Formalisation of Ownership and Immutability Generic Java (OIGJ) - Technical Report

Alex Potanin, Paley Li, Yoav Zibin, Michael D. Ernst

Contents

Chapter 1

Introduction

This technical report presents the full set of formal rules and proofs that accompany our paper called "Ownership and Immutability in Generic Java (OIGJ)". Questions regarding this technical report should be directed to Alex Potanin (alex@ecs.vuw.ac.nz).

Chapter 2

Type Rules

For simplicity we do not model the `@Assignable` field annotation and constructors in this formalism. We also don't model inner classes and make the change of `this` immutability explicit, avoiding the need to model cJ [1].

2.1 FOIGJ Program

FOIGJ program consists of class declarations followed by the program's expression (like FGJ program [2]). Each class declaration is stored inside a class table CT for lookup purposes. Each class declaration is type checked using the FOIGJ-CLASS rule. Finally, the program's expression is also type checked using the appropriate expression type rules. In FOIGJ we also assume that class `Object<O,I>` is pre-declared and that there is no more than one method with the same name per class. In this technical report we prove that any FOIGJ program that is type checked using the rules presented in the paper and in this chapter provides appropriate ownership and immutability guarantees as given by the relevant theorems in Chapter 3.

2.2 Syntax

FOIGJ follows FGJ conventions. `X` represents type variables. `N` represents nonvariable types. `O` represents ownership type variable and nonvariable ownership types are: `Dominator`, and `Modifier`. A special nonvariable ownership type `World` is also allowed as the bound for the ownership type variable. `I` represents immutability type variable and nonvariable immutability types are: `ReadOnly`, `Immutable`, and `Mutable`. For the methods, nonvariable immutability type also includes `AssignsFields`.

Ownership and immutability types are no different from normal generic types. We syntactically distinguish the first type parameter in the list of either class's or method's type parameters as *ownership type parameter* and the second as *immutability type parameter*. The hierarchy of ownership and immutability parameters is shown in Figure 2.1. This is very similar in style to FOGJ's treatment of ownership type parameters. Thus, ownership and immutability are not required to be listed as part of the syntax shown in Figure 2.2.

Future Work: We plan to make location types to be looked up directly from store rather than duplicating their type information in the general enviornment Δ.

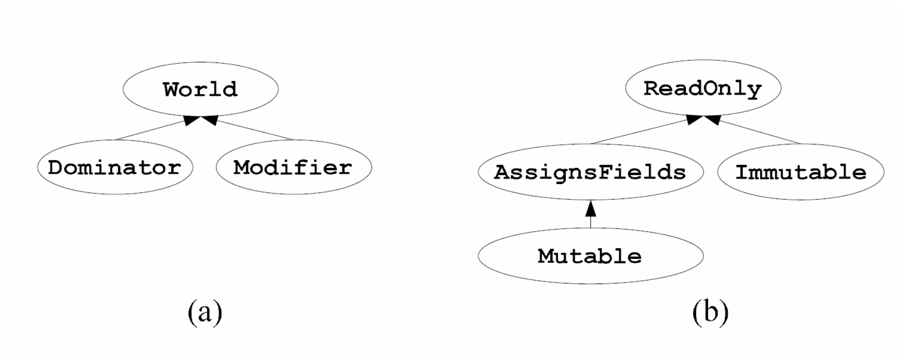

Figure 2.1: Ownership and Immutability Parameters

$\mathtt{T} ::= \mathtt{X} \mid \mathtt{N}$	Type.
$\mathtt{N} ::= \mathtt{C} \mathtt{<}\overline{\mathtt{T}}\mathtt{>}$	Nonvariable type.
$\mathtt{L} ::= \mathtt{class}\ \mathtt{C}\mathtt{<}\overline{\mathtt{X}} \triangleleft \overline{\mathtt{N}}\mathtt{>} \triangleleft \mathtt{N}\ \{\overline{\mathtt{T}}\ \overline{\mathtt{f}};\ \overline{\mathtt{M}}\}$	Class declaration.
$\mathtt{M} ::= \mathtt{<}\overline{\mathtt{X}} \triangleleft \overline{\mathtt{N}}\mathtt{>}\ \mathtt{T}\ \mathtt{m}(\overline{\mathtt{T}}\ \overline{\mathtt{x}})\ \{\mathtt{return}\ \mathtt{e};\}$	Method declaration.
$\mathtt{e} ::= \mathtt{e_s} \mid l \mid l\ \mathtt{>}\ \mathtt{e} \mid \mathtt{error}$	Expressions.
$\mathtt{e_s} ::= \mathtt{x} \mid \mathtt{e.f} \mid \mathtt{e.m}\mathtt{<}\overline{\mathtt{T}}\mathtt{>}(\overline{\mathtt{e}}) \mid \mathtt{new}\ \mathtt{N}() \mid (\mathtt{N})\ \mathtt{e}$ $\mid \mathtt{e.f} = \mathtt{e} \mid \mathtt{null}$	Source expressions.
$v ::= l \mid \mathtt{null}$	Values.
$l \in \mathit{locations}$	Locations.
$S ::= \{l \mapsto \mathtt{N}(\overline{v})\}$	Store.
$\Delta = \{\mathtt{x} \mapsto \mathtt{T}\} \cup \{\mathtt{X} \mapsto \mathtt{N}\} \cup \{l \mapsto \mathtt{N}\}$	Environment that maps (1) variables to their types, (2) type variables to nonvariable types, (3) locations to their types.

Figure 2.2: FOIGJ Syntax

$\Delta \vdash \mathtt{T}\ \mathtt{OK}$	Type $\mathtt{T}$ is OK.
$\Delta \vdash \mathtt{T} <: \mathtt{U}$	Type $\mathtt{T}$ is a subtype of type $\mathtt{U}$.
$\Delta \vdash \mathtt{e} : \mathtt{T}$	Expression $\mathtt{e}$ is well typed.
$\Delta \vdash S\ \mathtt{OK}$	Store (heap) is well formed.
$\Delta\ \mathtt{OK}$	All locations are well-typed in Δ.
$\Delta \vdash \mathtt{<}\overline{\mathtt{Y}} \triangleleft \overline{\mathtt{P}}\mathtt{>}\ \mathtt{T}\ \mathtt{m}(\overline{\mathtt{T}}\ \overline{\mathtt{x}})\{\mathtt{return}\ \mathtt{e_0};\}\ \mathtt{OK}$	Method $\mathtt{m}$ definition is OK.
$\mathtt{class}\ \mathtt{C}\mathtt{<}\overline{\mathtt{X}} \triangleleft \overline{\mathtt{N}}\mathtt{>} \triangleleft \mathtt{N}\ \{\overline{\mathtt{T}}\ \overline{\mathtt{f}};\ \overline{\mathtt{M}}\}\ \mathtt{OK}$	Class $\mathtt{C}$ definition is OK.

Figure 2.3: FOIGJ Judgements

$CT(\mathtt{C})$	The class lookup function for class `C`
$\mathit{fields}(\mathtt{C}\texttt{<}\overline{\mathtt{T}}\texttt{>})$	The fields lookup function
$\mathit{mtype}(\mathtt{m},\ \mathtt{C}\texttt{<}\overline{\mathtt{T}}\texttt{>})$	The method type lookup function
$\mathit{mbody}(\mathtt{m}\texttt{<}\overline{\mathtt{V}}\texttt{>},\ \mathtt{C}\texttt{<}\overline{\mathtt{T}}\texttt{>})$	The method type lookup function
$\mathit{bound}_\Delta(\mathtt{T})$	The bound of type function
$\mathit{typeparams}_\Delta(\mathtt{T})$	Recursively look up all type parameters
$O_\Delta(\mathtt{T})$	The owner of type `T`
$I_\Delta(\mathtt{T})$	The immutability type of type `T`
$FV(\mathtt{C})$	The free variable function for class `C`

Figure 2.4: FOIGJ Functions

`this`	The special variable `this`
$\mathtt{Dominator}_l$	The runtime dominator owner parameter for location l
$\mathtt{Modifier}_l$	The runtime modifier owner parameter for location l

Figure 2.5: FOIGJ Special Variables

2.3 Type Judgements and Auxiliary Functions

Following FGJ, `Y` corresponds to type variables and `P` corresponds to nonvariable types (like `X` and `N`). Figure 2.3 shows FOIGJ judgements which are very close to FOGJ. Figure 2.4 shows FOIGJ functions. Figure 2.5 shows specially treated variables in FOIGJ. Figure 2.6 shows FOIGJ bound function that is identical to FGJ bound function.

2.4 Lookup Functions

Figure 2.7 shows ownership and immutability lookup functions. Figure 2.9 shows the standard lookup functions based on FGJ (the only rule that is slightly different is F-OBJECT). Figure 2.8 shows the additional lookup function used by FOIGJ.

2.5 Well-formed Types and Subtyping

Figure 2.10 shows type well-formedness rules. They include the owner nesting rule for the class's type parameters from OGJ that is used to enforce deep ownership. Note that the nesting of `Dominator <: O <: World` is enforced by FOIGJ-CLASS rule. We also omit the immutability parameters in the ownership nesting check in WF-TYPE rule.

Bound of Type:

$$\mathit{bound}_\Delta(\mathtt{X}) = \Delta(\mathtt{X})$$
$$\mathit{bound}_\Delta(\mathtt{N}) = \mathtt{N}$$

Figure 2.6: FOIGJ Bound Function (Identical to FGJ)

Owner Lookup :

$$
\begin{array}{lcl}
O_\Delta(\mathtt{O}) & = & \mathtt{O} \\
O_\Delta(\mathtt{World}) & = & \mathtt{World} \\
O_\Delta(\mathtt{Dominator}) & = & \mathtt{Dominator} \\
O_\Delta(\mathtt{Modifier}) & = & \mathtt{Modifier} \\
O_\Delta(\mathtt{X}) & = & O_\Delta(\Delta(\mathtt{X})) \\
O_\Delta(\mathtt{C<O,\ I,\ \overline{T}>}) & = & \mathtt{O} \\
O_\Delta(\mathtt{Object<O,\ I>}) & = & \mathtt{O}
\end{array}
$$

Immutability Lookup:

$$
\begin{array}{lcl}
I_\Delta(\mathtt{X}) & = & I_\Delta(\Delta(\mathtt{X})) \\
I_\Delta(\mathtt{C<O,\ I,\ \overline{T}>}) & = & \mathtt{I} \\
I_\Delta(\mathtt{Object<O,\ I>}) & = & \mathtt{I}
\end{array}
$$

Figure 2.7: FOIGJ Ownership and Immutability Lookup Functions

$$
\begin{array}{c}
typeparams_\Delta(\mathtt{X}) = \{\Delta(\mathtt{X})\} \\
typeparams_\Delta(\mathtt{C<\overline{T}>}) = \overline{\mathtt{T}} \cup \bigcup_{\mathtt{T}\in\overline{\mathtt{T}}} typeparams_\Delta(\mathtt{T})
\end{array}
$$

Figure 2.8: FOIGJ Additional Lookup Function

We enforce that `Dominator` is inside `Modifier` without using the subtyping rule because making `Dominator` a subtype of `Modifier` will permit declaration of invalid classes.

Future Work: We plan to express the interaction of modifiers and dominators in a nicer way than the exception for `Modifier` in WF-TYPE.

Figure 2.11 shows the FOIGJ subtyping rules. These include the partial variance rules allowed by FIGJ. Please refer to the next section for *CoVariant* definition.

Finally, note that the owner parameter can never be subject to variance for ownership guarantees.

Future Work: We plan to prove the S-OWNER rule relationship since it is implied by our construction in FOIGJ-CLASS rule.

2.6 NoVariant Definition

Figure 2.12 shows the definition of *CoVariant* as used in the subtyping rule to detect the cases when variance of read-only or immutable objects should be prohibited due to potential loophole due to additional generic type parameters. Please see the IGJ paper for a more detailed description of this problem.

Field Lookup:

$$fields(\texttt{Object<O, I>}) = \bullet \qquad \text{(F-Object)}$$

$$\frac{CT(\texttt{C}) = \texttt{class C<}\overline{\texttt{X}} \triangleleft \overline{\texttt{N}}\texttt{>} \triangleleft \texttt{N} \{\overline{\texttt{S}}\ \overline{\texttt{f}};\ \overline{\texttt{M}}\} \qquad fields([\overline{\texttt{T}}/\overline{\texttt{X}}]\texttt{N}) = \overline{\texttt{U}}\ \overline{\texttt{g}}}{fields(\texttt{C<}\overline{\texttt{T}}\texttt{>}) = \overline{\texttt{U}}\ \overline{\texttt{g}},\ [\overline{\texttt{T}}/\overline{\texttt{X}}]\overline{\texttt{S}}\ \overline{\texttt{f}}} \qquad \text{(F-Class)}$$

Method Type Lookup:

$$\frac{CT(\texttt{C}) = \texttt{class C<}\overline{\texttt{X}} \triangleleft \overline{\texttt{N}}\texttt{>} \triangleleft \texttt{N} \{\overline{\texttt{S}}\ \overline{\texttt{f}};\ \overline{\texttt{M}}\} \qquad \texttt{<}\overline{\texttt{Y}} \triangleleft \overline{\texttt{P}}\texttt{> U m(}\overline{\texttt{U}}\ \overline{\texttt{x}}\texttt{)\{ return e; \}} \in \overline{\texttt{M}}}{mtype(\texttt{m},\ \texttt{C<}\overline{\texttt{T}}\texttt{>}) = [\overline{\texttt{T}}/\overline{\texttt{X}}](\texttt{<}\overline{\texttt{Y}} \triangleleft \overline{\texttt{P}}\texttt{>}\overline{\texttt{U}} \to \texttt{U})} \qquad \text{(MT-Class)}$$

$$\frac{CT(\texttt{C}) = \texttt{class C<}\overline{\texttt{X}} \triangleleft \overline{\texttt{N}}\texttt{>} \triangleleft \texttt{N} \{\overline{\texttt{S}}\ \overline{\texttt{f}};\ \overline{\texttt{M}}\} \qquad \texttt{m} \notin \overline{\texttt{M}}}{mtype(\texttt{m},\ \texttt{C<}\overline{\texttt{T}}\texttt{>}) = mtype(\texttt{m},\ [\overline{\texttt{T}}/\overline{\texttt{X}}]\texttt{N})} \qquad \text{(MT-Super)}$$

Method Body Lookup:

$$\frac{CT(\texttt{C}) = \texttt{class C<}\overline{\texttt{X}} \triangleleft \overline{\texttt{N}}\texttt{>} \triangleleft \texttt{N} \{\overline{\texttt{S}}\ \overline{\texttt{f}};\ \overline{\texttt{M}}\} \qquad \texttt{<}\overline{\texttt{Y}} \triangleleft \overline{\texttt{P}}\texttt{> U m(}\overline{\texttt{U}}\ \overline{\texttt{x}}\texttt{)\{ return e}_0\texttt{; \}} \in \overline{\texttt{M}}}{mbody(\texttt{m<}\overline{\texttt{V}}\texttt{>},\ \texttt{C<}\overline{\texttt{T}}\texttt{>}) = \overline{\texttt{x}}.[\overline{\texttt{T}}/\overline{\texttt{X}},\ \overline{\texttt{V}}/\overline{\texttt{Y}}]\texttt{e}_0} \qquad \text{(MB-Class)}$$

$$\frac{CT(\texttt{C}) = \texttt{class C<}\overline{\texttt{X}} \triangleleft \overline{\texttt{N}}\texttt{>} \triangleleft \texttt{N} \{\overline{\texttt{S}}\ \overline{\texttt{f}};\ \overline{\texttt{M}}\} \qquad \texttt{m} \notin \overline{\texttt{M}}}{mbody(\texttt{m<}\overline{\texttt{V}}\texttt{>},\ \texttt{C<}\overline{\texttt{T}}\texttt{>}) = mbody(\texttt{m<}\overline{\texttt{V}}\texttt{>},\ [\overline{\texttt{T}}/\overline{\texttt{X}}]\texttt{N})} \qquad \text{(MB-Super)}$$

Figure 2.9: FOIGJ Lookup Functions (Almost Identical to FGJ)

$$\frac{\texttt{X} \in dom(\Delta)}{\Delta \vdash \texttt{X OK}} \qquad \text{(WF-TypeVar)}$$

$$\frac{\Delta \vdash \texttt{I <: ReadOnly} \qquad \Delta \vdash \texttt{O <: World}}{\Delta \vdash \texttt{Object<O, I> OK}} \qquad \text{(WF-Object)}$$

(WF-Type):

$$\frac{\begin{array}{c} CT(\texttt{C}) = \texttt{class C<}\overline{\texttt{X}} \triangleleft \overline{\texttt{N}}\texttt{>} \triangleleft \texttt{N} \{\ldots\} \qquad \Delta \vdash \overline{\texttt{T}}\ \texttt{OK} \qquad \Delta \vdash \overline{\texttt{T}} \texttt{<:} [\overline{\texttt{T}}/\overline{\texttt{X}}]\overline{\texttt{N}} \\ \Delta \vdash \texttt{N <: Object<O, I>} \qquad \Delta \vdash \texttt{I <: ReadOnly} \qquad \Delta \vdash \texttt{O <: World} \\ \forall \texttt{T} \in \overline{\texttt{T}} : \neg(\Delta \vdash \texttt{T <: ReadOnly}) \Rightarrow (O_\Delta(\texttt{C<}\overline{\texttt{T}}\texttt{>}) \texttt{ <: } O_\Delta(\texttt{T})) \\ (O_\Delta(\texttt{C<}\overline{\texttt{T}}\texttt{>}) = \texttt{Dominator} \wedge O_\Delta(\texttt{T}) = \texttt{Modifier}) \end{array}}{\Delta \vdash \texttt{C<}\overline{\texttt{T}}\texttt{> OK}}$$

Figure 2.10: FOIGJ Type Well-Formedness Rules

Subtyping:

$$\frac{}{\Delta \vdash \mathtt{T} <: \mathtt{T}} \quad \text{(S-REFL)} \qquad \frac{}{\Delta \vdash \mathtt{X} <: \Delta(\mathtt{X})} \quad \text{(S-TYPEVAR)}$$

$$\frac{\Delta \vdash \mathtt{S} <: \mathtt{T} \qquad \Delta \vdash \mathtt{T} <: \mathtt{U}}{\Delta \vdash \mathtt{S} <: \mathtt{U}} \quad \text{(S-TRANS)}$$

$$\frac{CT(\mathtt{C}) = \mathtt{class}\ \mathtt{C} < \overline{\mathtt{X}} \triangleleft \overline{\mathtt{N}} > \ \triangleleft\ \mathtt{N}\ \{\ldots\}}{\Delta \vdash \mathtt{C} < \overline{\mathtt{T}} > <: [\overline{\mathtt{T}}/\overline{\mathtt{X}}]\mathtt{N}} \quad \text{(S-CLASS)}$$

(S-O-HIERARCHY):

$$\frac{}{\Delta \vdash \mathtt{Dominator} <: \mathtt{World}} \qquad \frac{}{\Delta \vdash \mathtt{Modifier} <: \mathtt{World}}$$

(S-I-HIERARCHY):

$$\frac{}{\Delta \vdash \mathtt{AssignsFields} <: \mathtt{ReadOnly}}$$

$$\frac{}{\Delta \vdash \mathtt{Mutable} <: \mathtt{AssignsFields}}$$

$$\frac{}{\Delta \vdash \mathtt{Immutable} <: \mathtt{ReadOnly}}$$

$$\frac{l \in dom(\Delta)}{\Delta \vdash \mathtt{Dominator}_l <: O_\Delta(\Delta(l))} \quad \text{(S-OWNER)}$$

(S-VARIANCE):

$$\frac{\begin{array}{c} \mathtt{T} = \mathtt{C} < \mathtt{O},\ \mathtt{I},\ \overline{\mathtt{S}} > \qquad \mathtt{T}' = \mathtt{C} < \mathtt{O},\ \mathtt{I}',\ \overline{\mathtt{P}} > \qquad \Delta \vdash \mathtt{I} <: \mathtt{I}' \\ CT(\mathtt{C}) = \mathtt{class}\ \mathtt{C} < \overline{\mathtt{X}} \triangleleft \overline{\mathtt{N}} > \ \triangleleft\ \mathtt{N}\ \{\ldots\} \\ \forall \mathtt{S}, \mathtt{P}, \mathtt{X} \in \overline{\mathtt{S}}, \overline{\mathtt{P}}, \overline{\mathtt{X}} : (\mathtt{S} = \mathtt{P}) \vee \\ (\Delta \vdash \mathtt{Immutable} <: \mathtt{I}') \wedge (\Delta \vdash \mathtt{S} <: \mathtt{P}) \wedge (CoVariant(\mathtt{X}, \mathtt{C})) \end{array}}{\Delta \vdash \mathtt{T} <: \mathtt{T}'}$$

Figure 2.11: FOIGJ Subtyping Rules

2.6.1 NoVariant Rule

Figure 2.13 shows how class `Bar` has all three type parameters marked as novariant because of their use in mutable superclass, mutable not this-owned field, and in the position of another no-variant type parameter.

Future Work: Rewrite cv rule and the novariant rules to require `@NoVariant` declaration and make the type rules in class declaration simply check that it is used correctly. This will avoid the "not" oddity in cv since the proofs of "not provable" are too hard. There also may be a better way that can be thought of later.

NoVariant `I`-rule A type parameter must be no-variant if it is used in a mutable

NoVariant:

$$\frac{CT(\mathtt{C}) = \mathtt{class}\ \mathtt{C}\texttt{<}\overline{\mathtt{X}} \triangleleft \overline{\mathtt{N}}\texttt{>}\ \triangleleft\ \mathtt{N}\ \{\overline{\mathtt{T}}\ \overline{\mathtt{f}};\ \overline{\mathtt{M}}\} \qquad \mathtt{N} = \mathtt{D}\texttt{<}\overline{\mathtt{P}}\texttt{>} \quad \mathtt{X} \in \overline{\mathtt{P}} \wedge I_{\Delta}(\mathtt{N}) = \mathtt{Mutable}}{NoVariant(\mathtt{X}, \mathtt{C})} \quad \text{(NV-Superclass-Mutable)}$$

$$\frac{\begin{array}{c} CT(\mathtt{C}) = \mathtt{class}\ \mathtt{C}\texttt{<}\overline{\mathtt{X}} \triangleleft \overline{\mathtt{N}}\texttt{>}\ \triangleleft\ \mathtt{N}\ \{\overline{\mathtt{T}}\ \overline{\mathtt{f}};\ \overline{\mathtt{M}}\} \\ \forall \mathtt{T} \in \overline{\mathtt{T}} : (\mathtt{T} = \mathtt{D}\texttt{<}\overline{\mathtt{P}}\texttt{>}) \Rightarrow \mathtt{X} \in \overline{\mathtt{P}} \wedge \\ (I_{\Delta}(\mathtt{T}) = \mathtt{Mutable} \vee I_{\Delta}(\mathtt{T}) = \mathtt{AssignsFields}) \wedge \\ (O_{\Delta}(\mathtt{T}) = \mathtt{Dominator} \vee O_{\Delta}(\mathtt{T}) = \mathtt{Modifier}) \end{array}}{NoVariant(\mathtt{X}, \mathtt{C})} \quad \text{(NV-Field-Mutable)}$$

$$\frac{\begin{array}{c} CT(\mathtt{C}) = \mathtt{class}\ \mathtt{C}\texttt{<}\overline{\mathtt{X}} \triangleleft \overline{\mathtt{N}}\texttt{>}\ \triangleleft\ \mathtt{N}\ \{\overline{\mathtt{T}}\ \overline{\mathtt{f}};\ \overline{\mathtt{M}}\} \\ \forall \mathtt{T} \in \overline{\mathtt{T}}, \mathtt{N} : (\mathtt{T} = \mathtt{D}\texttt{<}\overline{\mathtt{P}}\texttt{>}) \Rightarrow NoVariant(\mathtt{X}, \mathtt{D}) \end{array}}{NoVariant(\mathtt{X}, \mathtt{C})} \quad \text{(NV-NoVariant-Position)}$$

$$\frac{\begin{array}{c} CT(\mathtt{C}) = \mathtt{class}\ \mathtt{C}\texttt{<}\overline{\mathtt{X}} \triangleleft \overline{\mathtt{N}}\texttt{>}\ \triangleleft\ \mathtt{N}\ \{\overline{\mathtt{T}}\ \overline{\mathtt{f}};\ \overline{\mathtt{M}}\} \\ \forall \mathtt{T} \in \overline{\mathtt{T}}, \mathtt{N} : (\mathtt{T} = \mathtt{D}\texttt{<}\overline{\mathtt{P}}\texttt{>}) \Rightarrow \\ \forall \mathtt{P} \in \overline{\mathtt{P}} : (\mathtt{P} = \mathtt{D}'\texttt{<}\overline{\mathtt{P}'}\texttt{>}) \Rightarrow NoVariant(\mathtt{X}, \mathtt{D}') \end{array}}{NoVariant(\mathtt{X}, \mathtt{C})} \quad \text{(NV-NoVariant-Subterms)}$$

CoVariant:

$$\frac{\neg NoVariant(\mathtt{X}, \mathtt{C})}{CoVariant(\mathtt{X}, \mathtt{C})} \quad \text{(CV)}$$

Figure 2.12: FOIGJ Novariant definition

```
class Bar<O extends World, I extends ReadOnly,
          E1 extends Object, E2 extends Object, E3 extends Object>
          extends Foo<Dominator, Mutable, E1> {
    Foo<World, Mutable, E2> f1;
    Foo<World, Immutable, E3, E2, E1> f2;
}
```

Figure 2.13: Example of NoVariant derivation

superclass, a mutable field or an assignable field that is not `this`-owned, or in the position of another no-variant type parameter.

2.7 Expressions

Figures 2.14 and 2.15 gives the FOIGJ expression typing rules.

(T-NEW):

$$\frac{\Delta \vdash \mathtt{N}\ \mathtt{OK} \qquad \mathit{fields}(\mathtt{N}) = \overline{\mathtt{T}}\ \overline{\mathtt{f}} \qquad \Delta \vdash \overline{\mathtt{e}} : \overline{\mathtt{S}} \qquad \Delta \vdash \overline{\mathtt{S}} <: \overline{\mathtt{T}}}{\Delta \vdash \mathtt{new}\ \mathtt{N}(\overline{\mathtt{e}})\ :\ \mathtt{N}}$$

$$\frac{\Delta \vdash \mathtt{N}\ \mathtt{OK} \qquad \Delta \vdash \mathtt{e}_0\ :\ \mathtt{T}_0}{\Delta \vdash (\mathtt{N})\mathtt{e}_0\ :\ \mathtt{N}}\ \text{(T-CAST)} \qquad \frac{\Delta \vdash \mathtt{e}\ :\ \mathtt{T}}{\Delta \vdash l\ \mathtt{>}\ \mathtt{e} :\ \mathtt{T}}\ \text{(T-CONTEXT)}$$

$$\frac{}{\Delta \vdash \mathtt{x}\ : \Delta(\mathtt{x})}\ \text{(T-VAR)} \qquad \frac{}{\Delta \vdash l\ :\ \Delta(l)}\ \text{(T-LOC)}$$

$$\frac{\Delta \vdash \mathtt{T}\ \mathtt{OK}}{\Delta \vdash \mathtt{error}\ :\ \mathtt{T}}\ \text{(T-ERROR)} \qquad \frac{\Delta \vdash \mathtt{T}\ \mathtt{OK}}{\Delta \vdash \mathtt{null}\ :\ \mathtt{T}}\ \text{(T-NULL)}$$

Figure 2.14: FOIGJ Expression Typing 1 of 2

Future Work: Add the case of immutability of the field being immutable to the rule on viewpoint adaptation in the paper.

Future Work: Why are we so conservative on the viewpoint adaptation? Why not only replace with readonly the immutability parameters recursively under the modifier one and not all of them in the type?

2.7.1 T-Field Rule

Field access `O`-rule Accessing `o.f`, where `o` $\neq$ `this`, is illegal when $O(\mathtt{f}) = \mathtt{Dominator}$, and requires view-point adaptation when one of the owners in the type of $O(\mathtt{f})$ is `Modifier`. We use auxiliary *typeparams* function to recursively lookup all the type parameters involved in the full type signature.

2.7.2 T-Field-Set Rule

Field assignment `I`-rule `o.f = ...` is legal iff one of the following holds:

$I(\mathtt{o}) = \mathtt{Mutable}$
$I(\mathtt{o}) = \mathtt{AssignsFields}$ and ($\mathtt{o} = \mathtt{this}$ or $\underline{O(\mathtt{o}) \doteq \mathtt{this}}$)
`f` is annotated as `@Assignable`

2.7.3 T-Method Rule

Method invocation `I`-rule `o.m(...)` is legal iff $I(\mathtt{o}) <: I(\mathtt{m})$ and $(I(\mathtt{m}) = \mathtt{AssignsFields}$ implies that ($\mathtt{o} = \mathtt{this}$ or $\underline{\emptyset(\mathtt{o} \doteq \mathtt{this}}$)).

2.8 Class and Method Definitions

Figure 2.16 gives FOIGJ Method and FOIGJ Class definition rules.

(T-Field-Dominator):

$$\frac{\begin{array}{c}\Delta \vdash \mathtt{e}_0 \ : \ \mathtt{T}_0 \qquad \mathit{fields}(\mathit{bound}_\Delta(\mathtt{T}_0)) = \ \overline{\mathtt{T}}\ \overline{\mathtt{f}} \qquad \Delta \vdash \mathtt{T}\ \mathtt{OK} \\ O_\Delta(\mathtt{T}_i) = \mathtt{Dominator} \Rightarrow \mathtt{e}_0 = \mathtt{this} \qquad \mathtt{T} = \mathtt{T}_i\end{array}}{\Delta \vdash \mathtt{e}_0.\mathtt{f}_i \ : \ \mathtt{T}}$$

(T-Field-Other):

$$\frac{\begin{array}{c}\Delta \vdash \mathtt{e}_0 \ : \ \mathtt{T}_0 \qquad \mathit{fields}(\mathit{bound}_\Delta(\mathtt{T}_0)) = \ \overline{\mathtt{T}}\ \overline{\mathtt{f}} \qquad \Delta \vdash \mathtt{T}\ \mathtt{OK} \\ O_\Delta(\mathtt{T}_i) \neq \mathtt{Dominator} \\ \mathtt{Modifier} \in \mathit{typeparams}_\Delta(\mathtt{T}_i) \Rightarrow \mathtt{e}_0 = \mathtt{this} \qquad \mathtt{T} = \mathtt{T}_i\end{array}}{\Delta \vdash \mathtt{e}_0.\mathtt{f}_i \ : \ \mathtt{T}}$$

(T-Field-VA):

$$\frac{\begin{array}{c}\Delta \vdash \mathtt{e}_0 \ : \ \mathtt{T}_0 \qquad \mathit{fields}(\mathit{bound}_\Delta(\mathtt{T}_0)) = \ \overline{\mathtt{T}}\ \overline{\mathtt{f}} \qquad \Delta \vdash \mathtt{T}\ \mathtt{OK} \\ O_\Delta(\mathtt{T}_i) \neq \mathtt{Dominator} \qquad \mathtt{Modifier} \in \mathit{typeparams}_\Delta(\mathtt{T}_i) \qquad \mathtt{e}_0 \neq \mathtt{this} \\ (\mathtt{T} = [\mathtt{ReadOnly/Mutable}, \mathtt{ReadOnly/AssignsFields}, \mathtt{ReadOnly/I}]\mathtt{T}_i \wedge \\ I_\Delta(\mathtt{T}_i) \neq \mathtt{Immutable}) \vee (\mathtt{T} = \mathtt{T}_i \wedge I_\Delta(\mathtt{T}_i) = \mathtt{Immutable}))\end{array}}{\Delta \vdash \mathtt{e}_0.\mathtt{f}_i \ : \ \mathtt{T}}$$

(T-Field-Set):

$$\frac{\begin{array}{c}\Delta \vdash \mathtt{e}_0 \ : \ \mathtt{T}_0 \qquad \Delta \vdash \mathtt{e} \ : \ \mathtt{T} \qquad \mathit{fields}(\mathit{bound}_\Delta(\mathtt{T}_0)) = \overline{\mathtt{T}}\ \overline{\mathtt{f}} \\ \Delta \vdash \mathtt{T} \mathrel{<:} \mathtt{T}_i \qquad \Delta\mathtt{T}\ \mathtt{OK} \qquad (I_\Delta(\mathtt{T}_0) = \mathtt{Mutable} \ \wedge \\ ((O_\Delta(\mathtt{T}_i) = \mathtt{Dominator} \vee O_\Delta(\mathtt{T}_i) = \mathtt{Modifier}) \Rightarrow \mathtt{e}_0 = \mathtt{this})) \ \vee \\ ((\mathtt{e}_0 = \mathtt{this} \vee O_\Delta(\mathtt{T}_0) = \mathtt{Modifier} \vee O_\Delta(\mathtt{T}_0) = \mathtt{Dominator}) \\ \wedge (I(\mathtt{e}_0) = \mathtt{AssignsFields}))\end{array}}{\Delta \vdash \mathtt{e}_0.\mathtt{f}_i = \mathtt{e} \ : \ \mathtt{T}}$$

(T-Method):

$$\frac{\begin{array}{c}\Delta \vdash \overline{\mathtt{e}} \ : \ \overline{\mathtt{S}} \qquad \Delta \vdash \mathtt{T}\ \mathtt{OK} \qquad \forall \mathtt{S}' \in \overline{\mathtt{S}} \ : \ (\Delta \vdash \mathtt{S}'\ \mathtt{OK}) \qquad \Delta \vdash \mathtt{e}_0 \ : \ \mathtt{T}_0 \\ \mathit{mtype}(\mathtt{m}, \ \mathit{bound}_\Delta(\mathtt{T}_0)) = \mathtt{<}\,\overline{\mathtt{Y}} \triangleleft \overline{\mathtt{P}}\,\mathtt{>}\,\overline{\mathtt{U}} \rightarrow \mathtt{U} \\ \mathit{mbody}(\mathtt{m}, \ \mathit{bound}_\Delta(\mathtt{T}_0)) = \overline{\mathtt{e}}.\mathtt{e}' \qquad \Delta \vdash \mathtt{e}' \ : \ \mathtt{Q} \qquad \mathtt{Q} \mathrel{<:} \mathtt{U} \\ \forall \mathtt{V}' \in \overline{\mathtt{V}} \ : \ (\Delta \vdash \mathtt{V}'\ \mathtt{OK} \ \vee \ \Delta \vdash \mathtt{V}' \mathrel{<:} \mathtt{World} \ \vee \ \Delta \vdash \mathtt{V}' \mathrel{<:} \mathtt{ReadOnly}) \\ \forall \mathtt{V}' \in \overline{\mathtt{V}} \ : \ (\Delta \vdash \mathtt{V}' \mathrel{<:} \mathtt{World} \Rightarrow \Delta \vdash \mathtt{O} \mathrel{<:} O_\Delta(\mathtt{V}')) \\ I_\Delta(\mathtt{Q}) \mathrel{<:} I_\Delta(\mathtt{U}) \qquad I_\Delta(\mathtt{T}_0) \mathrel{<:} \mathtt{I} \qquad \mathtt{I} = \mathtt{AssignsFields} \Rightarrow \\ (\mathtt{e}_0 = \mathtt{this} \vee O_\Delta(\mathtt{T}_0) = \mathtt{Modifier} \vee O_\Delta(\mathtt{T}_0) = \mathtt{Dominator}) \\ \mathtt{T}' = [\overline{\mathtt{V}}/\overline{\mathtt{Y}}]\mathtt{U} \wedge ((\ O_\Delta(\mathtt{U}) = \mathtt{Dominator}) \Rightarrow (\mathtt{e}_0 = \mathtt{this})) \\ \Delta \vdash \overline{\mathtt{V}} \mathrel{<:} [\overline{\mathtt{V}}/\overline{\mathtt{Y}}]\overline{\mathtt{P}} \ \wedge \ (\forall \mathtt{P}' \in \overline{\mathtt{P}} \ : \ ((O_\Delta(\mathtt{P}') = \mathtt{Dominator}) \Rightarrow (\mathtt{e}_0 = \mathtt{this}))) \\ \Delta \vdash \overline{\mathtt{S}} \mathrel{<:} [\overline{\mathtt{V}}/\overline{\mathtt{Y}}]\overline{\mathtt{U}} \ \wedge \ (\forall \mathtt{U}' \in \overline{\mathtt{U}} \ : \ ((O_\Delta(\mathtt{U}') = \mathtt{Dominator}) \Rightarrow (\mathtt{e}_0 = \mathtt{this}))) \\ (\mathtt{Modifier} \in \mathit{typeparams}_\Delta(\mathtt{T}') \wedge \mathtt{e}_0 \neq \mathtt{this} \wedge I_\Delta(\mathtt{T}') \neq \mathtt{Immutable}) \Rightarrow \\ \mathtt{T} = [\mathtt{ReadOnly/Mutable}, \mathtt{ReadOnly/AssignsFields}, \mathtt{ReadOnly/I}]\mathtt{T}' \\ ((\mathtt{Modifier} \notin \mathit{typeparams}_\Delta(\mathtt{T}') \vee \mathtt{e}_0 = \mathtt{this} \vee I_\Delta(\mathtt{T}') = \mathtt{Immutable}) \Rightarrow \mathtt{T} = \mathtt{T}'\end{array}}{\Delta \vdash \mathtt{e}_0.\mathtt{m}\,\mathtt{<}\,\mathtt{O}, \ \mathtt{I}, \ \overline{\mathtt{V}}\,\mathtt{>}(\overline{\mathtt{e}}) \ : \ \mathtt{T}}$$

Figure 2.15: FOIGJ Expression Typing 2 of 2

FOIGJ Method Definition (FOIGJ-METHOD):

$$\frac{\begin{array}{c}\Delta = \overline{\mathtt{Y}} \mathrel{<:} \overline{\mathtt{P}},\ \overline{\mathtt{X}} \mathrel{<:} \overline{\mathtt{N}} \\ CT(\mathtt{C}) = \mathtt{class\ C{<}O' \triangleleft World, I' \triangleleft ReadOnly,}\ \overline{\mathtt{X}} \triangleleft \overline{\mathtt{N}}\mathtt{>}\ \triangleleft\ \mathtt{N}\ \{\ldots\} \\ \Delta \vdash \overline{\mathtt{T}}, \overline{\mathtt{P}}, \mathtt{T}\ \mathtt{OK} \qquad \Delta,\ \overline{\mathtt{x}}\ :\overline{\mathtt{T}},\ \mathtt{this}\ :\ \mathtt{C{<}O',\ I,}\ \overline{\mathtt{X}}\mathtt{>}\,;\mathtt{C} \vdash \mathtt{e}_0 : \mathtt{S} \\ \Delta \vdash \mathtt{S} \mathrel{<:} \mathtt{T} \qquad I(\mathtt{S}) \mathrel{<:} \mathtt{I}\end{array}}{\Delta \vdash \mathtt{<O \triangleleft World,\ I \triangleleft ReadOnly,}\ \overline{\mathtt{Y}}\ \triangleleft\ \overline{\mathtt{P}}\mathtt{>\ T\ m(}\overline{\mathtt{T}}\ \overline{\mathtt{x}}\mathtt{)\{\ return\ e_0;\ \}\ OK}}$$

FOIGJ Class Definition (FOIGJ-CLASS):

$$\frac{\begin{array}{c}\Delta = \{\overline{\mathtt{X}} \mathrel{<:} \overline{\mathtt{N}}, \mathtt{O} \mathrel{<:} \mathtt{World}, \mathtt{I} \mathrel{<:} \mathtt{ReadOnly}, \mathtt{Dominator} \mathrel{<:} \mathtt{O}\} \cup \\ placeholderowners_{\Delta}(\overline{\mathtt{N}}) \cup\ (\bigcup_{\mathtt{X}' \in \overline{\mathtt{X}}} \mathtt{O} \mathrel{<:}\ O_{\Delta}(\mathtt{X}')\}) \\ \forall \mathtt{N}' \in \overline{\mathtt{N}} : \Delta \vdash \mathtt{N}'\ \mathtt{OK} \vee \Delta \vdash\ O_{\Delta}(\mathtt{N}') \mathrel{<:} \mathtt{World} \vee \Delta \vdash\ O_{\Delta}(\mathtt{N}') \mathrel{<:} \mathtt{ReadOnly} \\ \Delta = \overline{\mathtt{X}} \mathrel{<:} \overline{\mathtt{N}} \qquad \Delta \vdash \mathtt{N}, \overline{\mathtt{T}}\ \mathtt{OK} \qquad \Delta \vdash \overline{\mathtt{M}}\ \mathtt{OK\ IN\ C} \\ \mathtt{N} = \mathtt{D{<}O,\ I,}\ \overline{\mathtt{T}'}\mathtt{>} \qquad \overline{\mathtt{T}} \mathrel{<:} \overline{\mathtt{T}'}\end{array}}{\mathtt{class\ C{<}O \triangleleft World,\ I \triangleleft ReadOnly,}\ \overline{\mathtt{X}} \triangleleft \overline{\mathtt{N}}\mathtt{>} \triangleleft \mathtt{N}\ \{\overline{\mathtt{T}}\ \overline{\mathtt{f}};\ \overline{\mathtt{M}}\}\ \mathtt{OK}}$$

Figure 2.16: FOIGJ Method and Class Rules

Placeholder Owners Function:

$$\begin{array}{lcl} placeholderowners_{\Delta}(\mathtt{C{<}O,\ I,}\ \overline{\mathtt{T}}\mathtt{>}) & = & \{\mathtt{O} \mathrel{<:} \mathtt{World}\} \cup \\ & & \{\mathtt{I} \mathrel{<:} \mathtt{ReadOnly}\} \cup \\ & & placeholderowners_{\Delta}(\overline{\mathtt{T}}) \\ & & \text{if } O_{\Delta}(\mathtt{C{<}O,\ I,}\ \overline{\mathtt{T}}\mathtt{>}) \notin dom(\Delta) \\ placeholderowners_{\Delta}(\mathtt{C{<}O,\ I,}\ \overline{\mathtt{T}}\mathtt{>}) & = & placeholderowners_{\Delta}(\overline{\mathtt{T}}) \\ & & \text{otherwise} \\ placeholderowners_{\Delta}(\mathtt{X}) & = & \{\} \end{array}$$

Figure 2.17: FOIGJ Placeholder Owners Function

Note that immutability of `this` is changed based on the method's immutability parameter.

Figure 2.17 gives the auxiliary function called *placeholderowners* used in the class definition rule.

2.9 Store

Figure 2.18 gives the FOIGJ Store Well-Formedness and Typing rules.

Store Well-Formedness:

$$\frac{\forall l \in dom(\Delta) : \Delta \vdash \ \Delta(l)\ \texttt{OK}}{\Delta\ \texttt{OK}}$$

Store Typing:

$$\frac{\begin{array}{c}\Delta\ \texttt{OK} \qquad dom_l(\Delta)^{\dagger} = dom(S) \\ S[l] = \texttt{N}(\overline{v}) \Longrightarrow \Delta(l) = \texttt{N} \qquad \Delta(l) = \texttt{N} \Longrightarrow \exists \overline{v} : S[l] = \texttt{N}(\overline{v}) \\ (S[l,i] = l') \wedge (\mathit{fields}(\Delta(l)) = \overline{\texttt{T}}\ \overline{\texttt{f}}) \\ \Longrightarrow \Delta \vdash \Delta(l') \texttt{<:} [\texttt{Dominator}_l/\texttt{Dominator}, \texttt{Modifier}_l/\texttt{Modifier}]\texttt{T}_i \\ (S[l,\ i] = l') \Longrightarrow \Delta \vdash \ \Delta(l')\ \texttt{OK}\end{array}}{\Delta \vdash S\ \texttt{OK}}$$

† $dom_l(\Delta)$ refers to the domain of Δ that is restricted to *locations* only.

Figure 2.18: FOIGJ Store

2.10 Reduction Rules

Figure 2.19 shows the context reduction rules and Figure 2.20 shows the rest of the reduction rules.

Note that $\neg$ is used for simplicity in the R-BAD-CAST rule.

Reduction Context Expression:

$$\begin{array}{rl} E ::= & [\ ] \\ & E.\texttt{f} \\ & E.\texttt{f} = \texttt{e} \\ & l.\texttt{f} = E \\ & E.\texttt{m}\texttt{<}\overline{\texttt{T}}\texttt{>}(\overline{\texttt{e}}) \\ & l.\texttt{m}\texttt{<}\overline{\texttt{T}}\texttt{>}(\overline{l}, E, \overline{\texttt{e}'}) \\ & (\texttt{N})E \\ & l \texttt{>} E \end{array}$$

Context Reduction Rule:

$$\frac{\texttt{e}, S \rightarrow \texttt{e}', S'}{E[\texttt{e}], S \rightarrow E[\texttt{e}'], S'}$$

Figure 2.19: FOIGJ Context Reduction Rule

(R-New):

$$\frac{l \notin dom(S) \qquad S' = S[l \mapsto \mathtt{N}(\overline{\mathtt{null}})] \qquad |\overline{\mathtt{null}}| = |\mathit{fields}(\mathtt{N})|}{\mathtt{new}\ \mathtt{N}(), S \to l, S'}$$

(R-Field):

$$\frac{S[l] = \mathtt{N}(\overline{v}) \qquad \mathit{fields}(\mathtt{N}) = \overline{\mathtt{T}}\ \overline{\mathtt{f}}}{l.\mathtt{f}_i, S \to v_i, S}$$

(R-Field-Set):

$$\frac{S[l] = \mathtt{N}(\overline{v}) \qquad \mathit{fields}(\mathtt{N}) = \overline{\mathtt{T}}\ \overline{\mathtt{f}} \qquad S' = S[l \mapsto \mathtt{N}(v_0, ..., v_{i-1}, v, v_{i+1}, ..., v_{|\overline{\mathtt{f}}|})]}{l.\mathtt{f}_i = v, S \to v, S'}$$

(R-Method):

$$\frac{S[l] = \mathtt{N}(\overline{v_l}) \qquad \mathit{mbody}(\mathtt{m}\texttt{<}\overline{\mathtt{V}}\texttt{>}, \mathtt{N}) = \overline{\mathtt{x}}.\mathtt{e}_0}{l.\mathtt{m}\texttt{<}\overline{\mathtt{V}}\texttt{>}(\overline{v}), S \to l\ \texttt{>}\ [\overline{v}/\overline{\mathtt{x}}, l/\mathtt{this}, \mathtt{Dominator}_l/\mathtt{Dominator}, \mathtt{Modifier}_l/\mathtt{Modifier}]\mathtt{e}_0, S}$$

(R-*-Null):

$$\frac{}{\mathtt{null}.\mathtt{m}\texttt{<}\overline{\mathtt{V}}\texttt{>}(\overline{v}), S \to \mathtt{error}, S}$$

$$\frac{}{\mathtt{null}.\mathtt{f}_i, S \to \mathtt{error}, S} \qquad \frac{}{\mathtt{null}.\mathtt{f}_i = v, S \to \mathtt{error}, S}$$

$$\frac{S[l] = \mathtt{N}(\overline{v}) \qquad \Delta \vdash \mathtt{N} \texttt{<:}\ \mathtt{P}}{(\mathtt{P})l, S \to l, S}\ \text{(R-Cast)} \qquad \frac{S[l] = \mathtt{N}(\overline{v}) \qquad \neg(\Delta \vdash \mathtt{N} \texttt{<:}\ \mathtt{P})}{(\mathtt{P})l, S \to \mathtt{error}, S}\ \text{(R-Bad-Cast)}$$

$$\frac{}{l\ \texttt{>}\ v, S \to v, S}\ \text{(R-Context)}$$

Figure 2.20: FOIGJ Reduction Rules

Chapter 3

Theorems and Proofs

This chapter states and proves four theorems: type preservation, progress, immutability invariant, and ownership invariant.

3.1 Preservation

The type preservation theorem says that if any FOIGJ expression reduces to another FOIGJ expression then the latter is always a subtype of the former. Before stating the theorem, let's define a shorthand for a well typed expression in a well typed store.

Definition 1. $\Delta \vdash \texttt{e}, S \ : \ \texttt{T} \equiv (\Delta \vdash \texttt{e} \ : \ \texttt{T}) \ \wedge \ (\Delta \vdash S)$

Theorem 1. ***(Type Preservation)*** *If* $\Delta \vdash \texttt{e}, S \ : \ \texttt{T}$ *and* $\texttt{e}$, $S \to \texttt{e}'$, S', *then* $\exists \Delta' \supseteq \Delta$ *and* $\exists \texttt{T}' \texttt{<:} \texttt{T}$ *such that* $\Delta' \vdash \texttt{e}', S' \ : \ \texttt{T}'$.

The following is the proof of the Type Preservation theorem.

R-Field

Proof. Assume the following:

(i) $S[l] = \texttt{N}(v_0, ..., v_{i-1}, v, v_{i+1}, ..., v_{|\overline{\texttt{f}}|}))$

(ii) $\mathit{fields}(\texttt{N}) = \overline{\texttt{T}}\ \overline{\texttt{f}}$

(iii) $\Delta \vdash l.\texttt{f}_i, S \ : \ \texttt{T}_i$

(iv) $l.\texttt{f}_i, S \to v_i, S$

Store: The reduction for this expression only retrieve information in location l, therefore store is not updated $S' = S$ and hence $\Delta' = \Delta$ by Store Typing.

Expression: Let $\texttt{e} = l.\texttt{f}_i : \texttt{T}_i$ and $\texttt{T} = [\texttt{Dominator}_l/\texttt{Dominator}, \texttt{Modifier}_l/\texttt{Modifier}]\texttt{T}_\texttt{i}$ by T-Field. Now let $\texttt{e}'$ be the expression reduced from $\texttt{e}$, so $\texttt{e}' = v_i$, also $\texttt{T}' = \Delta'(v_i)$ by T-Loc, hence $\texttt{e}' : \texttt{T}'$. There are two cases for $\texttt{e}'$.

If $\mathtt{e}' = \mathtt{null}$ then $\Delta' \vdash \mathtt{null} \mathrel{\texttt{<:}} \mathtt{T}_i$ by T-Null, therefore $\mathtt{T}' = \mathtt{T}$.
If $\mathtt{e}' \neq \mathtt{null}$ then $\Delta' = \Delta$ because we have not performed any store update, so by S-Dominator $\Delta' \vdash \Delta(v_i) \mathrel{\texttt{<:}} [\mathtt{Dominator}_l/\mathtt{Dominator}, \mathtt{Modifier}_l/\mathtt{Modifier}]\mathtt{T}_i$. Therefore $\Delta' \vdash \mathtt{T}' \mathrel{\texttt{<:}} \mathtt{T}$ will always hold true for R-Field.

□

R-Field-Set

Proof. Assume the following:

(i) $S[l] = \mathtt{N}(\overline{v})$

(i) $\mathit{fields}(\mathtt{N}) = \overline{\mathtt{T}}\ \overline{\mathtt{f}}$

(i) $S' = S[l \mapsto \mathtt{N}(v_0, ..., v_{i-1}, v, v_{i+1}, ..., v_{|\overline{\mathtt{f}}|})]$

(i) $l.\mathtt{f}_i = v, S \rightarrow v, S'$

Store: The only change to S is an update on the i^{th} field of the object at location l. The field is now pointing at v instead of v_i. T-Field-Set guarantees that $\Delta \vdash \Delta(v) \mathrel{\texttt{<:}} \Delta(v_i)$ and no new l is added into Δ, allowing us to define $\Delta' = \Delta$.

Expression: Let $\mathtt{e} = (l.\mathtt{f}_i = v) : \mathtt{T}$ then by the definition of Δ and T-Field-Set we can conclude that $\mathtt{e} = v : \mathtt{T}$ and $\mathtt{T} = \Delta(v)$. Let $\mathtt{e}' = v : \mathtt{T}'$ where $\mathtt{T}' = \Delta'(v)$. Finally by the definition of Δ' above, $\Delta' = \Delta$ thus $\Delta'(v) = \Delta(v)$, then $\mathtt{T}' = \Delta'(v) = \Delta(v) = \mathtt{T}$ hence $\Delta' \vdash \mathtt{T}' \mathrel{\texttt{<:}} \mathtt{T}$ hold by S-REFL.

□

R-Method

Proof. Assume the following:

(i) $S[l] = \mathtt{N}(\overline{v_l})$

(i) $\mathit{mbody}(\mathtt{m}\texttt{<}\overline{\mathtt{V}}\texttt{>}, \mathtt{N}) = \overline{\mathtt{x}}.\mathtt{e}_0$

(i) $l.\mathtt{m}\texttt{<}\overline{\mathtt{V}}\texttt{>}(\overline{v}), S \rightarrow l\ \texttt{>}\ [\overline{v}/\overline{\mathtt{x}}, l/\mathtt{this}, \mathtt{Dominator}_l/\mathtt{Dominator}, \mathtt{Modifier}_l/\mathtt{Modifier}]\mathtt{e}_0, S$

Store: The reduction for this expression only retrieve information in location l, therefore store is not updated $S' = S$ and hence $\Delta' = \Delta$ by Store Typing.

Expression: Let $\mathtt{e} = l.\mathtt{m}\texttt{<}\overline{\mathtt{V}}\texttt{>}(\overline{v}) : T,\ \mathtt{T} = [\overline{\mathtt{V}}/\overline{\mathtt{Y}}]\mathtt{U}$ by MT-Class $\mathit{mtype}(\mathtt{m},\ \mathit{bound}_\Delta(\Delta(l)) = \texttt{<}\overline{\mathtt{Y}} \triangleleft \overline{\mathtt{P}}\texttt{>}\overline{\mathtt{U}} \rightarrow \mathtt{U}$ and $[l/\mathit{this}] \Rightarrow [\mathtt{Dominator}_l/\mathtt{Dominator}]$ by T-METHOD. Now let $\mathtt{e}' = l\ \texttt{>}\ [\overline{v}/\overline{\mathtt{x}},\ l/\mathtt{this}, \mathtt{Dominator}_l/\mathtt{Dominator}, \mathtt{Modifier}_l/\mathtt{Modifier}]\mathtt{e}_0 : \mathtt{T}_i$, by T-METHOD and T-CONTEXT $\mathtt{T}' = \mathtt{Q}$. Lastly by MB-CLASS we can show that $\mathtt{T}' = [\overline{\mathtt{V}}/\overline{\mathtt{Y}}]\mathtt{Q}$ and by T-METHOD $\mathtt{Q} \mathrel{\texttt{<:}} \mathtt{U}$, therefore $\Delta \vdash \mathtt{T}' \mathrel{\texttt{<:}} \mathtt{T}$ □

R-New

Proof. Assume the following:

(i) $l \notin dom(S)$

(ii) $S' = S[l \mapsto \mathtt{N}(\overline{\mathtt{null}})]$

(iii) $|\overline{\mathtt{null}}| = |fields(\mathtt{N})|$

(iv) $\mathtt{new}\ \mathtt{N}(), S \rightarrow l, S'$

Store: To proof the preservation of the store we will proof by structural induction on each of the type inference rule of the store in Store Typing, hence proof $\Delta' \vdash S'$. Our induction hypothesis is $\Delta' = \Delta \cup \{l \rightarrow \mathtt{N}\}$. There are six case to consider.

First case: $\Delta\mathtt{OK}$

Proof. $$\frac{\forall l' \in dom(\Delta) : \Delta \vdash\ \Delta(l')\ \mathtt{OK} \qquad \Delta \cup \{l \mapsto \mathtt{N}\} \vdash\ \Delta \cup \{l \mapsto \mathtt{N}\}(l)\ \mathtt{OK}}{\Delta \cup \{l \mapsto \mathtt{N}\}\ \mathtt{OK}}$$

$$\frac{\forall l' \in dom(\Delta \cup \{l \mapsto \mathtt{N}\}) : \Delta \cup \{l \mapsto \mathtt{N}\} \vdash\ \Delta \cup \{l \mapsto \mathtt{N}\}(l')\ \mathtt{OK}}{\Delta \cup \{l \mapsto \mathtt{N}\}\ \mathtt{OK}}$$

By induction hyp.

$$\frac{\forall l' \in dom(\Delta') : \Delta' \vdash\ \Delta'(l')\ \mathtt{OK}}{\Delta'\ \mathtt{OK}}$$

□

Second case: $dom_l(\Delta)^\dagger = dom(S)$

Proof. $dom_l(\Delta \cup \{l \mapsto \mathtt{N}\})^\dagger = dom(S \cup \{l \mapsto \mathtt{N}\})$, then by (ii) and induction hyp $dom_l(\Delta')^\dagger = dom(S')$

□

Third case: $S[l] = \mathtt{N}(\overline{v}) \Longrightarrow \Delta(l) = \mathtt{N}$

Proof. Let $S'[l] = \mathtt{N}(\overline{v})$ and by the second case $l \in \Delta'$ as $dom_l(\Delta')^\dagger = dom(S')$. Then by the first case $\Delta'(l)\mathtt{OK}$ and T-NEW we ensure Δ' contain only well-formed types and that $\Delta \vdash \mathtt{N}\ \mathtt{OK}$. Finally by the induction hyp $\Delta'(l) = \mathtt{N}$ because there exist a mapping from $\{l \mapsto \mathtt{N}\}$ in the induction hyp. □

Fourth case: $\Delta(l) = \mathtt{N} \Longrightarrow \exists \overline{v} : S[l] = \mathtt{N}(\overline{v})$

Proof. Let $\Delta'(l) = \mathtt{N}$ then $l \in S'$ by the second case. Now by (ii) we know that S' contain a mapping of $\{l \mapsto \mathtt{N}\}$, therefore $S'[l] = \mathtt{N}(\overline{v})$ and $\overline{v} = \overline{\mathtt{null}}$ initially. □

Fifth case:$(S[l, i] = l') \wedge (fields(\Delta(l)) = \overline{\mathtt{T}}\ \overline{\mathtt{f}}) \Longrightarrow$ $\Delta \vdash$ $\Delta(l') \mathrel{<:} [\mathtt{Dominator}_l/\mathtt{Dominator}, \mathtt{Modifier}_l/\mathtt{Modifier}]\mathtt{T}_i$

Proof. Let $(S[l,i] = l')$ and $(\mathit{fields}(\Delta(l)) = \overline{\mathtt{T}}\ \overline{\mathtt{f}})$, so l' is the location of the i^{th} field in $\Delta(l)$. By (iii) $fields(\Delta(l')) = \mathtt{null}$, then $\Delta(l') = \mathtt{T}$ and by T-NULL $\mathtt{T}$ can be any well-formed type. Now let $\mathtt{T} = \mathtt{T}_i$ by the first case, since $\mathtt{T}_i$ is well-formed. Finally by S-Dominator$_l$ $\Delta \vdash \mathtt{T}_i \mathrel{<:} [\mathtt{Dominator}_l/\mathtt{Dominator}, \mathtt{Modifier}_l/\mathtt{Modifier}]\mathtt{T}_i$ therefore $\Delta \vdash \Delta(l') \mathrel{<:} [\mathtt{Dominator}_l/\mathtt{Dominator}, \mathtt{Modifier}_l/\mathtt{Modifier}]\mathtt{T}_i$. □

Sixth case: $(S[l,\ i] = l') \Longrightarrow \Delta \vdash\ \Delta(l')\ \mathtt{OK}$

Proof. Let $(S[l,i] = l')$. By the second case $l' \in \Delta$. By the first case $\Delta(l')\mathtt{OK}$. □

These six cases proofs that the store is preserved when a new type is created.

Expression: Let $\mathtt{e} = \mathtt{new}\ \mathtt{N}() : \mathtt{T}$ where $\mathtt{T} = \mathtt{N}$ by T-New. Also let $\mathtt{e}' = l : \mathtt{T}'$ and by T-Loc $\mathtt{T}' = \Delta'(l)$. By definition of the new store Δ', $\Delta'(l) = \mathtt{N}$, therefore $\mathtt{T}' = \mathtt{N}$. Finally $\Delta' \vdash \mathtt{T}' \mathrel{<:} \mathtt{T}$ by S-REFL on $\mathtt{T}' = \mathtt{N} = \mathtt{T}$. □

R-Context

Proof. Assume the following:

(i) $l \mathrel{>} v, S \rightarrow v, S$

Store: The reduction for this expression only retrieve information in location l, therefore store is not updated $S' = S$ and hence $\Delta' = \Delta$ by Store Typing.

Expression: Let $\mathtt{e} = l \mathrel{>} v : \mathtt{T}$ then by T-CONTEXT $\Delta; l \vdash v : \mathtt{T}$, therefore $\mathtt{T} = \Delta v$. Now let $\mathtt{e}' = v : \mathtt{T}'$ where $\mathtt{T}' = \Delta' v$. Finally by the definition of Δ' above, $\Delta' = \Delta$ thus $\Delta'(v) = \Delta(v)$, then $\mathtt{T}' = \Delta'(v) = \Delta(v) = \mathtt{T}$ hence $\Delta' \vdash \mathtt{T}' \mathrel{<:} \mathtt{T}$ hold by S-REFL. □

R-Cast

Proof. Assume the following:

(i) $S[l] = \mathtt{N}(\overline{v})$

(ii) $\mathtt{N} \mathrel{<:} \mathtt{P}$

(iii) $(\mathtt{P})l, S \rightarrow l, S$

Store: The reduction for this expression only retrieve information in location l, therefore store is not updated $S' = S$ and hence $\Delta' = \Delta$ by Store Typing.

Expression: Let $\mathtt{e} = (\mathtt{P})l : \mathtt{T}$ where $\mathtt{T} = \mathtt{P}$ by T-CAST. Let $\mathtt{e}' = l : \mathtt{T}'$ and $\mathtt{T}' = \Delta(l)$ then by (i) $\mathtt{T}' = \mathtt{N}$. Finally by (ii) and the definition of Δ' , where $\Delta' = \Delta$, $\Delta' \vdash \mathtt{T}' \mathrel{<:} \mathtt{T}$. □

R-Bad-Cast

Proof. Assume the following:

(i) $S[l] = \mathtt{N}(\overline{v})$

(ii) $\mathtt{N} \not<: \mathtt{P}$

(iii) $(\mathtt{P})l, S \to \mathtt{error}, S$

Store: The reduction for this expression only retrieve information in location l, therefore store is not updated $S' = S$ and hence $\Delta' = \Delta$ by Store Typing.

Expression: Let $\mathtt{e} = (\mathtt{P})l : \mathtt{T}$ where $\mathtt{T} = \mathtt{P}$ by T-CAST. Let $\mathtt{e}' = \mathtt{error} : \mathtt{T}'$ and by T-ERROR let $\mathtt{T}' = \mathtt{P}$ as $\mathtt{error}$ can have the type of any well-formed type. Then by the definition of Δ' , where $\Delta' = \Delta$, and S-REFL on $\mathtt{T} = \mathtt{P} = \mathtt{T}'$, $\Delta' \vdash \mathtt{T}' <: \mathtt{T}$ hold. □

R-METHOD/FIELD/FIELD-SET-Null

Proof. Assume the following:

(i) $\mathtt{null.m}<\overline{\mathtt{V}}>(\overline{v}), S \to \mathtt{error}, S$

(ii) $\mathtt{null.f}_i, S \to \mathtt{error}, S$

(iii) $\mathtt{null.f}_i = v, S \to \mathtt{error}, S$

Store: The reduction for this expression only retrieve information in location l, therefore store is not updated $S' = S$ and hence $\Delta' = \Delta$ by Store Typing.

Expression: For each of the reduction rule on null $\mathtt{e} = \mathtt{null} : \mathtt{T}$, because the location `null` does not contain any fields or methods. By T-NULL $\mathtt{T}$ is any well-formed type, for the sake of convenience let $\mathtt{T} = \mathtt{T}''$ where $\mathtt{T}'' \in \Delta$ and $\Delta \vdash \mathtt{T}''\ \mathtt{OK}$. Then for each rule we have $\mathtt{e}' = \mathtt{error} : \mathtt{T}'$, similar to T-NULL, T-ERROR allow us to have $\mathtt{T}' = \mathtt{T}''$. Then by the definition of Δ' , where $\Delta' = \Delta$, and S-REFL on $\mathtt{T} = \mathtt{T}'' = \mathtt{T}'$, $\Delta' \vdash \mathtt{T}' <: \mathtt{T}$ hold. □

Context Reduction Rules

Proof. The context reduction rules trivially do not change either the store or the type of the expression. □

3.2 Progress

The progress theorem shows that FOIGJ programs do not get "stuck" and that any well typed FOIGJ expression that does not contain free variables (closed) can be reduced to some value or FOIGJ's `error` (the latter includes failed downcasts due to R-Bad-Cast reducing them to `error`).

Theorem 2. ***(Progress)*** *Suppose* $\mathtt{e}$ *is a closed well-typed FOIGJ expression. Then either* $\mathtt{e}$ *is a value (or* `error`*) or there is an applicable reduction rule that contains* $\mathtt{e}$ *on the left hand side.*

Proof. In order to proof the Progress Theorem we must go through each expression and show that progress is achieved. There are seven cases, six expressions and a set of expressions.

First case: When $\mathtt{e}$ is an `error`, `null`, variable $\mathtt{x}$ or location l then $\mathtt{e}$ is a closed expression that can not be reduced by any reduction rule. $\mathtt{e}$ becomes a value of the closed expression it represents.

Second case: When $\mathtt{e} = l$ > e' by R-CONTEXT rule $\mathtt{e}$ will be reduced to be $\mathtt{e}'$ and there are no additional requirements on $\mathtt{e}$.

Third case: When $\mathtt{e} = (\mathtt{N})\mathtt{e}'$, $\mathtt{e}$ can be reduced by two reduction rules. If the type of $\mathtt{e}'$ is subtype of $\mathtt{N}$ then $\mathtt{e}$ reduces to $\mathtt{e}'$ by R-CAST. Otherwise $\mathtt{e}$ will reduce to `error` by R-BAD-CAST, which is a value of the system.

Fourth case: When $\mathtt{e} = \mathtt{e}'.\mathtt{f}_i$ T-FIELD and R-FIELD ensures the type($\mathtt{N}$) of e' is well-formed and the field $\mathtt{f}_i$ is in the class bounded by $\mathtt{N}$. $\mathtt{e}$ will reduce into the value that is contained in $\mathtt{f}_i$ by R-FIELD.

Fifth case: When $\mathtt{e} = (\mathtt{e}'.\mathtt{f}_i = e'')$ similar to the fourth case, T-FIELD-SET AND R-FIELD-SET will ensure the type($\mathtt{N}$) of e' is well-formed and the field $\mathtt{f}_i$ is in the class bounded by $\mathtt{N}$. Furthermore R-FIELD-SET guarantee the i^{th} field in $\mathtt{N}$ will be directed at the value reduced from $\mathtt{e}''$. $\mathtt{e}$ will reduce into the expression $\mathtt{e}''$ by R-FIELD-SET.

Sixth case: When $\mathtt{e} = \mathtt{e}'.\mathtt{m}(\overline{\mathtt{e}''})$ the reduction rule R-METHOD will apply. R-METHOD retrieve the expression($\mathtt{e}'''$) in the body of the method($\mathtt{m}$) consequently ensuring the types in $\mathtt{m}$ are correct. R-CONTEXT will also be called upon to ensure a value produced on reduction of $\mathtt{e}$. $\mathtt{e}$ will be reduced to the expression($\mathtt{e}'''$) R-METHOD.

Seventh case: When $\mathtt{e} = \mathtt{new}\ \mathtt{N}(\overline{\mathtt{e}'})$ R-NEW will reduce $\mathtt{e}$ to become the allocated location(l) in the store(S) that contains the instances of the type($\mathtt{N}$). R-NEW will also ensure the fields of $\mathtt{N}$ are initialized with `null` and T-NEW ensures $\mathtt{N}$ is well-formed as are the type of the fields in $\mathtt{N}$.

□

3.3 Immutability and Ownership

The central ownership and immutability guarantees are just a combination of the normal invariants: (1, IGJ) an (im)mutable reference always points to an (im)mutable objectl and (2, OGJ) for mutable references if one object refers to another, then the owner of the latter will be *inside* the owner of the former.

Since OIGJ combines dominator and modifier kinds of ownership, we define the following OIGJ-specific guarantees:

Theorem 3. *Consider two objects o_1 and o_2.*

Dominator-property: *Object o_1 can point to o_2 iff $o_1 \preceq_D D(o_2)$. Method can point to o (meaning it can have a pointer to o on its stack) iff there exists o_i such that $o_i \preceq_D D(o)$.*

Modifier-property: *Object o_1 can modify o_2 (meaning the modification occurs in a method call whose receiver is o_1) iff $o_1 \preceq_O O(o_2)$. Method can modify o iff there exists o_i such that $o_i \preceq_O O(o)$.*

In the statement above, D is defined as follows: $D(o) = [\texttt{World}/\texttt{Modifier}]O(o)$, so that any modifier edge is replaced by a global `World` owner. Since we do not model `Stack` owner for simplicity, `World` owner will suffice.

Proof. For the objects (i.e. fields), given o_1 and o_2 at locations l_1 and l_2 respectively, by the store typing and well-formedness rules, $\Delta \vdash \Delta(l_1) \texttt{<:} [\texttt{Dominator}_{\texttt{l}_1}/\texttt{Dominator}, \texttt{Modifier}_l /\texttt{Modifier}]\Delta(l_2)$. If the owner of o_2 is `Modifier`, then by definition of D it will be replaced by `World` which is going to be outside by the definition of the inside relationship. If the owner of o_2 is not `Modifier` or `World`, then well-formedness preserves owner class nesting and we have defined the nesting to be $\texttt{Dominator}_{\texttt{l}_1} \texttt{<:} \texttt{Owner} \texttt{<:} \texttt{O}$ where `Owner` is the owner of o_2 and `O` is any of the owners of the type parameters of the class of o_2. Hence, $o_1 \preceq_D D(o_2)$ as required. Furthermore, for the modifier property will hold for the same reasons since we preserve the nesting of both dominator owners and modifier owners in the same way.

For the methods, the only additional requirement is that the owners involved in the method via methods' type parameters are only allowed to keep mutable annotations when the owner is not `Modifier`, so by construction of the `T-Method` rule, both dominator and modifier property will hold for methods. □

Theorem 4. ***(Immutability Invariant)*** *Let $\Delta, \texttt{S} \vdash \texttt{e} : \texttt{T}$, and $\texttt{e}, \texttt{S} \rightarrow^* \texttt{l}, \texttt{S}'$, where $\texttt{S}'[\texttt{l}] = \texttt{N}(\overline{\texttt{l}})$. Then $\Delta \vdash I_\Delta \texttt{N} \texttt{<:} I_\Delta \texttt{T}$.*

Proof. From preservation theorem, $\Delta \vdash \texttt{N} \texttt{<:} \texttt{T}$. Thus $\Delta \vdash I_\Delta(\texttt{N}) \texttt{<:} I_\Delta(\texttt{T})$. □

Theorem 5. ***(Dominator Invariant)*** *l refers to l' only if $\texttt{Dominator}_l \prec O_\Delta(l')$ or $\texttt{I}(\Delta(l')) = \texttt{ReadOnly}$ or $\texttt{I}(\Delta(l')) = \texttt{Immutable}$.*

Proof. For read-only and immutable references the proof is immediate. For fields by FGO-Store, $\Delta \vdash \Delta(l') \texttt{<:} [\texttt{Dominator}_\texttt{l}/\texttt{Dominator}, \texttt{Modifier}_l/\texttt{Modifier}]\ \texttt{T}_\texttt{i}$. If owner is `World` or `Dominator`, then the theorem holds by the definition of $\prec$. If owner is anything else then since well-formedness preserves owner class nesting for fields and $\texttt{Dominator} \texttt{<:} \texttt{Owner} \texttt{<:} \texttt{O}$ (where `O` is the set of owners of type parameters) holds, one has $\Delta \vdash \texttt{Dominator}_l \texttt{<:} \texttt{Dominator}_{l'}$. □

Chapter 4

Additional Discussion

This section contains the material that didn't make it into the full paper for space reasons.

4.1 Related Work

Huang et al. [1] propose an extension of Java (called cJ) that allows methods to be provided only under some static subtyping condition. For instance, a cJ generic class, `Date<I>`, can define

$<\texttt{I}extends\texttt{Mutable}>?\ voidsetDate(\ldots)$

which will be provided only when the type provided for parameter `I` is a subtype of `Mutable`.

It is possible to use cJ syntax, instead of OIGJ's method-annotation, which makes **`thisI`-rule** redundant. Then, an iterator can use two mutability immutability parameters: one for the iterator (`I`) and one for the collection (`CI`).

```
interface Iterator<I extends ReadOnly, O extends World, CI extends ReadOnly, E>
{
  boolean hasNext();
  <I extends Mutable> E next();
  <CI extends Mutable> void remove();
}
```

The inner class will now have its own immutability and ownership, and therefore it will have several different "this" instances.

4.2 Refactoring of the Clone Method

Ownership and method `clone`

Method `clone` is very tricky to implement correctly, and we believe its design is poor for the following reasons:

Code duplication All the collections implement `clone` first by calling `super.clone`. From the documentation:

"By convention, the returned object should be obtained by calling `super.clone`. If a class and all of its superclasses (except Object) obey this convention, it will be the case that

`x.clone().getClass()==x.getClass()`"

An unaware programmer might implement `clone`, e.g., in `LinkedList`, by: `return new LinkedList(this);`

Furthermore, after calling `super.clone` whose return type is `Object`, there is always a *downcast*, which is another source for mistakes.

Accessibility `clone` is protected in `Object`, but it is usually made public in subclasses. For example, you can clone an `ArrayList` but not a `List`.

Copy constructor Cloning constructs a new object (similarly to copy-constructors in C++), however they do not have the privileges of constructors. For instance, `final` fields of the clone cannot be assigned. Therefore, `header` in `LinkedList` cannot be made `final` (also due to method `readObject`, which implements deserialization and is also a form of constructor). Note that in JAVA5 memory model (JSR 133 [3]), `final` fields allow safe multi-threaded access of immutable objects without the overhead of synchronization using a process called `final` *field freeze* at the end of a constructor.

Ownership Aliases are tricky to control and understand. Cloning creates a shallow copy, i.e., only immediate fields were copied. An inexperienced programmer might implement `clone` in `LinkedList` as follows:

```
LinkedList result = (LinkedList) super.clone();
result.clear();
result.addAll(this);
return result;
```

Because cloning creates a shallow copy, calling `clear` also clears the content of `this`, and the final result is an empty linked list.

Finally, Sun's implementation assigns to <u>`result`</u>`.header` which is a `this`-owned field. This violates **Field assignment `O`-rule** that only permits assignment to <u>`this`</u>`.header`.

A partial solution to the above problems uses the idea of *inversion of control*: instead of initializing the cloned `result` from `this`, we refactor the code into a `constructFrom` method that initializes `this`from a parameter.

Fig. 4.1 shows the refactoring done on the `clone` method of `LinkedList`. Originally, `clone` directly assigned to the `this`-owned fields of `result` (line 3), which is illegal in OIGJ. Therefore, we had to refactor these assignments into a newly created `constructFrom` method (line 16), where such assignments are legal (line 18).

The `clone` method on lines 7–14 is automatically generated by the compiler in order to enforce ownership and immutability properties. After calling `super.clone` on line 8, the compiler sets all the reference fields to `null`, and then calls `constructFrom`. Auto-generating the `clone` method solves most of the afore-mentioned problems: a programmer

```
1: public Object clone() { {Original code}
2:  LinkedList result =(LinkedList) super.clone();
3:  result.header = new Entry(); {Would be illegal in OIGJ!}
4:  return result;
5: }
6: {Auto-generated clone method}
7: public @OI Object clone() @OReadOnly {
8:  @OI LinkedList result =
9:     (@OI LinkedList) super.clone();
10: {The next two lines are safe but illegal in OIGJ.}
11:  result.header = null; {nullify pointers to preserve ownership}
12:  result.constructFrom(this); {construct the result}
13:  return result;
14: }
15: {User-generated constructFrom method}
16: protected void constructFrom(
17:    @WildCardReadOnly LinkedList l) @OAssignsFields {
18:  this.header= new @DominatorI Entry(); {Legal}
19: }
```

Figure 4.1: Refactoring of method `clone` in class `LinkedList`.

would only override `constructFrom` and therefore cannot forget to call `super.clone`, the cast on line 2 is fail-safe, and subtle aliasing issues are avoided by nullifying the reference fields.

We note that lines 11–12 are safe but illegal in OIGJ. Line 11 assigns `null` into a `this`-owned field not via `this`, which is illegal according to **Field assignment `O`-rules**. However, it is safe because we assign a `null` value and not another reference. Line 12 calls `constructFrom` whose immutability is `AssignsFields` on a reference whose immutability is `I`, which is illegal according to **Method invocation `I`-rule**. However, because `result` was just created on line 8 and did not escape the method (yet), it is safe to continue its construction. Furthermore, a compiler can give `constructFrom` all the privileges of a constructor, e.g., assigning to `final` fields. Therefore, field `header` could be declared `final`.

Bibliography

[1] S. S. Huang, D. Zook, and Y. Smaragdakis. cJ: Enhancing Java with safe type conditions. In *AOSD*, pages 185–198. ACM Press, Mar. 2007.

[2] A. Igarashi, B. C. Pierce, and P. Wadler. Featherweight Java: a minimal core calculus for Java and GJ. *(TOPLAS)*, 23(3):396–450, May 2001.

[3] W. Pugh. JSR 133: JAVA memory model and thread specification revision. http://jcp.org/en/jsr/detail?id=133, Sept. 30, 2004.