{ RC-ArgInvk} \\
\\
\frac{S; \theta \vdash e \mapsto e', S'}{S; \theta \vdash (T)e \mapsto (T)e', S'} \text{ RC-Cast} \\
\\
\frac{S; \ell \vdash e \mapsto e', S'}{S; \theta \vdash \ell \triangleright e \mapsto \ell \triangleright e', S'} \text{ RC-Context}
\end{array}$$

Figure 3: Congruence and Error Rules

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.

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

$$\begin{array}{c}
\frac{}{\emptyset; \Sigma; n_{this} \vdash \Diamond} \textit{T-Env-Empty} \\
\\
\frac{\Gamma; \Sigma; n_{this} \vdash T \quad x \notin \text{dom}(\Gamma)}{\Gamma, x : T; \Sigma; n_{this} \vdash \Diamond} \textit{T-Env-X} \\
\\
\frac{\Gamma; \Sigma; n_{this} \models n_{this} \rightarrow \text{owner}(T) \quad \Gamma; \Sigma; n_{this} \models \text{assumptions}(T)}{\Gamma; \Sigma; n_{this} \vdash T} \textit{T-Type}
\end{array}$$

Figure 4: Well Formed Environment and Types Rules

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  $\Gamma; \Sigma; n_{this} \models \text{assumptions}(C\langle\bar{p}\rangle)$  as a shorthand for  $\forall (p_1 \rightarrow p_2) \in \text{assumptions}(C\langle\bar{p}\rangle), \Gamma; \Sigma; n_{this} \models p_1 \rightarrow p_2$ . To avoid duplication, we also use *T-Type* to check that dynamic types  $C\langle\bar{\ell}'.d\rangle$  are well formed.

$$\begin{array}{c}
\frac{CT(C) = \mathbf{class} \ C\langle\bar{\alpha}, \bar{\beta}\rangle \ \mathbf{extends} \ C'\langle\bar{\alpha}\rangle \ \dots}{C\langle\bar{p}, \bar{p}'\rangle <: C'\langle\bar{p}\rangle} \textit{Subtype-Class} \\
\\
\frac{}{\overline{T} <: T} \textit{Subtype-Reflex} \\
\\
\frac{T <: T' \quad T' <: T''}{T <: T''} \textit{Subtype-Trans} \\
\\
\frac{}{\text{ERROR} <: T} \textit{Subtype-Error}
\end{array}$$

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

$$\begin{array}{c}
\frac{\Gamma; \Sigma; n_{this} \models \text{assumptions}(C \langle \bar{p} \rangle) \quad \Gamma; \Sigma; n_{this} \vdash \bar{e} : \bar{T}' \quad \text{fields}(C \langle \bar{p} \rangle) = \bar{T} \ \bar{f} \quad \bar{T}' <: \bar{T} \quad \Gamma; \Sigma; n_{this} \vdash n_{this} : T_{this} \quad \text{owner}(C \langle \bar{p} \rangle) \in (\text{domains}(T_{this}) \cup \text{owner}(T_{this}))}{\Gamma; \Sigma; n_{this} \vdash \bar{T}' \quad \Gamma; \Sigma; n_{this} \vdash T_{this}} \text{[T-NEW]} \\
\Gamma; \Sigma; n_{this} \vdash \mathbf{new} \ C \langle \bar{p} \rangle (\bar{e}) : C \langle \bar{p} \rangle \\
\\
\frac{\Gamma; \Sigma; n_{this} \vdash e_0 : T_0 \quad \text{fields}(T_0) = \bar{T} \ \bar{f}}{\Gamma; \Sigma; n_{this} \vdash T_0} \text{[T-READ]} \\
\Gamma; \Sigma; n_{this} \vdash e_0.f_i : [e_0/\mathbf{that}]T_i \\
\\
\frac{\Gamma; \Sigma; n_{this} \vdash e_0 : T_0 \quad \text{fields}(T_0) = \bar{T} \ \bar{f} \quad T_i \in \bar{T} \quad \Gamma; \Sigma; n_{this} \vdash e_R : T_R \quad T_R <: [e_0/\mathbf{that}]T_i}{\Gamma; \Sigma; n_{this} \vdash T_0 \quad \Gamma; \Sigma; n_{this} \vdash T_R} \text{[T-WRITE]} \\
\Gamma; \Sigma; n_{this} \vdash e_0.f_i = e_R : T_R \\
\\
\frac{\Gamma; \Sigma; n_{this} \vdash e_0 : T_0 \quad \Gamma; \Sigma; n_{this} \vdash \bar{e} : \bar{T}_a \quad \text{mtype}(m, T_0) = \bar{T} \rightarrow T_R \quad \text{mbody}(m, T_0) = (\bar{x}, e_R) \quad \bar{T}_a <: [\bar{e}/\bar{x}, e_0/\mathbf{that}]\bar{T}}{\Gamma; \Sigma; n_{this} \vdash T_0 \quad \Gamma; \Sigma; n_{this} \vdash \bar{T}_a} \text{[T-INVK]} \\
\Gamma; \Sigma; n_{this} \vdash e_0.m(\bar{e}) : [\bar{e}/\bar{x}, e_0/\mathbf{that}]T_R \\
\\
\frac{\Gamma(x) = C \langle \bar{p} \rangle}{\Gamma; \Sigma; n_{this} \vdash x : C \langle \bar{p} \rangle} \text{[T-VAR]} \quad \frac{\Sigma(\ell) = C \langle \bar{p} \rangle}{\Gamma; \Sigma; n_{this} \vdash \ell : C \langle \bar{p} \rangle} \text{[T-LOC]} \\
\\
\frac{\Gamma; \Sigma; n_{this} \vdash e : T' \quad \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] \quad \Gamma; \Sigma; \ell \vdash e : T \quad \Gamma; \Sigma; \ell \vdash T}{\Gamma; \Sigma; n_{this} \vdash \ell \triangleright e : T} \text{[T-CONTEXT]}
\end{array}$$

Figure 6: Typechecking rules.

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.

## 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 instanti-

ated 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 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

$$\begin{array}{c}
\frac{\overline{M} \text{ OK in } C \quad \text{fields}(C' \langle \overline{\alpha} \rangle) = \overline{T'} \ \overline{g} \quad \overline{L} \text{ OK in } C \langle \overline{\alpha}, \overline{\beta} \rangle \quad \overline{F} = \overline{T} \ \overline{f} \quad \{\mathbf{this} : C \langle \overline{\alpha}, \overline{\beta} \rangle; \emptyset; \mathbf{this} \vdash \overline{T}\} \quad K = C(\overline{T'} \ \overline{g}, \overline{T} \ \overline{f}) \{ \text{super}(\overline{g}); \text{this}.\overline{f} = \overline{f}; \}}{\text{class } C \langle \overline{\alpha}, \overline{\beta} \rangle \text{ extends } C' \langle \overline{\alpha} \rangle \text{ assumes } \overline{\gamma} \rightarrow \overline{\delta} \{ \overline{D}; \overline{L}; \overline{F}; K \ \overline{M}; \} \text{ OK}} \text{ClsOK} \\
\\
\frac{\begin{array}{c} CT(C) = \text{class } C \langle \overline{\alpha}, \overline{\beta} \rangle \text{ extends } C' \langle \overline{\alpha} \rangle \dots \\ \text{override}(m, C' \langle \overline{\alpha} \rangle, \overline{T} \rightarrow T_R) \\ \{\overline{x} : \overline{T}; \mathbf{this} : C \langle \overline{\alpha}, \overline{\beta} \rangle; \emptyset; \mathbf{this} \vdash e : T'_R\} \\ \{\overline{x} : \overline{T}; \mathbf{this} : C \langle \overline{\alpha}, \overline{\beta} \rangle; \emptyset; \mathbf{this} \vdash T'_R\} \\ \{\overline{x} : \overline{T}; \mathbf{this} : C \langle \overline{\alpha}, \overline{\beta} \rangle; \emptyset; \mathbf{this} \vdash T'_R \quad T'_R <: T_R\} \\ \{\overline{x} : \overline{T}; \mathbf{this} : C \langle \overline{\alpha}, \overline{\beta} \rangle; \emptyset; \mathbf{this} \vdash \overline{T}\} \end{array}}{T_R \ m(\overline{T} \ \overline{x}) \{ \text{return } e; \} \text{ OK in } C} \text{MethOK} \\
\\
\frac{\begin{array}{c} \{p_1, p_2\} \cap \text{domains}(C \langle \overline{\alpha} \rangle) \neq \emptyset \\ p_1 \notin \text{domains}(C \langle \overline{\alpha} \rangle) \implies (\mathbf{this} : C \langle \overline{\alpha} \rangle; \emptyset; \mathbf{this} \models p_1 \rightarrow \text{owner}(C \langle \overline{\alpha} \rangle)) \\ p_2 \notin \text{domains}(C \langle \overline{\alpha} \rangle) \implies (\mathbf{this} : C \langle \overline{\alpha} \rangle; \emptyset; \mathbf{this} \models \mathbf{this} \rightarrow p_2) \end{array}}{\text{link } p_1 \rightarrow p_2 \text{ OK in } C \langle \overline{\alpha} \rangle} \text{LinkOK} \\
\\
\frac{\forall \ell \in \text{domain}(\Sigma) \ (\text{links}(\Sigma[\ell])) \subseteq \Delta}{\Sigma_\Delta \text{ OK}} \text{T-Assumptions} \\
\\
\frac{\begin{array}{c} \text{domain}(S) = \text{domain}(\Sigma) \quad S[\ell] = C \langle \overline{\ell'}.d \rangle(\overline{v}) \iff \Sigma[\ell] = C \langle \overline{\ell'}.d \rangle \\ \text{fields}(\Sigma[\ell]) = \overline{T} \ \overline{f} \implies (S[\ell, i] = \ell'') \wedge (\Sigma[\ell''] <: T_i) \\ \Sigma_\Delta \text{ OK} \\ (S[\ell, i] = \ell'') \implies (\text{owner}(\Sigma[\ell]) \rightarrow \text{owner}(\Sigma[\ell''])) \in \Delta \end{array}}{\Sigma_\Delta \vdash S} \text{T-Store}
\end{array}$$

Figure 7: Class, Method and Store Typing.

check verifies link soundness for the store: if object  $\ell$  refers to object  $\ell''$  in its  $i$ 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  $\Sigma[\ell'']$ .

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

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:  $\Gamma; \Sigma; n_{\text{this}} \models n \rightarrow p$  and  $\Gamma; \Sigma; n_{\text{this}} \models p \rightarrow p'$ . The first form of rule is read, “Given the type environment  $\Gamma$ , the store type

$$\begin{array}{c}
\frac{(\ell_1.d_1 \rightarrow \ell_2.d_2) \in \Delta}{\Gamma; \Sigma; n_{this} \models \ell_1.d_1 \rightarrow \ell_2.d_2} \textit{T-DynamicLink} \\
\\
\frac{\Gamma; \Sigma; n_{this} \vdash n_{this} : T \quad (p_1 \rightarrow p_2) \in \textit{linkdecls}(T)}{\Gamma; \Sigma; n_{this} \models p_1 \rightarrow p_2} \textit{T-DeclaredLink} \\
\\
\frac{}{\Gamma; \Sigma; n_{this} \models n \rightarrow n.d} \textit{T-ChildRef} \qquad \frac{}{\Gamma; \Sigma; n_{this} \models p \rightarrow p} \textit{T-SelfLink} \\
\\
\frac{\Gamma; \Sigma; n_{this} \vdash n : T \quad \Gamma; \Sigma; n_{this} \models \textit{owner}(T) \rightarrow p}{\Gamma; \Sigma; n_{this} \models n \rightarrow p} \textit{T-LinkRef} \\
\\
\frac{\Gamma; \Sigma; n_{this} \vdash n : T \quad \Gamma; \Sigma; n_{this} \models p \rightarrow \textit{owner}(T) \quad \textit{public}(d)}{\Gamma; \Sigma; n_{this} \models p \rightarrow n.d} \textit{T-PublicLink} \\
\\
\frac{\Gamma; \Sigma; n_{this} \vdash n : T \quad \Gamma; \Sigma; n_{this} \models n_s \rightarrow \textit{owner}(T) \quad \textit{public}(d)}{\Gamma; \Sigma; n_{this} \models n_s \rightarrow n.d} \textit{T-PublicRef} \\
\\
\frac{}{\Gamma; \Sigma; n_{this} \models p \rightarrow \textbf{shared}} \textit{T-LinkShared} \\
\\
\frac{}{\Gamma; \Sigma; n_{this} \models n \rightarrow \textbf{shared}} \textit{T-LinkSharedRef}
\end{array}$$

Figure 8: Link Permission Rules

$\Sigma$ , 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.

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  $\ell_0.d_{\text{shared}}$ . *T-LinkShared* and *T-LinkSharedRef* rules state that any object and any domain has permission to access **shared**.

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.

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

$$\begin{array}{c}
\frac{(\text{public domain } d) \in \bar{D}}{\text{public}(d)} \text{ Aux-Public} \\
\\
\frac{\bar{D} = \overline{\text{public}_{opt} \text{ domain } d_C} \quad \text{domains}(C' < \bar{p} >) = \bar{d}'}{\text{domains}(C < \bar{p}, \bar{p}' >) = \overline{\text{this}.d_C}, \bar{d}'} \text{ Aux-Domains} \\
\\
\frac{}{\text{domains}(\text{Object} < \bar{\alpha}_0 >) = \emptyset} \text{ Aux-Domains-Obj} \\
\\
\frac{\text{class } C < \bar{\alpha} >}{\text{params}(C) = \bar{\alpha}} \text{ Aux-Params} \\
\\
\frac{T = C < \bar{p} >}{\text{tparams}(T) = \bar{p}} \text{ Aux-TParams} \\
\\
\frac{\bar{L} = \overline{\text{link}} \bar{p}_c \rightarrow \bar{p}'_c \quad \text{links}(C' < \bar{p} >) = \bar{p}_s \rightarrow \bar{p}'_s}{\text{links}(C < \bar{p}, \bar{p}' >) = ([\text{that}/\text{this}][\bar{p}/\bar{\alpha}, \bar{p}'/\bar{\beta}] (\bar{p}_c \rightarrow \bar{p}'_c)), \bar{p}_s \rightarrow \bar{p}'_s} \text{ Aux-Links} \\
\\
\frac{\text{assumptions}(C' < \bar{p} >) = \bar{p}_s \rightarrow \bar{p}'_s}{\text{assumptions}(C < \bar{p}, \bar{p}' >) = ([\text{that}/\text{this}][\bar{p}/\bar{\alpha}, \bar{p}'/\bar{\beta}] (\bar{\gamma} \rightarrow \bar{\delta})), \bar{p}_s \rightarrow \bar{p}'_s} \text{ Aux-Assume} \\
\\
\frac{\bar{F} = \overline{T} \bar{f} \quad \text{fields}(C' < \bar{p} >) = \bar{T}' \bar{f}'}{\text{fields}(C < \bar{p}, \bar{p}' >) = ([\text{that}/\text{this}][\bar{p}/\bar{\alpha}, \bar{p}'/\bar{\beta}] \bar{T} \bar{f}), \bar{T}' \bar{f}'} \text{ Aux-Fields} \\
\\
\frac{}{\text{fields}(\text{Object} < \bar{\alpha}_0 >) = \emptyset} \text{ Aux-Fields-Obj} \\
\\
\frac{}{\text{linkdecls}(C < \bar{p} >) = \text{links}(C < \bar{p} >) \cup \text{assumptions}(C < \bar{p} >)} \text{ Aux-LinkDecls} \\
\\
\frac{}{\text{owner}(C < \bar{p} >) = p_1} \text{ Aux-Owner} \\
\\
\frac{(T_R \ m(\bar{T} \ \bar{x}) \ \{ \text{return } e; \}) \in \bar{M}}{\text{mtype}(m, C < \bar{p}, \bar{p}' >) = [\text{that}/\text{this}][\bar{p}/\bar{\alpha}, \bar{p}'/\bar{\beta}] \bar{T} \rightarrow T_R} \text{ Aux-MType1} \\
\\
\frac{m \text{ is not defined in } \bar{M}}{\text{mtype}(m, C < \bar{p}, \bar{p}' >) = \text{mtype}(m, C' < \bar{p} >)} \text{ Aux-MType2} \\
\\
\frac{(T_R \ m(\bar{T} \ \bar{x}) \ \{ \text{return } e; \}) \in \bar{M}}{\text{mbody}(m, C < \bar{p} >) = [\text{that}/\text{this}][\bar{p}/\bar{\alpha}] (\bar{x}, e)} \text{ Aux-MBody1} \\
\\
\frac{m \text{ is not defined in } \bar{M}}{\text{mbody}(m, C < \bar{p}, \bar{p}' >) = \text{mbody}(m, C' < \bar{p} >)} \text{ Aux-MBody2} \\
\\
\frac{(\text{mtype}(m, C < \bar{p} >) = \bar{T}' \rightarrow T') \implies (\bar{T} = \bar{T}' \wedge T = T')}{\text{override}(m, C < \bar{p} >, \bar{T} \rightarrow T)} \text{ Aux-Override}
\end{array}$$

16  
Figure 9: Auxiliary Definitions

### 3 Ownership Domains Properties

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

**Lemma 1 (Lemma)**

If  $mtype(m, C' < \bar{p} >) = \bar{T} \rightarrow T_R$  then  $mtype(m, C < \bar{p}, \bar{p}' >) = \bar{T} \rightarrow T_R$  for all  $C < \bar{p}, \bar{p}' > <: C' < \bar{p} >$ .

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

**Lemma 2 (Substitution Lemma)**

If

$\Gamma, \bar{x} : \bar{T}_f; \Sigma; \text{this} \vdash e : T,$

$\Gamma, \bar{x} : \bar{T}_f; \Sigma; \text{this} \vdash T,$

$\Gamma; \Sigma; \text{that} \vdash \bar{v} : \bar{T}_a$  where  $\bar{T}_a <: \bar{T}_f,$

$\Gamma; \Sigma; \text{that} \vdash \bar{T}_a$

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

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

then

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

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

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

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

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

$$\begin{array}{c}
\text{\textit{T-New}} \\
\frac{\Gamma; \Sigma; n_{this} \models \text{\textit{assumptions}}(C<\bar{p}>) \quad \Gamma; \Sigma; n_{this} \vdash \bar{e} : \bar{T}' \quad \text{\textit{fields}}(C<\bar{p}>) = \bar{T} \ \bar{f} \quad \bar{T}' <: \bar{T} \quad \Gamma; \Sigma; n_{this} \vdash n_{this} : T_{this} \quad \text{\textit{owner}}(C<\bar{p}>) \in (\text{\textit{domains}}(T_{this}) \cup \text{\textit{owner}}(T_{this}))}{\Gamma; \Sigma; n_{this} \vdash \bar{T}' \quad \Gamma; \Sigma; n_{this} \vdash T_{this}} \\
\hline
\Gamma; \Sigma; n_{this} \vdash \text{\textbf{new}} \ C<\bar{p}>(\bar{e}) : C<\bar{p}> \\
\\
\text{\textit{R-New}} \\
\frac{\ell \notin \text{\textit{domain}}(S) \quad S' = S[\ell \mapsto C<\bar{p}>(\bar{v})]}{S; \theta \vdash \text{\textbf{new}} \ C<\bar{p}>(\bar{v}) \mapsto \ell, S'}
\end{array}$$

Take  $n_{this} = \theta$

$S[\ell] = C<\bar{p}>(\bar{v})$  By hypothesis

$\text{\textit{fields}}(C<\bar{p}>) = \bar{T} \ \bar{f}$  By hypothesis

$\emptyset; \Sigma; \theta \vdash \bar{v} : \Sigma[\bar{v}]$  By hypothesis

$\emptyset; \Sigma; \theta \vdash \text{\textbf{new}} \ C<\bar{p}>(\bar{v}) : C<\bar{p}>$  By T-New

$\Sigma[\bar{v}] <: \bar{T}$  By subderivation of T-New

$\emptyset; \Sigma; \theta \vdash \Sigma[\bar{v}]$  By subderivation of T-New

$\emptyset; \Sigma; \theta \models \text{\textit{assumptions}}(\Sigma[\bar{v}])$  By subderivation of T-New

This concludes the proof. ■

### Lemma 5 (Method Lemma)

If  $mtype(m, C<\bar{p}, \bar{p}'>) = \bar{T} \rightarrow T_R$

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

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

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

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

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

*If*

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

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

*then*

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

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

**Proof:** By ClsOK. ■

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

*If*

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

*$T_i \ f_i \in \bar{F}$ ,*

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

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

**Proof:**

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

■

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

*If*

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

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

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

*then*

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

**Proof:**

$\Gamma; \Sigma; n_{this} \models \mathit{assumptions}(C<\bar{p}>)$	By subderivation of T-New
$\mathit{assumptions}(C<\bar{p}>) \in \mathit{linkdecls}(T_{this})$	By subderivation of T-DeclaredType
$p_i \rightarrow p_j \in \mathit{assumptions}(C<\bar{p}>)$	By hypothesis
$p_i \rightarrow p_j \in \mathit{linkdecls}(T_{this})$	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*

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

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

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

$$\forall T. \text{ such that } T' <: T$$

*then*

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

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

$$\emptyset; \Sigma; n_{this} \vdash e : T' \quad \text{By induction hypothesis} \quad (1)$$

$$\emptyset; \Sigma; n_{this} \vdash T' \quad \text{By induction hypothesis} \quad (2)$$

$$\emptyset; \Sigma; n_{this} \models n_{this} \rightarrow owner(T') \quad \text{Since (2), by subderivation of T-Type} \quad (3)$$

$$T' <: T \quad \text{by hypothesis} \quad (4)$$

$$owner(T') = owner(T) \quad \text{By Subtype-Class} \quad (5)$$

$$\emptyset; \Sigma; n_{this} \models n_{this} \rightarrow owner(T) \quad \text{Since } owner(T') = owner(T) \quad (6)$$

$$\emptyset; \Sigma; n_{this} \models assumptions(T') \quad \text{Since (2), by subderivation of T-Type} \quad (7)$$

$$assumptions(T') \in linkdecls(T_{this}) \quad \text{By subderivation of T-DeclaredLink} \quad (8)$$

$$assumptions(T) \subseteq assumptions(T') \quad \text{By subderivation of Aux-Assume since } T' <: T \quad (9)$$

$$assumptions(T) \subseteq linkdecls(T_{this}) \quad \text{By transitivity of set inclusion} \quad (10)$$

$$\emptyset; \Sigma; n_{this} \models assumptions(T) \quad \text{By inversion of T-DeclaredLink} \quad (11)$$

$$\emptyset; \Sigma; n_{this} \vdash T \quad \text{Since (6) and (11), by inversion of T-Type} \quad (12)$$

■

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

$$\emptyset; \Sigma; \theta \vdash e : T \quad \text{By induction hypothesis} \quad (1)$$

$$\emptyset; \Sigma; \theta \vdash T \quad \text{By induction hypothesis} \quad (2)$$

$$\emptyset; \Sigma; \theta \models \theta \rightarrow \text{owner}(T) \quad \text{Since (2), by subderivation of T-Type} \quad (3)$$

$$\emptyset; \Sigma; \theta \models \text{assumptions}(T) \quad \text{Since (2), by subderivation of T-Type} \quad (4)$$

$$\text{assumptions}(T) \subseteq \Delta \quad \text{by subderivation of T-DynamicLink} \quad (5)$$

$$T' <: T \quad \text{by hypothesis} \quad (6)$$

$$\text{owner}(T') = \text{owner}(T) \quad \text{By Subtype-Class} \quad (7)$$

$$\emptyset; \Sigma; \theta \models \theta \rightarrow \text{owner}(T') \quad \text{Since } \text{owner}(T') = \text{owner}(T) \quad (8)$$

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

Note: in the proof, we showed  $\text{assumptions}(T) \in \text{assumptions}(T') \in \text{assumptions}(\Sigma[\theta']) \in \text{linkdecls}(\Sigma[\theta])$ .

At (18),  $\text{assumptions}(T') \notin \text{links}(\Sigma[\theta'])$  because  $\text{links}(\Sigma[\theta'])$  and  $\text{assumptions}(\Sigma[\theta'])$  are disjoint sets.

■

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

*If*

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

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

$$\Sigma_{\Delta} \vdash S,$$

$$\text{and } S; \theta \vdash e \mapsto e', S'$$

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

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

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

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

**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  $e = \text{new } C < \bar{p} > (\bar{v})$        $e' = \ell$

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

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:

$$\begin{array}{c}
\Gamma; \Sigma; n_{this} \models \text{assumptions}(C \langle \bar{p} \rangle) \quad \Gamma; \Sigma; n_{this} \vdash \bar{e} : \bar{T}' \\
\text{fields}(C \langle \bar{p} \rangle) = \bar{T} \ \bar{f} \quad \bar{T}' <: \bar{T} \quad \Gamma; \Sigma; n_{this} \vdash n_{this} : T_{this} \\
\text{owner}(C \langle \bar{p} \rangle) \in (\text{domains}(T_{this}) \cup \text{owner}(T_{this})) \\
\hline
\boxed{\Gamma; \Sigma; n_{this} \vdash \bar{T}'} \quad \boxed{\Gamma; \Sigma; n_{this} \vdash T_{this}} \quad \text{[T-NEW]} \\
\Gamma; \Sigma; n_{this} \vdash \mathbf{new} \ C \langle \bar{p} \rangle (\bar{e}) : C \langle \bar{p} \rangle
\end{array}$$

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

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

By T-New

$\Sigma'[\ell] = C \langle \bar{\ell}'.d \rangle$

By definition of  $\Sigma'$

$\emptyset; \Sigma'; \theta \vdash \ell : C \langle \bar{\ell}'.d \rangle$

By inversion of T-Loc

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

By  $e' = \ell$

Take  $T' = T = C \langle \bar{\ell}'.d \rangle$

This proves (1)

$\Sigma[\theta] = T_{this}$

By subderivation of T-New

$\text{owner}(C \langle \bar{\ell}'.d \rangle) \in \text{owner}(\Sigma[\theta]) \cup \text{domains}(\Sigma[\theta])$

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
$S[\ell_1] = C<\overline{\ell'_1}.d>(\bar{v}) \iff \Sigma[\ell_1] = C<\overline{\ell'_1}.d>$	By T-Store
$fields(\Sigma[\ell_1]) = \bar{T} \bar{f} \implies (S[\ell_1, i] = \ell''_1) \wedge (\Sigma[\ell''_1] <: T_i)$	By T-Store
$\Sigma_\Delta OK$	By T-Store
$(S[\ell_1, i] = \ell''_1) \implies (owner(\Sigma[\ell_1]) \rightarrow owner(\Sigma[\ell''_1])) \in \Delta$	By T-Store
$\forall \ell_2 \in domain(\Sigma) \text{ links}(\Sigma[\ell_2]) \subseteq \Delta$	By $\Sigma_\Delta OK$
$S' = S[\ell \mapsto C<\overline{\ell'}.d>(\bar{v})]$	Sub-derivation of <i>R-New</i>
$domain(S') = domain(S) \cup \{\ell\}$	
$domain(\Sigma') = domain(\Sigma) \cup \{\ell\}$	
$domain(S') = domain(\Sigma')$	by T-Store, $domain(S) = domain(\Sigma)$
$S'[\ell] = C<\overline{\ell'}.d>(\bar{v}) \iff \Sigma'[\ell] = C<\overline{\ell'}.d>$	by T-Store and def. of $S'$ and $\Sigma'$

To show (3):

$$fields(\Sigma'[\ell]) = \bar{T} \bar{f} \implies (S'[\ell, i] = \ell'') \wedge (\Sigma'[\ell''] <: T_i) \quad (4)$$

$$\Sigma'_{\Delta'} OK \quad (5)$$

$$(S'[\ell, i] = \ell'') \implies (owner(\Sigma[\ell]) \rightarrow owner(\Sigma'[\ell''])) \in \Delta' \quad (6)$$

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

$$S'[\ell, i] = v_i \quad \Sigma'[v_i] = \Sigma[v_i] = T'_i \implies (S'[\ell, i] = v_i) \wedge (\Sigma'[v_i] <: T_i) \quad \text{By Store Lemma}$$

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

$$\emptyset; \Sigma'; \theta \models \text{assumptions}(\Sigma'[\ell])$$

By subderivation of T-New

$$\text{assumptions}(\Sigma'[\ell]) \subseteq \Delta$$

By subderivation of T-DynamicLink

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

$$\text{links}(\Sigma'[\ell]) \in \Delta'$$

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

$$\text{owner}(\Sigma'[v_i]) = \text{owner}(T_i)$$

by  $\Sigma'[v_i] <: T_i$

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

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

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

$$\emptyset; \Sigma'; \theta \models \text{owner}(\Sigma'[\ell]) \rightarrow \text{owner}(\Sigma'[\ell])$$

By T-SelfLink

$$\emptyset; \Sigma'; \theta \models \text{owner}(\Sigma'[\ell]) \rightarrow \text{owner}(\Sigma'[v_i])$$

By subcase hypothesis

$$\text{owner}(\Sigma'[\ell]) \rightarrow \text{owner}(\Sigma'[v_i]) \in \Delta'$$

By subderivation of T-DynamicLink

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

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

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

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

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

$\emptyset; \Sigma'; \theta \models owner(\Sigma'[\ell]) \rightarrow owner(\Sigma'[v_i])$  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:  $owner(\Sigma'[v_i]) \in \{\ell'_j.d_j \in \overline{\ell'.d}\}$

$owner(\Sigma'[\ell]) \rightarrow \ell'_j.d_j \in assumptions(\Sigma'[\ell])$  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

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

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

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

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

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

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

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

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



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

$$\frac{S[\ell] = C\langle \overline{\ell'.d} \rangle(\bar{v}) \quad fields(C\langle \overline{\ell'.d} \rangle) = \overline{T} \ \overline{f}}{S; \theta \vdash \ell.f_i \mapsto v_i, S} \text{ } R\text{-Read}$$

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:

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

$$S[\ell] = C\langle \overline{p} \rangle(\bar{v}) \quad \text{Sub-derivation of R-Read}$$

$$S' = S, \text{ Take } \Sigma' = \Sigma, \Delta' = \Delta$$

$$\Sigma_{\Delta} \vdash S \quad \text{By i. h.}$$

This proves (9).

$\emptyset; \Sigma; \theta \vdash \ell : T_0$	Take $\ell = e_0$
$S[\ell] = C\langle \bar{p} \rangle(\bar{v}) \iff \Sigma[\ell] = C\langle \bar{p} \rangle$	By T-Store
$T_0 = C\langle \bar{p} \rangle$	By T-Store
$fields(T_0) = \bar{T} \bar{f}$	By subderivation of T-Read
$\emptyset; \Sigma; \theta \vdash \ell.f_i : [\ell/\mathbf{that}]T_i$	By T-Read
$\emptyset; \Sigma; \theta \vdash \bar{v} : \Sigma[\bar{v}]$ , where $\Sigma[\bar{v}] <: \bar{T}$	By Store Lemma
$assumptions(\Sigma[\bar{v}]) \subseteq \Delta$	By Store Lemma
$v_i \in \bar{v}$	
$\emptyset; \Sigma; \theta \vdash v_i : \Sigma[v_i]$	By T-Loc
Take $T = [\ell/\mathbf{that}]T_i \quad T' = \Sigma[v_i] \quad T' <: T$	
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*:**  $e = \ell.f_i = v \quad e' = v$

$$\frac{S[\ell] = C\langle \bar{p} \rangle(\bar{v}) \quad S' = S[\ell \mapsto C\langle \bar{p} \rangle([v/v_i]\bar{v})]}{S; \theta \vdash \ell.f_i = v \mapsto v, S'} \text{ *R-Write*}$$

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:

$$\begin{array}{c} \Gamma; \Sigma; \theta \vdash e_0 : T_0 \quad fields(T_0) = \bar{T} \bar{f} \quad T_i \in \bar{T} \quad \Gamma; \Sigma; \theta \vdash e_R : T_R \quad T_R <: [e_0/\mathbf{that}]T_i \\ \hline \Gamma; \Sigma; \theta \vdash T_0 \quad \Gamma; \Sigma; \theta \vdash T_R \\ \hline \Gamma; \Sigma; \theta \vdash e_0.f_i = e_R : T_R \end{array} \text{ [T-WRITE]}$$

$$\emptyset; \Sigma; \theta \vdash \ell.f_i = v : T_R$$

By T-Write

$$\emptyset; \Sigma; \theta \vdash v : T_R$$

By subderivation of T-Write  $v = e_R$

$$\text{Take } T = T' = T_R \quad T' <: T$$

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  $T' = T_R$

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

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

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

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

$S[\ell_1] = C\langle \overline{\ell'_1.d} \rangle(\bar{v}) \iff \Sigma[\ell_1] = C\langle \overline{\ell'_1.d} \rangle$  By T-Store

$fields(\Sigma[\ell_1]) = \bar{T} \bar{f} \implies (S[\ell_1, i] = \ell''_1) \wedge (\Sigma[\ell''_1] <: T_i)$  By T-Store

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

$(S[\ell_1, i] = \ell''_1) \implies (owner(\Sigma[\ell_1]) \rightarrow owner(\Sigma[\ell''_1])) \in \Delta$  By T-Store

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

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

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

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

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

$\Sigma'_\Delta OK$  Since  $\Sigma' = \Sigma, \Delta' = \Delta$

$$\begin{array}{ll}
S'[\ell, i] = v & \text{By } S' = S[\ell \mapsto C \langle \bar{p} \rangle ([v/v_i] \bar{v})] \\
\Sigma[v] = \Sigma'[v] & \text{By By T-Store} \\
owner(\Sigma'[v]) = owner(T_i) & \text{by } \Sigma'[v] <: T_i \\
owner(\Sigma'[v]) \in owner(\Sigma'[\ell]) \cup domains(\Sigma'[\ell]) \cup \\
\cup \{\ell'_j.d_j \in \overline{\ell'.d}\} \cup \{v_k.d_k \mid v_k \in \bar{v}, v_k \neq v, public(d_k)\} & \text{By } owner(v) = owner(T_i) \text{ and ClsOK}
\end{array}$$

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

$$\begin{array}{ll}
\emptyset; \Sigma'; \theta \models owner(\Sigma'[\ell]) \rightarrow owner(\Sigma'[\ell]) & \text{By T-SelfLink} \\
\emptyset; \Sigma'; \theta \models owner(\Sigma'[\ell]) \rightarrow owner(\Sigma'[v]) & \text{By subcase hypothesis} \\
owner(\Sigma'[\ell]) \rightarrow owner(\Sigma'[v]) \in \Delta' & \text{By subderivation of T-DynamicLink} \\
\text{Take } \ell'' = v. \text{ This proves (12)} &
\end{array}$$

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

$$\begin{array}{ll}
\text{Take } \ell.d \in domains(\Sigma'[\ell]) & \\
\emptyset; \Sigma'; \theta \models \ell \rightarrow \ell.d & \text{By T-ChildLink} \\
\emptyset; \Sigma'; \theta \models owner(\Sigma'[\ell]) \rightarrow \ell.d & \text{By subderivation of T-LinkRef} \\
\emptyset; \Sigma'; \theta \models owner(\Sigma'[\ell]) \rightarrow owner(\Sigma'[v_i]) & \text{By subcase hypothesis} \\
owner(\Sigma'[\ell]) \rightarrow owner(\Sigma'[v]) \in \Delta' & \text{By subderivation of T-DynamicLink} \\
\text{Take } \ell'' = v. \text{ This proves (12)} &
\end{array}$$

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

$owner(\Sigma'[\ell]) \rightarrow \ell'_j.d_j \in assumptions(\Sigma'[\ell])$  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

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

$owner(\Sigma'[\ell]) \rightarrow owner(\Sigma[v]) \in \Delta'$  By subcase hypothesis

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

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

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

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

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

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

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

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

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

$$\frac{S[\ell] = C\langle \bar{p} \rangle(\bar{v}) \quad mbody(m, C\langle \bar{p} \rangle) = (\bar{x}, e_0)}{S; \theta \vdash \ell.m(\bar{v}) \mapsto \ell \triangleright [\bar{v}/\bar{x}, \ell/\mathbf{this}]e_0, S} \text{ } R\text{-Invk}$$

To show:

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

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

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

$$S' = S, \text{ Take } \Sigma' = \Sigma, \Delta' = \Delta$$

$$\Sigma_{\Delta} \vdash S$$

By i. h.

This proves (15).

From *T-Invk*:

$$\begin{array}{c} \Gamma; \Sigma; \theta \vdash e_0 : T_0 \quad \Gamma; \Sigma; \theta \vdash \bar{e} : \overline{T_a} \\ mtype(m, T_0) = \overline{T} \rightarrow T_R \quad mbody(m, T_0) = (\bar{x}, e_R) \quad \overline{T_a} <: [\bar{e}/\bar{x}, e_0/\mathbf{that}]\overline{T} \\ \hline \Gamma; \Sigma; \theta \vdash T_0 \quad \Gamma; \Sigma; \theta \vdash \overline{T_a} \\ \hline \Gamma; \Sigma; \theta \vdash e_0.m(\bar{e}) : [\bar{e}/\bar{x}, e_0/\mathbf{that}]T_R \end{array} \text{ } [T\text{-INVK}]$$

$\emptyset; \Sigma'; \theta \vdash \ell.m(\bar{v}) : T$	By induction hypothesis.
$\{\bar{x} : \bar{T}, \text{this} : C\langle\bar{\alpha}, \bar{\beta}\rangle\}, \emptyset, \text{this} \vdash e_R : T_R \quad T_R <: T$	By MethOK
$S[\ell] = C\langle\bar{p}, \bar{p}'\rangle(\bar{v})$	By inversion of R-Invk
$mbody(m, C\langle\bar{p}, \bar{p}'\rangle) = (\bar{x}, e_R)$	By inversion of R-Invk
$e_0 = \ell$	
$\Sigma[\ell] = C\langle\bar{p}, \bar{p}'\rangle = T_0$	T-Store
$\emptyset; \Sigma'; \theta \vdash e_0 : C\langle\bar{p}, \bar{p}'\rangle$	By subderivation of T-Invk
$mtype(m, C\langle\bar{p}, \bar{p}'\rangle) = \bar{T} \rightarrow T_R$	By subderivation of T-Invk
$\emptyset; \Sigma'; \theta \vdash \bar{v} : \bar{T}_a$	By subderivation of T-Invk
$\bar{T}_a <: [\bar{v}/\bar{x}, \ell/\text{this}] \bar{T}$	for some $\bar{T}_a$ and $\bar{T}$
There are some $D\langle\bar{p}\rangle$ and $T'_R$ s.t. $T'_R <: T_R$ and $C\langle\bar{p}, \bar{p}'\rangle <: D\langle\bar{p}\rangle$	
s.t. $\{\bar{x} : \bar{T}\}; \emptyset, \text{this} : D\langle\bar{p}\rangle \vdash e_R : T'_R$	By Method Lemma
$\{\bar{x} : \bar{T}\}; \Sigma; \text{this} : D\langle\bar{p}\rangle \vdash e_R : T'_R$	By Weakening
$\Gamma; \Sigma; \theta \vdash \bar{v} : \bar{T}_a$ and $\bar{T}_a <: [\bar{v}/\bar{x}, \ell/\text{this}] \bar{T}$	By Weakening
$\Gamma; \Sigma; \ell \vdash [\bar{v}/\bar{x}, \ell/\text{this}]e_R : T_S$ for some $T_S <: [\bar{v}/\bar{x}, \text{that}/\text{this}]T'_R$	By Substitution Lemma
$T_S <: T'_R$ and $[\bar{v}/\bar{x}, \text{that}/\text{this}]T'_R <: [\bar{v}/\bar{x}, \text{that}/\text{this}]T_R$	By above
$T_S <: [\bar{v}/\bar{x}, \text{that}/\text{this}]T_R$	By transitivity of $<:$
$\emptyset; \Sigma; \theta \vdash \Sigma[\ell]$	By subderivation of T-Invk
$\{\bar{x} : \bar{T}; \text{this} : D\langle\bar{p}\rangle\}; \emptyset; \text{this} : D\langle\bar{p}\rangle \vdash T'_R$	By MethOK
$\Gamma; \Sigma; \ell \vdash T_S$	By Substitution Lemma
$\Gamma; \Sigma; \ell \vdash [\bar{v}/\bar{x}, \ell/\text{this}]T'_R$	By Subtype Lemma
$\emptyset; \Sigma; \theta \vdash \ell \triangleright [\bar{v}/\bar{x}, \ell/\text{this}]e_R : T_S$	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.
$S; \theta \vdash \ell.m(\bar{v}) \mapsto \ell \triangleright [\bar{v}/\bar{x}, \ell/\mathbf{this}]_{e_0}, S$	By induction hypothesis.
$\emptyset; \Sigma; \theta \vdash \ell \triangleright [\bar{v}/\bar{x}, \ell/\mathbf{this}]_{e_R} : T'$	By (13)
$T' <: T$	By (13)
$\emptyset; \Sigma; \theta \vdash T'$	By Super Type Lemma

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

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

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*:

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

$$S = S', \text{ Take } \Sigma = \Sigma', \Delta = \Delta'$$

$$\Sigma_{\Delta} \vdash S \quad \text{By induction hypothesis}$$

This proves (18).

$$\emptyset; \Sigma'; \theta \vdash e : C < \overline{\ell'}. \vec{d} >$$

By hypothesis

$$\emptyset; \Sigma'; \ell \vdash v : C < \overline{\ell'}. \vec{d} >$$

By subderivation of T-Context

$$\Sigma[v] = C < \overline{\ell'}. \vec{d} >$$

By subderivation of T-Loc

$$\emptyset; \Sigma'; \theta \vdash C < \overline{\ell'}. \vec{d} >$$

By inversion of T-Loc

This proves (16). Take  $T' = T = C < \overline{\ell'}. \vec{d} >$

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

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*

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

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

$$\text{and } \Sigma_{\Delta} \vdash S$$

— 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  $T\text{-Var}$ :** Then  $e = x$ .

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

Not applicable. Variable is not a closed term.

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

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

$e$  is a value.

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

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

**Subcase**  $\bar{e} = \bar{v}$ , that is  $e = \mathbf{new} \ C \langle \bar{p} \rangle (\bar{v})$ . Then *R-New* applies:

$$\frac{\ell \notin \text{domain}(S) \quad S' = S[\ell \mapsto C \langle \bar{\ell}'.d \rangle (\bar{v})]}{S; \theta \vdash \mathbf{new} \ C \langle \bar{p} \rangle (\bar{v}) \mapsto \ell, S'} \quad R\text{-New}$$

Take  $e' = \ell$ .

**Subcase**  $e = \mathbf{new} \ C \langle \bar{p} \rangle (v_{1..i-1}, e_i, e_{i+1..n})$ . Then, *RC-New* applies.

$$\frac{S; \theta \vdash e_i \mapsto e'_i, S'}{S; \theta \vdash C \langle \bar{\ell}'.d \rangle (v_{1..i-1}, e_i, e_{i+1..n}) \mapsto C \langle \bar{\ell}'.d \rangle (v_{1..i-1}, e'_i, e_{i+1..n}), S'} \quad RC\text{-New}$$

Take  $e' = C \langle \bar{p} \rangle (v_{1..i-1}, e'_i, e_{i+1..n})$ .

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

$$\frac{\emptyset; \Sigma; \theta \vdash e_0 : T_0 \quad \text{fields}(T_0) = \bar{T} \ \bar{f} \quad \boxed{\emptyset; \Sigma; \theta \vdash T_0}}{\emptyset; \Sigma; \theta \vdash e_0.f_i : [e_0/\mathbf{that}]T_i} \quad [\text{T-READ}]$$

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

$$\frac{S[\ell] = C \langle \bar{\ell}'.d \rangle (\bar{v}) \quad \text{fields}(C \langle \bar{\ell}'.d \rangle) = \bar{T} \ \bar{f}}{S; \theta \vdash \ell.f_i \mapsto v_i, S} \quad R\text{-Read}$$

Take  $e' = v_i$ .

**Subcase**  $e_0$  is not a value, that is  $e = e'_0.f_i$ . Then *RC-Read* applies.

$$\frac{S; \theta \vdash e'_0 \mapsto e''_0, S'}{S; \theta \vdash e'_0.f_i \mapsto e''_0.f_i, S'} \quad RC\text{-Read}$$

Take  $e' = e''_0.f_i$ .

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

$$\begin{array}{c}
\emptyset; \Sigma; \theta \vdash e_0 : T_0 \quad \text{fields}(T_0) = \overline{T} \ \overline{f} \quad T_i \in \overline{T} \quad \emptyset; \Sigma; \theta \vdash e_R : T_R \quad T_R <: [e_0/\mathbf{that}]T_i \\
\hline
\emptyset; \Sigma; \theta \vdash T_0 \quad \emptyset; \Sigma; \theta \vdash T_R \\
\hline
\emptyset; \Sigma; \theta \vdash e_0.f_i = e_R : T_R
\end{array} \text{[T-WRITE]}$$

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

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

Take  $e' = v$ .

**Subcase**  $e_0$  is not a value, that is  $e = (e'_0.f_i = e_{arg})$ . Then, *RC-RecvWrite* applies.

$$\frac{S; \theta \vdash e'_0 \mapsto e''_0, S'}{S; \theta \vdash e'_0.f_i = e_{arg} \mapsto e''_0.f_i = e_{arg}, S'} \text{RC-RecvWrite}$$

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.

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

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

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

$$\begin{array}{c}
\emptyset; \Sigma; \theta \vdash e_0 : T_0 \quad \emptyset; \Sigma; \theta \vdash \overline{e} : \overline{T}_a \\
mtype(m, T_0) = \overline{T} \rightarrow T_R \quad mbodv(m, T_0) = (\overline{x}, e_R) \quad \overline{T}_a <: [\overline{e}/\overline{x}, e_0/\mathbf{that}]\overline{T} \\
\hline
\emptyset; \Sigma; \theta \vdash T_0 \quad \emptyset; \Sigma; \theta \vdash \overline{T}_a \\
\hline
\emptyset; \Sigma; \theta \vdash e_0.m(\overline{e}) : [\overline{e}/\overline{x}, e_0/\mathbf{that}]T_R
\end{array} \text{[T-INVK]}$$

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

$$\frac{S[\ell] = C\langle\bar{\ell}'.d\rangle(\bar{v}) \quad mbody(m, C\langle\bar{\ell}'.d\rangle) = (\bar{x}, e_0)}{S; \theta \vdash \ell.m(\bar{v}) \mapsto \ell \triangleright [\bar{v}/\bar{x}, \ell/\mathbf{this}]e_0, S} \textit{R-Invk}$$

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

**Subcase**  $e_0 = e'_0$ , that is,  $e_0$  is not a value  $e = e'_0.m(\bar{e})$ . Then, *RC-RecvInvk* applies.

$$\frac{S; \theta \vdash e'_0 \mapsto e''_0, S'}{S; \theta \vdash e'_0.m(\bar{e}) \mapsto e''_0.m(\bar{e}), S'} \textit{RC-RecvInvk}$$

Take  $e' = e''_0.m(\bar{e})$ .

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

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

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

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

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

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

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

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:**

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

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

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

Take  $e' = v$ .

**Subcase**  $e_0$  is not a value, that is  $e = \ell \triangleright e'_0$ . Then  $RC$ -Context applies.

$$\frac{S; \ell \vdash e'_0 \mapsto e''_0, S'}{S; \theta \vdash \ell \triangleright e'_0 \mapsto \ell \triangleright e''_0, S'} RC\text{-Context}$$

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  $T \neq \text{ERROR}$  then  $\emptyset, \Sigma, n_{this} \models n_{this} \rightarrow \text{owner}(T)$ .

**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.



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

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

Preservation **Proof:**

**Case *R-New*:** Then  $e = \mathbf{new} \ C<\bar{p}>(\bar{v})$        $e' = \ell$

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

To show:

$$\emptyset; \Sigma'; \theta \vdash e' : T' \quad (1)$$

$$\emptyset; \Sigma'; \theta \vdash T' \quad (2)$$

$$\Sigma'_{\Delta'} \vdash S' \quad (3)$$

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  $e' = \ell$

$$\text{Take } T' = T = C\langle \overline{\ell'}.d \rangle$$

This proves (1)

$$\Sigma[\theta] = T_{this}$$

By subderivation of T-New

$$\emptyset; \Sigma; \theta \models \text{assumptions}(C\langle \overline{\ell'}.d \rangle)$$

By subderivation of T-New

$$\emptyset; \Sigma; \theta \models \theta \rightarrow \text{owner}(C\langle \overline{\ell'}.d \rangle)$$

By subderivation of T-New

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

By inversion of T-Type

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

**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 remaining of the proof does not change.

■

## Acknowledgements

## 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. Poetsch-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.